Internet Architecture Board (IAB)
Request for Comments: 7452
Category: Informational
ISSN: 2070-1721

H. Tschofenig
ARM Ltd.
J. Arkko
D. Thaler
D. McPherson

March 2015

Architectural Considerations in Smart Object Networking

Abstract

The term "Internet of Things" (IoT) denotes a trend where a large number of embedded devices employ communication services offered by Internet protocols. Many of these devices, often called "smart objects", are not directly operated by humans but exist as components in buildings or vehicles, or are spread out in the environment. Following the theme "Everything that can be connected will be connected", engineers and researchers designing smart object networks need to decide how to achieve this in practice.

This document offers guidance to engineers designing Internet-connected smart objects.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Architecture Board (IAB) and represents information that the IAB has deemed valuable to provide for permanent record. It represents the consensus of the Internet Architecture Board (IAB). Documents approved for publication by the IAB are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7452.

Table of Contents

1. Introduction 2
2. Smart Object Communication Patterns 4
2.1. Device-to-Device Communication Pattern 4
2.2. Device-to-Cloud Communication Pattern 6
2.3. Device-to-Gateway Communication Pattern 7
2.4. Back-End Data Sharing Pattern 9
3. Reuse Internet Protocols 10
4. The Deployed Internet Matters 13
5. Design for Change 14
6. Security Considerations 16
7. Privacy Considerations 18
8. Informative References 19
Appendix A. IAB Members at the Time of Approval 23
Acknowledgements 23
Authors' Addresses 24

1. Introduction

RFC 6574 [RFC6574] refers to smart objects as devices with constraints on energy, bandwidth, memory, size, cost, etc. This is a fuzzy definition, as there is clearly a continuum in device capabilities and there is no hard line to draw between devices that can run Internet protocols and those that can't.

Interconnecting smart objects with the Internet enables exciting new use cases and products. An increasing number of products put the Internet Protocol Suite on smaller and smaller devices and offer the ability to process, visualize, and gain insight from the collected sensor data. The network effect can be increased if the data collected from many different devices can be combined.

Developing embedded systems is a complex task, and designers must make a number of design decisions such as:

- How long is the device designed to operate?
- How does it interact with the physical world? Is it a sensor or actuator or both?
- How many "owners" does it have? One? Many? Is the owner likely to change over the lifetime of the device?
- Is it continuously or intermittently powered? Does it sleep?
- Is it connected to a network, and if so, how?
- Will it be physically accessible for direct maintenance after deployment? How does that affect the security model?

While developing embedded systems is itself a complex task, designing Internet-connected smart objects is even harder since it requires expertise with Internet protocols in addition to software programming and hardware skills. To simplify the development task, and thereby to lower the cost of developing new products and prototypes, we believe that reuse of prior work is essential. Therefore, we provide high-level guidance on the use of Internet technology for the development of smart objects, and connected systems in general.

Utilize Existing Design Patterns

Design patterns are generally reusable solutions to a commonly occurring design problem (see [Gamma] for more discussion). Existing smart object deployments show communication patterns that can be reused by engineers with the benefit of lowering the design effort. As discussed in the sections below, individual patterns also have an implication on the required interoperability between the different entities. Depending on the desired functionality, already-existing patterns can be reused and adjusted. Section 2 talks about various communication patterns.

Reuse Internet Protocols

Most smart object deployments can make use of the already-standardized Internet Protocol Suite. Internet protocols can be applied to almost any environment due to their generic design and typically offer plenty of potential for reconfiguration, which allows them to be tailored for the specific needs. Section 3 discusses this topic.

The Deployed Internet Matters

When connecting smart objects to the Internet, take existing deployment into consideration to avoid unpleasant surprises. Assuming an ideal, clean-slate deployment is, in many cases, far too optimistic since the already-deployed infrastructure is convenient to use. In Section 4, we highlight the importance of this topic.

Design for Change

The Internet infrastructure, applications, and preferred building blocks evolve over time. Especially long-lived smart object deployments need to take this change into account, and Section 5 is dedicated to that topic.

2. Smart Object Communication Patterns

This section illustrates a number of communication patterns utilized in the smart object environment. It is possible that more than one pattern can be applied at the same time in a product. Developers reusing those patterns will benefit from the experience of others as well as from documentation, source code, and available products.

2.1. Device-to-Device Communication Pattern

Figure 1 illustrates a communication pattern where two devices developed by different manufacturers are desired to interoperate and communicate directly. To pick an example from [RFC6574], consider a light switch that talks to a light bulb with the requirement that each may be manufactured by a different company, represented as Manufacturer A and B. Other cases can be found with fitness equipment, such as heart rate monitors and cadence sensors.

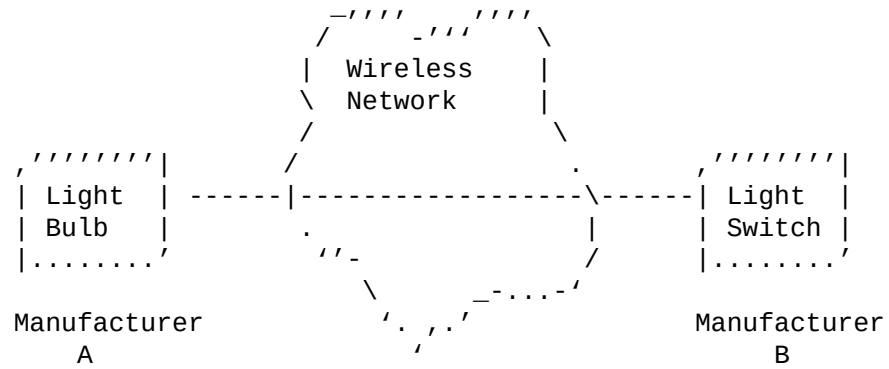


Figure 1: Device-to-Device Communication Pattern

In order to fulfill the promise that devices from different manufacturers are able to communicate out of the box, these vendors need to agree on the protocol stack. They need to make decisions about the following protocol-design aspects:

- Which physical layer(s) should be supported? Does it use low-power radio technologies (e.g., Bluetooth Smart, IEEE 802.15.4)?

- Can devices be IPv6-only, or must they also support IPv4 for backward-compatibility reasons? What IPv4-IPv6 transition technologies are needed?

- Which IP address configuration mechanism(s) is integrated into the device?

- Which communication architectures shall be supported? Which devices are constrained, and what are those constraints? Is there a classical client-server model or rather a peer-to-peer model?

- Is there a need for a service-discovery mechanism to allow users to discover light bulbs they have in their home or office?

- Which transport-layer protocol (e.g., UDP) is used for conveying the sensor readings/commands?

- Which application-layer protocol is used (for example, the Constrained Application Protocol (CoAP) [RFC7252])?

- What information model is used for expressing the different light levels?

- What data model is used to encode information? (See [RFC3444] for a discussion about the difference between data models and information models.)

- Finally, security and privacy require careful thought. This includes questions like: What are the security threats? What security services need to be provided to deal with the identified threats? Where do the security credentials come from? At what layer(s) in the protocol stack should the security mechanism(s) reside? What privacy implications are caused by various design decisions?

This list is not meant to be exhaustive but aims to illustrate that for every usage scenario, many design decisions will have to be made in order to accommodate the constrained nature of a specific device in a certain usage scenario. Standardizing such a complete solution

to accomplish a full level of interoperability between two devices manufactured by different vendors takes time, but there are obvious rewards for end customers and vendors.

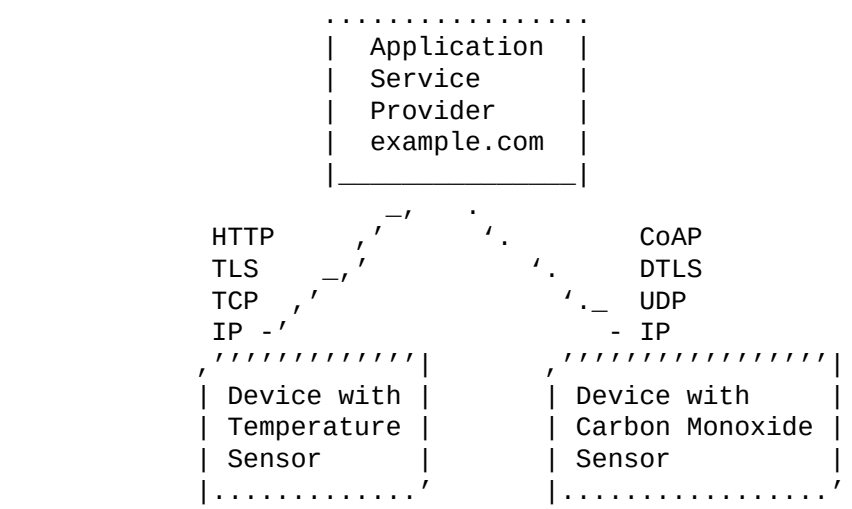
2.2. Device-to-Cloud Communication Pattern

Figure 2 shows a communication pattern for uploading sensor data to an application service provider. Often the application service provider (example.com in our illustration) also sells smart objects. In that case, the entire communication happens internal to the provider and no need for interoperability arises. Still, it is useful for example.com to reuse existing specifications to lower the design, implementation, testing, and development effort.

While this pattern allows using IP-based communication end to end, it may still lead to silos. To prevent silos, example.com may allow third-party device vendors to connect to their server infrastructure as well. For those cases, the protocol interface used to communicate with the server infrastructure needs to be made available, and various standards are available, such as CoAP, Datagram Transport Layer Security (DTLS) [RFC6347], UDP, IP, etc., as shown in Figure 2. A frequent concern from end users is that a change in the business model (or bankruptcy) of the IoT device/service provide might make the hardware become unusable. Companies might consider the possibility of releasing their source code for the IoT device or allowing other IoT operating systems (plus application software) to be installed on the IoT device.

Similarly, in many situations it is desirable to change which cloud service a device connects to, such as when an application service provider changes its hosting provider. Again, standard Internet protocols are needed.

Since the access networks to which various smart objects are connected are typically not under the control of the application service provider, commonly used radio technologies (such as WLAN, wired Ethernet, and cellular radio) together with the network access authentication technology have to be reused. The same applies to standards used for IP address configuration.



TLS = Transport Layer Security

Figure 2: Device-to-Cloud Communication Pattern

2.3. Device-to-Gateway Communication Pattern

The device-to-cloud communication pattern, described in Section 2.2, is convenient for vendors of smart objects and works well if they choose a radio technology that is widely deployed in the targeted market, such as Wi-Fi based on IEEE 802.11 for smart home use cases. Sometimes, less-widely-available radio technologies are needed (such as IEEE 802.15.4) or special application-layer functionality (e.g., local authentication and authorization) has to be provided or interoperability is needed with legacy, non-IP-based devices. In those cases, some form of gateway has to be introduced into the communication architecture that bridges between the different technologies and performs other networking and security functionality. Figure 3 shows this pattern graphically. Often, these gateways are provided by the same vendor that offers the IoT product, for example, because of the use of proprietary protocols, to lower the dependency on other vendors or to avoid potential interoperability problems. It is expected that in the future, more generic gateways will be deployed to lower cost and infrastructure complexity for end consumers, enterprises, and industrial environments. Such generic gateways are more likely to exist if IoT device designs make use of generic Internet protocols and not require application-layer gateways that translate one application-layer protocol to another one. The use of application-layer gateways will, in general, lead to a more fragile deployment, as has been observed in the past with [RFC3724] and [RFC3238].

This communication pattern can frequently be found with smart object deployments that require remote configuration capabilities and real-time interactions. The gateway is thereby assumed to be always connected to the Internet.

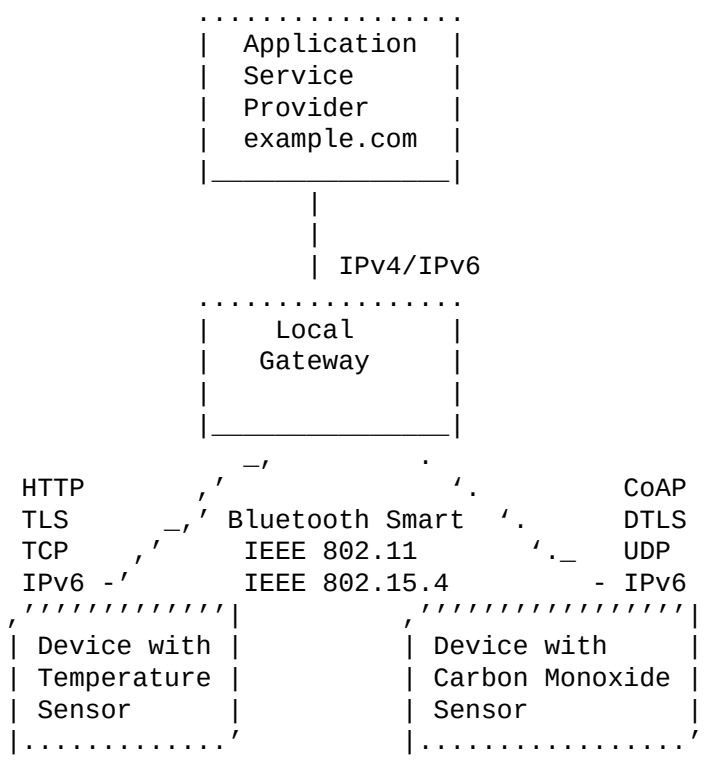


Figure 3: Device-to-Gateway Communication Pattern

If the gateway is mobile, such as when the gateway is a smartphone, connectivity between the devices and the Internet may be intermittent. This limits the applicability of such a communication pattern but is nevertheless very common with wearables and other IoT devices that do not need always-on Internet or real-time Internet connectivity. From an interoperability point of view, it is worth noting that smartphones, with their sophisticated software update mechanism via app stores, allow new functionality to be updated regularly at the smartphone and sometimes even at the IoT device. With special apps that are tailored to each specific IoT device, interoperability is mainly a concern with regard to the lower layers of the protocol stack, such as the radio interface, and less so at the application layer (if users are willing to download a new app for each IoT device).

It is also worth pointing out that a gateway allows supporting both IPv6 and IPv4 (for compatibility with legacy application service providers) externally, while allowing devices to be IPv6-only to reduce footprint requirements. If devices do not have the resources to support both IPv4 and IPv6 themselves, being IPv6-only (rather than IPv4-only) with a gateway enables the most flexibility, avoiding the need to update devices to support IPv6 later, whereas IPv4 address exhaustion makes it ill-suited to scale to smart object networks. See [RFC6540] for further discussion.

2.4. Back-End Data Sharing Pattern

The device-to-cloud pattern often leads to silos; IoT devices upload data only to a single application service provider. However, users often demand the ability to export and to analyze data in combination with data from other sources. Hence, the desire for granting access to the uploaded sensor data to third parties arises. This design is shown in Figure 4. This pattern is known from the Web in case of mashups and is, therefore, reapplied to the smart object context. To offer familiarity for developers, typically a RESTful API design in combination with a federated authentication and authorization technology (like OAuth 2.0 [RFC6749]) is reused. While this offers reuse at the level of building blocks, the entire protocol stack (including the information/data model and RESTful Web APIs) is often not standardized.

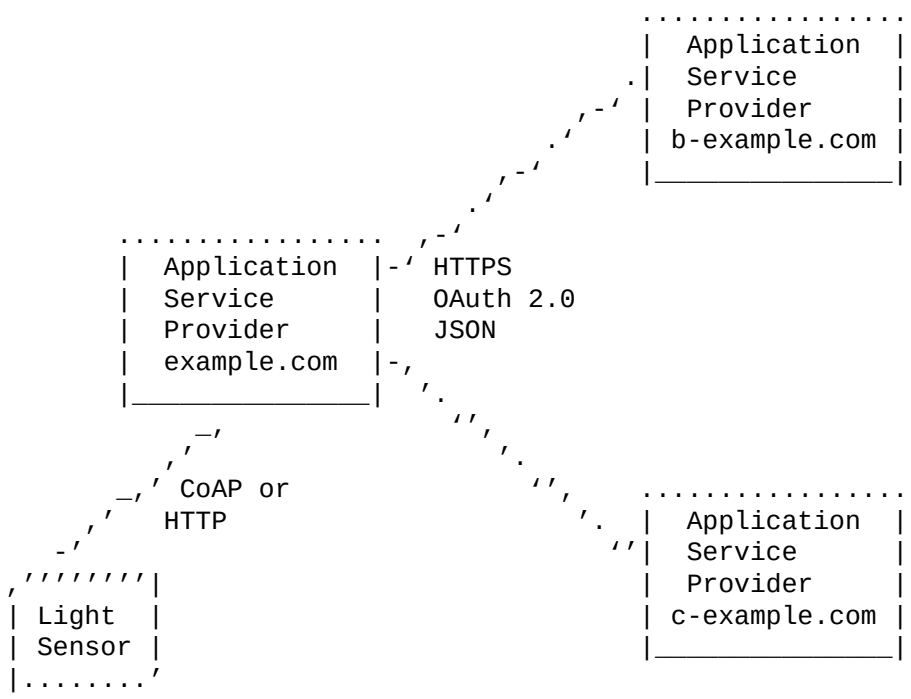


Figure 4: Back-End Data Sharing Pattern

3. Reuse Internet Protocols

When discussing the need for reuse of available standards versus extending or redesigning protocols, it is useful to look back at the criteria for success of the Internet.

RFC 1958 [RFC1958] provides lessons from the early days of the Internet and says:

> The Internet and its architecture have grown in evolutionary fashion from modest beginnings, rather than from a Grand Plan.

And adds:

> A good analogy for the development of the Internet is that of constantly renewing the individual streets and buildings of a city, rather than razing the city and rebuilding it.

Yet, because building very small, battery-powered devices is challenging, it may be difficult to resist the temptation to build solutions tailored to specific applications, or even to redesign networks from scratch to suit a particular application.

While developing consensus-based standards in an open and transparent process takes longer than developing proprietary solutions, the resulting solutions often remain relevant over a longer period of time.

RFC 1263 [RFC1263] considers protocol-design strategy and the decision to design new protocols or to use existing protocols in a non-backward compatible way:

> We hope to be able to design and distribute protocols in less time than it takes a standards committee to agree on an acceptable meeting time. This is inevitable because the basic problem with networking is the standardization process. Over the last several years, there has been a push in the research community for lightweight protocols, when in fact what is needed are lightweight standards. Also note that we have not proposed to implement some entirely new set of 'superior' communications protocols, we have simply proposed a system for making necessary changes to the existing protocol suites fast enough to keep up with the underlying change in the network. In fact, the first standards organization that realizes that the primary impediment to standardization is poor logistical support will probably win.

While [RFC1263] was written in 1991 when the standardization process was more lightweight than today, these thoughts remain relevant in smart object development.

Interestingly, a large number of already-standardized protocols are relevant for smart object deployments. RFC 6272 [RFC6272], for example, made the attempt to identify relevant IETF specifications for use in smart grids.

Still, many commercial products contain proprietary or industry-specific protocol mechanisms, and researchers have made several attempts to design new architectures for the entire Internet system. There are several architectural concerns that deserve to be highlighted:

Vertical Profiles

> The discussions at the IAB workshop (see Section 3.1.2 of [RFC6574]) revealed the preference of many participants to develop domain-specific profiles that select a minimum subset of protocols needed for a specific operating environment. Various standardization organizations and industry fora are currently engaged in activities of defining their preferred profile(s).

Ultimately, however, the number of domains where smart objects can be used is essentially unbounded. There is also an ever-evolving set of protocols and protocol extensions.

However, merely changing the networking protocol to IP does not necessarily bring the kinds of benefits that industries are looking for in their evolving smart object deployments. In particular, a profile is rigid and leaves little room for interoperability among slightly differing or competing technology variations. As an example, Layer 1 through 7 type profiles do not account for the possibility that some devices may use different physical media than others, and that in such situations, a simple router could still provide an ability to communicate between the parties.

Industry-Specific Solutions

The Internet Protocol Suite is more extensive than merely the use of IP. Often, significant benefits can be gained from using additional, widely available, generic technologies, such as the Web. Benefits from using these kinds of tools include access to a large available workforce, software, and education already geared towards employing the technology.

Tight Coupling

Many applications are built around a specific set of servers, devices, and users. However, often the same data and devices could be useful for many purposes, some of which may not be easily identifiable at the time the devices are deployed.

In addition to the architectural concerns, developing new protocols and mechanisms is generally more risky and expensive than reusing existing standards, due to the additional costs involved in design, implementation, testing, and deployment. Secondary costs, such as the training of technical staff and, in the worst case, the training of end users, can be substantial.

As a result, while there are some cases where specific solutions are needed, the benefits of general-purpose technology are often compelling, be it choosing IP over some more specific communication mechanism, a widely deployed link layer (such as wireless LAN) over a more specific one, web technology over application-specific protocols, and so on.

However, when employing these technologies, it is important to embrace them in their entirety, allowing for the architectural flexibility that is built into them. As an example, it rarely makes

sense to limit communications to on-link or to specific media. Design your applications so that the participating devices can easily interact with multiple other applications.

4. The Deployed Internet Matters

Despite the applicability of Internet protocols for smart objects, picking the specific protocols for a particular use case can be tricky. As the Internet has evolved, certain protocols and protocol extensions have become the norm, and others have become difficult to use in all circumstances.

Taking into account these constraints is particularly important for smart objects, as there is often a desire to employ specific features to support smart object communication. For instance, from a pure protocol-specification perspective, some transport protocols may be more desirable than others. These constraints apply both to the use of existing protocols as well as designing new ones on top of the Internet protocol stack.

The following list illustrates a few of those constraints, but every communication protocol comes with its own challenges.

In 2005, Fonseca, et al. [IPoptions] studied the usage of IP options-enabled packets in the Internet and found that overall, approximately half of Internet paths drop packets with options, making extensions using IP options "less ideal" for extending IP.

In 2010, Honda, et al. [HomeGateway] tested 34 different home gateways regarding their packet dropping policy of UDP, TCP, the Datagram Congestion Control Protocol (DCCP), the Stream Control Transmission Protocol (SCTP), ICMP, and various timeout behavior. For example, more than half of the tested devices do not conform to the IETF-recommended timeouts for UDP, and for TCP the measured timeouts are highly variable, ranging from less than 4 minutes to longer than 25 hours. For NAT traversal of DCCP and SCTP, the situation is poor. None of the tested devices, for example, allowed establishing a DCCP connection.

In 2011, the behavior of networks with regard to various TCP extensions was tested in [TCPextensions]: "From our results we conclude that the middleboxes implementing layer 4 functionality are very common -- at least 25% of paths interfered with TCP in some way beyond basic firewalling."

Extending protocols to fulfill new uses and to add new functionality may range from very easy to difficult, as [RFC6709] explains in great detail. A challenge many protocol designers are facing is to ensure

incremental deployability and interoperability with incumbent elements in a number of areas. In various cases, the effort it takes to design incrementally deployable protocols has not been taken seriously enough at the outset. RFC 5218 on "What Makes For a Successful Protocol" [RFC5218] defines wildly successful protocols as protocols that are widely deployed beyond their envisioned use cases.

As these examples illustrate, protocol architects have to take developments in the greater Internet into account, as not all features can be expected to be usable in all environments. For instance, middleboxes [RFC3234] complicate the use of extensions in basic IP protocols and transport layers.

RFC 1958 [RFC1958] considers this aspect and says "... the community believes that the goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network." This statement is challenged more than ever with the perceived need to develop intermediaries interacting with less intelligent end devices. However, RFC 3724 [RFC3724] has this to say about this crucial aspect: "One desirable consequence of the end-to-end principle is protection of innovation. Requiring modification in the network in order to deploy new services is still typically more difficult than modifying end nodes." Even this statement will become challenged, as large numbers of devices are deployed, and it indeed might be the case that changing those devices will be hard. But RFC 4924 [RFC4924] adds that a network that does not filter or transform the data that it carries may be said to be "transparent" or "oblivious" to the content of packets. Networks that provide oblivious transport enable the deployment of new services without requiring changes to the core. It is this flexibility that is perhaps both the Internet's most essential characteristic as well as one of the most important contributors to its success.

5. Design for Change

How to embrace rapid innovation and at the same time accomplish a high level of interoperability is one of the key aspects for competing in the marketplace. RFC 1263 [RFC1263] points out that "protocol change happens and is currently happening at a very respectable clip...We simply propose [for engineers developing the technology] to explicitly deal with the changes rather [than] keep trying to hold back the flood."

In [Tussles], Clark, et al. suggest to "design for variation in outcome, so that the outcome can be different in different places, and the tussle takes place within the design, not by distorting or violating it. Do not design so as to dictate the outcome. Rigid

designs will be broken; designs that permit variation will flex under pressure and survive." The term "tussle" refers to the process whereby different parties, which are part of the Internet milieu and have interests that may be adverse to each other, adapt their mix of mechanisms to try to achieve their conflicting goals, and others respond by adapting the mechanisms to push back.

In order to accomplish this, Clark, et al. suggest to:

1. Break complex systems into modular parts, so that one tussle does not spill over and distort unrelated issues.

2. Design for choice to permit the different players to express their preferences. Choice often requires open interfaces.

The main challenge with the suggested approach is predicting how conflicts among the different players will evolve. Since tussles evolve over time, there will be changes to the architecture, too. It is certainly difficult to pick the right set of building blocks and to develop a communication architecture that will last a long time, and many smart object deployments are envisioned to be rather long lived.

Luckily, the design of the system does not need to be cast in stone during the design phase. It may adjust dynamically since many of the protocols allow for configurability and dynamic discovery. But, ultimately, software update mechanisms may provide the flexibility needed to deal with more substantial changes.

A solid software update mechanism is needed not only for dealing with the changing Internet communication environment and for interoperability improvements but also for adding new features and for fixing security bugs. This approach may appear to be in conflict with classes of severely restricted devices since, in addition to a software update mechanism, spare flash and RAM capacity is needed. It is, however, a trade-off worth thinking about since better product support comes with a price.

As technology keeps advancing, the constraints that technology places on devices evolve as well. Microelectronics have become more capable as time goes by, often making it possible for new devices to be both less expensive and more capable than their predecessors. This trend can, however, be in some cases offset by the desire to embed communications technology in even smaller and cheaper objects. But it is important to design communications technology not just for today’s constraints but also for tomorrow’s. This is particularly important since the cost of a product is not only determined by the

cost of hardware but also by the cost of not reusing already-available protocol stacks and software libraries by developing custom solutions.

Software updates are common in operating systems and application programs today. Without them, most devices would pose a latent risk to the Internet at large. Arguably, the JavaScript-based web employs a very rapid software update mechanism with code being provided by many different parties (e.g., by websites loaded into the browser or by smartphone apps).

6. Security Considerations

Security is often even more important for smart objects than for more traditional computing systems, since interacting directly with the physical world can present greater dangers, and smart objects often operate autonomously without any human interaction for a long time period. The problem is compounded by the fact that there are often fewer resources available in constrained devices to actually implement security (e.g., see the discussion of "Class 0 devices" in Section 3 of [RFC7228]). As such, it is critical to design for security, taking into account a number of key considerations:

- A key part of any smart object design is the problem of how to establish trust for a smart object. Typically, bootstrapping trust involves giving the device the credentials it needs to operate within a larger network of devices or services.

- Smart objects will, in many cases, be deployed in places where additional physical security is difficult or impossible. Designers should take into account that any such device can and will be compromised by an attacker with direct physical access. Thus, trust models should distinguish between devices susceptible to physical compromise and devices with some level of physical security. Physical attacks, such as timing, power analysis, and glitching, are commonly applied to extract secrets [PhysicalAttacks].

- Smart objects will, in many cases, be deployed as collections of identical or near identical devices. Protocols should be designed so that a compromise of a single device does not result in compromise of the entire collection, especially since the compromise of a large number of devices can enable additional attacks such as a distributed denial of service. Sharing secret keys across an entire product family is, therefore, also problematic since compromise of a single device might leave all devices from that product family vulnerable.

- Smart objects will, in many cases, be deployed in ways that the designer never considered. Designers should either seek to minimize the impact of misuse of their systems and devices or implement controls to prevent such misuse where applicable.

- It is anticipated that smart objects will be deployed with a long (e.g., 5-40 years) life cycle. Any security mechanism chosen at the outset may not be "good enough" for the full lifespan of the device. Thus, long-lived devices should start with good security and provide a path to deploy new security mechanisms over the lifetime of the device.

- Security protocols often rely on random numbers, and offering randomness in embedded devices is challenging. For this reason, it is important to consider the use of hardware-based random number generators during early states of the design process.

A more detailed security discussion can be found in the "Report from the Smart Object Security Workshop" [RFC7397] that was held prior to the IETF meeting in Paris, March 2012, and in the report from the National Science Foundation's "Cybersecurity Ideas Lab" workshop [NSF] that was held in February 2014. For example, [NSF] includes, among other recommendations, these recommendations specific to the Internet of Things:

> Enhance the Security of the Internet of Things by Identifying Enclaves: The security challenges posed by the emerging Internet of Things should be addressed now, to prepare before it is fully upon us. By identifying specific use segments, or "enclaves", Internet of Things infrastructure stakeholders can address the security requirements and devise event remediations for that enclave.
>
> Create a Framework for Managing Software Updates: The Internet of Things will challenge our current channels for distributing security updates. An environment must be developed for distributing security patches that scales to a world where almost everything is connected to the Internet and many "things" are largely unattended.

Finally, we reiterate that use of standards that have gotten wide review can often avoid a number of security issues that could otherwise arise. Section 3.3 of [RFC6574] reminds us about the IETF work style regarding security:

In the development of smart object applications, as with any other protocol application solution, security has to be considered early in the design process. As such, the recommendations currently provided to IETF protocol architects, such as RFC 3552 [RFC3552], and RFC 4101 [RFC4101], apply also to the smart object space.

In the IETF, security functionality is incorporated into each protocol as appropriate, to deal with threats that are specific to them. It is extremely unlikely that there is a one-size-fits-all security solution given the large number of choices for the 'right' protocol architecture (particularly at the application layer). For this purpose, [RFC6272] offers a survey of IETF security mechanisms instead of suggesting a preferred one.

7. Privacy Considerations

This document mainly focuses on an engineering audience, i.e., those who are designing smart object protocols and architectures. Since there is no value-free design, privacy-related decisions also have to be made, even if they are just implicit in the reuse of certain technologies. RFC 6973 [RFC6973] and the threat model in [CONFIDENTIALITY] were written as guidance specifically for that audience and are also applicable to the smart object context.

For those looking at privacy from a deployment point of view, the following additional guidelines are suggested:

Transparency: Transparency of data collection and processing is key to avoid unpleasant surprises for owners and users of smart objects. Users and impacted parties must be put in a position to understand what items of personal data concerning them are collected and stored, as well for what purposes they are sought.

Data Collection / Use Limitation: Smart objects should only store personal data that is adequate, relevant, and not excessive in relation to the purpose(s) for which they are processed. The use of anonymized data should be preferred wherever possible.

Data Access: Before deployment starts, it is necessary to consider who can access personal data collected by smart objects and under which conditions. Appropriate and clear procedures should be established in order to allow data subjects to properly exercise their rights.

Data Security: Standardized data security measures to prevent unlawful access, alteration, or loss of smart object data need to be defined and deployed. Robust cryptographic techniques and proper authentication frameworks have to be used to limit the risk of unintended data transfers or unauthorized access.

A more detailed treatment of privacy considerations that extend beyond engineering can be found in a publication from the Article 29 Working Party [WP223].

8. Informative References

[CONFIDENTIALITY]
Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", Work in Progress, draft-iab-privsec-confidentiality-threat-04, March 2015.

[Gamma] Gamma, E., "Design Patterns: Elements of Reusable Object-Oriented Software", 1995.

[HomeGateway]
Eggert, L., "An Experimental Study of Home Gateway Characteristics", In Proceedings of the 10th annual Internet Measurement Conference, 2010, <http://eggert.org/papers/2010-imc-hgw-study.pdf>.

[IPoptions]
Fonseca, R., Porter, G., Katz, R., Shenker, S., and I. Stoica, "IP options are not an option", Technical Report UCB/EECS2005-24, 2005, <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.123.4251>.

[NSF] National Science Foundation, "Interdisciplinary Pathways towards a More Secure Internet", A report on the NSF-sponsored Cybersecurity Ideas Lab held in Arlington, Virginia, February 2014, <http://www.nsf.gov/cise/news/CybersecurityIdeasLab_July2014.pdf>.

[PhysicalAttacks]
Koeune, F. and F. Standaert, "A Tutorial on Physical Security and Side-Channel Attacks", in Foundations of Security Analysis and Design III: FOSAD 2004/2005 Tutorial Lectures; Lecture Notes in Computer Science, Vol. 3655, pp. 78-108, September 2005, <http://link.springer.com/chapter/10.1007%2F11554578_3>.

[RFC1263] O'Malley, S. and L. Peterson, "TCP Extensions Considered Harmful", RFC 1263, October 1991, <http://www.rfc-editor.org/info/rfc1263>.

[RFC1958] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996, <http://www.rfc-editor.org/info/rfc1958>.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002, <http://www.rfc-editor.org/info/rfc3234>.

[RFC3238] Floyd, S. and L. Daigle, "IAB Architectural and Policy Considerations for Open Pluggable Edge Services", RFC 3238, January 2002, <http://www.rfc-editor.org/info/rfc3238>.

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, January 2003, <http://www.rfc-editor.org/info/rfc3444>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003, <http://www.rfc-editor.org/info/rfc3552>.

[RFC3724] Kempf, J., Austein, R., and IAB, "The Rise of the Middle and the Future of End-to-End: Reflections on the Evolution of the Internet Architecture", RFC 3724, March 2004, <http://www.rfc-editor.org/info/rfc3724>.

[RFC4101] Rescorla, E. and IAB, "Writing Protocol Models", RFC 4101, June 2005, <http://www.rfc-editor.org/info/rfc4101>.

[RFC4924] Aboba, B. and E. Davies, "Reflections on Internet Transparency", RFC 4924, July 2007, <http://www.rfc-editor.org/info/rfc4924>.

[RFC5218] Thaler, D. and B. Aboba, "What Makes For a Successful Protocol?", RFC 5218, July 2008, <http://www.rfc-editor.org/info/rfc5218>.

[RFC6272] Baker, F. and D. Meyer, "Internet Protocols for the Smart Grid", RFC 6272, June 2011, <http://www.rfc-editor.org/info/rfc6272>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6540] George, W., Donley, C., Liljenstolpe, C., and L. Howard, "IPv6 Support Required for All IP-Capable Nodes", BCP 177, RFC 6540, April 2012, <http://www.rfc-editor.org/info/rfc6540>.

[RFC6574] Tschofenig, H. and J. Arkko, "Report from the Smart Object Workshop", RFC 6574, April 2012, <http://www.rfc-editor.org/info/rfc6574>.

[RFC6709] Carpenter, B., Aboba, B., and S. Cheshire, "Design Considerations for Protocol Extensions", RFC 6709, September 2012, <http://www.rfc-editor.org/info/rfc6709>.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, July 2013, <http://www.rfc-editor.org/info/rfc6973>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7397] Gilger, J. and H. Tschofenig, "Report from the Smart Object Security Workshop", RFC 7397, December 2014, <http://www.rfc-editor.org/info/rfc7397>.

[TCPextensions] Honda, M., Nishida, Y., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it Still Possible to Extend TCP?", In Proceedings of the ACM Internet Measurement Conference (IMC), Berlin, Germany, November 2011, <http://conferences.sigcomm.org/imc/2011/docs/p181.pdf>.

[Tussles] Clark, D., Wroclawski, J., Sollins, K., and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet", In Proceedings of ACM SIGCOMM, 2002, <http://conferences.sigcomm.org/sigcomm/2002/papers/tussle.html>.

[WP223] Article 29 Data Protection Working Party, "Opinion 8/2014 on the Recent Developments on the Internet of Things", 14/EN, WP 223, September 2014, <http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp223_en.pdf>.

Appendix A. IAB Members at the Time of Approval

Jari Arkko
Mary Barnes
Marc Blanchet
Joel Halpern
Ted Hardie
Joe Hildebrand
Russ Housley
Eliot Lear
Xing Li
Erik Nordmark
Andrew Sullivan
Dave Thaler
Brian Trammell

Acknowledgements

We would like to thank the participants of the IAB Smart Object workshop for their input to the overall discussion about smart objects.

Furthermore, we would like to thank Mike St. Johns, Jan Holler, Patrick Wetterwald, Atte Lansisalmi, Hannu Flinck, Bernard Aboba, Markku Tuohino, Wes George, Robert Sparks, S. Moonsesamy, Dave Crocker, and Steve Crocker in particular for their review comments.

Authors' Addresses

Hannes Tschofenig
ARM Ltd.
6060 Hall in Tirol
Austria

EMail: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Jari Arkko
Jorvas 02420
Finland

EMail: jari.arkko@piuha.net

Dave Thaler
One Microsoft Way
Redmond, WA 98052
United States

EMail: dthaler@microsoft.com

Danny McPherson
12061 Bluemont Way
Reston, VA 20190
United States

EMail: dmcpherson@verisign.com