Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 5, 2015

H. Tschofenig

J. Arkko

D. Thaler

D. McPherson

July 4, 2014

Architectural Considerations in Smart Object Networking
draft-iab-smart-object-architecture-04.txt

Abstract

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.

Table of Contents

1. Introduction

RFC 6574 [1] refers to smart objects (also called "Things", as in Internet of Things in other publications) 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 creates exciting new innovative 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 new 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 designing Internet connected smart objects is even harder since it "requires expertise with Internet protocols in addition to software programming and hardware skills. To simply the development task, and thereby to lower the cost of developing new products and prototypes, we believe that re-use of prior work is essential. Therefore, we provide high-

level guidance on the use of Internet technology for the development of smart objects.

Utilize Existing Design Patterns

Design patterns are generally reusable solutions to a commonly occurring design problem. Existing smart object deployments show patterns that can be re-used by engineers with the benefit of lowering the design effort. Individual patterns also have an implication on the required interoperability between the different entities. Depending on the desired functionality, already existing patterns can be re-used and adjusted. Section 2 talks about various design patterns.

Re-Use Internet Protocols

Most, if not all, smart object deployments can make use of the already standardized Internet protocol suite. The Internet protocols can be applied to almost any environment due to their generic design, and typically offer plenty of potential for re-configuration, which allows the 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 deployments 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, the 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. Utilize Design Patterns

This section illustrates a number of design pattern utilized in the smart object environment. Note that some patterns can be applied at the same time in a product. Developers re-using 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 design pattern where two devices developed by different manufacturers are desired to interoperate. To pick an example from [1], consider a light bulb 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.

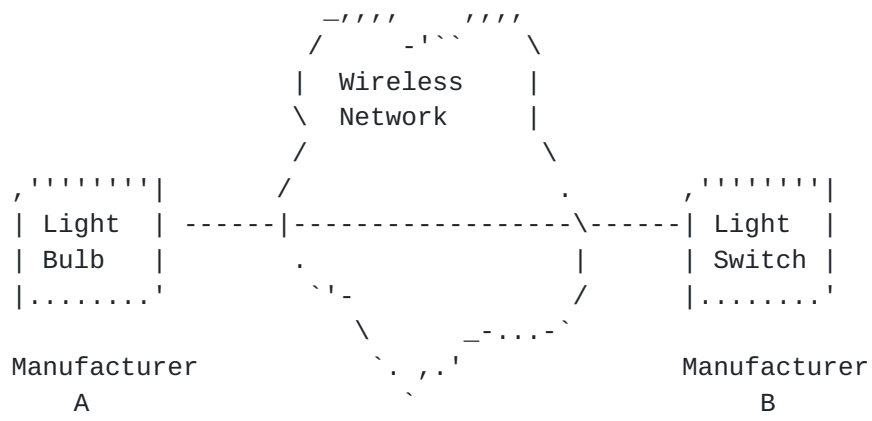

Figure 1: Device-to-Device Communication Pattern

In order to fulfill the promise that devices from different manufacturers are able to communicate out-of-the-box, these vendors need to get together and agree on the protocol stack. Such a consortium needs to make a decision about the following protocol design aspects:

- Which physical layer(s) should be supported?
- Which IP version(s) should be used?
- Which IP address configuration mechanism(s) are integrated into the device?
- Which communication architecture 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 is used for conveying the sensor readings/sensor commands? (e.g., UDP)

- Which application-layer protocol is used? (for example, CoAP)
- How are requests and responses encoded? (e.g., JSON)
- What information model is used for expressing the different light levels? What is the encoding of the information (in a data model)?
- Finally, some thoughts will have to be spent about the security architecture. 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 reside?

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 design pattern for uploading sensor data to a cloud-based infrastructure. Often the application service provider (example.com in our illustration) also sells smart objects as well. In that case the entire communication happens internally to the provider and no need for interoperability arises. Still, it is useful for example.com to re-use 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, DTLS, UDP, IP, etc as shown in Figure 2.

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 re-used. The same applies to standards used for IP address configuration.

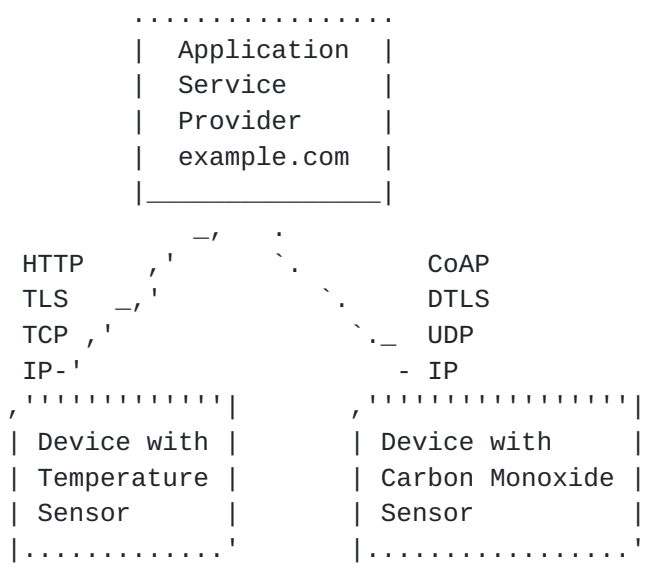

Figure 2: Device-to-Cloud Communication Pattern

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 use choose a radio technology that is widely deployed in the targeted market, such as IEEE 802.11-based Wifi 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. In those cases a gateway has to be introduced into the communication architecture that bridges between the different physical layer/link layer 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.

This design 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.

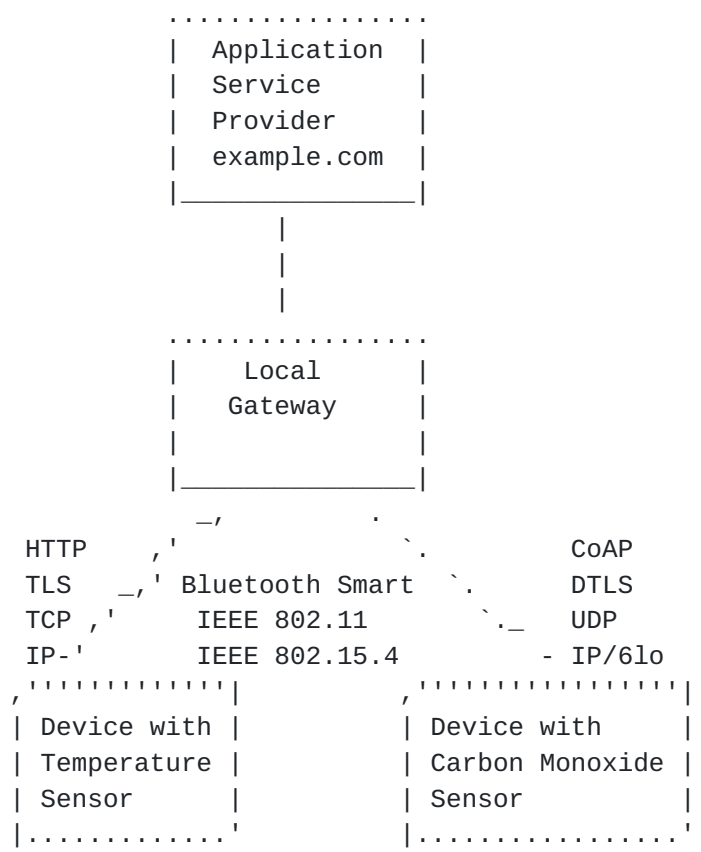

Figure 3: Device-to-Gateway Communication Pattern

A variation of this model is the case where the gateway role is actually incorporated into the smart phone. Of course, if the smart phone is not connected to smart objects, for example because the phone moved out of range, they are not connected with the Internet anymore. This limits the applicability of such a design 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 smart phones with their sophisticated software update mechanism via app stores allow new functionality to be updated regularly at the smart phone 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.

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 urge 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 re-applied 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 [13]) is re-used. While this offers re-use at the level of building blocks, the entire protocol stack (including the data model and the API definition) is often not standardized.

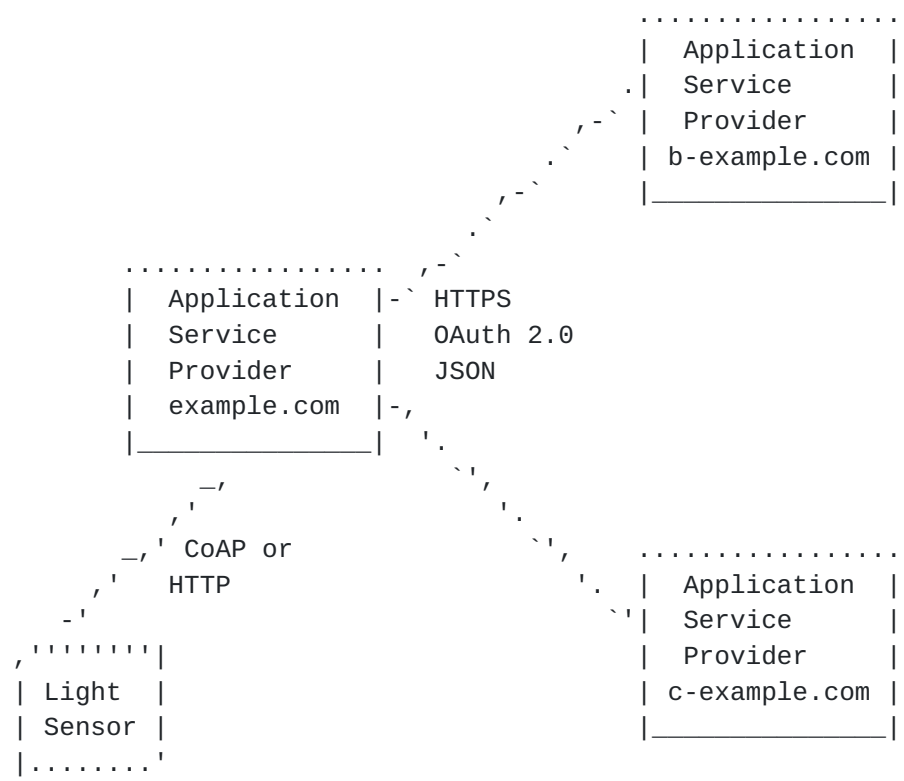

Figure 4: Backend Data Sharing Pattern

3. Re-Use Internet Protocols

When discussing the need for re-use of available standards vs. extending or re-designing protocols, it is useful to look back at the criteria for success of the Internet.

RFC 1958 [6] 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 a specific applications, or even to re-design 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 [4] 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 [4] was written in 1991 when the standardization process was more lightweight than today, these thoughts remain relevant in smart object development.

Interestingly, a large range of already standardized protocols are relevant for smart object deployments. RFC 6272 [5], 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 [1]) 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 web services. 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 that the devices are deployed.

As a result, the following recommendations can be made. First, 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 onto 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 the Internet Protocols for smart objects, picking the specific protocols for a particular use case can be tricky. As the Internet has evolved over time, 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. [15] 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. [17] tested 34 different home gateways regarding their packet dropping policy of UDP, TCP, DCCP, 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, [16] tested the behavior of networks with regard to various TCP extensions: "From our results we conclude 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 [2] 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?" [9] 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 [8] complicate the use of extensions in the basic IP protocols and transport-layers.

RFC 1958 [6] 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 clever intermediaries interacting with dumb end devices. However, RFC 3724 [12] 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 is hard. But RFC 4924 [7] 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 market place. RFC 1263 [4] 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 keep trying to hold back the flood.".

In [18] 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 to predict 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 tradeoff worth thinking about since better product support comes with a price.

As technology keeps advancing, the constraints that the technology places on devices evolve as well. Microelectronics became more capable as time goes by, sometimes making it even 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 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 writing custom protocol stacks and embedded system software.

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 (i.e., by websites loaded into the browser or by smart phone apps).

6. Security Considerations

Section 3.3 of [1] reminds us about the IETF work style regarding security:

> In the development of smart object applications, as with any other protocol application solution, security must be considered early in the design process. As such, the recommendations currently provided to IETF protocol architects, such as RFC 3552 [10], and RFC 4101 [11], 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, [5] offers a survey of IETF security mechanisms instead of suggesting a preferred one.

A more detailed security discussion can be found in the report from the 'Smart Object Security' workshop [14] that was held prior to the IETF meeting in Paris, March 2012.

As current attacks against embedded systems demonstrate, many of the security vulnerabilities are quite basic and remind us about the lessons we should have learned in the late 90's: software has to be tested properly, it has to be shipped with a secure default configuration (which includes no default accounts, no debugging interfaces enabled, etc.), and software and processes need to be available to provide patches. While these aspects are typically outside the realm of standardization, they are nevertheless important to keep in mind.

7. Privacy Considerations

This document mainly focuses on an engineering audience, i.e., those who are designing smart object protocols and architecture. Since there is no value-free design, privacy-related decisions also have to be made, even if they are just implicit in the re-use of certain technologies. RFC 6973 [3] was written as guidance specifically for that audience and it is 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, except in rare cases, 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 Quality: 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.

8. IANA Considerations

This document does not require actions by IANA.

9. 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 Jan Holler, Patrick Wetterwald, Atte Lansisalmi, Hannu Flinck, Joel Halpern, Bernard Aboba, and Markku Tuohino for their review comments.

10. Informative References

[1] Tschofenig, H. and J. Arkko, "Report from the Smart Object Workshop", RFC 6574, April 2012.

[2] Carpenter, B., Aboba, B., and S. Cheshire, "Design Considerations for Protocol Extensions", RFC 6709, September 2012.

[3] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, July 2013.

[4] O'Malley, S. and L. Peterson, "TCP Extensions Considered Harmful", RFC 1263, October 1991.

[5] Baker, F. and D. Meyer, "Internet Protocols for the Smart Grid", RFC 6272, June 2011.

[6] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996.

[7] Aboba, B. and E. Davies, "Reflections on Internet Transparency", RFC 4924, July 2007.

[8] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

[9] Thaler, D. and B. Aboba, "What Makes For a Successful Protocol?", RFC 5218, July 2008.

[10] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[11] Rescorla, E. and IAB, "Writing Protocol Models", RFC 4101, June 2005.

[12] 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.

[13] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[14] Gilger, J. and H. Tschofenig, "Report from the 'Smart Object Security Workshop', March 23, 2012, Paris, France", draft-gilger-smart-object-security-workshop-02 (work in progress), October 2013.

[15] Fonseca, R., Porter, G., Katz, R., Shenker, S., and I. Stoica, "IP options are not an option, Technical Report UCB/EECS", 2005.

[16] Honda, M., Nishida, Y., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it Still Possible to Extend TCP? In Proc. ACM Internet Measurement Conference (IMC), Berlin, Germany", Nov 2011.

[17] Eggert, L., "An experimental study of home gateway characteristics, In Proceedings of the '10th annual conference on Internet measurement'", 2010.

[18] Clark, D., Wroslawski, J., Sollins, K., and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet, In Proc. ACM SIGCOMM", 2002.

Authors' Addresses

Hannes Tschofenig
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
US

Email: dthaler@microsoft.com

Danny McPherson
US

Email: danny@tcb.net