CoRE Working Group
Internet Draft
<draft-bormann-core-6lowpan-fluff-minus-01>
Intended status: Informational
Expires: September 2011

C. Bormann
K. Hartke
Universität Bremen TZI

March 18, 2011

Garrulity and Fluff

draft-bormann-core-6lowpan-fluff-minus-01

Status of this Memo

Copyright Notice

Abstract

In engineering digital networks and systems, we have a hard time getting rid of things. Our Architectures may start strong (at best), but get diluted ever after.

Constrained node/networks may give us a unique chance to build strong protocols again. This note takes the position that we should do that.

1. Introduction

In engineering digital networks and systems, we have a hard time getting rid of things we no longer want.

Our PCs still have BIOSes that can only be understood with a history book and the help of a couple of retired employees. Sometimes the historical baggage is just at the level of some pesky idiosyncrasies. Sometimes it is the source of significant complexities, as anyone trying to set up a device in the PC "architecture" can tell.

Looking at protocols (to pick out one innocent victim: [RFC2131]) with a significant history and a lot of success [RFC5218] quickly gives one the same feeling. Often, an existing protocol [RFC0822] is almost completely repurposed [RFC2045], but cannot get rid of the remnants of the old protocol; this causes not only inefficiencies but above all complexity.

As a strategy for growing meaningful systems by starting simple and growing sideways, this isn't even bad. However, can this layering of complexity over complexity go on indefinitely? Sometimes protocols simply fall out of favor [RFC0959] when there are newer protocols that are more aligned with current requirements [RFC2616]. However, as often, no such replacement happens to appear, and we are saddled with all the accreted complexity.

Constrained nodes often cannot sustain all the complexity of these protocols. In some cases, IETF has started to generate simplified [I-D.ietf-core-coap] or optimized [I-D.ietf-6lowpan-nd] versions of an existing protocol.

The objective of these new designs is not so much to save bytes (this *is* an objective, just not the overriding one), but to reduce complexity. Some of this complexity comes from the multiple layers of re-work existing protocols sometimes exhibit [ARCH]. Some of it also comes from other sources which we will discuss below.

But isn't there a danger that the new designs will accrete all the layers of fat that the existing successful protocols already have? There definitely is. The most important antidote is having a well-defined set of Architectural Constraints, such as the REST considerations that led to the CoAP design [REST]. But still, we have to be vigilant. The forces that pull in additional complexity are strong.

2. The Seven Fallacies

No manifesto is complete without a list of seven somethings. Here, we will describe seven fallacies of protocol design that will need to be actively avoided in the area of constrained node/network design: we have much, much less space for fluff and garrulity than the general Internet.

2.1 The fallacy of giving in to complexity

The introduction has described how protocols manage to *accrete* complexity over time. However, sometimes complexity is *designed* into a protocol.

The reality is complex, so won't our protocols have to be? Actually not, if there is a good set of *architectural constraints* governing the protocol.

Some of this complexity may be needed for business reasons by the users of the protocol. This is the most difficult complexity to avoid — even if it is foreseeable that the need for the complexity will go away due to a changing environment (e.g., in the course of digital convergence). SIP [RFC3261] is one particular victim of this process, and the present essay has little to offer to mitigate it, except maybe for a strong taste about which business requirements should indeed be catered to.

The proverbial source of designed-in complexity is "design by committee". Why is that so? Some problems stem from the shortcomings of committees themselves. In standards work, there also often is a lack of shared architectural vision — even if some of the members are strong on architecture, it is often hard to transfer the more subtle aspect of design taste to everyone. Success here sometimes requires the occasional appeal to authority.

A more dangerous source of complexity is the divergence in perception of objectives. First of all, each committee member has their own personal experience to guide their design taste. Second, the business requirements of their organizations may differ. There may also be different perspective where the area of application (or the industry as a whole) is going. This can lead to mission creep: The protocol is trying to address an ever wider set of requirements (which see below), with less and less focus on important architectural constraints.

This is a particular problem of "attractive protocols". For a while, SIP was such an attractive protocol that there were proposals to do just about everything with SIP, even where SIP provided no appreciable benefit. Attractiveness here is a nuisance to the protocol designer. Unfortunately, a protocol set up for success is quite likely to become "attractive" in this sense.

The worst source of complexity, however, is the need to appease stakeholders. If there is somebody that needs to be won to achieve consensus, that somebody may innocently (not knowing better) ask for changes that destroy the architectural foundations of a protocol. The desire to get another patent into the "mandatory to implement" pool is the worst source of stakeholder "input", but fortunately patents are generally actively shunned in the IETF. Trying to get a "design win" for one's favorite protocol, pure NIH thinking, or the simple desire to be co-author on a draft, are still strong stakeholder forces that must be recognized and resisted.

2.2 The fallacy of design requirements

Some engineering disciplines that do not involve computers have a concept of design requirements, where a specific condition to be fulfilled by the engineering product is foreseen before the design starts (must survive a wind load of …). Software engineering in the past decades has been painfully learning that the 1969 analogy of software design to bridge design is not working, and didn't really capture the essence of bridge design in the first place.

Worse, in protocol design, "requirements" are often used to force a specific design by making sure a "user requirement" is anchored into a set of design requirements that in turn makes sure the competing solution cannot be chosen. Working group members then tend to fight about requirements in order to pre-meditate the selection of their favorite solution.

The IETF has mostly grown to manage this problem by wasting less time on requirements, and, if they are needed, casting them into the form of *objectives*, i.e. tabulating desirable goals for a solution each of which still must be weighed against their cost.

As a best practice, protocols are *designed*, not engineered from a fixed set of requirements: The solution chosen grows in parallel with the understanding of the costs and benefits of each of the objectives chosen.

However, sometimes falling back into "engineering requirement" think is observed; this must then be quickly rooted out.

2.3 The fallacy of protocol requirements

Protocol specifications essentially regulate the behavior of the participating entities, outlawing or requiring some behavior (MUST NOT, MUST [RFC2119]) or encouraging/discouraging others.

It is important to always keep in mind that the objective of protocol specification is interoperability, not conformance. Protocol specifications are written to influence implementors. Other forces also pull on implementers. If there is a short path between points A and B, and the protocol REQUIRES taking a longer path, there is no guarantee that the longer path will indeed be taken. Actually, it is almost certain that a large part of the implementations will simply ignore an onerous requirement.

This is less of a problem with "check-mark requirements" such as the need to implement IPsec with IPv6 — implementers can always resort to building something that gets the checkmark (or doesn't if nobody pays attention), but doesn't really work toward the objectives of that checkmark anyway.

Ignoring MUSTs is not necessarily even a bad thing from an engineering point of view. E.g., when text-based protocols such as SMTP or HTTP became popular, one of their nice properties was that one could write a shell script to implement a useful subset of the protocol. Most definitely, these script-based implementations violated a lot of protocol requirements. Requirements that get in the way of such highly useful but minimal implementations will simply be ignored by a sizable population of implementers, forcing everyone else to implement a protocol that includes all ad-hoc variants of circumventing the requirement.

2.4 The fallacy of overhead

In constrained node/network systems, efficiency is important. This may lead people to sacrifice an architectural principle for a byte shaved off somewhere. The overall cost of an extra byte in a header is minimal compared with the cost of continually having to struggle with the baroqueness of a system for the next couple of decades.

"Premature optimization is the root of all evil."

Optimization is premature unless we already know precisely the characteristics of the environment the protocol will be used in.

(Rule 1 of protocol design: If it is not a factor of 10, ignore it. Well, maybe unless there is *zero* additional complexity.)

Systems that try to reduce overhead by inducing complexity generally get overtaken by leaner systems that because of their agility can simply be further ahead on the curve described by Moore's law.

For example, the CoAP protocol [I-D.ietf-core-coap] has evolved into a clearly separated "message layer" with a "request/response layer" on top. While munging them up might shave off a byte here and there, separating the two layers will allow both for leaner implementations and for faster evolution of each of the layers. The design win cannot be expressed in the number of bytes saved per message (yet), but will become obvious over the

life of the protocol.

There is a flip side to this observation. Wantonly ignoring the constraints of an environment will lead to pain [RFC1925]. (This is sometimes called "premature pessimization".) E.g., using a protocol that assumes multicast is free and highly reliable (as it is on Ethernet and trivially on point-to-point links) in a network where subnet-wide multicast is extremely expensive and unreliable for fundamental reasons, is the wrong approach.

[I-D.ietf-6lowpan-nd] was started as a "culturally compatible" replacement of [RFC4861] for networks with expensive non-local multicast. (Unfortunately, it accreted too many features, and in the middle of its completion, it was replaced by overpainting the undesirable features of [RFC4861] with the salient ones of its replacement; partially to appease stakeholders. Due to the design taste of its creators, the result still is quite good though.)

2.5 The fallacy of flexibility

One of the worst sources of complexity is the desire to be flexible. Making a feature an option (often the result of committee indecision) surprisingly often leads to more code (handling both the case the feature is present and the case it is not) than simply implementing the feature everywhere. Handling multiple slightly different ways to do the same thing also is a source of complexity and unexpected interoperability problems. At least the IETF does not negotiate byte ordering as some other protocols do (see also "overhead" below) and has agreed on using UTF-8 for pretty much everything.

There are other examples where gratuitous flexibility leads to complexity in implementations and interoperability nightmares. E.g., the "RFC 822" header format used in many IETF protocols from [RFC0822] to [RFC2616] and [RFC3261] is both very "flexible" with respect to the case, writing style, and whitespace used and allows any ordering. An "RFC 822" parser pretty much needs to handle an infinite number of ways to express the same semantics.

Contrast this to the design of CoAP, which has strived to provide exactly one way to encode a specific set of semantics. CoAP "options" are sent in order (there is no MUST involved, as the encoding makes sure there simply is no way to send them out of order). Unforeseen cases (e.g., an option with an unsupported length) are handled like unknown options. A CoAP parser is very sure at which place it has to expect what part of the message, so it may be able to process a message entirely on the fly. More importantly, interoperability testing need test only a single variant of encoding a specific set of semantics. (As with any architectural principle, there are limits to how far this can go, but it is good wherever it *can* be achieved.)

(A related set of problems comes up where an encoding surprisingly allows useless cases. In the encoding of length information in IPv4, the length often includes fixed parts, leading to minimal values that MUST NOT be underrun — e.g., the second nibble of the IPv4 header or the option length encoding. This not only wasted potential for extensibility, as with the well-known limitations of the TCP header option space, but also caused surprisingly common implementation errors such as infinite loops in option processing. This wasn't caused by too much flexibility, but by simply not being aware of the principle not even to allow encoding meaningless values.)

A related form of flexibility is the *Postel principle* that is usually quoted as "Be liberal in what you accept and conservative in what you send" — it turns out this principle is much less desirable in the long run than for early experimentation. Taking it to the limit actually is responsible for one of the worst interoperability nightmares of today's Internet: HTML. Once "liberal in what you accept" became a marketing feature for browsers, these started to compete on the forgiveness implemented, as in: my browser can show a (bad) page that your browsers can't (i.e., rightfully reject). As this went on, conforming implementations were less and less able to make sense of the HTML pages out there in the Web. While liberal receiver implementations made minimal creator implementations easier (as in using Notepad to type up a web page), the resulting "race to the bottom" of protocol conformance has mostly destroyed the architecture of HTML as a language (which therefore has recently been resurrected at the DOM level).

Summary: Flexibility is overrated.

Generality, i.e., the lack of artificial limitations, however, is always a good property *if it doesn't generate additional complexity*.

Justifying complexity with the potential future need for a feature is mostly a fallacy, as it is extremely hard to foresee the future. There is only a single requirement that is certain: the need for change. Strong architectural foundations may support the needed change much better than a weak hodgepodge of alternatives none of which quite fits the future.

2.6 The fallacies of cognitive dissonance

There are good reasons to strive for familiarity. Implementers who are familiar with a concept can reach a working implementation in less time, at a lower cost (a point that may get amplified if there is some actual reusability of code).

However, familiarity often is used as an implicit measure of quality by individuals that try to assess a new protocol. Not doing things "the usual way" (a way that the assessors have familiarized themselves with before) causes cognitive dissonance.

Actually, we *need* to design protocols to maintain "familiarity" and to avoid cognitive dissonance wherever a design decision is essentially a bike shed color issue. But we can't have cognitive dissonance guide our architectures; sometimes we do have to ask our implementers to familiarize themselves with the new architecture.

Designing a new protocol to exploit the familiarity with a different protocol may increase the acceptance at first, but will lead to false expectations later. SIP was designed to be "almost, but not entirely unlike" HTTP, and many intuitions of early designers and implementers were led astray by this. A related danger has to be actively avoided all the time in the design of CoAP — "just because HTTP does something in some way" is only a good guidance if this is an expression of a strong architectural principle [REST] or if this is a bike-shed issue (say, status code numbering). In all other cases, CoAP needs to do the thing that is right for CoAP, not necessarily the thing that evolved historically with HTTP and its usage.

A related, but different, problem of cognitive dissonance is the tendency to ignore aspects of reality that are unpleasant. E.g., the IETF has for a long time ignored the presence of NATs on the Internet. While NATs do violate the principles of the Internet architecture, acting as if they were not happening created countless problems for protocols that were willfully designed in the knowledge they would be hard to deploy in NATted environments. Similarly, the design of the host-to-router protocols for Ethernet has ignored the needs for configuration and authorization, which has caused relatively strange protocols such as PPPoE to pop up.

2.7 The fallacy of perfection

In the IETF, we want to create protocols of high quality. This can lead to multi-year "big design up front" efforts, where in reality the agreement on a small base protocol, that is then used as the base of a growing ecosystem, would be more fruitful.

It turns out all successful protocols started relatively small and had a multi-decade lifetime where they adapted to their growing ecosystem (this observation may be a tautology based on our definition of success [RFC5218]). So, on one hand, we need to "design for decades", to enable the protocol to survive the growth. On the other hand, we don't have the slightest idea what turns the uses of the protocol and the resulting changing requirements will take, so designing the protocol to cover all current and potential future protocols is futile (and will lead to lots of unneeded complexity).

Where we are designing replacements for existing protocols, we must be extra careful not to fall into Fred Brook's "second system syndrome". The constrainedness of constrained node/network systems may help us here; e.g., when designing the *observe* feature of CoAP [I-D.ietf-core-observe], we managed to resist the urge to do another baroque publish-subscribe system just because there may be "requirements" for it and we are "familiar" with them.

In the IETF, the desire for high quality often leads to a struggle at the end to convince the

IESG to accept the result. See “appeasing stakeholders”, “big design up front”, “check-mark requirement” above for some of the results. It may be better for a protocol to leave a flank wide open, and look for the actual requirements to be fulfilled in its actual use, then to design in a half-hearted solution appeasing the blocking IESG member that then quickly becomes a piece of fluff when it is replaced (or, worse, over-painted) by the real thing. (The stick of keeping a protocol “experimental” instead of “standards track” is then often used to push those “solutions” through.)

A related result of the struggle for perfection may be strong coupling of unrelated elements. If protocol A needs a solution for a problem that leads to protocol B, there is a natural tendency to make the two protocols depend on each other. If one succeeds and the other needs to be replaced later, it may be hard to surgically remove that strong coupling. (The flip side, of course, is trying to emancipate B into a full protocol if it is just a point solution that *should* stay part of A. These decisions do require design taste.)

3. IANA Considerations

Good architectures have well-defined extensibility points, many of which have to be maintained in the form of registries.

Conversely, the use of registries to punt design decisions is one of the contributors to fluff.

4. Security Considerations

Simple systems with strong architectures are much easier to secure than kitchen sink systems.

5. Acknowledgements

Much of the fervor of the present research note stems from discussions we had at Dagstuhl 11042 — thanks to all the participants that made this a useful event, and in particular to Henning Schulzrinne for further fanning that fervor.

Thanks to Oliver Widder for an excellent, if exaggerated illustration of one of our main problems.

This work was partially funded by the Klaus Tschira Foundation.

6. References

6.1 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2 Informative References

[RFC1925] Callon, R., "The Twelve Networking Truths", RFC 1925, April 1996.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[RFC5218] Thaler, D. and B. Aboba, "What Makes For a Successful Protocol?", RFC 5218, July 2008.

[RFC0822] Crocker, D.H., "Standard for the format of ARPA Internet text messages", STD 11, RFC 822, August 1982.

[RFC2045] Freed, N. and N.S. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", Internet-Draft draft-ietf-core-coap-05 (work in progress), March 2011.

[I-D.ietf-core-observe] Hartke, K. and Z. Shelby, "Observing Resources in CoAP", Internet-Draft draft-ietf-core-observe-02 (work in progress), March 2011.

[I-D.ietf-6lowpan-nd] Shelby, Z., Chakrabarti, S., and E. Nordmark, "Neighbor Discovery Optimization for Low-power and Lossy Networks", Internet-Draft draft-ietf-6lowpan-nd-15 (work in progress), December 2010.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000.

[ARCH] Widder, O., "Architectural Best Practices", 2011, <http://j.mp/gsD4r7>.

Authors' Addresses

Carsten Bormann
Universität Bremen TZI
Postfach 330440
Bremen, D-28359
Germany
Phone: +49-421-218-63921
Fax: +49-421-218-7000
EMail: cabo@tzi.org

Klaus Hartke
Universität Bremen TZI
Postfach 330440
Bremen, D-28359
Germany
Phone: +49-421-218-63905
Fax: +49-421-218-7000
EMail: hartke@tzi.org