CoRE
Internet-Draft
Intended status: Informational
Expires: August 22, 2010

Z. Shelby
Sensinode
M. Garrison Stuber
Itron
D. Sturek
Pacific Gas & Electric
B. Frank
Tridium, Inc
R. Kelsey
Ember

February 18, 2010

CoAP Requirements and Features
draft-shelby-core-coap-req-00

Abstract

This document considers the requirements and resulting features needed for the design of the Constrained Application Protocol (CoAP). Starting from requirements for energy and building automation applications, the basic features are identified along with an analysis of possible realizations. The goal of the document is to provide a basis for protocol design and related discussion.

Status of this Memo

Table of Contents

1. Introduction

The use of web services on the Internet has become ubiquitous in most applications, and depends on the fundamental Representational State Transfer (REST) architecture of the web. The proposed Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). One of the main goals of CoRE is to design a generic RESTful protocol for the special requirements of this constrained environment, especially considering energy and building automation applications. The result of this work should be a Constrained Application Protocol (CoAP) which easily traslates to HTTP for integration with the web while meeting specialized requirements such as multicast support, very low overhead and simplicity.

This document first analyzes the requirements for CoAP from the proposed charter and related application requirement drafts in Section 1.1. The key features needed for the CoAP protocol are then identified in Section 2. Possible ways of realizing each feature are considered and recommendations made where possible. Finally, the applicability of these features to energy, building automation and general M2M applications is considered in Section 3.

1.1. CoAP Requirements

The following requirements for CoAP have been identified in the proposed charter of the working group (Feb 13, 2010 version), in the 6lowapp problem statement [I-D.bormann-6lowpan-6lowapp-problem], or

in the application specific requirement documents. This section is not meant to introduce new requirements, only to summarize the requirements from other sources. The requirements relevant to CoAP can be summarizes as follows:

REQ1: CoRE solutions must be of appropriate complexity for use by nodes have limited code size and limited RAM (e.g. microcontrollers used in low-cost wireless devices typically have on the order of 64-256K of flash and 4-12K of RAM). [charter], [I-D.sturek-6lowapp-smartenergy]

REQ2: Protocol overhead and performance must be optimized for constrained networks, which may exhibit extremely limited throughput and a high degree of packet loss. For example, multihop 6LoWPAN networks often exhibit application throughput on the order of tens of kbits/s with a typical payload size of 70-90 octets after transport layer headers. [charter]

REQ3: The ability to deal with sleeping nodes. Devices may be powered off at any point in time but periodically "wake up" for brief periods of time. [charter], [I-D.sturek-6lowapp-smartenergy], [I-D.gold-6lowapp-sensei]

REQ4: Protocol must support the caching of recent resource requests, along with caching subscriptions to sleeping nodes. [charter]

REQ5: Must support the manipulation of simple resources on constrained nodes and networks. The architecture requires push, pull and a notify approach to manipulating resources. CoAP will be able to create, read, update and delete a Resource on a Device. [charter], [I-D.sturek-6lowapp-smartenergy], [I-D.martocci-6lowapp-building-applications], [I-D.gold-6lowapp-sensei]

REQ6: The ability to allow a Device to publish a value or event to another Device that has subscribed to be notified of changes, as well as the way for a Device to subscribe to receive publishes from another Device. [charter]

REQ7: Must define a mapping from CoAP to a HTTP REST API; this mapping will not depend on a specific application and must be as transparent as possible using standard protocol response and error codes where possible. [charter], [I-D.sturek-6lowapp-smartenergy], [I-D.gold-6lowapp-sensei]

REQ8: A definition of how to use CoAP to advertise about or query for a Device's description. This description may include the device name and a list of its Resources, each with a URL, an interface description URI (pointing e.g. to a Web Application Description Language (WADL) document) and an optional name or identifier. The name taxonomy used for this description will be consistent with other IETF work, e.g. draft-cheshire-dnsext-dns-sd. [charter]

REQ9: CoAP will support a non-reliable multicast message to be sent to a group of Devices to manipulate a resource on all the Devices simultaneously [charter]. The use of multicast to query and advertise descriptions must be supported, along with the support of unicast responses [I-D.sturek-6lowapp-smartenergy].

REQ10: The core CoAP functionality must operate well over UDP and UDP must be implemented on CoAP Devices. There may be optional functions in CoAP (e.g. delivery of larger chunks of data) which if implemented are implemented over TCP. [charter], [I-D.sturek-6lowapp-smartenergy], [I-D.martocci-6lowapp-building-applications]

REQ11: Reliability must be possible for application layer messages over UDP [I-D.sturek-6lowapp-smartenergy].

REQ12: Latency times should be mimimized of the Home Area Network (HAN), and ideally a typical exchange should consist of just a single request and a single response message. [I-D.sturek-6lowapp-smartenergy]

REQ13: Internet media type and transfer encoding type support. [I-D.sturek-6lowapp-smartenergy], [I-D.gold-6lowapp-sensei]

REQ14: Consider operational and manageability aspects of the protocol and at a minimum provide a way to tell if a Device is powered on or not. [charter]

2. CoAP Feature Analysis

This section introduces the minimum set of features needed to realize CoAP, and looks at the possible options for realizing them. These features are considered in light of the requirements listed in Section 1.1. The goal is to consider the cross-dependencies, benefits and drawbacks of alternatives for realizing CoAP and to narrow down the options where obvious.

2.1. Compact Header

There is a requirement for a header overhead appropriate for constrained networks and with limited complexity due to node limitations. The following header design options are considered:

Fixed approach: The simplest approach is to assume as fixed set of byte-aligned fields. The use of variable length fields should be avoided if possible, one obvious exception being a string URL (see Section 2.5). This results in a simple header that can be represented as a struct and easily parsed/created. The disadvantages are difficult evolvability and the tendency to design missing tranport features on-top of CoAP.

Extensible approach: The approach of [I-D.frank-6lowapp-chopan] is to encode HTTP headers as binary tuples, assuming that a large number of optional headers will be needed. A similar approach could be takenin CoAP, giving total header flexibility. The disadvantage is header parsing complexity.

Hybrid approach: It is unclear how much extensibility is really required from the headers of CoAP. Some of the fields in the

protocol will obviously require a sufficient value space for future extensions, such as for indicating content type. Other headers are clearly optional, such as those related to cache control (see Section 2.6) or even the URL (see Section 2.5). A hybrid approach would be to design a small fixed header, with the ability to include extension headers, such as in ICMP [RFC0792].

Considering the features foreseen by this document, some kind of extensible hybrid approach is recommended. Many features are fixed for messages, whereas some are expected to be optional.

2.2. Basic Messages

It is assumed that basic Request and Response messages will be required by the CoAP protocol. This also provides a natural mapping to HTTP (See Section 2.10) and the response may be useful as an acknowledgement in UDP reliability (See Section 2.8.1). It can be considered that CoAP methods are different kinds of Request messages, therefore a separate Request message is not needed.

2.3. REST Methods

The core methods of REST must be supported within CoAP. To minimize confusion with HTTP methods, having their own protocol semantics, in CoAP we call the basic REST methods CREATE, READ, UPDATE, DELETE (CRUD)
[http://en.wikipedia.org/wiki/Create,_read,_update_and_delete].

Additionally, CoAP must support a light-weight Subscribe/Notify mechanism (see Section 2.7). This may require a new NOTIFY method. The discovery mechanism of CoAP may also require a new method called DISCOVER which has different semantics than a READ (see Section 2.9). In order to maintain compatibility with HTTP, these new messages must be mapped to a standard HTTP method. See Section 2.10 for more about HTTP mapping.

2.4. Content-type encoding

In order to support hetergenous uses, it is important that CoAP is transparent to the use of different application payloads. In order for the application process receiving a packet to properly parse a payload, its content-type and encoding should be explicitly known from the header (as e.g. with HTTP). The use of typical binary encodings for XML is discussed in [I-D.shelby-6lowapp-encoding], which includes recommendations for header indication. The draft recommends the indication of at least 10 Internet media types (MIME) [RFC2046] and 2 content transfer encodings.

It is obvious that string names of Internet media types [RFC2046] are not appropriate for use in the CoAP header. But then how to make this small yet extensible? One possible solution is to simply assign codes to a small subset of common MIME and content transfer encoding types and have IANA maintain that. A field of 16-bits should be sufficient for encoding both media and content transfer encoding types. For extending some types, magic numbers can also be used from the beginning of the payload (as defined in associated Internet media type RFCs). This could be indicated by a header value something like "See magic numbers".

2.5. URLs

The Universal Resource Locator (URL) [RFC3986] is an important feature of the REST architecture, the relative part of the URL indicates which resource on the server is being manipulated. It is surely useful for CoAP to support string URLs, which requires a variable length-value field. Although URLs can be designed for compactness, this still often results in 10s of bytes of overhead. The encoding of the URL string also needs to be considered, as this is becoming increasingly complex. It is recommended that only US-ASCII is supported in URL strings for CoAP as defined in [RFC3986], or even a stricter subset as URL parsing is complex and may result in security problems on constrained devices.

Constrained devices are not general purpose web servers, and thus often won't host but a small set of resources with fixed URLs. Thus in addition to string URLs a feature for compressing fixed URLs would be useful.

One way of achieving this would be to assign an integer identifier (7-8 bits should be sufficient) to each fixed URL in an off-line interface description (e.g. Web Application Description Langauge (WADL)) or in its description. This identifier could be encoded in the URL length field instead of the string length with a flag.

2.6. Caching

The cachability of CoAP messages will be important, especially with the sleeping node configurations and power limitations typically found in constrained networks and nodes. What features of cachability are really required and how much energy are we willing to spend on it? Roughly 50% of the HTTP specifications are dedicated to sohpisticated caching. With CoAP we should look at the bare minimum caching feature possible.

Before talking about caching solutiongs, we should consider in what scenarios caching will actually be required. The following two scenarios have been identified:

- An intermediate CoAP proxy may cache resources and answer READ requests using a cached version. The resource may be cached from previous responses or notifications. This requires at least Max-Age cache control information about each resource.

- An intermediate CoAP proxy may cache subscriptions to a sleeping node. This requires at least Max-Age information about the subscription.

Three possible approaches have been identified for caching support.

In-band approach: One approach is suggested in [I-D.frank-6lowapp-chopan], which analyses the subset of features from HTTP that could be used for simple sensor data purposes. The proposal is that simply using the using the HTTP Age header (for resource age) and Cache-Control header (for max-age). Max-age may also be applied in requests. Both headers make use of a 2-byte value in seconds. The advantage of this approach is that cache control information is easily available from the header. The disadvantage is some header overhead.

Out-of-band approach: Here the CoAP protocol would be agnostic to the cachability of the resources it is carrying, instead leaving the definition of cache control parameters to the body of the resources in an application specific way. The disadvantage is that this makes proxies dependent on the application.

Discovery approach: In this approach the cache control information for resources is defined off-line in the list of a Device's resources. This approach is used e.g. in the SENSEI system

[I-D.gold-6lowapp-sensei]. The disadvantage is that the caching is dependent on the profile, which may not be a problem

if the cache information is in a universal format (see Section 2.9).

Based on the current analysis the In-band approach would be reasonable for CoAP considering the requirements. This doesn't prevent the inclusion of cachability information in a resource description as well.

2.7. Subscribe/Notify

CoAP is required to integrate a push model for interaction in addition to traditional request/response. This means that interested clients could subscribe to a resource (a URL), and receive notifications to a call-back URL of their choice. In its most basic form a notification would be sent each time the resource changes. There are many issues to consider including managing subscription leasing and timeouts, how to batch multiple changes and how to tune notification times. Before considering the details, there are a two general models possible for realizing the Subscribe/Notify mechanism:

Resource: Subscribe is realized using CREATE on a well known resource (e.g. /subsribe) with the URL of the resource of interest and a URL call-back in the body). Notifications would be made using a NOTIFY (or alternatively UPDATE) message to the call-back URL. Likewise, de-subscribe is realized using DELETE on the same well known resource with the URL in the body. Notifications would cease after the DELETE.

Watch: This method would require a CREATE to a new URI to "create" a new watch resource. UPDATE is then used to add/remove a set of URIs being "watched" along with call-backs.

Subscribe: An alternative to using CREATE on /subscribe or to make a new watch resource, would be to develop an explicit SUBSCRIBE method which is used directly on the URL of interest. The body of the SUBSCRIBE would include the call-back URL and other subscription information.

The mapping requirement of CoAP with HTTP requires that the mapping

between methods should be as simple as possible. Therefore the addition of new methods such as NOTIFY and SUBSCRIBE should be done with care. As NOTIFY is a push concept, this may at least be justified.

The complexity of subscription management should also be carefully considered when working with constrained devices. An increasing number of subscription options leads to greater complexity, especially when dealing with multiple subscriptions to a resource.

Subscription options to a proxy via HTTP may be quite a bit more complex than those via CoAP to a constrained device.

2.8. Transport Binding

The CoAP protocol will operate by default over UDP. There may be OPTIONAL functions in CoAP (e.g. delivery of larger chunks of data) which if implemented are implemented over TCP. In this section we look at transport issues.

2.8.1. UDP

The goal of binding CoAP to UDP is to provide the bare minimum features for the protocol to operate over UDP, going nowhere near trying to re-create the full feature set of TCP. The bare minimum features required would be:

- Stop-and-wait would be sufficient for reliability. A simple response message itself would suffice as an acknowledgement with retransmission support. Not all requests require reliability, thus this should be optional. Performance is not the key here and for more sophisticated reliability and flow control TCP could be used.

- A sequence number (transaction ID) is needed to match responses to open requests and would be generated by the client. A 12-16 bit unsigned interger would be sufficient. [I-D.frank-6lowapp-chopan] also considered this solution.

- Multicast support. Providing reliability with a multicast destination address would be very complex. Therefore the goal is to provide a non-reliable multicast service. In many cases there

may not be a response to a multicast message. A multicast command might result in an action being taken at a device, but no response being sent. Therefore a multicast request may be answered with a unicast response, however without reliability (retransmission e.g.).

2.8.2. TCP

The CoAP protocol also may also make use of TCP for some features. As TCP provides a reliable stream this binding does not require anything special from the CoAP protcol design. The same basic messages could be applied over TCP without stop-and-wait. A transaction ID should still be used over TCP. The question is for which features, or in which configurations would TCP be recommended? The following have been identified so far:

- Delivering a large chunk of data.
- Delivering a continuous stream of data, for example streaming sensor readings for a long period.
- TCP may also be useful for providing congestion control if CoAP is being applied across the wider Internet.

2.9. Resource Discovery

This document assumes that an application either a) knows that CoAP services are running on a default or pre-configured port in the network or b) a service discovery method has already been used to locate CoAP services. Thus service discovery is not considered in this document.

CoAP is required to support the querying and advertisement of resources offered by CoAP services. In order to achieve this, the protocol would need to suport multicast with optional responses for discovery, along with unicast or multicast advertisement of resource descriptions. A well-known resource (e.g. /resources) could be used to enable discovery through a new DISCOVER method (or alternatively a READ). The response to a DISCOVER message would include a list of resource URLs available, an optional URI to an interface description and an optional name or identifier. CoAP services could also

advertise their description by sending e.g. a multicast NOTIFY to /resources, or by posting their description to a central place (e.g. a proxy) with a CREATE to /resources.

In order to save overhead as descriptions for some nodes could become long, nodes could apply a technique similar to that used in XMPP. By default Devices would advertise a hash of their description or a hash of each resource in the description. A receiver would be able to match hashes of already known resources, and could DISCOVER the resource for its full description.

2.10. HTTP Mapping

It shall be possible to map from CoAP directly to HTTP, CoAP however only offers a small subset of the HTTP protocol features. As a result, programs implementing translation between HTTP and CoAP must either implement other HTTP 1.1 commands on behalf of the CoAP nodes (e.g. LINK, TRACE, OPTIONS), or must reject such request. The primary responsibility of a program translating between HTTP and CoAP is to rewrite the headers, translating between the highly optimized CoAP headers and plain text HTTP headers. It must also manage/ maintain TCP sessions necessary for HTTP. Depending on how some of the features of CoAP are realized, the mapping may also need to make

further translations for subscription or caching.

Subscription (see the exampe below) will require a proxy mapping CoAP-HTTP to support some kind of LONG POLL method as being defined in the HyBi WG in order to avoid continuous polling of a subscription status [draft-loreto-http-bidirectional-01].

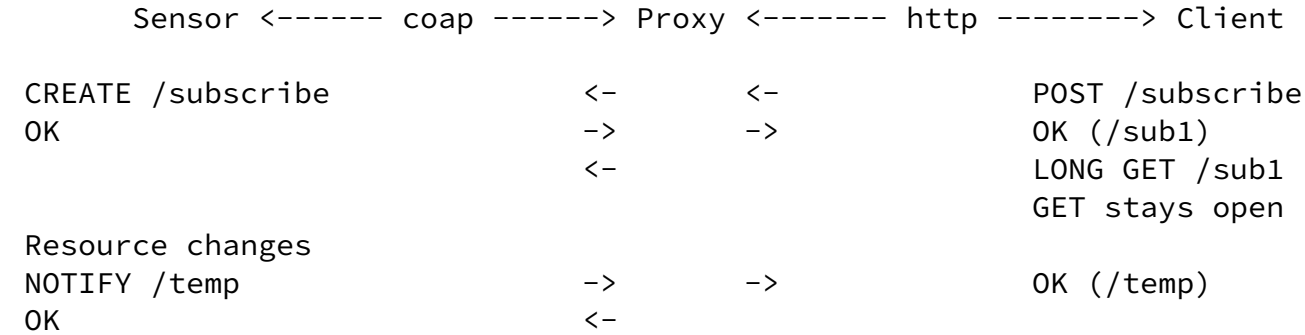

Figure 1: Example CoAP-HTTP mapping for subscription.

3. Applicability

This sections looks at the applicability of the CoAP features for energy, building automation and other macine-to-machine (M2M) applications.

3.1. Energy Applications

Rising energy prices, concerns about global warming and energy resource depletion, and societal interest in more ecologically friendly living have resulted in government mandates for Smart Energy solutions. In a Smart Energy environment consumers of energy have direct, immediate access to information about their consumption, and are able to take action based on that information. Smart Energy systems also allow device to device communication to optimize the transport, reliability, and safety of energy delivery systems. While often Smart Energy solutions are electricity-centric, i.e. Smart Grid, gas and water are also subject to the same pressures, and can benefit from the same technology.

Smart Energy Transactions typically include the exchange of current consumption information, text messages from providers to consumers, and control signals requesting a reduction in consumption. Advanced features such as billing information, energy prepayment transactions, management of distributed energy resources (e.g. generators and photo-voltaics), and management of electric vehicles are also being developed.

Smart Energy benefits from Metcalfe's Law. The more devices that are part of a smart energy network within the home or on the grid, the more valuable it becomes. Showing a consumer how much energy they are using is useful. Combining that with specific information about their major appliances, and enabling them to adjust their consumption based on current pricing and system demand is much much more powerful. To do this however requires a system that is resillient, low cost, and easy to install. In many areas this is being done with systems built around IEEE 802.15.4 radios. In the United States, there are over 30 million electric meters that will be deployed with these radios. These radios will be combined to form a mesh network,

enabling Smart Energy communication within the home. The maximum packet size for IEEE 802.15.4 is only 127 bytes. Additionally, there is the well known issue of how TCP manages congestion working sub-optimally over wireless networks. IEEE 802.15.4 is ideal for these applications because of its low cost and its support for battery powered devices; however, it is not as well suited for heavier protocols like HTTP. These technical issues with IEEE 802.15.4 networks combined with a desire to facilitate broader compatibility, makes a protocol like CoAP desireable. Its REST architecture will allow seamless compatibility with the rest of the Internet, allowing it to be easily integrated with web browsers and web-based service providers, while at the same time being appropriately sized for the low-cost networks necessary for its success.

3.2. Building Automation

Building automation applications were analyzed in detail including use cases in [I-D.martocci-6lowapp-building-applications]. Although many of the embedded control solutions for building automation make use of industry-specific application protocols like BACnet over IP, there is a growing use of web services in building monitoring, remote control and IT integration. The OASIS oBIX standard [ref] is one example of the use of web services for the monitoring and interconnection of heterogeneous building systems. Several of the CoAP requirements have been taken from [I-D.martocci-6lowapp-building-applications]. The resulting features should allow for peer-to-peer interactions as well as node-server request/response and push interfactions for monitoring and some control purposes. For building automation control with very strict timing requirements using e.g. multicast, further features may be required on top of CoAP.

3.3. General M2M Applications

CoAP provides a natural extension of the REST architecture into the domain of constrained nodes and networks, aiming at requirements from automation applications in energy and building automation. A very

wide range of machine-to-machine (M2M) applications have similar requirements to those considered in this document, and thus it is foreseen that CoAP may be widely applied in the industry. One standardization group considering a general M2M architecture and API

is the ETSI M2M TC [ref], which considers a wide range of applications including energy. Another group developing solutions for general embedded device control is the OASIS Device Proile Web Services (DPWS) group. The consideration of DPWS over 6LoWPAN is available in [I-D.moritz-6lowapp-dpws-enhancements].

4. Conclusions

This document analyzed the requirements associated with the design of the foreseen Constrained Application Protocol (CoAP). Based on these requirements a list of minumum features was analyzed along with different options for realizing them. If possible a recommendation was also made where obvious. Finally, the identified features of CoAP are considered for energy, building automation and M2M applications. This document is meant to serve as a basis for the design of the CoAP protocol and relevant discussion.

CoAP is proposed as a transport agnostic extension of REST for deployment in confined computing environments. The intent is to align CoAP with HTTP wherever possible to leverage the web services computing environment already in place.

Whereas REST envisions just 4 primitives (CRUD), CoAP may propose to extend this paradigm with e.g. a NOTIFY primitive to enable publish/ subscribe along with a DISCOVER primitive to support multicast discovery of services denoted by URL. The main architectural difference between READ and the new discovery primitive is the support for multicast and a possible matching feature.

Finally, CoAP seeks to preserve the caching facilities of HTTP and extend that capability for power saving devices that are not always active on the network.

5. Security Considerations

Some of the features considered in this document will need further security considerations during a protocol design. For example the use of string URLs may have entail security risks due to complex processing on limited microcontroller implementations.

The CoAP protocol will be designed for use with e.g. (D)TLS or object security. A protocol design should consider how integration

with these security methods will be done, how to secure the CoAP header and other implications.

6. IANA Considerations

This draft requires no IANA consideration.

7. Acknowledgments

Thanks to Cullen Jennings, Guido Moritz, Peter Van Der Stok, Adriano Pezzuto, Lisa Dussealt, Gilbert Clark and Salvatore Loreto for helpful comments and discussions.

8. References

8.1. Normative References

[I-D.frank-6lowapp-chopan]
Frank, B., "Chopan - Compressed HTTP Over PANs", draft-frank-6lowapp-chopan-00 (work in progress), September 2009.

[I-D.gold-6lowapp-sensei]
Gold, R., Krco, S., Gluhak, A., and Z. Shelby, "SENSEI 6lowapp Requirements", draft-gold-6lowapp-sensei-00 (work in progress), October 2009.

[I-D.martocci-6lowapp-building-applications]
Martocci, J. and A. Schoofs, "Commercial Building Applications Requirements", draft-martocci-6lowapp-building-applications-00 (work in progress), October 2009.

[I-D.shelby-6lowapp-encoding]
Shelby, Z., Luimula, M., and D. Peintner, "Efficient XML Encoding and 6LowApp", draft-shelby-6lowapp-encoding-00 (work in progress), October 2009.

[I-D.sturek-6lowapp-smartenergy]
Sturek, D., Shelby, Z., Lohman, D., Stuber, M., and S. Ashton, "Smart Energy Requiements for 6LowApp", draft-sturek-6lowapp-smartenergy-00 (work in progress), October 2009.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

8.2. Informative References

[I-D.bormann-6lowpan-6lowapp-problem]
Bormann, C., Sturek, D., and Z. Shelby, "6LowApp: Problem Statement for 6LoWPAN and LLN Application Protocols", draft-bormann-6lowpan-6lowapp-problem-01 (work in progress), July 2009.

[I-D.moritz-6lowapp-dpws-enhancements]
Moritz, G., "DPWS for 6LoWPAN", draft-moritz-6lowapp-dpws-enhancements-00 (work in progress), December 2009.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com

Michael Garrison Stuber
Itron
2111 N. Molter Road
Liberty Lake, WA 99025

U.S.A.

Phone: +1.509.891.3441
Email: Michael.Stuber@itron.com

Don Sturek
Pacific Gas & Electric
77 Beale Street
San Francisco, CA
USA

Phone: +1-619-504-3615
Email: d.sturek@att.net

Brian Frank
Tridium, Inc
Richmond, VA
USA

Phone:
Email: brian.tridium@gmail.com

Richard Kelsey
Ember
47 Farnsworth Street
Boston, MA 02210
U.S.A.

Phone: +1.617.951.1201
Email: richard.kelsey@ember.com