CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 13, 2012

J. Nieminen, Ed.
B. Patil
T. Savolainen
M. Isomaki
Nokia
Z. Shelby
Sensinode
C. Gomez
Universitat Politecnica de Catalunya/i2CAT
M. Ersue
Nokia Siemens Networks

March 12, 2012

Constrained Application Autoconfiguration draft-nieminen-core-service-discovery-02

Abstract

The number and types of devices that are Internet connected continues to grow. Sensors, appliances, utility meters, medical devices etc. are Internet connected for various reasons. Many of these newer devices are constrained in terms of processing power, memory, communication capability and, available power. Devices such as sensors and similar very small devices often lack a proper user interface and hence configuring even the most basic parameters for enabling Internet connectivity on these is extremely difficult. Some of the existing configuration protocols can help in autoconfiguring various parameters of the IP stack needed for Internet connectivity. However this is not sufficient if the devices are using a web service application. There is a need for additional information such as service provider name and username/password etc. for authentication etc. A configuration protocol solution for resource constrained devices is needed in order to enable the potential enabled by Internet connectivity. This document outlines the use cases and requirements for user friendly configuration of such information on a constrained device, and specifies a Constrained Application Protocol (CoAP) based mechanism to meet the requirements.

Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 13, 2012.

Table of Contents

1. Introduction

Communication via IP over radio links such as 802.15.4 (aka Zigbee) [RFC4944] and Bluetooth Low Energy (BT-LE) [I-D.ietf-6lowpan-btle] have been specified. This has enabled very constrained devices such as sensors and various types of gadgets to become Internet connected. Various types of web services and applications leverage the information and capability of such devices. The 6LoWPAN working group in the IETF has defined an optimized approach to the transmission of IPv6 over low power radio links. Additionally the Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] provides a standard way of implementing web services in a compact manner in environments wherein the device is constrained in terms of processing power, memory, communication capability and, energy.

Devices such as sensors or other types of appliances may not have a proper user interface (UI). There may be no display, keyboard, buttons or other ways to interact in order to configure them. Some form of external configuration capability is needed for such devices. The IP stack itself can be autoconfigured via IPv6 stateless autoconfiguration [RFC4862]. However in addition to the IP stack, the application running in the constrained device, such as a CoAP client reporting temperature measurements, needs some amount of configuration. Typically it needs to know at least a (CoAP) servers IP address or a Uniform Resource Locator (URL), and often also the credentials to authenticate with to the server (in the simplest form a username and password). This type of configuration is usually impossible to perform without some kind of user intervention. For instance, the user may need to decide which temperature collection service provider and account to use.

Various existing solutions could be considered. Protocols such as DHCP [RFC2131] or Multicast DNS [I-D.cheshire-dnsext-multicastdns] could be used to discover and deliver configuration information within the local domain. Often, devices without a UI, such as Wi-Fi access points or home routers, are configured via a web browser. WiFi APs/routers run a simple web (HTTP) server with a set of HTML forms which are used for configuring them via a browser client from a UI capable device such as computer or tablet. This type of approach to configuration may be very good for many types of devices. It may however not be feasible to support such protocols and mechanisms in the most extremely constrained devices, that have just enough room for a simple CoAP implementation.

This document outlines the use cases and requirements for user friendly application configuration for constrained devices. It defines a CoAP based mechanism for automatic configuration discovery and delivery between a constrained device and another host in its

local network. It also defines a standard format for the basic application configuration information. Finally, it discusses the applicability of this solution, describing the requirements of the environment where it can be securely used.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

Constrained device

The host that obtains configuration infromation for its application from the helper device. For this purpose it runs a configuration client using CoAP protocol.

Helper device

The host that provides configuration information for an application on the constrained device. For this purpose it runs a configuration server using CoAP protocol.

Configuration client

A CoAP client program searching for, fetching and saving the configuration data on the constrained device.

Configuration server

A CoAP server program having configuration data and responding to the requests sent by configuration clients.

2. Use cases and Requirements

The main use cases lie with devices that are sold directly to consumers and are supposed to be able to communicate over the Internet with any service provider of user's choice. This means that the device can't be pre-configured to work with a particular service provider when the consumer purchases it. This is analogous with installing a new IMAP mail client or buying a generic SIP phone. One of the first things the user needs to do is to configure them to work with her chosen service provider.

The device could be for instance a Bluetooth Low Energy capable wrist

unit, that supports some kind of a standard application protocol agreed by the sport wrist vendors, so it could talk to any compliant Internet service. The user would pair it with her smartphone for Internet connectivity. Or, the device could be a temperature sensor connected to a home access point that supports some open temperature reporting protocol supported by more than one service provider. Before these devices can do anything useful, they need information about the service provider and credentials for the user account.

One assumption made is that the device is extremely constrained. It uses CoAP as the protocol to communicate with its server, but can't afford to support many additional protocol stacks. For instance, running an HTTP server with HTML forms just for configuration would be impractical.

Another assumption is that the consumer has another device at her use that does have a user interface and can be used in the configuration process. For instance, she can have a smartphone or a tablet. In the wrist unit example this is quite clear, as something like a smartphone is used as a router for Internet connectivity. In the home network case, the user would likely have other devices connected to the network via Wi-Fi, for instance. It is assumed that the configuration can be either manually entered or delivered in some way to this helper device. The exact method how tha is done is out of scope of this document, but there are ways such as Short Message Service (SMS) or Open Mobile Alliance Device Management to deliver configuration for mobile phones. It would also be very simple for a service provider to have a website where the user could save the configuration information on the helper device via its browser. If the service provider has a dedicated application, that could be used. Entering the information manually should not be discluded either. After all, this is how many people manage to configure their IMAP accounts.

The requirements stemming from the use case and assumptions given are:

1. There must be a way for the configuration client in the constrained device to search and discover whether any of the hosts in its local network running a configuration server that is able provide the client with suitable application configuration information.

2. It must be possible to identify the type of application for which the configuration is valid. (This is to avoid the case where a refridgerator is accidentally given the configuration meant for a coffee machine.) Also, it must be possible to identify the exact device/application instance for which the configuration is valid.

(This is to deal with cases such as that the user has two similar wrist units, but wants them to use different service providers or user accounts.)

3. It must be possible, after discovery, for the configuration client to fetch the configuration information from the configuration server.

4. It must be possible for the configuration client in the constrained device and the configuration server in the helper device to authenticate and authorize each other, and ensure the confidentiality and integrity of the configuration request and the configuration information transfered between the two. The configuration server should only provide the configuration information for an authenticated and authorized client, and the client should only accept configuration information from an authenticated and authorized server. Authorization may require user interaction.

3. CoAP Based Configuration Discovery and Delivery

This Section defines how a constrained device running a configuration client uses CoAP to discover and fetch configuration for a CoAP based application from a helper device running a configuration server.

Discovery is based on CoRE link format [I-D.ietf-core-link-format]. To perform discovery, the configuration client sends a CoAP GET request either to the IP address of its default gateway, or to a well known multicast address. The request is targeted to URI /.well-known/core and the Resource Type parameter is set with the value "core-aconf" in the query string. Upon success, the response will contain a link format entry for a suitable configuration formatted in JSON [RFC4627] as recommended in [I-D.bormann-core-links-json].

Implementations MUST support the discovery request to the default gateway, where sending the discovery request to a well known multicast address is seen as OPTIONAL.

OPEN ISSUE: It could be possible to contact a configuration outside the local network as well, but that would presumably require some preconfiguration in the client.

It is mostly an implementation issue, when exactly the configuration client searches for configuration. It MUST do it when the constrained device is booted up, and the configuration client starts, and it determines it has no existing configuration. It SHOULD do it at start-up also when it does have existing configuration, to check

if there is a new version available. It SHOULD also do it if the actual CoAP application is not able to connect to its server or access its user account, to make sure configuration is still valid. It is RECOMMENDED that the constrained device also has some kind of a mechanism for the user to explictly ask for a new configuration to be fetched. This could be a physical button or some other procedure.

OPEN ISSUE: The query should also contain the application type and potentially a device identifier. How should these be encoded?

After the constrained device has received a CoAP response with the link for the configuration information in the payload, it issues a CoAP GET request to the URI in the link to retrieve the actual configuration information. A successful response will include the configuration information in its payload.

Basic configuration information contains three pieces: Server IP address, username and password. Only the server IP address is mandatory. The configuration information is encoded in JSON as shown in the example below.

3.1. Example

The constrained device searches for an available configuration service and gets a response from a host having that resource:

```
REQ: GET /.well-known/core?rt=core-aconf

RES: 2.05 "Content"
</config/app>;rt="core-aconf"
```

The constrained device then fetches the configuration from the resource it discovered:

```
REQ: GET /config/app

RES: 2.05 "Content"
[{ address : "host",
   username : "Foo",
   pwd : "S.%$"!"
}]
```

The value for "host" is interpreted as described in Section 6.1 of [I-D.ietf-core-coap]. If host is provided as an IP-literal or IPv4address, then the server is located at that IP address. If host is a registered name, then that name is considered an indirect

identifier and the end-point might use a name resolution service, such as DNS, to find the address of that host.

4. Security Considerations

NOTE: This section still needs more work.

The security considerations described throughout [I-D.ietf-core-coap] apply here as well.

User and device specific configuration information is highly sensitive. At least the following types of attacks are possible against the CoAP based configuration mecahnism, unless proper protection is in place:

1. An attacker can reply to configuration client's requests, pretending to be a valid configuration server, and provide the client with false configuration information. This can make the actual application client to connect to a wrong service or a wrong user account. False configuration can also be entered by an active man-in-the-middle with an access to the communication medium and ability to alter messages between the client and the server.

2. An attacker can send out configuration requests and obtain configuration information that it is not supposed to get. With this information, the attacker may be to access user's account for the service the configuration is valid. Congfiguration can also be obtained by a passive eavesdropper with access to the communication medium between the client and the server.

To protect against these attacks, the configuration client and server need to have a mechanism to autenticate each other and protect both integrity and confidentiality of the configuration information.

There are three approaches that can provide the necessary security, each having their pros and cons:

1. CoAP over DTLS between the configuration client and server. This is the default security mechanism for CoAP, as described in Section 10.1 of [I-D.ietf-core-coap]. The challenge lies with the provision of keys. It is possible to use either pre-shared key, raw public keys, or certificates. It can be assumed that a pre-shared key or client's public key can be set on the server as part of the configuration process. However, setting the pre-shared key or server's public key on the constrained device is more problematic due to its limitations. It may also be

impractical to have a DTLS stack with X.509 certificate check to be implemented in the types of devices this configuration scheme is targeting to. With client's public key, the server could authenticate the client and communications could be secured. This would still leave it unresolved how the client could authenticate the server. One option is that the constrained device comes pre-configured with some DTLS key just for the configuration purpose, that the user has to keep secret.

2. CoAP over DTLS between the configuration client and server, with additional JSON object security for the configuration information carried as CoAP payload. This differs from the previous option so that the configuration information would be encrypted and signed separately.

3. Link layer security between the constrained device and the helper device. This can be only used in environments where the constrained device is actually directly connected to the helper device over a single link. An example of such environment is an accessory connected to a smartphone over BT-LE. In that case BT-LE's link layer pairing and security mechanism can be applied to authenticate the devices and protect integrity and confidentiality of all communications between them. The initial pairing requires some amount of user interaction, but once it is done, the two devices can securely communicate with each other. What is then needed is a binding from the CoAP/UDP/IP level to the link layer security, so that the configuration client and server know they are communicating over a secure channel. Authentication is based on device identities.

5. IANA Considerations

TBD. It may be that some kind of registry to identify the type of applications the configuration is valid needs to be setup.

6. Acknowledgements

Salvatore Loreto, Cullen Jennings, Zach Shelby and Zhen Cao have provided comments and support for this work.

7. References

7.1. Normative References

[I-D.bormann-core-links-json]
Bormann, C., "Representing CoRE Link Collections in JSON", draft-bormann-core-links-json-00 (work in progress), February 2012.

[I-D.ietf-core-coap]
Frank, B., Bormann, C., Hartke, K., and Z. Shelby, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-08 (work in progress), October 2011.

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-11 (work in progress), January 2012.

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

7.2. Informative References

[I-D.cheshire-dnsext-multicastdns]
Cheshire, S. and M. Krochmal, "Multicast DNS", draft-cheshire-dnsext-multicastdns-15 (work in progress), December 2011.

[I-D.ietf-6lowpan-btle]
Nieminen, J., Patil, B., Savolainen, T., Isomaki, M., Shelby, Z., and C. Gomez, "Transmission of IPv6 Packets over Bluetooth Low Energy", draft-ietf-6lowpan-btle-06 (work in progress), March 2012.

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

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, September 2007.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

Authors' Addresses

Johanna Nieminen (editor)
Nokia
Itaemerenkatu 11-13
FI-00180 Helsinki
Finland

Email: johanna.1.nieminen@nokia.com

Basavaraj Patil
Nokia
6021 Connection drive
Irving, TX 75039
USA

Email: basavaraj.patil@nokia.com

Teemu Savolainen
Nokia
Hermiankatu 12 D
FI-33720 Tampere
Finland

Email: teemu.savolainen@nokia.com

Markus Isomaki
Nokia
Keilalahdentie 2-4
FI-02150 Espoo
Finland

Email: markus.isomaki@nokia.com

Zach Shelby
Sensinode
Hallituskatu 13-17D
FI-90100 Oulu
Finland

Email: zach.shelby@sensinode.com

Carles Gomez
Universitat Politecnica de Catalunya/i2CAT
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu

Mehmet Ersue
Nokia Siemens Networks
St.-Martin-Strasse 53
Munich 81541
Germany

Email: mehmet.ersue@nsn.com