IoT for tiny devices: Let's talk MQTT-SN

2 September 2014 | **Software Engineering** | **Stefan Roth**, **Christian Heger**
Reading time: 7 minutes

When Size Matters

Working on systems offering memory and clock rates measured in gigs, we should not underestimate the requirements of small devices. For a device operating an industrial size conveyor belt or a cutting machine which costs 100.000 Euros or more, the overall price for the device might not matter at all. But if you want to produce the device in large numbers or operate something considerabely cheap like a coffee machine, the costs will have a large impact. But what constraints will be put on a device that targets production costs of less than 10 €?

TCP/IP as the culprit

Comparing the ethernet chip for an Arduino to its CPU, we can immediately see what it means for a small device to be TCP/IP enabled. When compared to the rest of the system, TCP and its buffering requirements demand a lot of resources.

In order to shrink the device, TCP has to be replaced by something less expensive. Replacing TCP means to replace reliability mechanisms like message deduplication and fragmentation handling. It also means to replace protocol stacks that are built on top of TCP. To keep things simple, the replacement may shift some of the mechanisms into other layers, like the application layer.

The diagram shows the relationships of some non-prorietary protocol stacks in a simplifying way. As you can see most frequently used protocols like HTTP rely on TCP. Only a few protocols like CoAP and MQTT-SN are capable of other transport layer protocol stacks. Protocols like 6LoWPAN can take over message fragmentation handling while maintaining a low profile.

When a device is supposed to communicate with internet or intranet, TCP might become indispensable. Apart from the necessary reliability mechanisms and security layers built on top of it, the web server and firewall infrastructure might require TCP and HTTP communication. One way of solving this dilemma would be to provide a gateway in between, which speaks a TCP based protocol on one side and a protocol suitable for small devices on the other. As there are only few gateways required per local device network, the hardware requirements for the gateway might be more generous.

Let’s have a closer look on the TCP-less protocols mentioned above.

CoAP

A prominent representative of the TCP-less protocols is the **Co**nstrained **A**pplication **P**rotocol (CoAP). A standardization is in progress but not finished yet (see also RFC 7252). It uses a REST-like approach offering ressources with unique URI’s for consumption by CoAP clients or the setting and update of such ressources. Currently the standardization proposal defines three binding methods:

1. Polling: This is used to request a ressource periodically
2. Observe: A client subscribes for any changes of a chosen ressource and gets notified about each change any time that the change occurred.
3. Push: Quite similar to the observe binding updates are pushed to a client, but this client is pre-configured and doesn’t need to subscribe. This is a simple message forwarding.

There are local discovery services to discover a ressouce and a proxying mechanism between CoAP and HTTP. Using CoAP means to rely on known IP addresses using point-to-point connections in the end; there is currenty no support for things like broadcast to an unknown destination or any means of skipping or bypassing IP.

MQTT-SN

Another TCP-less communication protocol is MQTT-SN. Unlike CoAP, it does not even require the IP protocol layer. As its postfix “SN” implies, The MQTT spin-off is in particular designed for constrained sensor networks. It adds some improvements to the protocol, like an error status or a more concise message header, whose absence in MQTT is criticised by Clemens Vasters. MQTT-SN allows to build up a network of constrained devices with a central broker connected to many clients. Message distribution is controlled by a message bus like pub-sub mechanism and topic registration.

The complex connection mechanism is mandatory, even if the entire network only consists of two devices trying to communicate with each other.

In order to keep the message size as small as possible, stateful endpoints are used which basically violates the idempotence principle. One example is the numeric topic id, which replaces the long topic string a MQTT client has to send with each message. As a prerequisite, the client has to negotiate the topic id with the broker. The protocol does not define how the broker or client has to react on a restart of its counterpart when all subscriptions and topic registrations are gone.

Currently, MQTT-SN is only supported by a few platforms, and there is only one free broker implementation known to the authors, the Really Small Message Broker. You might end up in the implementation of your own low-level, bit mingling platform client.

Wireless implications

Wireless protocols such as Bluetooth and ZigBee support their own reliability mechanisms which are intended to replace quality aspects which TCP provides. However those protocols are difficult to use, due to an incomplete implementation of the complex standard by different vendors and breaking changes in different versions of the protocol.
As a result, building up a large network of wireless devices from multiple vendors can be a challenging experience, or it might turn out to be just impossible. Abstracting away those implications on the network layer would improve the modularity and interchangeability of wireless components.
One problem that wireless communications have to face is the short message length. The IEEE standard for local and metropolitan area networks IEEE 802.15.4 for example requires a maximum of 127 bytes. A short message avoids network congestion, but the recomposition of message fragments is a complex process which does not fit very well into a constrained device. Avoiding message fragmentation is a strong restriction, when UTF-8 strings and challenge-response mechanisms are required (See Shafagh for more details).

Hands-on experience with MQTT-SN

During the annual coding Camp of Zühlke we experimented with MQTT-SN using a C++ implementation as a "server" sending some test data periodically to a Really Small Message Broker (see above). We tried to create a MQTT-SN-aware client using C#, which was supposed to bridge to a Web Service requiring JSON over HTTP.
Implementing the client meant to reproduce every single hop of the MQTT-SN registration and topic subscription message exchange with the broker, as defined in the protocol specification. This meant bit-shifting and mingling, converting from bytes to real world values hence and back again and dealing with big/little endian byte order.
Our goal was to write a rudimentary client able to receive messages from the broker within three days. This was not possible. One category of problems was a lack of conciseness of the protocol specification still allowing some interpretation. Another one was the broker itself which did not have any means to tell us what exactly was wrong about the message we sent to it. Further we experienced weird behaviour such as changing topic ids when talking to the broker. We ended up not being able to send a message to the correct topic id.
Summing it all up, dealing with MQTT-SN is a lot of work and it means doing it all from scratch by your own hands. At the end we had the certainty that we would have to invest much more time and effort to get this thing up and running. Doing it all yourself obviously has its price.

Conclusion

When we try to omit TCP to cut down on costs of small devices, there is currently only a small

selection of non-proprietary protocols available. MQTT-SN is a candidate, however the lack of platforms supporting the protocol and the complex pub-sub mechanism have to be considered. For simple scenarios, CoAP might be a better choice.