Making Sense of Internet of Things Protocols and Implementations

Author: Kim Rowe, RoweBots

Higher level protocols for Internet of Things have various features and offer different capabilities. Most of these protocols were developed by specific vendors, and these vendors typically promote their own protocol choices, don't clearly define their assumptions and ignore the other alternatives. For this reason, relying on vendor information to select IoT protocols is problematic and most comparisons which have been produced are insufficient to understand tradeoffs. The analysis presented here is intended to clarify the features and provide a more comprehensive system's level analysis to ensure that developers can select the best protocol for their application.

IoT protocols are often bound to a business model. Sometimes these protocols are incomplete and/or used to support existing business models and approaches. Other times, they offer a more complete solution but the resource requirements are unacceptable for smaller sensors. In addition, the key assumptions behind the use of the protocol are not clearly stated which makes comparison difficult.

Users should realize that there are fundamental assumptions generally associated with all IoT applications:

- various wireless connections will be used,
- devices will range from tiny MCUs to high performance systems with the emphasis on small MCUs,
- security is a core requirement,
- and that data will be processed in the cloud in part.

Other assumptions made by the protocol developers require deeper investigation and will strongly influence their choices. This analysis is broader and more general than others have attempted in order to include a broader set of technical concerns which arise as real systems are implemented. These concerns include separating applications and device management (privacy), remote field service with security (secure and maintainable system), application specific connectivity (protocols which match resources and security) and scalability (protocols which service big niches). The analysis is primarily focused on MCU based sensors and the cloud based support that is required for deployment. It does cover large niche sensor system solutions (ie 10M users).

A Simplified System Model

W3 developed the IoTA model <<ref>> which defines three types of terminals which vary in communications capabilities, six communications architectures, security, key generation and security scenarios, and defines and discusses a general protocol stack. While the approach is reasonably general and intended to be used for standards work, a more continuous model is required with many options supported for terminal types. Here, by building on the IoTA model, a simpler approach to using the model is developed which is consistent with the more complex view of many terminal types with small changes between them.

Three basic terminal types are defined as before but the need for a set of terminals which continuously vary in the resources available is recognized. The first type of terminal is powerful enough to provide unconstrained gateway features (TT1). You can think of this as an ARM Cortex A8 machine running embedded Linux as a small configuration. The second type is typically an ARM Cortex M3 type machine (TT2) while truly tiny devices (TT3), used in RFID tags for example, have very limited capabilities. A simple change of a TT2 machine to an ARM Cortex M4 class machine (or PIC32 or RX) will provide most of the gateway functions and support for advanced wireless, routing and security features making it a TT1 machine. These features may in

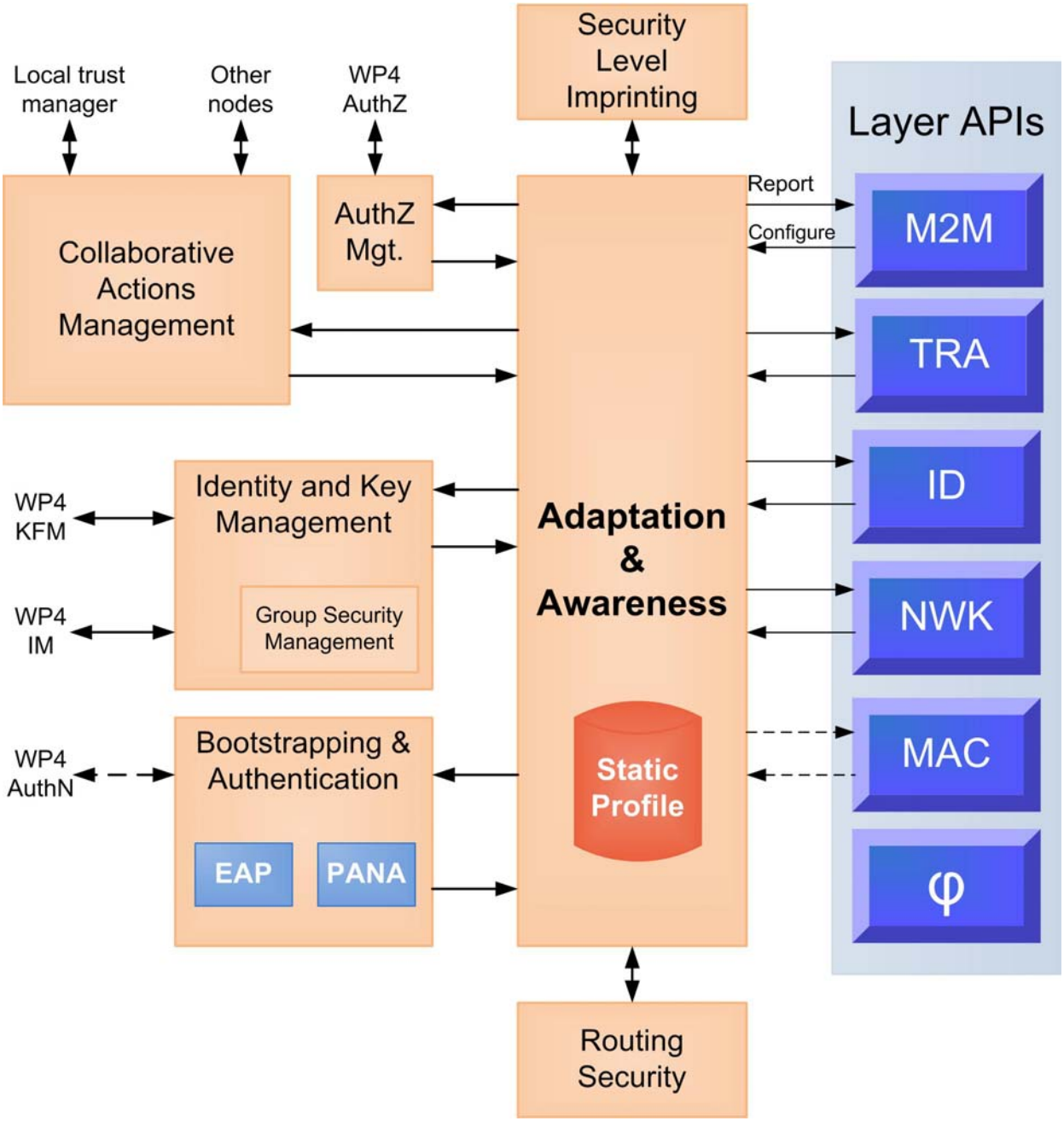
fact run on some ARM Cortex M3 devices making them TT1 devices too. Similarly, with the right software, ARM Cortex M0 processors generally though of as TT3 machines, can assume the capabilities of low end TT2 terminals. This really is a continuum of terminals not three distinct types but the idea of TT1, TT2 and TT3 is good for discussion.

The advanced model provides six communication architectures covering all cases of unicast, multicast, gateways, and constrained (TT3 and TT2) to unconstrained terminals (TT1). Since IP technology is at the heart of the system, the main contribution of this discussion is to identify the types of security features required for various architectures and security scenarios. These protocols use standard IT security solutions including point to point encryption on wireless links, TLS/SSL and IPSec or VPNs. More advanced solutions require filtering or firewall capabilities. Some protocols attempt to provide leaner security options to support lower cost MCUs; for example, CoAP redefines concentrator to cloud security with a unique security protocol.

One of the key issues for the communications architectures is the issue of wireline vs wireless network connections and where they transition. If the core assumption is that the network will be wireless throughout and that an constrained gateway (TT2) will provide a link to the cloud (CoAP) then an evaluation of the protocol based on this architecture will be good. If the problem is such that the gateway needs other features in addition to communication and security or standard security protocols are required then a small unconstrained node (TT1) may be used to connect from the TT1 node to the cloud. For example, a Zigbee based set of sensors could connect to the gateway (using only wireless link security) which acts as a Zigbee concentrator. This concentrator can then provides a wireline link to the cloud using TLS – this offers a standard secure means to provide secure sensor data without the more limited CoAP approach. Each architecture must be evaluated separately – one protocol does not fit all cases.

A second key issue is the performance and delivery of data. Some protocols use UDP to improve performance but by doing so, they eliminate key features which are required for some applications. Some protocols then try and add these features back in. While this might work in some instances, it seems poorly thought out. Stream sockets suffer a performance hit but they do guarantee in order delivery of all data. The performance hit on sending sensor data on an STM32F4 at 167MHz is less than 16.7% (measured with 2KB packets, smaller packets reduce the performance hit). By taking the approach of stream sockets, standard security protocols can also be used which simplifies the environment (although DTLS could be used with UDP if available). The point is again that each architecture must be evaluated separately including performance measurements – one protocol does not fit all cases even if it makes the claim for architectures which resemble yours.

The general protocol stack from the advanced IoTA model covers simple communication and security features. Figure 1 illustrates the protocol layers and functions. From the diagram, the ID layer is an addition to standard protocol stacks. IP addresses, since the advent of IP as the global addressing technology in the Internet, have been used as both “locators” (where a given entity can be found) and “identifiers” (the name of the resource, which somewhat includes its properties). Recently, there has been an increasing interest in the IoT community in splitting these two roles, by using IP addresses as locators and adding a further protocol layer, referred to as ID layer, to deal with the resource identification task. This ID layer is optional. Note that there is not a separate provision for management and applications, clear definition of support for remote field service, application of scalability models or other big niche features. This will be discussed below.



IoT or M2M Protocols

There is a broad set of protocols which are promoted as the silver bullet of IoT communication for the higher level M2M protocol in the protocol stack. Note that these IoT or M2M protocols focus on the application data transfer and processing. The following list summarizes the protocols generally considered.

- CoAP
- Continua – Home Health Devices
- DDS
- DPWS: WS-Discovery, SOAP, WSAddressing, WDSL, & XML Schema
- HTTP/REST
- MQTT
- UPnP
- XMPP
- Yaml4 – research only today

- ZeroMQ

These protocols have their features summarized in the following table. Several key factors related to infrastructure and deployment are considered separately below.

Protocol	*Transport*	*Messaging*	*2G,3G,4G (1000's)*	*LowPower and Lossy (1000's)*	*Compute Resources*	*Security*	*Success Stories*	*Arch*
CoAP	UDP	Rqst/Rspnse	Excellent	Excellent	10Ks/RAM Flash	Medium - Optional	Utility field area ntwks	Tree
Continua HDP	UDP	Pub/Subsrb Rqst/Rspnse	Fair	Fair	10Ks/RAM Flash	None	Medical	Star
DDS	UDP	Pub/Subsrb Rqst/Rspnse	Fair	Poor	100Ks/RAM Flash +++	High-Optional	Military	Bus
DPWS	TCP		Good	Fair	100Ks/RAM Flash ++	High-Optional	Web Servers	Client Server
HTTP/ REST	TCP	Rqst/Rspnse	Excellent	Fair	10Ks/RAM Flash	Low-Optional	Smart Energy Phase 2	Client Server
MQTT	TCP	Pub/Subsrb Rqst/Rspnse	Excellent	Good	10Ks/RAM Flash	Medium - Optional	IoT Msging	Tree
SNMP	UDP	Rqst/Response	Excellent	Fair	10Ks/RAM Flash	High-Optional	Network Monitoring	Client-Server
UPnP		Pub/Subscrb Rqst/Rspnse	Excellent	Good	10Ks/RAM Flash	None	Consumer	P2P Client Server
XMPP	TCP	Pub/Subsrb Rqst/Rspnse	Excellent	Fair	10Ks/RAM Flash	High-Manditory	Rmt Mgmt White Gds	Client Server
ZeroMQ	UDP	Pub/Subscrb Rqst/Rspnse	Fair	Fair	10Ks/RAM Flash	High-Optional	CERN	P2P

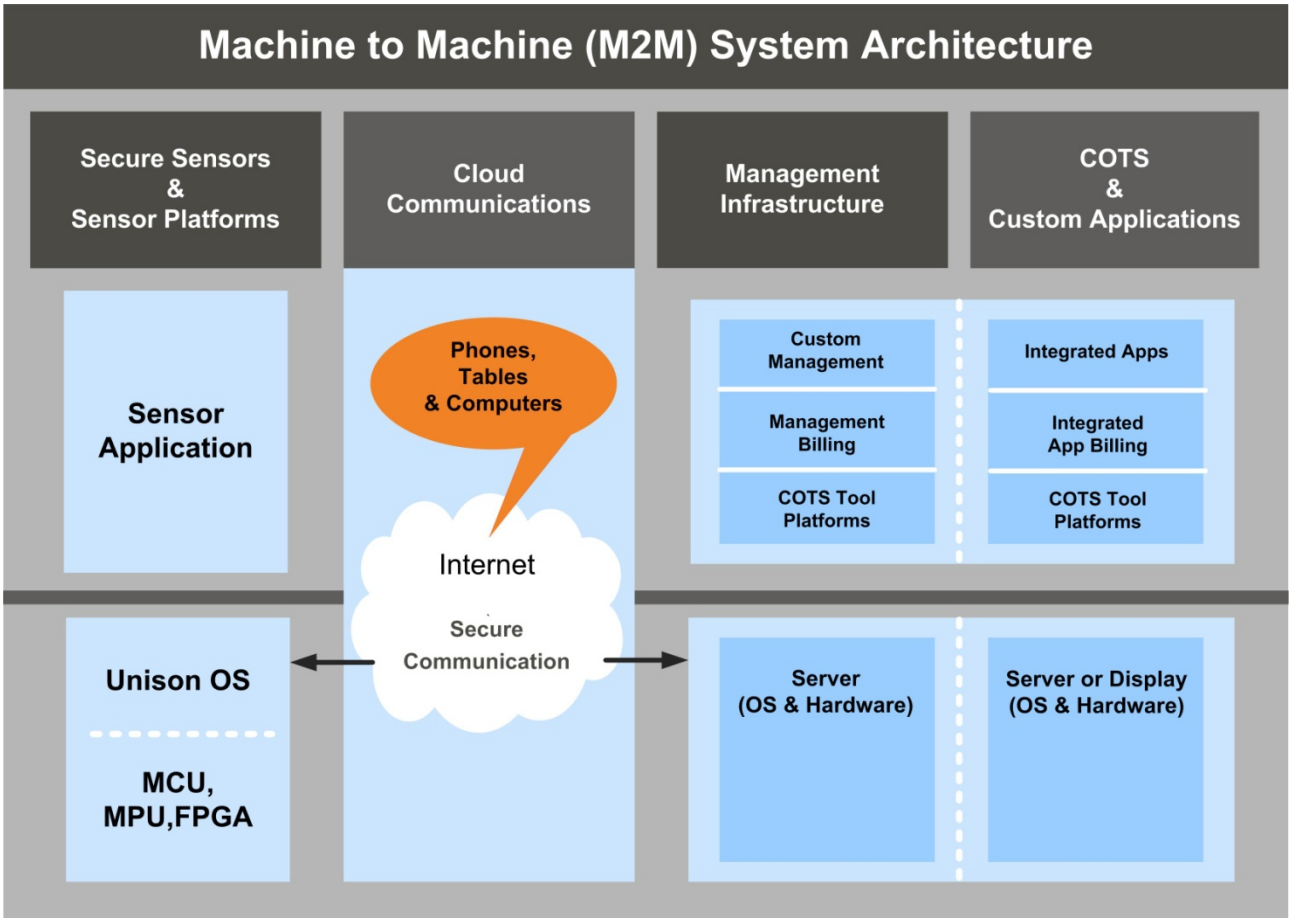
IoT Deployment Architecture

When an IoT system is deployed, it has more than the ability to collect data and share it to the cloud or setup parameters linked to the operation of the sensors. Consider the following factors:

- How will IP addresses be configured for all the devices?
- How will this process of IP addresses scale?
- Is the system secure?
- Will it meet all privacy laws?
- Can I manage devices remotely?
    - For billing purposes?
    - For maintenance purposes?

If this broader set of questions is considered, an architecture which includes the management system as well as one which includes the application architecture must be included. Although all components of this architecture are not required for all systems most certainly some components will be required for every system. Simply eliminate what you don't require and consider this as a general architecture for your application.

Any M2M or IoT protocols can be supported with ease if a POSIX/Linux API is available. The Unison OS is being fitted with all available IoT protocols as off the shelf options.



In the system architecture diagram we show the two separate components inside the cloud required for system management and application processing to satisfy privacy laws. Both components may have separate billing options and can run in separate environments. The management station may also include:

- system initialization
- remote field service options (ie field upgrades, reset to default parameters, remote test, ...)
- control for billing purposes (account disable, account enable, billing features, ...)
- control for theft purposes (the equivalent of bricking the device)

Given this type of architecture, there is additional protocols and programs which should be considered:

- Custom developed management applications on cloud systems.
- SNMP management for collections of sensor nodes.
- Billing integration programs in the cloud.
- SQLite support for gateway databases running Unison OS on the target environment.

In addition, the core application support is required and complete system security is required.

Management Features

Some of the M2M or IoT or IoE protocols deliver on the pre-configuration of the system while others do not. Provision must be made for system configuration either via an automatic setup sequence which runs when the device is first initialized, or via an application which connects to the uninitialized wireless device and provides information to configure it. Examples of this are UPnP address generation, CoAP Ipv6 address formation using the Ethernet MAC, Auto-IP use and

Zeroconf use. In some cases this is sufficient for a broad area network but in most cases this is more suited to local use and other approaches will be required.

SNMP is an ideal management solution proven for many years. It comes complete with full security and remote control features suitable for an unlimited number of nodes. While it is a proven option for managing medium and larger size nodes, its footprint (15K of Flash and 2K of RAM) becomes unsuitable for very small nodes (ie ARM Cortex M0 with 32K Flash). SNMP is not mentioned as an M2M protocol by others; however it is often the best protocol for system management when coupled with remote field service.

Remote field service options which support reflashing the devices with encrypted images and supporting an automatic fall back approach is essential to improve customer satisfaction. Off the shelf solutions eliminate extensive development to support these features and enhance customer retention and in use experience. It also provides a means to easily solve customer problems in the field and reduce support costs.

Although billing support is not necessarily required for all devices, it is for some devices. In fact, developers should think in terms of a recurring revenue model for successful business development. The ability to remotely control the device can be helpful for both billing control and theft management.

Applications Features

As discussed, SNMP is often the best management protocol. Custom management systems can be built with any of the protocols in table 1, and should be strongly considered if SNMP is too large. Of course, to make this choice, the IoT or M2M protocol must also have an extremely small footprint on the target device to make this a viable option.

SQLite is supported on medium and larger nodes which is critical for very intelligent nodes supporting discontinuous operation. In this case, users should think of storing information in the cloud and having interim storage in the device. With advanced artificial intelligence algorithms in the device, and/or the cloud superior user experiences can be created. Low cost availability of SQLite is dependent on a POSIX/Linux API.

Any M2M or IoT protocols can be supported with ease if a POSIX/Linux API is available. Linux or other MMU based RTOS products require significant resources offering TT1 support. Unison OS offers these capabilities with options for TT1 and/or TT2 implementations as well as enhanced security over that offered by Linux or other larger RTOS solutions.

System Security

Standard IT security solutions are the core security mechanisms for most of these protocols which offer security. These security approaches are based on:

- TLS
- IPSec / VPN
- SSH
- SFTP
- Secure bootloader and automatic fallback
- Filtering
- HTTPS
- SNMP v3
- Encryption and decryption
- DTLS (for UDP only security)

As systems will be fielded for many years, design with security as part of the package is essential.

Summary of Additional Features

Protocol	*Pre-configuration*	*Integrated Security*	*Discontinuous Big Data*	*Separate Management*	*Separate Application*	*Remote Field Service*
CoAP	N	Medium - Optional	N	N	N	N
Continua HDP	Y	None	N	N	N	N
DDS	N	High – Optional	Y	N	N	N
DPWS	Y	High – Optional	Y	N	N	N
HTTP/ REST	N	Low – Optional	N	N	N	N
MQTT	N	Medium - Optional	N	N	N	N
SNMP	Y	High – Optional	N	Y	Possible	N
UPnP	Y	None	N	N	N	N
XMPP	Y	High - Manditory	N	N	N	N
ZeroMQ	N	High – Optional	N	N	N	N

From table 2, none of the proposed protocols have dealt with all the issues. It is expected that the various development groups will evolve these protocols by adding support for other protocols to address these shortcomings or rely on associated supporting systems to address these issues. The combination of various protocols and systems can provide solutions for specific cases.

Unison RTOS Microcontroller Example

By combining the following off the shelf components a TT2 level system with management and application capabilities could be constructed.

Hardware

The small nodes will be TT2 level systems and be implemented with an ARM Cortex M3 with an 802.15.4 radio while the gateway devices will be implemented with a PIC32MZ with an 802.15.4 radio and a wireline interface for TCP communications.

Software – TT2 Node

- Core Unison OS
- SNMP with SNMPv3
- Unison Bootloader and Remote Field Service
- TCP with 6loWPAN (and secure radio link)
- Basic filtering and firewall issues to exclude proximity attacks

Software – TT1 Node

- Core Unison OS
- SNMP with SNMPv3
- Unison Bootloader and Remote Field Service
- TCP with 6loWPAN (and secure radio link) with TLS for host communications
- SFTP for host to gateway file transfer
- Filtering or firewall capabilities (restricting communications for wireless nodes to the gateway and excluding external unwanted communications)

This simple example illustrates how various systems could be constructed taking protocols from elsewhere and standard modules to construct comprehensive secure systems. Other examples are also quite evident:

- Replace 802.15.4 and 6loWPAN with Bluetooth support
- Replace 802.15.4 and 6loWPAN with WiFi support
- Replace SNMP with a proprietary protocol for management
- Add MQTT support to communicate from the TT2 nodes to the TT1 node
- Add CoAP and use the CoAP protocol security to communicate to the cloud eliminating.
- Add Https/REST to the gateway for cloud communication

Many other options exist. The Unison OS offers all the various components either as off the shelf options or as factory special implementations.

Data Serialization

One set of features has been obscured during this discussion and at times is critical to the system performance. This is the issue of data representation and serialization. Many of the options presented include serialization protocols. A list of these data representations include:

- XML (with SOAP)
- EXI – a binary version of XML representations
- JSON
- BSON – Binary JSON
- ASN.1
- XDR
- YAML

These protocols were originally text based and then migrated to binary for efficiency. In the above protocols where data transfer over wireless links with limited bandwidth is required, many of these protocols added binary serialization of data and data descriptions. Examples of this are in pairs:

- XML and EXI
- JSON and BSON

ASN.1 is the original binary version protocol uses for this type of data representation while XDR is an early protocol which supports binary transmission. Yaml is a serialization protocol which has evolved into a more complete IoT protocol for research purposes called Yaml4 by the researcher involved.

Summary

A broad set of M2M or Internet of Things (IoT) protocols are available but often the correct protocol choices are obscured by vendors with vested interests in their offering. Users must understand their specific requirements and limitations and have a precise system specification to make sure that the correct set of protocols is chosen for the various management, application and communications features.

Simple examples for the Unison OS were provided to illustrate how easy it is to assemble IoT systems using a variety of different protocols. Any IoT protocol which makes sense for your application should be an option as these protocols will evolve and standardize over time. The Unison OS provides the capabilities that are required to achieve these implementations.