Designing IoT: Part 1 - IoT Devices and Local Networks

Thinking About the Internet of Things (IoT) What does the phrase "Internet of Things" mean? It depends a lot on where you stand in the supply chain. Many have tried to define it, and the definitions are often coloured by the needs of their own industries and their own agendas. But, as a hardware or software engineer, you already understand the essential element: to build products that are interconnected. And, embedded systems will play – and are playing – a crucial role in the development of the IoT.

From what I see today, there are four main components of systems that support the Internet of Things, as depicted in Figure 1.

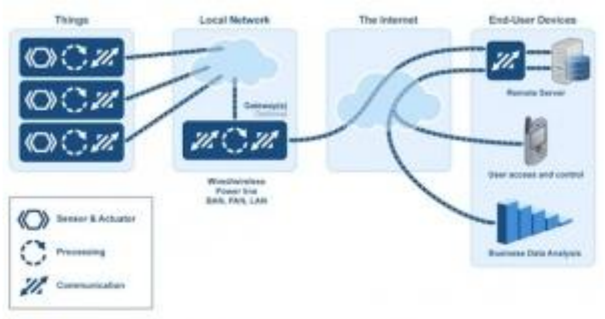


Figure 1 - IoT from an embedded point of view

As you see, there are four separate areas:
- The Thing itself (the device)
- The Local Network, which moves data in and out of the device. This may include a gateway to translate proprietary communication protocols to the IP family of protocols.
- The Internet itself
- End user devices (desktop, laptop, smartphones) or enterprise data systems that receive and manipulate data (backend data analytics)

IoT is not complicated in conception, but it is complex in its execution. Each individual component is simple, but there are many required components for IoT. What is important to understand is that even if new hardware and software are still under development, we already have all the tools we need now to start making IoT a reality.

This article will look at the end devices in an IoT system from the embedded developer point of view. This article has four parts, divided along the same grouping found in Figure 1. I would like to start with the "Thing". But to understand well the "Thing" challenges, we need to understand the communication technologies in play with the IoT devices.

What are the Local Networks?

As mentioned above, the value in IoT is in interconnected devices, and the data and metadata they will generate. The choice of communication technology affects the amount of software required, which in turn affects hardware requirements and cost. Furthermore, IoT devices are deployed in such a huge number of different ways—in clothing, houses, buildings, campuses, factories, and even in your body—that no single networking technology can fit all bills.

Wireless Sensor Networks (WSN)

For an installation of IoT devices in a specific location (say, a factory), a large number of sensors and actuators may be scattered over a wide area. A wireless technology is the best fit for such devices. There are as many types of edge/sensing node as there are system types, and some of these systems already have associated standards. This is why there are so many machine-to-machine (M2M) communication technologies in use. Some of these technologies pre-date the concepts of IoT and M2M, and they all have their particularities in terms of radio frequency, power consumption or protocol complexity. So, choosing a local networking technology can be a serious problem for embedded developers. There is often no single best choice. The technologies currently available are narrowly specialised for specific vertical markets (smart healthcare, smart grid, and so on).

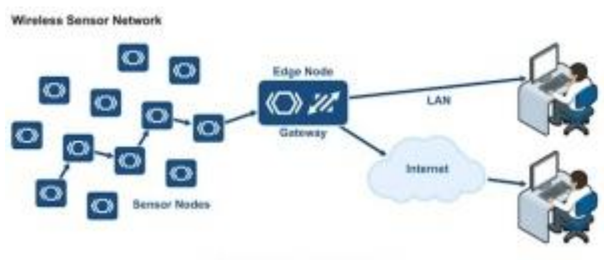


Figure 2 – Wireless Sensor Network

Figure 2 shows the position of the nodes and edge nodes in a Wireless Sensor Network. A description of these devices follows below.

WSN Node

This type of embedded system will likely represent the largest product volume. A WSN node is an embedded system performing one, or a very few, functions (reading an environmental variable like temperature or pressure, turning on a light or a motor). WSN Nodes are very low cost, so they can be deployed in very high volume. They are also very low power so that they can run on battery or even use energy harvesting.

Energy harvesting derives energy from external sources (e.g. solar power, thermal energy, wind energy, electromagnetic radiation, kinetic energy and more), captures and stores it for small, low power wireless autonomous devices, like the nodes on a WSN.

WSN Edge Node

A WSN edge node is a WSN node with IP connectivity. It acts as a gateway between the WSN and the IP network. It can also perform additional local processing, local storage, and can have a user interface.

WSN Technologies

The battle for the position of preferred networking protocol is far from over. There are multiple candidates. Initially, there were proprietary mesh networking stacks such as EmberZNet and TinyOS from Berkeley. But market growth required the use of standards-based solutions to allow for interoperability, and multiple sourcing for suppliers. The application protocols are the last element developed for mesh networking and wireless sensor networks (WSN). These efforts lead to emerging standards such as Zigbee, Z-Wave and Ant.

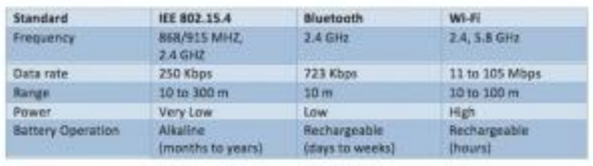
The first networking technology candidate for an IoT device is Wi-Fi, because it is so ubiquitous. Certainly, Wi-Fi can be a good solution for many applications. Almost every house that has Internet connectivity has a WiFi router. Any other device in the house can use this Wi-Fi access point, and it is IP capable. A good example is the thermostat by NEST Labs, (now part of Google) or garage door openers by Craftsman, Inc.
However, Wi-Fi needs a fair amount of power. In the "Thing" category, there are myriad devices that can't afford that level of power—battery operated devices, for example, or sensors positioned in locations that are difficult to power from the grid.

Newer networking technologies for wireless sensor networks are pushing the development of low-cost, low-power solutions. These technologies support the creation of very large networks of very small intelligent devices for sensing and collecting data. Currently, major R&D efforts are concentrated on:
- Low-power and efficient radios, allowing several years of battery life
- Energy harvesting as a power source for IoT devices, especially considering the new low power radios
- New reliable mesh networking and protocols for unattended long-term operation without human intervention (e.g., M2M networks)
- New standard application protocols and data formats, enabling autonomous operation

One of the major IoT enablers is the IEEE 802.15.4 radio standard, which was released in 2003. Commercial radios meeting this standard provide the basis for low-power systems. This IEEE standard was extended and improved in 2006 and in 2011 with the 15.4e and 15.4g amendments. Power consumption of commercial RF devices is now cut in half compared to only a few years ago, and we are expecting another 50% reduction with the next generation of devices.

EnOcean, for example, is one of the companies that have patented energy-harvesting wireless technology to meet the power consumption challenge. EnOcean’s technology works in the frequencies of 868 MHz for Europe and 315 MHz for North America. The transmit range is up to 30 metres in buildings and up to 300 metres outdoors.

For a device to take advantage of energy-harvesting technology, the processor and the software running on the device must be able to perform their tasks in the shortest time possible, which means that the transmitted messages must be as short as possible. This requirement has implications for protocol design. And it is one of the reasons why 6LoWPAN (short for IPv6 over Low power Wireless Personal Area Networks) has been adopted by ARM (Sensinode acquisition) and Cisco (ArchRock acquisition), as its encapsulation and header compression mechanisms allow for briefer transmission times.



Standard	IEE 802.15.4	Bluetooth	Wi-Fi
Frequency	868/915 MHZ, 2.4 GHZ	2.4 GHz	2.4, 5.8 GHz
Data rate	250 Kbps	723 Kbps	11 to 105 Mbps
Range	10 to 300 m	10 m	10 to 100 m
Power	Very Low	Low	High
Battery Operation	Alkaline (months to years)	Rechargeable (days to weeks)	Rechargeable (hours)

Table 1- Wireless radio technologies

There are many wireless networks available that are specialised for various industries. The following is a brief list:
- 6LoWPAN
- ZigBee and ZigBee IP
- ANT
- Z-Wave
- Bluetooth
- Wireless HART
- Wireless M-Bus
- ISA100
- DASH7

The connectivity requirements for IoT devices are so diverse that a wide range of different technologies are needed. One technology cannot meet all the range, power, size and cost requirements. It is my belief, however, that any protocol that can carry IP packets has an advantage over all others. These technologies will, in my humble opinion, be the winners. I believe that 6LoWPAN will be the choice for WSNs and light IP based protocols in the Internet back-end services.

IPv6—a major IoT enabler

If the system envisioned is local and M2M only, the wireless protocols discussed above are all good candidates. But if the goal is to remotely control a device over the Internet, or take the device's sensor data and store it on a remote server, the Internet Protocol (IP) must be involved. If the M2M network does not handle IP natively (or uses IP in a modified form), this can still be done using an intermediate gateway that translates the local protocol into an Internet-capable one.

Kaivan Karimi from Freescale, and Gary Atkinson from ARM, in a white paper titled, What the Internet of Things (IoT) Needs to Become a Reality, say that they believe that some communications technologies, such as Bluetooth Low Energy (BLE), are becoming de facto standards in certain vertical segments, such as the health care industry. But in the fields of industrial control and automation, the battle for the preferred standard – between ZigBee, low-power Wi-Fi technologies, and 6LowPAN – has just begun.

If at all possible, it is crucial that local area networks (LANs), personal area networks (PANs) or body area networks (BANs) all make use of the suite of Internet Protocols (IP, UDP, TCP, SSL, HTTP, and so on).

The usefulness of IoT devices resides not only in local communication, but also in global communication. This means using the Internet, but not necessarily the Internet as we understand it today, with HTTP and browsers. Other protocols are being developed to better address the IoT needs. Furthermore, these networks must support IP version 6, as the current IP version 4 standard faces a global addressing limitation, as well as limited multicast support and poor global mobility.

IPv6's addressing scheme provides more addresses than there are grains of sand on earth — some have calculated that it could be as high as 10^{30} addresses per person (Compare that number to the fact that there are 10^{28} atoms in a human body!) IPv6 was designed to never run out of IP addresses. With IPv6, it is much simpler for an IoT device to obtain a global IP address, which enables efficient peer-to-peer communication.

The importance of IP to the Internet of Things does not automatically mean that non-IP networks are useless. It just means that where such networking technologies are used, a gateway is required to reach the Internet. The gateway does the translation between the IP realm and the specific networking technology used by the IoT device. An example is the ZigBee IP gateway; ZigBee is a communications protocol for low-power devices, and cannot carry IP packets. However, in March 2013, the situation as changed when the ZigBee Alliance released ZigBee IP, which is an open standard for transmitting IPv6 over low power wireless personal area networks (6LoWPAN).

Referring to Figure 1, IoT from an embedded point of view, we can see that the network that links devices together is only one part of the IoT. 6LowPAN, because it carries an IPv6 address with a compressed header, can at least offer the possibility to connect to the Internet. 6LoWPAN has also an advantage over other personal area networks, since peer-to-peer communication is simpler when each device has a global address. However, even though 6LoWPAN carries a special form of IPv6 packet, another software layer is still required to make the translation between the compressed IPv6 header and the regular IPv4 or IPv6 headers that are understood by your Internet service provider. This is where gateways or routers come into play.

Conclusion

The requirements for IoT device communication dictate the requirements for the device's software. Any one of the above-mentioned communication stacks, together with the application implemented in the device, represents a significant amount of software. The IoT device hardware design is greatly impacted by these choices.

The industry would prefer to build WSN nodes at the lowest possible cost, and to that end, a lot of work has been done to integrate the microcontroller and its peripherals onto a single chip. The software requirements put heavy demands on the microcontroller's flash memory and RAM. For the next few years, the cost of WSN nodes may still be slightly too high to allow for wide market penetration of these devices.

Improvements in chip design, including the move to lower transistor geometry, in conjunction with really small core design, are all factors that will eventually make low-cost IoT devices a reality.

I strongly believe that the Internet Protocol is mandatory for any IoT device. This is why I think that 6LoWPAN is the best WSN technology to use.

In the next segment, I will discuss IoT device hardware and software architectures. I will also provide suggestions for the processor architectures for each type of IoT devices.

Of course, any discussion of IoT would not be complete without looking into the Internet and Cloud services. Once we have an IoT device transmitting or receiving data encapsulated in an IP packet, we now have to see how this data can be stored, analysed and used to create more value. The last two segments in this series will touch these topics.

Useful resources: Postscapes

Designing IoT - Part II - the Thing

In the first article in this series, I presented the communication technologies used in Wireless Sensor Network (WSN) nodes. WSN nodes are not the only types of IoT device, but to provide the hardware and software architectures for each type of IoT node, I needed to establish the kind of software load that is typical on such equipment.

The IoT Devices

In this segment, I will describe what an IoT device actually is – the "Thing" in the Internet of Things. How is it structured, what processor should be used for each case, and which software languages should be used for the framework and the application?

What is a Thing?

The definition of a "Thing" in the Internet of Things varies a lot. I would define a **Thing** as *an embedded computing device (or embedded system) that transmits and receives information over a network for the purpose of controlling another device or interacting with a user*. A Thing is also a microcontroller- or microprocessor-based device. The capabilities of these embedded devices have been expanding at the speed of Moore's law.

What is an Embedded System?

When I am asked what an embedded system is, I like to use the following definition: "An embedded system is a microcontroller/microprocessor-based system encapsulating one single-purpose process for the lifetime of this system."

As the CTO of Micrium, I consider an embedded system to be a system based on a microcontroller (MCU). Some Linux and Android-based systems can also be defined as embedded systems, but usually, these general-purpose operating systems require an application processor, and have additional capabilities such as dynamic application loading (which does not match my definition of an embedded system). This is why MCU-based embedded systems are often described as deeply embedded systems, versus the more general definition of embedded systems. For us at Micrium, these deeply embedded systems are the Things in the Internet of Things.

MCUs featuring 32-bit architectures have dropped in price dramatically in the last several years, and are becoming increasingly common in embedded systems. The greater capabilities of 32-bit MCUs present new choices for embedded systems developers. For 8 and 16-bit MCUs, software has often been written using a foreground/background approach (that is, a super-loop). With 32-bit CPUs becoming more and more cost effective, using a real-time operating system (RTOS) for embedded devices has become the preferred option, allowing for more flexible and extensible software to run on these systems. A complete RTOS – with kernel, GUI, file system, USB Host and Device, TCP/IP (Ethernet, Wi-Fi), and more – can fit in a memory space of less than 1 MB. With an RTOS, the software architecture of an embedded system can be more flexible, and the device can perform its processing and communication tasks more easily. Troubleshooting and adding new features becomes drastically simplified. It is also simpler to perform firmware upgrades. In summary, it just makes sense to use an RTOS with a 32-bit processor.

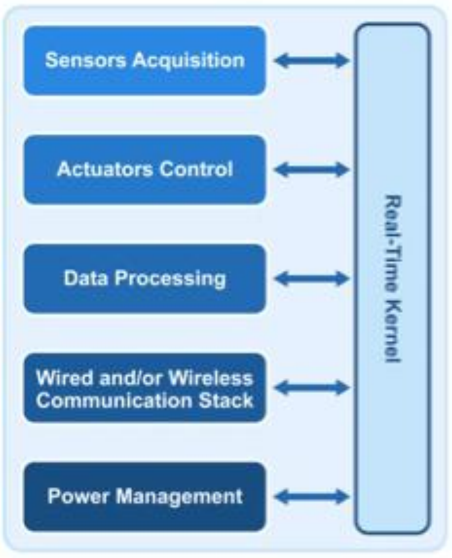


Figure 1 IoT device software architecture

Figure 1 illustrates how all activities within an IoT device can be mediated by a real-time operating system. Of course, the same can be done with Linux and Android, but those platforms require larger CPUs with many megabytes of RAM, and much greater power requirements. This is why the majority of the Things in IoT are – and will be – based on 32-bit MCUs.

Networking choices

An IoT device can be very simple and low cost (for example, the nodes in a WSN), or can be quite sophisticated (as, for example, the NEST Thermostat, which runs on a Cortex-A8 and uses Wi-Fi). In a WSN node, the networking technology used is usually a short-range access link. But there are a variety of other networking technologies used in devices where a WSN is inappropriate. These include:
Bluetooth
Zigbee, Zigbee IP, and other 802.15.4-based protocols such as 6LoWPAN
ANT, Z-Wave
Wi-Fi
DASH7
ISA100: Wireless Systems for Industrial Automation: Process Control and Related Applications
HART, WirelessHART
EnOcean
Wireless M-Bus

Ethernet, EtherCAT, EtherIP
Modbus, Profinet
HomePlug, GreenPhy, G.hnn (HomeGrid)

And more…

In a non-WSN device, the networking technology can be used to connect the device immediately to the Internet, or else to a gateway that will make the connection to the Internet. The connection to the Internet is qualified as wide area access link or cloud access link. Popular technologies for WAN access are:

Ethernet over twisted pair, over fibre
Cellular (GPRS, 3G, 4G, LTE…)
802.11ah, sub gigahertz (M2M and smart metering)
Satellite
Weightless
PowerLinc Modem, PowerLinc Controller (PLM /PLC) www.smarthome.com

And more…
Processors for the Things

In the choice of processor architecture, there is no clear winner. We are still looking at a long-term shake-out between Intel and ARM. The Intel Atom, for example, is positioned as an embedded processor, and with its greater software complexity and greater power consumption, it is geared more toward the Industrial Internet. The new Intel Quark is the latest attempt by Intel to capture a portion of the embedded system market.

ARM's family of processors includes a wide range of 32 bit architectures that are licensed to a large number of MCU suppliers. The ARM architectures are among the best low-power architectures, and processor software start-up is a lot simpler compared to Intel.

It's commonly believed that the hardware for the Things should always be as inexpensive as possible so that we can flood the planet with IoT devices. This may be true when you are talking about a simple "connected" light bulb, but low-cost devices present their own difficulties: they don't have the resources to run a full TCP/IP stack, together with a Web client or server or other IP protocols for the IoT.

Firstly, a TCP/IP stack is not a small piece of code. Of course, you can find open source TCP/IP stacks that fit within 32 kB of code space, but usually this is achieved by taking liberties with the TCP/IP standards. This can cause problems because you might need to build an IoT device that can be deployed anywhere. You will have to make sure that your device can operate on the vast majority of IP networks, private or public. Second, TCP needs a fair number of network buffers to work efficiently, which require precious RAM. Configuring a TCP/IP stack properly in terms of buffer allocation and correctly setting the variety of different timeouts is not trivial. And, if we want to use Java, then the IoT device needs some sort of OS or RTOS as the foundation to run the Java Virtual Machine (JVM). All these elements are working against the choice of very low cost processor architecture for the IoT device.

Which MCUs make good starting points when designing IoT devices? With ARM processors, the Cortex-M0 is perfect. For gateways, the ARM Cortex-M3/M4 or Cortex-A are good choices because of their greater processing capabilities. For non-ARM processors, a good option is the Renesas RL78 or RX100 for the IoT device, and the Renesas RX600 or RZ for the gateway. In both cases, ARM licensees and other silicon vendors are always introducing lower cost microcontrollers with more flash memory and more RAM to run all the Wireless Sensor Network code, plus local processing and TCP/IP stack.

A common engineering compromise is to use two processors in the sensor/actuator device. One low-power, low-cost processor (8 or 16 bit) is used for the real world interface, and a second 32-bit processor runs the local network interface. This second processor is often placed in a separate module, having already been certified for the protocol and FCC compliance (Zigbee, WiFi, 3G/4G, Bluetooth, and so on).

Until recently, we believed that the best way to save power in an embedded system was to execute as quickly as possible, and then go into sleep mode immediately. But there are processor core architectures that consume almost no power, but with greatly reduced performance, and this is a very attractive option for a WSN edge node design. This tradeoff of power for performance involves designing transistors that operate close to (or below) their threshold voltage. Three years ago, EE Times asked Mike Muller, CTO of ARM about the possibility of sub-threshold processor designs (EETimes 8/15/2013, "ARM Preps Near-Threshold Processor for IoT"). At that time, he expressed scepticism:

"If you really want to save power, the laws of physics may say it is a good idea, but you are still trading off performance against voltage; you could only get a few hundred kilohertz of clock frequency. And because there is no major demand, it is not possible to get foundries to produce qualified silicon."

This seems to have changed, because from the article mentioned above, we now know that ARM is working on a processor core optimised for operation close to the threshold voltage of CMOS transistors, and at clock frequencies of the order of tens of kilohertz. ARM's near-threshold design is compatible with the Cortex-M0 architecture, which is good news for the software community. Near-threshold is easier to do, as ARM can work with multiple foundries without having to characterise the chip process. This is not the case with sub-threshold design, which would require a custom manufacturing process, and which brings greater risk. According to Mike Muller, ARM has moved into development with partner companies and will be going to implementation. No dates have been proposed for the availability of such a part, but it will definitively be a game changer for a lot of IoT devices.

Programming Languages

In terms of programming languages for IoT applications, deeply embedded systems are programmed using C, C++ and sometimes Java, and I don't see this changing soon. For an embedded system, C is still the preferred language from my personal experience, and as per the UBM annual survey.

For IoT applications, such as medical applications, or smart grid applications, Java is an option, and many such applications have already been deployed. But Java runs on top of an operating system. So, it is not a choice between C/C++ or Java; it is C/C++ and Java.

Java is attractive for IoT devices because it brings to this industry the same potential as the apps represent for Apple's iOS devices and for Android-based devices. Oracle and ARM estimate there are about 450,000 embedded software engineers globally, and there are about 9 million Java developers. The potential for Java-based applications for IoT devices is then very significant.

The issue is; what processor do you need to run a Java-based device? The resource requirements for a Java engine are not negligible. Oracle's Java ME Embedded is designed for small devices, and Oracle estimates the following high-level system requirements:

System based on ARM architecture SOCs

Memory footprint (approximate) from 130 kB RAM/350 kB ROM (for a minimal, customised configuration) to 700 kB RAM/2000 kB ROM (for the full, standard configuration)

Very simple embedded kernel, or a more capable embedded OS/RTOS

At least one type of network connection (wired or wireless)

These numbers are not in the range of what I normally use for a deeply embedded device. The IoT industry is asking for the smallest and cheapest possible processor for the very large volume of nodes in a Wireless Sensor Network (WSN). With the above Java memory requirements, plus the embedded kernel and the communication stack, this brings the specifications into megabytes of ROM and RAM. This means that Java-based IoT devices will find their place in the WSN edge nodes, in standalone IoT devices with acceptable hardware, or in gateways.

Thing Design

With the ideal processor, the Thing architecture could be like the one in Figure 2 , if all performance and low power requirements could be met. In this case, the software architecture could be like in Figure 3 and the use of a real-time kernel is highly recommended for all the reasons exposed so far.

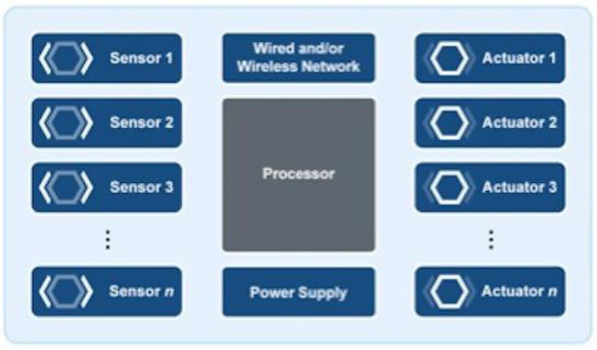


Figure 2 IoT device hardware

What we see more often today is this device architecture being split on two interconnected boards: one for the interface to the real world and one to the LAN. When two processors are used, a real-time kernel may not be required for the sensor/actuator signal processing but is strongly recommended for the communication module.

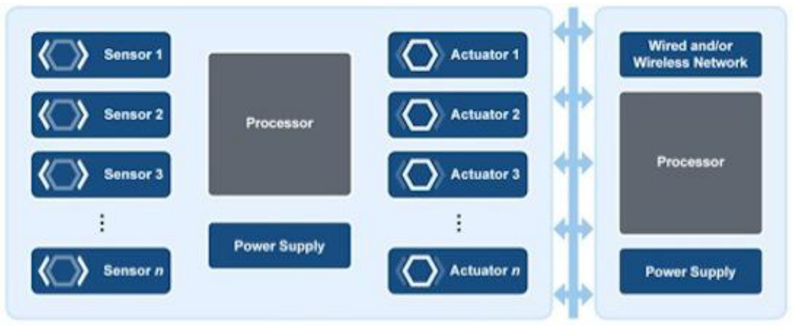


Figure 3 IoT device with two processors

Gateway Design

A gateway is a device that has a short range access link on one side, and a WAN access link on the other side. It is a bit like your home router, which has a local network access toward your home computers and other IP enabled devices, and WAN access toward your Internet service provider.

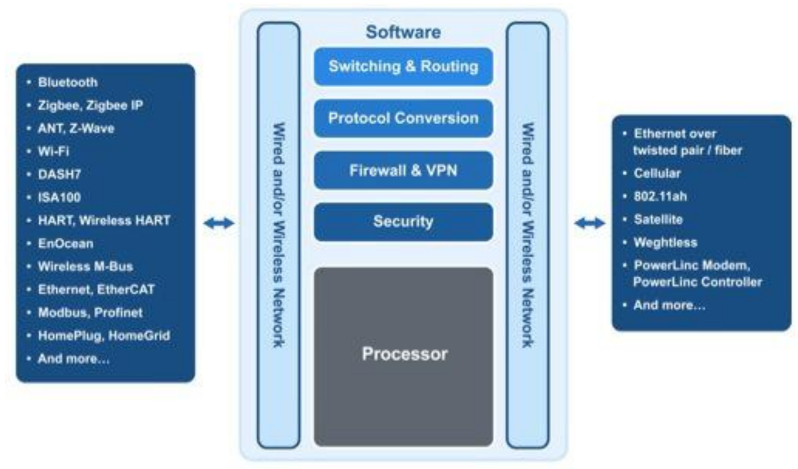


Figure 4 Gateway architecture

A gateway can be more complex than the diagram in the figure above, but the essential idea is that the gateway connects the two dissimilar networks that exist between IoT devices and Internet devices. For example, in home automation, different utilities companies may install a wide variety of IoT devices in the home, each with their own gateway. Just to name a few types of such utilities, we can think of electricity or gas, water, phone, Internet, cable/satellite, alarm system, healthcare provider (linking the medical devices in the house, handheld or not, to a central healthcare database), and so on. Some of these gateways may require additional functions, such as local storage, or a user interface (a screen and keyboard).

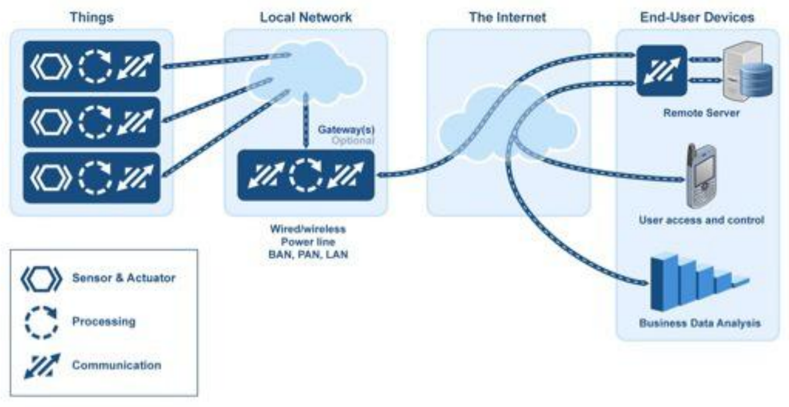


Figure 5 IoT end-to-end system

Figure 5 illustrates a typical IoT system, where the Internet and a backend service is used. From the point of view of embedded developers, I believe this diagram represents the majority of IoT implementations.

Conclusion

The "Thing" in the Internet of things is an embedded computing device. What you as the device designer need to determine is where in the system IP technology should be introduced in the design. Should it be in the end node, or in a gateway between the node and the backend service? In some cases, backend services are not even in play. For example, home entertainment systems that interact with your smartphone use Wi-Fi Direct, and so they don't even have to send or receive data via the Internet. We will look at this more closely in the next two parts.

ARM licensees are a massive presence in the IoT space. Intel is attempting once more to capture a larger part of this market. Intel's new Atom Quark is a step in the right direction. But for today, using an ARM-based processor is a safe bet.

C is also a safe choice when considering your IoT device code. But, as the applications on the devices grow and become more complex, we will need to find ways to develop and deploy these applications more rapidly. This is why the industry believes Java should be used. We still have a memory footprint issue with Java. With the state of Java technology today, it is fine to use Java for a gateway, where the processor and the resources meet the requirements of the Java VM. More development needs to happen before Java can be used to lift the capabilities of IoT devices. Solutions for deeply embedded systems exist, but they are not coming from Oracle.

To provide a complete IoT system, we need two more components. First, we need to understand the Internet, and how to leverage it. This is what I will address in the next part. Finally, I will look at the backend systems, or as it is often described in the press: The Cloud!

Reference

Peter Clarke, August 15 2013, EETimes 8/15/2013, "ARM Preps Near-Threshold Processor for IoT"

Designing for IoT - Part III - Internet Usage and Protocols

In this segment of this 4-part article, I will look at the Internet, its usage by IoT devices, and the existing and new Internet protocols supporting the explosive growth of this new industry.

The previous two articles in this series (here, and here) described the hardware and software required to build an IoT device. These devices (the "Things" in the Internet of Things) are an essential part of an IoT system. There are two more building blocks to complete our system design.

The Internet

The Internet is the sum of all the network equipment used to route IP packets from a source to a destination. The World Wide Web, by comparison, is an application system that runs on the Internet. The Web is a tool built for people to exchange information, and in the last twenty years, we have developed and refined the Web so that ordinary, non-technical people can use the Internet easily and productively. The human interface for the Internet now includes e-mail, search engines, browsers, mobile apps, Facebook and Twitter, and other popular social media.

By comparison, in IoT, the idea is for electronic devices to exchange information over the Internet. But these devices don't yet have the machine equivalent of browsers and social media to facilitate communication. IoT is also different from the Web because of the speeds, scales, and capabilities that IoT devices require in order to work together. These requirements are far beyond what people need or use. We are at the beginning of the development of these new tools and services, and this is one of the reasons why a definition for IoT is difficult to lock down. Many visions about what it can, or could be, collide.

TCP/IP Protocol Stack

The TCP/IP protocol stack is at the heart of the Internet and the Web. It can be represented using the OSI seven-layer reference model, as illustrated in Figure 1. The top three layers are grouped together, which simplifies the model.

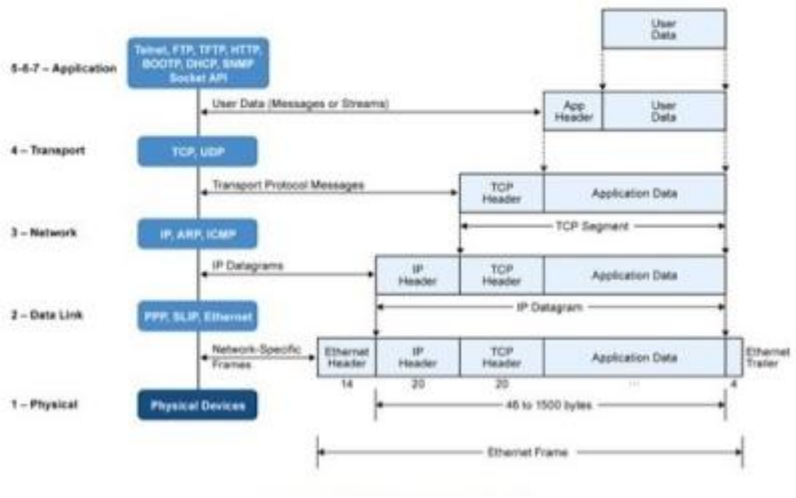


Figure 1 The TCP/IP reference model

The following is a quick description of the important layers from the perspective of embedded system integration:

Physical and Data Link Layers

The most common physical layer protocols used by embedded systems are:

- Ethernet (10, 100, 1G)
- WiFi (802.11b, g, n)
- Serial with PPP (point-to-point protocol)
- GSM 3G, LTE, 4G

Network Layer

This is where the Internet lives. The Internet — short for Inter-Network — is so named because it provides connections between networks, between the physical layers. This is where we find the ubiquitous IP address.

Transport Layer

Above IP, we have TCP and UDP, the two transport protocols. Because TCP is used for most of our human interactions with the Web (e-mail, Web browsing, etc.), it is often believed that TCP should be the only protocol used at the transport layer. TCP provides the notion of a logical connection, acknowledgement of packets transmitted, retransmission of packets lost, flow control, all of which are great things. But for an embedded system, TCP can be overkill. This is why UDP, even if it has long been relegated to network services such as DNS and DHCP, is now finding its place in the domains of sensor acquisition and remote control. If you need some type of management of your data, you can even write your own lightweight protocol on top of UDP and avoid the overhead imposed by TCP.

UDP is also better suited than TCP for real-time data applications such as voice and video. The reason is that TCP's packet acknowledgement and retransmission features are useless overhead for those applications. If a piece of data (such as a bit of spoken audio) does not arrive at its destination in time, there is no point in retransmitting the packet, as it would arrive out of sequence and would garble the message.

TCP is sometimes preferred to UDP because it provides a persistent connection. So to do the same thing with UDP, you have to implement this feature yourself in a protocol layer above UDP.

When you are deciding how to move data from the Thing's local network onto an IP network, you have several choices. Because the technologies used are familiar and available from a wide range of sources, you can link the two networks via a gateway, or you can build this functionality into the Thing itself. Many MCUs now have an Ethernet controller on chip, which makes this an easier task.

The IoT Protocols

It is certainly possible to build an IoT system with existing Web technologies, even if it is not as efficient as the newer protocols. HTTP(S) and Websockets are common standards, together with XML or JavaScript Object Notation (JSON) in the payload. When using a standard Web browser (HTTP client), JSON provides an abstraction layer for Web developers to create a stateful Web application with a persistent duplex connection to a Web server (HTTP server) by holding two HTTP connections open.

HTTP

HTTP is the foundation of the client-server model used for the Web. The safest method to implement HTTP in your IoT device is to include only a client, not a server. In other words, it is safer when the IoT device can initiate connections to a Web server, but is not able to receive connection requests. After all, we don't want to allow outside machines to have access to the local network where the IoT devices are installed.

WebSocket

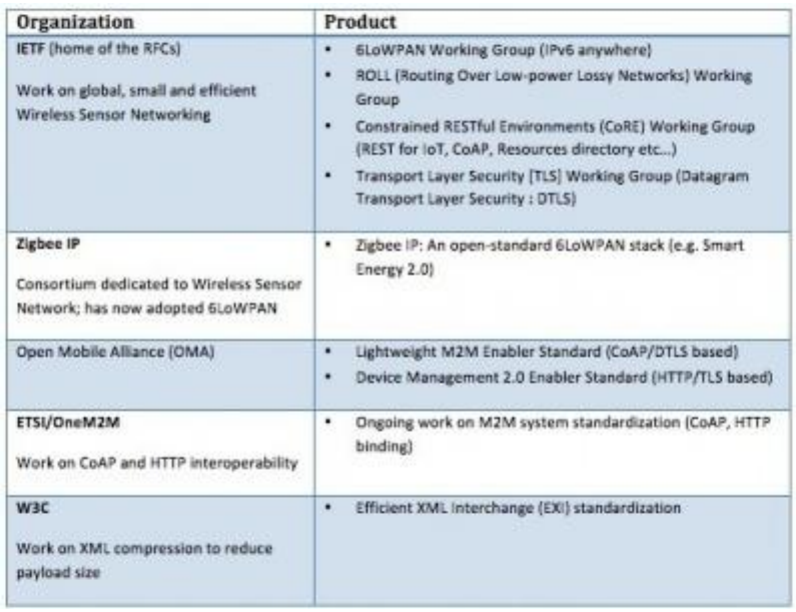
WebSocket is a protocol that provides full-duplex communication over a single TCP connection over which messages can be sent between client and server. It is part of the HTML 5 specification. The WebSocket standard simplifies much of the complexity around bi-directional Web communication and connection management.

XMPP

XMPP (Extensible Messaging and Presence Protocol) is a good example of an existing Web technology finding new use in the IoT space: See XMPP
XMPP has its roots in instant messaging and presence information, and has expanded into voice and video calls, collaboration, lightweight middleware, content syndication, and generalised routing of XML data. It is a contender for mass scale management of consumer white goods such as washing machines, dryers, refrigerators, and so on.
XMPP strengths are its addressing, security, and scalability. This makes it ideal for consumer-oriented IoT applications.
HTTP, Websocket, and XMPP are just examples of technologies being pressed into service for IoT. Other groups are also working furiously to develop solutions for the new challenges IoT is presenting us with.

Wannabe Generic Protocols

Many IoT experts refer to IoT devices as constrained systems because they believe IoT devices should be as inexpensive as possible and use the smallest MCUs available, while still running a communication stack. Currently, adapting the Internet for the IoT is one of the main priorities for many of the global standardisation bodies. Table 1 contains of short summary of the current activities.



Organization	Product
IETF (home of the RFCs) Work on global, small and efficient Wireless Sensor Networking	• 6LoWPAN Working Group (IPv6 anywhere) • ROLL (Routing Over Low-power Lossy Networks) Working Group • Constrained RESTful Environments (CoRE) Working Group (REST for IoT, CoAP, Resources directory etc…) • Transport Layer Security [TLS] Working Group (Datagram Transport Layer Security : DTLS)
Zigbee IP Consortium dedicated to Wireless Sensor Network; has now adopted 6LoWPAN	• Zigbee IP: An open-standard 6LoWPAN stack (e.g. Smart Energy 2.0)
Open Mobile Alliance (OMA)	• Lightweight M2M Enabler Standard (CoAP/DTLS based) • Device Management 2.0 Enabler Standard (HTTP/TLS based)
ETSI/OneM2M Work on CoAP and HTTP interoperability	• Ongoing work on M2M system standardization (CoAP, HTTP binding)
W3C Work on XML compression to reduce payload size	• Efficient XML Interchange (EXI) standardization

Table 1 Constrained systems standardisation efforts

If your system does not require the features of TCP, and can function with the more limited UDP capabilities, removing the TCP module entirely helps a lot to reduce the size of the total code footprint of your product. This is what 6LoWPAN (for WSN) and CoAP (light Internet protocol) bring to the IoT universe.

CoAP

Although the Web infrastructure is available and usable for IoT devices, it is too heavy for the majority of IoT applications. In July 2013, IETF released the Constrained Application Protocol (CoAP) for use with low-power and lossy (constrained) nodes and networks (LLNs). CoAP, like HTTP, is a RESTful protocol.
It is semantically aligned with HTTP and even has a one-to-one mapping to and from HTTP. Network devices are constrained by smaller microcontrollers with small quantities of flash memory and RAM, while the constraints on local networks such as 6LoWPAN are due to high packet error rates and a low throughput (tens of kilobits per second). CoAP can be a good protocol for devices operating on battery or energy harvesting.

Features of CoAP: CoAP uses UDP

- Because CoAP uses UDP, some of the TCP functionalities are replicated directly in CoAP. For example, CoAP distinguishes between confirmable (requiring an acknowledgement) and non-confirmable messages.
- Requests and responses are exchanged asynchronously over CoAP messages (unlike HTTP, where an existing TCP connection is used).
- All the headers, methods and status codes are binary encoded, which reduces the protocol overhead. However, this requires the use of a protocol analyser to troubleshoot network issues.
- Unlike HTTP, the ability to cache CoAP responses does not depend on the request method, but the Response Code.

CoAP fully addresses the needs for an extremely light protocol and with the nature of a permanent connection. It has semantic familiarity with HTTP and is RESTful (resources, resource identifiers and manipulating those resources via uniform Application Programming Interface (API)). If you have a Web background, using CoAP is relatively easy.

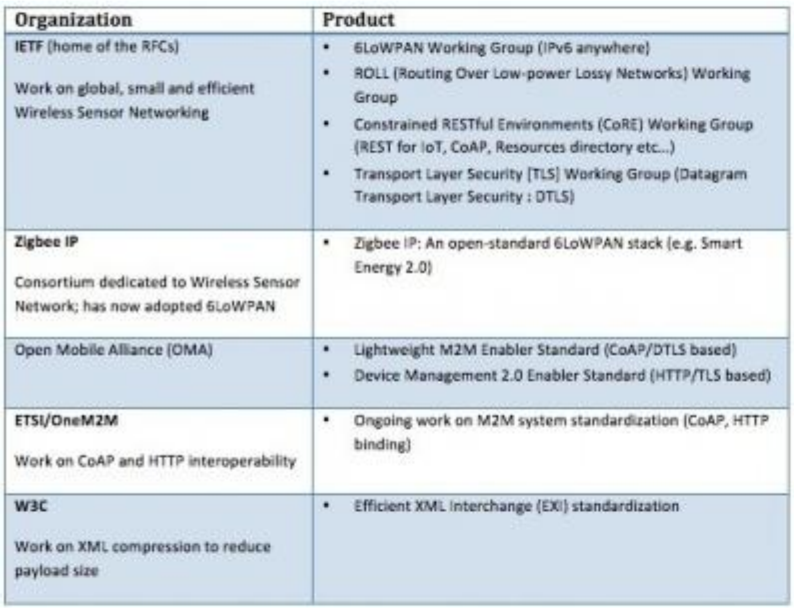
MQTT

MQ Telemetry Transport (MQTT) is an open source protocol that was developed and optimised for constrained devices and low-bandwidth, high-latency, or unreliable networks. It is a publish/subscribe messaging transport that is extremely lightweight and ideal for connecting small devices to networks with minimal bandwidth. MQTT is bandwidth-efficient, data-agnostic, and has continuous session awareness, as it uses TCP. It is intended to minimise device resource requirements while also attempting to ensure reliability and some degree of assurance of delivery with grades of service.

MQTT targets large networks of small devices that need to be monitored or controlled from a back-end server on the Internet. It is not designed for device-to-device transfer. It is also not designed to "multicast" data to many receivers. MQTT is simple, offering few control options. Applications using MQTT are generally slow in the sense that the definition of "real time" in this case is typically measured in seconds.

Comparison of potential IoT protocols

Cisco is at the heart of the Internet; their IP equipment is everywhere. Cisco is now actively participating in the evolution of IoT. It sees the potential for connecting physical objects and getting data from our environment and to process this data to improve our living standards. Table 2 is drawn from Cisco's work in IoT standards.



Organization	Product
IETF (home of the RFCs) Work on global, small and efficient Wireless Sensor Networking	• 6LoWPAN Working Group (IPv6 anywhere) • ROLL (Routing Over Low-power Lossy Networks) Working Group • Constrained RESTful Environments (CoRE) Working Group (REST for IoT, CoAP, Resources directory etc...) • Transport Layer Security (TLS) Working Group (Datagram Transport Layer Security : DTLS)
Zigbee IP Consortium dedicated to Wireless Sensor Network; has now adopted 6LoWPAN	• Zigbee IP: An open-standard 6LoWPAN stack (e.g. Smart Energy 2.0)
Open Mobile Alliance (OMA)	• Lightweight M2M Enabler Standard (CoAP/DTLS based) • Device Management 2.0 Enabler Standard (HTTP/TLS based)
ETSI/OneM2M Work on CoAP and HTTP interoperability	• Ongoing work on M2M system standardization (CoAP, HTTP binding)
W3C Work on XML compression to reduce payload size	• Efficient XML Interchange (EXI) standardization

*Table 2 Some of Cisco's work on IoT standards, see **Reference 1***

These Internet-specific IoT protocols have been developed to meet the requirements of devices with small amounts of memory, and networks with low bandwidth and high latency.

Figure 2 provides another good summary of the performance benefit that these protocols bring to IoT. The source is Zach Shelby in his presentation "Standards Drive the Internet of Things" (Reference 2).

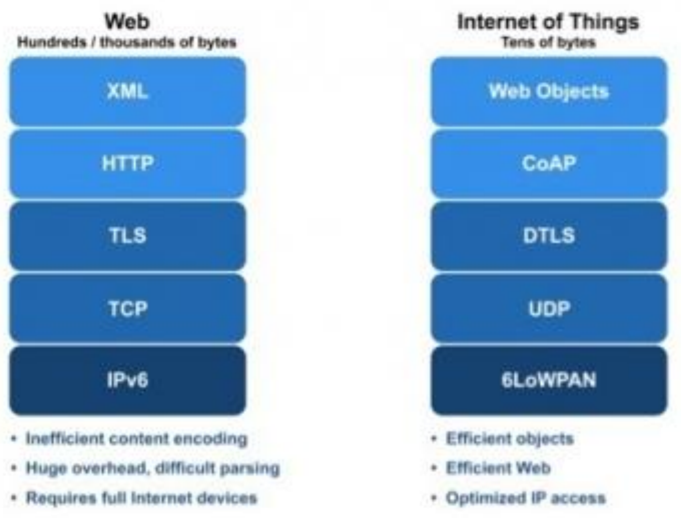


Figure 2. Comparison of Web and IoT protocol stacks

Conclusion

Connecting sensors and objects opens up an entirely new world of possible use cases—and it's precisely those use cases that will determine when to use the right protocols for the right applications.

The high-level positioning for each of these protocols is similar. With the exception of HTTP, all these protocols are positioned as real-time publish-subscribe IoT protocols with support for millions of devices. Depending on how you define "real time" (seconds, milliseconds or microseconds) and "things" (WSN node, multimedia device, personal wearable device, medical scanner, engine control, etc…) the protocol selection for your product is critical. Fundamentally, these protocols are very different.

Today, the Web runs on hundreds of protocols. The IoT will support hundreds more. What you need to do when designing your system is to define the system requirements very precisely, and chose the right protocol set to address these requirements.

The Internet Protocol (IP) is a carrier; it can encapsulate just as many protocols for the IoT as it does today for the Web. A lot of industry pundits are asking for protocol standardisation. But if there is such a large number of protocols for the Web, why wouldn't there be just as many for the IoT? You choose the protocols that meet your requirements. The only difference is that the IoT protocols are still fairly young, and must demonstrate their reliability. Remember that, when the Internet became a reality, IP version 4 was what made it possible. We are now massively deploying IP version 6, and IoT is the killer application that telecommunication carriers have been waiting for to justify the investment required.

The final article in this series will look at the role of Cloud computing in the Internet of Things.

References

[1] Paul Duffy, April 30 2013, "Beyond MQTT: A Cisco View on IoT Protocols", http://blogs.cisco.com/ioe/beyond-mqtt-a-cisco-view-on-iot-protocols/

[2]Zach Shelby, Chief Nerd, Sensinode, May 22 2013, "Standards Drive the Internet of Things", http://www.slideshare.net/zdshelby/standards-drive-the-internet-of-things

Designing for IoT—Part IV—the Cloud

In this last segment of this series, we will look at back-end services. Some IoT systems way not need back end services – for example, a smartphone controlling a TV – but the majority of the currently envisioned IoT systems reply on the collection, processing and usage of data by the IoT devices and for this, some form of Information Technology different than embedded systems is required.Placeholder IoT

This article is the conclusion to a four-part series:

- In Part 1, we reviewed the choices facing an embedded developer who needs to build wireless networking into an IoT device (that is, the Thing in the Internet of Things).
- In Part 2, we discussed the different type of IoT devices, and the design choices that you face when designing the hardware and software architectures.
- In Part 3, we looked at the Internet itself, how IoT devices make use of it.

The Cloud

The Cloud: This is another interesting buzzword. When I was Director of Engineering at Teleglobe Canada, every time we sat in a meeting room to design a network, we drew it as a cloud. A network is a cloud. The Internet is a cloud. And cloud computing is nothing more than an array of networked computers that allow you to offload processing tasks from your embedded system. The same is true for data storage: why store data locally, when you can store it in a secured data centre, with guaranteed power and back-ups?

There is, of course, one fundamental assumption in cloud computing: The network is always available!

So "cloud computing" is a term coined to put a simple name on something that has become very complex. Many companies have launched services that try their best to hide this complexity; these include Apple's iCloud, Google Cloud Platform, Microsoft SkyDrive, and others. These Cloud computing/storage systems are intended for use with desktop and mobile personal computers. Embedded developers need something similar for IoT devices.

Industry analysts are forecasting the creation of billions of IoT devices by 2020, and these devices will produce huge amounts of data. There are a few approaches for managing and processing all this data.

I see two trends developing among companies that are moving into IoT:
- Some companies are developing and selling their own proprietary solutions because they feel they have the lead, and want to leverage it to its maximum.
- Other companies don't have the capability to deploy a complete infrastructure, and so prefer to rely on emerging public or commercial solutions.

You can define your own IoT from end-to-end. Many large companies are trying to do just that, and want to capture a good portion of this emergent market. Others are specialising in certain parts, such as GE with its Predix platform for the data analytics of the industrial internet. Does it mean you absolutely need to buy the provider cloud service stuff? No, you can probably build your own or outsource the expertise you don't want to build.

Running a backend service must become a core competency for any company attempting to do so; there will be no room for dilettantes. But not all organisations have the DNA to put in place a server farm, guarantee the fail-safe operation of their network, and guarantee data back-ups, system redundancy, and all the crucial things that come with it. Can your system cope with a network failure? If so, for how long?

Early IoT deployment has revolved around sensors and actuators connected to the public Internet via a gateway or hub, and delivering data to an Internet-based server (cloud computing). This is often a vertical integration built around one primary vendor (such as your utility provider and/or telecommunication carrier). All these providers have little or no interest in working together, resulting in an unmanageable melting-pot of services.
The army of devices that compose the Internet of Things will generate more data than any individual Web application. IBM's chief executive, Virginia Rometty, in The Economist blog "The World in 2014" [Ref 2], provides IBM's estimates on the quantity of data to be processed in years to come: "By one estimate, there will be 5,200 Gigabytes of data for every human on the planet by 2020."

In his TED talk: The Internet of Things is Just Getting Started [Ref 1], Arlen Nipper calculates that to support the 30 billion connected devices expected to arrive by 2020, we would need to deploy about 340 application servers per day (120,000 servers per year), assuming that we want to deploy all these systems as segregated applications. Mr. Nipper suggests that one way to make IoT possible in the coming years will be to adopt cloud computing.

Around the year 2000, all the telecom carriers claimed that they could each provide the Internet all by themselves. They invested billions of dollars into equipment purchases. Everybody was looking for the "killer application," the application that would create the "gazillions" of bytes of traffic to fill these networks. At that time, the application that was generating most of the network traffic was e-mail. Today, social media and video sharing are replacing e-mails. When the predicted traffic did not materialise on this huge IP network, it triggered the dot-com bubble burst.

With IoT, we are finally seeing a new “killer application.” When billions of devices exchange information over the Internet, they will require significant network bandwidth, and especially enormous data storage and processing capabilities. A new term has been coined lately to represent this new trend: Big Data.

Backend Services

The business of getting data in and out of IoT devices is a gold mine. Your marketing department will pay a fortune for this data, as it will help you spot trends and help you learn how your products are being used.

If you search the Web a bit, you will find many companies offering new backend services. I see one of these new start-ups appear almost every week. The industry has to be careful not to reproduce the 2000 tech bubble. If every backend service provider believes he can build the IoT cloud by himself, then the glut of providers will almost certainly create another overcapacity problem. At the same time, just because it is fairly simple to build a server farm, it doesn’t mean that everybody should do it. Also, the big players such as Intel, Oracle, SalesForce.com and Google have already stated their intention to occupy that space.

It is said that every company will become a software company. Why so? Because the data created by a company’s products, and the systems managing that data, will soon have more value than the products themselves. So, if you are designing and manufacturing an IoT system, and require a backend service, you will need to choose whether to deploy your own, or outsource it. I believe the majority will outsource. If you are a thermostat manufacturer, you probably don’t want to develop IT competency; it will be better to leverage someone else’s.

If you do deploy your own, then you will have to choose from the technologies we reviewed in the Internet section, and then deploy servers to store and process your data, plus maintain and upgrade the systems. You will have to determine how you, and your customers, will access and manipulate this information. Will it be exclusively through Web browsers, or will you provide smartphone native applications (iOS, Android and Windows)?

Another vital feature of an IoT device is, or will be, to perform remote secured firmware upgrades. With the rapid evolution of the IoT field, it is almost impossible to plan for every contingency. This means that your hardware design must also be capable of being upgraded: your device will need enough flash memory, RAM, a bootloader, and all the other components required for your device to update firmware and applications securely.

The same questions need to be asked if you decide to outsource your backend services. But, in this case, you first have to verify that the provider can support the protocol you have selected. Otherwise, it could be your choice of backend service provider that dictates the protocol to be used in your IoT device. And you need to know if the provider can develop and support mobile data applications, or at least provide enough tools for you to do it yourself.
Similarly, are firmware upgrades included in your cloud computing outsourcing package?

Here is a short checklist to get you started. Your backend service should provide:

- Design services to help select communication hardware and software
- Support for cloud protocols (Websocket, RESTful, MQTT, CoAP, etc.)
- Support for secure remote firmware upgrades
- Web-based and mobile applications application for data viewing, processing, and remote device control
- Web and/or mobile application development (consulting services)
- Pricing...Per device/Per transaction/Per data type/Per bandwidth usage/Per storage usage; and by Monthly/annual subscription

Many of the companies offering such services can provide a sandboxed approach, where you can register a free account and test your implementation before going commercial. Some services will also offer you data from public sources to help you build a more complex application. For example, access to weather data or utility prices could be quite useful when building smart energy applications.

Most of the current offerings are based on REST or RESTful APIs, and various Java technologies. These solutions may be best suited for applications that need to move larger data payloads.

At the time of writing, there is no commercial offering for CoAP, as the standard is still in draft mode with IETF. With the ARM/Sensinode and Oracle partnership, we can expect this situation to change soon.

In addition to the list above, there are companies that offer more than cloud services. They also offer design services, or even hardware that can be used by your IoT device, providing a complete end-to-end M2M solution.

Data Analysis (Big Data)

The previous section touched on the issue of choosing a backend system to store and process the enormous quantities of data created by your own IoT devices. But the issue gets complicated quickly when you need to correlate your data with information originating in outside systems, whether the data is publicly-accessible or licensed. This is the definition of Big Data; advanced analytics that can crunch vast quantities of digital information to create economic value, and increase efficiency.

In “The World in 2014”[2] edition of The Economist, Virginia Rometty wrote about the type of systems that are required to process all this data: "Our world has become pervasively instrumented and interconnected, with computation infused into things nobody would think of as a computer. There are more than a trillion interconnected and intelligent objects and organisms—including a billion transistors for every person on the planet. About 2.7 billion of those people are now connected to the Internet, a number that is growing rapidly in every part of the world, thanks to the spread of mobile technology."

The result is a planet awash with information. It’s known as Big Data, with good reason.

A new generation of “cognitive” systems, built for big data, can keep up with the flood because they aren’t programmed, they learn—from their own experience and from our interactions with them. Cognitive systems can scrutinise structured information, such as databases, and unstructured information such as medical imaging and social media content, much faster thanks to Cloud computing.

This means we, the embedded community, may now want to learn about distributed database management systems such as Apache Cassendra, and learn Apache Hadoop to run analytics on all this data. Cassandra is an open source distributed database system that is designed for storing and managing large amounts of data across commodity servers. It is named after the prophetess of Greek mythology who had the power of prophecy and the curse of never being believed. Let’s hope we will believe the results of the analysis of our own data!

There are commercial approaches as well as open source ones. For example, GE Predix is a software platform for the Industrial Internet. Predix provides a standardised way to connect devices and data, and so enables industrial-scale analytics. The US government itself is supporting the initiative of GE and its partner in the Industrial Internet. The questions that remain to be answered, with billions of interconnected things are:
- What is the data representation model?
- What is the network discovery protocol? There are billions of devices predicted by 2020. On any type of network, a discovery protocol such as Bonjour or similar will be required.

And with all these billions of devices generating data, it makes sense to bring that data on the Internet and use what today we call Big Data to store and process all this data.

Conclusion

IoT is conceptually a simple idea. But in practice, it is made of a multitude of hardware and software layers. IoT is the sum of many elements: low-power design, sensors, wireless connectivity, gateways, servers, networks, security, cloud computing, and applications.

We have available to us an abundant supply of low power, highly integrated, memory-dense and efficient embedded processors. The IoT challenge is, therefore, not in the hardware (unless you are trying to fit TCP/IP and Java in a processor with 32k ROM and 8k RAM!). The challenge is in the networking and software.

I believe we already have all the building blocks to develop the Internet of Things, and we are held back only by our own imagination and creativity. Of course, better and cheaper technologies will appear in the years to come that will reduce development time and cost. Does that mean that the time is not yet right to start building your IoT system? Not at all! Just as in the PC market, there's always a cheaper, faster machine coming next month. Don't wait for all the lowest-cost IoT building blocks to arrive, because by the time that happens, the market space will be occupied by someone else.

Embedded systems hardware technology, software technology, infrastructure, and cloud computing are actively being tested and deployed, fostering the emerging Internet of Things. To be really successful, and realise the 30 billion devices that will be deployed by 2020, we require the same open standards, development and cooperation that supported the creation of the Internet of People (also known as the Web).

We need to separate the data from the application so that we can do new things with it, things we have not yet imagined. For the typical embedded system developer, thinking specifically about the data the system generates, manages and transmits requires a new frame of mind. There is a value in this data. In embedded system design, we understand very well how to architect the Thing, the local network and even the gateway. The processor, sensor, wireless connectivity, gateway, IP network and security are all elements that are well-known to the embedded developer.

The new challenge for the embedded community is to adapt the application to take advantage of the value inherent in our data by building on and exploiting some type of cloud computing. To monetise the embedded system's data, some type of cloud computing is absolutely required, be it private, outsourced or public. This is a new paradigm for the embedded community.

Billions of new embedded systems are predicted to appear in the next seven years. If we want to sell to this market, our new embedded systems need to be connected and inter-connected. The companies that don't get on this new wave and surf it may well disappear.

IoT for embedded systems is the new industrial revolution. The growth potential for the embedded industry is enormous. To realise this potential, the embedded industry needs to adopt the new IoT paradigm. We now understand what needs to be done. We may not yet have well-defined and established standards for each structural element of future IoT systems. But I really think this should not stop anyone from building a system that will provide value to their customers.

Embedded system developers: Catch the IoT wave. Go ahead, and innovate!

About the author

Christian Legare is Executive Vice-President with Micrium, home of uC/OS-II, the real-time kernel.
Christian has a Master’s degree in Electrical Engineering from the University of Sherbrooke, Quebec, Canada. 22 years in the telecom industry, he has been involved in large scale organisations as well as start-ups, mainly in Engineering and R&D, and was in charge of an IP (Internet Protocol) certification program at the International Institute of Telecom (IIT) in Montreal, Canada as their IP systems expert.

References

[1] Arlen Nipper at TEDxNewBedford : The Internet of Things is Just Getting Started Published on Oct 4, 2012

[2] Virginia Rometty, chief executive of IBM, The year of the smarter enterprise, Nov 18th 2013 | From “The World In 2014” print edition, http://www.economist.com/node/21589108