Internet of Things (IoT): Architecture and design

Conference Paper · May 2016

DOI: 10.1109/AIC-MITCSA.2016.7759958

1 author:

Ali Abed
University of Basrah
39 PUBLICATIONS **203** CITATIONS

SEE PROFILE

Internet of Things (IoT): Architecture and Design

Ali A. Abed

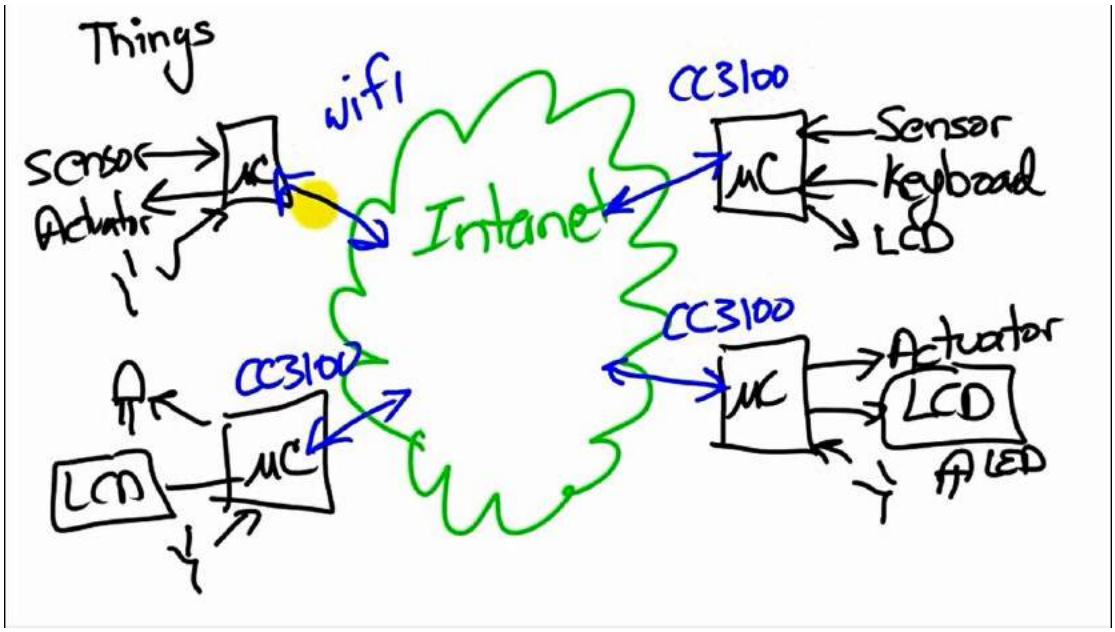
Computer Engineering Department/University of Basra

1. Devices and Local Networks

Many people have tried to define the Internet of Things. Here are some definitions:

Making some of the things that we are using has the ability of Internet access (smart things).

Anything that can be joined to it processing unit (microcontroller) and connected to the Internet is considered a thing in the world of IoT.



CC3100 is Wi-Fi and Internet-of-Things solution for MCU Applications. The CC3100 device is the industry's first Wi-Fi CERTIFIED chip used in the wireless networking solution. It is pin compatible with TM4C123GXL ARM Launchpad.

Microcontrollers can be Arduino, TM4C123G ARM processor, etc.

In IoT, we think how to build interconnected products.

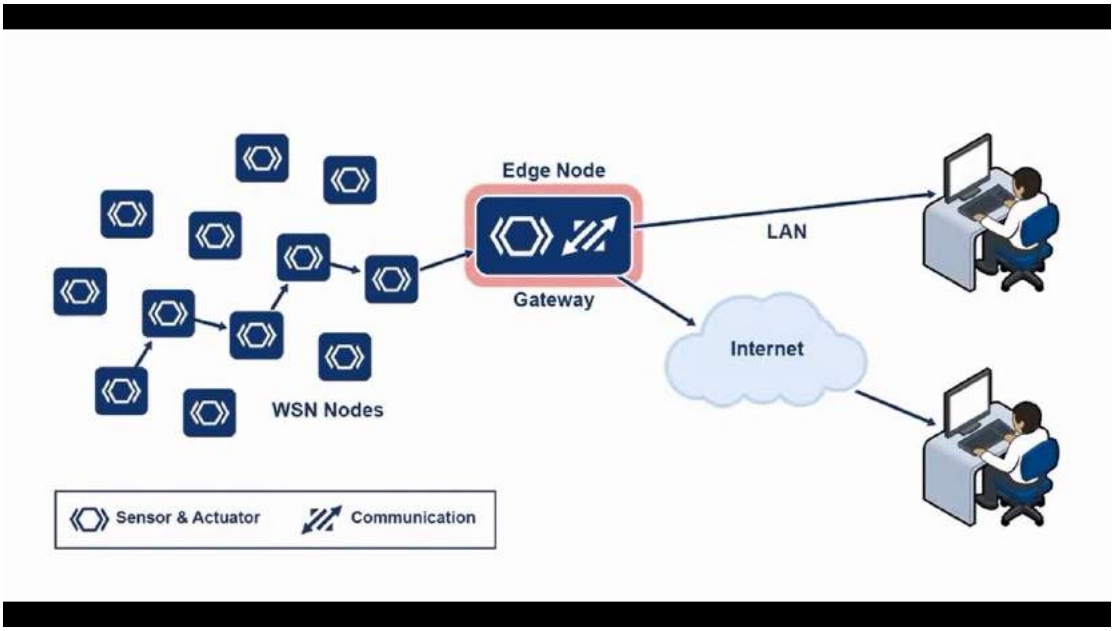
IoT hardware should always be low-cost, so that we can flood the planet with IoT devices.

IoT systems are not complicated, but designing and building them can be a complex task. We already have all the tools we need today to start making the IoT a reality.

The first step in building an IoT device is to figure out how it will communicate with the rest of the world. Your choice of communication technology directly affects your device's hardware requirements and costs. So which networking technology is the best choice?

For example: a factory as a typical case for an IoT system. A factory would need a large number of connected sensors and actuators scattered over a wide area. A wireless technology would be the best fit.

This is a wireless sensor network. Data from each sensor passes through the network node-to-node. The nodes in a wireless sensor network are low-cost devices, so they can be deployed in high volume. They also operate at low power so that they can run on battery, or even use technologies such as energy harvesting.



An edge node acts as a gateway between the wireless sensor network and the Internet.

It can also perform local processing, provide local storage, and can have a user interface.

The first obvious networking candidate for an IoT device is Wi-Fi, because it's everywhere.

There are newer networking technologies that allow for the development of low-cost, low-power solutions.

One of the major pieces of low-power wireless is the IEEE 802.15.4 radio standard. It was released in 2003.

6LoWPAN has been adopted by companies such as ARM and Cisco.

6LoWPAN provides encapsulation and header compression mechanisms that allow for briefer transmission times. 6LoWPAN will be the choice for wireless sensor networks and for other IoT systems that need IP-based protocols.

If IoT network is local and machine-to-machine, then the wireless protocols are good candidates. But if the goal is to remotely control devices or otherwise transmit data over the Internet, we'll need IPv6.

It's crucial that IoT networks all make use of the suite of Internet protocols. That's UDP, TCP, SSL, HTTP, and so on and must support IPv6. Why? Because the current IPv4 standard faces a global addressing shortage, as well as limited support for multicast, and poor global mobility.

With IPv6, it is much simpler for an IoT device to obtain a global IP address (public IP), which enables efficient peer-to-peer communication.

The importance of IP to the Internet of Things doesn't automatically mean that non-IP networks are useless. It just means that non-IP networks will require a gateway to reach the Internet.

2. Embedded Devices

What are the "Things" in the Internet of Things?

Thing is an embedded device, one that transmits and receives information over a network.

Embedded devices are based on microcontrollers, and run software with a small memory footprint.

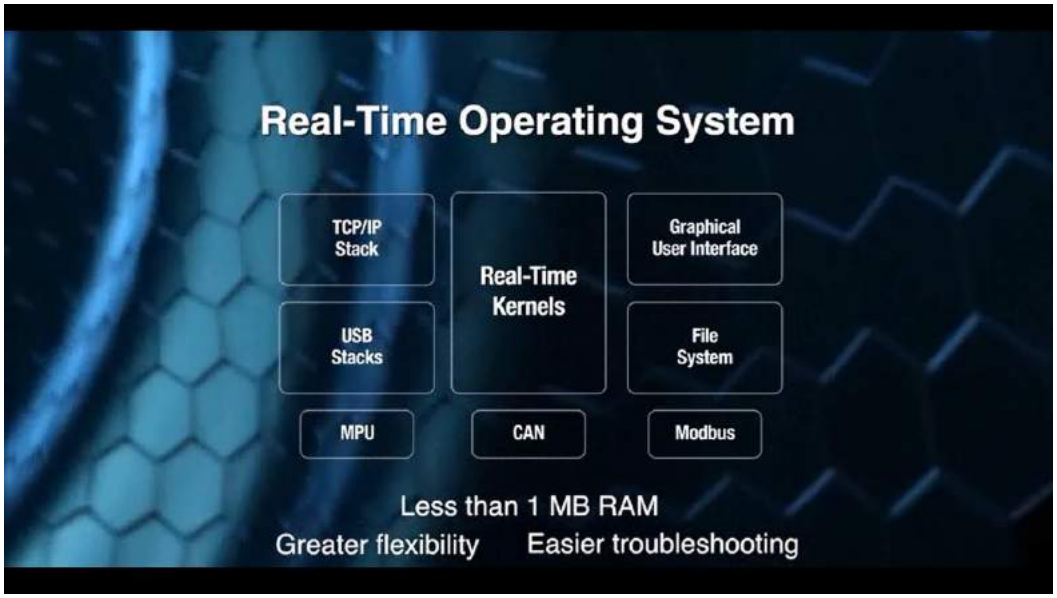
Some Linux and Android-based systems can also be described as embedded systems.

However, these general-purpose operating systems usually require an application processor, and have additional capabilities such as dynamic application loading.

This is why microcontroller-based embedded systems are often described as deeply embedded systems.

Embedded devices often use 16-bit or even 8-bit microcontrollers. But chips that feature 32-bit architectures have dropped in price over the last several years, and are becoming common in embedded systems. The greater capabilities of 32-bit chips present new choices for developers. Real-time operating system (RTOS) is now the preferred option, allowing for more flexible and extensible software to run on these systems.

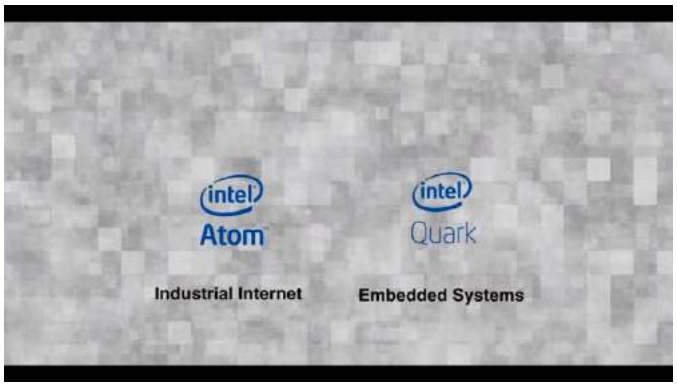
A complete RTOS – with a kernel, a user interface, a file system, USB support, networking, and more – can fit in a memory space of less than one megabyte.



With an RTOS, the software architecture of an embedded system can be more flexible.

So which processor architecture should be chosen?

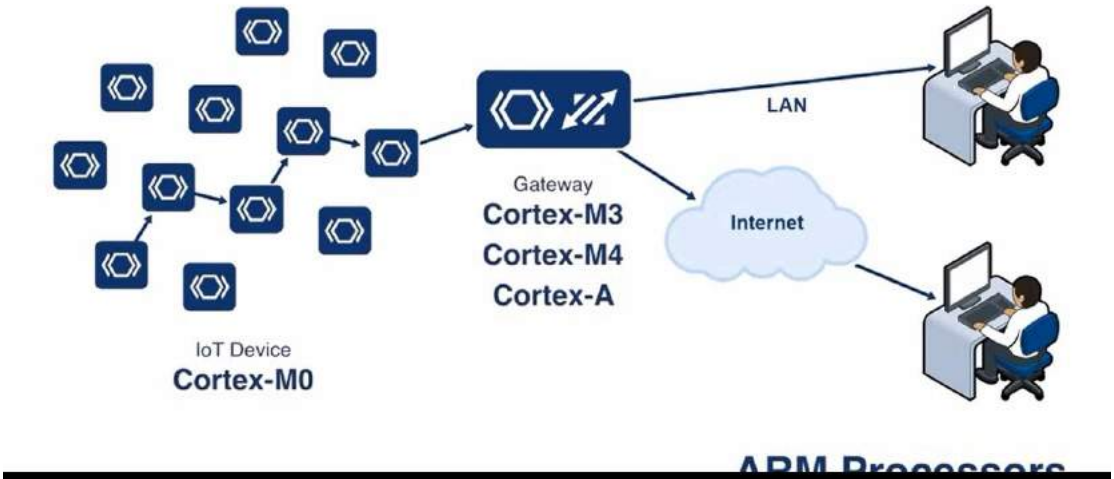
To date, the main contenders are Intel and ARM.



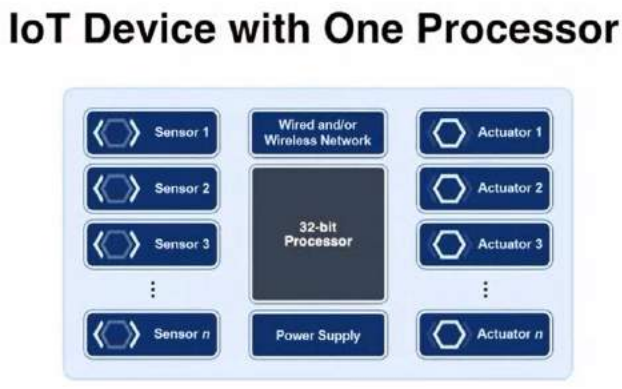
For an ARM processor in the IoT device, the Cortex-M0 is perfect. For gateways, the ARM Cortex-M3, M4 or Cortex-A are all good choices because of their greater processing capabilities.

The programming languages used in deeply embedded systems include C, C++ and sometimes Java. Java is attractive for IoT devices because of the huge number of Java developers worldwide.

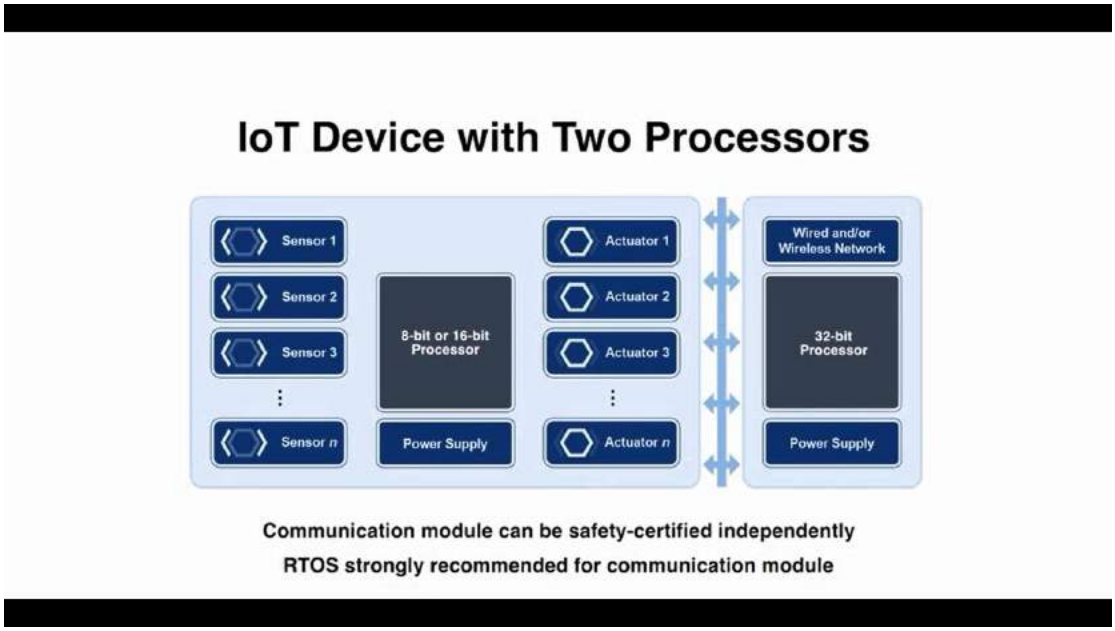
Oracle's Java ME Embedded is designed for small devices. Oracle recommends up to 700 Kilobytes of RAM and up to 2 megabytes of flash memory in the device for Java alone, plus support for a network connection.



When cost is not an issue, you can pick a single powerful processor to run all the tasks required of your device.



However, a common engineering compromise is to use two processors in the device. One low-cost processor is used for the physical-world interface, like a sensor. This would be an 8 or 16-bit chip. And a second 32-bit processor runs the network interface. This second processor is often placed in a separate module, one that can be certified separately for standards compliance.



3. Network Protocols and the Internet

People make use of the Internet through the Web, e-mail, and texting. In the Internet of Things, embedded devices exchange information with each other. But these devices don't have the machine equivalent of Web browsers and social media.

When designing IoT, one should think about how local network connects to the Internet. It can be done with a gateway or build this capability into the device itself.

The heart of the Internet is TCP/IP, the Internet protocol suite. It can be represented using the OSI seven-layer reference model.

UDP is a simpler protocol than TCP. It's commonly used for the DNS (domain name system) and for DHCP. But it's now finding a new home in sensor data acquisition and remote control. It's better suited for real-time data applications such as voice and video.

Can we build an IoT system using familiar Web technologies?

Yes, we can, but the result won't be as efficient as with the newer protocols.

HTTP and Web sockets are common Internet standards.

They can be used to deliver information encoded in XML or JavaScript Object Notation.

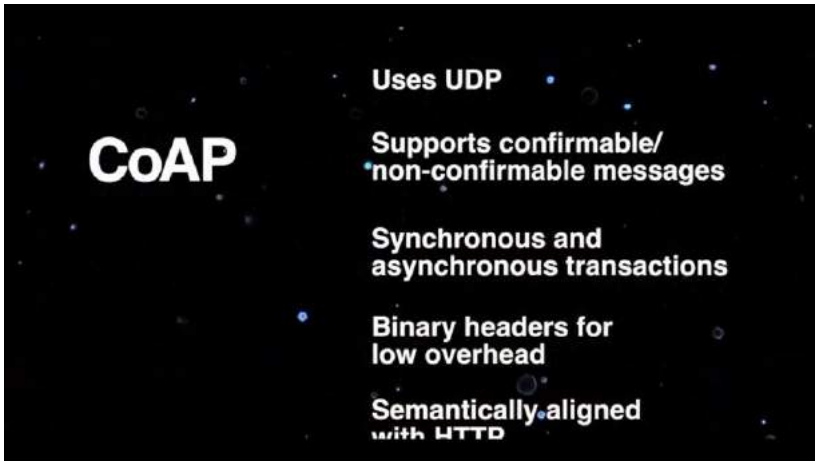
HTTP is the foundation of the client-server model used for the Web. Device should include only an HTTP client, not a server. Why?

An embedded device should never be set up to receive outside connections to prevent outsider's access to the local network.

Web Socket is a protocol that provides a full-duplex link between client and server. It's part of the HTML 5 specification. Web Socket simplifies much of the complexity around bi-directional Web communication.

The downside of these Web protocols is that they're often too data heavy for IoT applications.

By contrast, **CoAP** was designed especially for use in devices operating on battery or energy harvesting. It works a lot like HTTP.



MQTT is a publish-and-subscribe transport that is extremely lightweight. It helps minimize the resource requirements for IoT device, and can handle unreliable networks. MQTT runs on large networks of small devices that need to be monitored from a back-end server. Unlike CoAP, it's not designed for device-to-device transfer, or multicast.

4. The Cloud

In practical terms, cloud computing is an array of networked computers that allow you to offload processing tasks or storage from your embedded system. It's a simple idea, but hides a lot of underlying complexity.

Examples: Apple's iCloud, Google Cloud Platform, Microsoft OneDrive, and many others.

This cloud applications are for PC and mobile-based cloud computing.

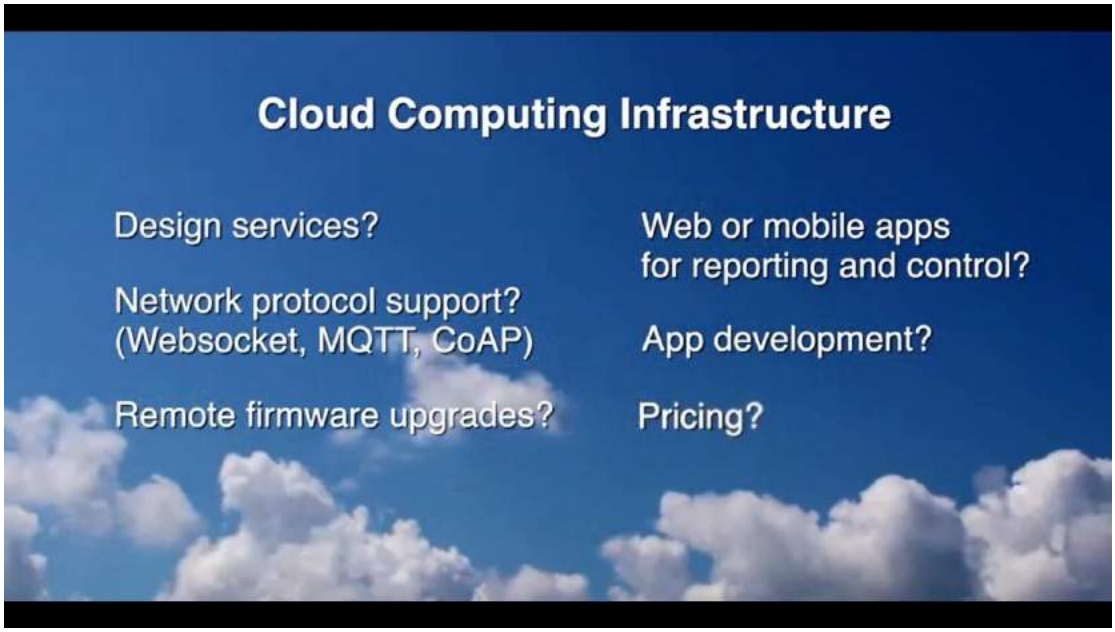
Embedded developers need something similar for embedded devices.

By the end of this decade, we will be surrounded by literally billions of IoT devices that will produce huge amounts of data.

By one estimate, there will be over five thousand gigabytes of data for every human on the planet by the year 2020. This is part of the reason why some people say that every company will become a software company.

If one thinking about outsourcing the infrastructure for cloud computing, some hard questions must be asked for service provider:

- Do they provide design services to help selection of communication hardware and software?
- Do they support cloud protocols such as Websocket, MQTT, CoAP, etc?
- Do they have support for remote and secure firmware upgrades?
- What software do they have for viewing and processing data, and for remote control of devices?
- Do they offer Web and mobile app development?
- What is their pricing structure?
- Also looking for companies that allow testing the system first with a free or low-cost account.
- Will the application need access to third-party data?



Some services offer data from public sources to help build a more useful application.

For example, if like to build a smart energy application, you might need real-time access to weather data or utility prices.

To keep up, data analytics will need systems that are smarter. We'll need systems that can learn based on their own experiences, and from our interactions with them. These are called Cognitive Systems.

Cloud computing have immediate access to structured information, like databases, and unstructured information, like medical imaging and social media content.

There are still unanswered questions about how these billions of interconnected devices will interact. All these data will need some common format, which is something we don't have yet. And what about protocols for network discovery? How will devices know how to connect to each other and to other networks?

There's much work left to do. And there are still places for innovative companies to make their mark.

5. IoT and RTOS

We will discuss the software that runs on embedded devices.

IoT devices are embedded systems that transmit and receive information over a network.

These networked devices need a capable operating system. The software for IoT device must be scalable, to accommodate a wide range of devices. The software should be modular, so you can choose only the components you need. It has to have a variety of connectivity options, such as ***Wi-Fi, Ethernet, USB, and Bluetooth***. And it must be reliable, so the device can be certified for safety-critical applications.

Why a real time operating system? Why not something like Linux?

Linux has a disadvantage when compared to a real-time operating system: Memory footprint. It simply will not run on 8-bit or 16-bit chips. And even many 32-bit MCUs don't have enough onboard RAM to run the Linux kernel.

Of course, Linux has many uses in embedded devices, particularly ones that provide graphically rich user interfaces. But there are thousands of applications for which Linux is not the right choice.

There are two broad categories of IoT devices: industrial and consumer. And the middleware needed by their applications can be different.

For example, wireless sensor device is a low-power, low-cost device that may run for years entirely on battery. It might typically use a Cortex-M chip. It would need an efficient wireless protocol such as 6LoWPAN to reduce transmission time and save power.

Compare this with a consumer IoT device. It typically might use a Cortex-A processor and it might need a Java virtual machine (JVM). It might need to support different kinds of connections, such as data over power lines (PLC) or specialized networking protocols.

These requirements will drive the choice of operating system, because we don't want the features of the device limited by the choice of platform. A flexible, scalable RTOS that runs on a variety of 16 and 32-bit chips will allow to meet tight memory requirements and reduce processor demands.

An important way to reduce the amount of RAM and flash memory needed in IoT device is to use modular software. By separating the kernel from middleware, protocols, and applications, one can pick and choose only the components needed and it simplifies the development process, especially if developing a family of devices.

The device must be able to connect to IP networks using efficient protocols such as 6LoWPAN. A modular operating system will allow selecting the specific protocol stacks needed, saving memory on the device, and reducing costs.

Another thing to consider: A lot of embedded devices are deployed in fields that demand perfect safety and reliability.

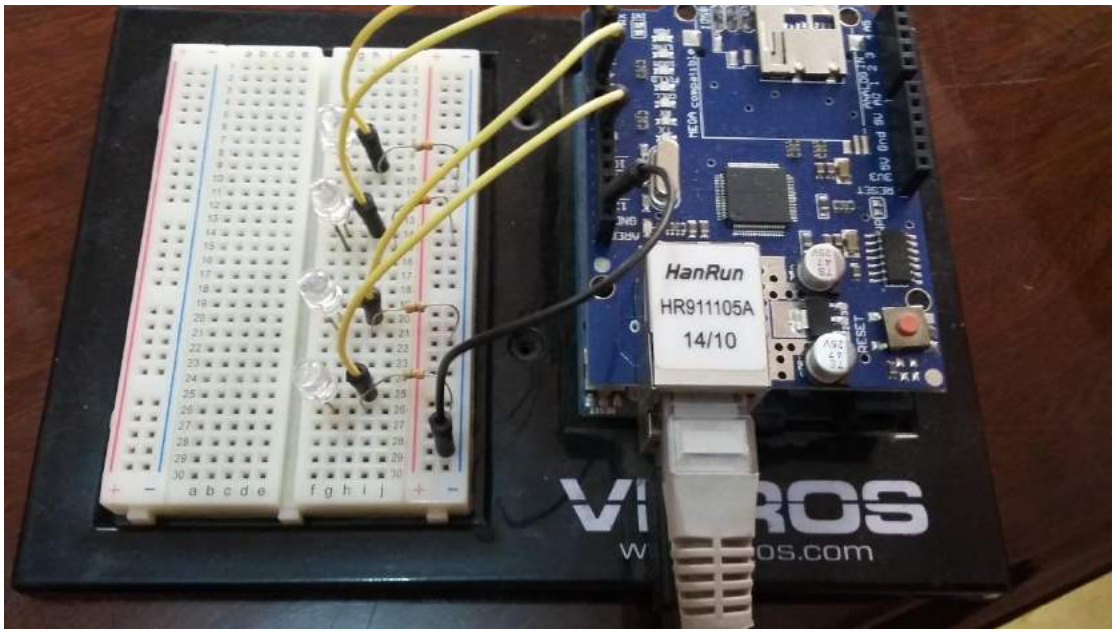
The components of IoT devices then may include: A RTOS, local networking with support for Ethernet, Wi-Fi, and Bluetooth. It provides support for IoT protocols, including HTTP client and server, and MQTT; it has a Java VM designed for deeply embedded systems; and it features Web services such as cloud-server interfaces, data brokering, and cloud storage. It runs on a huge number of MCUs.

6. Practical Implementation

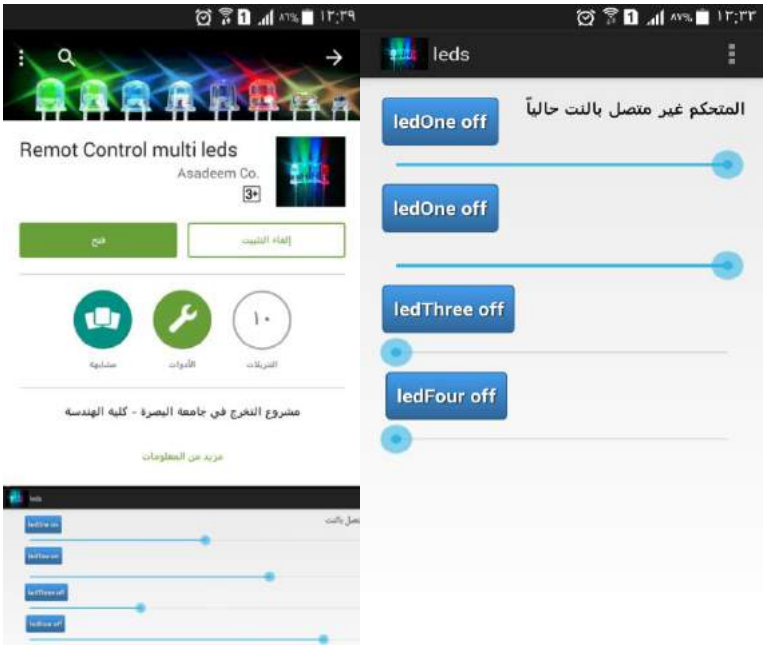
Two examples are done here for controlling things remotely via internet we are going to explain subsequently.

a. Arduino Server to control LED blinking and brightness

Arduino Uno microcontroller board with Ethernet port is programmed as a client server and connected to home router. As a simple plant to control is four different colors LEDs connected to digital output of the Arduino controller with series resistors as shown in Figure below:

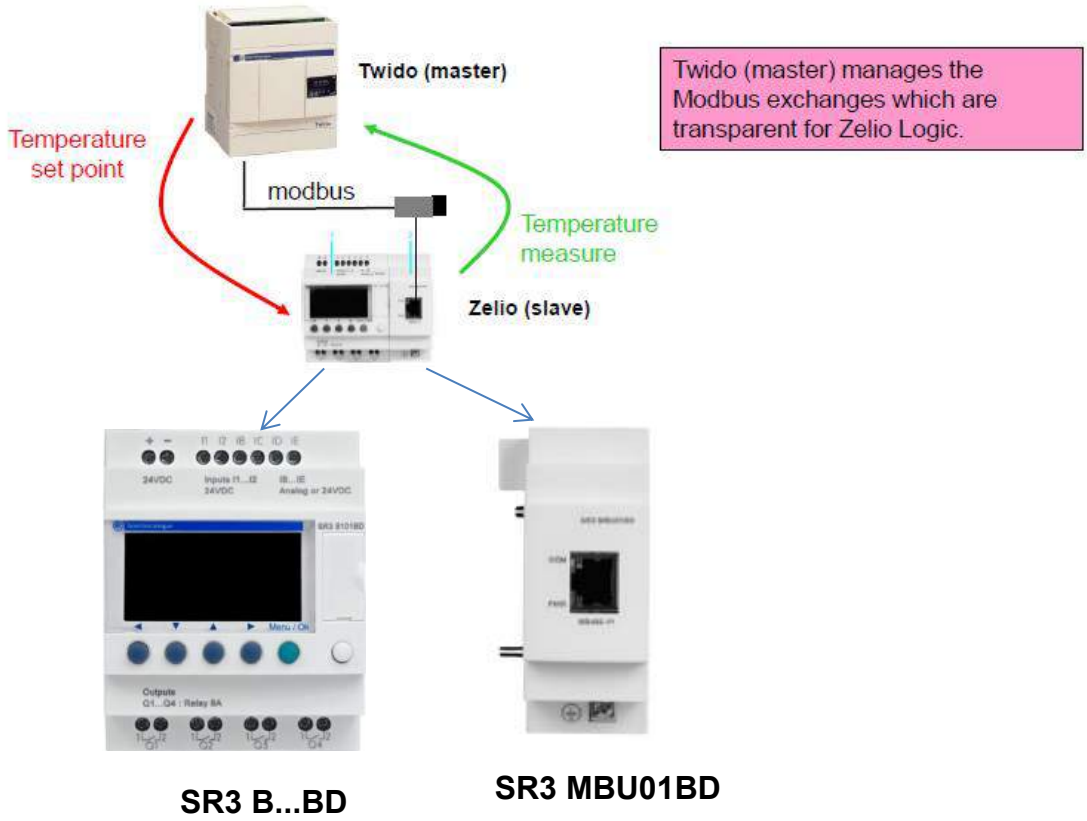


An android application is designed for LED ON/OFF and brightness control, which can be done easily from anywhere in the world, using Java Eclipse software and published on Google Play. Then, this application is downloaded from Google paly and installed on a mobile phone. User interface for this application in the Google Play (before downloading) and user interface after downloading into a mobile are shown in Figure below:

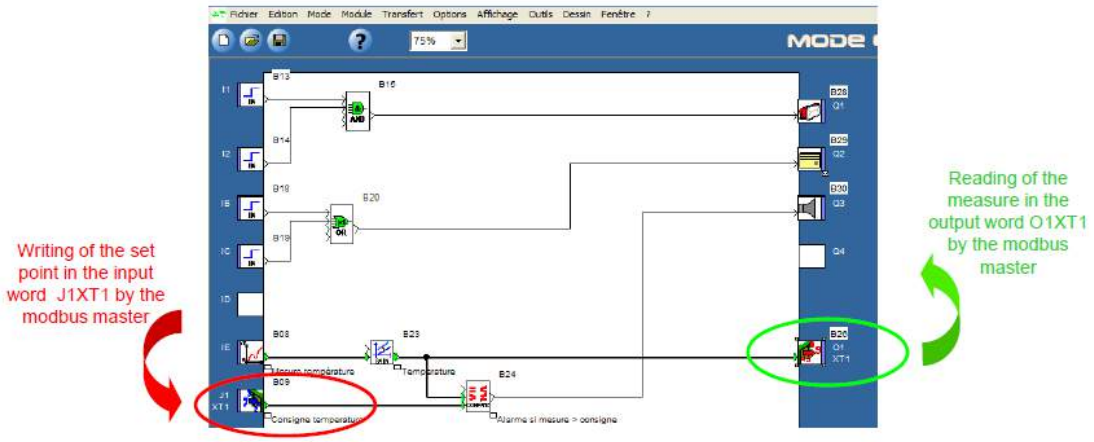


b. Remotely access HMI and PLC system

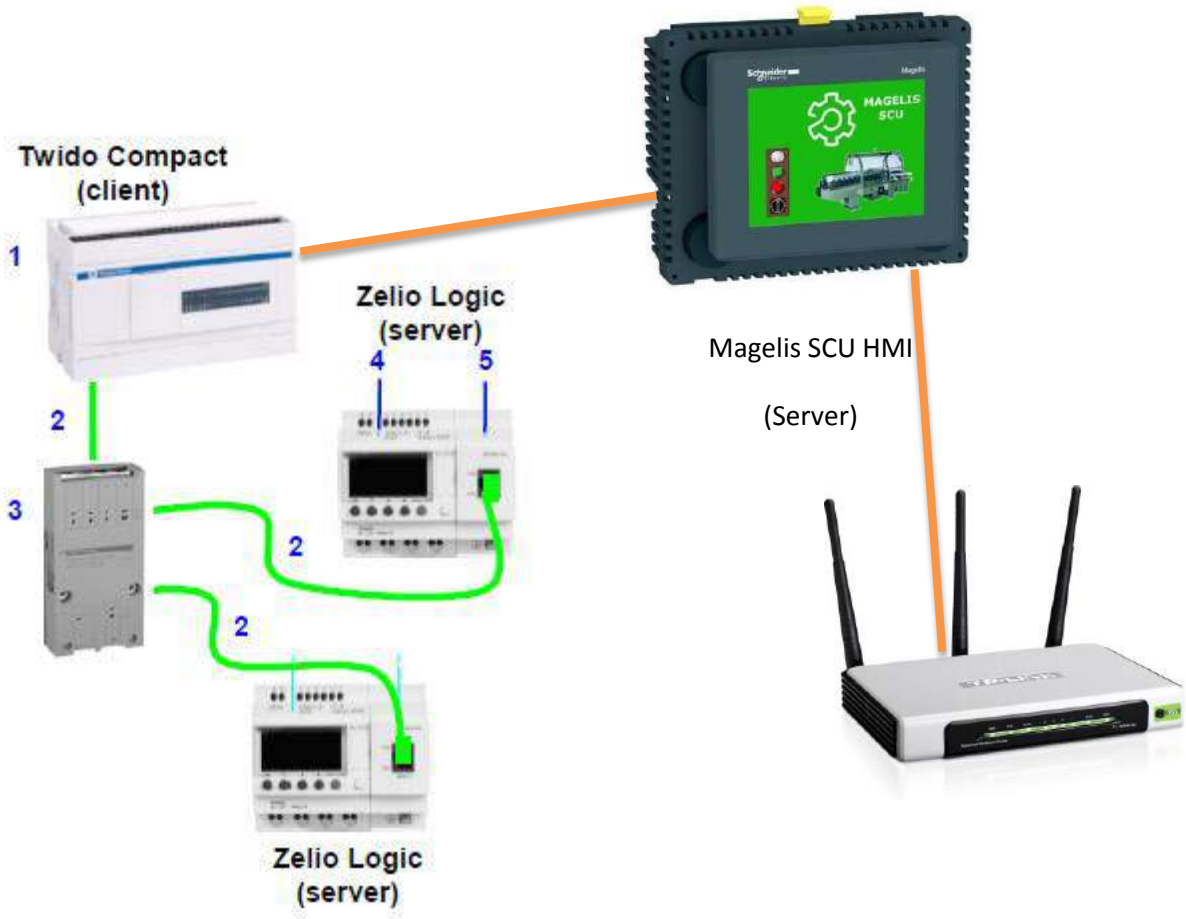
The application Zelio PLC checks the temperature while the Twido controller transmits the temperature set point (threshold) to Zelio via the fieldbus modbus. The measure of temperature is wiring to the Zelio analog input. If the temperature is higher than the threshold, an alarm is activated.

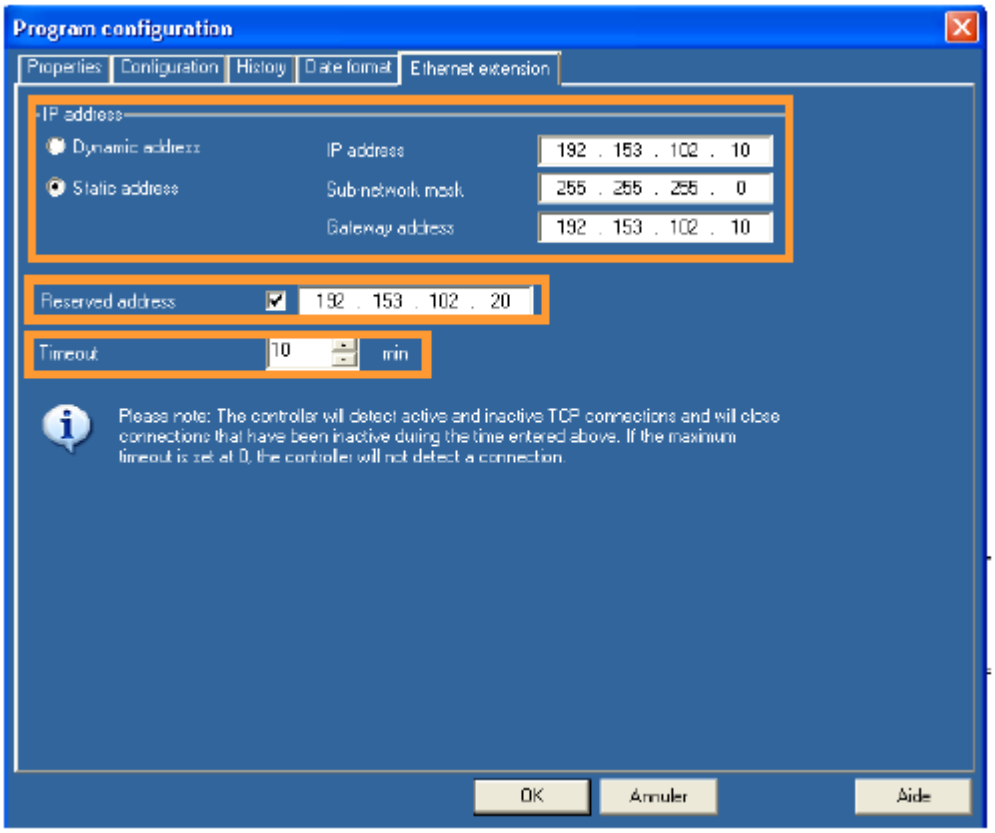


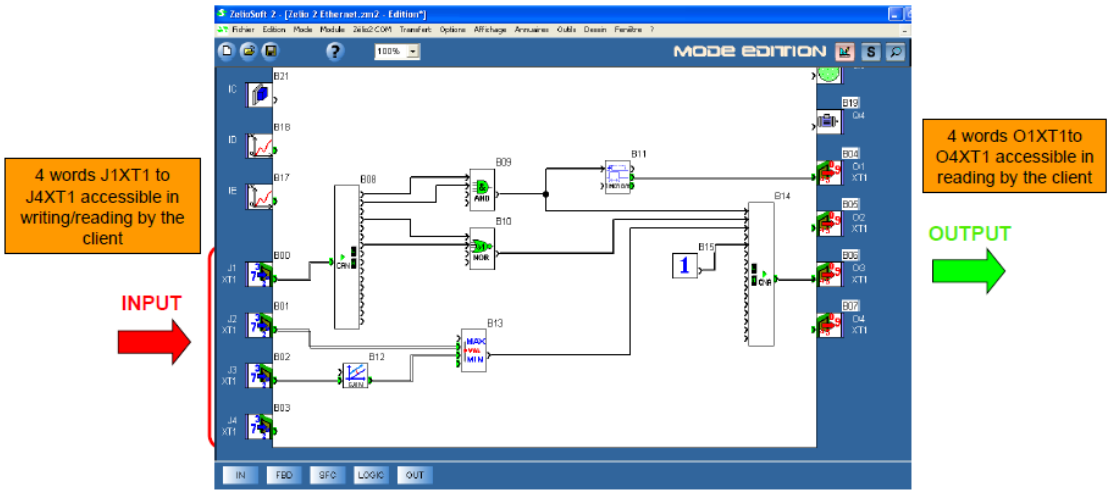
The temperature is measured by L35 high accuracy temperature sensor. Using the input word J1XT1 to write the temperature set point and the output word O1XT1 to read the measurement.



Internet Based Control







Note: The above two practical examples are implemented and will be show up in the showcase of the symposium with choosing the appropriate IP addresses.