**PAPER NAME : INTERNET OF THINGS** **PAPER CODE : 17PCSE06**

# UNIT – I

## INTERNET OF THINGS

## OVERVIEW

**INTRODUCTION:**

The IOT concept was coined by a member of the Radio Frequency Identification (RFID) development community in 1999, and it has recently become more relevant to the practical world largely because of the growth of mobile devices, embedded and ubiquitous communication, cloud computing and data analytics.

Imagine a world where billions of objects can sense, communicate and share information, all interconnected over public or private Internet Protocol (IP) networks. These interconnected objects have data regularly collected, analyzed and used to initiate action, providing a wealth of intelligence for planning, management and decision making. This is the world of the Internet of Things (IOT).

**DEFINITION:**

**Internet of things (IOT)** is a network of physical objects. The internet is not only a network of computers, but it has evolved into a network of device of all type and sizes , vehicles, smart phones, home appliances, toys, cameras, medical instruments and industrial systems, animals, people, buildings, all connected ,all communicating & sharing information based on stipulated protocols in order to achieve smart reorganizations, positioning, tracing, safe & control & even personal real time online monitoring , online upgrade, process control & administration.

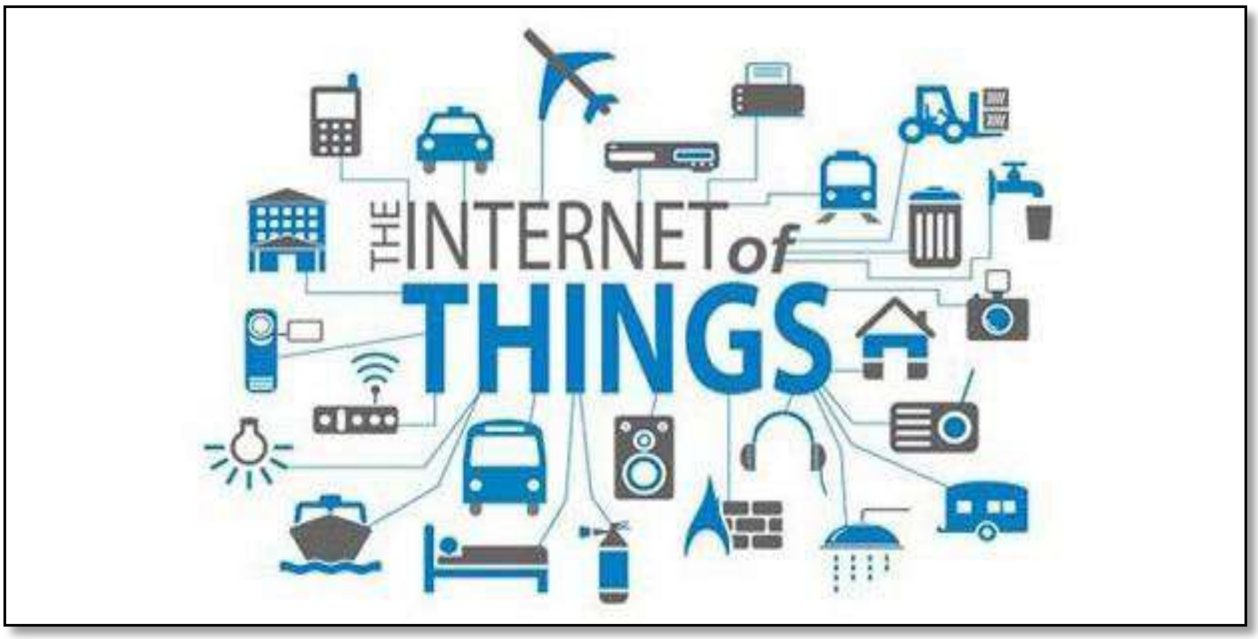
We define IOT into three categories as below:

Internet of things is an internet of three things:

(1). People to people,

(2). People to machine /things,

(3). Things /machine to things /machine, Interacting through internet.



Internet Of Things

**Internet of Things Vision :** Internet of Things (IoT) is a concept and a paradigm that considers pervasive presence in the environment of a variety of things/objects that through wireless and wired connections and unique addressing schemes are able to interact with each other and cooperate with other things/objects to create new applications/services and reach common goals. In this context the research and development challenges to create a

smart world are enormous. A world where the real, digital and the virtual are converging to create smart environments that make energy, transport, cities and many other areas more intelligent.
Internet of Things is refer to the general idea of things, especially everyday objects, that are readable, recognizable, locatable, addressable through information sensing device and/or controllable via the Internet, irrespective of the communication means (whether via RFID, wireless LAN, wide area networks, or other means). Everyday objects include not only the electronic devices we encounter or the products of higher technological development such as vehicles and equipment but things that we do not ordinarily think of as electronic at all - such as food , clothing, chair, animal, tree, water etc.
Internet of Things is a new revolution of the Internet. Objects make themselves recognizable and they obtain intelligence by making or enabling context related decisions thanks to the fact that they can communicate information about themselves. They can access information that has been aggregated by other things, or they can be components of complex services. This transformation is concomitant with the emergence of cloud computing capabilities and the transition of the Internet towards IPv6 with an almost unlimited addressing capacity.
The goal of the Internet of Things is to enable things to be connected anytime, anyplace, with anything and anyone ideally using any path/network and any service.

## CHARACTERISTICS

### Interconnectivity:

Anything can be interconnected with the global information and communication infrastructure.

### Things-related services:

The IoT is capable of providing thing-related services within the constraints of things, such as privacy protection and semantic consistency between physical things and their associated virtual things. In order to provide thing-related services within the constraints of things, both the technologies in physical world and information world will change.

### Heterogeneity:

The devices in the IoT are heterogeneous as based on different hardware platforms and networks. They can interact with other devices or service platforms through different networks.

### Dynamic changes:

The state of devices change dynamically, e.g., sleeping and waking up, connected and/or disconnected as well as the context of devices including location and speed. Moreover, the number of devices can change dynamically.

### Enormous scale:

The number of devices that need to be managed and that communicate with each other will be at least an order of magnitude larger than the devices connected to the current Internet.
Even more critical will be the management of the data generated and their interpretation for application purposes. This relates to semantics of data, as well as efficient data handling.

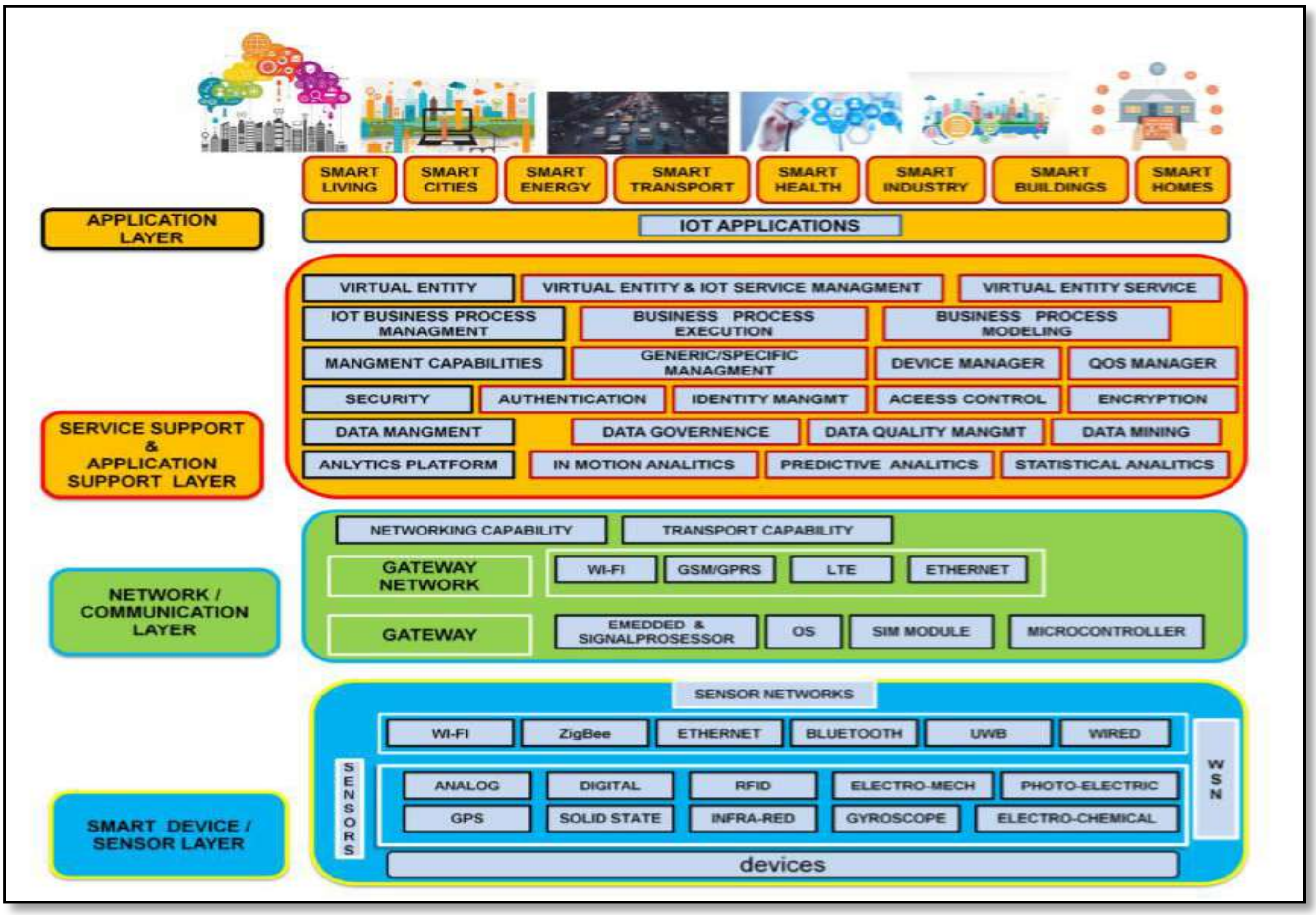
### Safety:

As we gain benefits from the IoT, we must not forget about safety. As both the creators and recipients of the IoT, we must design for safety. This includes the safety of our personal data and the safety of our physical well-being. Securing the endpoints, the networks, and the data moving across all of it means creating a security paradigm that will scale.

### Connectivity:

Connectivity enables network accessibility and compatibility. Accessibility is getting on a network while compatibility provides the common ability to consume and produce data.

## IOT ARCHITECTURE



IOT architecture consists of different layers of technologies supporting IOT. It serves to illustrate how various technologies relate to each other and to communicate the scalability, modularity and configuration of IOT deployments in different scenarios. Figure 4 shows detailed architecture of IOT.
The functionality of each layer is described below,

**A. smart device / sensor layer:**

The lowest layer is made up of smart objects integrated with sensors. The sensors enable the interconnection of the physical and digital worlds allowing real-time information to be collected and processed. There are various types of sensors for different purposes. The sensors have the capacity to take measurements such as temperature, air quality, speed, humidity, pressure, flow, movement and electricity etc. In some cases, they may also have a degree of memory, enabling them to record a certain number of measurements. A sensor can measure the physical property and convert it into signal that can be understood by an instrument. Sensors are grouped according to their unique purpose such as environmental sensors, body sensors, home appliance sensors and vehicle telematics sensors, etc.

Most sensors require connectivity to the sensor gateways. This can be in the form of a Local Area Network (LAN) such as Ethernet and Wi-Fi connections or Personal Area Network (PAN) such as ZigBee, Bluetooth and Ultra Wideband (UWB). For sensors that do not require connectivity to sensor aggregators, their connectivity to backend servers/applications can be provided using Wide Area Network (WAN) such as GSM, GPRS and LTE. Sensors that use low power and low data rate connectivity, they typically form networks commonly known as wireless sensor networks (WSNs). WSNs are gaining popularity as they can accommodate far more sensor nodes while retaining adequate battery life and covering large areas.

**B. Gateways and Networks**

Massive volume of data will be produced by these tiny sensors and this requires a robust and high performance wired or wireless network infrastructure as a transport medium. Current networks, often tied with very different protocols, have been used to support machine-to-machine (M2M) networks and their applications. With demand needed to serve a wider range of IOT services and applications such as high speed transactional services, context-aware applications, etc, multiple networks with various technologies and access protocols are needed to work with each other in a heterogeneous configuration. These networks can be in the form of a private, public or hybrid models and are built to support the communication requirements for latency, bandwidth or security. Various gateways (microcontroller, microprocessor...) & gateway networks (WI-FI, GSM, GPRS…)

**C. Management Service Layer**

The management service renders the processing of information possible through analytics, security controls, process modeling and management of devices.

One of the important features of the management service layer is the business and process rule engines. IOT brings connection and interaction of objects and systems together providing information in the form of events or contextual data such as temperature of goods, current location and traffic data. Some of these events require filtering or routing to postprocessing systems such as capturing of periodic sensory data, while others require response to the immediate situations such as reacting to emergencies on patient's health conditions. The rule engines support the formulation of decision logics and trigger interactive and automated processes to enable a more responsive IOT system.

In the area of analytics, various analytics tools are used to extract relevant information from massive amount of raw data and to be processed at a much faster rate. Analytics such as inmemory analytics allows large volumes of data to be cached in random access memory (RAM) rather than stored in physical disks. In-memory analytics reduces data query time and augments the speed of decision making. Streaming analytics is another form of analytics where analysis of data, considered as data-in-motion, is required to be carried out in real time so that decisions can be made in a matter of seconds.

Data management is the ability to manage data information flow. With data management in the management service layer, information can be accessed, integrated and controlled. Higher layer applications can be shielded from the need to process unnecessary data and reduce the risk of privacy disclosure of the data source. Data filtering techniques such as data anonymisation, data integration and data synchronization, are used to hide the details of the information while providing only essential information that is usable for the relevant applications. With the use of data abstraction, information can be extracted to provide a common business view of data to gain greater agility and reuse across domains. Security must be enforced across the whole dimension of the IOT architecture right from the smart object layer all the way to the application layer. Security of the system prevents system hacking and compromises by unauthorized personnel, thus reducing the possibility of risks.

**D. Application Layer**

The IoT application covers "smart" environments/spaces in domains such as: Transportation, Building, City, Lifestyle, Retail, Agriculture, Factory, Supply chain, Emergency, Healthcare, User interaction, Culture and tourism, Environment and Energy.

## APPLICATION AREAS

**A. IOsL (Internet of smart living):**

Remote Control Appliances: Switching on and off remotely appliances to avoid accidents and save energy.

Weather: Displays outdoor weather conditions such as humidity, temperature, pressure, wind speed and rain levels with ability to transmit data over long distances.

Smart Home Appliances**:** Refrigerators with LCD screen telling what's inside, food that's about to expire, ingredients you need to buy and with all the information available on a Smartphone app. Washing machines allowing you to monitor the laundry remotely, and. Kitchen ranges with interface to a Smartphone app allowing remotely adjustable temperature control and monitoring the oven's self-cleaning feature.

Safety Monitoring: cameras, and home alarm systems making people feel safe in their daily life at home.

Intrusion Detection Systems: Detection of window and door openings and violations to prevent intruders, Energy and Water Use: Energy and water supply consumption monitoring to obtain advice on how to save cost and resources, & many more…

**B. IOsC ( Internet of smart cities):**

Structural Health: Monitoring of vibrations and material conditions in buildings, bridges and historical monuments, Lightning: intelligent and weather adaptive lighting in street lights.
Safety: Digital video monitoring, fire control management, public announcement systems.
Transportation: Smart Roads and Intelligent High-ways with warning messages and diversions according to climate conditions and unexpected events like accidents or traffic jams.
Smart Parking: Real-time monitoring of parking spaces availability in the city making residents able to identify and reserve the closest available spaces.
Waste Management: Detection of rubbish levels in containers to optimize the trash collection routes. Garbage cans and recycle bins with RFID tags allow the sanitation staff to see when garbage has been put out.

**C. IOsE (Internet of smart environment):**

Air Pollution monitoring: Control of CO2 emissions of factories, pollution emitted by cars and toxic gases generated in farms.
Forest Fire Detection: Monitoring of combustion gases and preemptive fire conditions to define alert zones.
Weather monitoring: weather conditions monitoring such as humidity, temperature, pressure, wind speed and rain, Earthquake Early Detection.
Water Quality: Study of water suitability in rivers and the sea for eligibility in drinkable use.
River Floods: Monitoring of water level variations in rivers, dams and reservoirs during rainy days.
Protecting wildlife: Tracking collars utilizing GPS/GSM modules to locate and track wild animals and communicate their coordinates via SMS.

**D. IOsI (Internet of smart industry):**

Explosive and Hazardous Gases: Detection of gas levels and leakages in industrial environments, surroundings of chemical factories and inside mines, Monitoring of toxic gas and oxygen levels inside chemical plants to ensure workers and goods safety, Monitoring of water, oil and gas levels in storage tanks and Cisterns.
Maintenance and repair: Early predictions on equipment malfunctions and service maintenance can be automatically scheduled ahead of an actual part failure by installing sensors inside equipment to monitor and send reports.

**E. IOsH (Internet of smart health):**

Patients Surveillance: Monitoring of conditions of patients inside hospitals and in old people's home.
Medical Fridges: Control of conditions inside freezers storing vaccines, medicines and organic elements.
Fall Detection: Assistance for elderly or disabled people living independent.
Dental: Bluetooth connected toothbrush with Smartphone app analyzes the brushing uses and gives information on the brushing habits on the Smartphone for private information or for showing statistics to the dentist.
Physical Activity Monitoring: Wireless sensors placed across the mattress sensing small motions, like breathing and heart rate and large motions caused by tossing and turning during sleep, providing data available through an app on the Smartphone.

**F. IOsE (internet of smart energy):**

Smart Grid: Energy consumption monitoring and management.
Wind Turbines/ Power house: Monitoring and analyzing the flow of energy from wind turbines & power house, and two-way communication with consumers' smart meters to analyze consumption patterns.
Power Supply Controllers: Controller for AC-DC power supplies that determines required energy, and improve energy efficiency with less energy waste for power supplies related to computers, telecommunications, and consumer electronics applications.
Photovoltaic Installations: Monitoring and optimization of performance in solar energy plants.

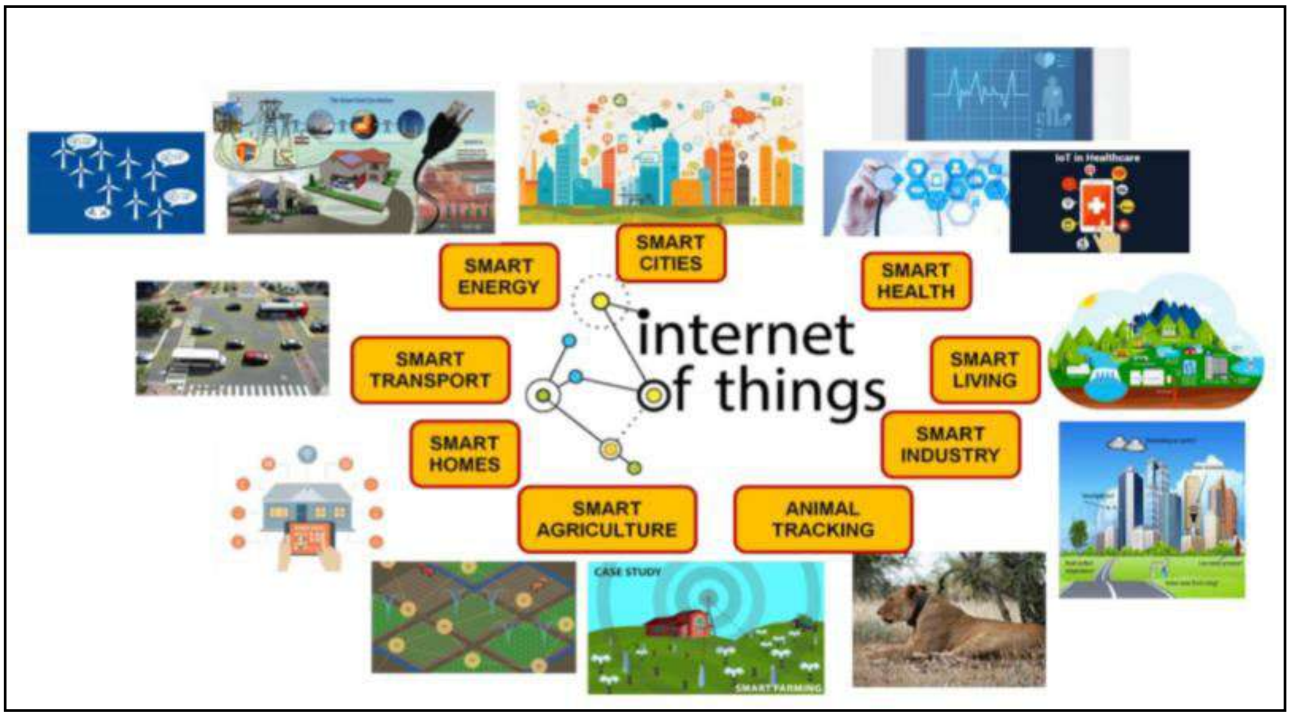
**G. IOsA (internet of smart agriculture):**

Green Houses: Control micro-climate conditions to maximize the production of fruits and vegetables and its quality.
Compost: Control of humidity and temperature levels in alfalfa, hay, straw, etc. to prevent fungus and other microbial contaminants.
Animal Farming/Tracking: Location and identification of animals grazing in open pastures or location in big stables, Study of ventilation and air quality in farms and detection of harmful gases from excrements.

Offspring Care: Control of growing conditions of the offspring in animal farms to ensure its survival and health.
field Monitoring: Reducing spoilage and crop waste with better monitoring, accurate ongoing data obtaining, and management of the agriculture fields, including better control of fertilizing, electricity and watering.
The IOT application area is very diverse and IOT applications serve different users. Different user categories have different driving needs. From the IOT perspective there are three important user categories:
(1) The individual citizens

(2) Community of citizens (citizens of a city, a region, country or society as a whole)

(3) The enterprises.



## THE TECHNOLOGY OF THE INTERNET OF THINGS

Internet Of things (IOT) is a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.

The Internet of Things the communication is extended via Internet to all the things that surround us. The Internet of Things is much more than machine to machine communication, wireless sensor networks, sensor networks , 2G/3G/4G,GSM,GPRS,RFID, WI-FI, GPS, microcontroller, microprocessor etc. These are considered as being the enabling technologies that make “Internet of Things” applications possible.

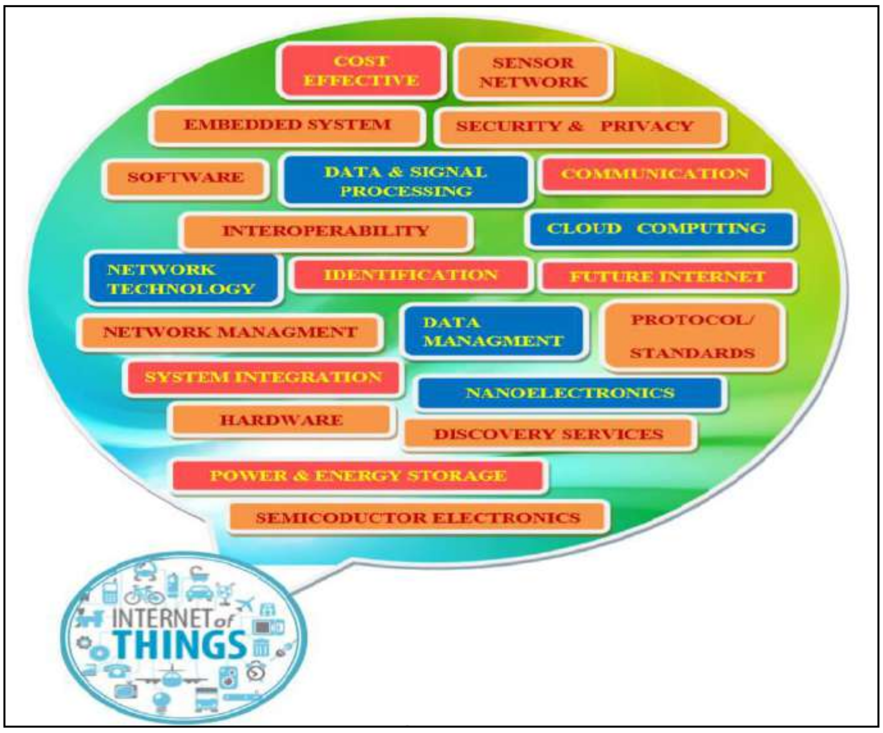
Enabling technologies for the Internet of Things are grouped by three categories:

(1) technologies that enable “things” to acquire contextual information

(2) technologies that enable “things” to process contextual information

(3) technologies to improve security and privacy.

The first two categories can be jointly understood as functional building blocks required building “intelligence” into “things”, which are indeed the features that differentiate the IoT from the usual Internet. The third category is not a functional but rather a de facto requirement, without which the penetration of the IoT would be severely reduced.



The Internet of Things is not a single technology, but it is a mixture of different hardware & software technology. The Internet of Things provides solutions based on the integration of information technology, which refers to hardware and software used to store, retrieve, and process data and communications technology which includes electronic systems used for communication between individuals or groups.

There is a heterogeneous mix of communication technologies, which need to be adapted in order to address the needs of IoT applications such as energy efficiency, speed, security, and reliability. In this context, it is possible that the level of diversity will be scaled to a number a manageable connectivity technologies that address the needs of the IoT applications, are adopted by the market, they have already proved to be serviceable, supported by a strong technology alliance. Examples of standards in these categories include wired and wireless technologies like Ethernet, WI-FI, Bluetooth, ZigBee, GSM, and GPRS.

**SMARTS OBJECTS IN IOT:**

- The concept of smart in IoT is used for physical objects that are active, digital, networked, can operate to some extent autonomously, reconfigurable and has local control of the resources. The smart objects need energy, data storage, etc.
- A smart object is an object that enhances the interaction with other smart objects as well as with people also. The world of IoT is the network of interconnected heterogeneous objects (such as smart devices, smart objects, sensors, actuators, RFID, embedded computers, etc.) uniquely addressable and based on standard communication protocols.

In a day to day life, people have a lot of object with internet or wireless or wired connection. Such as:

- Smartphone
- Tablets
- TV computer

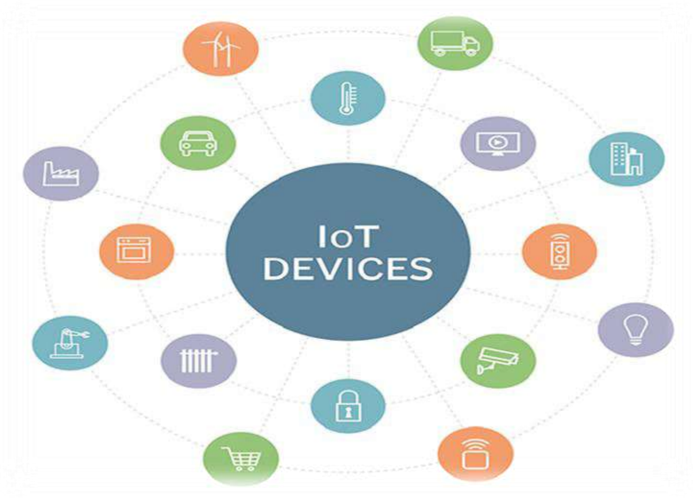
These objects can be interconnected among them and facilitate our daily life (smart home, smart cities) no matter the situation, localization, accessibility to a sensor, size, scenario or the risk of danger.

Smart objects are utilized widely to transform the physical environment around us to a digital world using the Internet of things (IoT) technologies.

A smart object carries blocks of application logic that make sense for their local situation and interact with human users. A smart object sense, log, and interpret the occurrence within themselves and the environment, and intercommunicate with each other and exchange information with people.

The work of smart object has focused on technical aspects (such as software infrastructure, hardware platforms, etc.) and application scenarios. Application areas range from supply-chain management and enterprise applications (home and hospital) to healthcare and industrial workplace support. As for human interface aspects of smart-object technologies are just beginning to receive attention from the environment.

**IOT DEVICES:**



Internet of Things Devices is non-standard devices that connect wirelessly to a network with each other and able to transfer the data. The IoT devices are enlarging the internet connectivity beyond standard devices such as smart phones, laptops, tablets, and desktops. Embedding these devices with technology enables us to communicate and interact over the networks and they can be remotely monitored and controlled.

There are large varieties of IoT devices available based on IEEE 802.15.4 standard. These devices range from wireless motes, attachable sensor-boards to interface-board which are useful for researchers and developers.

IoT devices include computer devices, software, wireless sensors, and actuators. These IoT devices are connected over the internet and enabling the data transfer among objects or people automatically without human intervention.

Some of the common and popular IoT devices are given below:

**Arduino Device:**

Arduino devices are the microcontrollers and microcontroller kit for building digital devices that can be sense and control objects in the physical and digital world. Arduino boards are furnished with a set of digital and analog input/output pins that may be interfaced to various other circuits. Some Arduino boards include USB (Universal Serial Bus) used for loading programs from the personal computer.

**Intel Galileo:**

The Intel Galileo Gen 2 Board includes the parts such as Intel Quark SoC processor, 256MB RAM, multiple ports and supports for Arduino device.

**Samsung Gear Fit:**

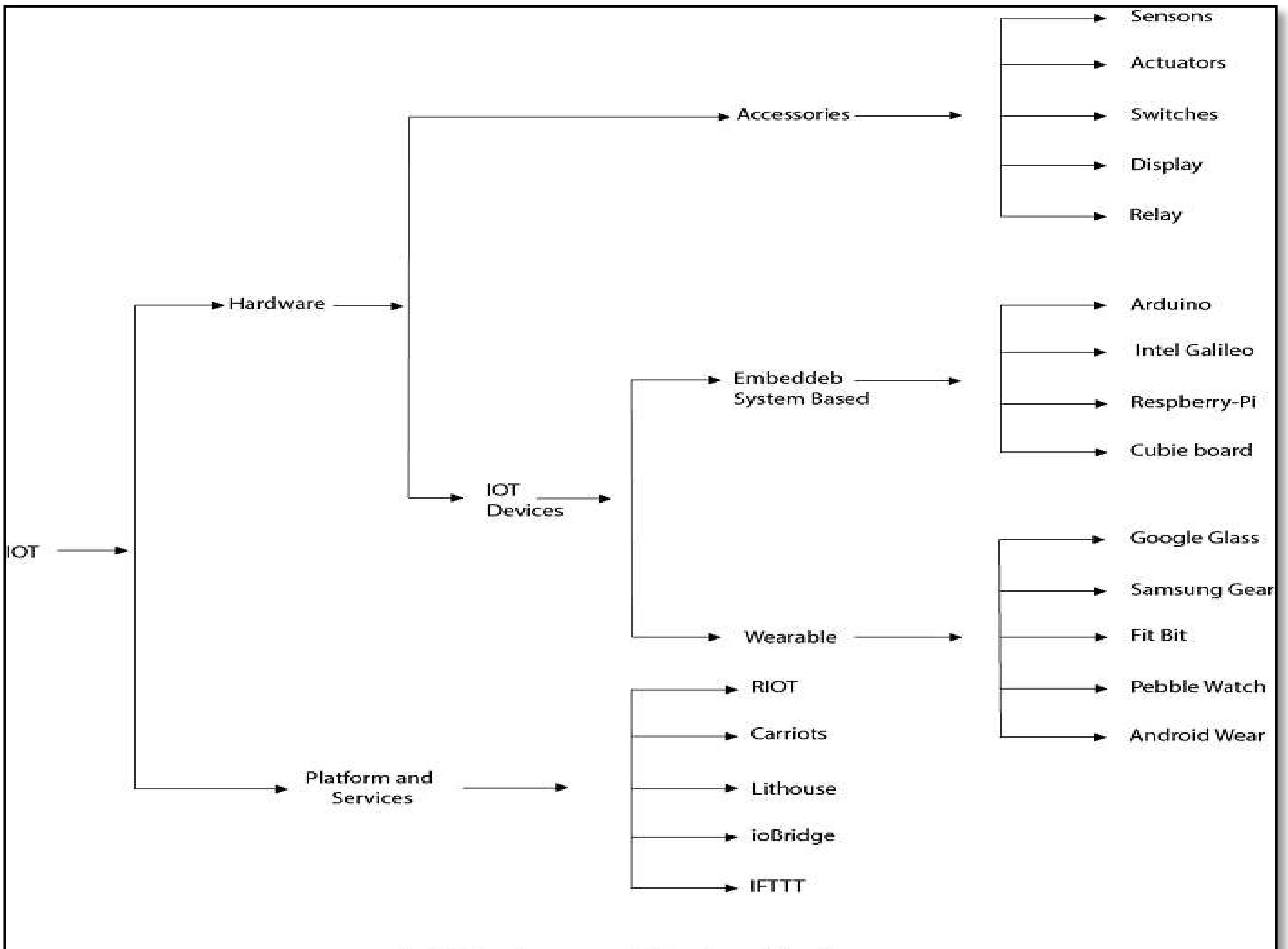
A Samsung Gear Fir device is a dustproof, water-resistant with fitness tracker features, a curved display, and long-lasting battery. This device receives alerts about emails and text messages, and it integrates with Samsung's S Health app.

**Sensor:**

A sensor is a device that reads the surrounding temperature, humidity, light, air quality control etc. There are different types of sensors available that reads different types of data. The sensors transmit these data over the networks or through which it is connected.

**Bluetooth Low Energy (BLE) Intelligent Beacon:**

A Bluetooth low energy beacon device is used to track the object located at a real time. Many companies use it to track the location of employees, assets, patients, and more in real time. This service primarily focuses on manufacturing, retail, and healthcare services.



IoT Devices and Technologies

Some of the essential properties of IoT devices are mention below:

- **Sense:** The devices that sense its surrounding environment in the form of temperature, movement, and appearance of things, etc.
- **Send and receive data:** IoT devices are able to send and receive the data over the network connection.
- **Analyze:** The devices can able to analyze the data that received from the other device over the internet networks.
- **Controlled:** IoT devices may control from some endpoint also. Otherwise, the IoT devices are themselves communicate with each other endlessly leads to the system failure.

## DESIGN PRINCIPLES FOR CONNECTED DEVICES

**What is IoT design?**

Designing for the Internet of Things (IoT) is the designing of connected products. IoT systems combine physical and digital components that collect data from physical devices and deliver actionable, operational insights.

These components include: physical devices, sensors, data extraction and secured communication, gateways, cloud servers, analytics, and dashboards.

Industry 4.0 Design Principles
Interoperability
Information transparency
Technical assistance
Decentralized decisions

1. **Interoperability**
   At the most fundamental level, a connected system requires sensors, machines, equipment, and sites, to communicate and exchange data. Interoperability is the underlying principle throughout all Industry 4.0 design processes.

2. **Information transparency**
   The rapid growth of connected devices means continuous bridging between the physical and digital worlds. In this context, information transparency means that physical processes should be recorded and stored virtually, creating a Digital Twin.

3. **Technical assistance**
   A driving benefit of IoT, technical assistance refers to the ability of connected systems to provide and display data that helps people to make better operational decisions and solve issues faster. In addition, IoT-enabled things should assist people in laborious tasks to improve productivity and safety.

4. **Decentralized decisions**
   The final principle of Industry 4.0 design is for the connected system to go beyond assisting and exchanging data, to be able to make decisions and execute requirements according to its defined logic.

**Designing with a Purpose**

In order for the Industrial IoT system to effectively fill its purpose, it must be designed with the relevant solution in mind.

Key industry 4.0 solutions:

1. Conditional monitoring
2. Digital twin analytics
3. Predictive quality
4. Predictive maintenance
5. Supply chain optimization

## CALM AND AMBIENT TECHNOLOGY

### Calm Technology:

"Designs that encalm and inform meet two human needs not usually met together. Information technology is more often the enemy of calm: pagers, cell phones, news services, the World Wide Web, email, TV, and radio bombard us frenetically."

It is for this important reason that Ubiquitous Computing goes hand in hand with a special technology, one which enables users to do more while at the same time a sense of calmness remains as well as the notion that you are still in full control of everything you are doing. This is called “Calm Technology”.

### Calm Technology Characteristics

A calm technology will move easily from the periphery of our attention, to the centre, and back.

This is fundamentally encalming, for two reasons:

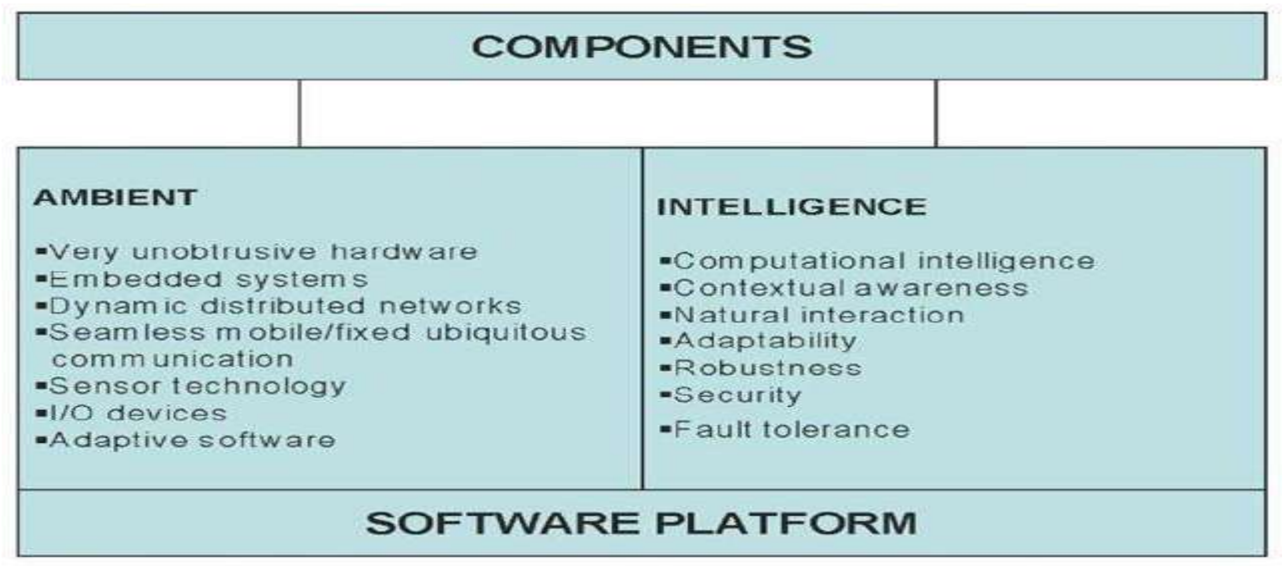
1. Placing things in the periphery

2. Bringing to center (of attention) when needed

### Ambient Technology:

Ambient Intelligence is the artificial intelligence which is totally human centric. Simplest example is the door that opens when it senses your presence. XBOX with Kinect is another simple example. But ambient intelligence systems are infinitely more complex.

In such a context, an emerging application and research field is the ambient intelligence, which refers to the capacity of an IoT system to sense the environment and to respond to the presence of people and builds upon pervasive computing, ubiquitous computing, profiling, context awareness, and human-centric computer.



## METAPHOR

A metaphor is a figure of speech that, for rhetorical effect, directly refers to one thing by mentioning another. It may provide clarity or identify hidden similarities between two ideas. Metaphors are often compared with other types of figurative language, such as antithesis, hyperbole, metonymy and simile.

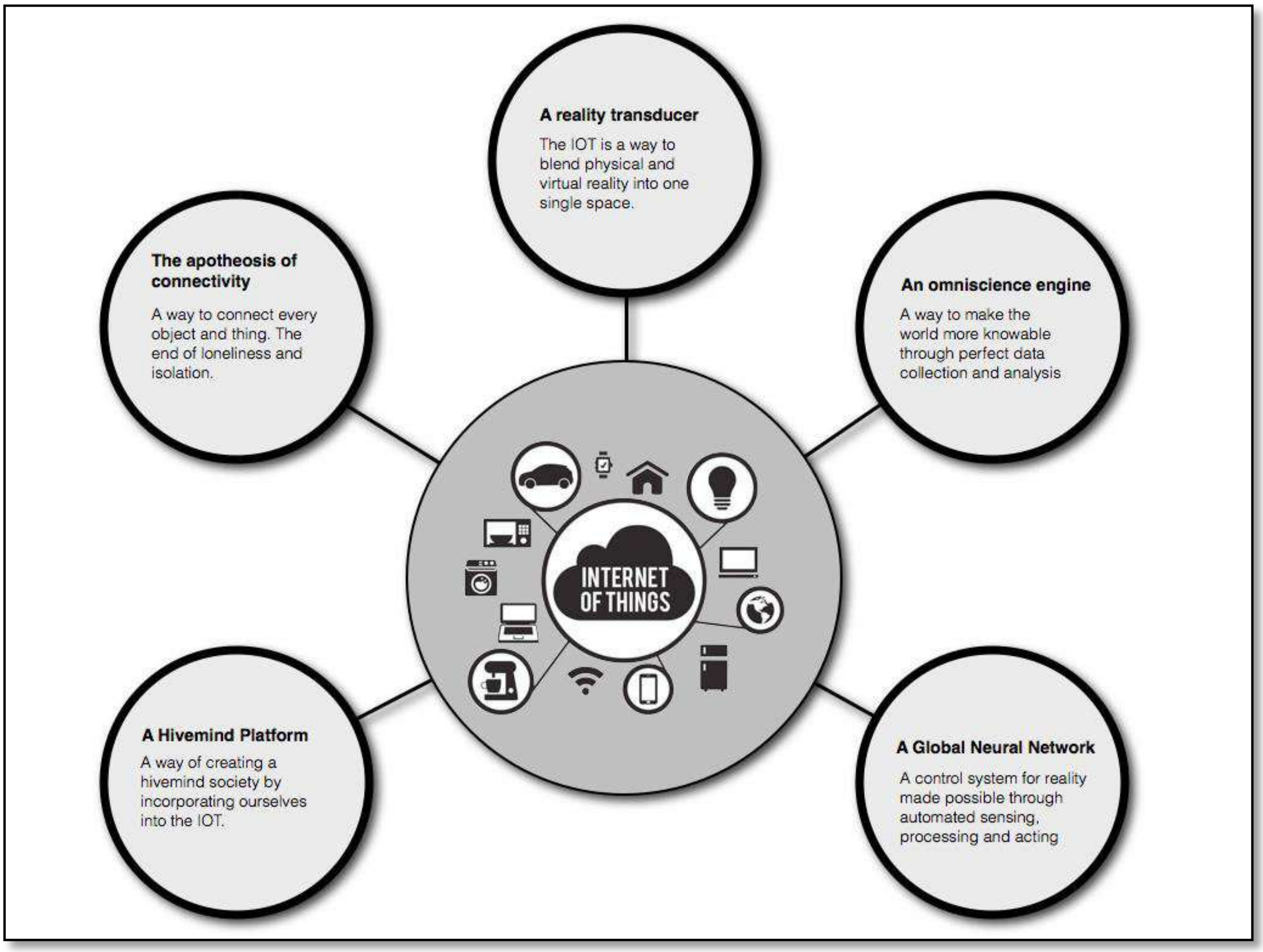
IoT is an enabler of three high-level objectives:

(1) DO MORE: Whether machines producing more (high throughput) because of less breakdowns or a weekend athlete burning more calories because she is able to keep her heart rate in the fat-burning zone, we are accomplishing more.

(2) HIGHER QUALITY: By monitoring environmental pollution, cities restrict automobile access into city center for better health outcomes over time.

(3) BETTER USER EXPERIENCE: In the near future, mass customization will allow me to find eyeglasses with perfect fit at low cost based on video inputs at connected additive manufacturing facilities.

Other five metaphors of IoT are below,


A reality transducer
The IOT is a way to blend physical and virtual reality into one single space.
The apotheosis of connectivity
A way to connect every object and thing. The end of loneliness and isolation.
An omniscience engine
A way to make the world more knowable through perfect data collection and analysis
INTERNET OF THINGS
A Hivemind Platform
A way of creating a hivemind society by incorporating ourselves into the IOT.
A Global Neural Network
A control system for reality made possible through automated sensing, processing and acting

## PRIVACY

**IoT security and privacy concerns**

Although IoT is rapidly growing, it still faces security and privacy issues:

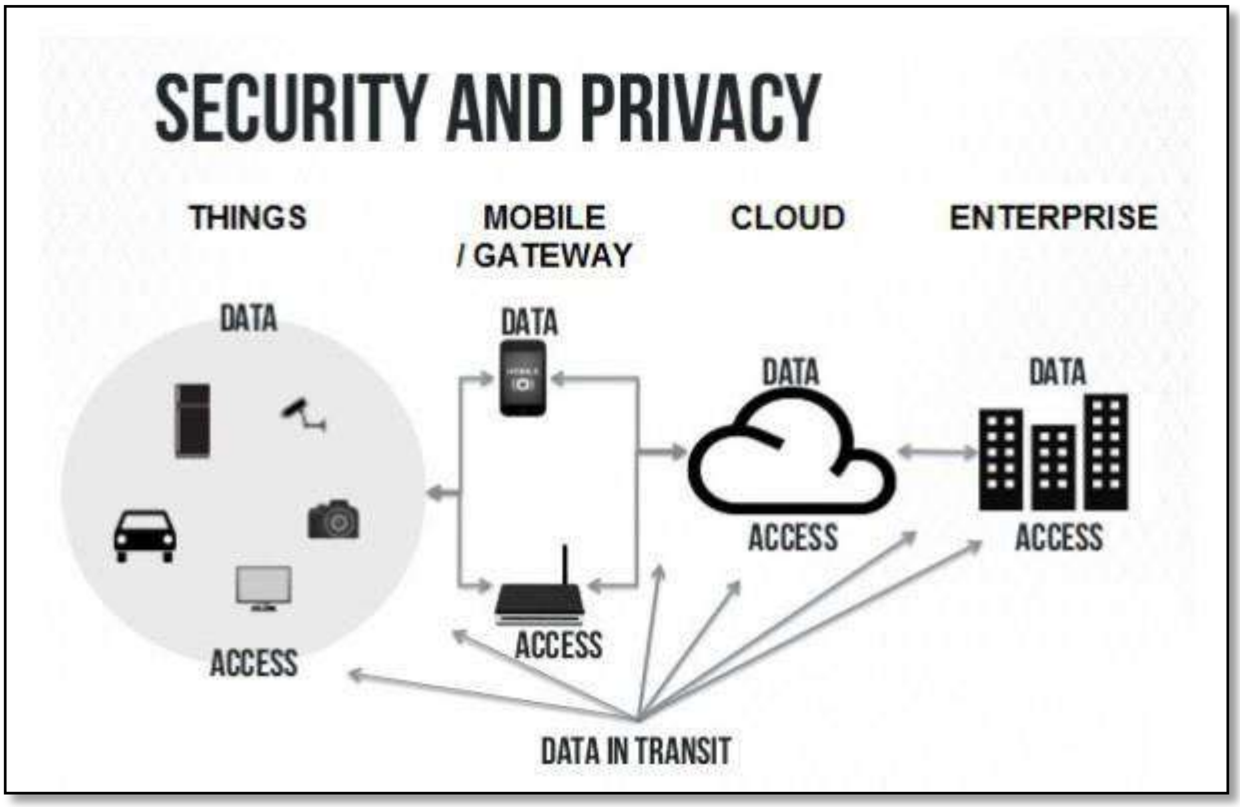
**Security Risks**

- IoT devices are connected to your desktop or laptop. Lack of security increases the risk of your personal information leaking while the data is collected and transmitted to the IoT device.
- IoT devices are connected with a consumer network. This network is also connected with other systems. So if the IoT device contains any security vulnerabilities, it can be harmful to the consumer's network. This vulnerability can attack other systems and damage them.
- Sometimes unauthorized people might exploit the security vulnerabilities to create risks to physical safety.

**Privacy Risks**

- In IoT, devices are interconnected with various hardware and software, so there are obvious chances of sensitive information leaking through unauthorized manipulation.
- All the devices are transmitting the user's personal information such as name, address, date of birth, health card information, credit card detail and much more without encryption.

Though there are security and privacy concerns with IoT, it adds values to our lives by allowing us to manage our daily routine tasks remotely and automatically, and more importantly, it is a game-changer for industries.



## WEB THINKING FOR CONNECTED DEVICES

The Internet of Things (IoT) is the network of connected devices (also known as “smart devices”) and other items embedded with electronics, software, sensors, and network connectivity which enable these devices to exchange data.

**IoT technology is expanding day by day, below are few common areas where IoT can be a good fit:**

- Smart cities
- Smart Agriculture
- Industrial Internet of things (or IIoT)
- Smart supply chain
- IoT in Healthcare

**Question Bank:**

1. Explain an overview about IoT.(10M)
2. Explicate design principles for connected devices. (10M)
3. Illustrate Internet of Thing. (5M)
4. Illuminate calm and ambient technology. (5M)
5. Elaborate web thinking for connected devices. (5M)
6. List the IoT application. (5M)
7. What are all the smart objects available in IoT? (5M)
8. Explain IoT with Architecture. (10M)

**PAPER NAME : INTERNET OF THINGS** **PAPER CODE : 17PCSE06**

## UNIT – II

## INTERNET PRINCIPLES

### INTERNET COMMUNICATIONS

Suppose that you wanted to send a message to the authors of this book, but you didn't have the postal address, and you didn't have any way to look up our phone number (because in this example you don't have the Internet).

You remember that we're from the UK, and London is the biggest city in the UK. So you send a postcard to your cousin Bob, who lives there.

Your cousin sees that the postcard is for some crazy hardware and technology people. So he puts the postcard in an envelope and drops it off at the London Hackspace because the guys there probably know what to do with it.

At the Hackspace, Jonty picks up the envelope and sees that it's for some people in Liverpool. Like all good Londoners, Jonty never goes anywhere to the north of Watford, but he remembers that Manchester is in the north too. So he calls up the Manchester Digital Laboratory (MadLab), opens the envelope to read the contents, and says, "Hey, I've got this message for Adrian and Hakim in Liverpool. Can you pass it on?"

The guys at MadLab ask whether anyone knows who we are, and it turns out that Hwa Young does. So the next time she comes to Liverpool, she delivers the postcard to us.

### IP

The preceding scenario describes how the Internet Protocol (IP) works. Data is sent from one machine to another in a packet, with a destination address and a source address in a standardised format (a "protocol"). Just like the original sender of the message in the example, the sending machine doesn't always know the best route to the destination in advance. Most of the time, the packets of data have to go through a number of intermediary machines, called routers, to reach their destination. The underlying networks aren't always the same: just as we used the phone, the postal service, and delivery by hand, so data packets can be sent over wired or wireless networks, through the phone system, or over satellite links.

In our example, a postcard was placed in an envelope before getting passed onwards. This happens with Internet packets, too. So, an IP packet is a block of data along with the same kind of information you would write on a physical envelope: the name and address of the server, and so on. But if an IP packet ever gets transmitted across your local wired network via an Ethernet cable—the cable that connects your home broadband router or your office local area network (LAN) to a desktop PC—then the whole packet will get bundled up into another type of envelope, an Ethernet Frame, which adds additional information about how to complete the last few steps of its journey to your computer.Of course, it's possible that your cousin Bob didn't know about the London Hackspace, and then maybe the message would have got stuck with him. You would have had no way to know whether it got there. This is how IP works. There is no guarantee, and you can send only what will fit in a single packet.

**TCP**

What if you wanted to send longer messages than fit on a postcard? Or wanted to make sure your messages got through?

What if everyone agreed that postcards written in green ink meant that we cared about whether they arrived. And that we would always number them, so if we wanted to send longer messages, we could. The person at the other end would be able to put the messages in order, even if they got delivered in the wrong order (maybe you were writing your letter over a number of days, and the day you passed the fifth one on to cousin Bob, he happened to visit Liverpool and passed on that postcard without relaying through London Hackspace or MadLab). We would send back postcard notifications that just told you which postcards we had received, so you could resend any that went missing.
That is basically how the Transmission Control Protocol (TCP) works. The simplest transport protocol on the Internet, TCP is built on top of the basic IP protocol and adds sequence numbers, acknowledgements, and retransmissions. This means that a message sent with TCP can be arbitrarily long and give the sender some assurance that it actually arrived at the destination intact.

Because the combination of TCP and IP is so useful, many services are built on it in turn, such as email and the HTTP protocol that transmits information across the World Wide Web.

**THE IP PROTOCOL SUITE (TCP/IP)**

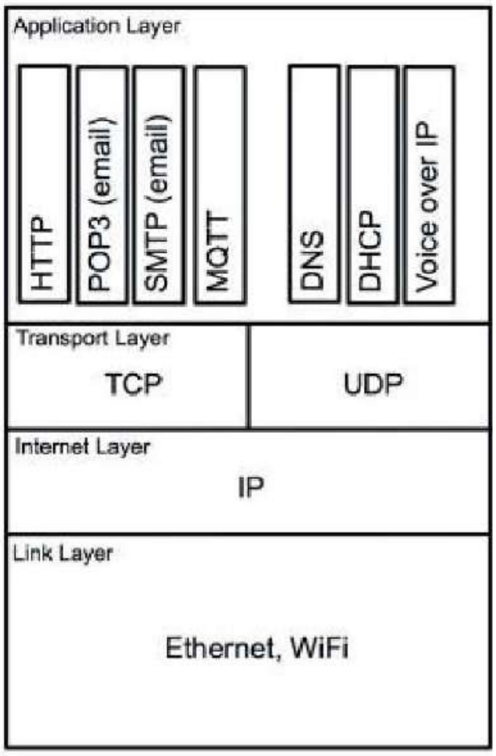
The combination of TCP and IP is so ubiquitous that we often refer simply to “TCP/IP” to describe a whole suite or stack of protocols layered on top of each other, each layer building on the capabilities of the one below.

- The low-level protocols at the link layer manage the transfer of bits of information across a network link. This could be by an Ethernet cable, by WiFi, or across a telephone network, or even by short-range radio standards such as IEEE 802.15.4 designed to carry data over the Personal Area Network (PAN), that is to say between devices carried by an individual.

- The Internet layer then sits on top of these various links and abstracts away the gory details in favour of a simple destination address.

- Then TCP, which lives in the transport layer, sits on top of IP and extends it with more sophisticated control of the messages passed.

- Finally, the application layer contains the protocols that deal with fetching web pages, sending emails, and Internet telephony. Of these, HTTP is the most ubiquitous for the web, and indeed for communication between Internet of Things devices.



The Internet Protocol suite.

**UDP**

As you can see, TCP is not the only protocol in the transport layer. Unlike TCP, but as with IP itself, in UDP each message may or may not arrive. No handshake or retransmission occurs, nor is there any delay to wait for messages in sequence. These limitations make TCP preferable for many of the tasks that Internet of Things devices will be used for.

The lack of overhead, however, makes UDP useful for applications such as streaming data, which can cope with minor errors but doesn't like delays. Voice over IP (VoIP)—computer-based telephony, such as Skype—is an example of this: missing one packet might cause a tiny glitch in the sound quality, but waiting for several packets to arrive in the right order could make the speech too jittery to be easy to understand. UDP is also the transport for some very important protocols which provide common, low-level functionality, such as DNS and DHCP, which relate to the discovery and resolution of devices on the network. We look at this topic in detail in the next section.

## IP ADDRESSES

We mentioned earlier that the Internet Protocol knows the addresses of the destination and source devices. But what does an "address" consist of? Here is a typical human (or in this case, hobbit) address:

- Bilbo Baggins
- "Bag End", Bagshot Row
- Hobbiton
- The Shire
- Middle Earth

In the world of low-level computer networking, however, numbers are much easier to deal with. So, IP addresses are numbers. In Internet Protocol version 4 (IPv4), almost 4.3 billion IP addresses are possible—4,294,967,296 to be precise, or 232. Though that is convenient for computers, it's tough for humans to read, so IP addresses are usually written as four 8-bit numbers separated by dots (from 0.0.0.0 to 255.255.255.255)—for example, 192.168.0.1 (which is often the address of your home router) or 8.8.8.8 (which is the address of one of Google's DNS servers).

This "dotted quad" is still exactly equivalent to the 32-bit number. As well as being simply easier for humans to remember, it is also easier to infer information about the address by grouping certain blocks of addresses together. For example,

| | | |
|---|---|---|
| 8.8.8.x | — | One of several IP ranges assigned to Google. |
| 192.168.x.x | — | A range assigned for private networks. Your home or office network router may well assign IP addresses in this range. |
| 10.x.x.x | — | Another private range. |

Every machine on the Internet has at least one IP address. That means every computer, every network-connected printer, every smartphone, and every Internet of Things device has one. If you already have a Raspberry Pi, an Arduino board, or any of the other microcontrollers described in Chapters 3 and 4, they will expect to get their own IP address, too. When you consider this fact, those 4 billion addresses suddenly look as if they might not be enough.

The private ranges such as 192.168.x.x offer one mitigation to this problem. Your home or office network might have only one publicly visible IP address. However, you could have all the IP addresses in the range 192.168.0.0 to 192.168.255.255 (2^16 = 65,536 addresses) assigned to distinct devices.

A better solution to this problem is the next generation of Internet Protocol, IPv6, which we look at later in this chapter.

## DNS

Although computers can easily handle 32-bit numbers, even formatted as dotted quads they are easy for most humans to forget. The Domain Name System (DNS) helps our feeble brains navigate the Internet. Domain names, such as the following, are familiar to us from the web, or perhaps from email or other services:

google.com
bbc.co.uk
wiley.com
arduino.cc

Each domain name has a top-level domain (TLD), like .com or.uk, which further subdivides into .co.uk and .gov.uk, and so on. This top-level domain knows where to find more information about the domains within it; for example, .com knows where to find google.com and wiley.com.

The domains then have information about where to direct calls to individual machines or services. For example, the DNS records for .google.com know where to point you for the following:

www.google.com
mail.google.com
calendar.google.com

The preceding examples are all instantly recognizable as website names, which is to say you could enter them into your web browser as, for example, http://www.google.com.

But DNS can also point to other services on the Internet—for example:

| | | |
|---|---|---|
| pop3.google.com | — | For receiving email from Gmail |
| smtp.google.com | — | For sending email to Gmail |
| ns1.google.com | — | The address of one of Google's many DNS servers |

Configuring DNS is a matter of changing just a few settings. Your registrar (the company that sells you your domain name) often has a control panel to change these settings. You might also run your own authoritative DNS server. The settings might contain entries like this one for roomofthings.com:

```
book A 80.68.93.60 3h
```

This entry means that the address book.roomofthings.com (which hosts the blog for this book) is served by that IP address and will be for the next three hours.

## STATIC IP ADDRESS ASSIGNMENT

How do you get assigned an IP address? If you have bought a server-hosting package from an Internet service provider (ISP), you might typically be given a single IP address. But the company itself has been given a block of addresses to assign. Historically, these were ranges of different sizes, typically separated into "classes" of 8 bits, 16 bits, or 24 bits:

| | | |
|---|---|---|
| Class A | — | From 0.x.x.x |
| Class B | — | From 128.0.x.x |
| Class C | — | From 192.0.0.x |

The class C ranges had a mere 8 bits (256 addresses) assigned to them, while the class A ranges had many more addresses and would therefore be given only to the very largest of Internet organisations. The rigid separation of address ranges into classes was not very efficient; every entity would want to keep enough spare addresses for future expansion, but this means that many addresses would remain unused. With the explosion of the number of devices connecting to the Internet (a theme throughout this chapter), the scheme has been superceded since 1993 by Classless Inter-Domain Routing (CIDR), which allows you to specify exactly how many bits of the address are fixed. (See RFCs 1518 and 1519, at http://tools.ietf.org/rfc/.) So, the class A addresses we mentioned above would be equivalent to 0.0.0.0/8, while a class C might be 208.215.179.0/24.

For example, you saw previously that Google had the range

8.8.8.x (which is equivalent to 8.8.8.0/24 in CIDR notation)

Google has chosen to give one of its public DNS servers the address

8.8.8.8

from this range, largely because this address is easy to remember.

In many cases, however, the system administrator simply assigns server numbers in order. The admininstrator makes a note of the addresses and updates DNS records and so on to point to these addresses. We call this kind of address static because once assigned it won't change again without human intervention.

Now consider your home network: every time you plug a desktop PC to your router, connect your laptop or phone to the wireless, or switch on your network-enabled printer, this device has to get an IP address (often in the range 192.168.0.0/16). You could assign an address sequentially yourself, but the typical person at home isn't a system administrator and may not keep thorough records. If your brother, who used to use the address 192.168.0.5 but hasn't been home for ages, comes back to find that your new laser printer now has that address, he won't be able to connect to the Internet.

## DYNAMIC IP ADDRESS ASSIGNMENT

Thankfully, we don't typically have to choose an IP address for every device we connect to a network. Instead, when you connect a laptop, a printer, or even a Twitter-following bubble machine, it can request an IP address from the network itself using the Dynamic Host Configuration Protocol (DHCP). When the device tries to connect, instead of checking its internal configuration for its address, it sends a message to the router asking for an address. The router assigns it an address. This is not a static IP address which belongs to the device indefinitely; rather, it is a temporary "lease" which is selected dynamically according to which addresses are currently available. If the router is rebooted, the lease expires, or the device is switched off, some other device may end up with that IP address.

This means that you can't simply point a DNS entry to a device using DHCP. In general, you can rely on the IP address probably being the same for a given work session, but you shouldn't hard-code the IP address anywhere that you might try to use it another time, when it might have changed.

Even the simplest computing devices such as the Arduino board, which we look at in Chapter 5, can use DHCP. Although the Arduino's Ethernet library allows you to configure a static IP address, you can also request one via DHCP. Using a static address may be fine for development (if you are the only person connected to it with that address), but for working in groups or preparing a device to be distributed to other people on arbitrary networks, you almost certainly want a dynamic IP address.

## IPv6

When IP was standardised, few could have predicted how quickly the 4.3 billion addresses that IPv4 allowed for would be allocated. The expected growth of the Internet of Things can only speed up this trend. If your mobile phone, watch, MP3 player, augmented reality sunglasses, and telehealth or sports-monitoring devices are all connected to the Internet, then you personally are carrying half a dozen IP addresses already. Perhaps you have a dedicated wallet server for micropayments? A personal web server that contains your contact details and blog? One or more webcams recording your day? Perhaps rather than a single health monitoring device, you have several distributed across your person, with sensors for temperature, heart rate, insulin levels, and any number of other stimuli.

At home you would start with all your electronic devices being connected. But beyond that, you might also have sensors at every door and window for security. More sensitive sound sensors to detect the presence of mice or beetles. Other sensors to check temperature, moisture, and airflow levels for efficiency. It is hard to predict what order of number of Internet connected devices a household might have in the near future. Tens? Hundreds? Thousands?

Enter IPv6, which uses 128-bit addresses, usually displayed to users as eight groups of four hexadecimal digits—for example, 2001:0db8:85a3:0042 :0000:8a2e:0370:7334. The address space (2^128) is so huge that you could assign the same number of addresses as the whole of IPv4 to every person on the planet and barely make a dent in it.

The new standard was discussed during the 1980s and finally released in 1996. In 2013, it is still less popular than IPv4. You can find many ways to work around the lack of public IP addresses using subnets, but there is a chicken-and-egg problem with getting people to use IPv6 without ISP support and vice versa. It was originally expected that mobile phones connected to the Internet (another huge growth area) would push this technology over the tipping point. In fact, mobile networks are increasingly using IPv6 internally to route traffic. Although this infrastructure is still invisible to the end user, it does mean that there is already a lot of use below the surface which is stacked up, waiting for a tipping point.

## IPv6 and Powering Devices

We can see that an explosion in the number of Internet of Things devices will almost certainly need IPv6 in the future. But we also have to consider the power consumption of all these devices. We know that we can regularly charge and maintain a small handful of devices. At any one moment, we might have a laptop, a tablet, a phone, a camera, and a music player plugged in to charge. The constant juggling of power sockets, chargers, and cables is feasible but fiddly. The requirements for large numbers of devices, however, are very different. The devices should be low power and very reliable, while still being capable of connecting to the Internet. Perhaps to accomplish this, these devices will team together in a mesh network. This is the vision of 6LoWPAN, an IETF working group proposing solutions for "IPv6 over Low power Wireless Personal Area Networks", using technologies such as IEEE 802.15.4. While a detailed discussion of 6LoWPAN and associated technologies is beyond the scope of this book, we do come back to many related issues, such as maximising battery life in Chapter 8 on embedded programming.

## Conclusion on IPv6

Although IPv6 is, or will be, big news, we do not go into further detail in this book. In 2013, you can find more libraries, more hardware, and more people that can support IPv4, and this is what will be most helpful when you are moving from prototype to production on an Internet of Things device. Even though we are getting close to the tipping point, existing IPv4 services will be able to migrate to IPv6 networks with minimal or possibly no rewriting.

If you are working on IPv6 network infrastructure or are an early adopter of 6LoWPAN, you will have specific knowledge requirements that are beyond the current scope of this book.

## MAC ADDRESSES

As well as an IP address, every network-connected device also has a MAC address, which is like the final address on a physical envelope in our analogy. It is used to differentiate different machines on the same physical network so that they can exchange packets. This relates to the lowest-level "link layer" of the TCP/IP stack. Though MAC addresses are globally unique, they don't typically get used outside of one Ethernet network (for example, beyond your home router). So, when an IP message is routed, it hops from node to node, and when it finally reaches a node which knows where the physical machine is, that node passes the message to the device associated with that MAC address.

**MAC stands for *Media Access Control*.** It is a 48-bit number, usually written as six groups of hexadecimal digits, separated by colons—for example:

01:23:45:67:89:ab

Most devices, such as your laptop, come with the MAC address burned into their Ethernet chips. Some chips, such as the Arduino Ethernet's WizNet, don't have a hard-coded MAC address, though. This is for production reasons: if the chips are mass produced, they are, of course, identical. So they can't, physically, contain a distinctive address. The address could be stored in the chip's firmware, but this would then require every chip to be built with custom code compiled in the firmware. Alternatively, one could provide a simple data chip which stores just the MAC address and have the WizNet chip read that. Obviously, most consumer devices use some similar process to ensure that the machine always starts up with the same unique MAC address. The Arduino board, as a low-cost prototyping platform for developers, doesn't bother with that nicety, to save time and cost. Yet it does come with a sticker with a MAC address printed on it. Although this might seem a bit odd, there is a good reason for it: that MAC address is reserved and therefore is guaranteed unique if you want to use it. For development purposes, you can simply choose a MAC address that is known not to exist in your network.

WizNet is a Korean manufacturer which specialises in networking chips for embedded devices. Many popular microcontrollers which we look at in Chapter 5 use these chips.

## TCP AND UDP PORTS

A messenger with a formal invitation for a wealthy family of the Italian Renaissance would go straight to the front entrance to deliver it. A grocer delivering a crate of the first artichokes of the season would go instead to a service entrance, where the crate could be taken quickly to the kitchen without getting in the way of the masters. The following engraving, by John Gilbert, is taken from Shakespeare's Romeo and Juliet. This reminds us that the house of the Capulets has at least one other entrance—on Juliet's balcony. If Romeo wants to see his beloved, that is the only way to go. If he climbs up the wrong balcony, he'll either wait outside (the nurse is fast asleep and can't hear his knocks) or get chased away by the angry father.

Similarly, when you send a TCP/IP message over the Internet, you have to send it to the right port. TCP ports, unlike entrances to the Capulet house, are referred to by numbers (from 0 to 65535).

**AN EXAMPLE: HTTP PORTS**

If your browser requests an HTTP page, it usually sends that request to port 80. The web server is “listening” to that port and therefore replies to it. If you send an HTTP message to a different port, one of several things will happen:

*Romeo and Juliet*, Act I, Scene 2, by John Gilbert, before 1873.
Public domain http://en.wikipedia.org/wiki/File:Scene_2.jpg

- Nothing is listening to that port, and the machine replies with an “RST” packet (a control sequence resetting the TCP/IP connection) to complain about this.
- Nothing is listening to that port, but the firewall lets the request simply hang instead of replying. The purpose of this (lack of) response is to discourage attackers from trying to find information about the machine by scanning every port. (Imagine Romeo knocking on the sleeping nurse’s window.)
- The client has decided that trying to send a message to that port is a bad idea and refuses to do it. Google Chrome does this for a fairly arbitrary list of “restricted ports”.
- The message arrives at a port that is expecting something other than an HTTP message. The server reads the client’s response, decides that it is garbage, and then terminates the connection (or, worse, does a nonsensical operation based on the message).

Ports 0–1023 are “well-known ports”, and only a system process or an administrator can connect to them.

Ports 1024–49151 are “registered”, so that common applications can have a usual port number. However, most services are able to bind any port number in this range.

The Internet Assigned Numbers Authority (IANA) is responsible for registering the numbers in these ranges. People can and do abuse them, especially in the range 1024–49151, but unless you know what you’re doing, you are better off using either the correct assigned port or (for an entirely custom application) a port above 49151.

You see custom port numbers if a machine has more than one web server; for example, in development you might have another server, bound to port 8080:

http://www.example.com:8080

Or if you are developing a website locally, you may be able to test it with a built-in test web server which connects to a free port. For example, Jekyll (the lightweight blog engine we are using for this book’s website) has a test server that runs on port 4000:

http://localhost:4000

The secure (encrypted) HTTPS usually runs on port 443. So these two URLs are equivalent:

https://www.example.com

https://www.example.com:443

**OTHER COMMON PORTS**

Even if you will rarely need a complete catalogue of all port numbers for services, you can rapidly start to memorize port numbers for the common services that you use daily. For example, you will very likely come across the following ports regularly:

` ▪ 80 HTTP

▪ 8080 HTTP (for testing servers)

▪ 443 HTTPS

▪ 22 SSH (Secure Shell)

▪ 23 Telnet

▪ 25 SMTP (outbound email)

▪ 110 POP3 (inbound email)

▪ 220 IMAP (inbound email)

All of these services are in fact application layer protocols.

**APPLICATION LAYER PROTOCOLS**

We have seen examples of protocols at the different layers of the TCP/IP stack, from the low-level communication across wired Ethernet, the low-level IP communication, and the TCP transport layer. Now we come to the highest layer of the stack, the application layer. This is the layer you are most likely to interact with while prototyping an Internet of Things project (and we look at this in greater detail in Chapter 7). It is useful here to pause and flesh out the definition of the word "protocol".

A protocol is a set of rules for communication between computers. It includes rules about how to initiate the conversation and what format the messages should be in. It determines what inputs are understood and what output is transmitted. It also specifies how the messages are sent and authenticated and how to handle (and maybe correct) errors caused by transmission.

Bearing this definition in mind, we are ready to look in more detail at some application layer protocols, starting with HTTP.

**HTTP**

The Internet is much more than just "the web", but inevitably web services carried over HTTP hold a large part of our attention when looking at the Internet of Things.

HTTP is, at its core, a simple protocol. The client requests a resource by sending a command to a URL, with some headers. We use the current version of HTTP, 1.1, in these examples. Let's try to get a simple document at http://book.roomofthings.com/hello.txt. You can see the result if you open the URL in your web browser.

A browser showing "Hello World!"

But let’s look at what the browser is actually sending to the server to do this. The basic structure of the request would look like this:

```
GET /hello.txt HTTP/1.1
Host: book.roomofthings.com
```

Notice how the message is written in plain text, in a human-readable way (this might sound obvious, but not all protocols are; the messages could be encoded into bytes in a binary protocol, for example).

We specified the GET method because we’re simply getting the page. We go into much more detail about the other methods in Chapter 7, “Prototyping Online Components”. We then tell the server which resource we want (/hello.txt) and what version of the protocol we’re using.

Then on the following lines, we write the headers, which give additional information about the request. The Host header is the only required header in HTTP 1.1. It is used to let a web server that serves multiple virtual hosts point the request to the right place.
Well-written clients, such as your web browser, pass other headers. For example, my browser sends the following request:

```
GET /hello.txt HTTP/1.1
Host: book.roomofthings.com
Accept: text/html,application/xhtml+xml,application/ xml;q=0.9,*/*;q=0.8
Accept-Charset: UTF-8,*;q=0.5
Accept-Encoding: gzip,deflate,sdch
Accept-Language :en-US,en;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
If-Modified-Since: Tue, 21 Aug 2012 21:41:47 GMT
If-None-Match: “8a25e-d-4c7cd7e3d1cc0”
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/537.1
(KHTML, like Gecko) Chrome/21.0.1180.77 Safari/537.1
```

The **Accept-** headers tell the server what kind of content the client is willing to receive and are part of “Content negotiation”. For example, if I had passed

```
Accept-Language: it,en-US,en;q=0.8
```

The server might agree to give me the Italian version of the site instead, reverting to English only if it doesn’t have that page in Italian.

The other fields give the server more information about the client (for statistics and for working around known bugs) and manage caching and so on.

Finally, the server sends back its response. We already saw what that looked like in the browser, but now let’s look at what the full request/response looks like if we speak the HTTP protocol directly. (Obviously, you rarely have to do this in real life. Even if you are programming an Internet of Things device, you usually have access to code libraries that make the request, and reading of the response, easier.)

## HTTPS: ENCRYPTED HTTP

We have seen how the request and response are created in a simple text format. If someone eavesdropped your connection (easy to do with tools such as Wireshark if you have access to the network at either end), that person can easily read the conversation. In fact, it isn't the format of the protocol that is the problem: even if the conversation happened in binary, an attacker could write a tool to translate the format into something readable. Rather, the problem is that the conversation isn't encrypted.

The HTTPS protocol is actually just a mix-up of plain old HTTP over the Secure Socket Layer (SSL) protocol. An HTTPS server listens to a different port (usually 443) and on connection sets up a secure, encrypted connection with the client (using some fascinating mathematics and clever tricks such as the "Diffie–Hellman key exchange"). When that's established, both sides just speak HTTP to each other as before!

## OTHER APPLICATION LAYER PROTOCOLS

All protocols work in a roughly similar way. Some cases involve more than just a two-way request and response. For example, when sending email using SMTP, you first need to do the "HELO handshake" where the client introduces itself with a cheery "hello" (SMTP commands are all four letters long, so it actually says "HELO") and receives a response like "250 Hello example. org pleased to meet you!" In all cases, it is worth spending a little time researching the protocol on Google and Wikipedia to understand in overview how it works. You can usually find a library that abstracts the details of the communication process, and we recommend using that wherever possible. Bad implementations of network protocols will create problems for you and the servers you connect to and may result in bugs or your clients getting banned from useful services. So, it is generally better to use a well-written, well-debugged implementation that is used by many other developers. In general, the only valid reasons for you, the programmer, to ever speak to any application layer protocol directly (that is, without using a library) are

- There is no implementation of the protocol for your platform (or the implementation is inefficient, incomplete, or broken).
- You want to try implementing it from scratch, for fun.
- You are testing, or learning, and want to make a particular request easily.

## THINKING ABOUT PROTOTYPING

NOW THAT WE'VE looked at the principles of design and the fundamentals of Internet communications, we hope you are itching to create an Internet of Things device! It's possible that you want a single device that is Just For You. But perhaps you have a fantastic idea and are planning to churn out millions of the products. In both cases, the most sensible approach is to start by making one Thing first: a prototype.

Making a prototype first has many benefits. You will inevitably come across problems in your design that you need to change and iterate. Doing this with a single object is trivial compared to modifying hundreds or thousands of products. With the Internet of Things, we are always looking at building three things in parallel: the physical Thing; the electronics to make the Thing smart; and the Internet service that we'll connect to. The last of these is relatively cheap and easy to change. You cannot change the physical object and its silicon controller unless you recall every item.

The prototype, therefore, is optimized for ease and speed of development and also the ability to change and modify it. Many Internet of Things projects start with a prototyping microcontroller, connected by wires to components on a prototyping board, such as a "breadboard", and housed in some kind of container (perhaps an old tin or a laser-cut box). This prototype is relatively inexpensive, but you will most likely end up with something that is serviceable rather than polished and that will cost more than someone would be willing to pay for it in a shop.

At the end of this stage, you'll have an object that works. It may be useful for you already. It may be a talking point to show your friends. And if you are planning to move to production, it's a demonstrable product that you can use to convince yourself, your business partners, and your investors that your idea has legs and is worth trying to sell.

Finally, the process of manufacture will iron out issues of scaling up and polish. You might substitute prototyping microcontrollers and wires with smaller chips on a printed circuit board (PCB), and pieces improvised out of 3D-printed plastic with ones commercially injection-moulded in their thousands. The final product will be cheaper per unit and more professional, but will be much more expensive to change.

## SKETCHING

There is a good chance that the first step you'll take when working on your prototype will be to jot down some ideas or draw out some design ideas with pen and paper. That is an important first step in exploring your idea and one we'd like to extend beyond the strict definition to also include sketching in hardware and software.

What we mean by that is the process of exploring the problem space: iterating through different approaches and ideas to work out what works and what doesn't. The focus isn't on fidelity of the prototype but rather on the ease and speed with which you can try things out.

For the physical design, that could mean digging out your childhood LEGO collection to prototype the mix of cogs and three-dimensional forms, or maybe attacking some foamcore or cardboard with a craft knife. We examine such techniques in more detail in Chapter 6, "Prototyping the Physical Design".

To show how you might approach "sketching" for the electronics and software, an example will help.

The Internet of Things design firm BERG invited me (Adrian) along to their inaugural Little Printer hackday in June 2012. They filled their office with a bunch of interesting techies and creatives and tasked them with seeing what they could do, in a day, with BERG's (at the time) soon-to-be-released cute Internet-connected diminutive printer. (For more on the Little Printer, see the case study in Chapter 10, "Moving to Manufacture".)

Most of the attendees focused on creating new publications for the Little Printer—a task that meant writing server code to wrangle data (working with the Google Calendar API, spotting meteors passing overhead, and so on) into shape and experimenting with ways of displaying that on a narrow strip of receipt paper. I (Adrian) decided that having a connected device as the output for the system wasn't enough and spent the day prototyping a custom-hardware input device, too. Called the Printernet Fridge, it was a nod to the age-old Internet of Things cliché, the Internet fridge—an exercise in seeing what a semi-automated shopping list would be like.

From the design constraints (mostly how the Little Printer publishing system works but also limited by the hardware that I had thrown into my bag to take to the event), it was clear early on that the problem could be broken into three broad areas: the graphic design of the printed publication; the physical hardware to easily add items to the shopping list; and some server software to tie the rest of the system together. Breaking the problem into these three parts meant that, initially at least, each could be addressed separately.

The first step was to pull together the scaffolding of the server software, which other parts of the system could be built on as they were developed.

As this was a prototype, rather than a system to be deployed to thousands of users, I used the simple framework Sinatra. Sinatra is a way to quickly build web services, much like the Dancer framework which we cover in Chapter 7, “Prototyping Online Components”, but using the Ruby programming language.

Given that the prototype was going to be demonstrated only to a handful of people and didn’t need to extend beyond a single user, I didn’t waste any time including a login system for multiple users or setting up any security to encrypt the requests to and from the service. That provided enough infrastructure to interact with the Little Printer publication system and to allow the addition of API hooks for the input device to call when it was ready. I created a crude placeholder image for the publication’s icon and dashed off the description text with a simple “prints shopping lists”, to let the focus stay on the server software rather than get sidelined into the design and polish. Similarly, the publication itself was just the bare minimum static text which was delivered whenever the publication was requested.

At this point it was possible to subscribe to the publication and have it print a set, one-line document on a Little Printer. Collaborating with one of the designers at the hackday, we iterated through the look and layout of the shopping list by tweaking the HTML, CSS, and images in the Sinatra app that made up the publication.

The content was still static—you would always get a list asking for two bottles of milk and some fresh orange juice—but the header image and text were refined and decisions made about how to convey the information to the user. The simple bullets for each list item were replaced with numbers of each item required, which then moved to the end of the list item so the bullet could be reinstated but as an empty check box to allow you to tick items off as you found them in the shop.
All it needed now was some live data. The workflow for that needed to be as easy as possible; it shouldn’t be a chore to add items. However, this wasn’t to be some seamless vision of the future where the fridge would order things for you. (The ultimate agency of the user is important.) It is a tool rather than an autonomous decision maker.

With only an Arduino Ethernet board in my bag, the decision over hardware platform was already made. For a production unit, the Ethernet cable would complicate installation, and encrypting the data being sent would stretch the board’s limits; however, for a quick prototype, the ease of wiring up different circuits and changing the onboard code easily outweighed the longer-term disadvantages.

Once a basic pushbutton was wired up to the Arduino, I could write some simple code to trigger whenever the button was pressed. The first step merely output some text to the serial port, which could be monitored immediately over the USB cable on my laptop.

Now that the basic physical interaction was working, the next step was to hook it into the web service. That required me to change the software on both the Arduino and the web service. I added a new API call to the Sinatra app which allows new items to be added to the list and tested it with a web browser (as that gives an easier view of what is happening if things go wrong).

For a more complex project, I would have chosen Rails over Sinatra as the web framework. Doing so would have let me pull in the RailsAdmin module to check things like whether the item had been added to the database correctly. Such helper modules let you focus on sketching out one feature, without having to divert to building chunks of additional infrastructure before you have decided that the feature is going to stay. In this case, I could easily switch the static shopping list over to use the live data, which served as both feature development and debugging aid.

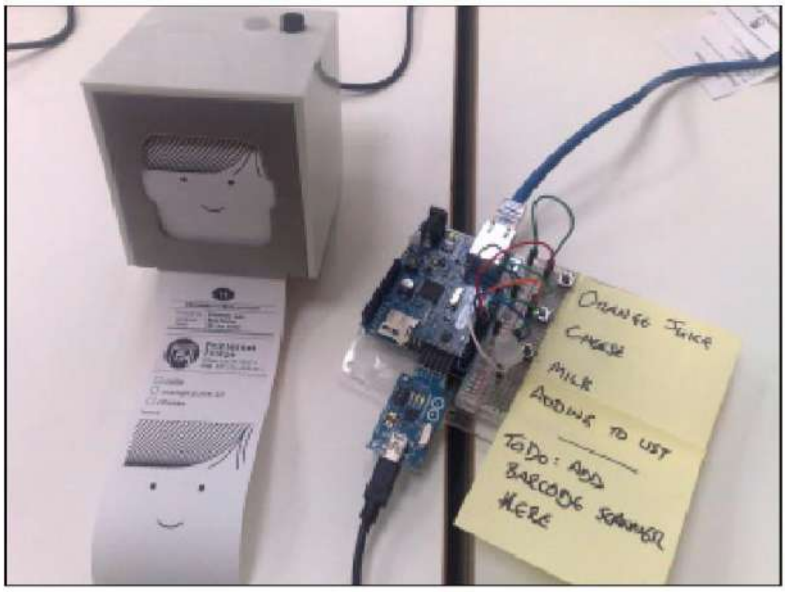
Then I pulled in some sample code to make a web request from the Arduino and modified it to call the newly written API on the web service. With that written, it was possible to push a button and have “milk” added to your shopping list. Success!

The next step was to add more buttons, as only being able to order more milk isn’t the most useful of shopping list applications. Two more buttons were added and connected up to allow “cheese” and “orange juice” to also be ordered.

A quick round of user testing (or showing it off to fellow hackday attendees, as it is also known in this case) highlighted a problem with the design as it stood. Although recording a new selection to the server took less than a second, it was still long enough for subsequent button presses to be missed if you were running through them quickly.

Given that the end of the day was looming large, and as I would be able to talk around the issue in the demonstration, I chose expediency over the larger amount of coding required to decouple the user interface of buttons from the network communication. A “busy” LED was added to the breadboard and illuminated whenever the Arduino was talking to the network.

The work on the hardware and software left little time for developing a case for the input device, which is just as well because I hadn’t room for any construction materials in my bag. Improvising with some sticky notes at least made the interface more self-explanatory. It also allowed room to hint at ways in which the prototype could be further extended, with “Add barcode scanner here” written on a note below the buttons. That was a result of further thinking through how the device might work in practice. You would want a number of buttons, with a web interface to let you reconfigure them for your set of non-packaged fridge goods, and then a barcode reader would allow scanning of anything that was in packaging.



The finished prototype of the Printernet Fridge.

**FAMILIARITY**

Another option to consider is familiarity. If you can already program like a whiz in Python, for example, maybe picking a platform such as Raspberry Pi, which lets you write the code in a language you already know, would be better than having to learn Arduino from scratch.

The same applies to the server software, obviously. When creating the Printernet Fridge prototype, Adrian hadn’t used Sinatra before but chose it because he was looking for a simple web framework and was already familiar with Ruby from writing a number of Ruby on Rails applications in the past. And if you’re already adept at fashioning sheets of foamcore into threedimensional structures, we’re not going to argue that you should ignore that expertise in favour of learning all about laser cutting.

## COSTS VERSUS EASE OF PROTOTYPING

Although familiarity with a platform may be attractive in terms of ease of prototyping, it is also worth considering the relationship between the costs (of prototyping and mass producing) of a platform against the development effort that the platform demands. This trade-off is not hard and fast, but it is beneficial if you can choose a prototyping platform in a performance/ capability bracket similar to a final production solution. That way, you will be less likely to encounter any surprises over the cost, or even the wholesale viability of your project, down the line.

For example, the cheapest possible way of creating an electronic device might currently be an AVR microcontroller chip, which you can purchase from a component supplier for about £3. This amount is just for the chip, so you would have to sweat the details of how to connect the pins to other components and how to flash the chip with new code. For many people, this platform would not be viable for an initial prototype.

Stepping upwards to the approximately £20 mark, you could look at an Arduino or similar. It would have exactly the same chip, but it would be laid out on a board with labelled headers to help you wire up components more easily, have a USB port where you could plug in a computer, and have a well-supported IDE to help make programming it easier. But, of course, you are still programming in C++, for reasons of performance and memory.

For more money again, approximately £30, you could look at the BeagleBone, which runs Linux and has enough processing power and RAM to be able to run a high-level programming language: libraries are provided within the concurrent programming toolkit Node.js for JavaScript to manipulate the input/output pins of the board.

If you choose not to use an embedded platform, you could think about using a smartphone instead. Smartphones might cost about £300, and although they are a very different beast, they have many of the same features that make the cheaper platforms attractive: connection to the Internet (usually by wireless or 3G phone connection rather than Ethernet), input capabilities (touchscreen, button presses, camera, rather than electronics components), and output capabilities (sound, screen display, vibration). You can often program them in a choice of languages of high or low level, from Objective C and Java, to Python or HTML and JavaScript.

Finally, a common or garden PC might be an option for a prototype. These PCs cost from £100 to £1000 and again have a host of Internet connection and I/O possibilities. You can program them in whatever language you already know how to use. Most importantly, you probably already have one lying around.

For the first prototype, the cost is probably not the most important issue: the smartphone or computer options are particularly convenient if you already have one available, at which point they are effectively zero-cost. Although prototyping a “thing” using a piece of general computing equipment might seem like a sideways step, depending on your circumstances, it may be exactly the right thing to do to show whether the concept works and get people interested in the project, to collaborate on it, or to fund it.

At this stage, you can readily argue that doing the easiest thing that could possibly work is entirely sensible. The most powerful platform that you can afford might make sense for now.
Of course, if your device has physical interactions (blowing bubbles, turning a clock’s hands, taking input from a dial), you will find that a PC is not optimized for this kind of work. It doesn’t expose GPIO pins (although people have previously kludged this using parallel ports). An electronics prototyping board,

unsurprisingly, is better suited to this kind of work. We come back to combining both of these options shortly.

An important factor to be aware of is that the hardware and programming choices you make will depend on your skill set, which leads us to the obvious criticism of the idea of "ease of prototyping", namely "ease... for whom?"

For many beginners to hardware development, the Arduino toolkit is a surprisingly good choice. Yes, the input/output choices are basic and require an ability to follow wiring diagrams and, ideally, a basic knowledge of electronics. Yet the interaction from a programming point of view is essentially simple—writing and reading values to and from the GPIO pins. Yes, the language is C++, which in the early twenty-first century is few people's idea of the best language for beginners. Yet the Arduino toolkit abstracts the calls you make into a setup() function and a loop() function. Even more importantly, the IDE pushes the compiled code onto the device where it just runs, automatically, until you unplug it. The lack of capabilities of the board presents an advantage in the fact that the interaction with it is also streamlined.

## PROTOTYPES AND PRODUCTION

Although ease of prototyping is a major factor, perhaps the biggest obstacle to getting a project started—scaling up to building more than one device, perhaps many thousands of them—brings a whole new set of challenges and questions.

## CHANGING EMBEDDED PLATFORM

When you scale up, you may well have to think about moving to a different platform, for cost or size reasons. If you've started with a free-form, powerful programming platform, you may find that porting the code to a more restricted, cheaper, and smaller device will bring many challenges. This issue is something to be aware of. If the first prototype you built on a PC, iPhone, BeagleBone, or whatever has helped you get investment or collaborators, you may be well placed to go about replicating that compelling functionality on your final target.

Of course, if you've used a constrained platform in prototyping, you may find that you have to make choices and limitations in your code. Dynamic memory allocation on the 2K that the Arduino provides may not be especially efficient, so how should that make you think about using strings or complex data structures? If you port to a more powerful platform, you may be able to rewrite your code in a more modern, high-level way or simply take advantage of faster processor speed and more RAM. But will the new platform have the same I/O capabilities? And you have to consider the ramping-up time to learn new technologies and languages.

In practice, you will often find that you don't need to change platforms. Instead, you might look at, for example, replacing an Arduino prototyping microcontroller with an AVR chip (the same chip that powers the Arduino) and just those components that you actually need, connected on a custom PCB. We look at this issue in much more detail in Chapter 10.

## PHYSICAL PROTOTYPES AND MASS PERSONALISATION

Chances are that the production techniques that you use for the physical side of your device won't translate directly to mass production. However, while the technique might change—injection moulding in place of 3D printing, for example—in most cases, it won't change what is possible.

An aspect that may be of interest is in the way that digital fabrication tools can allow each item to be slightly different, letting you personalize each device in some way. There are challenges in scaling this to production, as you will need to keep producing the changeable parts in quantities of one, but mass personalization, as the approach is called, means you can offer something unique with the accompanying potential to charge a premium.

## CLIMBING INTO THE CLOUD

The server software is the easiest component to take from prototype into production. As we saw earlier, it might involve switching from a basic web framework to something more involved (particularly if you need to add user accounts and the like), but you will be able to find an equivalent for whichever language you have chosen. That means most of the business logic will move across with minimal changes. Beyond that, scaling up in the early days will involve buying a more powerful server. If you are running on a cloud computing platform, such as Amazon Web Services, you can even have the service dynamically expand and contract, as demand dictates.

## OPEN SOURCE VERSUS CLOSED SOURCE

If you're so minded, you could spend a lifetime arguing about the definitions of "closed" and "open" source, and some people have, in fact, made a career out of it. Broadly, we're looking at two issues:

- Your assertion, as the creator, of your Intellectual Property rights
- Your users' rights to freely tinker with your creation

We imagine many of this book's readers will be creative in some sense, perhaps tinkerers, inventors, programmers, or designers. As a creative person, you may be torn between your own desire to learn how things work and modify and re-use them and the worry that if other people were to use that right on your own design/invention/software, you might not get the recognition and earnings that you expect from it.

In fact, this tension between the closed and open approaches is rather interesting, especially when applied to a mix of software and hardware, as we find with Internet of Things devices. While many may already have made up their minds, in one or the other direction, we suggest at least thinking about how you can use both approaches in your project.

### WHY CLOSED?

Asserting Intellectual Property rights is often the default approach, especially for larger companies. If you declared copyright on some source code or a design, someone who wants to market the same project cannot do so by simply reading your instructions and following them. That person would have to instead reverse-engineer the functionality of the hardware and software. In addition, simply copying the design slavishly would also infringe copyright. You might also be able to protect distinctive elements of the visual design with trademarks and of the software and hardware with patents.

Although getting good legal information on what to protect and how best to enforce those rights is hard and time-consuming, larger companies may well be geared up to take this route. If you are developing an Internet of Things device in such a context, working within the culture of the company may simply be easier, unless you are willing to try to persuade your management, marketing, and legal teams that they should try something different.

If you're working on your own or in a small company, you might simply trademark your distinctive brand and rely on copyright to protect everything else. Note that starting a project as closed source doesn't prevent you from later releasing it as open source (whereas after you've licensed something as open source, you can't simply revoke that license).

You may have a strong emotional feeling about your Intellectual Property rights: especially if your creativity is what keeps you and your loved ones fed, this is entirely understandable. But it's worth bearing in mind that, as always, there is a trade-off between how much the rights actually help towards this important goal and what the benefits of being more open are.

**WHY OPEN?**

In the open source model, you release the sources that you use to create the project to the whole world. You might publish the software code to GitHub (http://github.com), the electronic schematics using Fritzing (http:// fritzing.org) or SolderPad (http://solderpad.com), and the design of the housing/shell to Thingiverse (http://www.thingiverse.com).

If you're not used to this practice, it might seem crazy: why would you give away something that you care about, that you're working hard to accomplish? There are several reasons to give away your work:

- You may gain positive comments from people who liked it.
- It acts as a public showcase of your work, which may affect your reputation and lead to new opportunities.
- People who used your work may suggest or implement features or fix bugs.
- By generating early interest in your project, you may get support and mindshare of a quality that it would be hard to pay for.

Of course, this is also a gift economy: you can use other people's free and open source contributions within your own project. Forums and chat channels exist all over the Internet, with people more or less freely discussing their projects because doing so helps with one or more of the benefits mentioned here.
If you're simply "scratching an itch" with a project, releasing it as open source may be the best thing you could do with it. A few words of encouragement from someone who liked your design and your blog post about it may be invaluable to get you moving when you have a tricky moment on it. A bug fix from someone who tried using your code in a way you had never thought of may save you hours of unpleasant debugging later. And if you're very lucky, you might become known as "that bubble machine guy" or get invited to conferences to talk about your LED circuit.

If you have a serious work project, you may still find that open source is the right decision, at least for some of your work.

## Disadvantages of Open Source

The obvious disadvantage of open source—"but people will steal my idea!"—may, in fact, be less of a problem than you might think. In general, if you talk to people about an idea, it's hard enough to get them to listen because they are waiting to tell you about their great idea (the selfish cads). If people do use your open source contribution, they will most likely be using it in a way that interests them. The universe of ideas is still, fortunately, very large.

However, deciding to release as open source may take more resources. As the saying goes: the shoemaker's children go barefoot. If you're designing for other people, you have to make something of a high standard, but for yourself, you often might be tempted to cut corners. When you have a working prototype, this should be a moment of celebration. Then having to go back and fix everything so that you can release it in a form that doesn't make you ashamed will take time and resources.

Of course, the right way to handle this process would be to start pushing everything to an open repository immediately and develop in public. This is much more the “open source way”. It may take some time to get used to but may work for you.

After you release something as open source, you may still have a perceived duty to maintain and support it, or at least to answer questions about it via email, forums, and chatrooms. Although you may not have paying customers, your users are a community that you may want to maintain. It is true that, if you have volunteered your work and time, you are entirely responsible for choosing to limit that whenever you want. But abandoning something before you’ve built up a community around it to pass the reins to cannot be classed as a successful open source project.

Being a Good Citizen The idea that there is a “true way” to do open source is worth thinking about. There is in some way a cachet to “doing open source” that may be worth having. Developers may be attracted to your project on that basis. If you’re courting this goodwill, it’s important to make sure that you do deserve it. If you say you have an open platform, releasing only a few libraries, months afterwards, with no documentation or documentation of poor quality could be considered rude. Also, your open source work should make some attempt to play with other open platforms. Making assumptions that lock in the project to a device you control, for example, would be fine for a driver library but isn’t great for an allegedly open project.

In some ways, being a good citizen is a consideration to counterbalance the advantages of the gift economy idea. But, of course, it is natural that any economy has its rules of citizenship!
Open Source as a Competitive Advantage Although you might be tempted to be very misty-eyed about open source as a community of good citizens and a gift economy, it’s important to understand the possibility of using it to competitive advantage.

First, using open source work is often a no-risk way of getting software that has been tested, improved, and debugged by many eyes. As long as it isn’t licensed with an extreme viral licence (such as the AGPL), you really have no reason not to use such work, even in a closed source project. Sure, you could build your own microcontroller from parts and write your own library to control servo motors, your own HTTP stack, and a web framework. Or you could use an Arduino, the Arduino servo libraries and Ethernet stack, and Ruby on Rails, for example. Commercial equivalents may be available for all these examples, but then you have to factor in the cost and rely on a single company’s support forums instead of all the information available on the Internet.

Second, using open source aggressively gives your product the chance to gain mindshare. In this book we talk a lot about the Arduino—as you have seen in this chapter; one could easily argue that it isn’t the most powerful platform ever and will surely be improved. It scores many points on grounds of cost but even more so on mindshare. The design is open; therefore, many other companies have produced clones of the board or components such as shields that are compatible with it.
This has led to amusing things such as the Arduino header layout “bug” (http://forum.arduino.cc/index. php/topic,22737.0.html#subject_171839), which is the result of a design mistake that has nevertheless been replicated by other manufacturers to target the same community.

If an open source project is good enough and gets word out quickly and appealingly, it can much more easily gain the goodwill and enthusiasm to become a platform. The “geek” community often choose a product because, rather than being a commercial “black box”, it, for example, exposes a Linux shell or can communicate using an open protocol such as XML. This community can be your biggestally.

Open Source as a Strategic Weapon One step further in the idea of open source used aggressively is the idea of businesses using open source strategically to further their interests (and undermine their competitors).

In “Commoditizing your complements” (http://www.joelonsoftware. com/articles/StrategyLetterV.html), software entrepreneur Joel Spolsky argues that many companies that invest heavily in open source projects are doing just that. In economics, the concept of complements defines products and services that are bought in conjunction with your product—for example, DVDs and DVD players.
If the price of one of those goods goes down, then demand for both goods is likely to rise. Companies can therefore use improvements in open source versions of complementary products to increase demand for their products. If you manufacture microcontrollers, for example, then improving the open source software frameworks that run on the microcontrollers can help you sell more chips.

## MIXING OPEN AND CLOSED SOURCE

We’ve discussed open sourcing many of your libraries and keeping your core business closed. While many businesses can exist as purely one or the other, you shouldn’t discount having both coexist. As long as you don’t make unfounded assertions about how much you use open software, it’s still possible to be a “good citizen” who contributes back to some projects whether by contributing work or simply by helping others in forums while also gaining many of the advantages of open source.
While both of us tend to be keen on the idea of open source, it’s also true that not all our work is open source. We have undertaken some for commercial clients who wanted to retain IP. Some of the work was simply not polished enough to be worth the extra effort to make into a viable open release.
Adrian’s project Bubblino has a mix of licences:

- Arduino code is open source.
- Schematics are available but not especially well advertised.
- Server code is closed source.

The server code was partly kept closed source because some details on the configuration of the Internet of Things device were possibly part of the commercial advantage.

## CLOSED SOURCE FOR MASS MARKET PROJECTS

One edge case for preferring closed source when choosing a licence may be when you can realistically expect that a project might be not just successful but huge, that is, a mass market commodity. While “the community” of open source users is a great ally when you are growing a platform by word of mouth, if you could get an existing supply and distribution chain on your side, the advantage of being first to market and doing so cheaper may well be the most important thing.

Let’s consider Nest, an intelligent thermostat: the area of smart energy metering and control is one in which many people are experimenting. The moment that an international power company chooses to roll out power monitors to all its customers, such a project would become instantaneously mass market. This would make it a very tempting proposition to copy, if you are a highly skilled, highly geared-up manufacturer in China, for example. If you also have the schematics and full source code, you can even skip the investment required to reverse-engineer the product.

The costs and effort required in moving to mass scale show how, for a physical device, the importance of supply chain can affect other considerations. In 2001, Paul Graham spoke compellingly about how the choice of programming language (in his case, Lisp) could leave competitors in the dirt because all of his competitors chose alternative languages with much slower speed of development (www.paulgraham.com/avg.html). Of course, the key factor wasn’t so much about development platform as time to market versus your competitor’s time to market. The tension between open and closed source informs this as well.

**TAPPING INTO THE COMMUNITY**

We talked about the “community” in the previous section, but it would be disingenuous to pretend that this is exclusively a feature of open source projects.

While thinking about which platform you want to build for, having a community to tap into may be vital or at least useful. Again, this is a major reason for our current support of the Arduino platform. If you have a problem with a component or a library, or a question about how to do something (for example, controlling a servo motor with a potentiometer dial), you could simply do a Google search on the words “arduino servo potentiometer” and find a YouTube video, a blog post, or some code.

Many other cute platforms, such as the Chumby Hacker Board, do have communities of aficionados, but perhaps smaller ones. If you are doing something more obscure or need more detailed technical assistance, finding someone who has already done exactly that thing may be difficult.

Mindshare may be important as you scale up, too—for example, if you want confidence that you can hire people with skills in the platform you’ve chosen. This issue may be less important for a small, focused team which has a lot of expertise in a new or obscure platform but still may be a consideration.

When you are an inexperienced maker, using a platform in which other people can mentor you is invaluable. If you have a local meeting for makers, such as Maker Night Liverpool, or equivalents in hackspaces around the world, you will very often find someone who is willing to take you through the basics in Arduino or another similar system. Perhaps that person is an expert on it or has simply gone through the basics (getting an LED flashing or playing “Mary had a little lamb” with a piezo speaker) at the last meeting. These meetings can be invaluable for both student and mentor.

Local meetings are also a great way to discuss your own project and learn about others. While to discuss your project is in some way being “open” about it, you are at all times in control of how much you say and whom you say it to. If you’re not already open source minded, this approach can be much less intimidating than releasing your clever idea to the whole Internet at once.

The perceived danger of sharing an idea or an implementation or a question with other people is looking like an idiot in public. While many parts of many Internet communities are much more sympathetic to this fear than one might expect, the mask of anonymity on the Internet can seem to permit people to be less supportive or simply more rude than you might hope for. In general, face-to-face meetings at a hackspace may well be a friendlier and more supportive way to dip your toes into the idea of a “community” of Internet of Things makers.

One reason to be in touch with a (local or Internet) community of makers is that we are, in interaction designer and BERG hardware engineer Andy Huntington’s words, at the stage of the “Geocities of things”—that is, at the frontier of the Internet of Things, just as Geocities was at the frontier of making websites and blogging. Sure, the design of some of the things may be clunky, the things might be pointless, and a lot of people may simply be doing something that they saw someone else do before. But from this outpouring of creativity will come the next generation of successful businesses and projects that actually change the world. This is a fascinating time to be getting involved in the Internet of Things.

**Question Bank:**

1. Explain internet communication overview.(10M)
2. Explicate prototyping in detail.(10M)
3. Describe MAC.(5M)
4. Write a note on TCP/IP, UDP. (5M)
5. Elaborate DNS. (5M)
6. Elucidate application layer protocol.(5M)

**PAPER NAME : INTERNET OF THINGS** **PAPER CODE : 17PCSE06**

**UNIT – III**

## PROTOTYPING EMBEDDED DEVICES

### ELECTRONICS

Before we get stuck into the ins and outs of microcontroller and embedded computer boards, let's address some of the electronics components that you might want to connect to them.
Don't worry if you're scared of things such as having to learn soldering. You are unlikely to need it for your initial experiments. Most of the prototyping can be done on what are called solderless breadboards. They enable you to build components together into a circuit with just a push-fit connection, which also means you can experiment with different options quickly and easily.

When it comes to thinking about the electronics, it's useful to split them into two main categories:

- Sensors: Sensors are the ways of getting information into your device, finding out things about your surroundings.

- Actuators: Actuators are the outputs for the device—the motors, lights, and so on, which let your device do something to the outside world.

Within both categories, the electronic components can talk to the computer in a number of ways.

The simplest is through digital I/O, which has only two states: a button can either be pressed or not; or an LED can be on or off. These states are usually connected via general-purpose input/output (GPIO) pins and map a digital 0 in the processor to 0 volts in the circuit and the digital 1 to a set voltage, usually the voltage that the processor is using to run (commonly 5V or 3.3V).

If you want a more nuanced connection than just on/off, you need an analogue signal. For example, if you wire up a potentiometer to let you read in the position of a rotary knob, you will get a varying voltage, depending on the knob's location. Similarly, if you want to run a motor at a speed other than off or full-speed, you need to feed it with a voltage somewhere between 0V and its maximum rating.
Because computers are purely digital devices, you need a way to translate between the analogue voltages in the real world and the digital of the computer.

An analogue-to-digital converter (ADC) lets you measure varying voltages. Microcontrollers often have a number of these converters built in. They will convert the voltage level between 0V and a predefined maximum (often the same 5V or 3.3V the processor is running at, but sometimes a fixed value such as 1V) into a number, depending on the accuracy of the ADC. The Arduino has 10-bit ADCs, which by default measure voltages between 0 and 5V. A voltage of 0 will give a reading of 0; a voltage of 5V would read 1023 (the maximum value that can be stored in a 10-bits); and voltages in between result in readings relative to the voltage. 1V would map to 205; a reading of 512 would mean the voltage was 2.5V; and so on.

The flipside of an ADC is a DAC, or digital-to-analogue converter. DACs let you generate varying voltages from a digital value but are less common as a standard feature of microcontrollers. This is due to a technique called pulse-width modulation (PWM), which gives an approximation to a DAC by rapidly turning a digital signal on and off so that the average value is the level you desire. PWM requires simpler circuitry, and for certain applications, such as fading an LED, it is actually the preferred option.

For more complicated sensors and modules, there are interfaces such as Serial Peripheral Interface (SPI) bus and Inter-Integrated Circuit (I2C). These standardised mechanisms allow modules to communicate, so sensors or things such as Ethernet modules or SD cards can interface to the microcontroller.

Naturally, we can't cover all the possible sensors and actuators available, but we list some of the more common ones here to give a flavour of what is possible.

## SENSORS

Pushbuttons and switches, which are probably the simplest sensors, allow some user input. Potentiometers (both rotary and linear) and rotary encoders enable you to measure movement.

Sensing the environment is another easy option. Light-dependent resistors (LDRs) allow measurement of ambient light levels, thermistors and other temperature sensors allow you to know how warm it is, and sensors to measure humidity or moisture levels are easy to build.

Microphones obviously let you monitor sounds and audio, but piezo elements (used in certain types of microphones) can also be used to respond to vibration.

Distance-sensing modules, which work by bouncing either an infrared or ultrasonic signal off objects, are readily available and as easy to interface to as a potentiometer.

## ACTUATORS

One of the simplest and yet most useful actuators is light, because it is easy to create electronically and gives an obvious output. Light-emitting diodes (LEDs) typically come in red and green but also white and other colours. RGB LEDs have a more complicated setup but allow you to mix the levels of red, green, and blue to make whatever colour of light you want. More complicated visual outputs also are available, such as LCD screens to display text or even simple graphics.

Piezo elements, as well as responding to vibration, can be used to create it, so you can use a piezo buzzer to create simple sounds and music. Alternatively, you can wire up outputs to speakers to create more complicated synthesized sounds.

Of course, for many tasks, you might also want to use components that move things in the real world. Solenoids can by used to create a single, sharp pushing motion, which could be useful for pushing a ball off a ledge or tapping a surface to make a musical sound.

More complicated again are motors. Stepper motors can be moved in steps, as the name implies. Usually, a fixed number of steps perform a full rotation. DC motors simply move at a given speed when told to. Both types of motor can be one-directional or move in both directions. Alternatively, if you want a motor that will turn to a given angle, you would need a servo. Although a servo is more controllable, it tends to have a shorter range of motion, often 180 or fewer degrees (whereas steppers and DC motors turn indefinitely). For all the kinds of motors that we've mentioned, you typically want to connect the motors to gears to alter the range of motion or convert circular movement to linear, and so on.

If you want to dig further into the ways of interfacing your computer or microcontroller with the real world, the "Interfacing with Hardware" page on the Arduino Playground website (http://playground.arduino.cc//Main/ InterfacingWithHardware) is a good place to start. Although Arduino-focused, most of the suggestions will translate to other platforms with minimal changes. For a more in-depth introduction to electronics, we recommend Electronics for Dummies (Wiley, 2009).

## SCALING UP THE ELECTRONICS

From the perspective of the electronics, the starting point for prototyping is usually a "breadboard". This lets you push-fit components and wires to make up circuits without requiring any soldering and therefore makes experimentation easy. When you're happy with how things are wired up, it's common to solder the components onto some protoboard, which may be sufficient to make the circuit more permanent and prevent wires from going astray.

Moving beyond the protoboard option tends to involve learning how to lay out a PCB. This task isn't as difficult as it sounds, for simple circuits at least, and mainly involves learning how to use a new piece of software and understanding some new terminology.

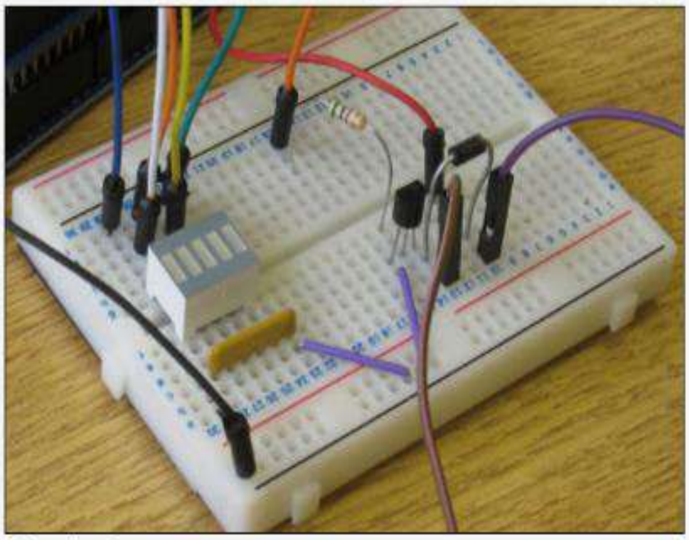
For small production runs, you'll likely use through-hole components, so called because the legs of the component go through holes in the PCB and tend to be soldered by hand. You will often create your designs as companion boards to an existing microcontroller platform—generally called shields in the Arduino community. This approach lets you bootstrap production without worrying about designing the entire system from scratch.

When you want to scale things even further, moving to a combined board allows you to remove any unnecessary components from the microcontroller board, and switching to surface mount components—where the legs of the chips are soldered onto the same surface as the chip—eases the board's assembly with automated manufacturing lines.
PCB design and the options for manufacturing are covered in much greater detail in Chapter 10, "Moving to Manufacture".

### Journey to a Circuit Board

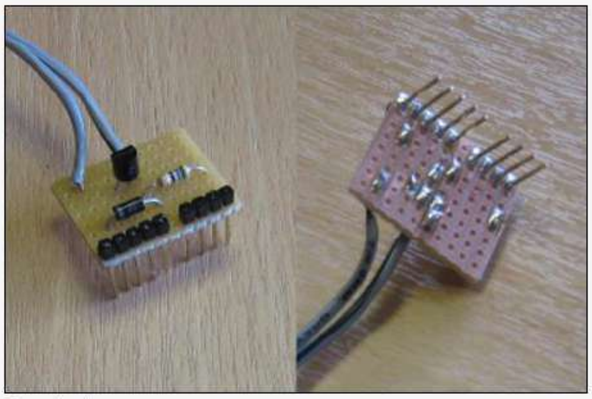
Let's look at the evolution of part of the Bubblino circuitry, from initial testing, through prototype, to finished PCB:

1. The first step in creating your circuit is generally to build it up on a breadboard. This way, you can easily reconfigure things as you decide exactly how it should be laid out.



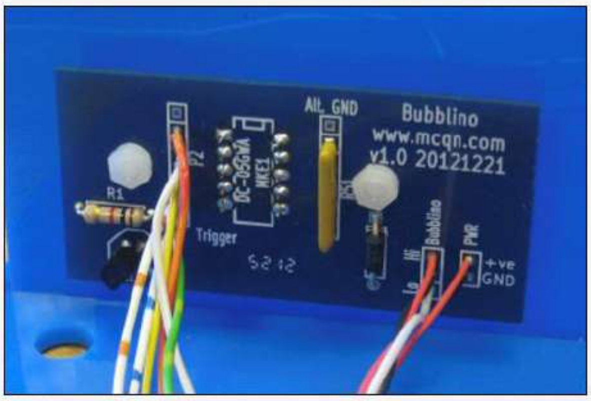
The breadboard.

2. When you are happy with how the circuit works, soldering it onto a stripboard will make the layout permanent. This means you can stop worrying about one of the wires coming loose, and if you're going to make only one copy of the circuit, that might be as far as you need take things.



The stripboard.

3. If you need to make many copies of the circuit, or if you want a professional finish, you can turn your circuit into a PCB. This makes it easier to build up the circuit because the position of each component will be labelled, there will be holes only where the components go, and there will be less chance of short circuits because the tracks between components will be protected by the solder resist.



The PCB.

## EMBEDDED COMPUTING BASICS

The rest of this chapter examines a number of different embedded computing platforms, so it makes sense to first cover some of the concepts and terms that you will encounter along the way.

Providing background is especially important because many of you may have little or no idea about what a microcontroller is. Although we've been talking about computing power getting cheaper and more powerful, you cannot just throw a bunch of PC components into something and call it an Internet of Things product. If you've ever opened up a desktop PC, you've seen that it's a collection of discrete modules to provide different aspects of functionality. It has a main motherboard with its processor, one or two smaller circuit boards providing the RAM, and a hard disk to provide the long-term storage. So, it has a lot of components, which provide a variety of general-purpose functionality and which all take up a corresponding chunk of physical space.

## MICROCONTROLLERS

Internet of Things devices take advantage of more tightly integrated and miniaturized solutions—from the most basic level of microcontrollers to more powerful system-on-chip (SoC) modules. These systems combine the processor, RAM, and storage onto a single chip, which means they are much more specialized, smaller than their PC equivalents, and also easier to build into a custom design.

These microcontrollers are the engines of countless sensors and automated factory machinery. They are the last bastions of 8-bit computing in a world that's long since moved to 32-bit and beyond. Microcontrollers are very limited in their capabilities—which is why 8-bit microcontrollers are still in use, although the price of 32-bit microcontrollers is now dropping to the level where they're starting to be edged out. Usually, they offer RAM capabilities measured in kilobytes and storage in the tens of kilobytes. However, they can still achieve a lot despite their limitations.

You'd be forgiven if the mention of 8-bit computing and RAM measured in kilobytes gives you flashbacks to the early home computers of the 1980s such as the Commodore 64 or the Sinclair ZX Spectrum. The 8-bit microcontrollers have the same sort of internal workings and similar levels of memory to work with. There have been some improvements in the intervening years, though—the modern chips are much smaller, require less power, and run about five times faster than their 1980s counterparts.

Unlike the market for desktop computer processors, which is dominated by two manufacturers (Intel and AMD), the microcontroller market consists of many manufacturers. A better comparison is with the automotive market. In the same way that there are many different car manufacturers, each with a range of models for different uses, so there are lots of microcontroller manufacturers (Atmel, Microchip, NXP, Texas Instruments, to name a few), each with a range of chips for different applications.

The ubiquitous Arduino platform is based around Atmel's AVR AT mega family of microcontroller chips. The on-board inclusion of an assortment of GPIO pins and ADC circuitry means that microcontrollers are easy to wire up to all manner of sensors, lights, and motors. Because the devices using them are focused on performing one task, they can dispense with most of what we would term an operating system, resulting in a simpler and much slimmer code footprint than that of a SoC or PC solution.

In these systems, functions which require greater resource levels are usually provided by additional single-purpose chips which at times are more powerful than their controlling microcontroller. For example, the WizNet Ethernet chip used by the Arduino Ethernet has eight times more RAM than the Arduino itself.

## SYSTEM-ON-CHIPS

In between the low-end microcontroller and a full-blown PC sits the SoC (for example, the Beagle Bone or the Raspberry Pi). Like the microcontroller, these SoCs combine a processor and a number of peripherals onto a single chip but usually have more capabilities. The processors usually range from a few hundred megahertz, nudging into the gigahertz for top-end solutions, and include RAM measured in megabytes rather than kilobytes. Storage for SoC modules tends not to be included on the chip, with SD cards being a popular solution.

The greater capabilities of SoC mean that they need some sort of operating system to marshal their resources. A wide selection of embedded operating systems, both closed and open source, is available and from both specialised embedded providers and the big OS players, such as Microsoft and Linux. Again, as the price falls for increased computing power, the popularity and familiarity of options such as Linux are driving its wider adoption.

## CHOOSING YOUR PLATFORM

How to choose the right platform for your Internet of Things device is as easy a question to answer as working out the meaning of life. This isn’t to say that it’s an impossible question—more that there are almost as many answers as there are possible devices. The platform you choose depends on the particular blend of price, performance, and capabilities that suit what you’re trying to achieve. And just because you settle on one solution, that doesn’t mean somebody else wouldn’t have chosen a completely different set of options to solve the same problem.

Start by choosing a platform to prototype in. The following sections discuss some of the factors that you need to weigh—and possibly play off against each other—when deciding how to build your device.

We cover the decisions that you need to make when scaling up both later in this chapter and in Chapter 10.

Processor Speed The processor speed, or clock speed, of your processor tells you how fast it can process the individual instructions in the machine code for the program it’s running. Naturally, a faster processor speed means that it can execute instructions more quickly.

The clock speed is still the simplest proxy for raw computing power, but it isn’t the only one. You might also make a comparison based on millions of instructions per second (MIPS), depending on what numbers are being reported in the datasheet or specification for the platforms you are comparing.

Some processors may lack hardware support for floating-point calculations, so if the code involves a lot of complicated mathematics, a by-the-numbers slower processor with hardware floating-point support could be faster than a slightly higher performance processor without it.

Generally, you will use the processor speed as one of a number of factors when weighing up similar systems. Microcontrollers tend to be clocked at speeds in the tens of MHz, whereas SoCs run at hundreds of MHz or possibly low GHz.

If your project doesn’t require heavyweight processing—for example, if it needs only networking and fairly basic sensing—then some sort of microcontroller will be fast enough. If your device will be crunching lots of data—for example, processing video in real time—then you’ll be looking at a SoC platform.

**RAM**

RAM provides the working memory for the system. If you have more RAM, you may be able to do more things or have more flexibility over your choice of coding algorithm. If you’re handling large datasets on the device, that could govern how much space you need. You can often find ways to work around memory limitations, either in code (see Chapter 8, “Techniques for Writing Embedded Code”) or by handing off processing to an online service (see Chapter 7, “Prototyping Online Components”).

It is difficult to give exact guidelines to the amount of RAM you will need, as it will vary from project to project. However, microcontrollers with less than 1KB of RAM are unlikely to be of interest, and if you want to run standard encryption protocols, you will need at least 4KB, and preferably more.

For SoC boards, particularly if you plan to run Linux as the operating system, we recommend at least 256MB.

Networking How your device connects to the rest of the world is a key consideration for Internet of Things products. Wired Ethernet is often the simplest for the user—generally plug and play—and cheapest, but it requires a physical cable. Wireless solutions obviously avoid that requirement but introduce a more complicated configuration.

WiFi is the most widely deployed to provide an existing infrastructure for connections, but it can be more expensive and less optimized for power consumption than some of its competitors.

Other short-range wireless can offer better power-consumption profiles or costs than WiFi but usually with the trade-off of lower bandwidth. ZigBee is one such technology, aimed particularly at sensor networks and scenarios such as home automation. The recent Bluetooth LE protocol (also known as Bluetooth 4.0) has a very low power-consumption profile similar to ZigBee’s and could see more rapid adoption due to its inclusion into standard Bluetooth chips included in phones and laptops. There is, of course, the existing Bluetooth standard as another possible choice. And at the boring but-very-cheap end of the market sit long-established options such as RFM12B which operate in the 434 MHz radio spectrum, rather than the 2.4 GHz range of the other options we’ve discussed.

For remote or outdoor deployment, little beats simply using the mobile phone networks. For low-bandwidth, higher-latency communication, you could use something as basic as SMS; for higher data rates, you will use the same data connections, like 3G, as a smartphone.

USB If your device can rely on a more powerful computer being nearby, tethering to it via USB can be an easy way to provide both power and networking. You can buy some of the microcontrollers in versions which include support for USB, so choosing one of them reduces the need for an extra chip in your circuit.

Instead of the microcontroller presenting itself as a device, some can also act as the USB “host”. This configuration lets you connect items that would normally expect to be connected to a computer—devices such as phones, for example, using the Android ADK, additional storage capacity, or WiFi dongles.

Devices such as WiFi dongles often depend on additional software on the host system, such as networking stacks, and so are better suited to the more computer-like option of SoC.

Power Consumption Faster processors are often more power hungry than slower ones. For devices which might be portable or rely on an unconventional power supply (batteries, solar power) depending on where they are installed, power consumption may be an issue. Even with access to mains electricity, the power consumption may be something to consider because lower consumption may be a desirable feature.

However, processors may have a minimal power-consumption sleep mode. This mode may allow you to use a faster processor to quickly perform operations and then return to low-power sleep. Therefore, a more powerful processor may not be a disadvantage even in a low-power embedded device.

Interfacing with Sensors and Other Circuitry In addition to talking to the Internet, your device needs to interact with something else—either sensors to gather data about its environment; or motors, LEDs, screens, and so on, to provide output. You could connect to the circuitry through some sort of peripheral bus—SPI and I2C being common ones—or through ADC or DAC modules to read or write varying voltages; or through generic GPIO pins, which provide digital on/off inputs or outputs. Different microcontrollers or SoC solutions offer different mixtures of these interfaces in differing numbers.

Physical Size and Form Factor The continual improvement in manufacturing techniques for silicon chips means that we've long passed the point where the limiting factor in the size of a chip is the amount of space required for all the transistors and other components that make up the circuitry on the silicon. Nowadays, the size is governed by the number of connections it needs to make to the surrounding components on the PCB.

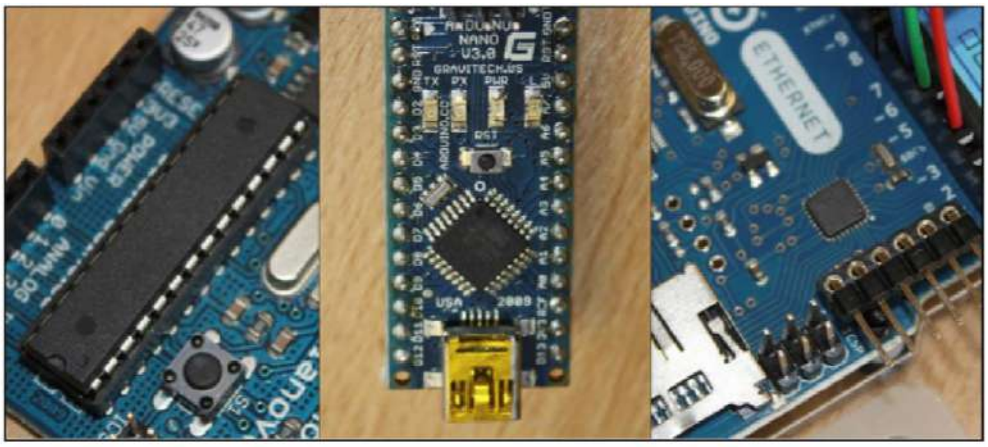
With the traditional through-hole design, most commonly used for homemade circuits, the legs of the chip are usually spaced at 0.1" intervals. Even if your chip has relatively few connections to the surrounding circuit—16 pins is nothing for such a chip—you will end up with over 1.5" (~4cm) for the perimeter of your chip. More complex chips can easily run to over a hundred connections; finding room for a chip with a 10" (25cm) perimeter might be a bit tricky!

You can pack the legs closer together with surface-mount technology because it doesn't require holes to be drilled in the board for connections. Combining that with the trick of hiding some of the connections on the underside of the chip means that it is possible to use the complex designs without resorting to PCBs the size of a table.

The limit to the size that each connection can be reduced to is then governed by the capabilities and tolerances of your manufacturing process. Some surface-mount designs are big enough for home-etched PCBs and can be hand-soldered. Others require professionally produced PCBs and accurate pick-and-place machines to locate them correctly.

Due to these trade-offs in size versus manufacturing complexity, many chip designs are available in a number of different form factors, known as packages. This lets the circuit designer choose the form that best suits his particular application.

All three chips pictured in the following figure provide identical functionality because they are all AVR ATmega328 microcontrollers. The one on the left is the through-hole package, mounted here in a socket so that it can be swapped out without soldering. The two others are surface mount, in two different packages, showing the reduction in size but at the expense of ease of soldering.



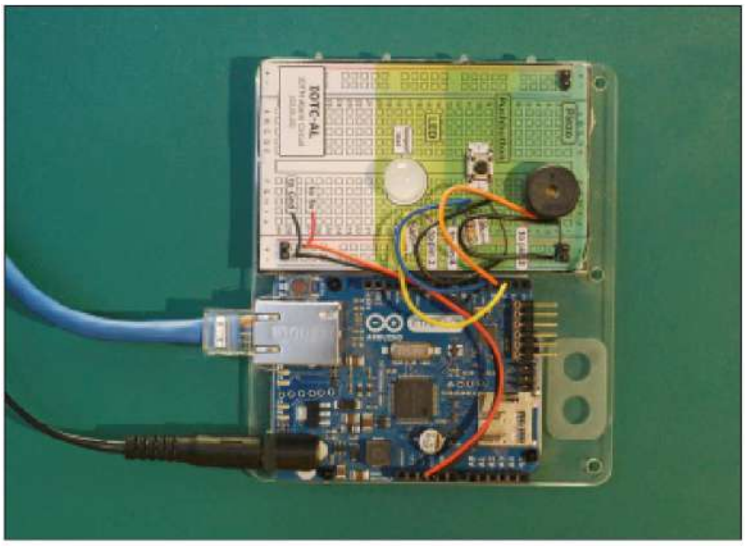
Through-hole versus surface-mount ATmega328 chips.

Looking at the ATmega328 leads us nicely into comparing some specific embedded computing platforms. We can start with a look at one which so popularised the ATmega328 that a couple of years ago it led to a worldwide shortage of the chip in the through-hole package, as for a short period demand outstripped supply.

## ARDUINO

Without a doubt, the poster child for the Internet of Things, and physical computing in general, is the Arduino.

These days the Arduino project covers a number of microcontroller boards, but its birth was in Ivrea in Northern Italy in 2005. A group from the Interaction Design Institute Ivrea (IDII) wanted a board for its design students to use to build interactive projects. An assortment of boards was around at that time, but they tended to be expensive, hard to use, or both.


An Arduino Ethernet board, plugged in, wired up to a circuit and ready for use.

So, the team put together a board which was cheap to buy—around £20— and included an onboard serial connection to allow it to be easily programmed. Combined with an extension of the Wiring software environment, it made a huge impact on the world of physical computing.

A decision early on to make the code and schematics open source meant that the Arduino board could outlive the demise of the IDII and flourish.  It also meant that people could adapt and extend the platform to suit their own needs.

As a result, an entire ecosystem of boards, add-ons, and related kits has flourished. The Arduino team's focus on simplicity rather than raw performance for the code has made the Arduino the board of choice in almost every beginner's physical computing project, and the open source ethos has encouraged the community to share circuit diagrams, parts lists, and source code. It's almost the case that whatever your project idea is, a quick search on Google for it, in combination with the word "Arduino", will throw up at least one project that can help bootstrap what you're trying to achieve. If you prefer learning from a book, we recommend picking up a copy of Arduino For Dummies, by John Nussey (Wiley, 2013).

The "standard" Arduino board has gone through a number of iterations: Arduino NG, Diecimila, Duemilanove, and Uno.

The Uno features an ATmega328 microcontroller and a USB socket for connection to a computer. It has 32KB of storage and 2KB of RAM, but don't let those meagre amounts of memory put you off; you can achieve a surprising amount despite the limitations.

The Uno also provides 14 GPIO pins (of which 6 can also provide PWM output) and 6 10-bit resolution ADC pins. The AT mega's serial port is made available through both the IO pins, and, via an additional chip, the USB connector.

If you need more space or a greater number of inputs or outputs, look at the Arduino Mega 2560. It marries a more powerful AT mega microcontroller to the same software environment, providing 256KB of Flash storage, 8KB of RAM, three more serial ports, a massive 54 GPIO pins (14 of those also capable of PWM) and 16 ADCs. Alternatively, the more recent Arduino Due has a 32-bit ARM core microcontroller and is the first of the Arduino boards to use this architecture. Its specs are similar to the Mega's, although it ups the RAM to 96KB.

## DEVELOPING ON THE ARDUINO

More than just specs, the experience of working with a board may be the most important factor, at least at the prototyping stage. As previously mentioned, the Arduino is optimized for simplicity, and this is evident from the way it is packaged for use. Using a single USB cable, you can not only power the board but also push your code onto it, and (if needed) communicate with it—for example, for debugging or to use the computer to store data retrieved by the sensors connected to the Arduino.

Of course, although the Arduino was at the forefront of this drive for ease-of use, most of the microcontrollers we look at in this chapter attempt the same, some less successfully than others.

Integrated Development Environment You usually develop against the Arduino using the integrated development environment (IDE) that the team supply at http://arduino.cc. Although this is a fully functional IDE, based on the one used for the Processing language (http://processing.org/), it is very simple to use. Most Arduino projects consist of a single file of code, so you can think of the IDE mostly as a simple file editor. The controls that you use the most are those to check the code (by compiling it) or to push code to the board.

Pushing Code Connecting to the board should be relatively straightforward via a USB cable. Sometimes you might have issues with the drivers (especially on some versions of Windows) or with permissions on the USB port (some Linux packages for drivers don't add you to the dialout group), but they are usually swiftly resolved once and for good. After this, you need to choose the correct serial port (which you can discover from system logs or select by trial and error) and the board type (from the appropriate menus, you may need to look carefully at the labelling on your board and its CPU to determine which option to select).

When your setup is correct, the process of pushing code is generally simple: first, the code is checked and compiled, with any compilation errors reported to you. If the code compiles successfully, it gets transferred to the Arduino and stored in its flash memory. At this point, the Arduino reboots and starts running the new code.

Operating System The Arduino doesn't, by default, run an OS as such, only the bootloader, which simplifies the code-pushing process described previously. When you switch on the board, it simply runs the code that you have compiled until the board is switched off again (or the code crashes).

It is, however, possible to upload an OS to the Arduino, usually a lightweight real-time operating system (RTOS) such as FreeRTOS/DuinOS. The main advantage of one of these operating systems is their built-in support for multitasking. However, for many purposes, you can achieve reasonable results with a simpler task-dispatching library.

If you dislike the simple life, it is even possible to compile code without using the IDE but by using the toolset for the Arduino's chip—for example, for all the boards until the recent ARM-based Due, the avr-gcc toolset.

The avr-gcc toolset (www.nongnu.org/avr-libc/) is the collection of programs that let you compile code to run on the AVR chips used by the rest of the Arduino boards and flash the resultant executable to the chip. It is used by the Arduino IDE behind the scenes but can be used directly, as well.

Language The language usually used for Arduino is a slightly modified dialect of C++ derived from the Wiring platform. It includes some libraries used to read and write data from the I/O pins provided on the Arduino and to do some basic handling for “interrupts” (a way of doing multitasking, at a very low level). This variant of C++ tries to be forgiving about the ordering of code; for example, it allows you to call functions before they are defined. This alteration is just a nicety, but it is useful to be able to order things in a way that the code is easy to read and maintain, given that it tends to be written in a single file.

The code needs to provide only two routines:

▪ setup(): This routine is run once when the board first boots. You could use it to set the modes of I/O pins to input or output or to prepare a data structure which will be used throughout the program.

▪ loop(): This routine is run repeatedly in a tight loop while the Arduino is switched on. Typically, you might check some input, do some calculation on it, and perhaps do some output in response.

To avoid getting into the details of programming languages in this chapter, we just compare a simple example across all the boards—blinking a single LED:

```
// Pin 13 has an LED connected on most Arduino boards.

// give it a name:

int led = 13;

// the setup routine runs once when you press reset:

void setup() {

// initialize the digital pin as an output.

pinMode(led, OUTPUT);

}

// the loop routine runs over and over again forever:

void loop()

{

digitalWrite(led, HIGH);

// turn the LED on   delay(1000);

// wait for a second  digitalWrite(led, LOW);

// turn the LED off  delay(1000);

 // wait for a second

}
```

Reading through this code, you'll see that the setup () function does very little; it just sets up that pin number 13 is the one we're going to control (because it is wired up to an LED).

Then, in loop (), the LED is turned on and then off, with a delay of a second between each flick of the (electronic) switch. With the way that the Arduino environment works, whenever it reaches the end of one cycle—on; wait a second; off; wait a second—and drops out of the loop () function, it simply calls loop() again to repeat the process.

Debugging Because C++ is a compiled language, a fair number of errors, such as bad syntax or failure to declare variables, are caught at compilation time. Because this happens on your computer, you have ample opportunity to get detailed and possibly helpful information from the compiler about what the problem is.

Although you need some debugging experience to be able to identify certain compiler errors, others, like this one, are relatively easy to understand:

Blink.cpp: In function 'void loop()':Blink:21:

error:'digitalWritee' was not declared in this scope

On line 21, in the function loop(), we deliberately misspelled the call to digitalWrite.

When the code is pushed to the Arduino, the rules of the game change, however. Because the Arduino isn't generally connected to a screen, it is hard for it to tell you when something goes wrong. Even if the code compiled successfully, certain errors still happen. An error could be raised that can't be handled, such as a division by zero, or trying to access the tenth element of a 9-element list. Or perhaps your program leaks memory and eventually just stops working. Or (and worse) a programming error might make the code continue to work dutifully but give entirely the wrong results.

If Bubblino stops blowing bubbles, how can we distinguish between the following cases?

- Nobody has mentioned us on Twitter.
- The Twitter search API has stopped working.
- Bubblino can't connect to the Internet.
- Bubblino has crashed due to a programming error.
- Bubblino is working, but the motor of the bubble machine has failed.
- Bubblino is powered off.

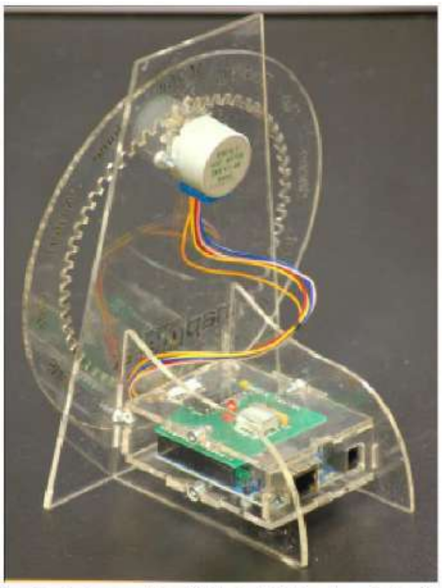
Adrian likes to joke that he can debug many problems by looking at the flashing lights at Bubblino's Ethernet port, which flashes while Bubblino connects to DNS and again when it connects to Twitter's search API, and so on. (He also jokes that we can discount the "programming error" option and that the main reason the motor would fail is that Hakim has poured bubble mix into the wrong hole. Again.) But while this approach might help distinguish two of the preceding cases, it doesn't help with the others and isn't useful if you are releasing the product into a mass market!

The first commercially available version of the Where Dial has a bank of half a dozen LEDs specifically for consumer-level debugging. In the case of an error, the pattern of lights showing may help customers fix their problem or help flesh out details for a support request.

Runtime programming errors may be tricky to trap because although the C++ language has exception handling, the avr-gcc compiler doesn't support it (probably due to the relatively high memory "cost" of handling exceptions); so the Arduino platform doesn't let you use the usual try... catch... logic.

Effectively, this means that you need to check your data before using it: if a number might conceivably be zero, check that before trying to divide by it. Test that your indexes are within bounds. To avoid memory leaks, look at the tips on writing code for embedded devices in Chapter 8, "Techniques for Writing Embedded Code".

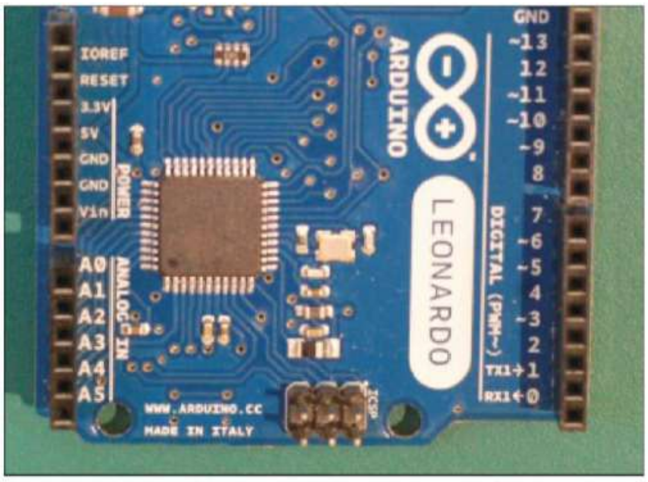
But code isn't, in general, created perfect: in the meantime you need ways to identify where the errors are occurring so that you can bullet-proof them for next time. In the absence of a screen, the Arduino allows you to write information over the USB cable using Serial. Write(). Although you can use the facility to communicate all kinds of data, debugging information can be particularly useful. The Arduino IDE provides a serial monitor which echoes the data that the Arduino has sent over the USB cable. This could include any textual information, such as logging information, comments, and details about the data that the Arduino is receiving and processing (to double-check that your calculations are doing the right thing).


Rear view of a transparent WhereDial. The bank of LEDs can be seen in the middle of the green board, next to the red "error" LED.

SOME NOTES ON THE HARDWARE The Arduino exposes a number of GPIO pins and is usually supplied with "headers" (plastic strips that sit on the pin holes, that provide a convenient solderless connection for wires, especially with a "jumper" connection). The headers are optimized for prototyping and for being able to change the purpose of the Arduino easily.

Each pin is clearly labelled on the controller board. The details of pins vary from the smaller boards such as the Nano, the classic form factor of the Uno, and the larger boards such as the Mega or the Due. In general, you have power outputs such as 5 volts or 3.3 volts (usually labelled 5V and 3V3, or perhaps just 3V), one or more electric ground connections (GND), numbered digital pins, and numbered analogue pins prefixed with an A.

You can power the Arduino using a USB connection from your computer. This capability is usually quite convenient during prototyping because you need the serial connection in any case to program the board. The Arduino also has a socket for an external power supply, which you might be more likely to use if you distribute the project. Either way should be capable of powering the microcontroller and the usual electronics that you might attach to it. (In the case of larger items, such as motors, you may have to attach external power and make that available selectively to the component using transistors.)



Close-up of an Arduino Leonardo board. Note the labelling of the power and analogue input connections.

Outside of the standard boards, a number of them are focused on a particular niche application—for example, the Arduino Ethernet has an on-board Ethernet chip and trades the USB socket for an Ethernet one, making it easier to hook up to the Internet. This is obviously a strong contender for a useful board for Internet of Things projects.

The Lilypad has an entirely different specialism, as it has a flattened form (shaped, as the name suggests, like a flower with the I/O capabilities exposed on its "petals") and is designed to make it easy to wire up with conductive thread, and so a boon for wearable technology projects.

Choosing one of the specialist boards isn't the only way to extend the capabilities of your Arduino. Most of the boards share the same layout of the assorted GPIO, ADC, and power pins, and you are able to piggyback an additional circuit board on top of the Arduino which can contain all manner of componentry to give the Arduino extra capabilities.

In the Arduino world, these add-on boards are called shields, perhaps because they cover the actual board as if protecting it.

Some shields provide networking capabilities—Ethernet, WiFi, or Zigbee wireless, for example. Motor shields make it simple to connect motors and servos; there are shields to hook up mobile phone LCD screens; others to provide capacitive sensing; others to play MP3 files or WAV files from an SD card; and all manner of other possibilities—so much so that an entire website, http://shieldlist.org/, is dedicated to comparing and documenting them.

In terms of functionality, a standard Arduino with an Ethernet shield is equivalent to an Arduino Ethernet. However, the latter is thinner (because it has all the components laid out on a single board) but loses the convenient USB connection. (You can still connect to it to push code or communicate over the serial connection by using a supplied adaptor.)

## OPENNESS

The Arduino project is completely open hardware and an open hardware success story.

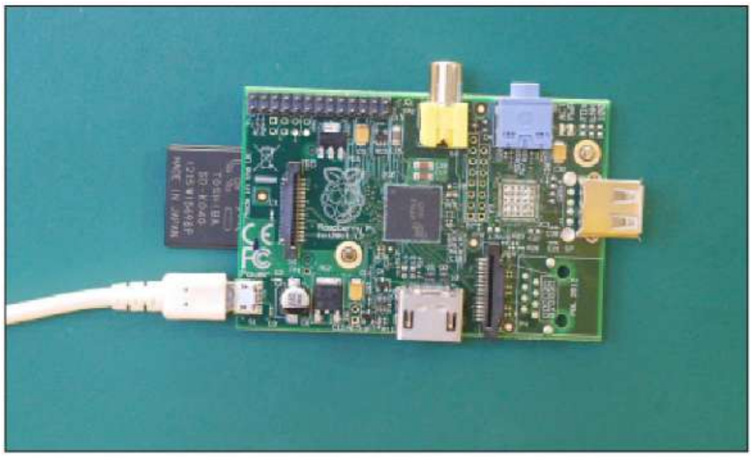
The only part of the project protected is the Arduino trademark, so they can control the quality of any boards calling themselves an Arduino. In addition to the code being available to download freely, the circuit board schematics and even the EAGLE PCB design files are easily found on the Arduino website.

This culture of sharing has borne fruit in many derivative boards being produced by all manner of people. Some are merely minor variations on the main Arduino Uno, but many others introduce new features or form factors that the core Arduino team have overlooked. In some cases, such as with the wireless-focused Arduino Fio board, what starts as a third-party board (it was originally the Funnel IO) is later adopted as an official Arduino-approved board.

## RASPBERRY

PI The Raspberry Pi, unlike the Arduino, wasn't designed for physical computing at all, but rather, for education. The vision of Eben Upton, trustee and cofounder of the Raspberry Pi Foundation, was to build a computer that was small and inexpensive and designed to be programmed and experimented with, like the ones he'd used as a child, rather than to passively consume games on. The Foundation gathered a group of teachers, programmers, and hardware experts to thrash out these ideas from 2006.

While working at Broadcom, Upton worked on the Broadcom BCM2835 system-on-chip, which featured an exceptionally powerful graphics processing unit (GPU), capable of high-definition video and fast graphics rendering. It also featured a low-power, cheap but serviceable 700 MHz ARM CPU, almost tacked on as an afterthought. Upton described the chip as "a GPU with ARM elements grafted on" (www.gamesindustry.biz/articles/ digitalfoundry-inside-raspberry-pi).


A Raspberry Pi Model B board. The micro USB connector only provides power to the board; the USB connectivity is provided by the USB host connectors (centre-bottom and centre-right).

The project has always taken some inspiration from a previous attempt to improve computer literacy in the UK: the “BBC Micro” built by Acorn in the early 1980s. This computer was invented precisely because the BBC producers tasked with creating TV programmes about programming realized that there wasn’t a single cheap yet powerful computer platform that was sufficiently widespread in UK schools to make it a sensible topic for their show. The model names of the Raspberry Pi, “Model A” and “Model B”, hark back to the different versions of the BBC Micro. Many of the other trustees of the Raspberry Pi Foundation, officially founded in 2009, cut their teeth on the BBC Micro. Among them was David Braben, who wrote the seminal game of space exploration, Elite, with its cutting-edge 3D wireframe graphics.

Due in large part to its charitable status, even as a small group, the Foundation has been able to deal with large suppliers and push down the costs of the components. The final boards ended up costing around £25 for the more powerful Model B (with built-in Ethernet connection). This is around the same price point as an Arduino, yet the boards are really of entirely different specifications.

The following table compares the specs of the latest, most powerful Arduino model, the Due, with the top-end Raspberry Pi Model B:

| | **Arduino Due** | **Raspberry Pi Model B** |
|---|---|---|
| CPU Speed | 84 MHz | 700 MHz ARM11 |
| GPU | None | Broadcom Dual-Core VideoCore IV Media Co-Processor |
| RAM | 96KB | 512MB |
| Storage | 512KB | SD card (4GB +) |
| OS | Bootloader | Various Linux distributions, other operating systems available |
| Connections | 54 GPIO pins | 8 GPIO pins |
| | 12 PWM outputs | 1 PWM output |
| | 4 UARTs | 1 UART |
| | SPI bus | SPI bus with two chip selects |
| | I²C bus | I²C bus |
| | USB 16U2 + native host | 2 USB host sockets |
| | 12 analogue inputs (ADC) | Ethernet |
| | 2 analogue outputs (DAC) | HDMI out |
| | | Component video and audio out |

So, the Raspberry Pi is effectively a computer that can run a real, modern operating system, communicate with a keyboard and mouse, talk to the Internet, and drive a TV/monitor with high-resolution graphics. The Arduino has a fraction of the raw processing power, memory, and storage required for it to run a modern OS. Importantly, the Pi Model B has built-in Ethernet (as does the Arduino Ethernet, although not the Due) and can also use cheap and convenient USB WiFi dongles, rather than having to use an extension “shield”.

Note that although the specifications of the Pi are in general more capable than even the top-of-the-range Arduino Due, we can’t judge them as “better” without considering what the devices are for! To see where the Raspberry Pi fits into the Internet of Things ecosystem, we need to look at the process of interacting with it and getting it to do useful physical computing work as an Internet-connected “Thing”, just as we did with the Arduino! We look at this next.

However, it is worth mentioning that a whole host of devices is available in the same target market as the Raspberry Pi: the Chumby Hacker Board, the BeagleBoard, and others, which are significantly more expensive. Yes, they may have slightly better specifications, but for the price difference, there may seem to be very few reasons to consider them above the Raspberry Pi. Even so, a project might be swayed by existing hardware, better tool support for a specific chipset, or ease-of-use considerations. In an upcoming section, we look at one such board, the BeagleBone, with regards to these issues.

## CASES AND EXTENSION BOARDS

Still, due to the relative excitement in the mainstream UK media, as well as the usual hacker and maker echo chambers, the Raspberry Pi has had some real focus. Several ecosystems have built up around the device. Because the Pi can be useful as a general-purpose computer or media centre without requiring constant prototyping with electronic components, one of the first demands enthusiasts have had was for convenient and attractive cases for it. Many makers blogged about their own attempts and have contributed designs to Thingiverse, Instructables, and others. There have also been several commercial projects. The Foundation has deliberately not authorized an “official” one, to encourage as vibrant an ecosystem as possible, although staffers have blogged about an early, well-designed case created by Paul Beech, the designer of the Raspberry Pi logo (http://shop.pimoroni. com/products/pibow).

Beyond these largely aesthetic projects, extension boards and other accessories are already available for the Raspberry Pi. Obviously, in the early days of the Pi’s existence post launch, there are fewer of these than for the Arduino; however, many interesting kits are in development, such as the Gertboard (www.raspberrypi.org/archives/tag/gertboard), designed for conveniently playing with the GPIO pins.

Whereas with the Arduino it often feels as though everything has been done already, in the early days of the Raspberry Pi, the situation is more encouraging. A lot of people are doing interesting things with their Pis, but as the platform is so much more high level and capable, the attention may be spread more thinly—from designing cases to porting operating systems to working on media centre plug-ins. Physical computing is just one of the aspects that attention may be paid to.

## DEVELOPING ON THE RASPBERRY PI

Whereas the Arduino’s limitations are in some ways its greatest feature, the number of variables on the Raspberry Pi are much greater, and there is much more of an emphasis on being able to do things in alternative ways. However, “best practices” are certainly developing. Following are some suggestions at time of writing. (It’s worth checking on the Raspberry Pi websites, IRC channels, and so on, later to see how they will have evolved.)

If you want to seriously explore the Raspberry Pi, you would be well advised to pick up a copy of the Raspberry Pi User Guide, by Eben Upton and Gareth Halfacree (Wiley, 2012).

## Operating System

Although many operating systems can run on the Pi, we recommend using a popular Linux distribution, such as

▪ Raspbian: Released by the Raspbian Pi Foundation, Raspbian is a distro based on Debian. This is the default "official" distribution and is certainly a good choice for general work with a Pi.

▪ Occidentalis: This is Adafruit's customized Raspbian. Unlike Raspbian, the distribution assumes that you will use it "headless"—not connected to keyboard and monitor—so you can connect to it remotely by default. (Raspbian requires a brief configuration stage first.)

For Internet of Things work, we recommend something such as the Adafruit distro. You're most probably not going to be running the device with a keyboard and display, so you can avoid the inconvenience of sourcing and setting those up in the first place. The main tweaks that interest us are that

▪ The sshd (SSH protocol daemon) is enabled by default, so you can connect to the console remotely.

▪ The device registers itself using zero-configuration networking (zeroconf) with the name raspberrypi. local, so you don't need to know or guess which IP address it picks up from the network in order to make a connection.

When we looked at the Arduino, we saw that perhaps the greatest win was the simplicity of the development environment. In the best case, you simply downloaded the IDE and plugged the device into the computer's USB. (Of course, this elides the odd problem with USB drivers and Internet connection when you are doing Internet of Things work.) With the Raspberry Pi, however, you've already had to make decisions about the distro and download it. Now that distro needs to be unpacked on the SD card, which you purchase separately. You should note that some SD cards don't work well with the Pi; apparently, "Class 10" cards work best. The class of the SD card isn't always clear from the packaging, but it is visible on the SD card with the number inside a larger circular "C".

At this point, the Pi may boot up, if you have enough power to it from the USB. Many laptop USB ports aren't powerful enough; so, although the "On" light displays, the device fails to boot. If you're in doubt, a powered USB hub seems to be the best bet.

An Electric Imp (left), next to a micro SD card (centre), and an SD card (right).

After you boot up the Pi, you can communicate with it just as you'd communicate with any computer—that is, either with the keyboard and monitor that you've attached, or with the Adafruit distro, via ssh as mentioned previously. The following command, from a Linux or Mac command line, lets you log in to the Pi just as you would log in to a remote server:

```
$ ssh root@raspberrypi.local
```

From Windows, you can use an SSH client such as PuTTY (www.chiark. greenend.org.uk/~sgtatham/putty/). After you connect to the device, you can develop a software application for it as easily as you can for any Linux computer. How easy that turns out to be depends largely on how comfortable you are developing for Linux.

Programming Language One choice to be made is which programming language and environment you want to use. Here, again, there is some guidance from the Foundation, which suggests Python as a good language for educational programming (and indeed the name "Pi" comes initially from Python).

Let's look at the "Hello World" of physical computing, the ubiquitous "blinking lights" example:

```
import RPi.GPIO as GPIO from
time import sleep

GPIO.setmode(GPIO.BOARD)        # set the numbering scheme to be the
                                # same as on the board
GPIO.setup(8, GPIO.OUT)         # set the GPIO pin 8 to output mode

led = False

GPIO.output(8, led)             # initiate the LED to off

while 1:

GPIO.output(8, led)

led = not led                   # toggle the LED status on/off for the next
                                # iteration
sleep(10)                       # sleep for one second
```

As you can see, this example looks similar to the C++ code on an Arduino. The only real differences are the details of the modularization: the GPIO code and even the sleep() function have to be specified. However, when you go beyond this level of complexity, using a more expressive "high-level" language like Python will almost certainly make the following tasks easier:

- Handling strings of character data
- Completely avoiding having to handle memory management (and bugs related to it)
- Making calls to Internet services and parsing the data received
- Connecting to databases and more complex processing
- Abstracting common patterns or complex behaviours

Also, being able to take advantage of readily available libraries on PyPi (https://pypi.python.org/pypi) may well allow simple reuse of code that other people have written, used, and thoroughly tested.

So, what's the catch? As always, you have to be aware of a few trade-offs, related either to the Linux platform itself or to the use of a high-level programming language. Later, where we mention "Python", the same considerations apply to most higher-level languages, from Python's contemporaries Perl and Ruby, to the compiled VM languages such as Java and C#. We specifically contrast Python with C++, as the low-level language used for Arduino programming.

▪ Python, as with most high-level languages, compiles to relatively large (in terms of memory usage) and slow code, compared to C++. The former is unlikely to be an issue; the Pi has more than enough memory. The speed of execution may or may not be a problem: Python is likely to be "fast enough" for most tasks, and certainly for anything that involves talking to the Internet, the time taken to communicate over the network is the major slowdown. However, if the electronics of the sensors and actuators you are working with require split-second timing, Python might be too slow. This is by no means certain; if Bubblino starts blowing bubbles a millisecond later, or the DoorBot unlocks the office a millisecond after you scan your RFID card to authenticate, this delay may be acceptable and not even noticeable.

▪ Python handles memory management automatically. Because handling the precise details of memory allocation is notoriously fiddly, automatic memory management generally results in fewer bugs and performs adequately. However, this automatic work has to be scheduled in and takes some time to complete. Depending on the strategy for garbage collection, this may result in pauses in operation which might affect timing of subsequent events. Also, because the programmer isn't exposed to the gory details, there may well be cases in which Python quite reasonably holds onto more memory than you might have preferred had you been managing it by hand. In worse cases, the memory may never be released until the process terminates: this is a so-called memory leak. Because an Internet of Things device generally runs unattended for long periods of time, these leaks may build up and eventually end up with the device running out of memory and crashing. (In reality, it's more likely that such memory leaks happen as a result of programming error in manual memory management.)

▪ **Linux itself arguably has some issues for "real-time" use.** Due to its being a relatively large operating system, with many processes that may run simultaneously, precise timings may vary due to how much CPU priority is given to the Python runtime at any given moment. This hasn't stopped many embedded programmers from moving to Linux, but it may be a consideration for your case.

▪ **An Arduino runs only the one set of instructions, in a tight loop, until it is turned off or crashes.** The Pi constantly runs a number of processes. If one of these processes misbehaves, or two of them clash over resources (memory, CPU, access to a file or to a network port), they may cause problems that are entirely unrelated to your code. This is unlikely (many well-run Linux computers run without maintenance for years and run businesses as well as large parts of the Internet) but may result in occasional, possibly intermittent, issues which are hard to identify and debug.

We certainly don't want to put undue stress on the preceding issues! They are simply trade-offs that may or may not be important to you, or rather more or less important than the features of the Pi and the access to a high-level programming language.

The most important issue, again, is probably the ease of use of the environment. If you're comfortable with Linux, developing for a Pi is relatively simple. But it doesn't approach the simplicity of the Arduino IDE. For example, the Arduino starts your code the moment you switch it on. To get the same behaviour under Linux, you could use a number of mechanisms, such as an initialization script in /etc/init.d.

First, you would create a wrapper script—for example, /etc/init.d/ StartMyPythonCode. This script would start your code if it's called with a start argument, and stop it if called with stop. Then, you need to use the chmod command to mark the script as something the system can run: chmod +x

/etc/init.d/StartMyPythonCode. Finally, you register it to run when the machine is turned on by calling sudo update-rc.d StartMyPythonCode defaults.
If you are familiar with Linux, you may be familiar with this mechanism for automatically starting services (or indeed have a preferred alternative). If not, you can find tutorials by Googling for “Raspberry Pi start program on boot” or similar. Either way, although setting it up isn’t hard per se, it’s much more involved than the Arduino way, if you aren’t already working in the IT field.

## Debugging

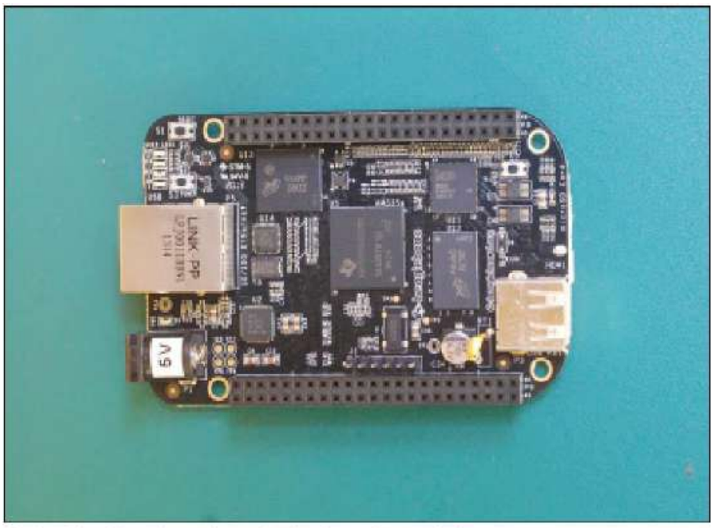
While Python’s compiler also catches a number of syntax errors and attempts to use undeclared variables, it is also a relatively permissive language (compared to C++) which performs a greater number of calculations at runtime. This means that additional classes of programming errors won’t cause failure at compilation but will crash the program when it’s running, perhaps days or months later.
Whereas the Arduino had fairly limited debugging capabilities, mostly involving outputting data via the serial port or using side effects like blinking lights, Python code on Linux gives you the advantages of both the language and the OS. You could step through the code using Python’s integrated debugger, attach to the process using the Linux strace command, view logs, see how much memory is being used, and so on. As long as the device itself hasn’t crashed, you may be able to ssh into the Raspberry Pi and do some of this debugging while your program has failed (or is running but doing the wrong thing).

Because the Pi is a general-purpose computer, without the strict memory limitations of the Arduino, you can simply use try... catch... logic so that you can trap errors in your Python code and determine what to do with them. For example, you would typically take the opportunity to log details of the error (to help the debugging process) and see if the unexpected problem can be dealt with so that you can continue running the code. In the worst case, you might simply stop the script running and have it restart again afresh!

Python and other high-level languages also have mature testing tools which allow you to assert expected behaviours of your routines and test that they perform correctly. This kind of automated testing is useful when you’re working out whether you’ve finished writing correct code, and also can be rerun after making other changes, to make sure that a fix in one part of the code hasn’t caused a problem in another part that was working before.

## BEAGLEBONE BLACK

The BeagleBone Black is the latest device to come from the BeagleBoard group. This group largely consists of employees of Texas Instruments, and although the products are not TI boards as such, they use many components with their employer’s blessing. The relationship is thus similar to that of the Raspberry Pi Foundation with Broadcom. Similarly, the BeagleBoard team want to create “powerful, open, and embedded devices” with a goal of contributing to the open source community, including facilitating education in electronics. However, there is less of an emphasis on creating a general-purpose computer for education. these boards are very much designed with the expectation that they will be used for physical computing and experimentation in electronics.


The latest board from the BeagleBone family: the BeagleBone Black.

The BeagleBone Black is the smallest and cheapest of the team's boards, with a form factor comparable to that of the Raspberry Pi. Although the specs of the two are mostly comparable, there are some interesting trade-offs.

The original BeagleBone has no video or audio outputs built in, but it does have a far larger number of GPIO pins, extracted into two rows of female headers. It also has the crucial ADC pins for analogue input which the Raspberry Pi lacks. This shows the development team's focus on building something for working with electronics rather than as a general-purpose computer.

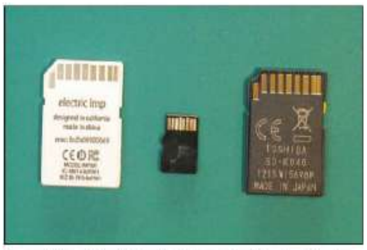
The BeagleBone was released before the Raspberry Pi, and its price reflects that. If you think of it as a more powerful embedded development board than any of the Arduino offerings, then a £63 price tag looks quite reasonable. When you compare it to a £25 Raspberry Pi, however, it makes less sense.

The influence of the Pi can be seen in the latest revision of the BeagleBone platform, the BeagleBone Black. Although still missing the analogue video and audio connectors, it adds a micro-HDMI connector to provide digital outputs for both audio and video. The price is also much closer at £31, and it retains the much better electronics interfacing capabilities of the original BeagleBone.

Let's compare the specs of the Raspberry Pi Model B with the new BeagleBone Black:

| | BeagleBone Black | Raspberry Pi Model B |
|---|---|---|
| CPU Speed | 1GHz ARM Cortex-A8 | 700 MHz ARM11 |
| GPU | SGX530 3D | Broadcom Dual-Core VideoCore IV Media Co-Processor |
| RAM | 512MB | 512MB |
| Storage | 2GB embedded MMC, plus uSD card | SD card (4GB +) |
| OS | Various Linux distributions, Android, other operating systems available | Various Linux distributions, other operating systems available |
| Connections | 65 GPIO pins, of which 8 have PWM<br>4 UARTs<br>SPI bus<br>I²C bus<br>USB client + host<br>Ethernet<br>Micro-HDMI out<br>7 analog inputs (ADC)<br>CAN bus | 8 GPIO pins, of which 1 has PWM<br>1 UART<br>SPI bus with two chip selects<br>I²C bus<br>2 USB host sockets<br>Ethernet<br>HDMI out<br>Component video and audio out |

## ELECTRIC IMP


The rear of (from left to right): an Electric Imp, a micro SD card, an SD card.

Although we're featuring the Electric Imp here, it's a less mature and, in some ways, more problematic offering than the other boards we've discussed, and we can't tell, at the time of writing, if the platform will develop into a viable choice. But it is worth discussing in some detail, as a possible paradigm shift in the way that developers approach consumer electronics and physical computing.

Hugo Fiennes, formerly engineering manager on Apple's iPhone team, was attempting to connect LED lights to Google's share price. He evaluated various home automation options, like ZigBee, but realised that they were mostly single-vendor solutions, often using their own radio standards rather than based on open platforms (http://www.edn.com/electronicsnews/4373185/Former-Apple-Google-Facebook-engineerslaunch-IoT-startup-item-2). The Electric Imp uses a number of existing standards, such as WiFi, and the form factor of SD cards but ends up being very much less of an open platform than all the other devices that we've looked at in this chapter. Fiennes collaborated on the project with Kevin Fox, a former Gmail designer, and firmware engineer Peter Hartley. As you'll see, the startup feels as though it has much of the DNA (for good and bad) of the beautiful, technically polished walled gardens that are the iPhone and Gmail.

All the smarts of the Electric Imp, and also its WiFi connectivity, are located in an SD card–shaped microcontroller. It's important to note that the Imp isn't actually an SD card; it's just shaped like one. Using the same form factor means that producing the Imps is cheaper because the team can reuse existing cases and tooling, as well as existing component connectors for the impee (the name Electric Imp use for the rest of the circuit that the Imp plugs into). This last factor is important, as you see in the upcoming "Openness" section.

Although an SD card feels very robust, it is, on the outside, effectively a small, flat piece of plastic. It offers only one affordance to connect it to anything: namely, plug it into a device. Just as you would insert an SD card into a music player, computer, or printer, you insert an Imp into an impee. This host board provides power, GPIO connections to sensors and actuators, and an ID chip so that the Imp knows which device it's plugged into.

The Imp costs around £20, while an impee costs less than half that. Here, having used the standard SD form factor turns out to be a great choice for the prototyper. For prototyping a number of projects, you need only a single Imp, which can be reused across all the projects. You will see shortly that reconfiguring the Imp to run on a different impee is automatic, which is a very nice feature.

## PROTOTYPING THE PHYSICAL DESIGN

### NONDIGITAL METHODS

We've already seen how pen and paper remain essential tools in the designer's arsenal, but they aren't the only ones to have survived the digital revolution. Many of what could be deemed more traditional craft techniques are just as valid for use when prototyping the physical form of your device.

One of the key advantages that these techniques have over the newer digital fabrication methods is their immediacy. Three-dimensional printing times are often measured in hours, and although laser cutting is much faster, performing a cut still takes minutes. And all this is without including the time taken to revise the design on the computer first.

Compare that to the speed with which you can reconfigure a model made from clay or from LEGO—and that isn't just down to the hours of practice you put in while you were growing up! Keeping the feedback loop as short as possible between having an idea and trying it out frees you up for more experimentation.

Let's look at some of the more common options here:

- **Modelling clay:** The most well-known brands are Play-Doh and Plasticine, but you can find a wealth of different versions with slightly different qualities. Some, like Play-Doh, have a tendency to dry out and crack if left exposed to the air. Plasticine doesn't suffer from this problem, but as it remains malleable, it isn't ideal for prototypes which are going to be handled. Modelling clay is best used for short-term explorations of form, rather than longer-term functional prototypes.

- **Epoxy putty:** You might have encountered this product as the brand Milliput. It is similar to modelling clay although usually available in fewer colours. It comes in two parts, one of which is a hardener. You mix equal parts together to activate the epoxy. You then mould it to the desired shape, and in about an hour, it sets solid. If you like, you can then sand it or paint it for a better finish, so this product works well for more durable items.

- **Sugru:** Sugru is a mouldable silicone rubber. Like epoxy putty, it can be worked for only a short time before it sets (about 30 minutes, and then about a day to fully cure); but unlike epoxy, once cured, it remains flexible. It is also good at sticking to most other substances and gives a soft-touch grippy surface, which makes it a great addition to the designer's (and hacker's) toolkit.

- **Toy construction sets:** We've already mentioned the ubiquitous LEGO sets, but you might also consider Meccano (or Erector Sets in the United States) and plenty of others. If you're lucky, you already have some gathering dust in the attic or that you can borrow from your children. The other interesting feature of these sets is the availability of gears, hinges, and other pieces to let you add some movement to your model. You can purchase systems to control LEGO sets from a computer, but there's no requirement for you to use them. Many hackers combine an Arduino for sensing and control with LEGO for form and linkages, as this provides an excellent blend of flexibility and ease of construction.

- **Cardboard:** Cardboard is cheap and easy to shape with a craft knife or scissors, and available in all manner of colours and thicknesses. In its corrugated form, it provides a reasonable amount of structural integrity and works well for sketching out shapes that you'll later cut out of thin plywood or sheets of acrylic in a laser cutter (a topic we return to when we look at laser cutting later in the chapter).

- **Foamcore or foamboard:** This sheet material is made up of a layer of foam sandwiched by two sheets of card. It's readily available at art supplies shops and comes in 3mm or 5mm thicknesses in a range of sizes. Like cardboard, it is easily cut with a craft knife, although it is more rigid than corrugated cardboard. There are also specialist foamboard craft knives which allow easy 45-degree cuts for mitred edges and have two blades—spaced 3mm apart—which make it trivial to cut slots into which you can insert another sheet of foamboard to generate three-dimensional shapes.

- **Extruded polystyrene:** This product is similar to the expanded polystyrene that is used for packaging but is a much denser foam that is better suited to modelling purposes. It is often referred to as "blue foam", although it's the density rather than the colour which is important. Light yet durable, it can be easily worked: you can cut it with a craft knife, saw it, sand it, or, for the greatest ease in shaping it, buy a hot-wire cutter. Sheets of extruded polystyrene are much thicker than foamboard, usually between 25mm and 165mm. As a result, it is great for mocking up solid three-dimensional shapes. If you need something thicker than the sheet itself, you can easily glue a few layers together. The dust from sanding it and the fumes given off when cutting it with a hot-wire cutter aren't too nice, so make sure you wear a dust mask and keep the area ventilated when working with it.

Having reviewed the sorts of techniques which you learnt when your design education started, back in primary school, we can move on to looking at some of the newer tools. Like most aspects of modern life, computers have also swept through manufacturing, opening new possibilities in rapid prototyping. The combination of Moore's Law driving down the cost of computing and the expiration of the patents from the early developments in the 1980s has brought such technology within the reach of the hobbyist or small business.

## LASER CUTTING

Although the laser cutter doesn't get the same press attention as the 3D printer, it is arguably an even more useful item to have in your workshop. Three-dimensional printers can produce more complicated parts, but the simpler design process (for many shapes, breaking it into a sequence of two-dimensional planes is easier than designing in three dimensions), greater range of materials which can be cut, and faster speed make the laser cutter a versatile piece of kit.

Laser cutters range from desktop models to industrial units which can take a full 8' by 4' sheet in one pass. Most commonly, though, they are floorstanding and about the same size as a large photocopier.

Most of the laser cutter is given over to the bed; this is a flat area that holds the material to be cut. The bed contains a two-axis mechanism with mirrors and a lens to direct the laser beam to the correct location and focus it onto the material being cut. It is similar to a flatbed plotter but one that burns things rather than drawing on them.

The computer controls the two-axis positioning mechanism and the power of the laser beam. This means that not only can the machine easily cut all manner of intricate patterns, but it can also lower the power of the laser so that it doesn't cut all the way through. At a sufficiently low power, this feature enables you to etch additional detail into the surface of the piece. You can also etch things at different power levels to achieve different depths of etching, but whilst the levels will be visibly different, it isn't precise enough to choose a set fraction of a millimeter depth.

## CHOOSING A LASER CUTTER

When choosing a laser cutter, you should consider two main features:

▪ The size of the bed: This is the place where the sheet of material sits while it's being cut, so a larger bed can cut larger items. You don't need to think just about the biggest item you might create; a larger bed allows you to buy material in bigger sheets (which is more cost effective), and if you move to small-scale production, it would let you cut multiple units in one pass.

▪ The power of the laser: More powerful lasers can cut through thicker material. For example, the laser cutter at our workplace has a 40W laser, which can cut up to 10mm-thick acrylic. Moving a few models up in the same range, to one with a 60W laser, would allow us to cut 25mmthick acrylic.

## 3D PRINTING

Additive manufacturing, or 3D printing as it's often called, is fast becoming one of the most popular forms in rapid prototyping—largely down to the ever-increasing number of personal 3D printers, available at ever-falling costs. Now a number of desktop models, available for less than £500, produce decent quality results.

The term additive manufacturing is used because all the various processes which can be used to produce the output start with nothing and add material to build up the resulting model. This is in contrast to subtractive manufacturing techniques such as laser cutting and CNC milling, where you start with more material and cut away the parts you don't need.

Various processes are used for building up the physical model, which affect what materials that printer can use, among other things. However, all of them take a three-dimensional computer model as the input. The software slices the computer model into many layers, each a fraction of a millimeter thick, and the physical version is built up layer by layer.

One of the great draws of 3D printing is how it can produce items which wouldn't be possible with traditional techniques. For example, because you can print interlocking rings without any joins, you are able to use the metal 3D printers to print entire sheets of chain-mail which come out of the printer already connected together. If only the medieval knights had had access to a metal laser-sintering machine, their armour would have been much easier to produce.

## TYPES OF 3D PRINTING:

▪ **Fused filament fabrication (FFF):** Also known as fused deposition modeling (FDM), this is the type of 3D printer you're most likely to see at a maker event. The RepRap and MakerBot designs both use this technique, as does the Stratasys at the industrial level.

- **Laser sintering:** This process is sometimes called selective laser sintering (SLS), electron beam melting (EBM), or direct metal laser sintering (DMLS). It is used in more industrial machines but can print any material which comes in powdered form and which can be melted by a laser. It provides a finer finish than FDM, but the models are just as robust, and they're even stronger when the printing medium is metal. This technique is used to print aluminium or titanium, although it can just as easily print nylon.

- **Powder bed:** Like laser sintering, the powder-bed printers start with a raw material in a powder form, but rather than fusing it together with a laser, the binder is more like a glue which is dispensed by a print head similar to one in an inkjet printer. The Z Corp. machines use this technique and use a print medium similar in texture to plaster. After the printing process, the models are quite brittle and so need post-processing where they are sprayed with a hardening solution. The great advantage of these printers is that when the binder is being applied, it can be mixed with some pigment; therefore, full-colour prints in different colours can be produced in one pass.

- **Laminated object manufacturing (LOM):** This is another method which can produce full-colour prints. LOM uses traditional paper printing as part of the process. Because it builds up the model by laminating many individual sheets of paper together, it can print whatever colours are required onto each layer before cutting them to shape and gluing them into place. The Mcor IRIS is an example of this sort of printer.

- **Stereolithography and digital light processing:** Stereolithography is possibly the oldest 3D printing technique and has a lot in common with digital light processing, which is enjoying a huge surge in popularity and experimentation at the time of this writing. Both approaches build their models from a vat of liquid polymer resin which is cured by exposure to ultraviolet light. Stereolithography uses a UV laser to trace the pattern for each layer, whereas digital light processing uses a DLP projector to cure an entire layer at a time. Whilst these approaches are limited to printing with resin, the resultant models are produced to a fine resolution. The combination of this with the relatively low cost of DLP projectors makes this a fertile area for development of more affordable high-resolution printers.

## CNC MILLING

Computer Numerically Controlled (CNC) milling is similar to 3D printing but is a subtractive manufacturing process rather than additive. The CNC part just means that a computer controls the movement of the milling head, much like it does the extruder in an FDM 3D printer. However, rather than building up the desired model layer by layer from nothing, it starts with a block of material larger than the finished piece and cuts away the parts which aren't needed—much like a sculptor chips away at a block of stone to reveal the statue, except that milling uses a rotating cutting bit (similar to an electric drill) rather than a chisel.

Because cutting away material is easier, CNC mills can work with a much greater range of materials than 3D printers can. You still need an industrialscale machine to work with hardened steel, but wax, wood, plastic, aluminium, and even mild steel can be readily milled with even desktop mills.

CNC mills can also be used for more specialised (but useful when prototyping electronic devices) tasks, such as creating custom printed circuit boards. Rather than sending away for your PCB design to be fabricated or etching it with acid, you can convert it into a form which your CNC mill can rout out; that is, the CNC mills away lines from the metal surface on the board, leaving the conductive paths. An advantage of milling over etching the board is that you can have the mill drill any holes for components or mounting at the same time, saving you from having to do it manually afterwards with your drill press.

Beyond size and accuracy, the other main attribute that varies among CNC mills is the number of axes of movement they have:

■ 2.5 axis: Whilst this type has three axes of movement—X, Y, and Z—it can move only any two at one time.

■ 3 axis: Like the 2.5-axis machine, this machine has a bed which can move in the X and Y axes, and a milling head that can move in the Z. However, it can move all three at the same time (if the machining instructions call for it).

■ 4 axis: This machine adds a rotary axis to the 3-axis mill to allow the piece being milled to be rotated around an extra axis, usually the X (this is known as the A axis). An indexed axis just allows the piece to be rotated to set points to allow a further milling pass to then be made, for example, to flip it over to mill the underside; and a fully controllable rotating axis allows the rotation to happen as part of the cutting instructions.

■ 5 axis: This machine adds a second rotary axis—normally around the Y—which is known as the B axis.

■ 6 axis: A third rotary axis—known as the C axis if it rotates around Z—completes the range of movement in this machine.

For prototyping work, you're unlikely to need anything beyond a 3-axis mill, although a fourth axis would give you some extra flexibility. The 5- and 6-axis machines tend to be the larger, more industrial units.

As with 3D printing, the software you use for CNC milling is split into two types:

■ CAD (Computer-Aided Design) software lets you design the model.

■ CAM (Computer-Aided Manufacture) software turns that into a suitable toolpath—a list of co-ordinates for the CNC machine to follow which will result in the model being revealed from the block of material.

The toolpaths are usually expressed in a quasi-standard called G-code. Whilst most of the movement instructions are common across machines, a wide variety exists in the codes for things such as initializing the machine. That said, a number of third-party CAM packages are available, so with luck you will have a choice of which to use. For a rundown of the possibilities, along with lots more information about getting started with CNC milling, see http://lcamtuf.coredump.cx/gcnc/.

## REPURPOSING/RECYCLING

So far, we've talked just about how you would go about creating a new object completely from scratch. Owning the designs of and knowing how to create all of the components of your device put you in a great position, but they aren't necessarily the overriding concerns in all prototyping scenarios.

As with the other elements of building your connected device, a complete continuum exists from buying-in the item or design through to doing-it yourself. So, just as you wouldn't think about making your own nuts and bolts from some iron ore, sometimes you should consider reusing more complex mechanisms or components.

One reason to reuse mechanisms or components would be to piggyback onto someone else's economies of scale. If sections or entire subassemblies that you need are available in an existing product, buying those items can often be cheaper than making them in-house. That's definitely the case for your prototypes but may extend to production runs, too, depending on the volumes you'll be manufacturing. For example, the bubble machine used in Bubblino is an off-the-shelf unit from a children's game.

In the batch production volumes that Bubblino is currently being built, it’s cheaper to buy them, even at retail price, than it would be to manufacture the assorted gears, fans, bubble ring, and casing in-house.

**Question Bank:**

1. Explain prototyping Embedded Devices. (10M)
2. Explicate prototyping the physical design. (10M)
3. Describe raspberry Pi. (5M)
4. Illustrate Beagle Bone Black. (5M)
5. Elaborate Laser Cutting. (5M)
6. Illuminate 3D printing. (5M)
7. What is CNC milling? (5M)

## UNIT – IV

## PROTOTYPING ONLINE COMPONENTS

### API

The most important part of a web service, with regards to an Internet of Things device, is the Application Programming Interface, or API. An API is a way of accessing a service that is targeted at machines rather than people. If you think about your experience of accessing an Internet service, you might follow a number of steps. For example, to look at a friend's photo on Flickr, you might do the following:

1. Launch Chrome, Safari, or Internet Explorer.
2. Search for the Flickr website in Google and click on the link.
3. Type in your username and password and click "Login".
4. Look at the page and click on the "Contacts" link.
5. Click on a few more links to page through the list of contacts till you see the one you want.
6. Scroll down the page, looking for the photo you want, and then click on it.

### MASHING UP APIS

Perhaps the data you want is already available on the Internet but in a form that doesn't work for you? The idea of "mashing up" multiple APIs to get a result has taken off and can be used to powerful effect. For example:

- Using a mapping API to plot properties to rent or buy—for example, Google Maps to visualise properties to rent via Craigslist, or Foxtons in London showing its properties using Mapumental.
- Showing Twitter trends on a global map or in a timeline or a charting API.
- Fetching Flickr images that are related to the top headlines retrieved from The Guardian newspaper's API.

### SCRAPING

In many cases, companies or institutions have access to fantastic data but don't want to or don't have the resources or knowledge to make them available as an API. While you saw in the Flickr example above that getting a computer to pretend to be a browser and navigate it by looking for UI elements was fragile, that doesn't mean that doing so is impossible. In general, we refer to this, perhaps a little pejoratively, as "screen-scraping".

### LEGALITIES

Screen-scraping may break the terms and conditions of a website. For example, Google doesn't allow you to screen-scrape its search pages but does provide an API. Even if you don't think about legal sanctions, breaking the terms and conditions for a company like Google might lead to its denying you its other services, which would be at the very least inconvenient.

## WRITING A NEW API

Assuming the data you want to play with isn't available or can't be easily mashed up or scraped using other existing tools and sources, perhaps you want to create an entirely new source of information or services. Perhaps you plan to assemble the data from free or licensed material you have and process it. Or perhaps your Internet-connected device can populate this data.

Example: Clockodillo

## CLOCKODILLO

As we saw earlier, Clockodillo is an Internet-connected task timer. The user can set a dial to a number of minutes, and the timer ticks down until completed. It also sends messages to an API server to let it know that a task has been started, completed, or cancelled.

A number of API interactions deal precisely with those features of the physical device:

- Start a new timer
- Change the duration of an existing timer
- Mark a timer completed
- Cancel a timer

Some interactions with a timer data structure are too complicated to be displayed on a device consisting mostly of a dial—for example,

- View and edit the timer's name/description

And, naturally, the user may want to be able to see historical data:

- Previous timers, in a list
    - Their name/description
    - Their total time and whether they were cancelled

## SECURITY

Security is depending a lot on how sensitive the information being passed is and whether it's in anyone's interest to compromise it.

The two main cases here are as follows:

- Someone who is targeting a specific user and has access to that person's wired or (unencrypted) wireless network. This attacker could read the details and use them (to create fake timers or get information about the user).

- Someone who has access to one of the intermediate nodes. This person won't be targeting a specific device but may be looking to see what unencrypted data passes by, to see what will be a tempting target.

## IMPLEMENTING THE API

An API defines the messages that are sent from client to server and from server to client. Ultimately, you can send data in whatever format you want, but it is almost always better to use an existing standard because convenient libraries will exist for both client and server to produce and understand the required messages.

Here are a few of the most common standards that you should consider:

- **Representational State Transfer (REST):** Access a set of web URL.

- **JSON-RPC:** Access a single web URL.

- **XML-RPC:** This standard is just like JSON-RPC but uses XML instead of JSON.

- **Simple Object Access Protocol (SOAP):** This standard uses XML for transport like XML-RPC but provides additional layers of functionality, which may be useful for very complicated systems.

## REAL-TIME REACTIONS

To establish an HTTP request requires several round-trips to the server. There is the TCP "three-step handshake" consisting of a SYN (synchronize) request from the client, a SYN-ACK from the server to "acknowledge" the request, and finally an ACK from the client. Although this process can be near instantaneous, it could also take a noticeable amount of time.

We look at two options here: polling and the so-called "Comet" technologies. And then, in the section on non-HTTP protocols, MQTT, XMPP, and CoAP offer alternative solutions.

## POLLING

If you want the device or another client to respond immediately, how do you do that? You don't know when the event you want to respond to will happen, so you can't make the request to coincide with the data becoming available. Consider these two cases:

- The Where Dial should start to turn to "Work" the moment that the user has checked into his office.

- The moment that the task timer starts, the client on the user's computer should respond, offering the opportunity to type a description of the task.

But this would put load on the following:

- **The server:** If the device takes off, and there are thousands of devices, each of them polling regularly, you will have to scale up to that load.

- **The client:** This is especially important if, as per the earlier Arduino example, the microcontroller blocks during each connect.

## COMET

Comet is an umbrella name for a set of technologies developed to get around the inefficiencies of polling. As with many technologies, many of them were developed before the "brand" of Comet was invented; however, having a name to express the ideas is useful to help discuss and exchange ideas and push the technology forward.

**Long Polling (Unidirectional)**

The first important development was “long polling”, which starts off with the client making a polling request as usual. However, unlike a normal poll request, in which the server immediately responds with an answer, even if that answer is “nothing to report”, the long poll waits until there is something to say. This means that the server must regularly send a keep-alive to the client to prevent the Internet of Things device or web page from concluding that the server has simply timed out.

**Multipart XMLHttpRequest (MXHR) (Unidirectional)**

When building web applications, it is common to use a JavaScript API called XMLHttpRequest to communicate with the web server without requiring a full new page load. From the web server’s point of view, these requests are no different from any other HTTP request, but because the intended recipient is some client-side code, conventions and support libraries (both client- and server-side) have developed to address this method of interaction specifically.

**HTML5 Web Sockets (Bidirectional)**

Traditionally, the API used to talk directly to the TCP layer is known as the socket’s API. When the web community was looking to provide similar capabilities at the HTTP layer, they called the solution Web Sockets.

**OTHER PROTOCOLS**

As you have seen, although HTTP is an extremely popular protocol on the Internet, it isn’t ideally suited to all situations. Rather than work around its limitations with one of the preceding solutions, another option—if you have control of both ends of the connection—is to use a different protocol completely.

There are plenty of protocols to choose from, but we will give a brief rundown of some of the options better suited to Internet of Things applications.

1. MQ TELEMETRY TRANSPORT

MQTT is a lightweight messaging protocol, designed specifically for scenarios where network bandwidth is limited or a small code footprint is desired. It was developed initially by IBM but has since been published as an open standard, and a number of implementations, both open and closed source, are available, together with libraries for many different languages.

2. EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL

Another messaging solution is the Extensible Messaging and Presence Protocol, or XMPP. XMPP grew from the Jabber instant messaging system and so has broad support as a general protocol on the Internet. This is both a blessing and a curse: it is well understood and widely deployed, but because it wasn’t designed explicitly for use in embedded applications, it uses XML to format the messages. This choice of XML makes the messaging relatively verbose, which could preclude it as an option for RAM-constrained microcontrollers.

3. CONSTRAINED APPLICATION PROTOCOL

The Constrained Application Protocol (CoAP) is designed to solve the same classes of problems as HTTP but, like MQTT-S, for networks without TCP. There are proposals for running CoAP over UDP, SMS mobile phone messaging, and integration with 6LoWPAN. CoAP draws many of its design features from HTTP and has a defined mechanism to proxies to allow mapping from one protocol to the other.

# TECHNIQUES FOR WRITING EMBEDDED CODE

## MEMORY MANAGEMENT

When you don't have a lot of memory to play with, you need to be careful as to how you use it. This is especially the case when you have no way to indicate that message to the user. The computer user presented with one too many "low memory" warning dialog boxes will try rebooting, and so will the system administrator who spots the server thrashing its disk as it pages memory out to the hard drive to increase the amount of virtual memory. On the other hand, an embedded platform with no screen or other indicators will usually continue blindly until it runs out of memory completely—at which point it usually "indicates" this situation to the user by mysteriously ceasing to function.

### TYPES OF MEMORY

1. Rom
2. Flash
3. Ram

## PERFORMANCE AND BATTERY LIFE

When it comes to writing code, performance and battery life tend to go hand in hand—what is good for one is usually good for the other. Whether either or both of these are things that you need to optimize depends on your application. A device which is tethered to one place and powered by an AC adaptor plugged into the wall isn't as reliant on energy conservation, for example. However, consuming less energy is something to which all devices should aspire.

## LIBRARIES

These days, when developing software for server or desktop machines, you are accustomed to having a huge array of possible libraries and frameworks available to make your life easier. Need to parse a chunk of RSS XML? No problem. Just pull in the RSS parsing library for your language of choice.

here are a few which might be of interest:

▪ **lwIP:** lwIP, or LightWeight IP, is a full TCP/IP stack which runs in low-resource conditions. It requires only tens of kilobytes of RAM and around 40KB of ROM/flash. The official Arduino WiFi shield uses a version of this library.

▪ **uIP:** uIP, or micro IP, is a TCP/IP stack targeted at the smallest possible systems. It can even run on systems with only a couple of kilobytes of RAM. It does this by not using any buffers to store incoming packets or outgoing packets which haven't been acknowledged. This means that some of the retransmission logic for the TCP layer bleeds into the application code, making your code more tightly coupled and more complex. It's quite common on Arduino systems which don't use the standard Ethernet shield and library, such as the Nanode board, using the Ethercard port for AVR.

▪ **uClibc:** uClibc is a version of the standard GNU C library (glibc) targeted at embedded Linux systems. It requires far fewer resources than glibc and should be an almost drop-in replacement. Changing code to use it normally just involves recompiling the source code.

▪ **Atomthreads:** Atomthreads is a lightweight real-time scheduler for embedded systems. You can use it when your code gets complicated enough that you need to have more than one thing happening at the same time (not quite literally, but the scheduler switches between the tasks quickly enough that it looks that way, just like the multitasking on your PC).

- **BusyBox:** Although not really a library, BusyBox is a collection of a host of useful UNIX utilities into a single, small executable and a common and useful package to provide a simple shell environment and commands on your system.

## DEBUGGING

One of the most frustrating parts of writing software is knowing your code has a bug, but it's not at all obvious where that bug is. In embedded systems, this situation can be doubly frustrating because there tend to be fewer ways to inspect what is going on so that you can track down the issue.

Building devices for the Internet of Things complicates matters further by introducing both custom electronic circuits (which could be misbehaving or incorrectly designed) and communication with servers across a network. Troubleshooting electronic circuits is outside the scope of this book, but we'll cover some ways to debug the network communication.

## Question Bank:

1. Explain prototyping online components. (10M)
2. Explicate techniques for writing embedding code. (10M)
3. Describe API. (5M)
4. Illustrate memory management. (5M)
5. Elaborate libraries. (5M)
6. Illuminate performances and battery life. (5M)
7. What is Debugging? (5M)

**PAPER NAME : INTERNET OF THINGS** **PAPER CODE : 17PCSE06**

**UNIT – V**

## BUSINESS MODELS

### BUSINESS MODELS

If you are primarily a maker or a programmer, and not an entrepreneur, you may have only a dim idea of what a "business model" is. In casual discussion, this expression seems to refer almost exclusively to how the business makes money.

This definition brings together a number of factors:

- A group of people (customers)
- The needs of those customers
- A thing that your business can do to meet those needs
- Organizational practices that help to achieve this goal—and to be able to carry on doing so, sustainably
- A success criterion, such as making a profit

All these aspects are relevant as much to hobbyist or not-for-profit projects as they are to commercial enterprises, though for the last point profit might be substituted for "improving the world" or "having fun" as criteria for success.

### HISTORY OF BUSINESS MODELS

#### SPACE AND TIME

While neighbouring tribes might have discovered variants in the local area's resources—animal, vegetable or mineral—it is when trade develops with others from far-off lands that it becomes really interesting. A merchant might sell silks made in his village to a region where these cloths are rare and in demand in exchange for aromatic spices which will be highly prized back home. But long-distance trade brings with it a whole set of problems: while nomadic hunter-gatherers were adept at finding food and making a home on the move, merchants have to carry larger quantities of goods for sale and want to maximize the time travelling rather than doing the myriad tasks required for subsistence and shelter.

#### FROM CRAFT TO MASS PRODUCTION

When Gutenberg demonstrated his printing press circa 1450, books changed from being priceless treasures, hand-crafted by monks and artisans, to a commodity that could be produced. Soon every bourgeois family could afford their own books, at least a copy of the Gutenberg Bible, the first mass-produced book. It is no exaggeration to suggest that the invention laid the foundations for an information culture which is currently exemplified by the Internet and the World Wide Web.

THE LONG TAIL OF THE INTERNET

As we have seen, huge changes in business practice are usually facilitated by, or brought about as a consequence of, technological change. One of the greatest technological paradigms shifts in the twentieth century was the Internet. From Tim Berners-Lee's first demonstration of the World Wide Web in 1990, it took only five years for eBay and Amazon to open up shop and emerge another five years later as not only survivors but victors of the dot-com bubble. Both companies changed the way we buy and sell things. Chris Anderson of Wired magazine coined and popularized the phrase "long tail" to explain the mechanism behind the shift.

## LEARNING FROM HISTORY

We've seen some highlights of business models over the sweep of human history, but what have we learnt that we could apply to an Internet of Things project that we want to turn into a viable and profitable business,

- First, we've seen that some models are ancient, such as Make Thing Then Sell It. The way you make it or the way you sell it may change, but the basic principle has held for millennia.
- Second, we've seen how new technologies have inspired new business models. We haven't yet exhausted all the new types of business facilitated by the Internet and the World Wide Web.… If our belief that the Internet of Things will represent a similar sea change in technology is true, it will be accompanied by new business models we can barely conceive of today.
- Third, although there are recurring patterns and common models, there are countless variations. Subtle changes to a single factor, such as the manufacturing process or the way you pay for a product or resource, can have a knock-on effect on your whole business.

## THE BUSINESS MODEL CANVAS

One of the most popular templates for working on a business model is the Business Model Canvas by Alexander Osterwalder and his startup, the Business Model Foundry. The canvas is a Creative Commons–licensed single-page planner.

At first sight, it looks as though each box is simply an element in a form and the whole thing could be replaced by a nine-point checklist. However, the boxes are designed to be a good size for sticky notes, emphasizing that you can play with the ideas you have and move them around. Also the layout gives a meaning and context to each item.

Let's look at the model, starting with the most obvious elements and then drilling down into the grittier details that we might neglect without this kind of template.

At the bottom right, we have Revenue Streams, which is more or less the question of "how are you going to make money?" we used to start this chapter. Although its position suggests that it is indeed one of the important desired outputs of the business, it is by no means the only consideration!

The central box, Value Propositions, is, in plainer terms, what you will be producing—that is, your Internet of Things product, service, or platform.

WHO IS THE BUSINESS MODEL FOR?

Primarily, the reason to model your business is to have some kind of educated hypothesis about whether it might deliver what you want from it. Even if you don't use a semi-formal method like the canvas we just discussed, anyone who starts up any business will have thought, at least briefly, about whether she can afford to do it, what the business is, and whether she'll get paid.

As a programmer or a maker, you might believe it counterintuitive to think of a piece of paper with nine boxes in it as a "tool", but when you have a well-tested separation of factors to consider, the small amount of structure the canvas provides should help you think about the business and give you ways to brainstorm different ideas:

- What if we target the product at students instead of businesses?
- What if we outsource our design to an agency?
- What if we sell at low volume/high value instead?

Let us look at some of these likely questions, from the wider field of Internet products in general.

▪ Why should I waste time trying out Yet Another Social Network? I think I'll wait and see whether all my friends join it first. This first question is about your "Value Proposition" (that is, the product) and a reasonable concern if you are trying to get into a market that already has good or popular solutions.

▪ Will my Internet-connected rabbit become an expensive paperweight if you go bust? This happened with Nabaztag, one of the earliest consumer products in the field of Internet of Things. These rabbit shaped devices delighted their owners by muttering and moving their ears in response to stimuli received via the Internet until the French company Violet went bankrupt. The new owners, Mindscape, factored in this concern by open sourcing the code for Nabaztag (and its successor Karotz) to ensure that customers can continue to use the product no matter what happens to the company. This question is asked with a degree of consumer savvy about business risk. Potential customers have seen other companies fall under and don't want the inconvenience or waste it entails for them.

▪ Your online document collaboration looks great, but is it worth my moving my whole business to it? If you stop trading or change the platform, we may have to redo all the work again. Such customers may well be interested in the details of your business model to calculate whether the risk they've identified is worth their commitment. This isn't just a concern about viability of the company: It's unlikely that Google will stop trading, yet many businesses are still unwilling to rely on Google Drive for editing documents. The reason is, in part, that they don't understand where the product fits in Google's strategy and can't guarantee that the service won't be discontinued or crippled, or predict a cost structure for it in the event that it stops being free.

▪ This free service is fantastic, but why don't you let me pay for it, so I can get consistency, receive support, and avoid adverts? Lastly, many customers are aware of alternative charging models that they would prefer and might prefer a different one. Not all customers vote for the free option.

**MODELS**

We have looked at the Business Model Canvas as a tool for generating and analyzing models. As we saw from our history, the models have many common variants. It is a good idea to have a look at some of the models that Internet of Things companies have used or might use and consider some of the parameters these models relate to on the canvas.

- make thing, sell thing
- subscriptions
- customization
- be a key resource
- provide infrastructure: sensor networks
- take a percentage

## FUNDING AN INTERNET OF THINGS STARTUP

As important as future costs and revenues are to a well-planned business model, there will most likely be a period when you have only costs and no income. The problem of how to get initial funding is a critical one, and looking at several options to deal with it is worthwhile.

### HOBBY PROJECTS AND OPEN SOURCE

If your project is also your hobby, you may have no extra costs than what you would spend anyway on your free-time activity. This is perfectly valid, although if you are reading this chapter with an intent to turn your product into a successful business or community, you may wish to proceed at a less leisurely pace than a pure hobby might entail.

Some benefits from the relationship with the community:

- Many pairs of eyes and hands testing, reporting problems, fixing them, and building new features.

- Many passionate users with real use cases and opinions about the product—better than any focus group.

- The goodwill of that community, with its ready-made network of personal recommendations and social-media marketing.

### VENTURE CAPITAL

The venture capital (VC) round is similar, but instead of your courting individual investors, the investor is a larger group with significant funds, whose sole purpose is to discover and fund new companies with a view to making significant profit. VCs may be interested if angels have already funded you and will certainly be interested if other VC companies are already looking at funding you. VCs will certainly want equity, probably a significant amount of it, and a position on your board of directors. Again, this last role may be as much to help fill gaps that your management team don’t cover as much as it is to keep an eye on you and their money. Typically, VC funding will be larger chunks of money, from half a million pounds up.

Current accelerators that may be specialized in the Internet of Things, or cover the field as part of their area of interest, include

- HAXLR8R

- PCH Accelerator

- Berlin Hardware Accelerator

- Bolt

- Lemnos Labs

Typically, you have only two exits:

- You get bought by a bigger company
- You do an IPO (initial public offering)—that is, float on the stock market

GOVERNMENT FUNDING

- Outputs
- Spending constraints

CROWDFUNDING

We've already looked at the long tail as a business model; we can think of crowdfunding as the long tail of funding projects. Getting many people to contribute to a project isn't exactly a new phenomenon.

**LEAN STARTUPS**

We've looked at the advantages of running a startup on a low budget. The mentality needed to do this includes spending time and money only when it's really necessary—staying hungry and lean. The concept of a "lean startup," pioneered by Silicon Valley entrepreneur Eric Ries, springs from this idea (The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, 2011). The option in the preceding section of crowdfunding a project presented an even more appealing step on this route: running the project only if there is a demonstrable niche market for it.

For instance:

- **Zoom-in pivot:** Focus on what was only a part of the value proposition, and turn that into the whole Minimum Viable Product.
- **Customer segment pivot:** Realize that the people who will actually buy your product aren't the ones you were originally targeting. While you can continue to make exactly the same product, you have been marketing it to the wrong people.
- **Technology pivot:** Accomplish the same goals as before, but change the implementation details. While prototyping will almost certainly involve many changes in technology while you establish the best way to make the product from an engineering perspective, this pivot would be a business decision, made to improve manufacturing costs, speed, or quality.

## MOVING TO MANUFACTURE

MANUFACTURING PRINTED CIRCUIT BOARDS

Now that you've designed your PCB, the next step is to make one or lots of them. If you want only a couple of boards, or you would like to test a couple of boards (a very wise move) before ordering a few hundred or a few thousand, you may decide to make them in-house.

ETCHING BOARDS

The most common PCB-making technique for home use is to etch the board. Some readily available kits provide all you need.

The first step is to get the PCB design onto the board to be etched. This process generally involves printing out the design from your PCB design software onto a stencil. If you're using photo-resist board, it will be onto a stencil which masks off the relevant areas when you expose it to UV light; or if you're using the toner-transfer method, it will be for your laser printer to print onto glossy paper ready to transfer.

Your stencil then needs to be transferred to the board. For photo-resist board, you will expose it under a bright lamp for a few minutes; and for the toner-transfer method, you'll use a super-hot iron.

With the board suitably prepared, you can immerse it into the etching solution, where its acidic make-up eats away the exposed copper, leaving the tracks behind.

After all the unnecessary copper has been etched away, and you've removed the board from the etching bath and cleaned off any remaining etchant, your board is almost ready for use.

The last step is to drill the holes for any mounting points or through-hole components. You can do this by hand, or, if you have access to a CNC mill, you can export the drill file from your PCB design package to provide the drill locations for your mill.

## MILLING BOARDS

In addition to using a CNC mill to drill the holes in your PCB, you can also use it to root out the copper from around the tracks themselves. To do this, you need to export the copper layers from your PCB software as Gerber files. These were first defined by Gerber Systems Corp., hence the name, and are now the industry standard format used to describe PCBs in manufacture.

To translate your Gerber file into the G-code that your mill needs requires another piece of software. (See Chapter 6 for more on CNC mills and G-code.) Some CNC mills come with that software already provided, or you can use a third-party program such as Line Grinder.

## THIRD-PARTY MANUFACTURING

If your design has more than two layers, if you want a more professional finish, or if you just don't want to go to the trouble of making the PCBs yourself, many companies can manufacture the boards for you.

The price for getting the boards made varies based on the complexity and the size of the design but also varies quite a bit from company to company, so it's worth getting a few quotes before deciding which one to use.

## ASSEMBLY

After your PCBs have been manufactured, you still need to get the components soldered onto them.

If you're selling them as kits, the customers will solder things up, so you just need to pack everything into bags and let them get on with it. Otherwise, you have to take responsibility for making that happen.

For small runs, you can solder them by hand. For through-hole boards, break out your soldering iron. Surface-mount assembly is a little more involved but quite achievable if you don't have any components with particularly complicated package types.

For assembling surface-mount boards, you need one more item from your PCB design Gerber collection: the solder paste layer. You use it to generate a stencil that allows you to apply the solder. You can laser-cut one from a thin sheet of Mylar plastic or have one made for you out of thin steel. Obviously, the steel one will last much longer and let you solder up lots more boards before you need to replace it.

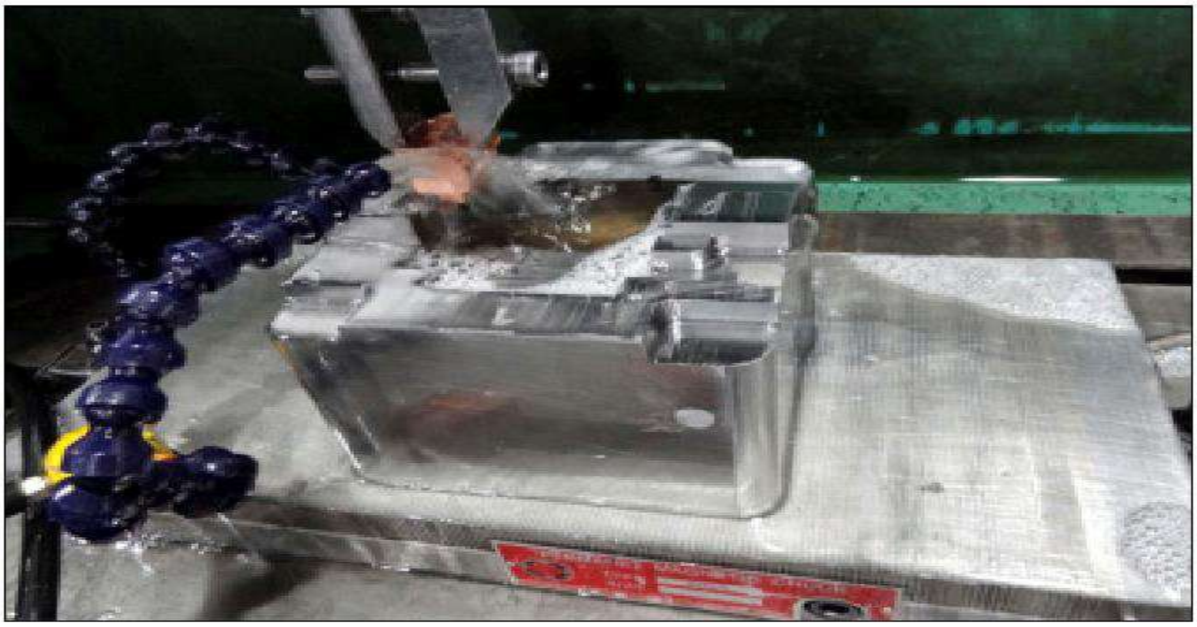
## MASS-PRODUCING THE CASE AND OTHER FIXTURES

We've covered how to scale up manufacture of the electronics side of things, but what about any custom casing or other subassemblies used to build up the final product.

A good rule of thumb for keeping down the costs of production is to minimize the amount of time a person has to work on each item. Machines tend to be cheaper than people, and the smaller the proportion of labour is in your costs, the more you'll be able to afford to pay a decent wage to the people who are involved in assembling your devices.

In "Prototyping the Physical Design", your design uses some of the newer, digital manufacturing techniques such as laser cutting or 3D printing, you might already have little labour in your assembly process.

However, whilst minimizing labour costs is a good target, it's not the only factor you need to consider in your production recipe; production rates are also important. Though they're fairly labour free, 3D printers and laser cutters aren't the fastest of production techniques. Waiting a couple of hours for a print is fine if you just want one, but a production run of a thousand is either going to take a very long time or require a lot of 3D printers.



One of the moulds for BERG's Little Printer being finished on an EDM machine.

The mould also needs to include space for the ejection pins to remove the part after it's made and a route for the plastic to flow into the mould. If you've ever put together a model plane or car, you are familiar with those pathways; they're the excess sprue, the plastic scaffolding that holds each piece together in the kit and that you have to snap away. In assembled products, the parts are naturally removed from the sprue during production.

## CERTIFICATION

One of the less obvious sides of creating an Internet of Things product is the issue of certification. If you forget to make the PCB or write only half of the software for your device, it will be pretty obvious that things aren't finished when it doesn't work as intended. Fail to meet the relevant certification or regulations, and your product will be similarly incomplete—but you might not realise that until you send it to a distributor, or worse still, after it is already on sale.

For the main part, these regulations are there for good reason. They make the products you use day in, day out, safer for you to use; make sure that they work properly with complementary products from other suppliers; and ensure that one product doesn't emit lots of unwanted electromagnetic radiation and interfere with the correct operation of other devices nearby.

You may not have noticed before, but if you take a closer look at any gadget that's near to hand, you will find a cluster of logos on it somewhere…CE, FCC, UL.… Each of these marks signifies a particular set of regulations and tests that the item has passed: the CE mark for meeting European standards; FCC for US Federal Communications Commission regulations; and UL for independent testing laboratory UL's tests.

## COSTS

As we've seen in the rest of this chapter, you have many things to consider as you move to higher volume manufacturing. Unfortunately, lots of them involve sizeable up-front costs. In fact, the further you get into the process, the less you will need your hardcore coding or critical design skills, and the more time you'll spend balancing cash flow and fund-raising.

## SCALING UP SOFTWARE

Producing a physical thing as a prototype or as a manufactured product turn out to be two entirely different propositions. The initial prototype may well be of different size, shape, color, materials, finish, and quality to what ends up on the shelf. Yet software is intangible and malleable. There are no parts to order, no Bill of Materials to pay for. The bits of information that make up the programs which run in the device or on the Internet are invisible. The software you wrote during project development and what ends up in production will be indistinguishable to the naked eye.

## ETHICS

The emotions raised by technology may be strong and suggest two competing grand narratives, moving either from or towards the philosophical ideal of the "good life":

- The downward spiral of mankind from a better state: This could be identified with religious concepts such as the Christian "fall of man" or simply with traditional values.

- An inexorable and definingly human advance towards a full self-realization through technology: This might lead to a new state for the species (the post-human singularity or the spread of mankind to other planets) or simply suggest that every advance leads us to a stable utopia.

### PRIVACY

The Internet, as a massive open publishing platform, has been a disruptive force as regards the concept of privacy. Everything you write might be visible to anyone online: from minutiae about what you ate for breakfast to blog posts about your work, from articles about your hobbies to Facebook posts about your parties with friends. There is a value in making such data public: the story told on the Internet becomes your persona and defines you in respect of your friends, family, peers, and potential employers.

### CONTROL

Some of the privacy concerns we looked at in the preceding sections really manifest only if the "data subject" is not the one in control of the data. The example of the drunken photo is more sinister if it was posted by someone else, without your permission. This is a form of cyberbullying, which is increasingly prevalent in schools and elsewhere.

### DISRUPTING CONTROL

The other major possibility that Eaves suggests is that "The Internet Destroys the State". This is also a hard and uncomfortable scenario to imagine. However, toning down this idea a little, we can see a more likely one of "the Internet" fighting back against an attempt by the state or corporations to co-opt it. When we refer to a technology as "disruptive", we mean that it affects the balance of power.

## ENVIRONMENT

We have already touched on several environmental issues in the preceding sections, and we'll come back to the themes of data, control, and the sensor commons. First, let's look at the classic environmental concerns about the production and running of the Thing itself.

## PHYSICAL THING

Creating the object has a carbon cost, which may come from the raw materials used, the processes used to shape them into the shell, the packing materials, and the energy required to ship them from the manufacturing plant to the customer. It's easier than ever to add up the cost of these emissions: for example, using the ameeConnect API (www.amee.com/ pages/api), you can find emissions data and carbon costs for the life-cycle use of different plastics you might use for 3D printing or injection moulding. Calculating the energy costs for manufacture is harder.

## ELECTRONICS

The electronics contained in a Thing have their own environmental cost. Buying PCBs locally or from a foreign manufacturer affects the carbon cost of shipping the completed units. Considering the potential cost savings, even a responsible manufacturer may find it reasonable to offset the extra carbon emissions.

If your product needs to conform to RoHS legislation, then every single component that could be extracted from it must be RoHS compliant. As we have seen, this is not too onerous.

## INTERNET SERVICE

As Nicholas Negroponte (founder of MIT's Media Lab) preaches, "Move bits, not atoms" (Being Digital, Vintage, 1996). In the digital world, moving data rather than physical objects is faster, is safer, and has a lower environmental cost. Of course, "data" doesn't exist in the abstract. The stone tablets, parchment scrolls, and libraries of paper books or microfiche that have historically been used to store analogue data always had their own environmental cost. Now, running the Internet has a cost: the electricity to run the routers and the DNS lookups, plus establishing the infrastructure—laying cabling across the sea, setting up microwave or satellite links, and so on.

## SOLUTIONS

Compared to a simple, physical object, an instrumented Internet of Things device does seem to use vastly more resources in its production, daily use, and waste disposal. Considering our starting point—that this kind of instrumentation is now cheap enough to put everywhere—it seems as though the mass rollout of the Internet of Things will only contribute to environmental issues! Assuming that you want to go ahead with manufacturing a Thing regardless, we hope that you will be aware of the various possibilities and consider ways to reduce your impact and also consider contributing to offsetting schemes.

## CAUTIOUS OPTIMISM

Between the tempting extremes of technological Luddism and an unquestioning positive attitude is the approach that we prefer: one of cautious optimism. Yes, the Luddites were right—technology did change the world that they knew, for the worse, in many senses. But without the changes that disrupted and spoilt one world, we wouldn't have arrived at a world, our world, where magical objects can speak to us, to each other, and to vastly powerful machine intelligences over the Internet.

When designing the Internet of Things, or perhaps when designing anything, you have to remember two contrasting points:

▪ Everyone is not you. Though you might not personally care about privacy or flood levels caused by global warming, they may be critical concerns for other people in different situations.

▪ You are not special. If something matters to you, then perhaps it matters to other people too.

## THE OPEN INTERNET OF THINGS DEFINITION

The Open IoT Assembly 2012 culminated in the drafting of the "Open Internet of Things Definition". An emergent document, created after two days of open discussion, it seeks to define and codify the points of interest around the technology of the Internet of Things and to underscore its potential to "deliver value, meaning, insight, and fun". This document touches on many of the topics that we discussed in this chapter, so let us walk through some of them to see the conclusions that this more formal treatment has come to.

We can summarize the main goals of the definition as follows:

▪ Accessibility of data: As a stated goal, all open data feeds should have an API which is free to use, both monetarily and unrestricted by proprietary technologies with no alternative open source implementation.

▪ Preservation of privacy: The Data Subjects should know what data will be collected about them and be able to decide to consent or not to that data collection. This is a very strong provision (and most likely unworkable for data which is inherently anonymous in the first instance) but one which would provide real individual protection if it were widely followed. As with any information gathering, "reasonable efforts" should be made to retain privacy and confidentiality.

▪ Transparency of process: Data Subjects should be made aware of their rights—for example, the fact that the data has a license—and that they are able to grant or withdraw consent.

## Question Bank:

1. Explain business models. (10M)
2. Explicate manufacturing. (10M)
3. Explain Ethics. (10M)
4. Describe history of business model. (5M)
5. Illustrate lean startup. (5M)
6. Elaborate designing kits. (5M)
7. Illuminate certification. (5M)
8. What is privacy? (5M)