- ***When simple things need pictures, labels, or instructions, the design has failed.***
- What are the affordances of digitally enhanced objects?
- How do we convey to the user of an object that it can communicate with the cloud?
- An important start is to keep the existing affordances of the object being enhanced.
- **Users who don't realise that a device has any extra capabilities should still be able to use it as if it hasn't**.
- Similar rules apply when designing physical interfaces.
- Don't overload familiar connectors with unfamiliar behaviours.

Chapter 3
INTERNET PRINCIPLES

- **INTERNET COMMUNICATIONS:AN OVERVIEW**
- **IP (Internet Protocol)**
- **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").**
- 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.
- 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.
- 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**?
- 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.

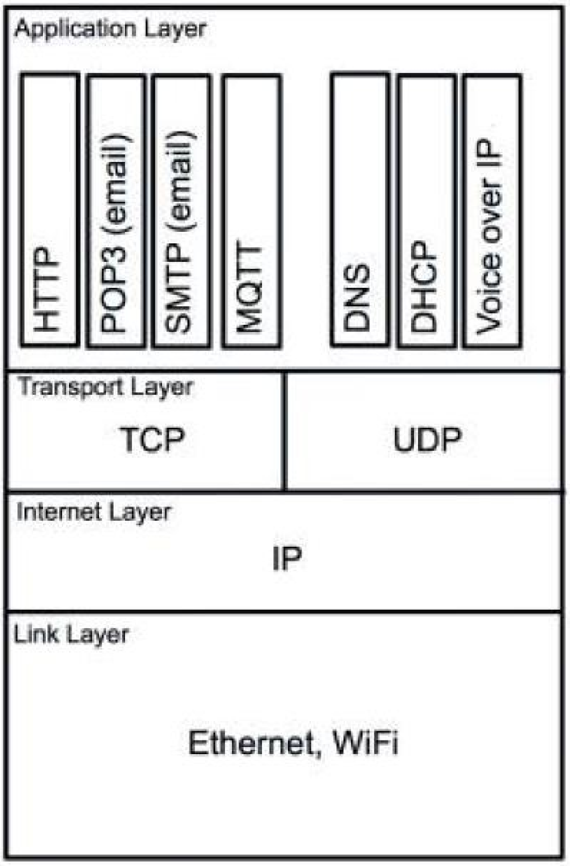
THE IP PROTOCOL SUITE (TCP/IP)

- 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.
- The ***Internet layer uses IP 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.

UDP

- It is protocol in the transport layer.
- **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.



The Internet Protocol suite.

- **IP ADDRESSES:**
- 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**), 2^32 addresses are possible.
- **usually written as four 8-bit numbers separated by dots** (from 0.0.0.0 to 255.255.255.255).
- This "dotted quad" is still exactly equivalent to the 32-bit number.
- Every machine on the Internet has at least one IP address.
- **Your home or office network might have only one publicly visible IP address.**

- **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 brains navigate the Internet.
- Domain names such as the following:
- google.com
- bbc.co.uk
- **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
- **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
- **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.
- With the explosion of the number of devices connecting to the Internet we can use **Classless Inter-Domain Routing (CIDR)**, which allows you to specify exactly how many bits of the address are fixed.
- 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**.
- In many cases, the **system administrator** simply **assigns server numbers in order**.
- He 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.**

- **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, **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.**
- 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

- If your mobile phone, watch, MP3 player, telehealth or sports-monitoring **devices are all connected to the Internet, then you personally are carrying half a dozen IP addresses already.**
- 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, 001:0db8:85a3:0042 :0000:8a2e:0370:7334.
- The address space (2^128).
- You can find many ways to work around the lack of public IP addresses using subnets.
- **IPv6 and Powering Devices**
- We know that **we can regularly charge and maintain a small handful of devices**.
- 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.**

- **MAC ADDRESSES**
- 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.**
- 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.
- (*WizNet is a Korean manufacturer which specialises in networking chips for embedded devices.*)

TCP AND UDP PORTS

- when you send a **TCP/IP message** over the Internet, **you have to send it to the right port**.
- TCP ports 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:
- **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 client has decided that trying to send a message to that port is a bad idea and refuses to do it. (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.**
- **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.
- 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
- 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
- 22 SSH (Secure Shell)
- 23 Telnet
- 25 SMTP (outbound email)
- 110 POP3 (inbound email)
- 220 IMAP (inbound email)

APPLICATION LAYER PROTOCOLS

- This is the **layer you are most likely to interact with** while prototyping an Internet of Things project.
- 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 errors caused by transmission**.
- **HTTP**
- The **client requests a resource by sending a command to a URL, with some headers**.
- Ex: try to get a simple document at http://book.roomofthings.com/hello.txt.
- The basic structure of the request would look like this:
- GET /hello.txt HTTP/1.1
- Host: book.roomofthings.com
- We specified the **GET method** because we're **simply getting the page.**
- We then tell the server **which resource we want** (/hello.txt) and **what version of the protocol we're using**.
- 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**.
- Accept: text/html,application/xhtml+xml, application/ xml;
- Accept-Charset: UTF-8
- Accept-Encoding: gzip
- Accept-Language :en-US
- The **Accept- headers** tell the server what kind of content the client is willing to receive and are part of **"Content negotiation"**.
- Finally, the server sends back its response.
- The server replies, giving us a 200 status code (which it summarizes as "OK"; that is, the request was successful).

HTTPS: ENCRYPTED HTTP

- **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**.
- 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.
- When that's established, both sides just speak HTTP to each other as before!
- *Diffie–Hellman (D-H) key exchange is a way for two people to exchange cryptographic keys in public.*
- *without an eavesdropper being able to decode their subsequent conversation.*
- *This is done by each side performing mathematical calculations which are simple to do but not to undo.*
- *Neither side ever sends their own secret key unencrypted, but only the result of multiplying it with a shared piece of information.*

Chapter 4
THINKING ABOUT PROTOTYPING

- 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 **prototype** 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.
- At the end of this stage, you'll **have an object that works.**
- **it's a *demonstrable* product** that you can use to convince yourself, your business partners, and your investors.
- 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).**

- **SKETCHING**
- 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.**

FAMILIARITY
- If you **can already program 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**.

COSTS VERSUS 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**.
- It is beneficial if you can **choose a prototyping platform in a performance/ capabilities bracket similar to a final production solution**.
- That way, **you will be less likely to encounter any surprises over the cost**.
- **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.
- **if your device has physical interactions , you will find that a PC is not optimized for this kind of work.**
- **An electronics prototyping board**, unsurprisingly, *is* **better suited to this kind of work.**
- An important factor to be aware of is that the **hardware and programming choices you make will depend on your skill set**.
- **For many beginners to hardware development, the Arduino toolkit is a surprisingly good choice.**
- 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**.
- And the language is C++.
- (A **general-purpose input/output** (**GPIO**) is an uncommitted digital signal pin on electronic circuit board whose behavior—including whether it acts an input or output—is controllable by the user at run time.)

- The **IDE pushes the compiled code onto the device where it just runs, automatically, until you unplug it.**

Case Study: Bubblino

- Its original purpose was precisely to demonstrate "how to use Arduino to do Internet of Things stuff".
- So the original **hardware connected an Arduino to the motor** for an off-the-shelf bubble machine.
- The original prototype **had a Bluetooth-enabled Arduino, which was meant to connect to Nokia phone,** which was **programmed with Python**.
- The **phone did** the hard **work of connecting to the Internet** and simply **sent the Arduino a number, being the number of recent tweets**.
- **Bubblino responded by blowing bubbles for that many seconds**.
- The **current devices are based on an Arduino Ethernet**.
- This means that the **Twitter search and XML processing are done on the device**, so **it can run completely independently of any computer, as long as it has an Ethernet connection**.
- **In a final** twist, the concept of Bubblino has been **released as an iPhone app, "Bubblino and Friends"**, which simply **searches Twitter for defined keywords and plays an animation and tune.**
- A kit such as an Arduino easily connects to a computer via USB, and you can speak to it via the serial port in a standard way in any programming language.

PROTOTYPES AND PRODUCTION

- *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 the first prototype you built on a PC, iPhone
- 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.
- 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.

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**.
- **digital fabrication** tools can allow each item to be slightly different, **letting you personalise each device in some way.**

- (*mass personalisation*, as the approach is called, means you can **offer something unique**).

CLIMBING INTO THE CLOUD

- The server software is the easiest component to take from prototype into production.
- 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

- 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**

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.
- You might also be able to **protect distinctive elements of the visual design with trademarks and of the software and hardware with patents**.
- **Note that starting a project as closed source doesn't prevent you from later releasing it as open source.**

WHY OPEN?

- In the open source model, you release the sources that you use to create the project to the whole world.
- 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.
- 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**.

- **Disadvantages of Open Source**
- deciding to release as open source may take more resources.
- 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.
- 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.
- **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.

Being a Good Citizen:

- 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.

Open Source as a 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**.
- Second, using open source aggressively gives your product the **chance to gain mindshare**.
- (ex: Arduino : one could easily argue that it isn't the most powerful platform ever and will surely be improved.)
- 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.**

Open Source as a Strategic Weapon:

- 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.**
- While open sourcing your core business would be risky indeed, trying to standardise things that you use but which are core to *your competitor*'s business may, in fact, **help to undermine that competitor.**
- So **Google releasing Android as open source could undermine Apple's iOS platform**.

- **With the Internet of Things** because **several components in different spaces interact to form the final product: the physical design, the electronic components, the microcontroller, the exchange with the Internet, and the back-end APIs and applications**.
- **MIXING OPEN AND CLOSED SOURCE**
- We've discussed open sourcing many of your libraries and keeping your core business closed.
- 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.
- Server code is closed source.

CLOSED SOURCE FOR MASS MARKET PROJECTS

- **a project might be not just successful but *huge***, that is, a mass market commodity.
- 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.
- 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.

TAPPING INTO THE COMMUNITY

- While thinking about which platform you want to build for, having a community to tap into may be vital or at least useful.
- **If you have a problem with a component or a library, or a question about how to do something you could simply do a Google search on the words** "arduino servo potentiometer" and find a YouTube video, a blog post, or some code.
- 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.**
- **When you are an inexperienced maker, using a platform in which other people can mentor you is invaluable.**
- **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**.
- 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.