Unit 6: Networks

1. Internet communication theory

Learning objectives

1.2.3 Internet principles of operation

- show understanding of the role of the browser
- show understanding of the role of an Internet Service Provider (ISP)
- show understanding of what is meant by hypertext transfer protocol (http and https) and HTML
- distinguish between HTML structure and presentation
- show understanding of the concepts of MAC address, Internet Protocol (IP) address, Uniform Resource Locator (URL) and cookies

Introduction

- Read: Sections 2.4 of iGCSE Computer Science by Watson & Williams

Discussion & learning items

Watch Computer networks part 1/3: Introduction to networking theory (12m) and answer the following

- What is a LAN?
- What problem does the MAC address solve in Ethernet networks?
- What is collision detection?
- How does Ethernet fix collision detection?
- What role does a network switch play in an Ethernet network?
- In what way does message switching improve fault tolerance?
- Using packets is a solution to what problem?
- Packet switching was originally developed because of what historical concern?

Watch Computer networks part 2/3: The internet (12m) and answer the following

- LAN stands for?
- WAN stands for?
- ISP stands for?
- Which program on your computer will allow you to monitor the hop counts for your traffic?
- What are the two key parts of an IP packet? IP by itself is insufficient for what reason?
- How does UDP help resolve the above problem?
- How does a checksum work?
- UDP has two key problems for applications where data must get through, such as Email. Those two problems are?
- What additional features does TCP have over UDP?
- What type of applications don't really benefit from the ACK process of TCP?
- DNS stands for?
- What reason was the DNS established?
- Describe the domain structure used to help keep the DNS efficient

(You can stop at 9:50 - Don't worry about the layer stuff she mentions at the end)

Watch Computer networks part 3/3: The world wide web (12m) and answer the following:

- URL stands for?
- HTTP stands for?
- What HTTP status code indicates everything is ok?
- What are the two key tasks of a web browser?
- To r_____ pages and media
- To r_____ the content being returned
- Who created URL, HTTP, HTML?
- What are the three core components of a simple search engine?
- What was unique about Google's approach compared to the early search engines?

(Until 9:20 - You can, but don't have to watch, the net neutrality section)

Supplemental

URL - Uniform resource locator.

> Using the following: **`http://www.example.com/folder/index.html`**

- A URL consists of several parts -- the protocol, the domain, subdirectory, and (optionally) requested file..
- **http -** refers to the **protocol** in use. This is generally either http (hypertext transfer protocol) or https (hypertext transfer protocol secure), though it can also be other protocols such as ftp (file transfer protocol). Some documents may refer to this as a **scheme** instead of protocol.
- **www.example.com** - refers to the **domain** of the web server.
- **folder/** - refers to the **folder** on the web server containing the requested document.
- **Index.html** - the name of the requested file. If this is not included, the web server would have a default file name it would look on it's filesystem for. For **nginx** and **apache** web servers (the two most common), the default file name is usually set to **index.html**.

Exercises

What is the MAC address of your computer? (instructions available here)

Use traceroute to find the IP address of pbaumgarten.com (instructions follow)

Traceroute on Windows
1. Press the Start Button
2. Type "CMD" and press "Enter"
3. In the Command Prompt type "tracert pbaumgarten.com"

Traceroute on Mac
1. Click on the "Go" drop down menu
2. Click on "Utilities"
3. Open Terminal
4. Type "traceroute pbaumgarten.com"

2. Data transmission methods

Learning objectives

1.2.1 Data transmission

- show understanding of what is meant by transmission of data
- distinguish between serial and parallel data transmission
- distinguish between simplex, duplex and half-duplex data transmission
- show understanding of the reasons for choosing serial or parallel data transmission
- ~~show understanding of the need to check for errors~~
- ~~explain how parity bits are used for error detection~~
- show understanding of the use of serial and parallel data transmission, in Universal Serial Bus (USB) and Integrated Circuit (IC)

Introduction

Read: Sections 2.2 of iGCSE Computer Science by Watson & Williams

Discussion & learning items

Some definitions

Simplex data transmission - Communication between devices is in one direction.

Examples:

- Garage door opener
- Wireless microphone
- TV, radio signals from broadcaster to your receiver
- TV remote control
- Keyboard, mouse

Half-duplex data transmission - Signals travel in both directions but only one transmits at a time.

Example:

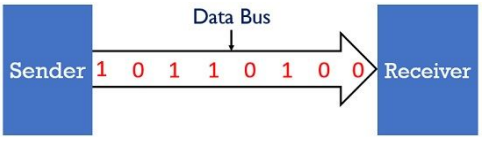
- Two way radios “walkie-talkie”

Duplex (or Full Duplex) data transmission - Signal travels in both directions simultaneously.

Example:

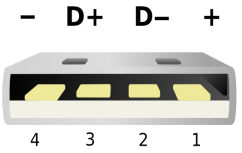
- A phone call

Serial data transmission



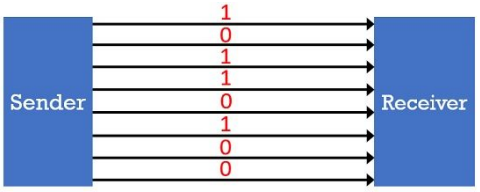
The idea behind serial data transmission is that one bit is transmitted at a time in a continuous flow.

In a modern serial interface such as USB (universal serial bus), for instance, there are only 4 wires (see diagram). Two wires supply power, and two wires for the data transmission. That is all that is required.



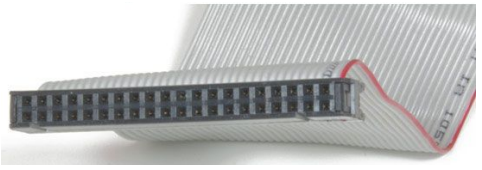
- One bit is sent per clock pulse.
- Serial interfaces are simpler, therefore cheaper to setup
- USB 1 & 2 were half-duplex, whereas USB 3+ is capable of full-duplex

Parallel data transmission



Contrasting serial is the idea of parallel data transmission.

Parallel transmission involves sending multiple bits per clock pulse. For example, if there are eight data wires, an entire byte can be transmitted per pulse instead of just one bit with serial. In theory this means parallel interfaces should be much faster than serial.



The increased complexity of parallel interfaces means they are more expensive to implement than serial. But this is not an "impossible to overcome" problem, so it raises the question: Why is it that serial based communications (such as USB) are so dominant at the moment? Wouldn't parallel make better sense to be faster?

Some reasons why serial is more widely used:

Overhead: In parallel transmission *n* bits are transferred simultaneously, hence we have to process each bit separately and line up them in an order at the receiver. Therefore the parallel gets converted to serial form anway. This is known as overhead in parallel transmission. 1

Synchronisation: In the parallel communication, *n* bits leave at a time, but may not be received at the receiver at the same time, some may reach later than others. This is caused by slight differences in path length and medium electrical properties can cause the bits of the parallel data lines to no longer line up with each other (fractions of a millimeter do make a difference when we are talking about sending megabits per second!). To overcome this problem, the receiving end has to synchronize with the transmitter and must wait until all the bits are received. This would mean that parallel transmission must be run slower than equivalent serial transmission to ensure that all the bits line up.

Crosstalk: When wires lie parallel to each, the signal in some particular wire may get attenuated or disturbed due to electrical induction. In other words… the electromagnetic interference between the signals can occur.

For more detail, https://www.youtube.com/watch?v=myU2x27FIIc (5:20)

Exercises

Because they are so cheap and easy to set up, serial interfaces are very common for electronic components. The Arduino, Raspberry Pi and Circuit Python boards all have multiple pins capable of interfacing with serial based electronic sensors. Commonly used serial protocols include SPI, i2C and UART.

FUTURE LESSON NOTE:
When the world is up and running again, insert an activity here to use serial sensors with the microcontrollers.

https://maker.pro/arduino/tutorial/common-communication-peripherals-on-the-arduino-uart-i2c-and-spi

3. Error detection & connection methods

Learning objectives

- 1.2.1 - show understanding of the need to check for errors
- 1.2.1 - explain how parity bits are used for error detection
- 1.1.3 - identify and describe methods of error detection and correction, such as parity checks, check digits, checksums and Automatic Repeat reQuests (ARQ)

Introduction

Read: Sections 2.3 of iGCSE Computer Science by Watson & Williams

Discussion & learning items

Network based communications are very error prone. It is amazing we have developed a world where high bandwidth data networks are so ubiquitous given all the environmental and technical challenges involved in getting it to work so seamingly seamlessly!

Electrical interference, damaged cables, weather, radio signals - can all interfere in data transmission over wire based or wireless based methods. The following is an overview of some common error detecting and correcting methods used.

Parity bits

Parity bits are used as the simplest form of error detecting code. Parity bits are generally applied to the smallest units of a communication protocol, typically 8-bit octets (bytes), although they can also be applied separately to an entire message string of bits.

7 bits of each byte are used for data, 1 bit is used for a parity check. The parity bit is usually the most significant bit (the left most of the 8).

An agreed protocol is established between the sender and receiver as to odd or even parity.

Odd and even parity

- Even parity - the sender will set the parity bit will be set to ensure an even number of 1 bits.
- Odd parity - the sender will set the parity bit will be set to ensure an odd number of 1 bits.

The receiver will check the byte received to ensure parity matches. If not, there was an error in transmission.

Check digits

We've looked at check digits already. This is where an algorithm is used to generate a digit that is added to the data to check (verify) it has been correctly received. Examples we looked at were credit card numbers, HKID numbers and ISBN numbers.

Most common check digit algorithms are somewhat based on the idea of using a rule similar to the following though you will remember credit cards and HKID used a slightly more complicated algorithm.

```
Check digit = (the sum of the digits) % 10
```

Check sum

Note: The "algorithm" given in "example 3, figure 2.15, page 21" of the book is dumb.

Involves adding extra data that can be used to verify the successful transmission of the message. Typically this will be one byte, but it could just as easily be 16 bit, 32 bit or 64 bit depending on the protocol. Calculating the checksum for data is straightforward. The following assumes we are using bytes.

```
Checksum = (Sum the values of all bytes being transmitted) % 256
```

For example, consider the following message:

Text	`To infinity and beyond.`
Hex	`54 6F 20 69 6E 66 69 6E 69 74 79 20 61 6E 64 20 62 65 79 6F 6E 64 2E`
Bytes	`01010100 01101111 00100000 01101001 01101110 01100110 01101001 01101110 01101001 01110100 01111001 00100000 01100001 01101110 01100100 00100000 01100010 01100101 01111001 01101111 01101110 01100100 00101110`
Decimal	`84 111 32 105 110 102 105 110 105 116 121 32 97 110 100 32 98 101 121 111 110 100 46`

Summing the values together gives `2159`.

```
2159 % 256 = 111
```

Therefore the checksum will be `6F` hex or `111` decimal.

There is also what is known as the "2s complement checksum". This is the word size minus the checksum. In this case, that would be 256 - 111 = `145` decimal or `91` hex. ("word size" in this sense being the length of each unit of data, which is 8 bits in this example).

ARQs (Automatic repeat requests)

The sender receives an acknowledgment from the receiver end implying that the frame or packet is received correctly before a timeout occurs, timeout is a specific time period within which the acknowledgment has to be sent by the receiver to the sender. If a timeout occurs: the sender does not receive the acknowledgment before the specified time, it is implied that the frame or packet has been corrupt or lost during the transmission. Accordingly, the sender retransmits the packet and these protocols ensure that this process is repeated until the correct packet is transmitted. *

Exercises

Parity bits:

- Calculate the parity bit for the questions in activity 2.2 of the textbook.
- Calculate which bytes have transmission errors in the activity 2.3 of the textbook

Checksum:

What would it take to build a Python program that input a string and generated a checksum value for it?

```
~~~ Checksum calculator ~~~
Enter text:The sky calls to us - Carl Sagan
The checksum is 203 or 0xcb
>
```

Some new Python functions you will find useful:

- `ord()` will give you the ASCII numerical value of any character.
    - For instance `ord('a')` will return `97`.
- `hex()` will give you the hex numerical value of any number.
    - For instance `hex(97)` will return `0x61`.
- `int()` you have used this to convert strings containing decimal numbers to an integer, but it will also optionally allow you to provide a second parameter indicating the number base. Therefore `int(string, base)` will allow you to convert a string of hex numbers to an integer.
    - For instance `int('0x61', 16)` will return `97`.

(by the way - If this takes more than 15 minutes then you are too out of practice with your Python)

4 Internet theory

Learning objectives

Program a simple web server so you can see how all the different elements work together including:

- HTTP requests
- URL components
- How does HTML, CSS and Javascript work
- Cookies
- IP addresses

Introduction

Section ____ of the textbook

Discussion & learning items

HTML

- Hyper text markup language
- Provides structure and presentation information
- Structure - Structure of the document: Headings, paragraphs, tables etc
- Presentation - Stylistic detail such as font, colour. These days this is 99% done through CSS.
- Behaviour - Javascript programming to improve the user interactive experience

Example HTML

```
<html>
    <link href="booyaa.css" rel="stylesheet">
    <body>
        <h1>Mr Baumgarten's rockin' website</h1>
        <img
src="https://pbs.twimg.com/profile_images/1132673339477155840/P0FpPyFg_400x400.png">
        <p>This is just the greatest thing in the world</p>
        <a class="blink mega movin" href="https://pbaumgarten.com">CLICK ME!</a>
        <p>I had too much fun making this horror.</p>
        <p>It's like the 90s all over again.</p>
    </body>
</html>
```

See this HTML in action here, https://paulbaumgarten.github.io/rockin/purehtml.html

Example CSS

```
h1 {
    font-size: 100pt;
}
p {
    color: yellow;
    font-size: 50pt;
}
img {
    width: 200px;
    height: 200px;
    position: absolute;
    top: 0px;
    right: 0px;
}
```

See what different the CSS can make here, https://paulbaumgarten.github.io/rockin/withcss.html

Cookies

- Small file (typically a few kb) on the browsing computer, placed by the web server. The content of the cookie is automatically attached back to any future request made to that server. It facilitates things like "logging on" to a website without having to enter your password every page you visit.
- It doesn't (or shouldn't) store your actual password. The server would typically create a "session number" unique to your visit, which it knows belongs to you (it creates the session number when you do the login). So future times you send that number, it knows it was you.
- So, basically it is used as a number to uniquely identify you amongst all the other visitors to a website.
- In the same way it can be used for convenience (remembering your login), it can also be used to track your browsing habits, curate advertising etc. As with most technologies, the tool itself is not inherently good or evil, it is up to the human to decide how to use it.
- Cookies are generally used to:
    - Store and maintain user preferences on a website
    - Track user behaviour (analytics)
    - Store items in shopping baskets
    - Help advertisers show relevant website adverts
- Cookies are not programs. They cannot perform any operations, they are not viruses or malware.
- Cookies can be disabled in your browser settings, however this could make some websites unusable (e.g. e-commerce).

URL components

- Protocol - http or https
- Domain - uses DNS to convert to an IP address (network address of an individual computer)
- Folder & file requested - if not provided then the server running on the target computer will supply a default page
-

For more on using Flask I do have a series of videos and some notes online:

5, 6. Programming exercise

Learning objectives

Program a simple web server so you can see how all the different elements work together including:

- HTTP requests
- URL components
- HTML responses
- Cookies
- IP addresses

Introduction

Install the following packages if you don't already have them:

- Flask
- flask_session

Discussion & learning items

For more on using Flask I do have a series of videos and some notes online:

- Flask 01: Your first Python web server (7m)
- Flask 02: Sending HTML files, and reading form data (10m)
- Flask 03: Variable substitutions with Jinga2 (6m)
- Flask 04: Uploading files, creating variable routes (12m)
- Flask 05 sessions (13m)
- https://pbaumgarten.com/python/flask.md

Exercises

Python code

```
import flask
from flask_session import Session
from datetime import datetime
import os

app = flask.Flask(__name__, template_folder=".")
app.config['SESSION_TYPE'] = 'filesystem'
# Store session data in a /cache folder
app.config['SESSION_FILE_DIR'] = os.path.join(app.root_path, "cache")
 # Allow up to 1000 sessions
app.config['SESSION_FILE_THRESHOLD'] = 1000
Session(app)

@app.route("/")
def homepage():
    cookies = flask.request.cookies
    user_agent = flask.request.headers['User-Agent']
    origin = flask.request.origin
    remote = flask.request.remote_addr
    print(f"Cookies received: {cookies}")
    print(f"User agent: {user_agent}")
    print(f"Point of origin: {origin}")
    print(f"Remote address: {remote}")
    if "remember" in flask.session:
        dont_forget = flask.session['remember']
    else:
        dont_forget = ""
    return flask.render_template('index.html', data=dont_forget)

@app.route("/remember")
def remember():
    print(flask.request.values)
    if "remember" in flask.request.values:
        dont_forget = flask.request.values['remember']
        flask.session['remember'] = dont_forget
        print(f"I have been asked to remember: {dont_forget}")
    return flask.redirect("/")

if __name__ == "__main__":
    if not os.path.exists(os.path.join(app.root_path, "cache")):
        os.mkdir(os.path.join(app.root_path, "cache"))
    app.run(host="0.0.0.0", port=8080, debug=True)
```

HTML code:

```
<html>
    <h1>Hello world!</h1>
    <hr>
        <p>Last time, you asked me to remember:</p>
        <p>{{data}}</p>
    <hr>
    <form action="/remember" enctype="multipart/form-data">
        <p>Would you like me to remember something for you?</p>
        <div>
            <textarea rows=10 cols=40 name="remember"></textarea>
        </div>
        <input type="submit">
    </form>
    <p>Most impressive, this is!</p>
</html>
```

7. Quiz