cisco

SWINBURNE UNIVERSITY OF TECHNOLOGY

Network Protocol Implementation

8.1 Application Layer Protocols

Text Based Protocols

Many Application Layer Protocols are Text Based. This was originally done for a multitude of reasons

UDP-based protocols instead typically are binary based

- Ease of debugging via direct examination of captured packets
- Ease of testing Protocol implementation by playing against a telnet client
- Text restricts choice of valid characters and so easy to delineate multiple messages
- Easy manipulation and protocol implementation in Perl

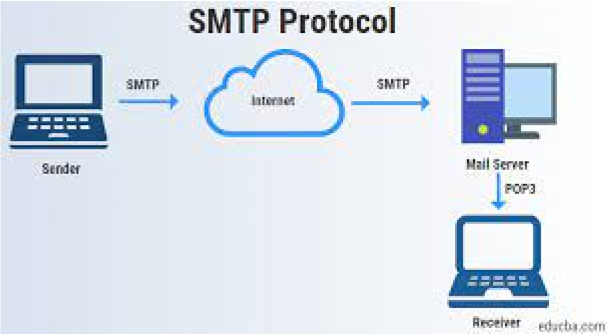
HTTP – Hyper Text Transfer Protocol

Text based protocol to request resources from a web server

- HTTP 1.0 – request/response/disconnect
- HTTP 1.1 – Maintain connection for multiple transactions

- You implemented this in your Lab last week
- Has options to send data to server (PUT/POST) – think Web Forms
- Lets try this using a telnet client

SMTP – Simple Mail Transfer Protocol

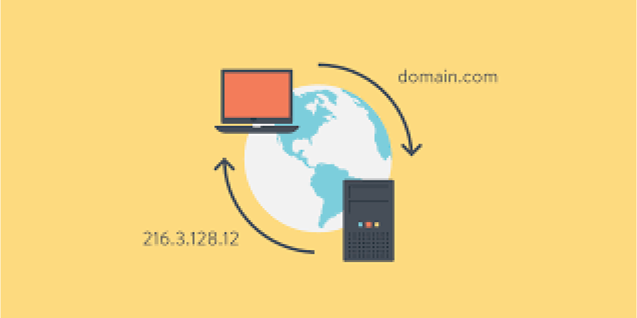


Text based protocol to send an email from a client to the email system

- SMTP is used to send email to final host server
- Email is retrieved from host server using either POP3 or IMAP Protocols

- Lets try this using a telnet client
- Getting harder to find servers that work without encryption – can't implement via telnet

DNS – Domain Name System



Binary – UDP – Protocol to convert a URL to an IP address

- Internet does not work without DNS
- DNS is a distributed database

- Even though it is a distributed server, your local DNS server will manage the query
- If your DNS server cannot provide an answer, it will query other server
- As such, the client implementation is simple, send a query, get a reply
- Lets examine an example via WireShark

8.2 TCP/UDP Considerations for Application Layer Protocols

Stream-Based vs Message-Based

Stream Based

- **TCP is Stream-Based**
- **Guaranteed, in-order delivery**
- **Congestion Control – delayed delivery**
- **Data arrives as a continuous stream**

- A call to **recv()** will not return the same block of data as sent by **send()**
- A call to **recv()** may not return the amount of data requested
- A request for data contains no guarantees:
  - Receive a whole message
  - Receive a single message
- Programmers responsibility to segment stream into delineated messages

TCP/UDP Considerations for Application Layer Protocols

Message Based

- **UDP is Message Based**
- **Best-effort**
- **No Congestion Control – near immediate delivery**
- **Data arrives already segmented**

- A call to **recv()/recvfrom()** will return the same block of data as sent by **send()/sendto()**
- A message sent may not arrive at the sender – programmers responsibility to detect and handle lost messages if important
- Messages sent may arrive out-of-order – programmers responsibility to detect and re-order if important

Designing Application Layer Protocols

Bespoke Protocols are not easy to Design

- Need to consider all eventualities
- How do you handle lost data
- Handle badly formed packets
- Sanitise requests/data
- Corner cases

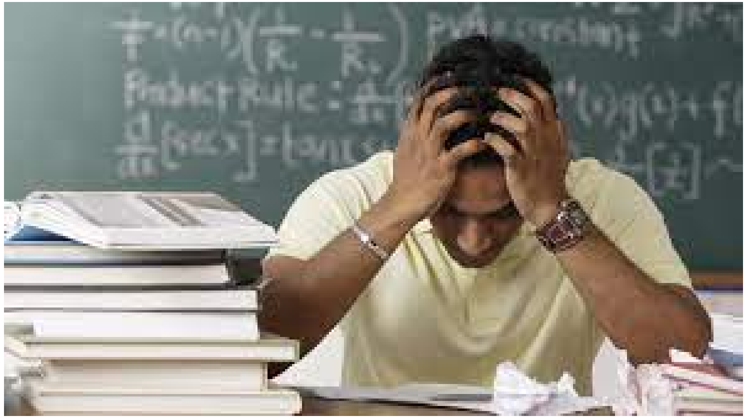
Re-use (extend) Existing Protocols

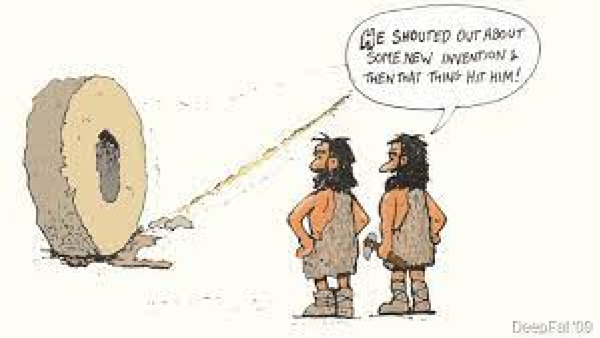
- Existing Protocols may provide what you require (or most of it)
- For example the Internet Printing Protocol (IPP) is essentially HTTP where each printer has a URL on the server and a print job is sent by POST-ing a file to the URL
- Existing protocols typically designed to cover all scenarios

8.3 Coding Considerations

Implementing a Protocol is Hard

- Making sure you cater for all the edge cases of the Protocol is complex
- **It is easy to miss something, or to place code in the wrong spot**
- Testing is also difficult – how do you ensure that you test all pathways through your code
- **Testing involves generating packets from a fake remote system to test your packet management code**
- Just as with Protocol Design, we do not want to re-invent the Wheel if it is not necessary





Use Libraries

Not those type of libraries!!

- Most of the popular protocols have already been implemented as Libraries
- **If somebody else has already implemented the network protocol, use it**
- Focus on making your application work, not on making your Protocol work
- **Let experts make sure that the Protocol is fully implemented**
- Search online for samples and examples

If you modify a Protocol, you may need to modify the Library implementation – this is not always and easy process

8.4 System Design for Networked Engineering Solutions

Application Requirements

How you intend to use the Internet for your Application will drive your solution architecture

- **Direct comms between devices**
- **Client-Server Model**
- **Peer-to-Peer Model**
- **IoT/Cloud Model**

Don’t choose your model to apply for your application

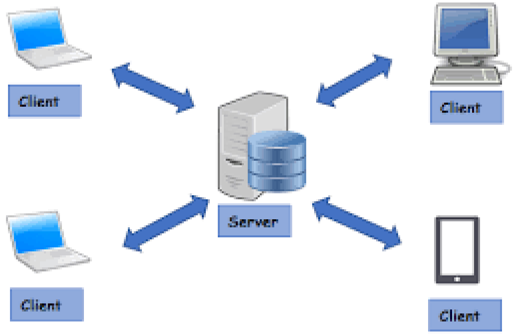
Direct Peer Communications

- **Direct communications means your system will only support sending messages between two pre-configured systems**
- **Establish single TCP/UDP connection and directly communicate**

- Best used for applications where you know:
  - There will only ever be two devices
  - Those devices only communicate state between each other
- An example may be to inform state in a machine between two stages of operation
- Consider TCP or UDP based on requirements – if TCP one host will nominally be the server
- If you ever expect your system to grow should plan for a different approach

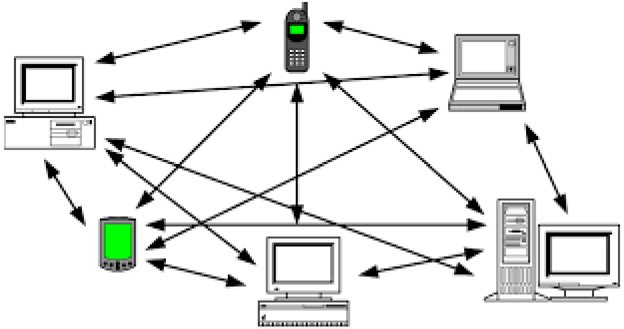
Client-Server Model

- **One part of your solution acts as a Server and central hub managing multiple parts of your system**
- **One system runs a TCP/UDP Server with multiple remote systems connecting as required**



- This is the most common architecture you are likely to deploy
- Your main Engineering solution runs on the server which then obtains state information from remote clients and issues commands for clients to execute
- Easily expandable – You can easily add more nodes to your solution with minimal work thanks to a little extra effort up-front
- Best suited to self-contained solutions where the system Interface is also the server

Peer-to-Peer Model

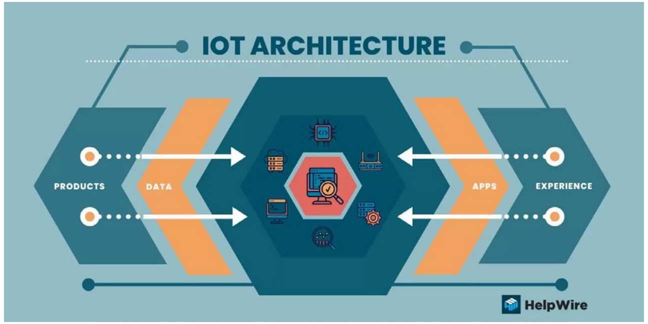


- **Direct communications between all nodes in your solution**
- **Establish multiple TCP/UDP connections at each node – no central Server**

- Most rarely seen/deployed model
- Requires your operation to be truly distributed and require little or no central control
- Difficult to develop and will require careful planning

IoT/Cloud Model

- **A central server acts as a broker between the systems part of your engineering solution and the user interface**
- **Supports addition of multiple application interfaces including mobile and web-based solutions**



- This is what you should be targeting as the best architecture – even if access is to be restricted
- Like the Client-Server model, the intelligence resides in the Cloud but this is not directly connected to the actuator nodes
- In this model the broker sits in the Cloud and:
  - Collates data from system nodes
  - Forwards application requests to system nodes
- The application is also situated in the Cloud and:
  - Provide an interface to the system
  - Manages commands back to the system via the broker
  - Supports multiple application types

Cybersecurity Concerns

- The previous slides have been glib on security
- Most systems you design should be restricted in some form or other
- There are many elements to securing your system, in this Unit we will highlight basic approaches
- Actual solutions presented in TNE30024
- Firewalls may not be enough – easy to fake IP addresses if you know how
- Can't just rely on usernames and passwords if they are sent over the network (particularly the Internet) unencrypted

8.5 Tutorials and Laboratory

Tute – Network Server Examples

SWIN BUR NE SWINBURNE UNIVERSITY OF TECHNOLOGY

A simple TCP-based echo server that echoes back everything sent to it by a remote client:

- Only supports one client at a time

Multi-threaded TCP-based echo server:

- More complete implementation that scales to support 100s of concurrent clients

Internet Chat Server:

- Use of **select()** to avoid multi-threads
- Notifications of users leaving the chat

All source code for these programs provided

cisco

Lab – Network Protocol Programming

In this lab, you will complete the following objectives:

- Write a UDP client and server application to implement a provided protocol
- Simple data transmission and acknowledgement of receipt protocol
- Supports multiple clients communicating data to server

Credit Task:

- Modify Protocol to work in a TCP environment
- Implement TCP client and server implementation

SWIN
BUR
NE

SWINBURNE
UNIVERSITY OF
TECHNOLOGY

cisco

What Did I Learn in this Module?

- The reasons for selecting different Transport Layer Protocol depending on your required application
- The standard interface for applications to communicate with the OS to establish and use network connections
- The basic functions provided for network communications
- The order and process for calling these functions

