COMP2221 Networks

David Head

University of Leeds

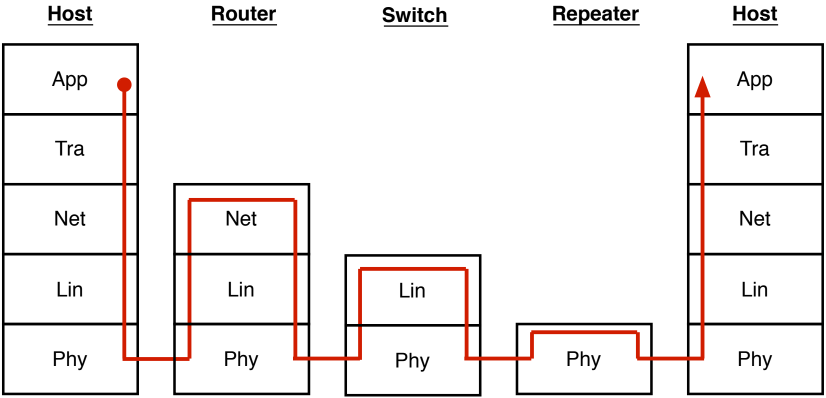
Lecture 20

Today's lecture

Today's lecture is intended to overview and revise the course in preparation for the exam:

- The exam format.
- What to expect in the exam.
- Summarise course material.
- Leave a few minutes at the end for module feedback.

The TCP/IP protocol stack (Lecture 2)



After a general introduction, progressed downwards through the 5-layer TCP/IP **protocol stack**:

- **Application** layer for **single processes** (Lectures 5-15).
- **Transport** layer that provides **ports** to control data transfer between processes (Lectures 3, 16).
- **Network** layer that handles **IP Addresses** for hosts and routers (Lectures 4, 5, 17, 18).
- **Link** layer responsible for data communication between adjacent nodes (Lecture 19).
- **Physical** layer responsible for sending individual bits (mentioned in Lecture 19).

Also described the **7-layer OSI stack** (Lecture 2).

Terminology and addresses

Packets of data are typically given different names for each layer.

All except Physical and Application layers also have a form of 'address'. Source and destination addresses appear in their headers.

Layer	Usual name for packet	Address or similar
Application	Message	-
Transport	Segment[1]	Port
Network	Datagram[1]	IP[2]
Link	Frame	MAC
Physical	-	-

[1] UDP = User **Datagram** Protocol exists in the Transport layer.
[2] Converted to **hostnames** using the Domain Name System.

Need for buffering

Three layers (Transport, Network and Link) **add headers** (and possibly trailers/footers) to messages.

- Sizes depending on the protocol (TCP/UDP; IPv4/IPv6; any number of Ethernet/IEEE protocols), but can be 20-40 bytes per header level.

This means that the message finally sent over the Physical layer can be *much* larger than the Application data.

- *e.g.* `telnet` only sends single characters.

This highlights the importance of **buffering** at the Application level to improve I/O performance.

- Conveniently implemented in Java using **streams** (Lecture 6).

Performance

Other ways to improve performance, in particular for servers:

1. Use **non-blocking I/O** (Lecture 12).
   - Allows a single server thread to deal with multiple clients.
   - Not easy to implement.

2. Use **multi-threading** (Lectures 9-11).
   - Each client handler has its own thread.
   - Easy to implement, especially with `Executor`'s thread pools.
   - Makes good use of modern, multi-core architectures.
   - Strategy applicable to most problems, not just I/O.

UDP

We also looked at the UDP (Lecture 14).

- **Connectionless**, unlike TCP where a connection is maintained until closed.
- **Unreliable**, so packets may be lost or arrive in a different order than sent (again, unlike TCP).

UDP could be considered for **streaming applications**.

UDP can also be used for **multicasting** (Lecture 15):

- Same data sent to multiple receivers without causing congestion.
- Not widely used, although supported by IPv6.

Security

We touched on **security**, an increasing important issue.

- Looked at `SSLSockets` in Lecture 13.
- The basics of **encryption** and **authentication**.
- **Symmetric** key algorithms, which use a **private key**.
- **Asymmetric** key algorithms, which use a private and **public key**.

This important topic is covered fully in the Level 3 module *COMP3911 Secure Computing*.

Lecture summary (1)

Lecture 1	:	Introduction to networks; also admin.
Lecture 2	:	Network architectures; 5 and 7-layer models.
Lecture 3	:	Ports, UDP and TCP; headers for lower levels.
Lecture 4	:	DNS and how it maps names to IP addresses.
Lecture 5	:	IP addresses, IPv4 and IPv6; CIDR and NAT; `InetAddress` in Java.
Lecture 6	:	Java I/O streams, including buffering and filters.
Lecture 7	:	The `Socket` class; construction involves internet access and binding to a port.
Lecture 8	:	The `ServerSocket` class, and how instances listen to ports.

Lecture summary (2)

Lecture 9	:	Parallel and concurrent programming in general.
Lecture 10	:	The Java `Thread` class; synchronisation; thread-per-client servers.
Lecture 11	:	Thread pool servers; `Executor` service.
Lecture 12	:	Non-blocking I/O; `Buffer`, `Channel` and `Selector`.
Lecture 13	:	Network security: Encryption and authentication; the `SSLSocket` class.
Lecture 14	:	UDP: `DatagramPacket` and `DatagramSocket`.
Lecture 15	:	One-to-many communication; multicasting with UDP; the `MultiSocket` class.

Lecture summary (3)

Lecture 16	:	Transport layer: Connection management and congestion control; finite state machines; UDP and TCP headers.
Lecture 17	:	Network layer: More details on CIDR; tunnelling; routers, switching fabrics and (generalised) forwarding tables.
Lecture 18	:	Routing algorithms: Dijkstra's algorithm, RIP and advertisements, hierarchical OSPF and BGP.
Lecture 19	:	Link layer: MAC addresses; multiple access protocols; some Ethernet standards; Physical layer.

Networking: The full picture

Kurose and Ross enumerated the number of steps that are taken 'behind the scenes' when a student connects their laptop to the university network *via* an **ethernet cable**.

- Using Wi-Fi would be similar.

They list **24** steps in total . . . !

A summary is useful at this stage to see how what we have learnt, for all of the various layers, come together for an everyday operation.

1. Device discovery

After connecting, the laptop needs an **IP address**:

- Laptop's OS places a **DHCP request**[1] into a **UDP segment**, itself placed into an **IP datagram**, which is then **broadcast** across the ethernet within a link-layer **frame**.
- Ethernet switch sends this message to all outgoing ports, including the **router** which extracts the request.
- Router returns an **DHCP message** (inside a UDP segment, inside an IP datagram, inside a link-layer frame).
- Laptop extracts DHCP message, its IP address, and the address of the DNS server it will use.

[1] Recall DHCP = Dynamic Host Configuration Protocol; *cf.* Lecture 17.

2. Getting the router's MAC address

The student now types `www.google.com` into their browser, which needs to be converted to an IP address using the DNS:

- Browser creates a **DNS query** and puts it into a **UDP segment** with a destination **port** of 53.
- Sent to the configured DNS server (as always, after placing in an IP datagram inside a link-layer frame).
- Does not yet know the MAC address of the router, so creates an **ARP query**[1] and broadcasts within a link-layer frame.
- Router sends an **ARP reply** (in a link-layer frame) to the laptop's OS.

[1]Recall ARP = Address Resolution Protocol; *cf.* Lecture 19.

3. Getting the IP address

The laptop can now send its DNS query *via* the router.

- Router extracts query (from the frame, datagram, segment) and uses its **forwarding table** to see where to send it.
- Places inside a segment, datagram, frame, and sends.
- Travels to the DNS server *via* various routers (RIP; OPSF; BGP; *cf.* Lecture 18).
- DNS server checks its **cache** and, presuming it is found, sends a **DNS reply** to the laptop (segment/datagram/frame).
- The Laptop's OS extracts the message and the IP address for `www.google.com`.

4. Downloading the web page

- Web browser creates a **TCP socket**, which first performs a **handshake with the web server**.
- Each stage in this handshake uses the port number for the browser, and port number 80 for the server.
- Browser creates an HTTP GET request which is forwarded *via* one or more router(s) to the server.
- The server returns an HTTP response message to the laptop.
- The laptop extracts the information and displays it.

Of course, all messages sent between the browser and the server are placed in a TCP segment, itself placed in an IP datagram, itself placed in a link-layer frame . . .

Finally. . .

Good luck in the exam