11 Application Layer

11.1 Goal of this section

Now that we have a set of transport protocols that provide a service to allow applications to exchange data, what might we want to build on top? This section covers some simple examples of key application layer protocols that you're likely to come across in networking that allow humans to do things over networks.

For the purposes of this course, readers should understand why these protocols exist, and how they work, but do **not** need to understand the details of the individual fields. Readers should be able to read the RFC defining the protocol to understand the definition of the protocol and then by combining that description with the Wireshark capture of the protocol be able to understand themselves how it works.

11.2 Real-time Transport Protocol (RTP) and Real-time Transport Control Protocol (RTCP) – RFC 3550.

11.2.1 Abstract and key points

RTP and RTCP are technically two separate protocols but are defined and used together to allow endpoints to monitor and maintain the quality of a flow of data traffic over a (potentially) lossy network. These are used for real-time data streams (voice and video), for which delay and jitter is important. These run over UDP (lots of data being delivered fast with low delay), but are themselves a transport protocol for data for other applications, so really belong in the last lecture!

RTP packets contain the following information to enable this:

- They contain a timestamp with the generation time of the first byte. This allows the receiver to play the data at the correct rate and enables synchronising multiple matching streams (for example if several video camera feeds and voice feeds are sent in separate streams). Note that these time-stamps are relative to the other packets in the stream – not absolute timestamps.
- They contain source correlators – to enable RTP to distinguish multiple streams from the same source application.
- They have sequence numbers to allow the receiver to detect missing packets.
- They have a payload type to indicate to the receiver how the enclosed data is encoded.

RTCP contains sender and receiver report information that summarises the information in the RTP flow since the last RTCP packet from that endpoint's point of view. This includes information like:

- an absolute timestamp for the RTCP packet being sent (so the receiver can estimate latency)
- the number of bytes of data sent or received
- the number of RTP packets lost (both the total number and the fraction lost)
- the jitter on the RTP packets received.

11.2.2 Example flow

11_RTP_and_RTCP.pcap contains a Voice call. The Voice call is set up using a suite of protocols called H.323 (packets in the trace marked H.225 and H.245) – which you'll see run over TCP. Those messages set up the IP addresses that the data in the call run between and the RTP packets contain that data. There's also an occasional RTCP packet.

11.3 Domain Name Service (DNS)

DNS implements address resolution, allowing conversion from human readable domains (www.google.com) to IP addressing (1.2.3.4). This allows humans and applications to use domain

addresses rather than having to remember IP addresses (and allows indirection so that if a service moves IP, this can be updated in just one place.)

DNS refers to both the protocol on the wire and the infrastructure. Importantly, DNS is responsible for address resolution, not allocation (which is handled separately) – in the same way that ARP handles resolution of IP addresses to MAC addresses but is not responsible for allocating either.

11.3.1 Domains

Domains are hierarchical with sub-domains left of the domain. For example,

- .ac.uk is owned by a central registration organisation
- .ox.ac.uk is owned by Oxford University
- .cs.ox.ac.uk is owned by the computer science dept, servers are .cs.ox.ac.uk
- linux.cs.ox.ac.uk identifies one of the computer science servers.

The structure may appear geographical, but there is no requirement for this structure to match either real geography or the geography of the subnets in the underlying IP networks.

Domain naming has a number of conventions, but there is no DNS significance to domains. (For example: **www.** or **.co.uk** does not mean anything particular to DNS.) Some conventions are very strongly enforced and give us meaning. For example, the organisation that owns .ac.uk will only hand out an .ac.uk subdomain if you can prove that you're an academic institution in the UK.

11.3.2 DNS infrastructure and protocol

The DNS protocol is an application usually running over UDP that operates a simple client-server query response.

The DNS infrastructure is hierarchical in the same manner as domains. A name server (server that contains DNS information) covers a tree of domains, but may delegate details out to subsequent servers.

The client thus asks its local server which asks up the chain to the root server at the top of the tree first, which returns the answer (if it has an exact match) or identifies another server based on the best match that it has. That server does the same thing, until eventually either there is a successful complete match and answer (or an explicit negative response if you query a domain that turns out not to exist).

This potentially causes a huge load on the root and top-level domain servers, and so *actually* lower down name servers cache the results – typically for 86400 seconds (1 day).

11.3.3 Example Flows and Try it yourself

11_DNS_Examples.pcap shows some sample DNS flows querying and resolving different types of addressing. This shows a client looking up a domain name and receiving IPv4 (Type A) and IPv6 (Type AAAA) responses, and doing a reverse lookup on an IPv4 address to receive a domain name. Note that the DNS queries in this case are all running over IPv4 – independently of the information being queried.

You can perform DNS lookups yourself, using the **nslookup** command (Windows/Mac/Linux)

11.4 TELNET

11.4.1 Abstract

Telnet provides a stream of ascii characters, running over TCP, to allow for things like remote command line connections. Telnet character flows are byte flows with characters transmitted as 7-bit ascii characters and the non-ascii space at top and bottom is reserved for special control characters. Key control characters include Authentication, Erase Character/Line, Carriage Return.

Telnet servers listen on well-known port 23.

11.4.2 Try it…

You now have enough understanding of networking to be able to set up a telnet server on your computer – and then connect to it with a telnet client. Both servers and clients exist as standard for ~any OS, and they are straightforward to set up, though exact instructions vary, and you will need to search online for the exact commands. Set one up, connect and type…

Safety and Security Note (also applies to FTP below).

There are two important security concerns to be aware of here.

- What you are doing here (or any time you set up some networking) is creating ways for people to access your computer. Yes, you can and should set up passwords etc. (and most OSs work hard to stop you providing unprotected access) but if you don't need it, don't use it. If you're not going to use it long term, once you've finished playing around with your server – turn it off again.

- Neither Telnet nor FTP (below) inherently provide any encryption, so all commands are transmitted in the clear (unencrypted) making it easy to eavesdrop. This makes it easy for you to snoop in Wireshark and see how these protocols encapsulate things, but mean you should be careful not to transmit any secret information.

See **11_Telnet.pcap** for an example telnet session. The interesting text starts around packet 26. Can you work out the user's login details and what they're doing?

11.5 File Transfer Protocol (FTP)

11.5.1 Abstract

File Transfer Protocol provides a way to push and pull files (which as far as FTP is concerned are just blocks of data) between two applications. It consists of two connections:

- The control connection uses telnet to transfer strings of ascii characters that the FTP application recognises as well-known commands. (For example the string **ls** tells the remote ftp application to send you a list of files (sent as ascii characters over the control channel), and the string **get <filename>** tells the ftp at the other end to send the file <filename> to you over the data connection.)
- The data connection is used for transferring file data.

Note that as with many protocols, FTP doesn't reinvent the wheel and instead just uses an existing protocol where it can. Telnet already provides a way to transfer basic text commands, so FTP just reuses that.

11.5.2 Try it…

Similar to the above with Telnet, it is straightforward to configure and use an FTP server. Congratulations you can now pass files around between computers. Again, beware that you're broadcasting in the clear. We'll note how to fix that in the security section later in the course!

As an example flow, see **11_FTP.pcap**. Again, how hard is it to find the user's login username and password?

11.6 HyperText Transfer Protocol (HTTP)

11.6.1 Abstract

Hypertext is ascii with additional mark-up tags – that allows references to other text and resources that the reader can then access. Web pages are the standard example use case. HTTP allows users to operate on hypertext, accessing it remotely.

11.6.2 HTTP Protocol messages

HTTP is a (another) simple client-server request-response protocol, with a small number of messages. The key protocol messages are as follows:

- **OPTIONS** – The options message allows a client to find out what options there are about a resource on a server, without actually looking at or retrieving the resource.
- **GET** – The get message retrieves a particular resource.
- **HEAD** – Head retrieves information associated with a resource but doesn't actually retrieve it. This is useful, for example, to check the size of a chunk of data in a resource before issuing a GET for it.
- **POST** – This allows the client to pass some information back to the server related to a resource obtained in a previous GET.
- **PUT** – This allows the client to pass resource information to a server (independently of any previous GETs.
- **DELETE** – This allows a client to remove a resource from a server.

The protocol flow for each of these messages is a request from client to server followed by a response from server to client.

Resources and Uniform Resource Locators (URLs)

HTTP identifies and resources using Uniform Resource Locators (URLs). URLs are used to uniquely identify an individual resource *and* how to access it – which might be via HTTP or might be via another protocol.

Some common examples of URLS:

- http://<hostname>/<path_and_file_name> ← This is a resource that you use HTTP to access.
- telnet://<user>@<hostname> ← This is a resource that you use Telnet to access.
- file://<hostname>/<path_and_file_name>
- mailto:<username>@<hostname>

11.6.3 HTTP Response Codes

HTTP responses include a 3-digit code indicating success, failure or more information, and these response codes are grouped by the initial digit. These codes are now commonly used (~consistently) across a whole host of protocols, following the same convention.

- 1XX – Informational.
- 2XX – Success. For example, you will come across "200 OK" responses throughout networking.
- 3XX – Redirect. A redirect indicates that the client should try again elsewhere (with the where information provided.)
- 4XX – Client error. An error that the server believes is the client's responsibility. For example, "404 Not Found" – the client has asked for something that does not exist.
- 5XX – Server error. An error that the server believes is its responsibility, perhaps because some internal process has failed.

Note that HTTP is ubiquitous and slowly becoming the base level networking interaction for application layer networking interactions, with other protocols built on top of it providing additional power and flexibility. There are two common strategies here: REST and SOAP

11.6.4 Representational State Transfer (REST)

Representational State Transfer (REST) makes web services appear to be http objects with identifying URLs. This is a design paradigm as much as a "protocol" where the key point is that the components are stateless. Note that this does not literally mean no state, but rather a requirement that each side must not need to know anything about the state of the other in order to communicate. This allows the client/server can use base HTTP to communicate without needing additional state.

11.6.5 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is an extension to HTTP to allow it to operate on XML, which in turn is leveraged to do "anything" – for example making remote procedure calls with parameters.

11.6.6 Example Flow

11_HTTP.pcap shows a simple HTTP GET request. Note the DNS query kicked off in the middle – it looks like someone is being served up some additional page advertising from a different server to go along with their web page.

11.7 A networking stack recap.

We've now reached the top of the networking stack. Gluing all of this together, you should be able to understand what is going on at all levels underneath you out in the real world as you, the human user, use networked applications. The rest of the sections in the course will cover more general topics relating to networking.