INTERNET SYSTEMS

6CCS3INS

Samhar Mahmoud

samhar.mahmoud@kcl.ac.uk

Hypertext Transfer Protocol (HTTP)

Outline

- Hypertext Transfer Protocol
- Proxies
- Uniform Resource Identifiers
- Requests and responses
- MIME
- Web servers

Hypertext Transfer Protocol

- The Hypertext Transfer Protocol (HTTP) is designed for communicating documents between computers
- It was designed in particular for transmitting hypertext documents, but is used for wider applications
- HTTP is commonly sent over TCP/IP to send and receive messages (but not necessarily)
- The current version is HTTP 1.1

Requests and responses

- HTTP uses the client-server model
- In HTTP, the client sends a **request** to the server and the server returns a **response**
- HTTP clients are commonly web browsers
- The client application sending a request is called the **user agent**
- A host acting as an HTTP server is called a **web server**
- 'Web server' also refers to HTTP-handling software

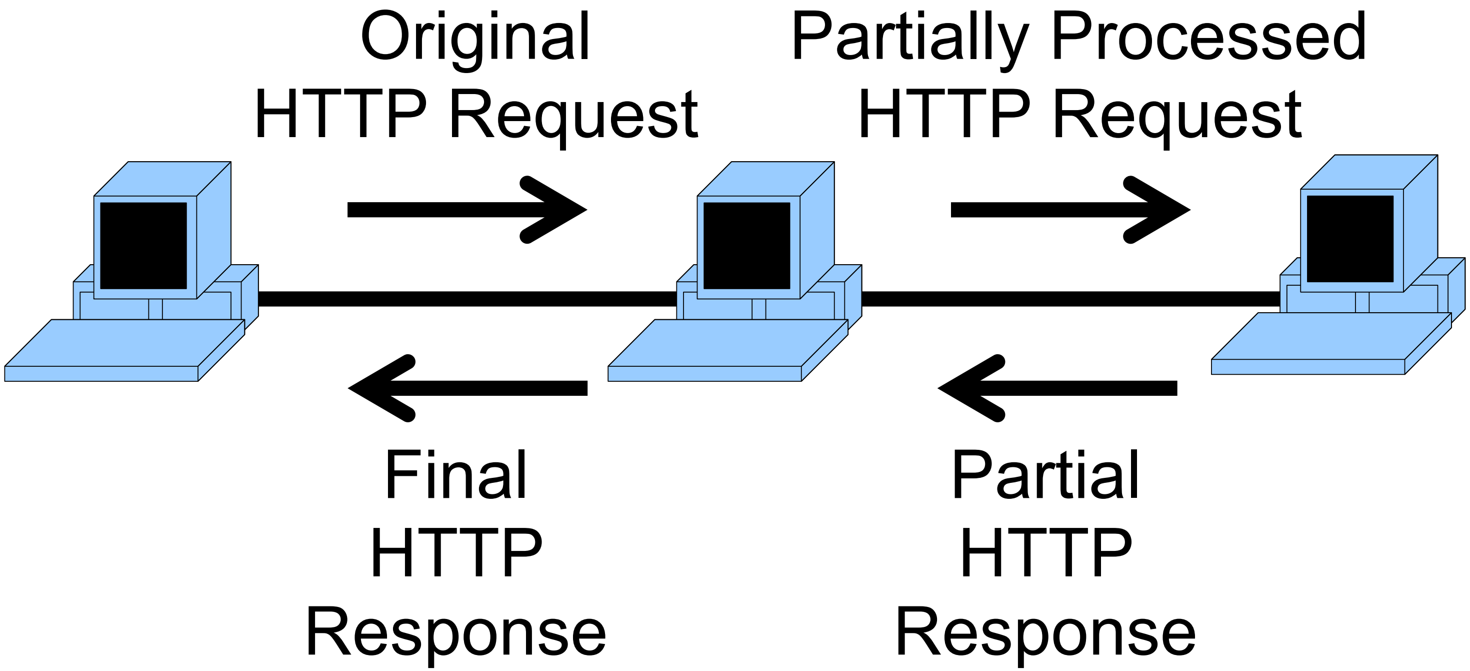
Request-response chains

- A chain of intermediary nodes can be placed between client and server, to partly process messages, pass firewalls etc.
- A client's request is passed through the chain until it reaches the final server
- The server's response is then passed back in the other direction down the chain to the client
- Intermediaries provide caching, translation and other such features

Proxy

Client knows it is using proxy to talk to server

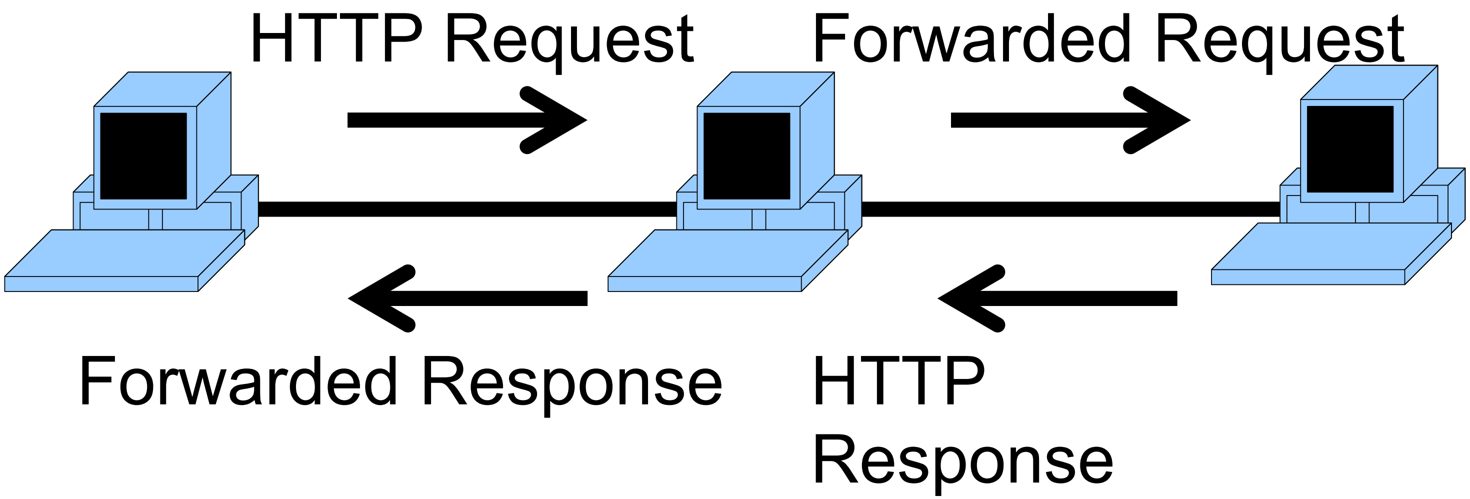
Proxy may transform messages en-route



Gateway

Client talks to gateway host as if final server

Gateway passes data on to final server



HTTP Request

Forwarded Request

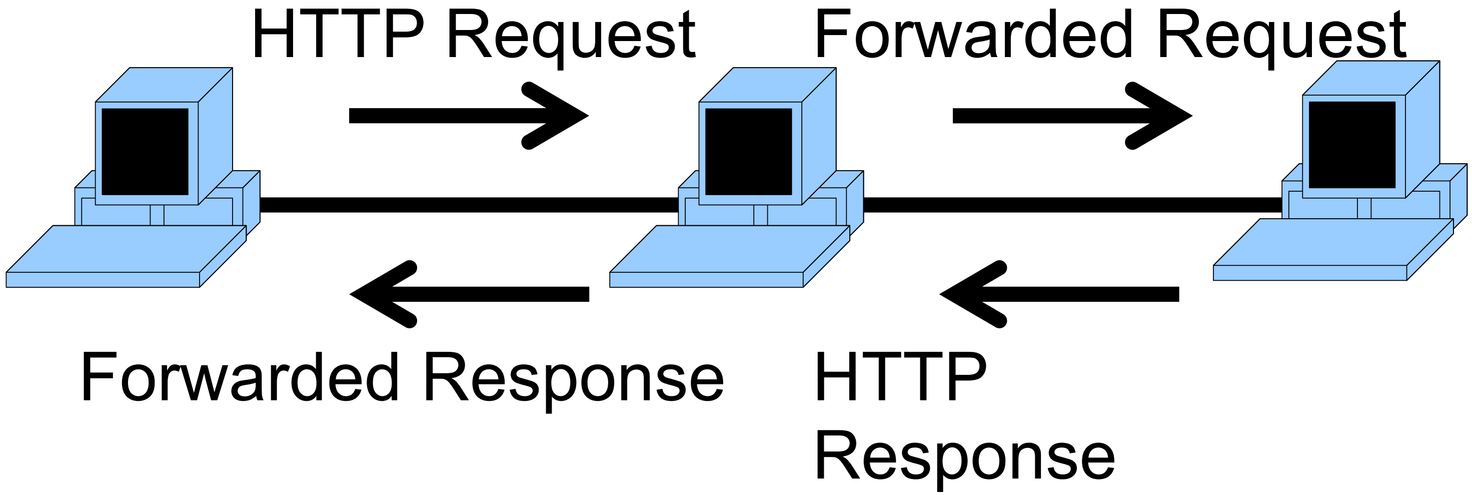
Forwarded Response

HTTP Response

Tunnel

Client talks to server as if tunnel was not there

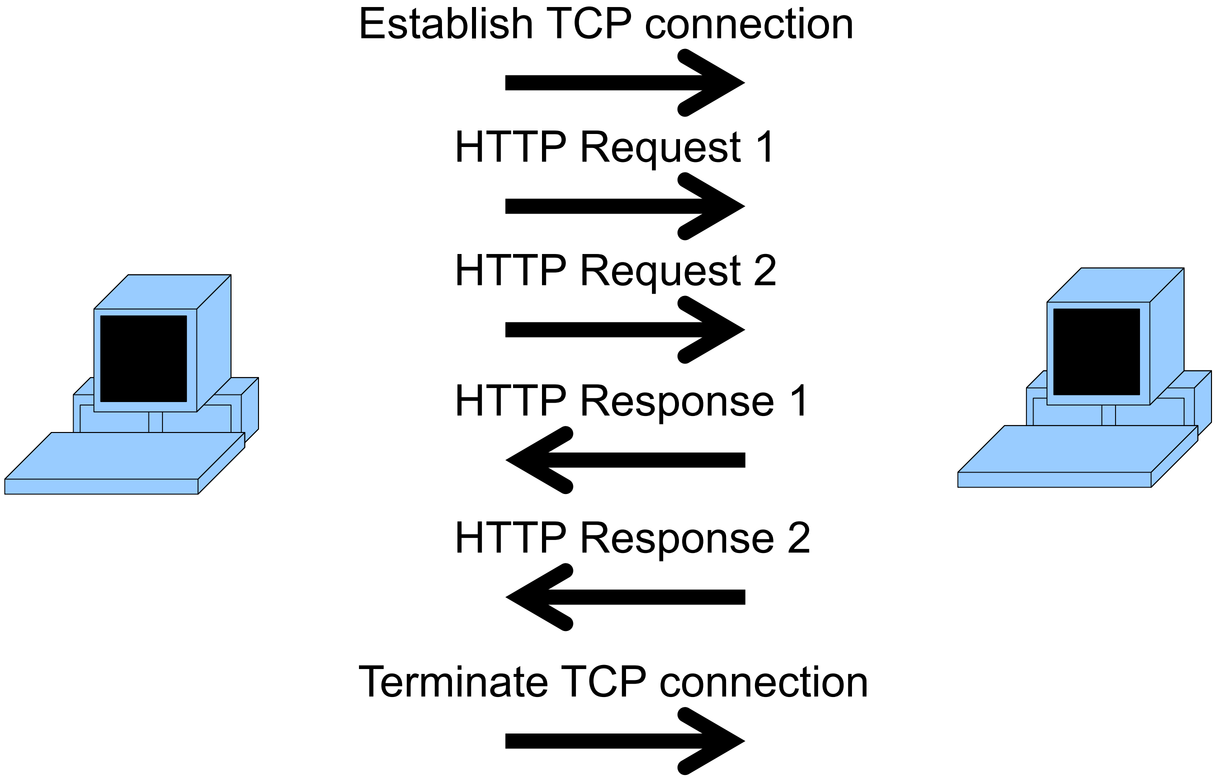
Tunnel does nothing but forwards messages



Pipelining

- For efficiency, where a client has several requests to a server, it can pipeline them
- This means the client establishes a single TCP connection, and sends its requests without waiting for a response for each in between
- A client distinguishes which response corresponds to which request by their contents
- A client ends a pipeline with a “connection: close” command

Pipelining



Establish TCP connection

HTTP Request 1

HTTP Request 2

HTTP Response 1

HTTP Response 2

Terminate TCP connection

Resources

- HTTP is used to get access to and manipulate resources
- A resource can be any network data object or service
- Common examples include:
  - Web pages
  - Database services
  - Downloadable files
  - Web Services
- A resource is identified by a **Uniform Resource Identifier (URI)**

Uniform Resource Identifiers

- If you are to refer to resources, to download them for example, you need a way to refer to them: a name
- Uniform Resource Identifiers (URI) are names for internet resources
- A URI can be of two kinds:
  - A Uniform Resource Name (URN) is a name for a resource that is globally unique and should still be meaningful after the resource ceases to exist (like the name of a person or ISBN of a book)
  - A Uniform Resource Locator (URL) is a URI that can also be used to retrieve the resource named
- A URI can be both URN and URL if it has the qualities of both

URI syntax

- A URI has a uniform structure:

<scheme>:<scheme-specific-part>

- The scheme defines how to interpret the scheme-specific part
- It is often used to refer to the application protocol by which the resource can be retrieved, e.g. HTTP, FTP

HTTP URL syntax

- An HTTP URL refines this structure for HTTP:

http://host [":" port] [path ["?" query]]

- **http:** denotes that it is an HTTP URL
- **host** is the domain name of a host with a resource
- **port** is the TCP port used
- **path** identifies the resource within the host
- **query** is an optional query to send to the resource, e.g. used with web page forms

Domain Name System

- IP addresses, being long numbers, are good for computers but not easy for people to remember
- Instead people use **domain names**, such as www.kcl.ac.uk or www.google.com
- The Domain Name System (DNS) translates domain names into IP addresses
- Like a telephone directory translates businesses' names to telephone numbers so you can call them
- We call such translation, **resolving** the domain name

Safe characters and IRIs

- Only a small set of characters can be used in the scheme-specific part of URIs:

A-Z a-z 0-9 $ - _ . + ! * ' ( )

- All other characters must be encoded as a % followed by their hexadecimal ASCII code

space	%20	@	%40	=	%3D
:	%3A	%	%25		

- **Internationalized Resource Identifiers** (IRIs) were developed to allow non-English URIs

URI examples

http://www.inf.kcl.ac.uk/cgi-bin/people.cgi?email=simon&search=1

https://www.topsecret.com/secure.html

mailto:simon.miles@kcl.ac.uk

file://C%3A/Temp/myfile.txt

ftp://myusername:mypassword@www.ctan.org/pub

Relative references

- A URI may refer to one resource relatively to another
- For example, on my website…

http://www.inf.kcl.ac.uk/staff/simonm/

- …a relative reference linked from that page…

contact.html

- …will be interpreted as being relative to my homepage URL, so equivalent to…

http://www.inf.kcl.ac.uk/staff/simonm/contact.html

Relative reference resolution

- Relative references should be read as UNIX file paths
- So, starting from:

http://www.inf.kcl.ac.uk/staff/simonm/

publications

- can be a relative subdirectory:

http://www.inf.kcl.ac.uk/staff/simonm/publications/

- while:

..

- refers to the parent directory (as in UNIX/DOS):

http://www.inf.kcl.ac.uk/staff/

URI Templates

- A **URI Template** is a compact sequence of characters for describing a range of Uniform Resource Identifiers through variable expansion, e.g.

  http://www.inf.kcl.ac.uk/staff/{username}/

- is the pattern for staff homepages in the Department, and can be instantiated by replacing variable {username} with a valid staff username

  {+basename}/faq.html

- describes the FAQ pages of websites, with prefix + meaning that reserved (unsafe) characters are allowed

HTTP methods

- Requests are typed by the **method** performed
- Commonly used methods are:
  - **GET** requests the retrieval of a resource, e.g. download a webpage to a browser
  - **POST** submits some resource to the server, e.g. when submitting a form on a webpage
  - **HEAD** requests just the header of the response that would be returned with a GET. Used to check a link is alive, finding when a resource was last modified etc.

Other methods

- The other available HTTP methods:
  - **OPTIONS** requests the server to send available communication options
  - **PUT** requests the storage of a resource
  - **DELETE** requests the resource be deleted
  - **TRACE** requests that the request itself be returned by the server, and is used just for diagnostics
  - **CONNECT** is used for tunnelling HTTP requests through a secure proxy (switching a proxy host to be a tunnel)

HTTP request format

- The format of an HTTP request is a series of newline-delimited lines
  - **Request line**: method, resource, HTTP version
  - **General headers**: Data about the message transmission or protocol
  - **Request headers**: Parameters of request
  - **Entity headers**: Describes request's data
  - **Entity body**: Data to be transferred (if any)

HTTP GET request example

```
GET /index.html HTTP/1.1                          Request Line
Connection: close                          General Header Line
Host: www.example.net                     Request Header Lines
Accept: text/html, text/plain
User-Agent: Mozilla/4.0 (compatible; MSTE 6.0)
```

Browser example

- If you type into a web browser:

  http://example.net:8080/dir/file.html

- The HTTP request will contain:

  GET /dir/file.html HTTP/1.1

  Host: example.net:8080

- The response message body will then contain the contents of **file.html**

- Default TCP port (if none given) is 80

Safe methods

- GET, HEAD, OPTIONS and TRACE are **safe** methods, because they do not request modifications to the server's data
- It is a requirement on the HTTP server implementation that these methods do nothing but retrieve and return entity data
- Knowing that a method is not safe allows the request to be confirmed with a user before something unfixable is done

Idempotent methods

- An **idempotent** method is one which will have the same effect whether done once or many times
- GET, HEAD, PUT, DELETE, OPTIONS and TRACE are idempotent, but POST in general is not
- For example, GET the same webpage more than once makes no difference, while POST more than once on a shopping site may mean you place more than one order for the same item

Idempotent sequences

- A sequence of requests can be non-idempotent even if the methods within it are idempotent
- For example: GET X, PUT X+1 is not idempotent because, run multiple times, the value of X will increase

GET X	->	Client set X = “A”
PUT X = “B”	->	Server sets resource X to “B”
GET X	->	Client receives “B”

Value of idempotence

- A client should ensure it does not pipeline non-idempotent methods and sequences

- This is because, if the connection breaks, the client may not know where in the pipelined sequence it has reached and so may need to repeat requests

HTTP response format

- The format of an HTTP response is a series of newline-delimited lines
  - **Status line**: HTTP version, status code, reason phrase
  - **General headers**: Data about the message transmission or protocol
  - **Response headers**: Parameters of the response
  - **Entity headers**: Description of the request's data
  - **Entity body**:The data to be transferred (if any)

Status codes

- The **status code** of a response is a 3 digit number representing whether the request was fulfilled correctly or, if not, why not
- Accompanied by a human-readable reason phrase
- The first digit of the code is the category of the status

Status code categories

Code	Category	Meaning
1YY	Information	General; no indication of success/failure of request
2YY	Success	Request understood and accepted by server
3YY	Redirection	Further action to complete
4YY	Client Error	Invalid request
5YY	Server Error	Unable to complete request

Common Status Codes

200: OK

300: Redirection, further interaction required

301: Resource moved permanently, the URI has changed and the client needs to request using the new identification

400: Bad request, syntax error by client

401: Unauthorised, request must include authentication

403: Forbidden, the request will not be authorised

404: Not found

500: Internal server error

505: HTTP version not supported

HTTP response example

```
HTTP/1.1 200 OK
```

Status Line

```
Date: Wed, 01 Aug 2007 11:13:09 GMT
Connection: close
```

General Header Lines

```
Server: Apache/1.3.31
Accept-Ranges: bytes
```

Response Header Lines

```
Content-Type: text/html, text/plain
Content-Length: 942
```

Entity Header Lines

```
<!DOCTYPE HTML PUBLIC = “-//W3C//DTD HTML 4.01
  Transitional//EN”>
<html>
<head>
<title>Dr. Simon Miles</title>
...
</html>
```

Entity Body

Multipurpose Internet Mail Extensions (MIME)

MIME

- HTTP and email were originally designed to carry ASCII text
- Now they can carry data of different kinds, e.g. an image attachment
- Must make the type of data explicit, so that an application can be chosen to process it
- MIME specifies how to declare the type of data being transferred
- MIME allows data to be encoded, to allow non-ASCII data transfer over ASCII procotols

Media types

- MIME provides a high level classification of data into 7 media types:
  - text
  - image
  - audio
  - video
  - application
  - multipart
  - message
- New types can be created but must start with “X-” to mark them as experimental

Media subtypes

- Each type can have many subtypes, giving a finer classification
- The full media type is written: **type/subtype**
- Common subtypes of text type MIME include:
  - text/plain
  - text/enriched
  - text/html
  - text/css

Parameterised types

- Optional parameters can be added to media type to give even more detail
- This takes the form: **type/subtype; options**

text/plain; charset=ISO-8859-1

Content type declaration

- In HTTP, email and other headers, the MIME type of the content is declared as:

Content-type: type/subtype; options

- This lets the receiving host know how to process the message data
- Example from an email:

Content-Type: text/plain; charset=ISO-8859-1

MIME in HTTP

- MIME types are used in HTTP messages, e.g.
- The type of the entity in an HTTP response:

Content-Type: text/plain

- The types accepted in response by an HTTP GET request:

Accepted: text/plain, text/html

MIME encoding

- Protocols have limits on:
  - The format of data, e.g. SMTP uses US-ASCII
  - The structure of data, e.g. SMTP limits line lengths to 1000 characters
- Have to encode data to make it transferable
- Have to declare what encoding was used, so that receiver can decode it (if necessary)
- MIME allows for such declarations

MIME encoding types

- MIME data can be encoded in one of 5 ways:
  - **7 bit**: ASCII compatible, fits in lines of 1000 characters
  - **8 bit**: 8-bit text character encoding, fits in lines of 1000 characters
  - **binary**: Encoded directly in binary form (cannot be used for direct transmission in SMTP)
  - **quoted-printable**: Non-standard ASCII text
  - **base64:** Binary data encoded as ASCII
- With 7bit, 8bit and binary, data is not changed to transmit

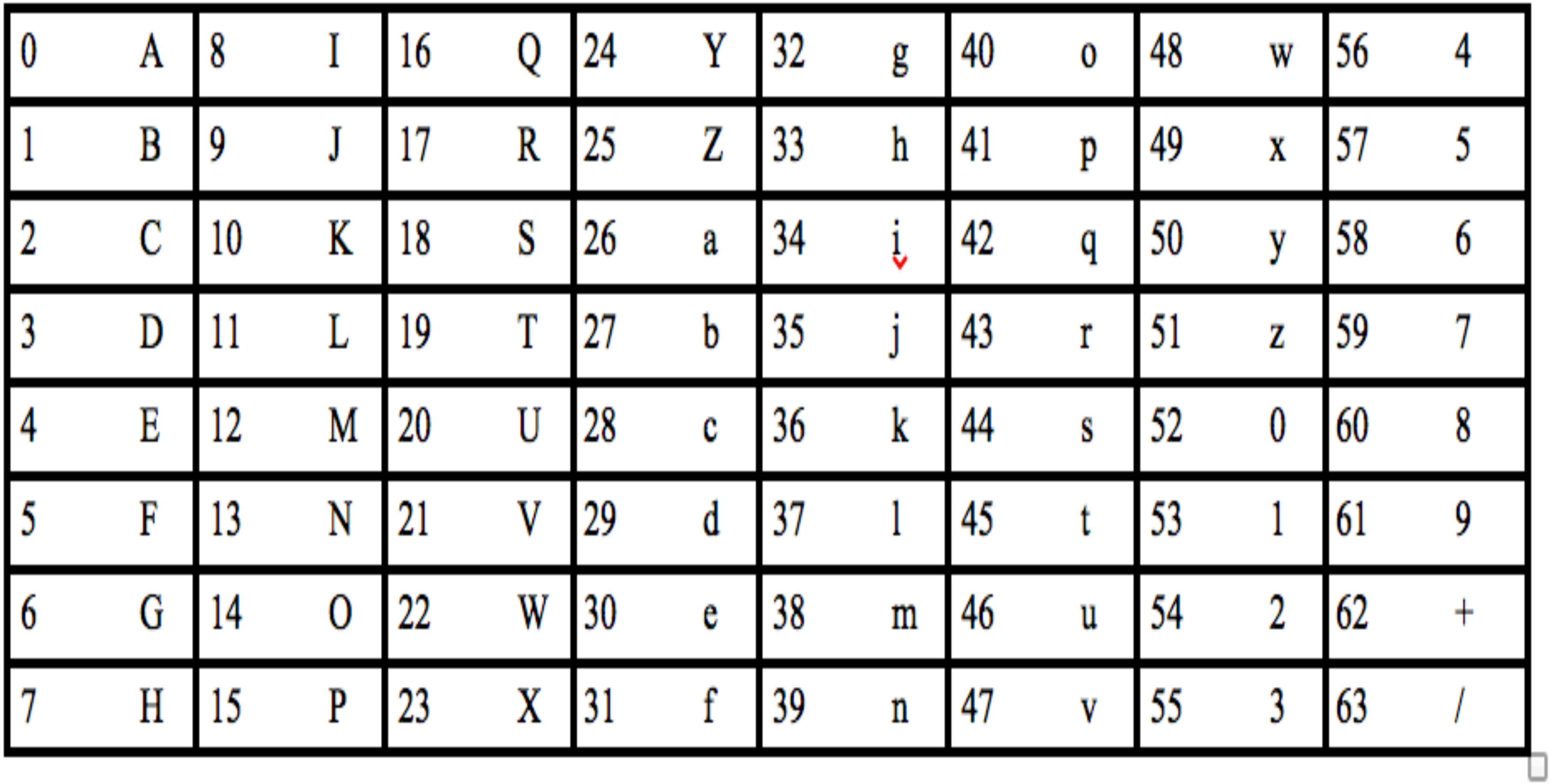
base64

- In base64, every 24 bits of data is split into four 6 bit chunks
- Each chunk is turned into a letter

0 -> A, 1 -> B, 2 -> C…

- Where data does not divide into 24 bit chunks, "=" is used to encode the missing chunks

base64 table



0	A	8	I	16	Q	24	Y	32	g	40	o	48	w	56	4
1	B	9	J	17	R	25	Z	33	h	41	p	49	x	57	5
2	C	10	K	18	S	26	a	34	i	42	q	50	y	58	6
3	D	11	L	19	T	27	b	35	j	43	r	51	z	59	7
4	E	12	M	20	U	28	c	36	k	44	s	52	0	60	8
5	F	13	N	21	V	29	d	37	l	45	t	53	1	61	9
6	G	14	O	22	W	30	e	38	m	46	u	54	2	62	+
7	H	15	P	23	X	31	f	39	n	47	v	55	3	63	/

base64 encoding example

- Encode the following bytes in base64:

  237, 162

- Convert the values to binary,

  11101101 10100010

- Split into 6-bit chunks (padding with 0s):

  111011 011010 001000 =

- Lookup values in table:

  59 -> 7 26 -> a 8 -> I

- Final encoding:

  7aI=

base64 decoding example

- Decode this base64 text back to bytes:

  SGVsbg8=

- Looking at the table, we see the encodings:

  6 -> G 18 -> S 21 -> V 27 -> b

  32 -> g 44 -> s 60 -> 8

- So we decode to decimal sequence:

  18, 6, 21, 44, 27, 32, 60

- In 6-bit binary, this is:

  010010/000110/010101/101100/011011/100000/111100

- As 8-bit decimals:

  72, 101, 108, 110, 15

Encoding declaration

- A header field (of HTTP, email etc.) is used to declare the encoding

Content-Transfer-Encoding: <encoding>

- For example:

Content-Transfer-Encoding: 7bit

MIME composite type

- A MIME type can be composite, i.e. for a collection of different types of data
- It has MIME type: **multipart/mixed**
- In this case, the parts are delimited by a named **boundary** in the email message
- Each part has its own MIME type
- The characters "--" are used to start each part and to end the list of parts

MIME composite type example

- Example: text and a GIF separated by a boundary named boundary-1

```
Content-Type: multipart/mixed; boundary=boundary-1

--boundary-1
Content-Type: text/plain; charset=US-ASCII
...
--boundary-1
Content-Type: image/gif
...
--boundary-1--
```

Web servers

Web servers

- A web server can be:
  - A program
  - A pre-packaged software/hardware appliance
  - Embedded in a consumer product, e.g. an ADSL modem/router
- Web server algorithm:
  1. Set up TCP connection
  2. Receive HTTP request
  3. Process request
  4. Access resource referred to by request
  5. Construct response
  6. Send response
  7. If requested, connection closed
  8. Log transaction

Resolving URIs

- Web servers need to resolve the resource URIs in requests to retrieve the correct documents

- On a host, a folder called the **document root** contains all files available to the web server
- The document root and request resource URI are combined to find the resource

Document root example

- Document root:

/dcs/web

- Request line:

GET /teaching/times.html HTTP/1.1

- Resource retrieved from:

/dcs/web/teaching/times.html

Virtual hosts

- Virtual hosts are host addresses that appear to be different, but use the same web server

  http://www.inf.kcl.ac.uk

  http://ais.inf.kcl.ac.uk

- The web server is configured to use a different document root for each virtual host

http://www.inf.kcl.ac.uk	/dcs/web
http://ais.inf.kcl.ac.uk	/dcs/web/research/groups/ais

IP and domain name virtual hosts

- Virtual hosts can be name-based or IP-based
- Name-based virtual hosting:
  - Multiple domain names will resolve to the same IP address (and so same host)
  - Each domain name corresponds to different virtual host, so has a different document root
- IP-based virtual hosting:
  - Multiple IP addresses are routed to the same host
  - A message sent to any of these addresses will go the host, which runs a web server
  - Each IP address corresponds to a different virtual host, so has a different document root

Experiment

- Open a telnet connection to a web server:

```
telnet www.inf.kcl.ac.uk 80 <ENTER>
GET /index.html HTTP/1.1 <ENTER>
Host: www.inf.kcl.ac.uk <ENTER>
<ENTER>
```

- Be quick or the web server will terminate the connection!

What we’ve covered

- Hypertext Transfer Protocol
- Proxies
- Uniform Resource Identifiers
- Requests and responses
- MIME
- Web servers