Networking Basics:
IP, DNS, URL, MIME

Computer Science and Engineering ■ College of Engineering ■ The Ohio State University

Lecture 11

Internet Protocol (IP) Addresses

- A unique 32-bit number
  - Assigned to device connected to internet
  - An address for delivery of packets
- Written in *dotted-decimal* notation
  - Divided into 4 fields separated by "."
  - Each field is 8 bits, ie 0-255 decimal

```
10100100011010110111101100000110
10100100.01101011.01111011.00000110
164.107.123.6
```

- Some are reserved: eg, 127.0.0.1

Value vs Encoding

- Value: 32-bit integer ie 0..4,294,967,295.
- Encodings
  - Dotted decimal
  - Dotted hex
  - Dotted octal
  - Hexadecimal
  - Decimal
  - Binary
  - Etc…
- Recall: abstraction, representation, *correspondence relation*

Address Space

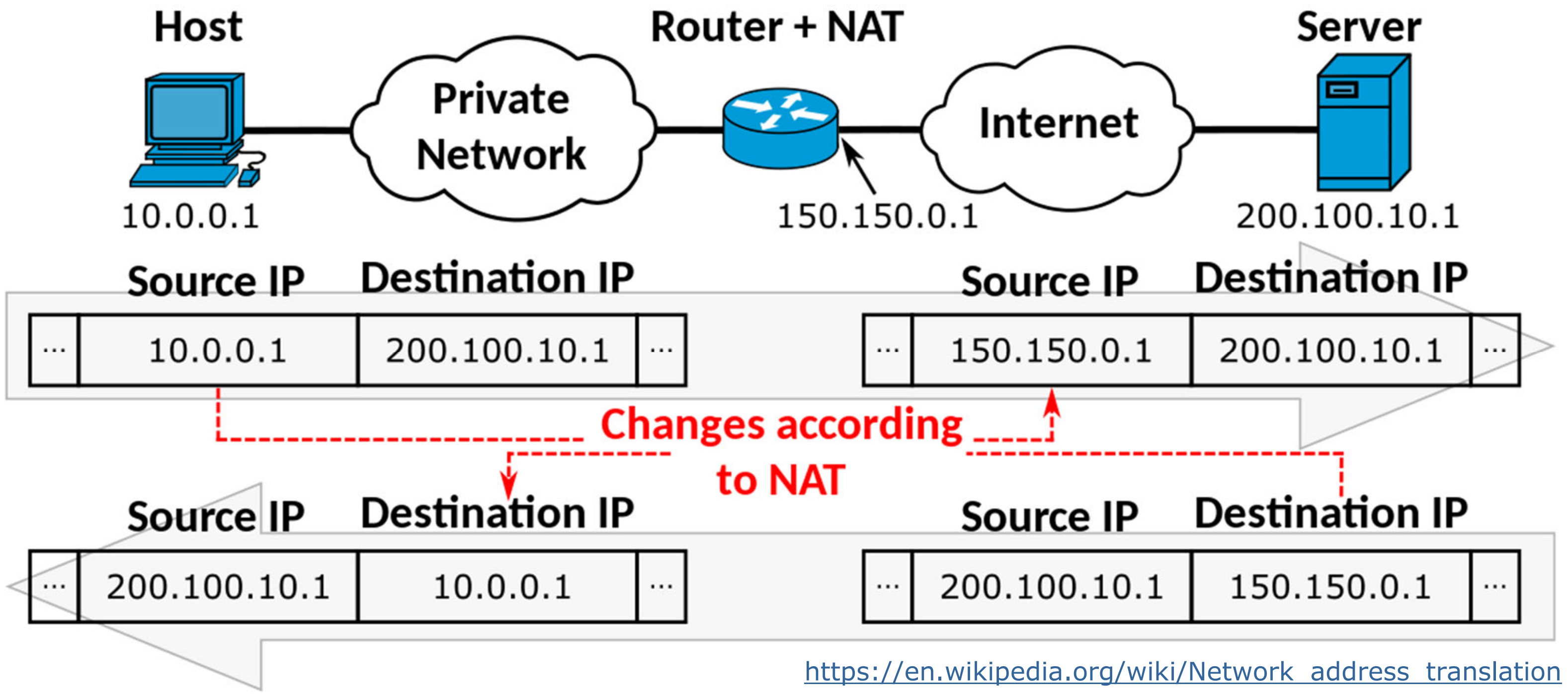
- Organizations are allocated blocks of contiguous address to use
- 32 bits = 4 billion distinct addresses
  - Population of the earth: 7 billion
  - Not enough addresses to go around!
  - Feb 2011: Last available top-level blocks were allocated (to regional internet registries)
  - Nov 2019: Last available address allocated

Address Space: Dealing with Scarcity

- Organizations are allocated blocks of contiguous address to use
- 32 bits = 4 billion distinct addresses
  - Population of the earth: 7 billion
  - Not enough addresses to go around!
  - Feb 2011: Last available top-level blocks were allocated (to regional internet registries)
  - Nov 2019: Last available address allocated
- Solutions:
  - NAT
  - IPv6

NAT: Network Address Translation

- When device on a local network sends data out:
  - Router replaces source's (private) IP address with its own
- When response comes back to the router:
  - Router replaces destination IP address with the private one of the original sender, forwarding response to that device



https://en.wikipedia.org/wiki/Network_address_translation

IPv6

- 128 bits
  - ~10^{40} addresses; we're good for a while
  - A growing fraction of IP traffic: GoogleIPv6 statistics
- Canonical format:
  - Divide bits into 8 fields, separated by ":"
  - Each field is 16 bits, written as 4 hex digits (0-FFFF)
  - Omit all *leading* 0's in a field (retaining at least one digit)
  - Replace the longest sequence of *consecutive* all-0 fields with "::"
  - Note: *at most one* such sequence is replaced
    - Replace left-most if there are multiple runs of this longest length
    - Result: unambiguous expansion back to 8 fields

Canonical Format: Uniqueness

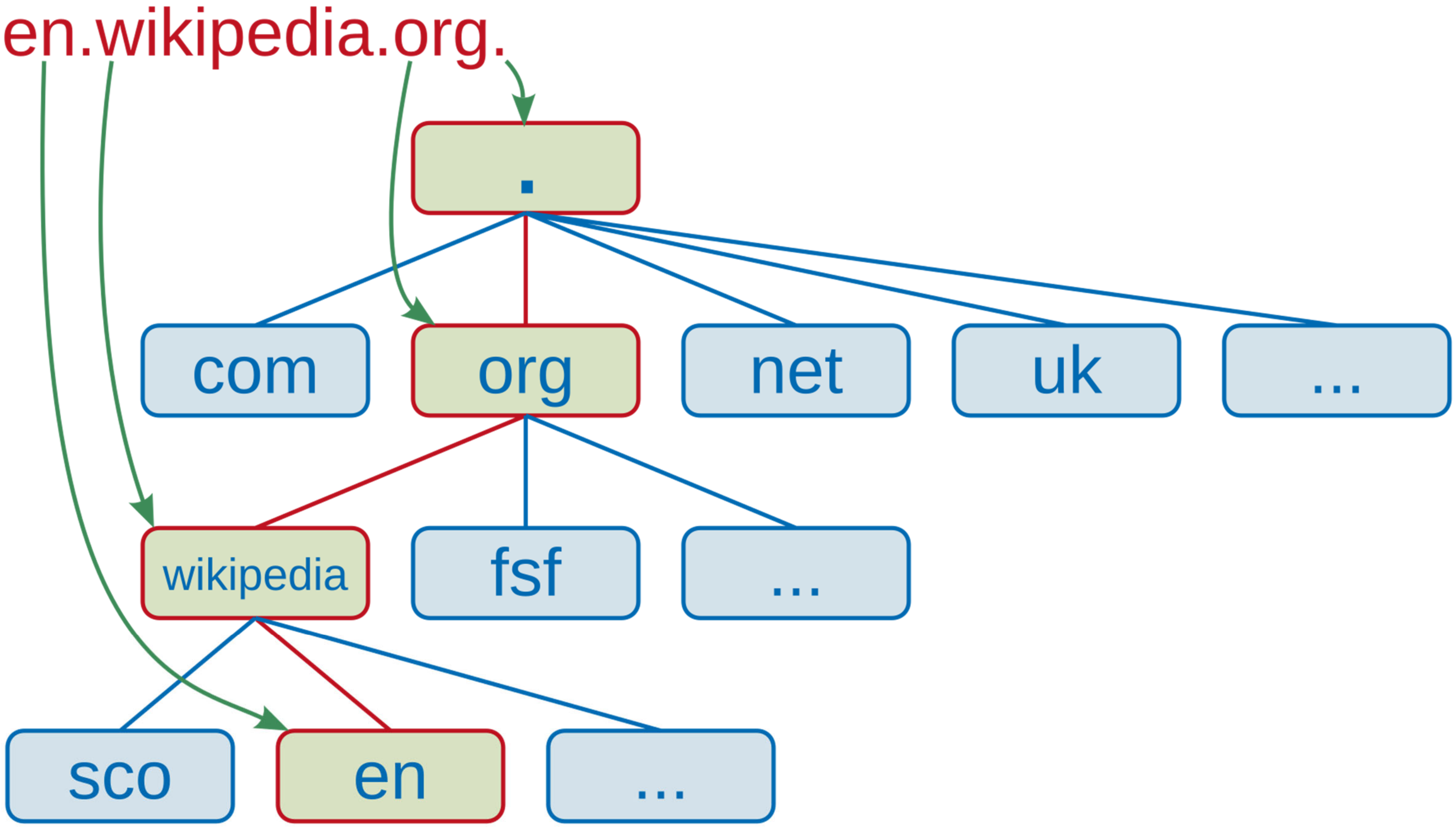

```
2001:0db8:0000:0000:0000:ff00:0042:8329
2001:0db8:0000:0000:0000:ff00:0042:8329
2001:db8:0:0:0:ff00:42:8329
2001:db8:0:0:0:ff00:42:8329
2001:db8::ff00:42:8329


::1                  # reserved for localhost
0:0:0:0:0:0:0:1
0000:0000:0000:0000:0000:0000:0000:0001
```

Domain Names and DNS

- Domain Names are human-friendly identifiers
  - `web.cse.ohio-state.edu` is easier than `164.107.129.161`
  - Case insensitive, but lower-case is standard
- Domain Name System (DNS): lookup service
  - Given a string, returns the corresponding IP address(es)
  - A string can map to multiple addresses!
  - Multiple strings can map to same address!
- Command-line tool: host

```
$ host cse.osu.edu                # 23.185.0.2
$ host www.cse.osu.edu            # 23.185.0.2
$ host osu.edu                 # 99.83.250.163, 75.2.76.225
$ host web.cse.ohio-state.edu # 164.107.129.16
```

Depiction of Domain Name Hierarchy



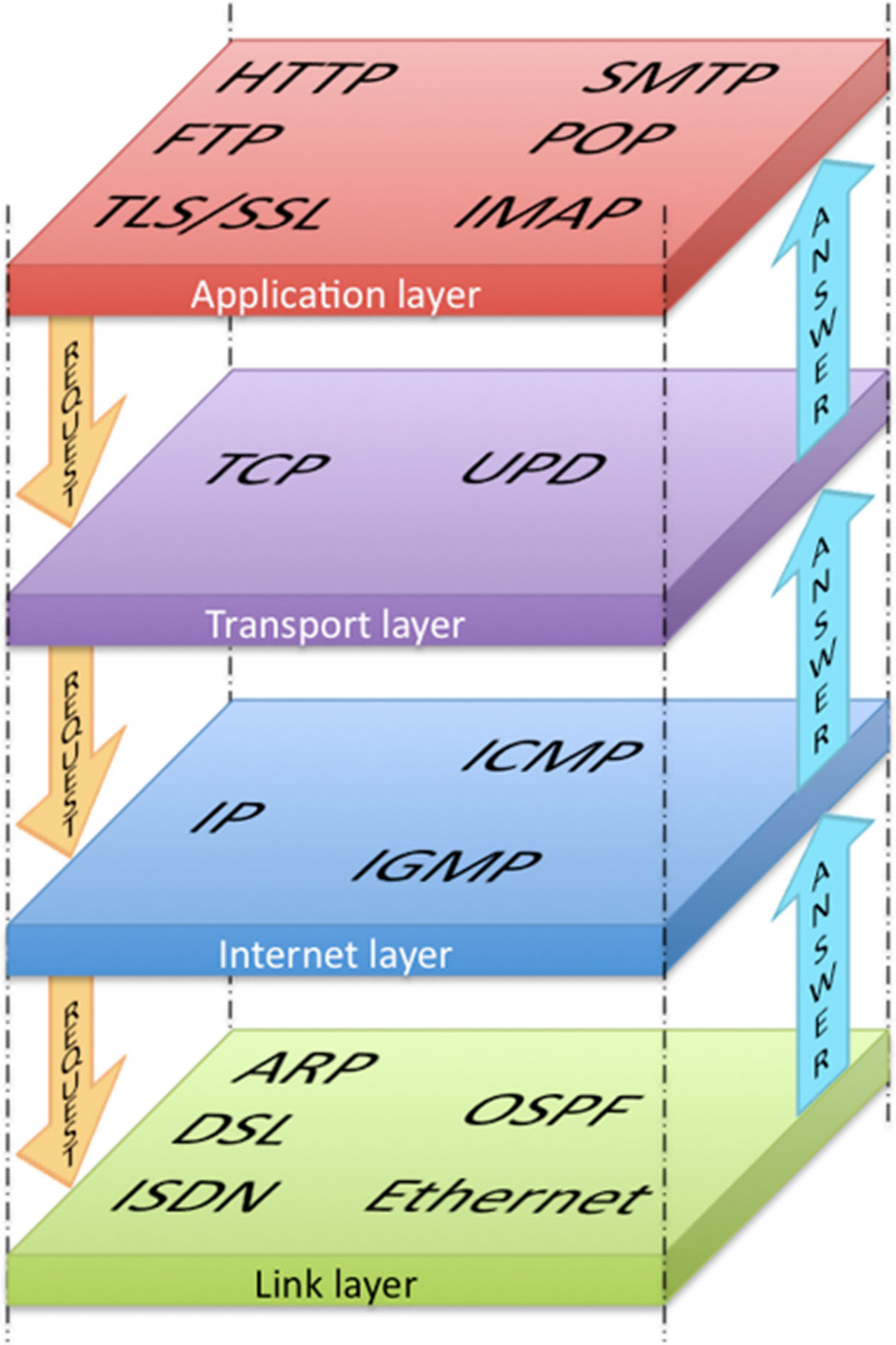
Domain Name Hierarchy

- Names separated by .'s: eg **`cse.osu.edu`**
  - This notation has nothing to do with IPv4 dotted decimal
- The right-most name is the *top level domain* (TLD)
  - **`.edu`**, **`.com`**, **`.net`**, **`.io`**, **`.gov`**, countries (**`.us`**, **`.ca`**, **`.it`**, …)
  - Second-level to its left, and so on…, eg **`www.sos.state.oh.us`**
- Domain registration (with Cloudflare, Porkbun, etc…) establishes ownership of a sub-tree
  - **`osu.edu, google.com, github.io, brutus.dev`**
  - Link Fully Qualified Domain Names (FQDN) to IP addresses
    - **`brutus.dev, www.brutus.dev, mail.brutus.dev,`** …
- Command-line tool: whois

```
$ whois osu.edu # vs ohio-state.edu, osu.com
```

Protocols

- Systematic ordering of messages
  - Phone rings
  - Callee answers by saying "Hello"
  - Caller answers by saying "Hello"
- Different protocols use different messages, different sequencing, etc
  - In Italy, callee answers by saying "*Pronto*"


HTTP
SMTP
FTP
POP
TLS/SSL
IMAP
Application layer
REQUEST
ANSWER
TCP
UPD
Transport layer
REQUEST
ANSWER
ICMP
IP
IGMP
Internet layer
REQUEST
ANSWER
ARP
DSL
OSPF
ISDN
Ethernet
Link layer

Network Layering: Abstraction

- One protocol is built on top of another
  - Application level: FTP, HTTP, SSH, SMTP, TELNET
  - Transport: TPC, UDP
  - Internet: IP
- Each protocol assumes certain behavior from layer below
  - IP routes packets to destination (unreliable)
  - TCP creates a reliable, in-order channel
  - HTTP delivers web pages

Network Ports

- A single host has many ports
- Each application-level protocol has a default port
  - ftp -> 20
  - http -> 80
  - imap ->143
  - ssh -> 22
  - smtp -> 25
  - telnet -> 23
- A "web server" is just a program, running, waiting, listening for an http call (on port 80)
  - See telnet

URL

- Uniform Resource Locator

  `scheme://FQDN:port/path?query#fragment`
- Schemes include http, https, ftp, mailto, file…
  - Case insensitive, but prefer lower case
- Port is optional (each scheme has a default)
  - 80 for http, 20 for ftp,
- Variety of formats, depending on scheme

  ```
  http://www.osu.edu/news/index.php
  ftp://doe@ftp.cse.ohio-state.edu
  mailto://brutus.1@osu.edu
  ```
- FQDN is case insensitive, prefer lower case

Document Root

- Web servers are configured to serve documents from a location in file system
  - "document root": `/var/www/html`
  - File: `/var/www/html/labs/lab2.html`
  - URL: `http://www.cse.osu.edu/labs/lab2.html`
- Slashes in path should be for server's OS (but forward slashes are common)
- Virtual servers: multiple doc roots
- Proxy servers: remote doc roots

Encoding (and Decoding)

- A value can be viewed at two levels, eg:
  - `HELLO`
  - `.... . .-.. .-.. ---`
- Motivation: different uses (reading vs transmission)
- May have
  - Different alphabets: letters vs dot-dash
  - Different requirements: letters must be upper-case
- Encoding/decoding is the translation between levels
  - cf. encrypting/decrypting
- Recall abstract value vs concrete representation

URL Encoding

- Invariant on abstract value (constraint)
  - Metacharacters (`;`, `:`, `&`, `#`, `@`…) are reserved
- Invariant on encoding (convention)
  - Small set of valid characters, others (eg space, `~`, newline…) are not allowed
- Result: some characters in abstract value are encoded as `%hh` (hex value of ASCII code)
  - ; is encoded as `%3B`
  - Space is encoded as `%20`
  - ~ is encoded as `%7E`
- Q: What about `%` in abstract value?
  - A: Encode it too! `%25`
- aka "percent encoding"

Example of URL Encoding

Reserved characters after percent-encoding

!	#	$	%	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%25	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

Common characters after percent-encoding (ASCII or UTF-8 based)

newline	space	"	%	-	.	<	>	\	^	_	`	{	|	}	~
%0A *or* %0D *or* %0D%0A	%20	%22	%25	%2D	%2E	%3C	%3E	%5C	%5E	%5F	%60	%7B	%7C	%7D	%7E

Value `Mascot "address": brutus@osu.edu`

Encoding `Mascot%20%22address%22%3A%20brutus%40osu.edu`

MIME

- Multipurpose Internet Mail Extensions
  - Originally developed for email attachments
- Content Type: How to interpret a *file*
  - File is a blob of bits (an encoding)
  - How should we decode this blob into an (abstract) value? Colors, sounds, characters?
  - Recall: *correspondence relation*
- Syntax of content types: **`type/subtype`**
  - text/plain, text/html, text/css, text/javascript
  - image/gif, image/png, image/jpeg
  - video/mpeg, video/quicktime
- Transfer Encoding: How to interpret a *msg*
  - How to decode the blob of bits that arrived into a file
  - A *layered* encoding
  - Examples: quoted-printable, base64

Email Example: Multiple Parts


```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=aFrontierString

This is a message with multiple parts in MIME format.
--aFrontierString
Content-Type: text/plain

This is the body of the message.
--aFrontierString
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB
0aGUg
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--aFrontierString--
```

Email Example: Content Type


```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=aFrontierString

This is a message with multiple parts in MIME format.
--aFrontierString
Content-Type: text/plain

This is the body of the message.
--aFrontierString
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB
0aGUg
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--aFrontierString--
```

Determining MIME Content Type

- The sender (web server) determines MIME (content) type of document being sent
  - Rules map file extensions to MIME types
- If file arrives without MIME info, receiver has to guess (see `file` command)
  - File extension *may* help
  - Contents *may* help: Some files begin with a "magic number"
    - JPG: `ff d8`…
    - PDF: `25 50 44 46 2D`… (ie %PDF-)
    - PNG: `89 50 4e 47 0d 0a 1a 0a`… (ie PNG…)
- Some types handled by browser itself
- Others require plugin or application
- Experimental MIME subtypes: x-
  - application/x-gzip

Email Exmple: Transfer Encoding


```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=aFrontierString

This is a message with multiple parts in MIME format.
--aFrontierString
Content-Type: text/plain

This is the body of the message.
--aFrontierString
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB
0aGUg
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
--aFrontierString--
```

Layered Encoding

source
(image)

Content-Type
image/jpeg

content
(bits)

ffd8ffe000104a464946…

Content-Transfer-Encoding
???

transfer encoded
(channel)
ASCII

/9j/4AAQSk…

Encoding (Binary) Data in ASCII

- Binary data: Any byte value is possible
  - 00 to FF (*i.e.* xxxx xxxx)
- ASCII data: bytes start with 0
  - 00 to 7F (*i.e.* 0xxx xxxx)
- Problem: channel can carry only ASCII data
  - Encoding must use ASCII alphabet (bytes that start with 0)
- Solution? Use Hex: 4 bits sent as 1 ASCII char

```
1101 0110 1100 1111 0010 0110
D    6    C    F    2    6
```

  - Problem?

Encoding (Binary) Data in ASCII: Problem

- Binary data: Any byte value is possible
  - 00 to FF (*i.e.* xxxx xxxx)
- ASCII data: bytes start with 0
  - 00 to 7F (*i.e.* 0xxx xxxx)
- Problem: channel can carry only ASCII data
  - Encoding must use ASCII alphabet (bytes that start with 0)
- Solution? Use Hex: 4 bits sent as 1 ASCII char

```
1101 0110 1100 1111 0010 0110
D    6    C    F    2    6
```

  - Problem?

```
0100 0100 0011 0110 0100 0011
     D         6         C
```

Quoted-Printable Encoding

- Observation: bytes that happen to be ASCII do not need to be encoded
  - If most data is text, savings are significant
- For each byte:
  - If first bit is 0, do nothing
  - If first bit is 1, encode with 3 bytes: **`=XY`** where XY is the hex value of byte
- Limit line length to 76 characters
- Finish lines with "="
- Q: What if data contains the byte "="?

Example


```
J'interdis aux marchands de vanter trop leur marchandises. Car ils se font vite pédagogues et
t'enseignent comme but ce qui n'est par essence qu'un moyen, et te trompant ainsi sur la route à suivre
les voilà bientôt qui te dégradent, car si leur musique est vulgaire ils te fabriquent pour te la vendre
une âme vulgaire.
```

```
J'interdis aux marchands de vanter trop leur marchandises. Car ils se font =
vite p=C3=A9dagogues et t'enseignent comme but ce qui n'est par essence qu'=
un moyen, et te trompant ainsi sur la route =C3=A0 suivre les voil=C3=A0 bi=
ent=C3=B4t qui te d=C3=A9gradent, car si leur musique est vulgaire ils te f=
abriquent pour te la vendre une =C3=A2me vulgaire.
```

Encoding Binary Data

- What if most data is *not* ASCII?
  - Raw (base 256): 8 bits are a digit (1 byte), 1:1

```
1101 0110 1100 1111 0010 0110
?         ?         &
```

  - Hex (base 16): 4 bits → 1 digit (1 byte), ie 2x

```
1101 0110 1100 1111 0010 0110
D    6    C    F    2    6
```

  - Quoted-Printable: 8 bits → ~3 characters (3 bytes), ie ~3x

```
1101 0110 1100 1111 0010 0110
=D   6    =C   F    &
```

  - Can we do better?

Encoding Binary Data: Base 64

- What if most data is *not* ASCII?
  - Raw (base 256): 8 bits are a digit (1 byte), 1:1

```
1101 0110 1100 1111 0010 0110
?         ?         &
```

  - Hex (base 16): 4 bits → 1 digit (1 byte), ie 2x

```
1101 0110 1100 1111 0010 0110
D    6    C    F    2    6
```

  - Base 64: 6 bits → 1 digit (1 byte), ie 1.33x

```
1101 0110 1100 1111 0010 0110
1      s      8      m
```

Base64 Alphabet

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

en.wikipedia.org/wiki/Base64

Layered Encoding: Base64

source
(image)

Content-Type
image/jpeg

content
(bits)

ffd8ffe000104a464946…

Content-Transfer-Encoding
base64

encoded
(alphabet)

/9j/4AAQSk…

ASCII

transmission
(bits)

2f396a2f344141536b…

Base64 Encoding

<table>
<tr><th>source ASCII (if <128)</th><th colspan="8">M</th><th colspan="8">a</th><th colspan="8">n</th></tr>
<tr><th>source octets</th><td colspan="8">77 (0x4d)</td><td colspan="8">97 (0x61)</td><td colspan="8">110 (0x6e)</td></tr>
<tr><th>Bit pattern</th><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><th>Index</th><td colspan="6">19</td><td colspan="6">22</td><td colspan="6">5</td><td colspan="6">46</td></tr>
<tr><th>Base64-encoded</th><td colspan="6">T</td><td colspan="6">W</td><td colspan="6">F</td><td colspan="6">u</td></tr>
<tr><th>encoded octets</th><td colspan="6">84 (0x54)</td><td colspan="6">87 (0x57)</td><td colspan="6">70 (0x46)</td><td colspan="6">117 (0x75)</td></tr>
</table>

<table>
<tr><th>Text content</th><th colspan="8">M</th><th colspan="8"></th><th colspan="8"></th></tr>
<tr><th>ASCII</th><td colspan="8">77 (0x4d)</td><td colspan="8">0 (0x00)</td><td colspan="8">0 (0x00)</td></tr>
<tr><th>Bit pattern</th><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><th>Index</th><td colspan="6">19</td><td colspan="6">16</td><td colspan="6">0</td><td colspan="6">0</td></tr>
<tr><th>Base64-encoded</th><td colspan="6">T</td><td colspan="6">Q</td><td colspan="6">=</td><td colspan="6">=</td></tr>
</table>

en.wikipedia.org/wiki/Base64

Summary

- IP address are unique on network
  - IPv4 vs IPv6
- DNS maps strings to IP addresses
  - Domains nested hierarchically
- URLs identify resources on network
  - Scheme, host, path
- MIME type defines a file’s encoding
  - Correspondence
  - Layered encodings are possible too