Data Communications Fundamentals: Signals, Codes, Compression, Integrity

Cristian S. Calude

July–August 2009

Thanks to

Nevil Brownlee and Ulrich Speidel for stimulating discussions and critical comments.

Goals

- Understand digital and analog signals
- Understand codes and encoding schemes
- Understand compression, its applications and limits
- Understand flow-control and routing
- Discuss recent issues related to the above topics

References

1. B. A. Forouzan. *Data Communications and Networking*, McGraw Hill, 4th edition, New York, 2007.
2. W. A. Shay. *Understanding Data Communications and Networks*, 3rd edition, Brooks/Cole, Pacific Grove, CA, 2004. **(course textbok)**

Pictures

All pictures included and not explicitly attributed have been taken from the instructor's documents accompanying Forouzan and Shay textbooks.

Factors determining data transmission

- cost of a connection
- amount of information transmitted per unit of time (bit rate)
- immunity to outside interference (noise)
- susceptibility to unauthorised "listening" (security)
- logistics
- mobility

Analog and digital signals

Connected devices have to "understand" each other to be able to communicate.

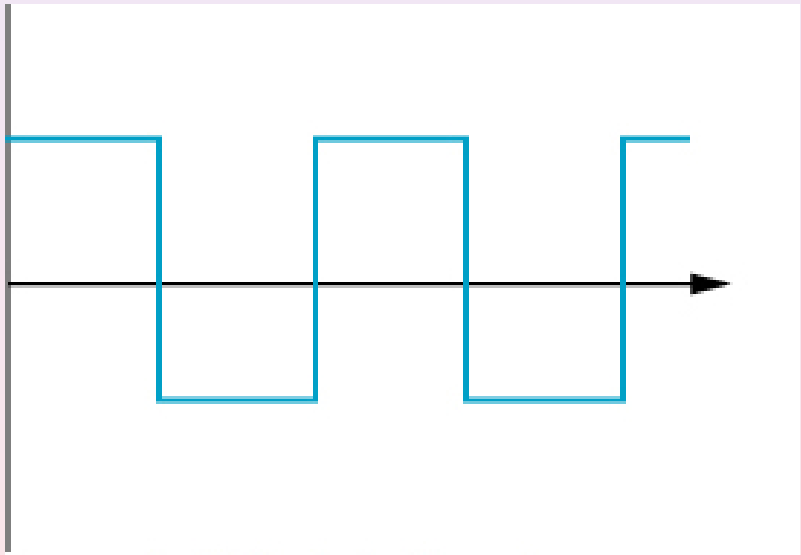
Communication standards assure that communicating devices represent and send information in a "compatible way".

There are two types of ways to transmit data:

- via **digital signals**, which can be represented either electronically (by sequences of specified voltage levels) or optically,
- via **analog signals**, which are formed by continuously varying voltage levels.

Digital signals 1

Digital signals are graphically represented as a square wave: the horizontal axis represents time and the vertical axis represents the voltage level.



(a) Digital signal

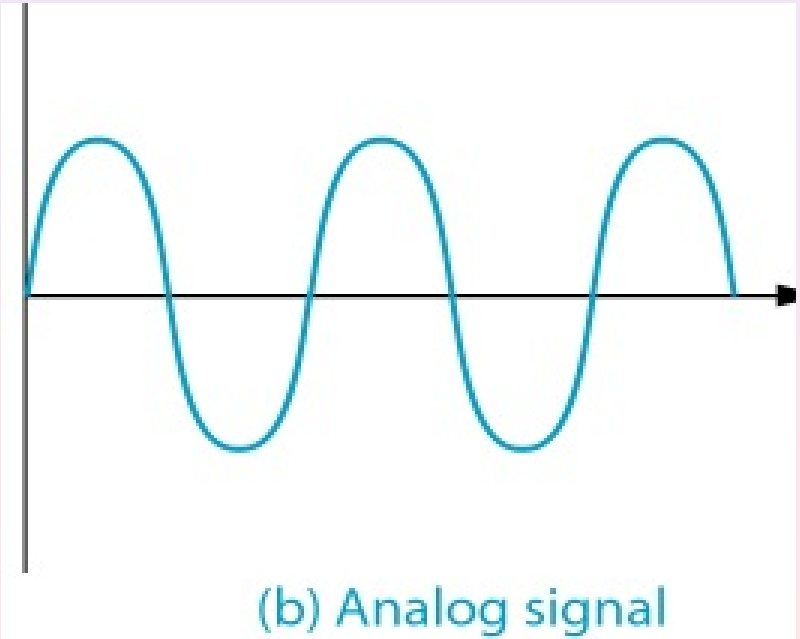
Digital signals 2

The alternating high and low voltage levels may be symbolically represented by 0s and 1s. This is the simplest way to represent a binary string (bit-string).

Each 0 or 1 is called a **bit**. Various codes combine bits to represent information stored in a computer.

Analog signals

PCs often communicate via modems over telephone lines using **analog signals** which are formed by continuously varying voltage levels:



How signals travel?

There are three types of transmission media, each with many variations:

- *conductive metal*, like copper or iron, that carries both digital and analog signals; coaxial cable and twisted wire pairs are examples,
- *transparent glass strand or optical fibre* that transmits data using light waves,
- *no physical connection* that transmits data using electromagnetic waves (as those used in TV or radio broadcast).

Unit and multi-units

Unit	Equivalent
pico-unit (pu)	10^{-12} u
nano-unit (nu)	10^{-9} u
micro-unit (μu)	10^{-6} u
milli-unit (mu)	10^{-3} u
unit (u)	1 u
kilo-unit (Ku)	10^{3} u
mega-unit (Mu)	10^{6} u
giga-unit (Gu)	10^{9} u
tera-unit (Tu)	10^{12} u

Costs

There are two forms of costs:

- costs of wires, cables, devices, etc.,
- number of bits transmitted per unit of time.

Bytes 1

For example, the storage capacity can be expressed in:

- B (bytes, 1 B = 8 b (bits)),
- KB (10^3 bytes),
- MB (10^6 bytes),
- GB (10^9 bytes),
- TB (10^{12} bytes),
- ...

Bytes 2

More about quantities of bytes

A **kibibyte** (**ki**lo **bi**nary **byte**) is a unit of information established by the International Electrotechnical Commission (2000).

Name (SI symbol)	Standard SI	Name (binary symbol)	Value
kilobyte (KB/kB)	$10^3 = 1000^1$	**kibibyte (KiB)**	2^{10}
megabyte (MB)	$10^6 = 1000^2$	mebibyte (MiB)	2^{20}
gigabyte (GB)	$10^9 = 1000^3$	gibibyte (GiB)	2^{30}
terabyte (TB)	$10^{12} = 1000^4$	tebibyte (TiB)	2^{40}
petabyte (PB)	$10^{15} = 1000^5$	pebibyte (PiB)	2^{50}
exabyte (EB)	$10^{18} = 1000^6$	exbibyte (EiB)	2^{60}
zettabyte (ZB)	$10^{21} = 1000^7$	zebibyte (ZiB)	2^{70}
yottabyte (YB)	$10^{24} = 1000^8$	yobibyte (YiB)	2^{80}

Bit rate

The **bit rate** is the number of bits transmitted per unit of time. The typical unit is *bits per second (b/s)*.

- b/s (bits per second),
- Kb/s (10^3 bits per second),
- Mb/s (10^6 bits per second),
- Gb/s (10^9 bits per second),
- Tb/s (10^{12} bits per second).

Depending on the medium and the application, bit rates vary from a few hundred b/s to gigabits per second and pushing into terabits per second.

Bit rates vary according to:

- downstream vs. upstream,
- the time of day — usually between 4pm to midnight speeds are likely to be slower,
- the web-sites you visit — some web-sites limit the speed at which they send out information,
- how many computers share the connection,
- the application(s) used,
- viruses and spyware on your computer.

Bandwidth and latency 1

Broadband is associated with high-speed connection, but speed is determined by various factors, among them bandwidth and latency.

- Bandwidth describes how much data can be sent over the network.
- Latency is the elapse time for a single byte (or packet) to travel from one host to another. There are propagation, transmission and processing delays.
- Errors appear due to interference, busy routers, link fails.

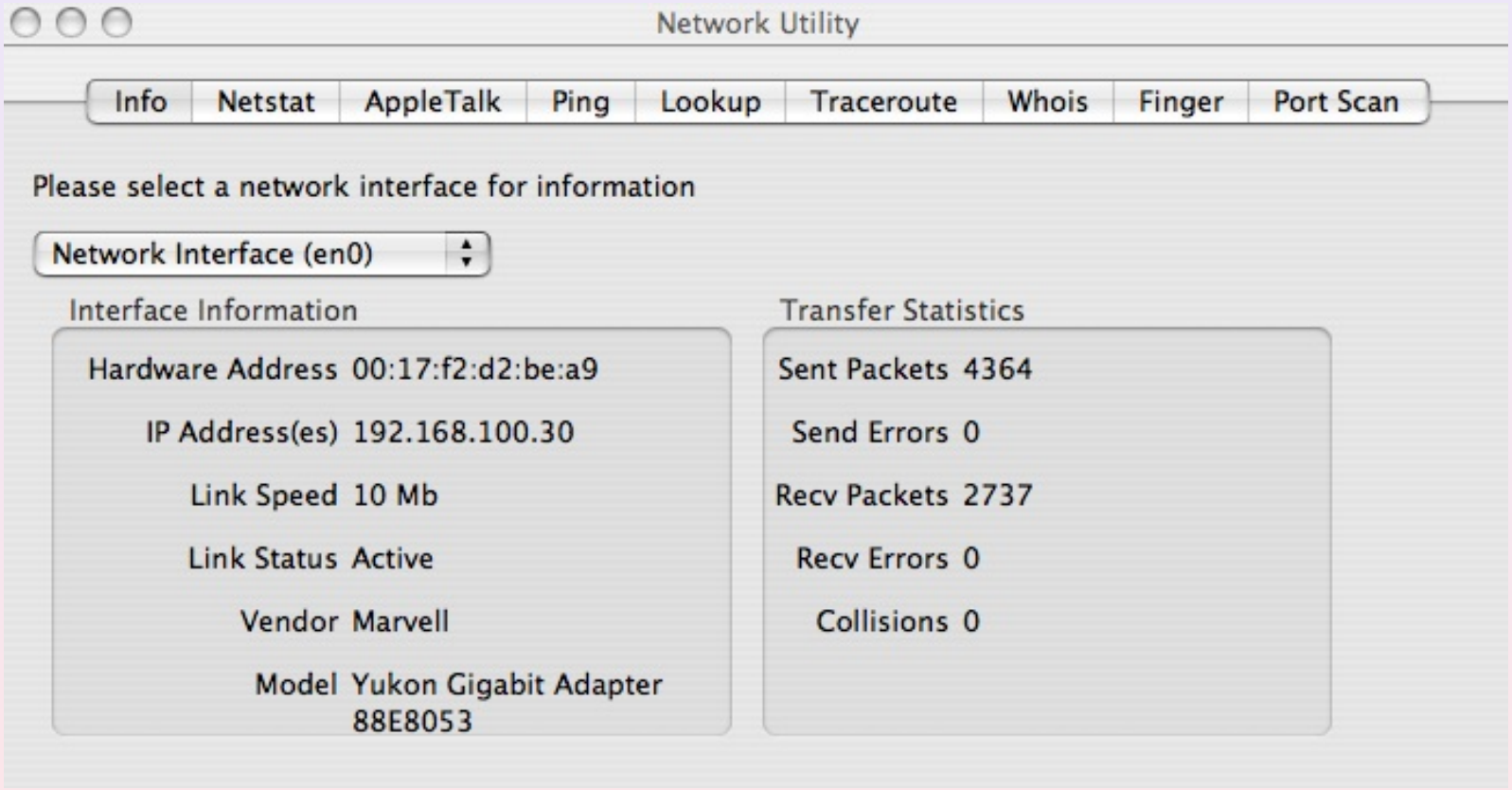
Bandwidth is limited to the poorest link, but latency and networks errors are cumulative. The further data has to travel between hosts, the more traffic it must compete with and the more resources it uses.

Bandwidth and latency 2

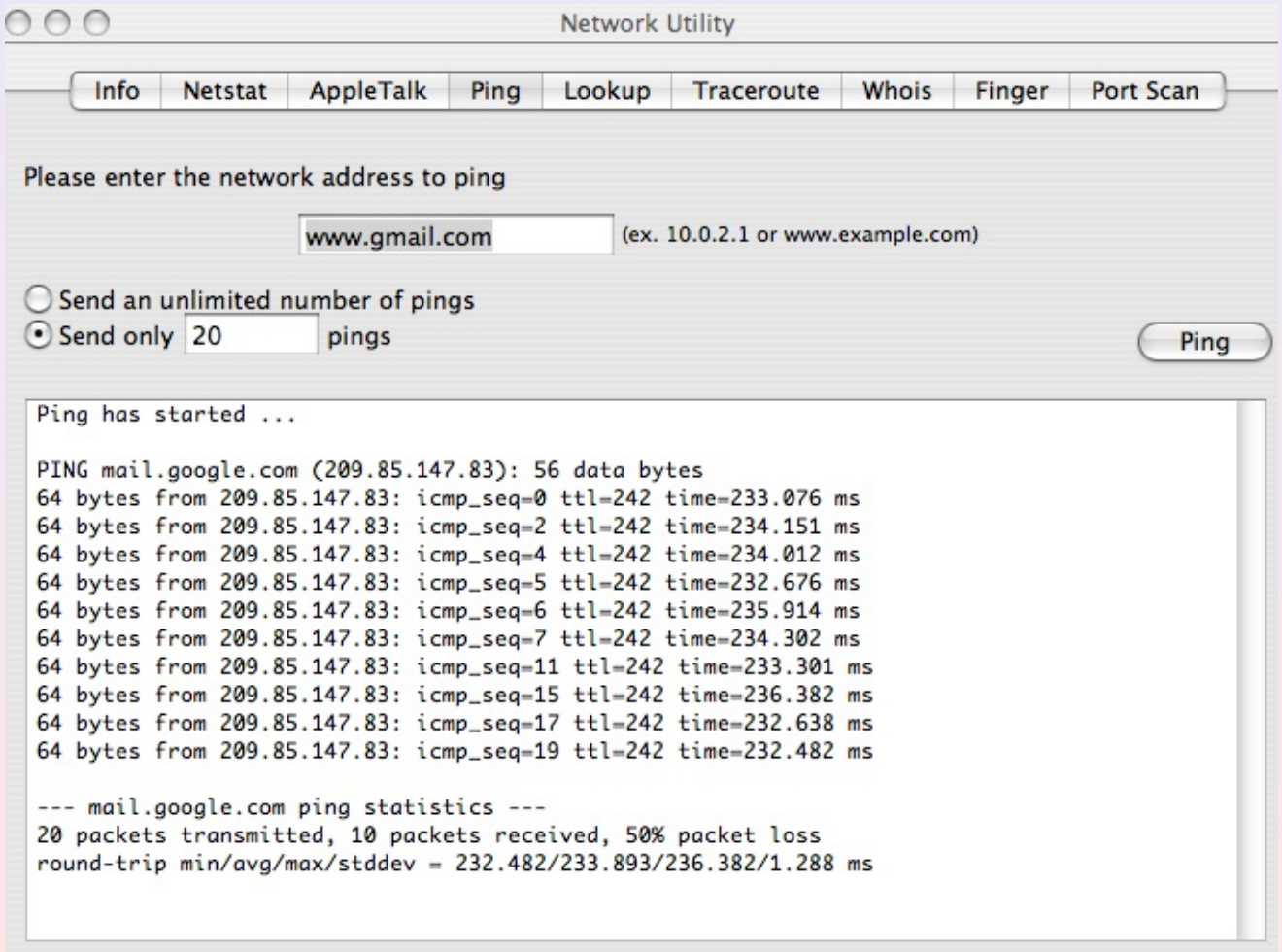
	Bandwidth	Latency (one-way)	Errors
28.8 Analog Modem	3.0 KB/s	120 ms	moderate
28.8 No Compression	2.0 KB/s	70 ms	high
56K Analog Modem	5.0 KB/s	100 ms	moderate
ISDN Digital Modem	4-8 KB/s	20 ms	low
Direct Serial	10-115 KB/s	20 ms	moderate
Ethernet (0-1 hops)	2-12 MB/s	$<$ 5 ms	low
Ethernet (2-3 hops)	1-4 MB/s	10-30 ms	low
Ethernet (4-6 hops)	0.5-1 MB/s	50-100 ms	moderate
Ethernet (distant)	$<$ 200 KB/s	$>$ 100 ms	mod-high

Table: Common network connections

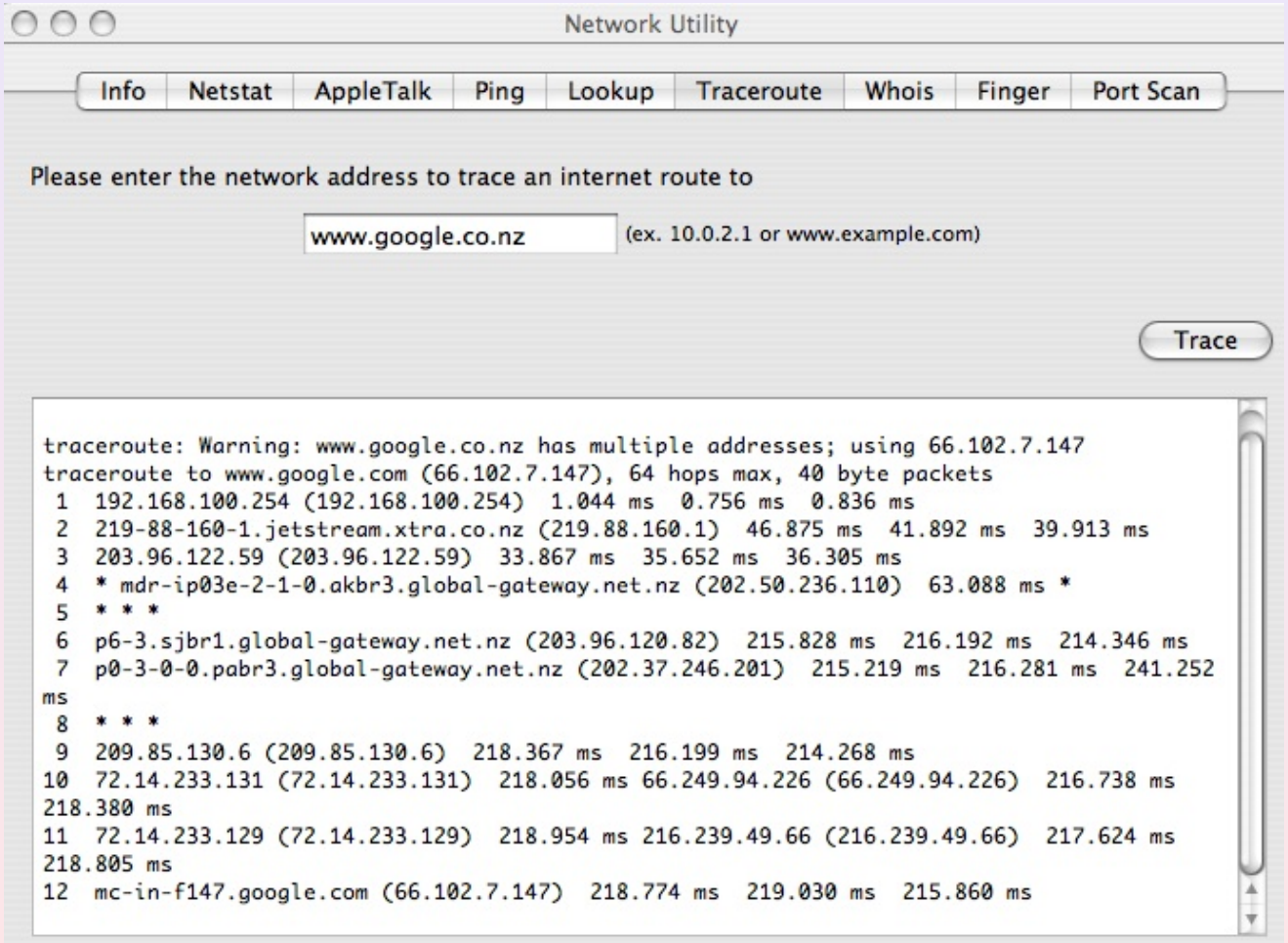
Network utility



Ping


Network Utility
Info
Netstat
AppleTalk
Ping
Lookup
Traceroute
Whois
Finger
Port Scan
Please enter the network address to ping
www.gmail.com
(ex. 10.0.2.1 or www.example.com)
Send an unlimited number of pings
Send only 20 pings
Ping
Ping has started ...
PING mail.google.com (209.85.147.83): 56 data bytes
64 bytes from 209.85.147.83: icmp_seq=0 ttl=242 time=233.076 ms
64 bytes from 209.85.147.83: icmp_seq=2 ttl=242 time=234.151 ms
64 bytes from 209.85.147.83: icmp_seq=4 ttl=242 time=234.012 ms
64 bytes from 209.85.147.83: icmp_seq=5 ttl=242 time=232.676 ms
64 bytes from 209.85.147.83: icmp_seq=6 ttl=242 time=235.914 ms
64 bytes from 209.85.147.83: icmp_seq=7 ttl=242 time=234.302 ms
64 bytes from 209.85.147.83: icmp_seq=11 ttl=242 time=233.301 ms
64 bytes from 209.85.147.83: icmp_seq=15 ttl=242 time=236.382 ms
64 bytes from 209.85.147.83: icmp_seq=17 ttl=242 time=232.638 ms
64 bytes from 209.85.147.83: icmp_seq=19 ttl=242 time=232.482 ms
--- mail.google.com ping statistics ---
20 packets transmitted, 10 packets received, 50% packet loss
round-trip min/avg/max/stddev = 232.482/233.893/236.382/1.288 ms

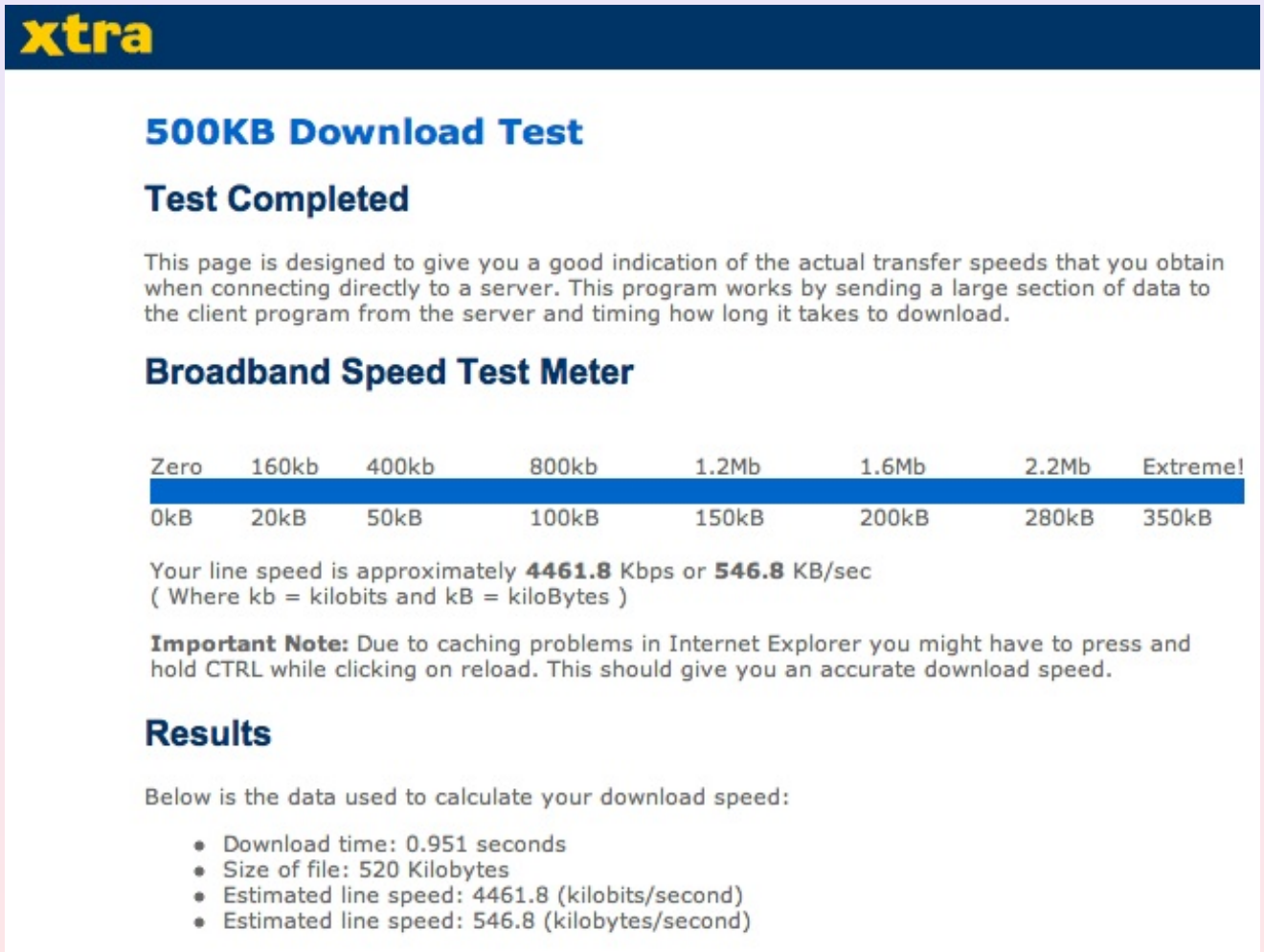
Trace route


Network Utility
Info
Netstat
AppleTalk
Ping
Lookup
Traceroute
Whois
Finger
Port Scan
Please enter the network address to trace an internet route to
www.google.co.nz
(ex. 10.0.2.1 or www.example.com)
Trace
traceroute: Warning: www.google.co.nz has multiple addresses; using 66.102.7.147
traceroute to www.google.com (66.102.7.147), 64 hops max, 40 byte packets
1 192.168.100.254 (192.168.100.254) 1.044 ms 0.756 ms 0.836 ms
2 219-88-160-1.jetstream.xtra.co.nz (219.88.160.1) 46.875 ms 41.892 ms 39.913 ms
3 203.96.122.59 (203.96.122.59) 33.867 ms 35.652 ms 36.305 ms
4 * mdr-ip03e-2-1-0.akbr3.global-gateway.net.nz (202.50.236.110) 63.088 ms *
5 * * *
6 p6-3.sjbr1.global-gateway.net.nz (203.96.120.82) 215.828 ms 216.192 ms 214.346 ms
7 p0-3-0-0.pabr3.global-gateway.net.nz (202.37.246.201) 215.219 ms 216.281 ms 241.252
ms
8 * * *
9 209.85.130.6 (209.85.130.6) 218.367 ms 216.199 ms 214.268 ms
10 72.14.233.131 (72.14.233.131) 218.056 ms 66.249.94.226 (66.249.94.226) 216.738 ms
218.380 ms
11 72.14.233.129 (72.14.233.129) 218.954 ms 216.239.49.66 (216.239.49.66) 217.624 ms
218.805 ms
12 mc-in-f147.google.com (66.102.7.147) 218.774 ms 219.030 ms 215.860 ms

Commercial bit rates in NZ (2007)

- Dial-up: [up to] 56 Kb/s
- Telecom (ADSL): "Maximum speed - as fast as your phone line allows."
- Telstra (ADSL): 2 Mb/s downstream and upstream
- Woosh (wireless): [up to] "40 times faster than dial-up"

Bit rates: xtra test (Ethernet, Glendowie) 1



xtra

500KB Download Test

Test Completed

This page is designed to give you a good indication of the actual transfer speeds that you obtain when connecting directly to a server. This program works by sending a large section of data to the client program from the server and timing how long it takes to download.

Broadband Speed Test Meter

Zero	160kb	400kb	800kb	1.2Mb	1.6Mb	2.2Mb	Extreme!
0kB	20kB	50kB	100kB	150kB	200kB	280kB	350kB

Your line speed is approximately **4461.8** Kbps or **546.8** KB/sec
(Where kb = kilobits and kB = kiloBytes)

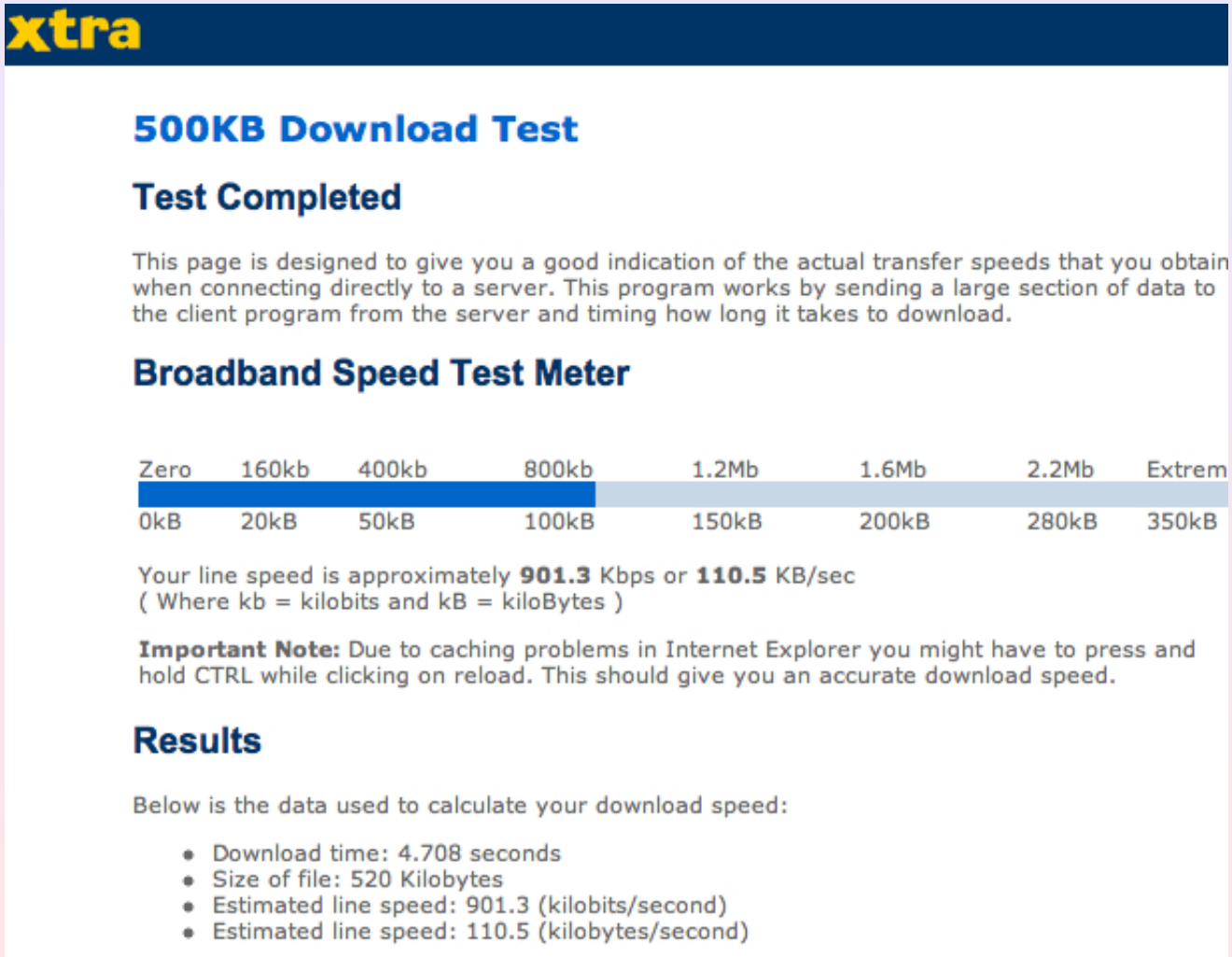
Important Note: Due to caching problems in Internet Explorer you might have to press and hold CTRL while clicking on reload. This should give you an accurate download speed.

Results

Below is the data used to calculate your download speed:

- Download time: 0.951 seconds
- Size of file: 520 Kilobytes
- Estimated line speed: 4461.8 (kilobits/second)
- Estimated line speed: 546.8 (kilobytes/second)

Bit rates: xtra test (Ethernet, Mt. Eden) 2



xtra

500KB Download Test

Test Completed

This page is designed to give you a good indication of the actual transfer speeds that you obtain when connecting directly to a server. This program works by sending a large section of data to the client program from the server and timing how long it takes to download.

Broadband Speed Test Meter

Zero	160kb	400kb	800kb	1.2Mb	1.6Mb	2.2Mb	Extrem
0kB	20kB	50kB	100kB	150kB	200kB	280kB	350kB

Your line speed is approximately **901.3** Kbps or **110.5** KB/sec
(Where kb = kilobits and kB = kiloBytes)

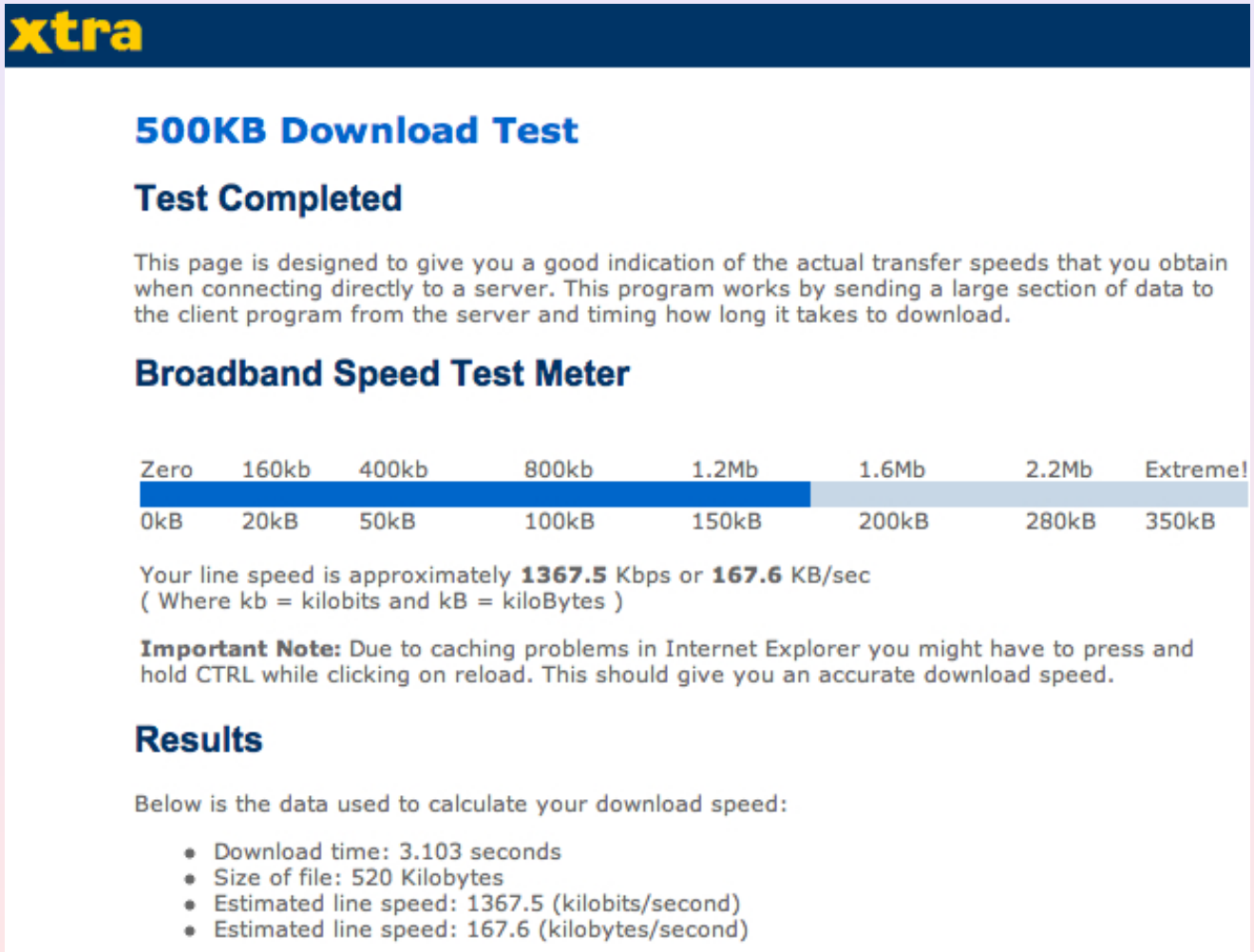
Important Note: Due to caching problems in Internet Explorer you might have to press and hold CTRL while clicking on reload. This should give you an accurate download speed.

Results

Below is the data used to calculate your download speed:

- Download time: 4.708 seconds
- Size of file: 520 Kilobytes
- Estimated line speed: 901.3 (kilobits/second)
- Estimated line speed: 110.5 (kilobytes/second)

Bit rates: xtra test (wireless, Mt. Eden) 3



xtra

500KB Download Test

Test Completed

This page is designed to give you a good indication of the actual transfer speeds that you obtain when connecting directly to a server. This program works by sending a large section of data to the client program from the server and timing how long it takes to download.

Broadband Speed Test Meter

Zero	160kb	400kb	800kb	1.2Mb	1.6Mb	2.2Mb	Extreme!
0kB	20kB	50kB	100kB	150kB	200kB	280kB	350kB

Your line speed is approximately **1367.5** Kbps or **167.6** KB/sec
(Where kb = kilobits and kB = kiloBytes)

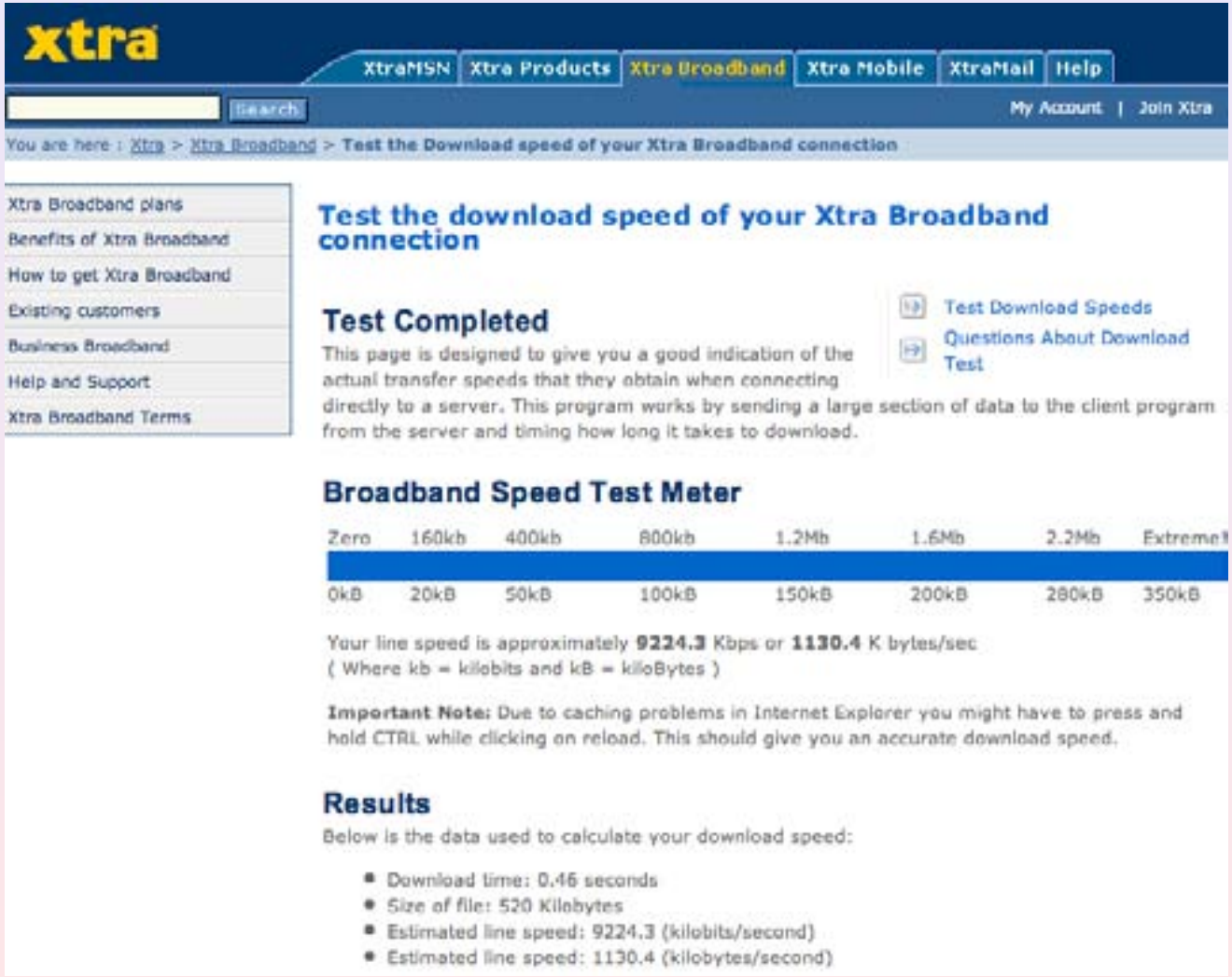
Important Note: Due to caching problems in Internet Explorer you might have to press and hold CTRL while clicking on reload. This should give you an accurate download speed.

Results

Below is the data used to calculate your download speed:

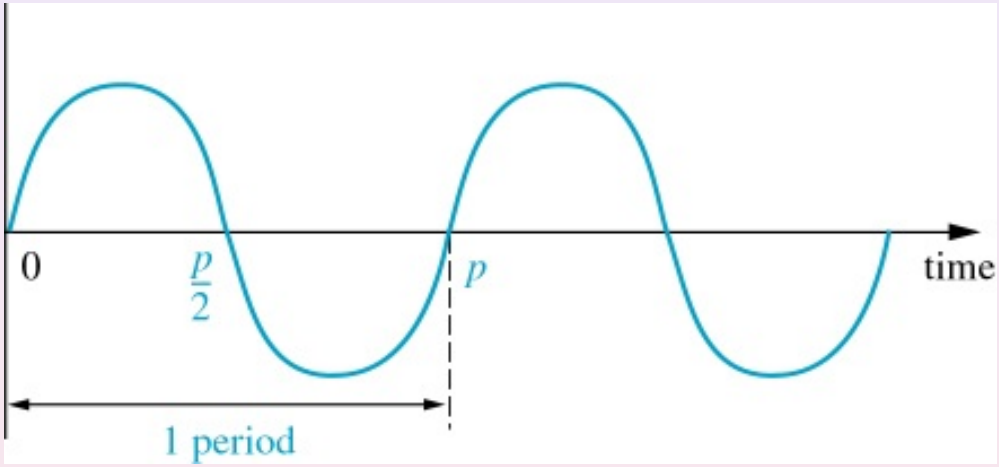
- Download time: 3.103 seconds
- Size of file: 520 Kilobytes
- Estimated line speed: 1367.5 (kilobits/second)
- Estimated line speed: 167.6 (kilobytes/second)

Bit rates: xtra test (Ethernet, UoA) 4



Bandwidth 1

Some analog signals



are **periodic**, i.e. they repeat a pattern/cycle continuously. The **period** of a signal is the time required by the signal to complete one cycle. The signal above has period p.

Bandwidth 2

A signal's frequency, f, is the number of cycles through which the signal can oscillate in a second.

The frequency and period are related as follows:

$$f = \frac{1}{p}.$$

The unit of measurement is "cycles per second", or **hertz (Hz)**.

Bandwidth 3

Suppose the period is

$$p = 0.5 \text{ microsecond } (\mu\text{s}) = 0.5 \times 10^{-6} \text{ s}$$

Then, the frequency is

$$f = \frac{1}{0.5 \times 10^{-6}} = 2 \times 10^{6} \text{ Hz } = 2 \text{ MHz.}$$

Bandwidth 4

The **bandwidth** is equal to the difference between the highest and lowest frequencies that can be transmitted.

- A telephone signal can handle frequencies in the range 300 to 3,300 Hz, so its bandwidth is equal to 3,300 - 300 = 3,000 Hz; very high (low) pitched sounds cannot pass through the telephone.
- Sometimes the term bandwidth is used for the amount of traffic between a web site and the rest of the Internet.

How much is?

- How much is 200MB?
  About 40 music files (average of 5MB per song) or stream 80 minutes of **news, sport and entertainment** (NSE).
- But 1GB?
  About 200 music files or stream 400 minutes of NSE.
- But 5GB (5,000MB)?
  About 1000 music files or stream 2000 minutes NSE.
- But 10GB (10,000MB)?
  About 2000 music files or stream 4000 minutes of NSE.
- But 20GB (20,000MB)?
  About 4000 music files or stream 8000 minutes of NSE.

Internet Protocol version 6 (IPv6) 1

People use words to navigate around the web, computers use numbers. You may type

`www.cs.auckland.ac.nz`

into a browser bar but the PC trying to reach that site will use a numerical equivalent that it gets from the net's *master address books*, in this case:

130.216.33.106

A tool to find the IP address of a website (plus the hosting) location can be found at:
`www.selfseo.com/find_ip_address_of_a_website.php`.

IP version 6 (IPv6) 2

IPv4, the fourth iteration IP, is widely deployed. IPv4 uses 32-bit (4-byte) addresses, which limits the address space to $2^{32} = 4,294,967,296$ possible unique addresses.

On 4 February 2008 the master (or root) servers for the net had a small number of records added that are written in IP version 6 (IPv6).
IPv6 supports $2^{128} \approx 3.4 \times 10^{38}$ addresses, or approximately 5×10^{28} addresses for each of the roughly 6.5 billion people alive today.

Projections suggest that the unallocated pool will run out by 2011 at the latest.

How is information coded?

Whether the medium uses light, electricity, or microwaves, we must answer perhaps the most basic of all communications questions:

How is information coded in a format suitable for transmission?

Bits

Regardless of implementation, all switches are in one of two states: open or closed, symbolically, 0 and 1.

Bits can store only two distinct pieces of information. Grouping them, allows for many combinations:

- two bits allow $2^2 = 4$ unique combinations: $00, 01, 10, 11$
- three bits allow for $2^3 = 8$ combinations,
- ten bits allow for $2^{10} = 1,024$ combinations,
- fifty bits allow for $2^{50} = 1,125,899,906,842,624$ combinations,
- n bits allow for 2^n combinations.

From bits to codes

Grouping bits allows one to associate certain combinations with specific items such as characters, numbers, pictures. Loosely speaking, this association is called a **code**. Not every association is a code as we shall soon learn.

A difficult problem in communications is to establish communications between devices that operate with different codes. There are standards, but not all standards are compatible!

> *The only problem with standards is that there are so many of them.*

Early codes: Morse 1

Originally created for Morse's electric telegraph in 1838, by the American inventor Samuel Morse, the **Morse code** was also extensively used for early radio communication beginning in the 1890s.

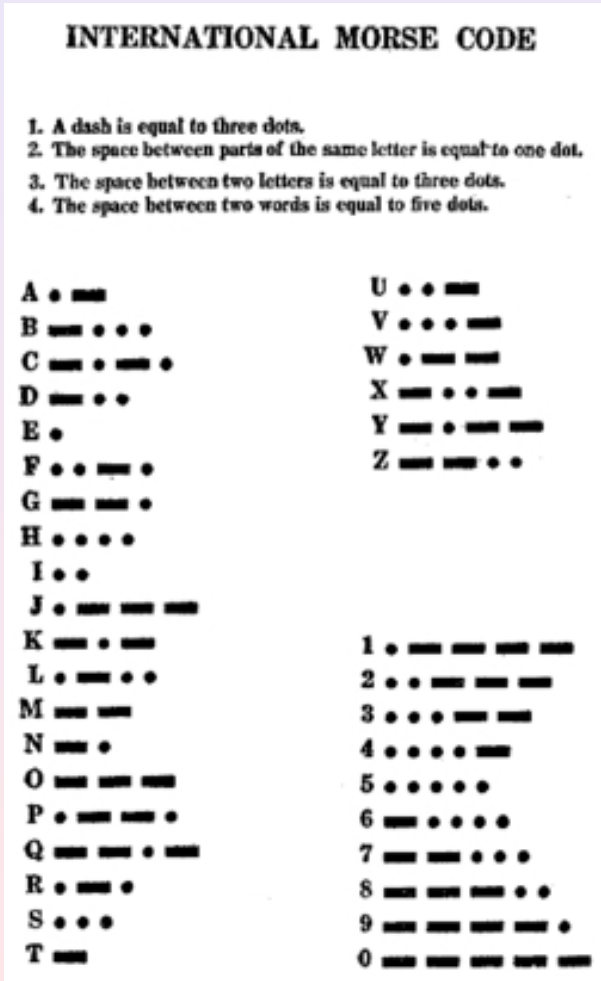
The telegraph required a human operator at each end. The sender would tap out messages in Morse code which would be transmitted down the telegraph wire to a human decoder translating them back into ordinary characters.

Early codes: Morse 2

Morse code is transmitted using just two states—on and off—so it was an early form of a digital code. It is redundant binary code, as the pause lengths are required to decode the information.
International Morse code is composed of six elements:

- **short mark, dot or 'dit' (·)**
- **longer mark, dash or 'dah' (-)**
- intra-character gap (between the dots and dashes within a character)
- short gap (between letters)
- medium gap (between words)
- long gap (between sentences — about seven units of time)

Early codes: Morse 3



INTERNATIONAL MORSE CODE

1. A dash is equal to three dots.
2. The space between parts of the same letter is equal to one dot.
3. The space between two letters is equal to three dots.
4. The space between two words is equal to five dots.

A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —	1	• — — — —
K	— • —	2	• • — — —
L	• — • •	3	• • • — —
M	— —	4	• • • • —
N	— •	5	• • • • •
O	— — —	6	— • • • •
P	• — — •	7	— — • • •
Q	— — • —	8	— — — • •
R	• — •	9	— — — — •
S	• • •	0	— — — — —
T	—		

Early codes: Morse 4

Morse code is a **variable-length code**:

a) letter codes have different lengths; the letter E code is a single dot, the letter H code has four dots;

b) codes for digits are also variable in length.

Reason: most frequent letters are assigned short codes, so messages can be send quickly.

Early codes: Morse 5

Morse code is still in use today by radio amateurs and until rather recently was used in shipping.

An experienced operator can handle about 30 words per minute (standard word is 5 characters as in "Paris") and some higher than that. This is faster than most people can hand-write.

Early codes: Baudot code 1

The **Baudot code**—also known as **International Telegraph Alphabet No 2 (ITA2)**—is named after its French inventor Émile Baudot. ITA2 is a fix-length code using 5 bits for each character (digits and letters). This code was developed around 1874.

With 5-bit codes we can name $2^5 = 32$ different objects, but we have 36 letters and digits (plus special characters) to code!

For example, the letter Q and digit 1 have the same code: 10111. In fact each digit's code duplicates that of some letter.

Early codes: Baudot code 2

00	01	02	03	04	05	06	07
NUL	E 3	LF	A -	SP	S ’	I 8	U 7
08	09	0A	0B	0C	0D	0E	0F
CR	D ENQ	R 4	J BEL	N ,	F !	C :	K (
10	11	12	13	14	15	16	17
T 5	Z +	L)	W 2	H £	Y 6	P 0	Q 1
18	19	1A	1B	1C	1D	1E	1F
O 9	B ?	G &	FIGS	M .	X /	V ;	LTRS

Letters | Figures | Control Chars.

Early codes: Baudot code 3

Do you think we have got a problem?
More precisely, how can we tell a digit from a letter?

Answer: using the same principle that allows a keyboard key to represent two different characters. On the keyboard we use the Shift key; the Baudot code uses the extra information

11111 (shift down) and 11011 (shift up)

to determine how to interpret a 5-bit code. Upon receiving a shift down, the receiver decodes all codes as letters till a shift up is received, and so on.

Early codes: Baudot code 4

Here is an example.
ABC123, is coded from left to right as follows:

11111 00011 11001 01110 11011 10111 10011 00001

Early codes: BCD, BCDIC, ASCII codes

- BCD stands for **binary-coded decimal**, a code developed by IBM for its mainframe computers using 6-bit codes;
- BCDIC stands for **binary-coded decimal interchange code**, an expansion of BCD including codes also for non-numeric data;
- ASCII (pronounced ['æski]) stands for the **American Standard Code for Information Interchange**; it is a 7-bit code that assigns a unique combination to every keyboard character and to some special functions.

ASCII code (decimal, binary, hexadecimal) 1

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII code (decimal, binary, hexadecimal) 2

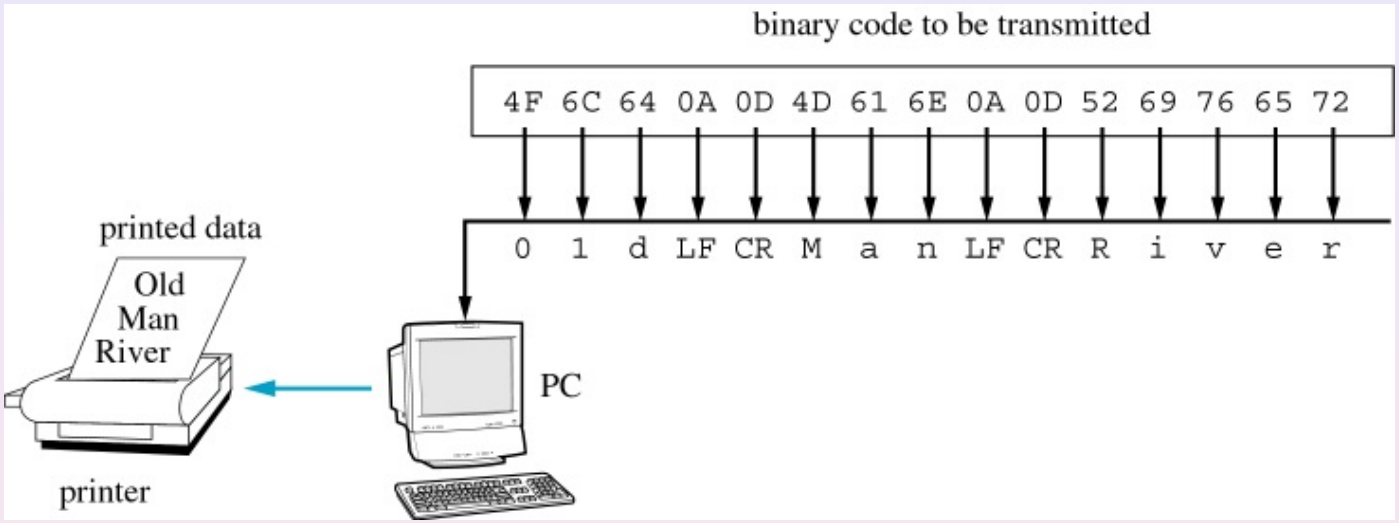
Each code corresponds to a printable or unprintable character.

Printable characters include letters, digits, and special punctuation (commas, brackets, question marks).

Unprintable characters are special functions (line feed, tab, carriage return).

Standard ASCII includes 128 different characters, and Extended ASCII (8-bit codes) adds an extra 128 characters.

ASCII code 3



If codes are sent with the leftmost first, as the printer receives each code, it analyses and takes some action: for 4F, 6C and 64 it prints 0, 1, and d. The next two codes, OA and OD, denote unprintable characters (LF = line feed, CR = carriage return). When OA is received, nothing is printed, but the mechanism to advance to the next line is activated.

What is a code? 1

We can now ask the important question:

What is a code?

Here is an example. The character "=" is represented by the binary code word "0111101". Why do we need the leading zero? Surely "111101" means the same thing because both "09" and "9" mean nine?

All ASCII codes have the same length. This ensures that an important property—called the prefix property—holds true for the ASCII code.

What is a code? 2

A **code** is the assignment of a unique string of characters (a **codeword**) to each character in an alphabet.

A code in which the codewords contain only zeroes and ones is called a **binary code**.

The encoding of a string of characters from an alphabet (the cleartext) is the concatenation of the codewords corresponding to the characters of the cleartext, in order, from left to right. A code is **uniquely decodable** if the encoding of every possible cleartext using that code is unique.

What is a code? 3

For example, here are two possible binary codes for the alphabet $\{a, c, j, l, p, s, v\}$:

	code 1	code 2
a	1	010
c	01	01
j	001	001
l	0001	10
p	00001	0
s	000001	1
v	0000001	101

What is a code? 4

Both code 1 and code 2 satisfy the definition of a code. However,

- code 1 is uniquely decodable, but
- code 2 is not uniquely decodable; for example, the encodings of the cleartexts “pascal” and “java” are both

001010101010

Prefix codes 1

A **prefix code** is a code with the "prefix property":

> *no codeword is a (proper) prefix of any other codeword in the set.*

The code $\{0, 10, 11\}$ has the prefix property; the code $\{0, 1, 10, 11\}$ does not, because "1" is a prefix of both "10" and "11".

Code 1 is a prefix code, but code 2 is not.

> *Why is the prefix property important?*

Because prefix codes are uniquely decodable.

Prefix codes 2

More examples:

- every fix-length code is a prefix code: no codeword can be a prefix of any other codeword because no codeword is any longer or shorter than any other;
- ASCII is a prefix code (it's a 7-bit code);
- the Morse code is not a prefix code: for example, the codeword for A is a prefix of the codeword for J (and L); how can the Morse code be useful?

Prefix codes 3

Given a sequence of lengths, can we construct a prefix code whose codewords have exactly those lengths?

Kraft's theorem. *A prefix code exists for codewords lengths* $l_1, l_2, \ldots, l_N$ *if and only if*

$$2^{-l_1} + 2^{-l_2} + \cdots + 2^{-l_N} \leq 1.$$

Arrange the lengths in increasing order so (after relabelling) $l_1 \leq l_2 \leq \ldots \leq l_N$. Take as the first codeword, $w_1 = 0^{l_1}$, i.e. $00\ldots0$ for l_1 times. If $w_1, w_2, \ldots, w_i$ have been constructed, choose w_{i+1} to be the first string (lexicographically) of length l_{i+1} such that no w_j is a prefix of it. The above inequality guarantees that w_{i+1} always exists.

Prefix codes 4

Here is an example. Consider the lengths $3, 2, 4$ which satisfy the condition in Kraft theorem:

$$2^{-3} + 2^{-2} + 2^{-4} = \frac{7}{16} < 1.$$

We arrange the lengths in increasing order, $2, 3, 4$, and we have:
$w_1 = \mathbf{00}$
w_2 = first string in the set 000,001,010,011,100, 101, 110,111 such that **00** is not its prefix = **010**
w_3 = first string in the set 0000,0001,0010,0011,0100, 0101, 0110,0111, 1000,1001,1010,1011,1100, 1101, 1110,1111 such that its prefixes do not include both **00** and **010** = **0110**

Prefix codes 5

Kraft's theorem does not assert that any code which satisfies the inequality therein must be a prefix code. For example, the code

$$\{00, 010, 0100\}$$

satisfies the inequality but is not a prefix code.

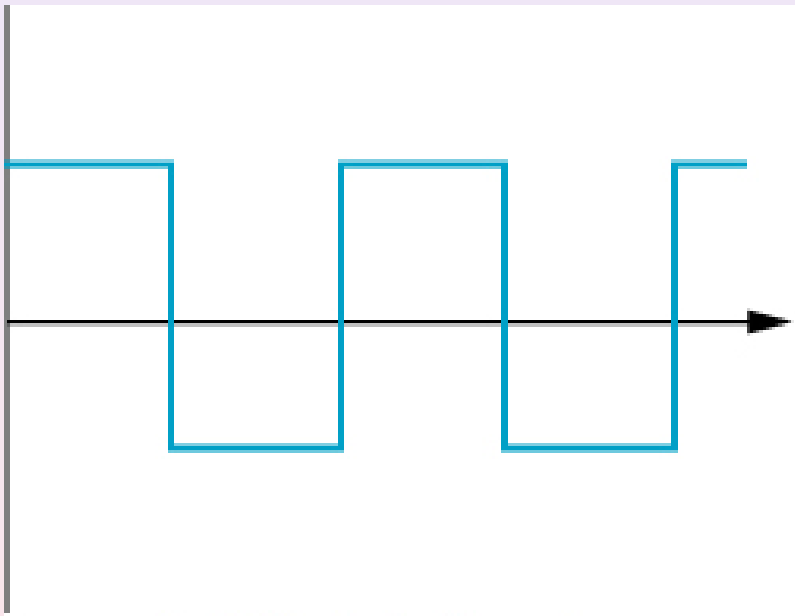
However,

> *Any uniquely decodable code can be replaced by a prefix code without changing any of the lengths of the codewords.*

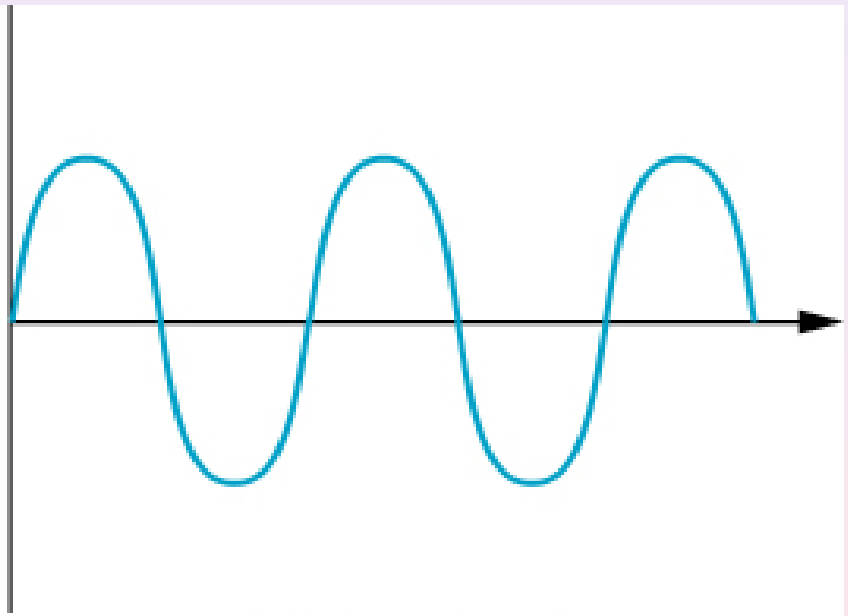
Questions

1. How symbolic data relate to electrical signals, microwaves, or light waves?
2. What does a 0 or a 1 actually look like as it travels through a wire, optical fibre, or space?
3. How many bits can a signal transmit per unit of time?
4. Are there limitations?
5. Can electrical interference (noise) affect data? If so, how?

Analog vs. digital signals



(a) Digital signal



(b) Analog signal

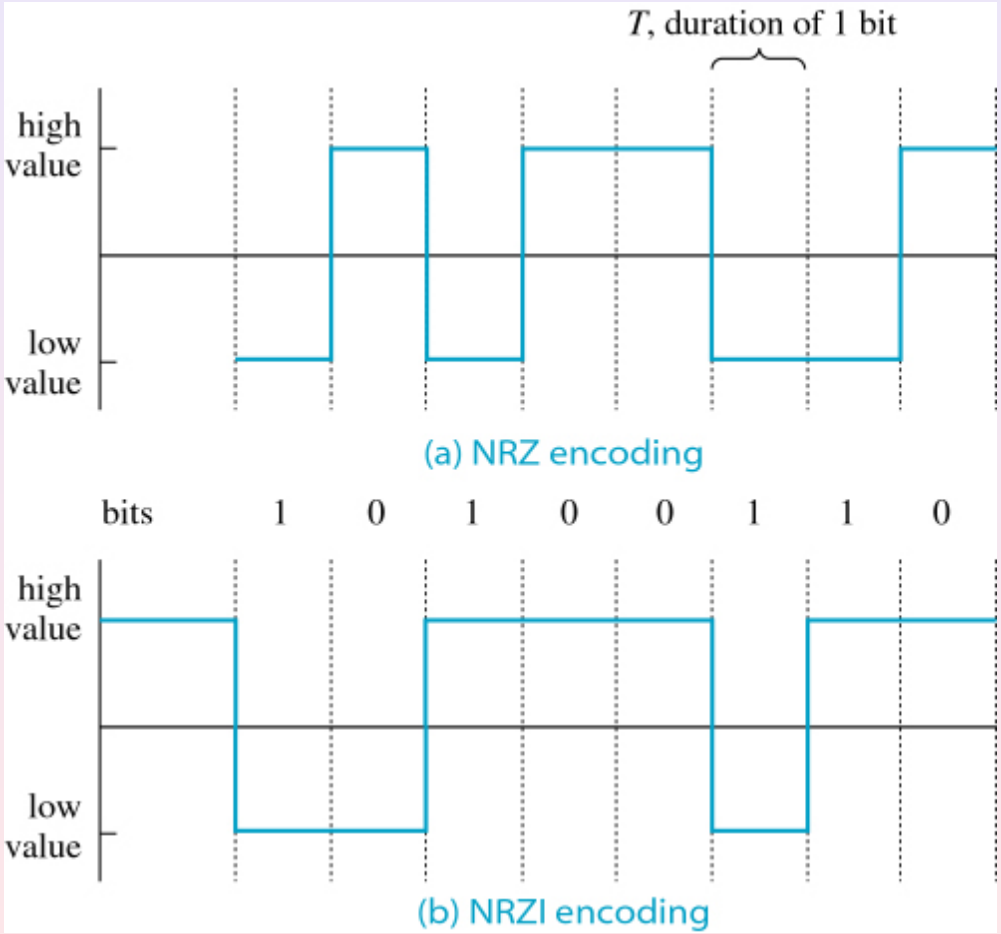
Analog signals 1

Because digital signals can alternate between two constant values—say "high voltage" and "low voltage"—we simply associate 0 with one value and 1 with the other. The actual values are irrelevant.

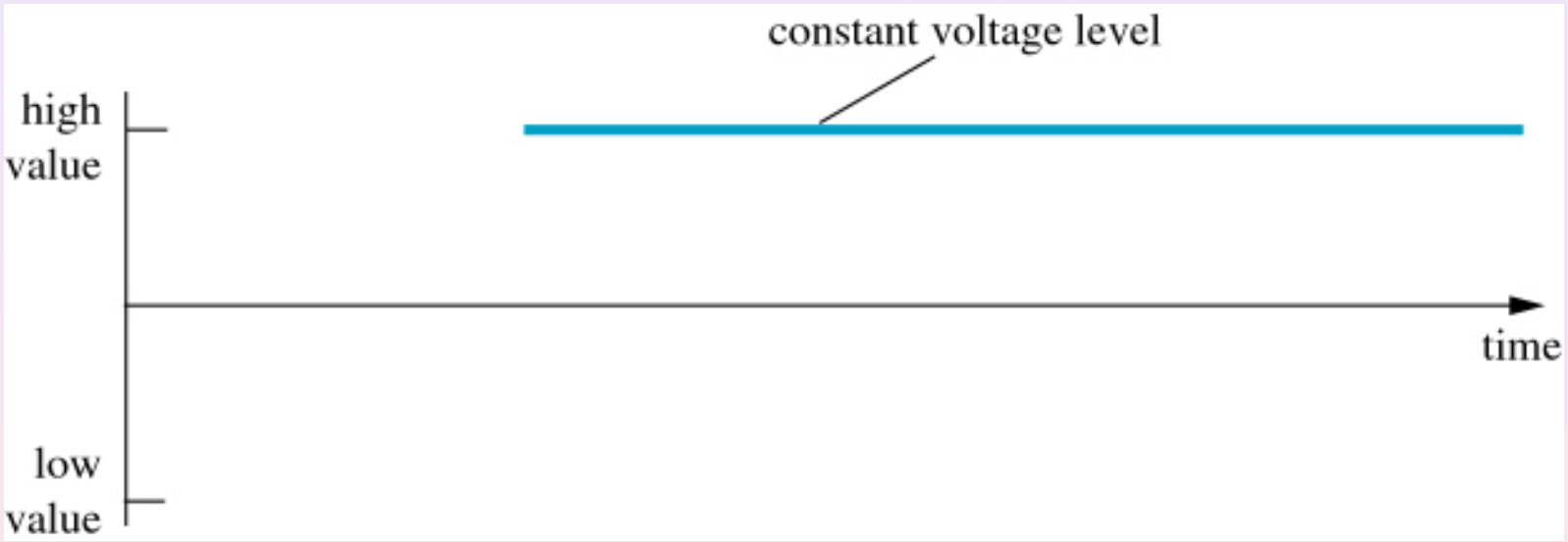
Non-Return to Zero (NRZ): a 0 is transmitted by raising the voltage level to high, and a 1 is transmitted using a low voltage. Alternating between high and low voltage allows for the transmission of any string of 0s and 1s. Used in Controller Area Network (CAN).

Non-Return-to-Zero-Inverted Encoding (NRZI): a 0 is encoded as no change in the level. However a 1 is encoded depending on the current state of the line. If the current state is 0 [low] the 1 will be encoded as a high, if the current state is 1 [high] the 1 will be encoded as a low. Used in USB.

Analog signals 2



Analog signals 3



What is being transmitted? A string of 0s. But, how many? One can use a clock, but clocks don't synchronise well.

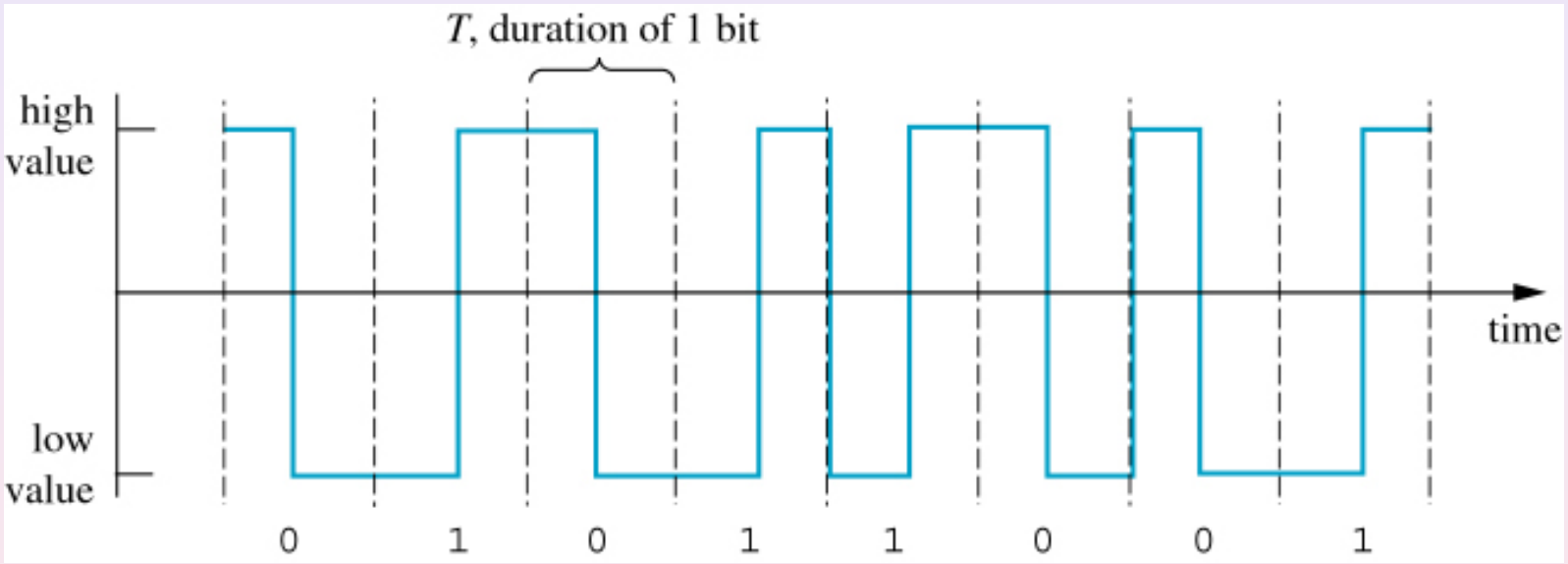
Analog signals 4

The **Manchester code** uses signal changes to keep the sending/receiving devices synchronised. It encodes 0 and 1 by changing the voltage:

0 is represented by a change from high to low and
1 is represented by a change from low to high.

The signal will never be held constant for a time longer than a single bit interval. Even for strings of 0s or 1s, the signal will change in the middle of the interval.

Analog signals 5



Clearly, this is a better code. Do we pay a price? Yes, for example, signals must change twice as frequently as with NRZ coding.

Conversion: analog to digital and back

The process of converting a digital signal (e.g. from a PC) to an analog signal is called **modulation**. The converse conversion, from analog to digital, is called **demodulation**.

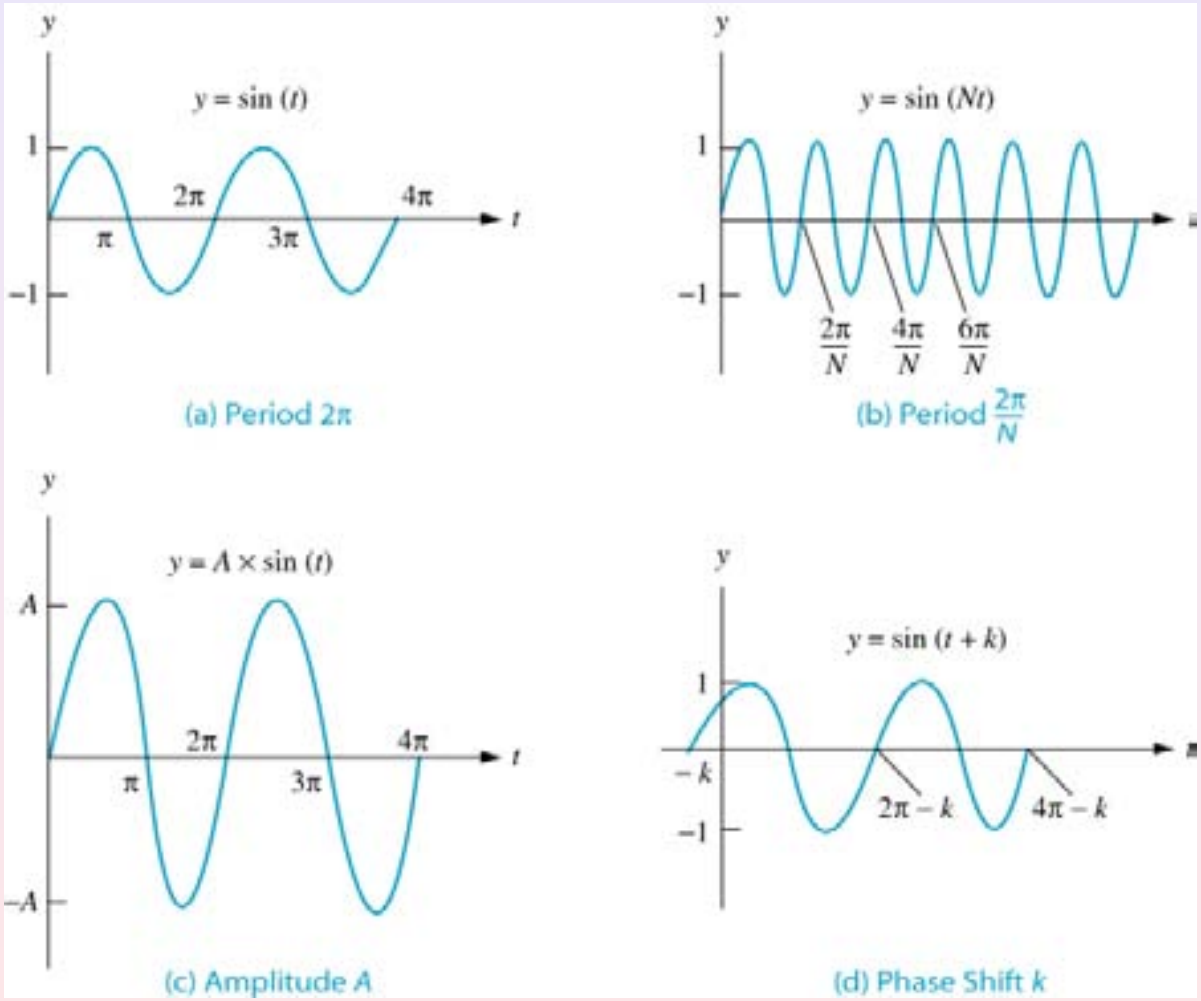
A **modem**, short for modulation/demodulation, is a device that does both conversions.

Understanding analog signals 1

An analog signal corresponds to a sine (wave) function. There are three main ways to "fine tune" such a signal:

1. changing the frequency,
2. changing the amplitude,
3. changing the phase.

Understanding analog signals 2



Understanding analog signals 3

1. the **period**, p, is the time it takes to complete a pattern,
2. the **frequency**, f, is the number of times the signal oscillates per unit of time ((hertz) Hz if time is measured in seconds); $f = 1/p$,
3. the **amplitude** is the range in which the signal oscillates; the **peak amplitude** is the absolute value of signal's highest intensity),
4. the **phase** shift describes the position of the signal relative to time zero; it is obtained by adding or subtracting k from the sine argument ($\sin(t + k)$, $\sin(t - k)$).

Understanding analog signals 4

Here are some numerical examples with reference to the previous picture:

In (a) the period for $y = sin(t)$ is $p = 2\pi$,

In (b) the period for $y = sin(Nt)$ is $p = 2\pi/N$,

In (b) the frequency is $f = 1/p = N/2\pi$ Hz,

The amplitudes in (a), (b), (d) is $[-1, 1]$, but in (c) is $[-A, A]$; the peak amplitudes are respectively, 1 and A.

Understanding analog signals 5

A sine function can be written in the form:

$$s(t) = A\sin(2\pi ft + k),$$

where $s(t)$ is the instantaneous amplitude at time t, A is the peak amplitude, f is the frequency, and k is the phase shift.

These three characteristics fully describe a sine function.

Understanding analog signals 6

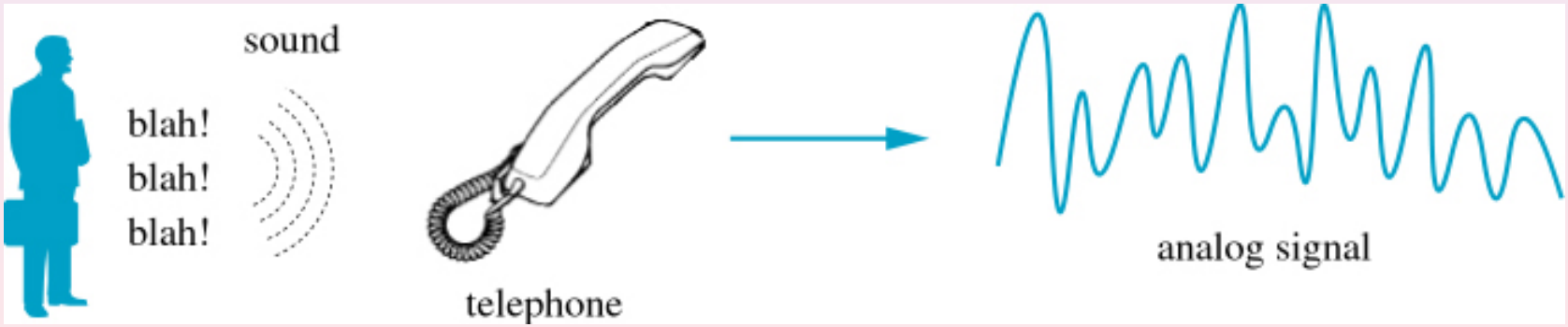
For electrical signals, the peak amplitude is measured in volts. The frequency is measured in hertz.

Unit	Equivalent	Unit	Equivalent
second (s)	1 s	hertz (Hz)	1Hz
millisecond (ms)	10^{-3} s	kilohertz (KHz)	10^{3} Hz
microsecond (μs)	10^{-6} s	megahertz (MHz)	10^{6} Hz
nanosecond (ns)	10^{-9} s	gigahertz (GHz)	10^{9} Hz
picosecond (ps)	10^{-12} s	terahertz (THz)	10^{12} Hz

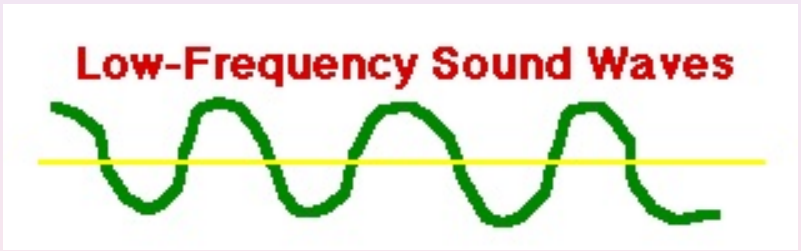
Understanding analog signals 7

Sound translates into electrical analog signals. There are nice relations between sound and signals:

- the amplitude reflects the volume,
- the frequency reflects the pitch.



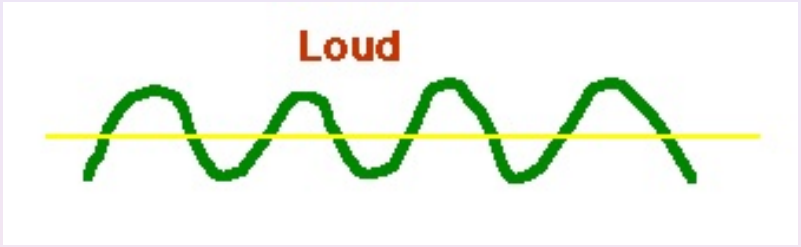
Understanding analog signals 8

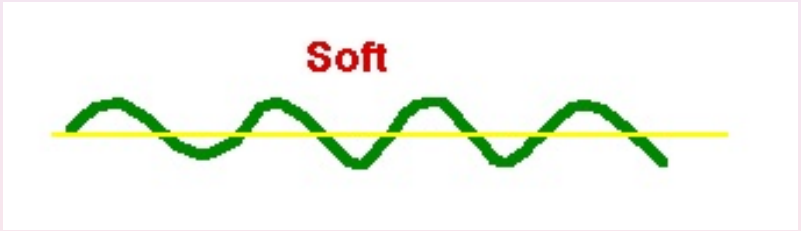


source: www.fi.edu/fellows/fellow2/apr99/soundvib.html

The upper part has a piece slanting to the left (at the top of the second high part), i.e. *backwards* in time!

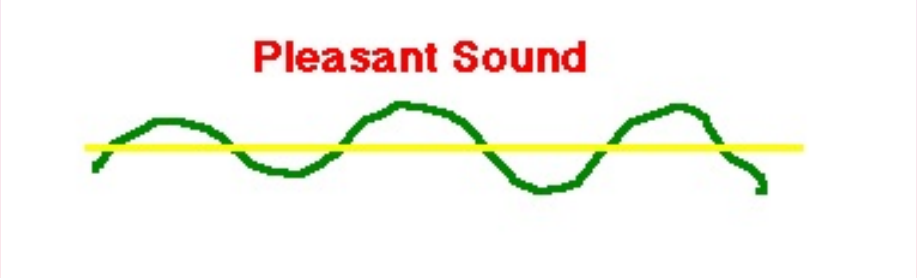
Understanding analog signals 9





source: www.fi.edu/fellows/fellow2/apr99/soundvib.html

Understanding analog signals 10



source: www.fi.edu/fellows/fellow2/apr99/soundvib.html

Understanding analog signals 11

A single sine function is not useful for data communications; we need to change one or more of its characteristics—amplitude, frequency and phase shift—to make it useful. We obtain composite signals.
Do we need different hardware devices for different signal types?
Do the functions that represent various signals require separate analysis?

The answer to both questions is no.

The French mathematician Jean Baptiste Fourier proved that any periodic function can be expressed as an infinite sum of sine and cosine functions of varying amplitudes, frequencies and phase shift—a **Fourier series**.

Understanding analog signals 12

A periodic signal $s(t)$ with period P can be described by a Fourier series as follows:

$$s(t) = \frac{a_0}{2} + \sum_{i=1}^{\infty} \left[a_i \times \cos \frac{2\pi it}{P} + b_i \times \sin \frac{2\pi it}{P} \right]$$

The coefficients $a_i, i = 0, 1, \ldots$ and $b_i, i = 1, 2, \ldots$ are determined by some formulas:

$$a_i = \ldots$$

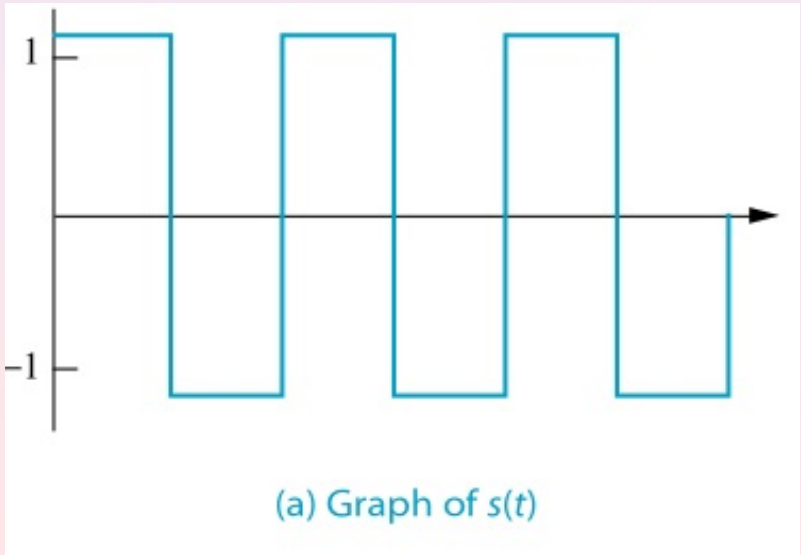
$$b_i = \ldots$$

Understanding analog signals 13

For example, the graph of the period function

$$s(t) = \begin{cases} 1, & \text{if } t \in [0,\pi) \cup [2\pi,3\pi) \cup [4\pi,5\pi) \cup \ldots, \\ -1, & \text{if } t \in [\pi,2\pi) \cup [3\pi,4\pi) \cup [5\pi,6\pi) \cup \ldots, \end{cases}$$

is



(a) Graph of $s(t)$

Understanding analog signals 14

How can we obtain the above graph with Fourier analysis? As the function $s(t)$ is periodic, it can be written as a Fourier series:

$$s(t) = \frac{a_0}{2} + \sum_{i=1}^{\infty} \left[a_i \times \cos \frac{2\pi i t}{P} + b_i \times \sin \frac{2\pi i t}{P} \right]$$

We need to determine P and the coefficients a_i and b_i: they are

$$P = 2\pi, a_i = 0, b_i = \begin{cases} 0, & \text{if } i \text{ is even}, \\ \frac{4}{\pi i}, & \text{if } i \text{ is odd}, \end{cases}$$

so

$$s(t) = \sum_{i=1, i \text{ odd}}^{\infty} \frac{4}{\pi i} \sin(it).$$

Understanding analog signals 15

Start with the series

$$s(t) = \sum_{i=1, i \text{ odd}}^{\infty} \frac{4}{\pi i} \sin(it) = \frac{4}{\pi} \sin(t) + \frac{4}{3\pi} \sin(3t) + \cdots$$

and write its general term as

$$\frac{4}{i\pi} \sin(it) = A \sin(2\pi fti + k),$$

where the amplitude, frequency and phase shift are

$$A = \frac{4}{i\pi}, f = \frac{1}{2\pi}, k = 0.$$

Understanding analog signals 16

We can write the series in the form

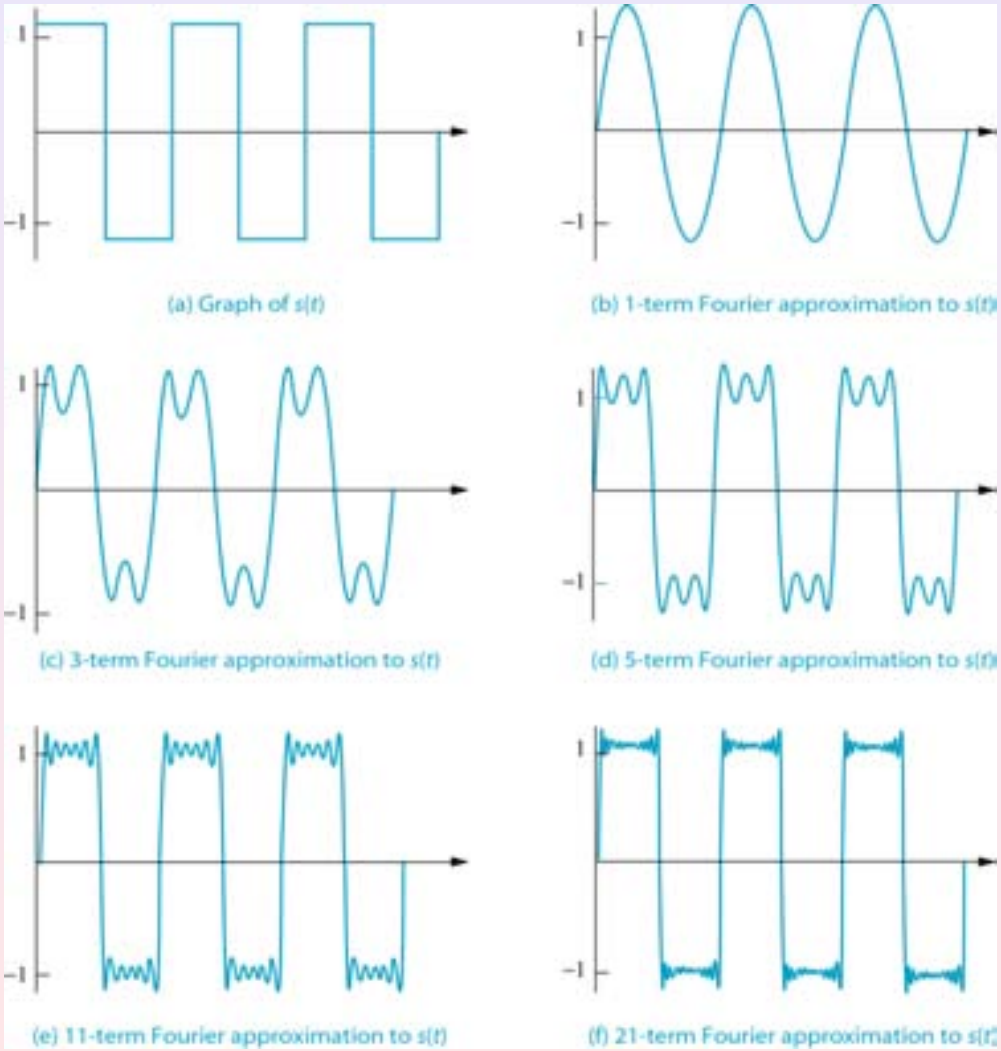
$$s(t) = \frac{4}{\pi}\sin(2\pi ft) + \frac{4}{3\pi}\sin[2\pi(3f)t] + \frac{4}{5\pi}\sin[2\pi(5f)t] + \cdots$$

So, the series sums up sine functions with frequencies $f, 3f, 5f, \ldots$ and amplitudes $\frac{4}{\pi}, \frac{4}{3\pi}, \frac{4}{5\pi}, \ldots$.

The term with frequency f is called the **fundamental frequency**; the term with frequency $3f$ is called the **third harmonic**, the term with frequency $5f$ is called the **fifth harmonic**, aso.

The best hope for calculating the infinite series is to approximate it using a finite number of its terms (using various techniques, *finite/fast Fourier transform*).

Understanding analog signals 17



Understanding analog signals 18

Two applications of Fourier transform:

- **Filters** block certain frequencies while allowing others to pass. Example: stereo equaliser.
- **Multiplexing**: receiving many TV channels is possible because various channels are assigned different frequencies (sound/image). Selecting a channel means allowing frequencies in a certain range to pass.

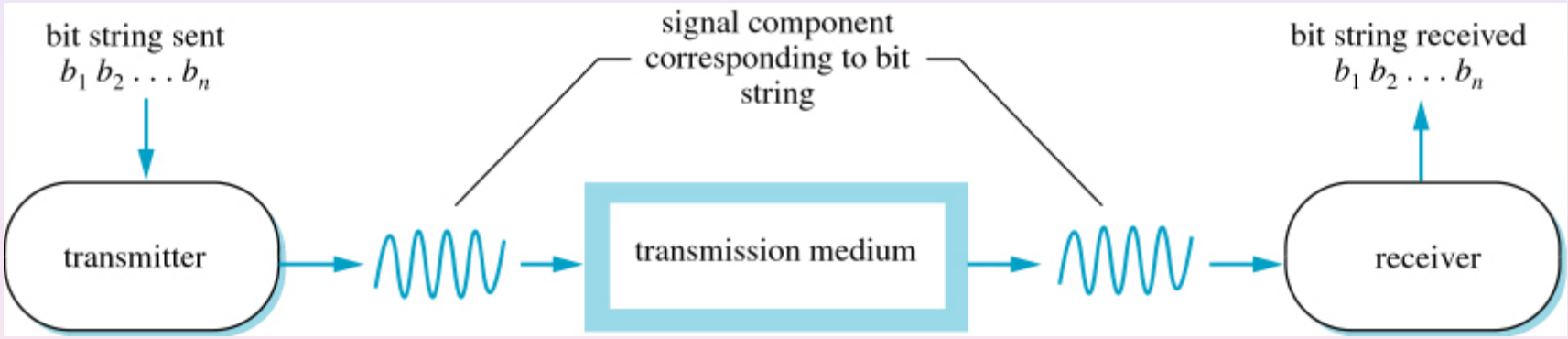
Bit rate 1

The **bit rate** describes a medium's capacity and is measured in bits per second (b/s). The range of frequencies a medium can pass is called **bandwidth**.

Simply put,

> *a higher-bandwidth medium is capable of a higher bit rate.*

Bit rate 2



Sending data via signals

The bit rate depends on: a) the frequency with which the components change, the **baud rate**, b) the number of bits transmitted.

Bit rate 3

We use **Baud** (after Émile Baudot) to measure the symbol rate, i.e. the number of distinct symbol changes made to the transmission medium per second in a digitally modulated signal; n is the number of bits transmitted. We have:

$$\text{bit rate} = \text{baud rate} \times n.$$

So, the bit rate can be increased by increasing the baud rate and the number of bits transmitted up to some point.
If the transmitter changes the signals at intervals of $1/2f$, then the receiver can completely reconstruct a signal by sampling it $2f$ times per second, i.e. the baud rate is $2f$.

Bit rate 4

Nyquist theorem. *In a distortion-free transmission,*

$$\textit{bit rate} = \textit{baud rate} \times n = 2 \times f \times n,$$

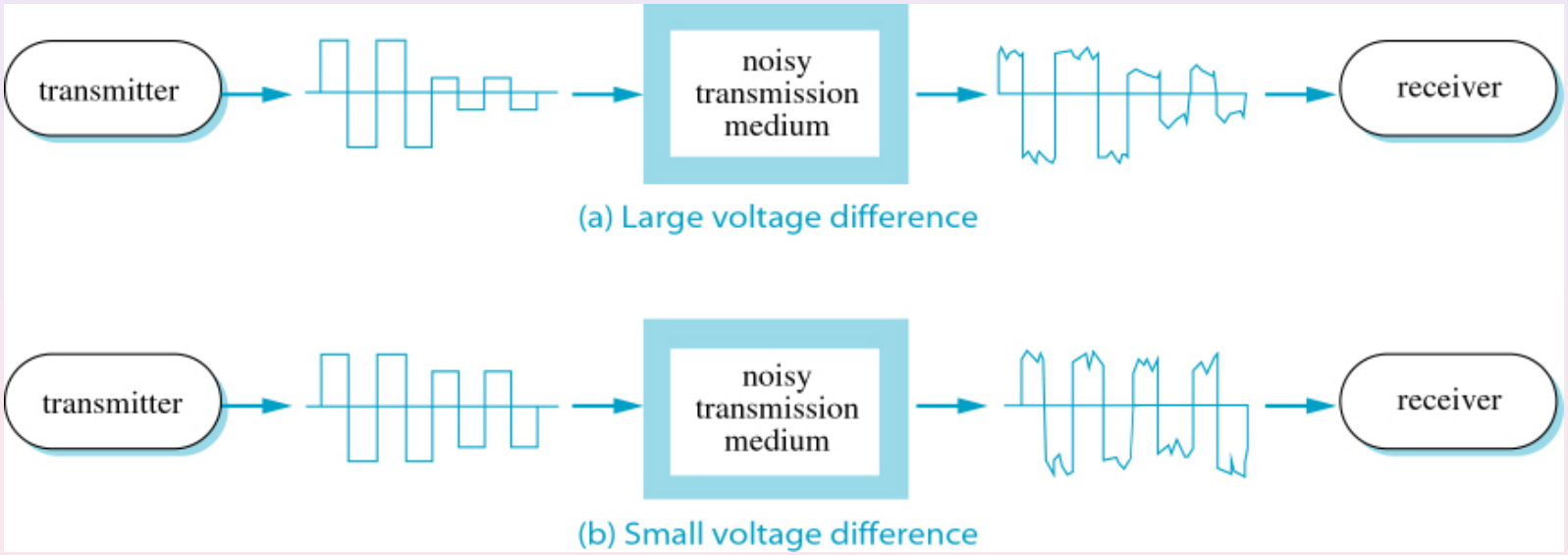
where f is the maximum frequency the medium can transmit.

n, number of bits per signal component	number of signal components	maximum bit rate (b/s)
1	2	6,600
2	4	13,200
3	8	19,800
4	16	26,400

Nyquist theorem for a maximum frequency (in symbols) of $f =$ 3300Hz

Noisy channels 1

Many channels are **noisy**. Note the voltage difference between the first and last part of the transmitted signal



Sending data via signals

Noisy channels 2

The **signal-to-noise (S/N) ratio** is measured by the ratio S/N, where S is the signal power and N is the noise power. A large signal-to-noise ratio indicates a clear signal, while a small signal-to-noise ratio means more distortion. As S is much larger than N, instead of S/N we use

$$\log_{10}(S/N)$$

where the unit of measurement is called **bel**. The more familiar **decibel** is defined by

$$1 \text{ dB} = 0.1 \text{ bel}.$$

Noisy channels 3

A rating of 25 dB means 2.5 bels, i.e.

$$B = \log_{10}(S/N) = 2.5.$$

This implies

$$S/N = 10^{2.5},$$

or

$$S = 10^{2.5} \times N = 100\sqrt{10} \times N \approx 316N.$$

Noisy channels 4

The American scientist Claude Shannon, inventor of the classical theory of information, improved Nyquist's theorem by taking into account the channel's noise:

Shannon's theorem. *In a noisy transmission,*

$$\textit{bit rate} = \ \textit{bandwidth} \times \log_2(1 + S/N).$$

If the signal-to-noise ratio is too small, noise can render different signals indistinguishable.

Noisy channels 5

The telephone has a bandwidth of about 3,000 Hz and a signal-to-ratio of approximately 35 dB. So,

$$3.5 = \log_{10}(S/N),$$

$$S = 10^{3.5} \times N \approx 3,162N.$$

Using Shannon's theorem we get:

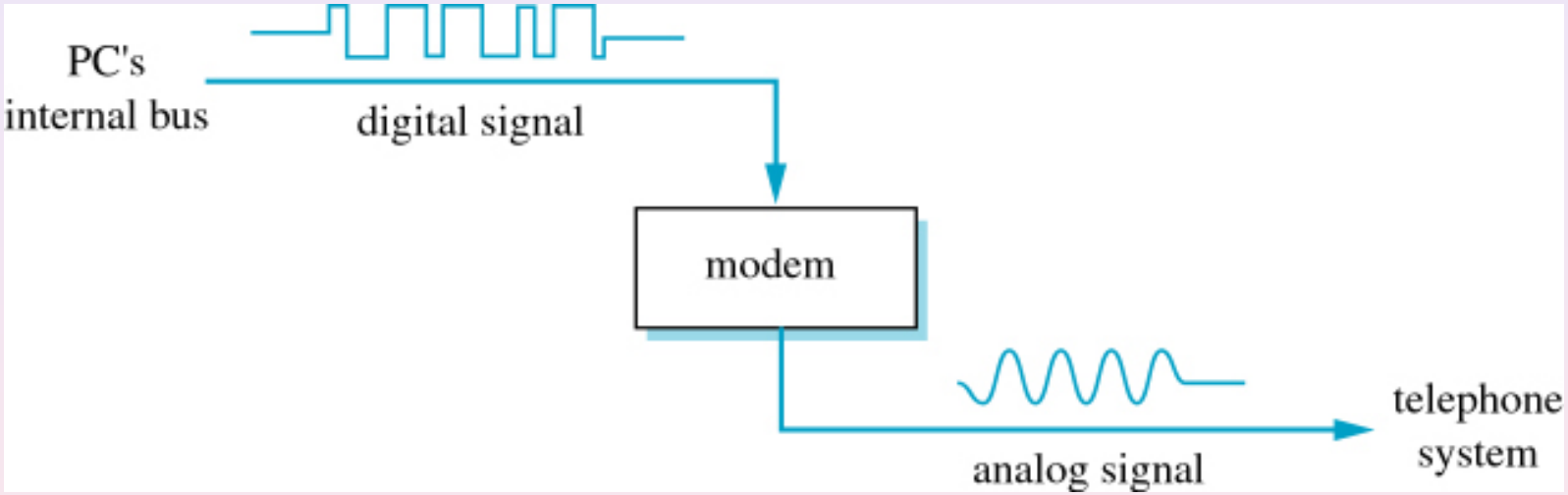
$$\begin{aligned} \text{bit rate} &= \text{bandwidth} \times \log_2(1 + S/N) \\ &= 3,000 \times \log_2(1 + 3,162) \text{ b/s} \\ &\approx 3,000 \times 11.63 \text{ b/s} \\ &\approx 34,880 \text{ b/s} \end{aligned}$$

Noisy channels 6

Why does a 56,000 b/s modem work over a 3,000Hz phone line contrary to what the slide seems to indicate?

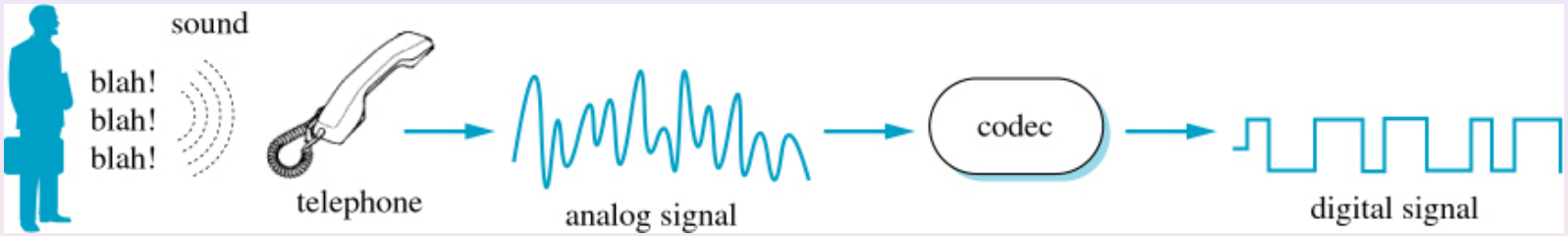
Answer: The 3,000 Hz is a minimum long-distance requirement—the local loop can support a slightly higher bandwidth at better S/N.

Digital to analog conversion 1



Computer data transmitted over telephone lines

Digital to analog conversion 2



Voice information transmitted digitally through optical fibres using **codec** (coder/decoder)

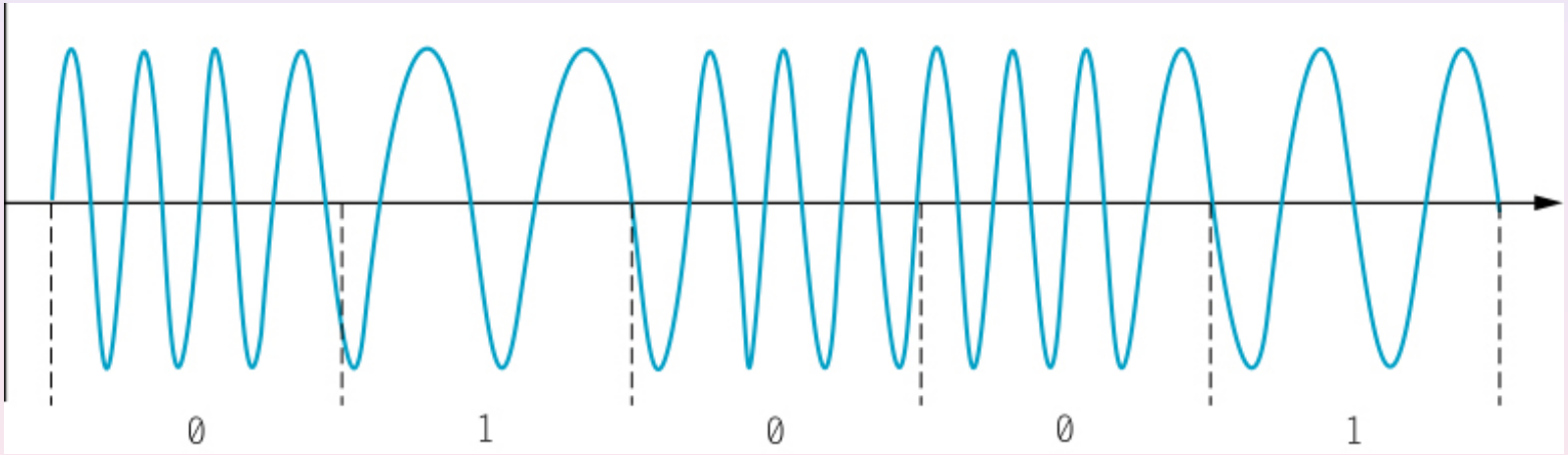
Digital to analog conversion 3

Converting a digital signal into an analog one is not difficult. There are three ways to do it:

1. by frequency, with the method called **frequency shift keying (FSK)**,
2. by amplitude modulation, with the method called **amplitude shift keying (ASK)**,
3. by phase modulation, with the method called **phase shift keying (PSK)**,
4. by a combination of amplitude and phase shift, with the method called **quadrature amplitude modulation (QAM)**.

Digital to analog conversion 4

FSK or **frequency modulation (FM)** assigns a digital 0 to one analog frequency and a 1 to another.

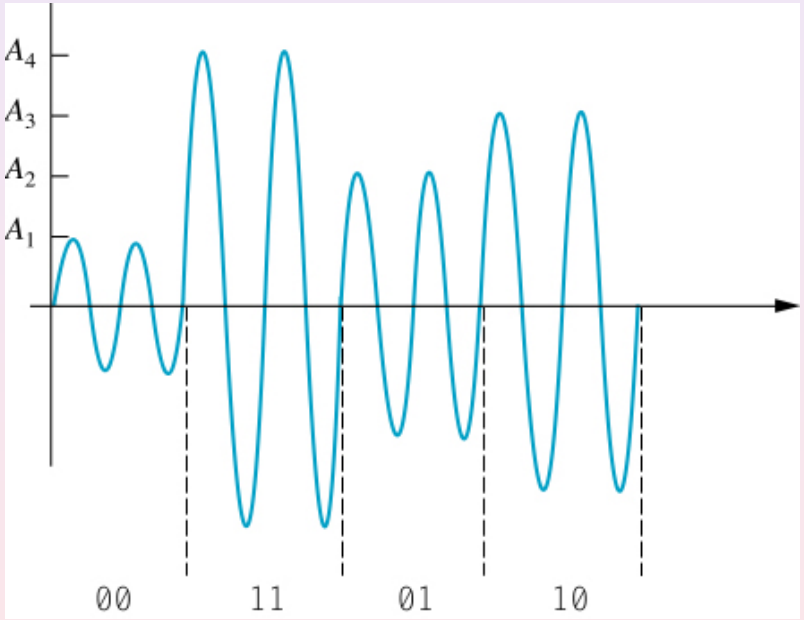


FSK one bit per baud

One can assign each n-bit string one frequency in a set of 2^n frequencies.

Digital to analog conversion 5

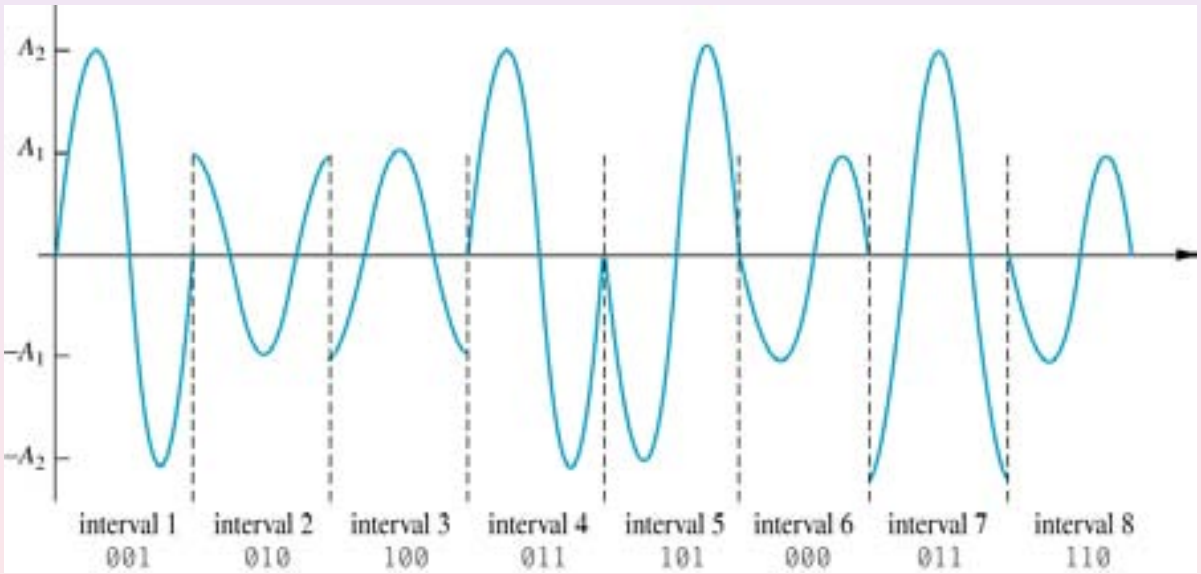
ASK or **amplitude modulation (AM)** assigns a digital 0 to one analog amplitude and a 1 to another. Again one can work with bit-strings instead of bits.



ASK with two bits per baud

Digital to analog conversion 6

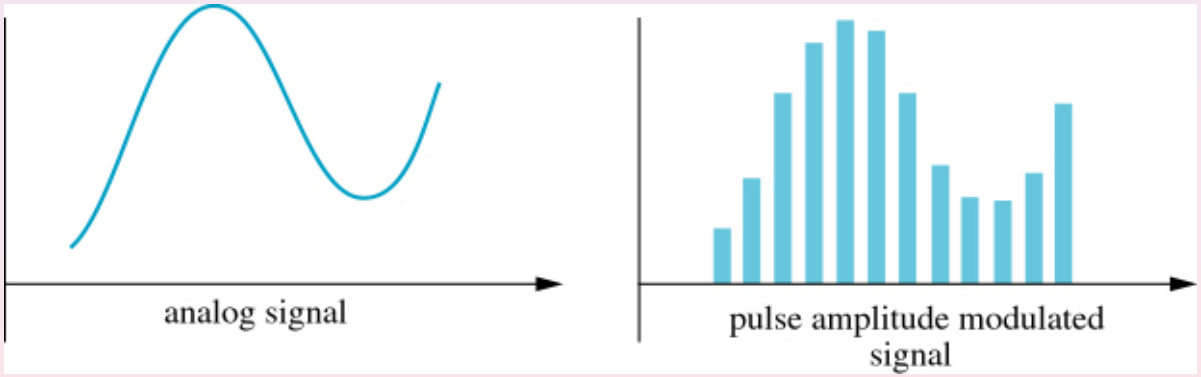
PSK or **phase modulation (PM)** assigns each bit-string of a fixed length to one analog phase shift. **Quadrature amplitude modulation (QAM)** is a combination of amplitude and phase shift.



QAM with two amplitudes and four phases, three bits per baud

Analog to digital conversion 1

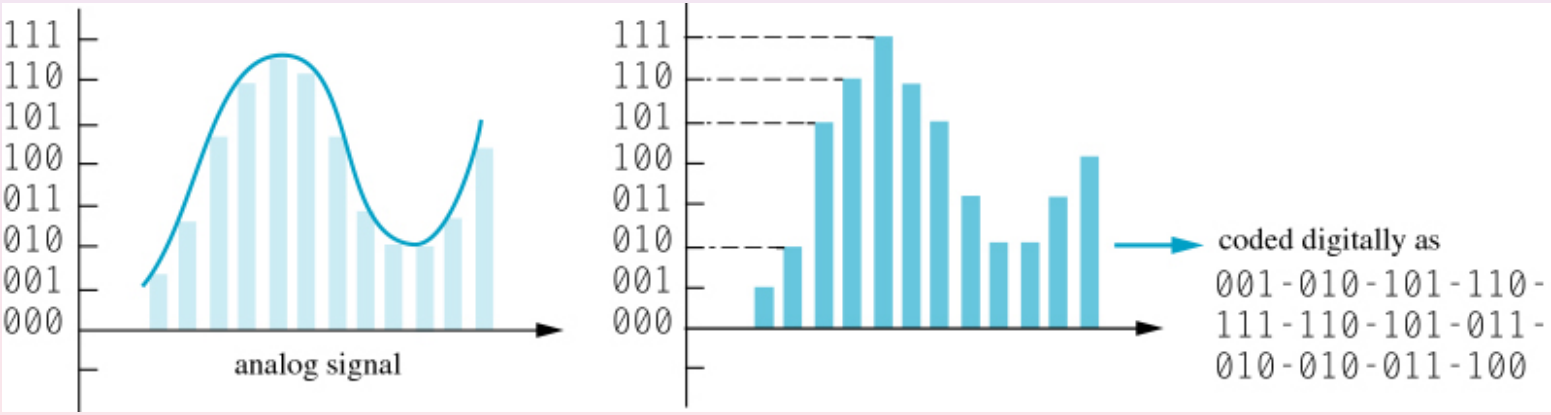
In principle one can reverse any method presented before to obtain an analog to digital conversion.
In **pulse amplitude modulation (PAM)** an analog signal is sampled at regular intervals and then a pulse with amplitude equal to that of the sampled signal is generated.



PAM

Analog to digital conversion 2

PAM-generated signals look digital, but they have analog characteristics (because a pulse may have any amplitude). The **pulse code modulation (PCM)** uses amplitudes assigned from a pre-defined set to the sampled signals. This process is called **quantisation**.



PCM

Why compression? 1

Nothing digital exists without compression. Compression has been achieved only using sophisticated mathematics and algorithms. Here is an example. A TV screen displays about 30 pictures (frames) per second. Each picture consists of approximately 200,000 **pixels**, each with different intensities of the primary colours of blues, green, and red. Other colours can be obtained by combinations of the primary colours.
If 8 bits represent each of the primary colours of one pixel, then a picture actually needs $200,000 \times 24 = 4,800,000$ bits. A two-hour movie has about 216,000 different pictures, i.e.

$$216,000 \times 4,800,000 = 1.0368 \times 10^{12} \text{ bits.}$$

Do they all fit on a DVD? **No**.

Why compression? 2

There are basically four different DVD sizes:

- DVD-5 (the most common, called 4.7 GB Media) holds around 4 ,700, 000, 000 bytes and that is $37,600,000,000 = 3.76 \times 10^{10}$ bits,
- DVD-9, holds around $8,540,000,000 \times 8 = 6.832 \times 10^{10}$ bits,
- DVD-10 holds around $9,400,000,000 \times 8 = 7.52 \times 10^{10}$ bits,
- DVD-18, holds around $17,080,000,000 \times 8 = 1.3664 \times 10^{11}$ bits,

all smaller in size than 1.0368×10^{12} bits. So, how can one beat these physical limits?
Answer: using **data compression** to reduce the size, but retain the meaning.

How do you reduce bits and still keep enough information?

Here is a simple example. Assume that you wish to email a large file consisting entirely of strings of capital letters. If the file has n characters each stored as an 8-bit ASCII code, then we need $8n$ bits.

However, we don't need all ASCII codes to code strings of capital letters: they use only 26 characters. We can make our own code with only 5-bit codewords ($2^5 = 32 > 26$), code the file using this coding scheme, send the encoded file via email, and finally decode it at the other end.

Big deal? The size of the file has decreased by $8n - 5n = 3n$, i.e. a 37.5% reduction.

Huffmann code 1

ASCII code is an 8-bit code which gives no preference in coding characters. However, some characters appear more frequently than others. A **frequency-dependent code** varies the lengths of codewords on frequency: more frequent characters have shorter codewords. An example of frequency-dependent code is the **Huffmann code code**.

Huffmann code 2

To illustrate assume that we have five characters, A-E, whose frequencies are as follows:

Letter	Frequency (%)
A	25
B	15
C	10
D	20
E	30

Huffmann code 3

To construct the Huffmann codeword for the string we follow the following algorithm:

1. To each character we associate a binary tree consisting of just one node. To each tree we assign the tree's *weight*. Initially, the weight of nodes is exactly its frequency: 0.25 for A, 0.15 for B, etc.
2. Calculate the two lightest-weight trees (choose any if there are more than two). Merge the two chosen trees into a single tree with a new root node whose left and right sub-trees are the two we chose. The weight of the new tree is the sum of the weights of the merged trees.
3. Repeat the procedure till one tree is left.

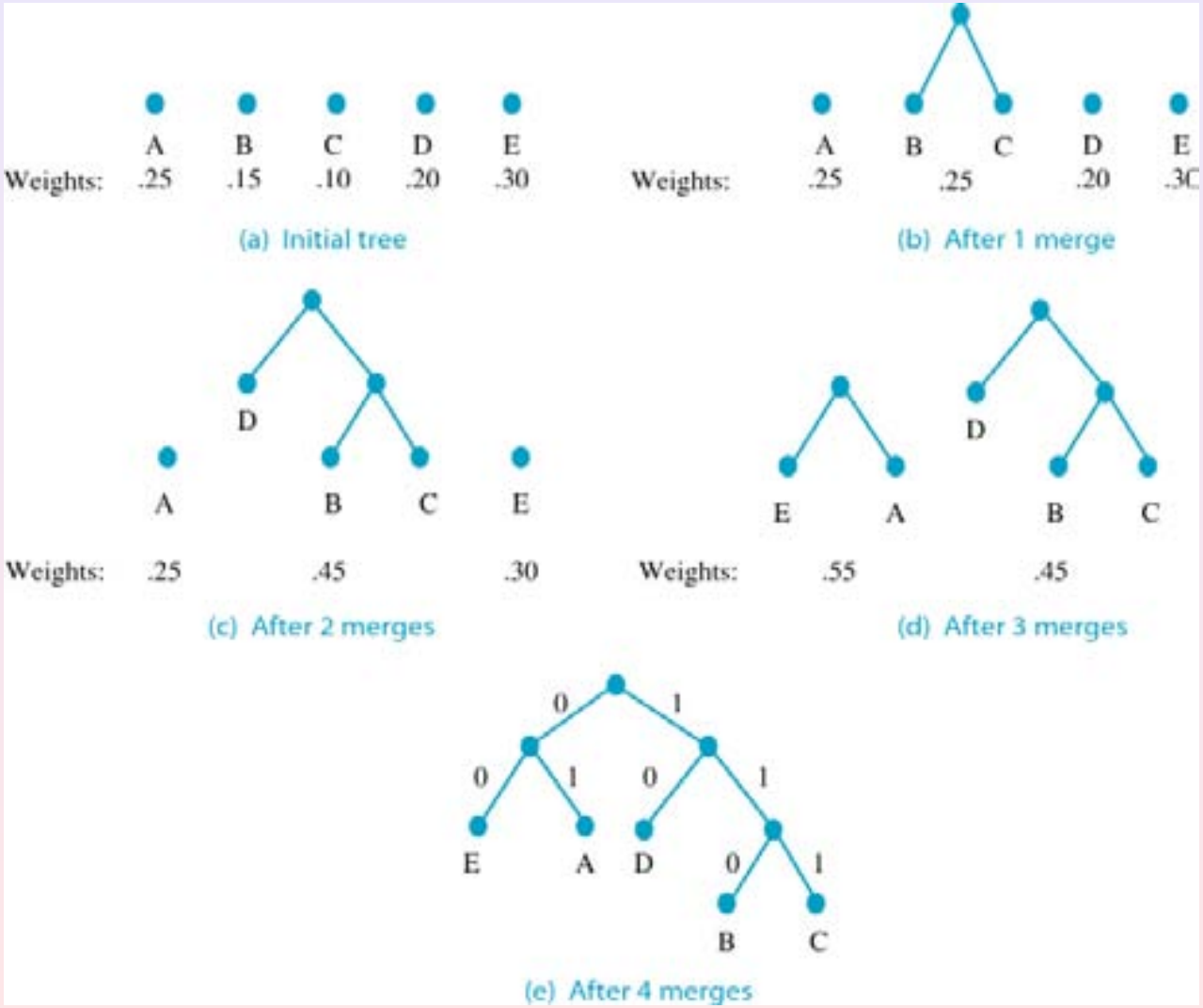
Huffmann code 4

When completed, each of the original nodes is a leaf in the final tree. Arcs are labelled with 0 and 1 as follows: we assign a 0 each time a left child pointer is followed and a 1 for each right child pointer.

As with any binary tree, there is a unique path from the root to any leaf. The path to any leaf defines the Huffmann code of the character on labelling the leaf.

Let us apply this procedure to the character strings A,B,C,D,E.

Huffmann code 5



Huffmann code 6

The Huffmann code is the following:

Letter	Frequency (%)	Code
A	25	01
B	15	110
C	10	111
D	20	10
E	30	00

Huffmann code 7

As one can see, the algorithm for constructing the Huffmann tree is very "tolerant", it gives a lot of flexibility. In the above example one could, for instance, choose putting D on the right-hand side of the merged tree in stage (c) and swap A and E in stage (d). The result will be the Huffmann code:

Letter	Frequency (%)	Code
A	25	00
B	15	100
C	10	101
D	20	11
E	30	01

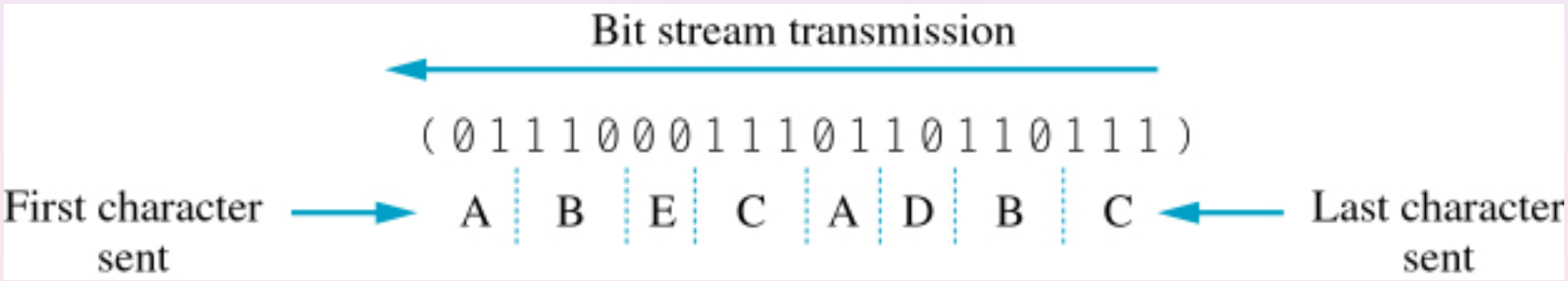
Huffmann code 8

Huffmann code properties:

- There are many Huffmann codes which can be associated to the same characters and weights. So, we should better speak of *a* Huffmann code instead of *the* Huffmann code.
- All Huffmann codes are, in general, variable-length, frequency-dependent, prefix codes. Consequently, Huffmann codes are uniquely decodable.

Huffmann code 9

Let us decode the codeword 011100011101101101111:



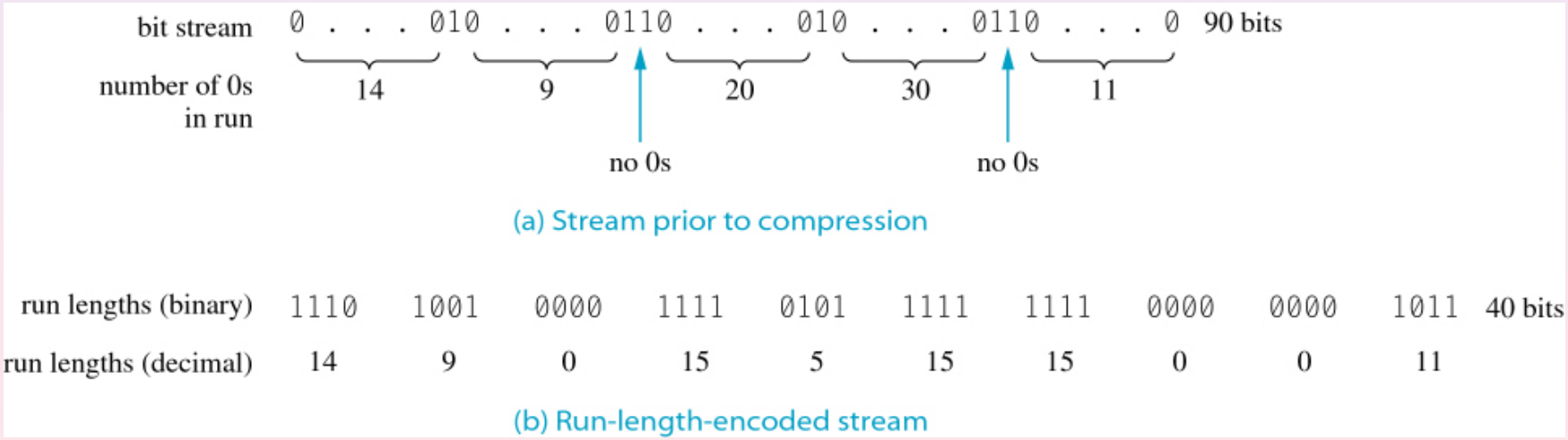
Run-length encoding 1

Huffmann codes are useful only in case we know the frequency of characters. Many items that travel the communications media, including binary files, fax data, and video signals, do not fall into this category.

Run-length encoding analyses bit-strings by looking for long runs of 0 or 1. Instead of sending all bits, *it sends only how many of them are in the run*. Fax transmission is well-served by this technique as potentially a large part of a page is white space, corresponding to a long run of 0s.

Run-length encoding 2

The sender transmits only the length of each run as a fixed-length integer; the receiver gets each length and generates the proper number of bits in the run, inserting the other bit in between. Here is an example:



This technique works well when there are many long 0 runs.

Run-length encoding 3

What about long runs of different bits or even characters?

Solution: send the actual character along with the run length. For example,

HHHHHHHUFFFFFFFFFFFFFFFYYYYYYY
YYYYYYYYYYYYYYYDGGGGGGGGGGG

can be sent as 7,H,1,U,14,F,20,Y,1,D,11,G.

Relative encoding 1

A single video image may contain little repetition, but *there is a lot of repetition over several images*.

Reason: a) a USA TV signal sends 30 pictures per second, b) each picture generally varies only slightly from the previous one.

The **relative encoding** or **differential encoding** works as follows:

- the first picture is sent and stored in a receiver's buffer;
- the second picture is compared with the first one and the encoding of differences are sent in a frame format; the receiver gets the frame and applies the differences to create the second picture;
- the second picture is stored in a receiver's buffer and the process continues.

Relative encoding 2

Here is an example of relative encoding:

First frame	Second frame	Third frame
5 7 6 2 8 6 6 3 5 6	5 7 6 2 8 6 6 3 5 6	5 7 6 2 8 6 6 3 5 6
6 5 7 5 5 6 3 2 4 7	6 5 7 6 5 6 3 2 3 7	6 5 8 6 5 6 3 3 3 7
8 4 6 8 5 6 4 8 8 5	8 4 6 8 5 6 4 8 8 5	8 4 6 8 5 6 4 8 8 5
5 1 2 9 8 6 5 5 6 6	5 1 3 9 8 6 5 5 7 6	5 1 3 9 7 6 5 5 8 6
5 5 2 9 9 6 8 9 5 1	5 5 2 9 9 6 8 9 5 1	5 5 2 9 9 6 8 9 5 1
	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0
	0 0 0 1 0 0 0 0 −1 0	0 0 1 0 0 0 0 1 0 0
	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0
	0 0 1 0 0 0 0 0 1 0	0 0 0 0 −1 0 0 0 1 0
	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0
	Transmitted frame contains the encoded differences between the first and second frames.	Transmitted frame contains the encoded differences between the second and third frames.

Lempel-Ziv compression

The **Lempel-Ziv compression** method looks for often-repeated strings and store them just once. It then replaces multiple occurrences with a code corresponding to the original.

This technique—store a piece of information in one place and reference it through special codes—is widely used:

1. UNIX compress command,
2. gzip on UNIX,
3. V.42bis compression standard for modems,
4. GIF (Graphics Interchange Format).

JPEG 1

Pixels are represented using 8 bits which can distinguish 256 shades of gray, ranging from white to black.

For colour pictures we can use an 8-bit group to represent each of the three primary colours. The intensity of each colour can be adjusted according to the 8-bit value to produce the desired colour, more precisely, with $3 \times 8 = 24$ bits we can describe $2^{24} = 16,777,216$ colours.

An alternative method uses three 8-bit groups as follows: a) one group for *luminance* (brightness), b) two groups for *chrominance* (split into Blue and Red components).

JPEG 2

We can express Y, I, Q (luminance and colours)—National Television Standard Committee (NTSC)—in terms of the primary colours R, G, B, as follows:

$$\begin{aligned} Y &= 0.30R + 0.59G + 0.11B, \\ I &= 0.60R - 0.28G - 0.32B, \\ Q &= 0.21R - 0.52G + 0.31B. \end{aligned}$$

The human eye can see more detail in the Y component than in I and Q.
This is exploited by JPEG compression.

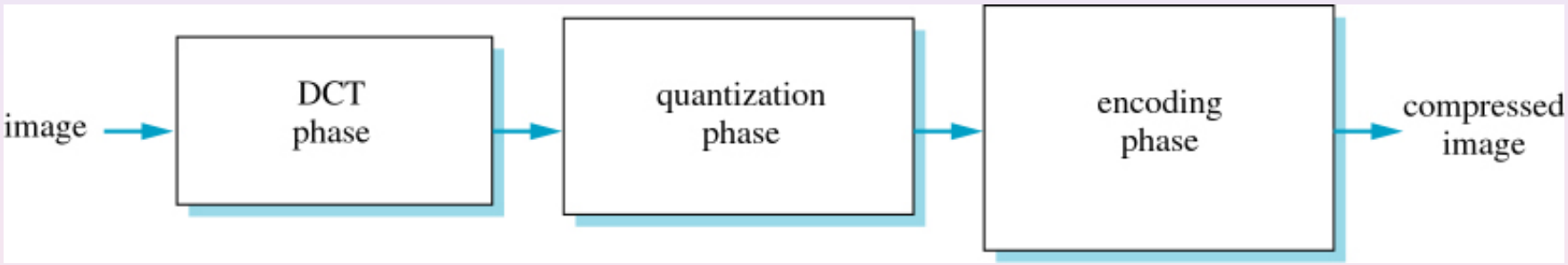
JPEG 3

JPEG is an acronym for the Joint Photographic Experts Group and **JPEG compression** is a **lossy** data compression method. This means that the image obtained after decompression may not coincide with the original image. Lossy compression is acceptable for images because of the inherent limitations of the human optical system.

There are three phases in a JPEG compression:

1. the discrete cosine transform (DCT),
2. quantisation,
3. encoding phase.

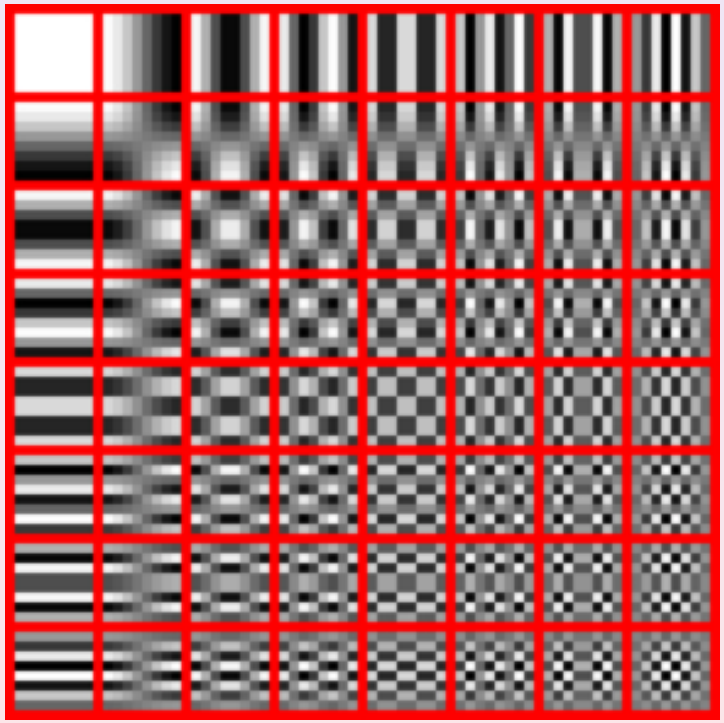
JPEG 4



JPEG three phases

JPEG 5

In the DCT phase each component (Y, I, Q) of the image is "tiled" into sections of 8×8 pixels each; dummy data fills incomplete blocks.

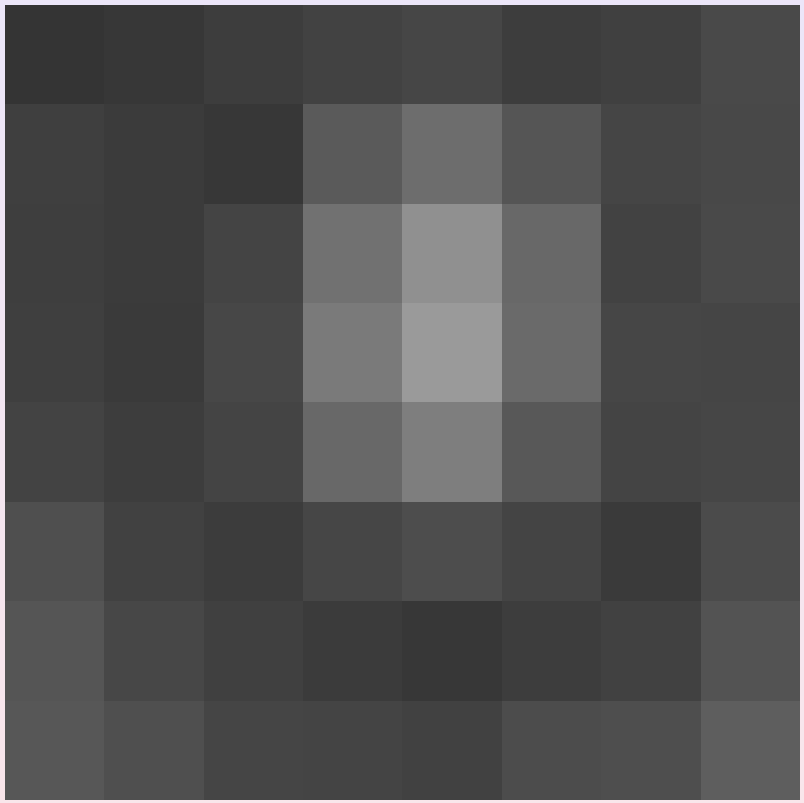


picture source: http://en.wikipedia.org/wiki/JPEG

JPEG 6

Then, each tile is converted to frequency space using a two-dimensional forward “discrete cosine transform”.

JPEG 7



picture source: http://en.wikipedia.org/wiki/JPEG

The 8×8 sub-image shown in 8-bit greyscale

JPEG 8

The human eye can see small differences in brightness over a relatively large area, but cannot distinguishing the exact strength of a high frequency brightness variation.

This fact facilitates the reduction of information in the high frequency components by simply dividing each component in the frequency domain by a constant, and then rounding to the nearest integer. This is the **quantisation phase**, in which the main lossy operation takes place.

As a result, in the **encoding** phase, many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers; they take many fewer bits to store. JPEG produces in typical applications compression ratios of 20:1 and better.

GIF

GIF (Graphics Interchange Format) compresses by: a) reducing the number of colours to 256, and b) trying to cover the range of colours in an image as closely as possible.

It replaces each 24-bit pixel value with an 8-bit index to a table entry containing the colour that matches the original "most closely". In the end, a variation of the Lempel-Ziv encoding is applied to the resulting bit values.

GIF files are lossy if the number of colours exceeds 256 and lossless otherwise.

MPEG and MP3 1

The **Moving Picture Experts Group (MPEG)** is a working group of ISO/IEC charged with the development of video and audio encoding standards.
MPEG uses methods similar to JPEG compression but also takes advantage of redundancy between successive frames to use "inter-frame compression": calculating differences between successive frames one can use motion prediction techniques to improve compression.

MPEG and MP3 2

MPEG has standardised the following compression formats and ancillary standards:

- MPEG-1: video and audio compression standard; it includes the popular Layer 3 (MP3) audio compression format
- MPEG-2: transport, video and audio standards for broadcast-quality television
- MPEG-4: expands MPEG-1 to support video/audio "objects", 3D content, low bit-rate encoding and support for Digital Rights Management

MPEG and MP3 3

MPEG-1 Audio Layer 3, better known as **MP3**, is a popular digital audio encoding, lossy compression format, and algorithm.

MP3 is based on the **psycho-acoustic model**, **auditory mask,** and **filter bank**.

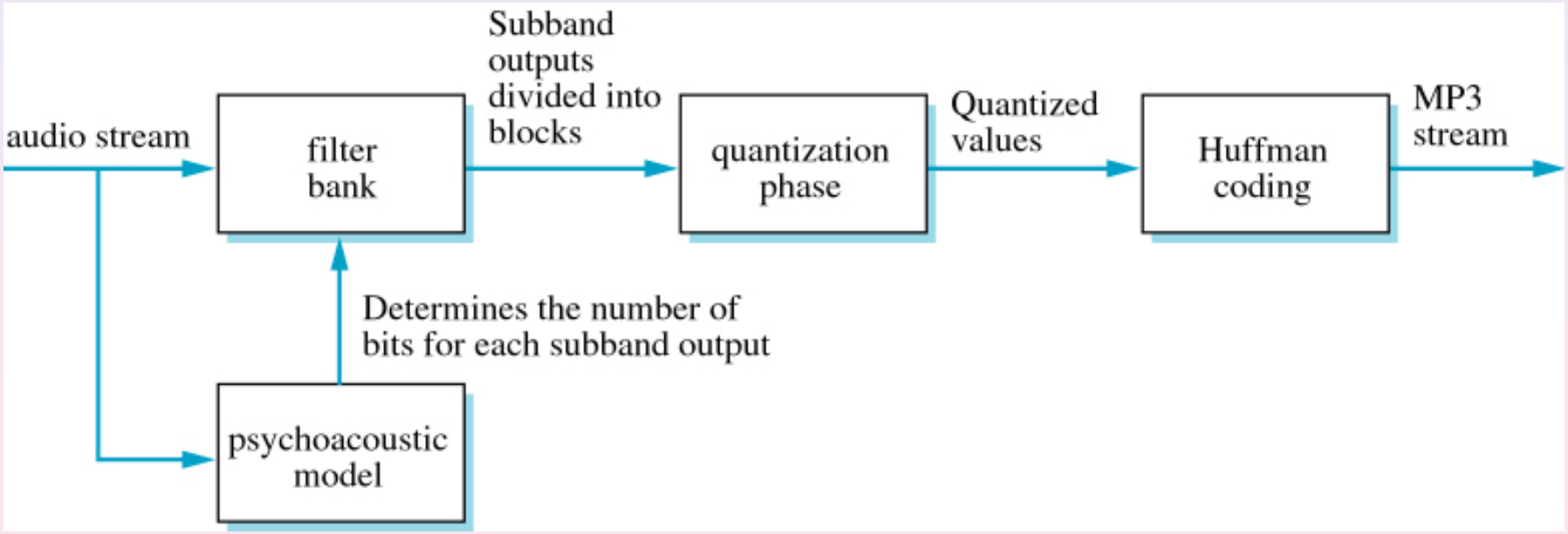
Psycho-acoustics is the study of the human auditory system to learn what we can hear and what sounds we can distinguish. In general we can hear sounds in the 20 Hz to 20 kHz range, but sounds with close frequencies (for example, 2000 Hz and 2001 Hz) cannot be distinguished.

MPEG and MP3 4

The auditory mask is the following phenomenon: if a sound with a certain frequency is strong, then we may be unable to hear a weaker sound with a similar frequency.

The filter bank is a collection of filters, each of which creates a stream representing signal components of a specified ranges. There is one filter for each of the many frequency ranges. Together they decompose the signal into **sub-bands**.

MPEG and MP3 5



MP3 encoding

Movie Piracy 1

picture source: http://www.mpaa.org/piracy.asp

Piracy

Movie Piracy 2

Copy protection technologies include:

- DVDs which use the Content Scrambling System (CSS),
- Pay-Per-View Copy Protection,
- dedicated DSL set-top boxes,
- digital encryption encoding of satellite signals and videocassettes (which contain Macrovision).

Movie Piracy 3

CSS (1996) uses a 40-bit encryption. 2^{40} is large number in human terms but unsafe: a brute force attack, testing a million keys per second, may break it in less than two weeks.

A DVD or digital cable/satellite boxes receiving a data stream encoded with Macrovision will cause most VCR's set to record it to fail. This is obtained by using a signal implanted within the offscreen range (vertical blanking interval) of the video signal. Macrovision technique may interfere with other electronic equipment.

Why check integrity?

Data can be corrupted during transmission—many factors can alter or even wipe out parts of data.

Reliable systems must have mechanisms for detecting and correcting **errors**.

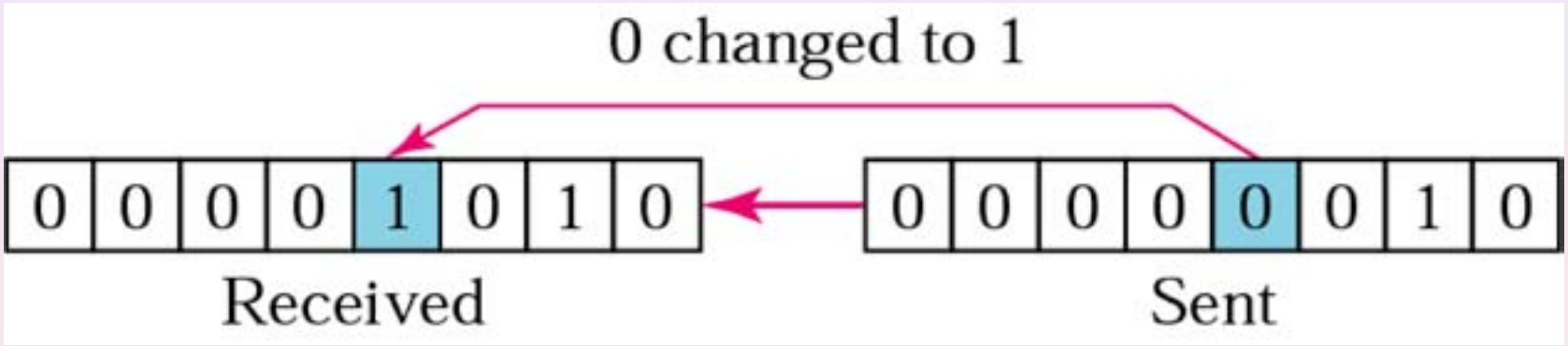
The capability to detect when a transmission has been changed is called **error detection**. In some cases a message with errors is discarded and sent again, but not always this is possible (e.g. in real-time viewing). In the later case the error has to be fixed (in real-time) and the mechanism doing this job is called **error correction**.

Types of errors

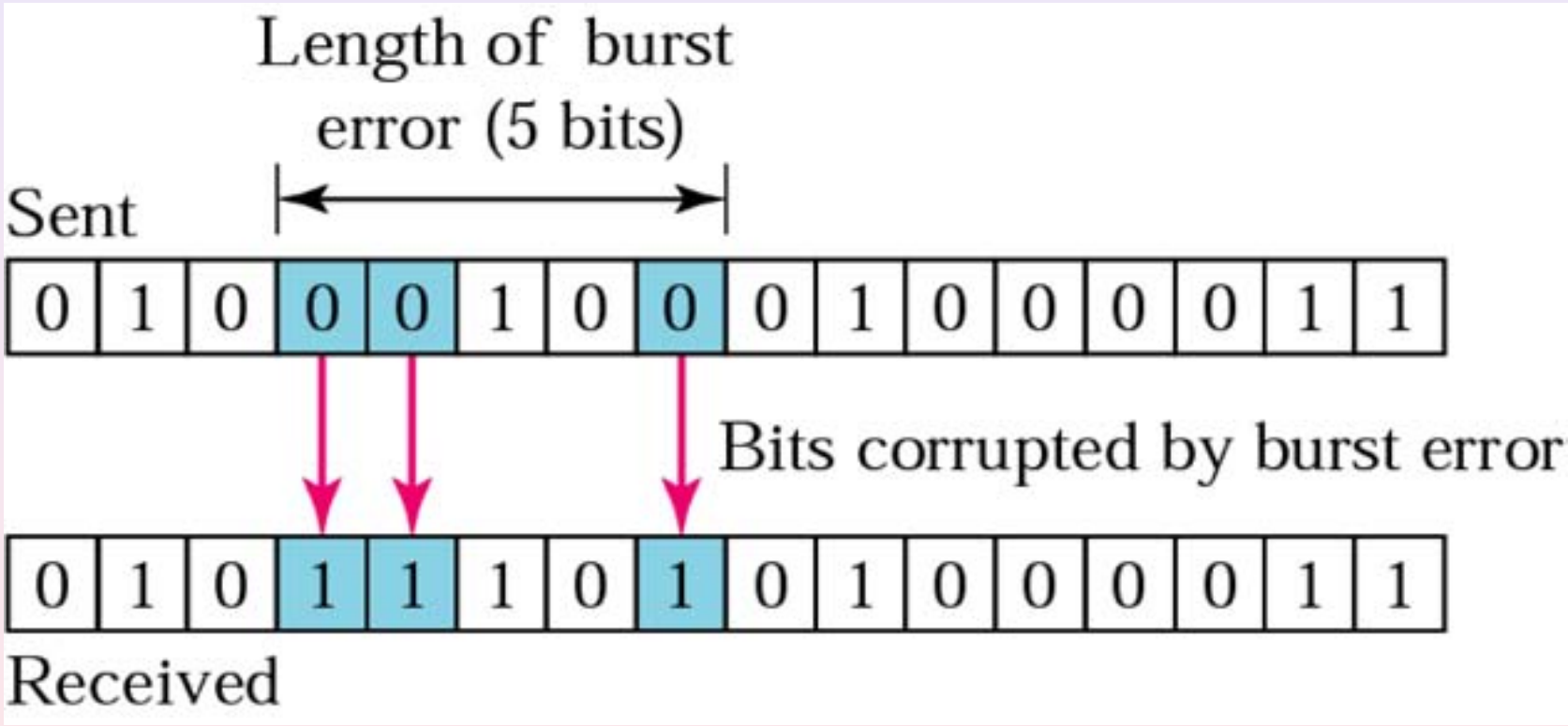
There are two types of errors:

1. single-bit error, when only one bit in the data has changed,
2. burst error, when two or more bits in the data have changed.

Single-error bit



Burst error

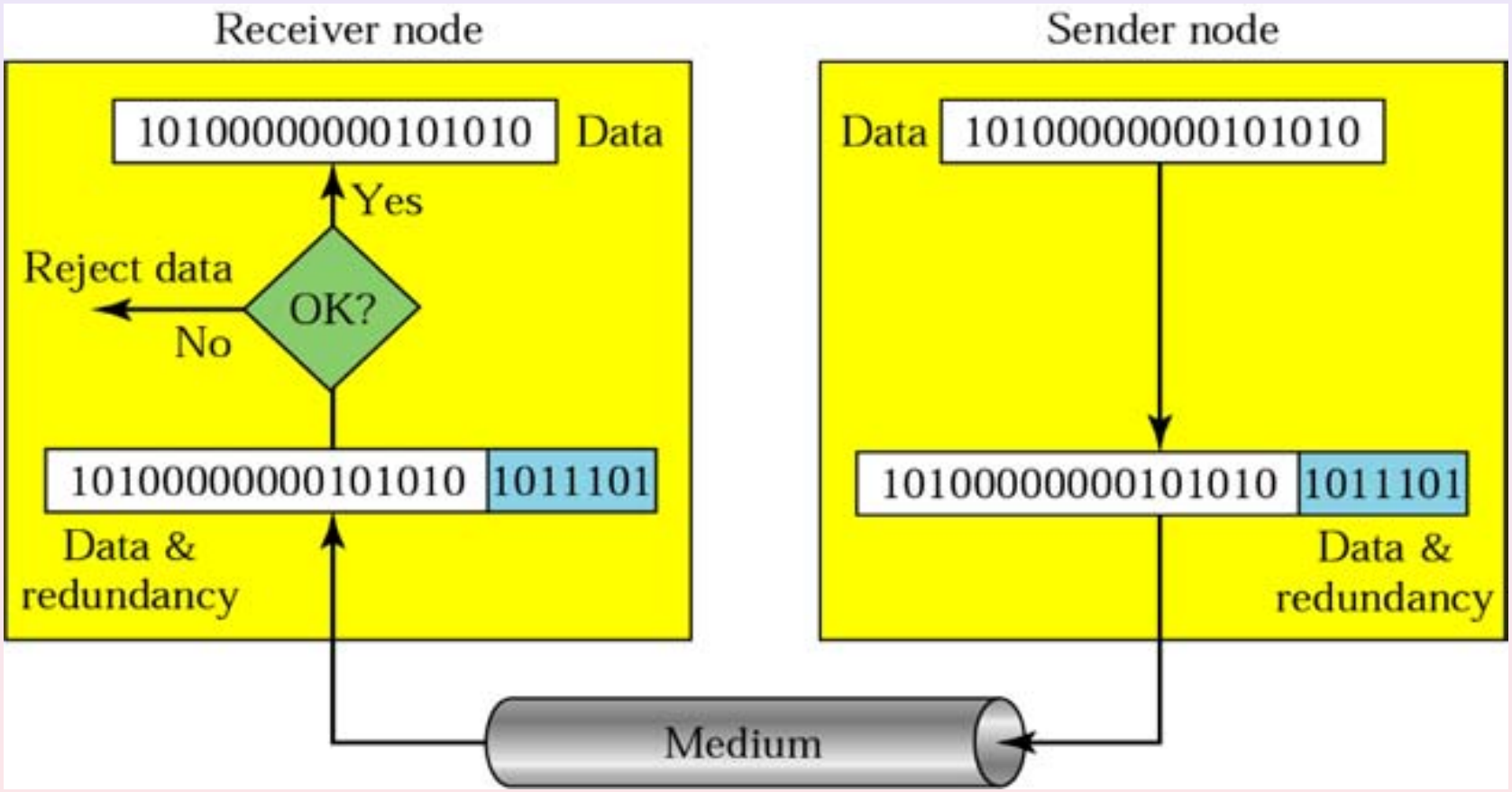

Length of burst error (5 bits)
Sent
0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
Bits corrupted by burst error
0 1 0 1 1 1 0 1 0 1 0 0 0 0 1 1
Received

Redundancy 1

One error detection method is to send every data unit twice. Comparison between the two versions identifies all errors. The system is accurate, but slow (transmission time doubles and comparison time is added).

A better approach is to add a fixed extra information to each unit—this technique is called **redundancy**. The extra information has no meaning and finally is discarded (natural languages are redundant).

Redundancy 2



Redundancy 3

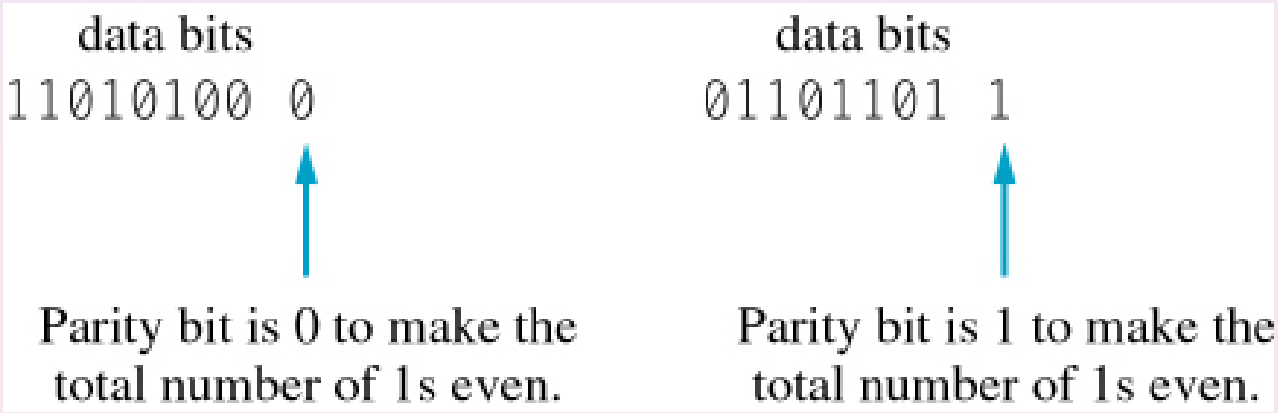
There are tree main types of redundancy checks:

1. parity check,
2. checksums,
3. cyclic redundancy check (CRC).

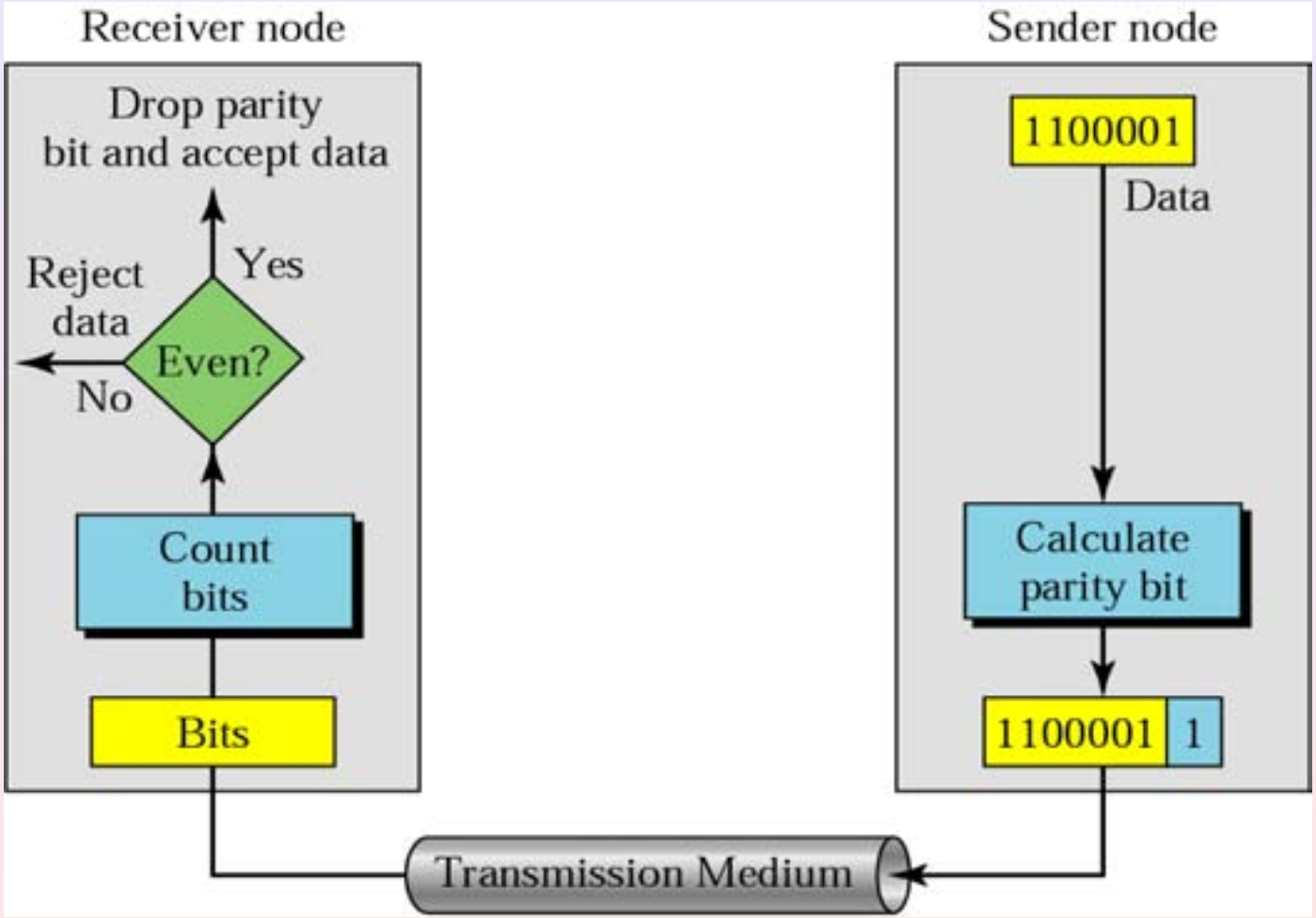
Low-density parity-check (LDPC) codes were the first to allow data transmission rates close to the theoretical maximum, the Shannon Limit.

Parity check: one-dimensional 1

A redundant bit, called **parity bit**, is added to every data unit so that the total number of 1s in the unit (including the parity bit) becomes even (or odd).



Parity check: one-dimensional 2



Parity check: one-dimensional 3

Simple parity check can detect all single-bit errors.

It can detect burst errors only if the total number of errors in each data is odd (even).

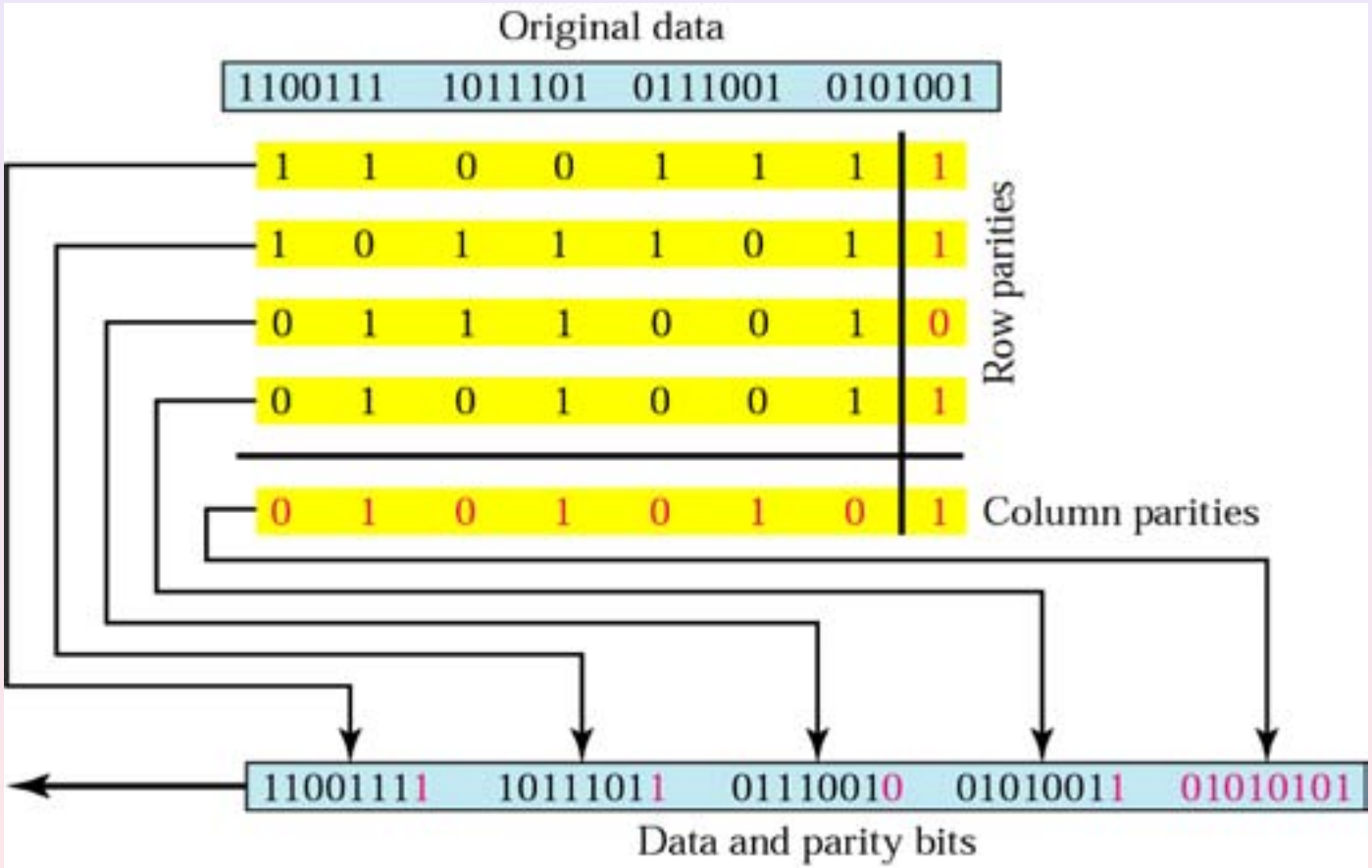
Parity check: two-dimensional 1

In a **two-dimensional parity check** a block of bits is organised in a table and parity is checked on both dimensions.

- First one calculates the parity bit for each data unit.
- Second one calculates the parity bit for each column and one creates a new row of 8 bits—the parity bits for the whole block.

Two-dimensional parity check increases the likelihood of burst errors detection.

Parity check: two-dimensional 2



Checksums 1

The method divides all data bits into 32-bit groups and treats each as an integer value. The sum of all these values gives the **checksum**. Any overflow that requires more than 32 bits is ignored.

An extra 32 bits representing in binary the checksum is then appended to the data before transmission.

The receiver divides the data bits into 32-bit groups and performs the same calculation. If the result does not coincide with the valued stored in the checksum and error has occurred.

Checksums 2

Checksum is better than simple parity checks, but it may not detect all errors. For example, if the error's effect is to *simultaneously* increase by a fixed number one of the 32-bit values and decrease by the same number a different 32-bit value.

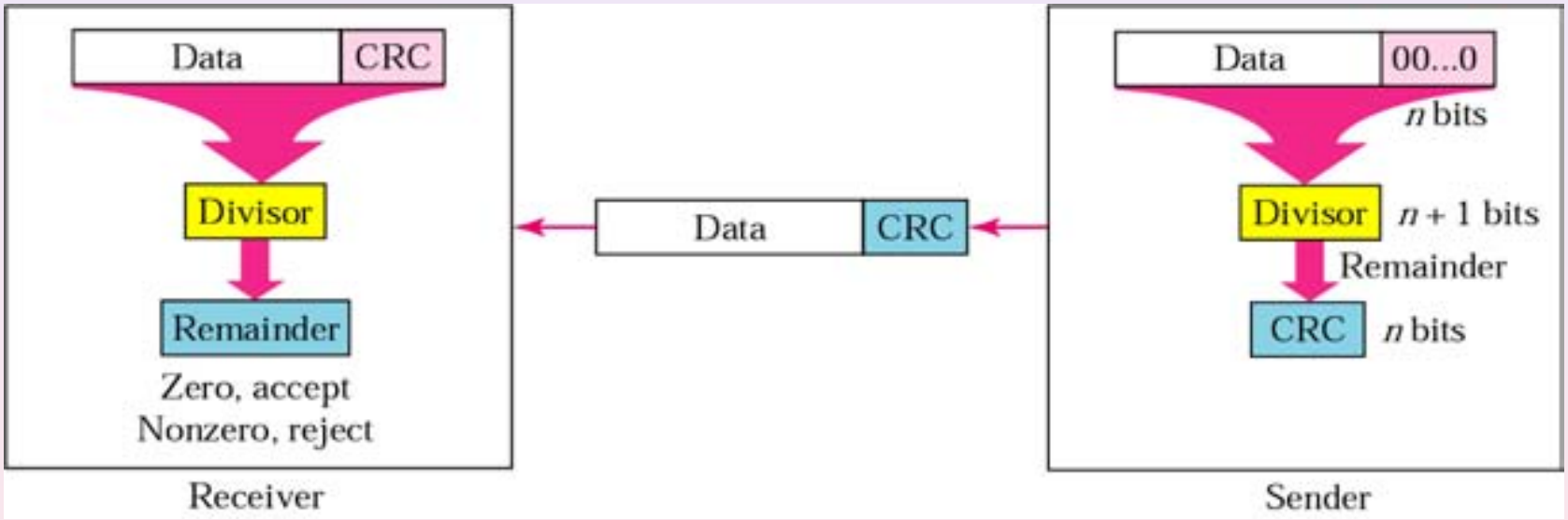
Cyclic redundancy check 1

The most powerful (and elaborate) redundancy checking technique is the **cyclic redundancy check (CRC)**.

CRC is based on binary division: a string of redundant bits—called the **CRC** or the **CRC remainder**—is appended to the end of the data unit such that the resulting data unit is exactly divisible by a second, predetermined binary number.

At destination the incoming data is divided by the same number; if there is no remainder, the unit of data is accepted, otherwise rejected.

Cyclic redundancy check 2



Error correction

Error-correcting codes include enough redundant info with the unit block for the receiver to be able to deduce what bits are wrong and reconstruct original.

These codes are used when the error rate is high (e.g. wireless). Detecting an error and than re-transmitting the message is not good because retransmission may have errors too.

If we allow *any* number of errors in data bits *and* in check bits, then no error-detection (or correction) method can guarantee to work, since any valid pattern can be transformed into any other valid pattern.

Forward error correction 1

We examine the simplest case, i.e. single-bit errors. A single additional bit can detect single-bit errors. Is it enough for correction?

To correct a single-bit error in an ASCII character we must determine which of the 7 bits has changed. There are **eight** possible situations: no error, error on the first bit, error on the second bit, . . . , error on the seventh bit. Apparently we need **three** bits to code the above eight cases. But what if an error occurs in the redundancy bits themselves?

Forward error correction 2

Clearly, the number of redundancy (or, error control) bits required to correct n bits of data **cannot be constant, it depends on** n.

To calculate the number of redundancy bits r required to correct a n bits of data we note that:

- with n bits of data and r bits of redundancy we get a code of length $n + r$,
- r must be able to handle at least $n + r + 1$ different states, one for no error, $n + r$ for each possible position,
- therefore $2^r \geq n + r + 1$.

Forward error correction 3

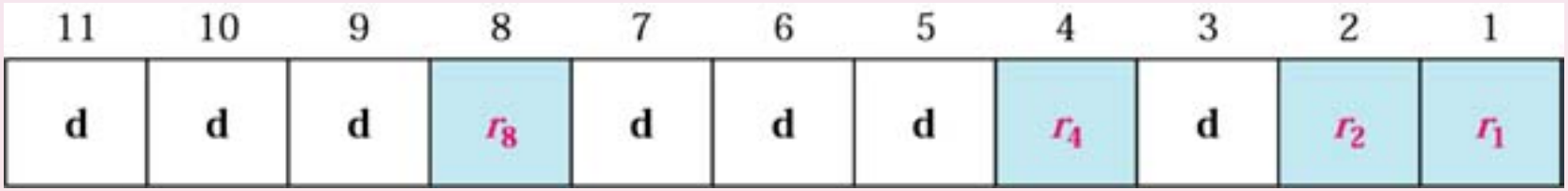
In fact we can choose r to be the smallest integer such that $2^r \geq n + r + 1$. Here are some examples:

Number of data bits (n)	Minimum number of redundancy bits (r)	Total bits ($n + r$)
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

The Hamming code 1

The **Hamming code** is a practical solution which detects and corrects all single-bit errors in data units of any length.

For a 7-bit ASCII code we need 4 redundancy bits which can be added in different positions of the original bits (typically, on power of 2 positions, 1,2,4,8). Data bits are marked with **d** and parity bits are denoted by r_8, r_4, r_2, r_1.



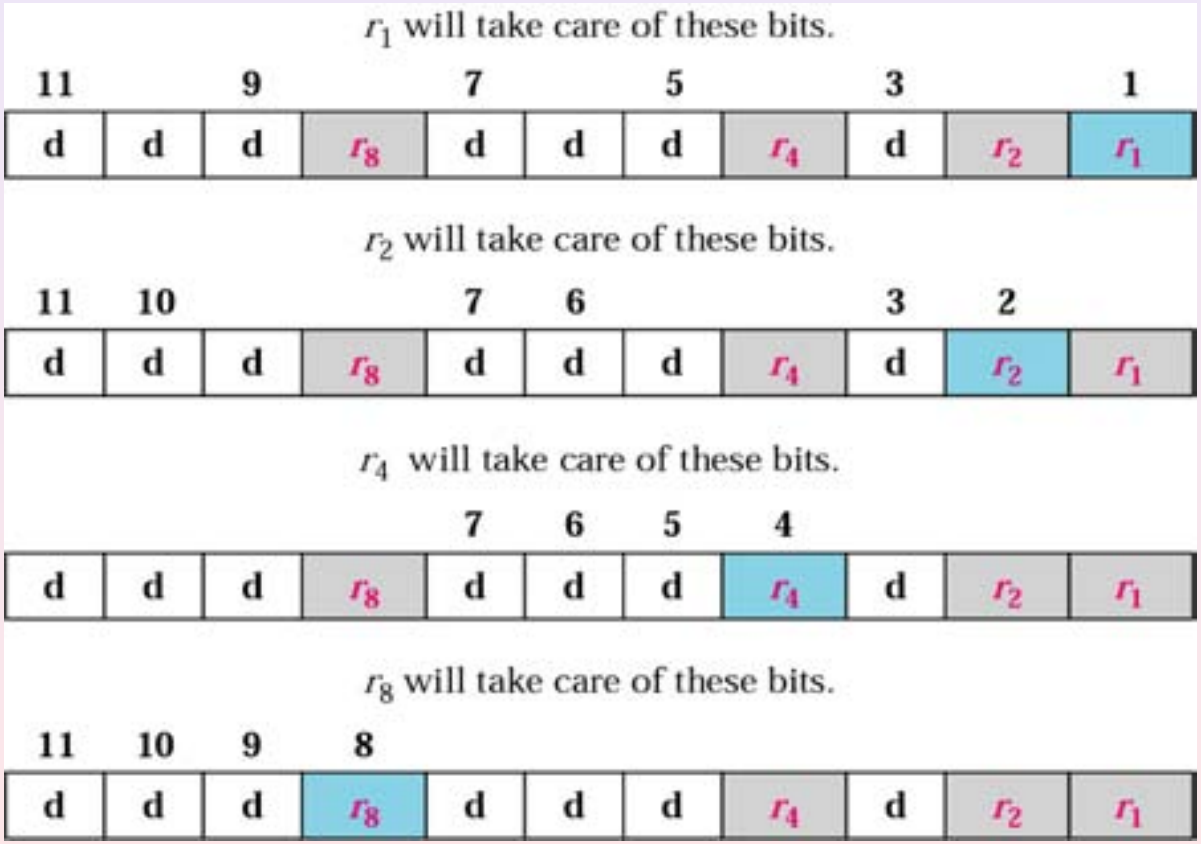
The Hamming code 2

In the Hamming code each r bit is the parity bit for one combination of data bits as follows:

- r_1: position 1, check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc., bits position 2, 1,3,5,7,9,11, 13, 15, ...
- r_2: position 2, check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc., bits 2,3,6,7,10,11,14,15, ...
- r_4: position 4, check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc., bits 4,5,6,7,12,13,14,15,20,21,22,23, ...
- r_8: position 8, check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc., bits 8,9,10,11,12,13,14,15, 24, ...
- r_{16}: position 16, check 16 bits, skip 16 bits, check 16 bits, skip 16 bits, etc., bits 16,17,...,31, 48, 49, ..., 63, 96, ...

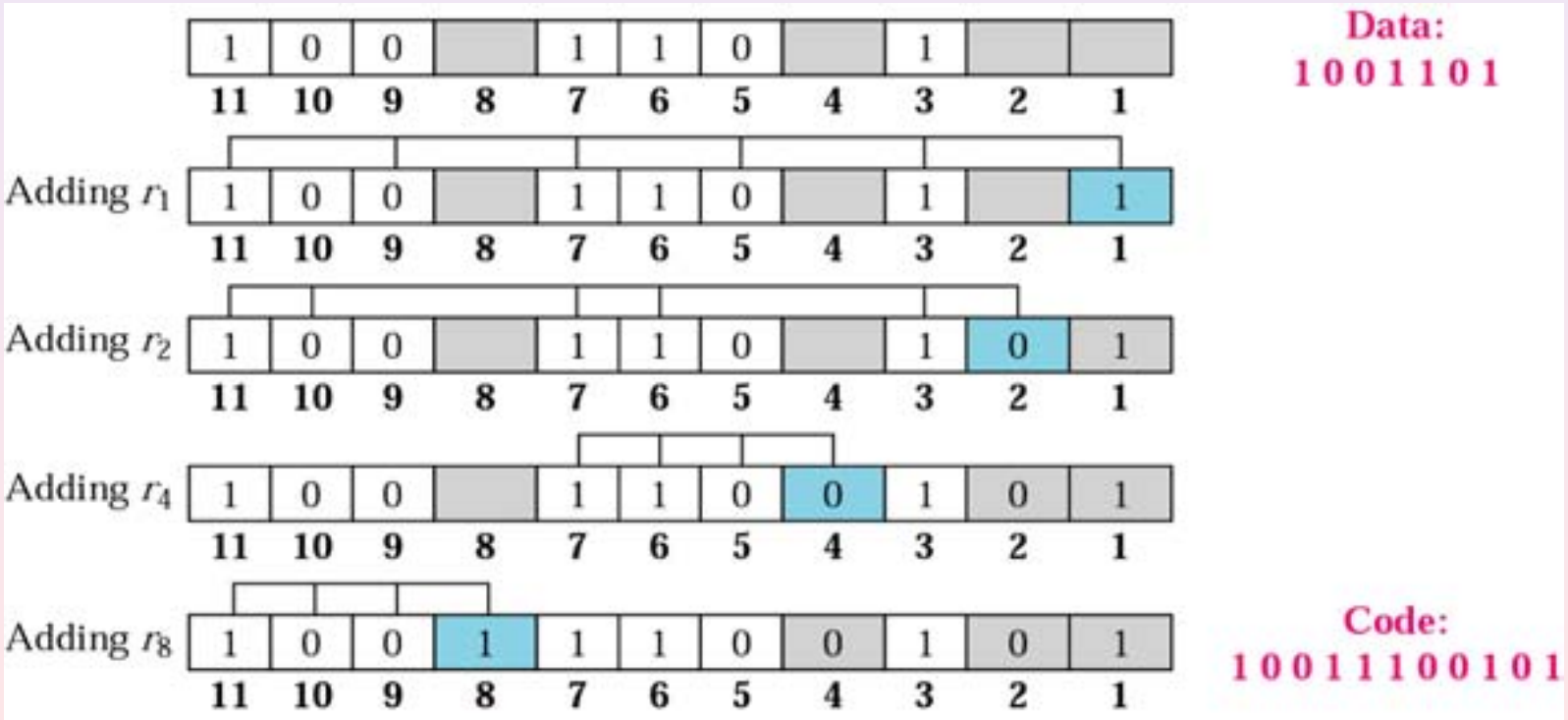
The Hamming code 3

Each data bit may be included in more than one calculation.



The Hamming code 4

This picture describes the calculation of r bits for the ASCII value of 1001101: start with data and repeatedly calculate, one by one, the parity bits r_1, r_2, r_4, r_8:

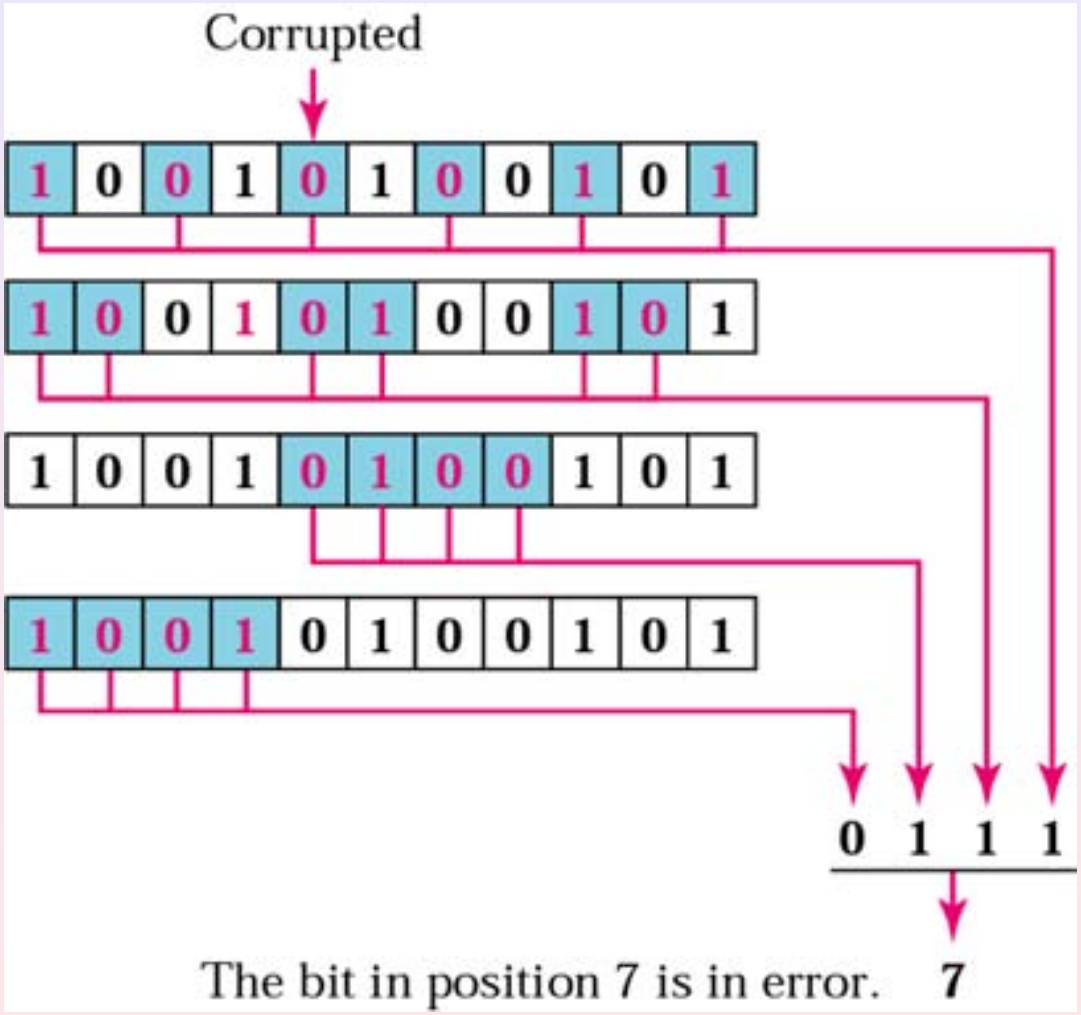


The Hamming code 5

Imagine that instead of the string 10011100101 the string 1001**0**100101 was received (the 7th bit has been changed from 1 to 0).

The receiver recalculates 4 new parity bits using the same sets of bits used by the sender plus the relevant parity r bit for each set. Then it assembles the new parity values into a binary number in the order used by the sender, r_8, r_4, r_2, r_1. This gives the location of the error bit. The sender can now reverse the value of the corrupted bit!

The Hamming code 6

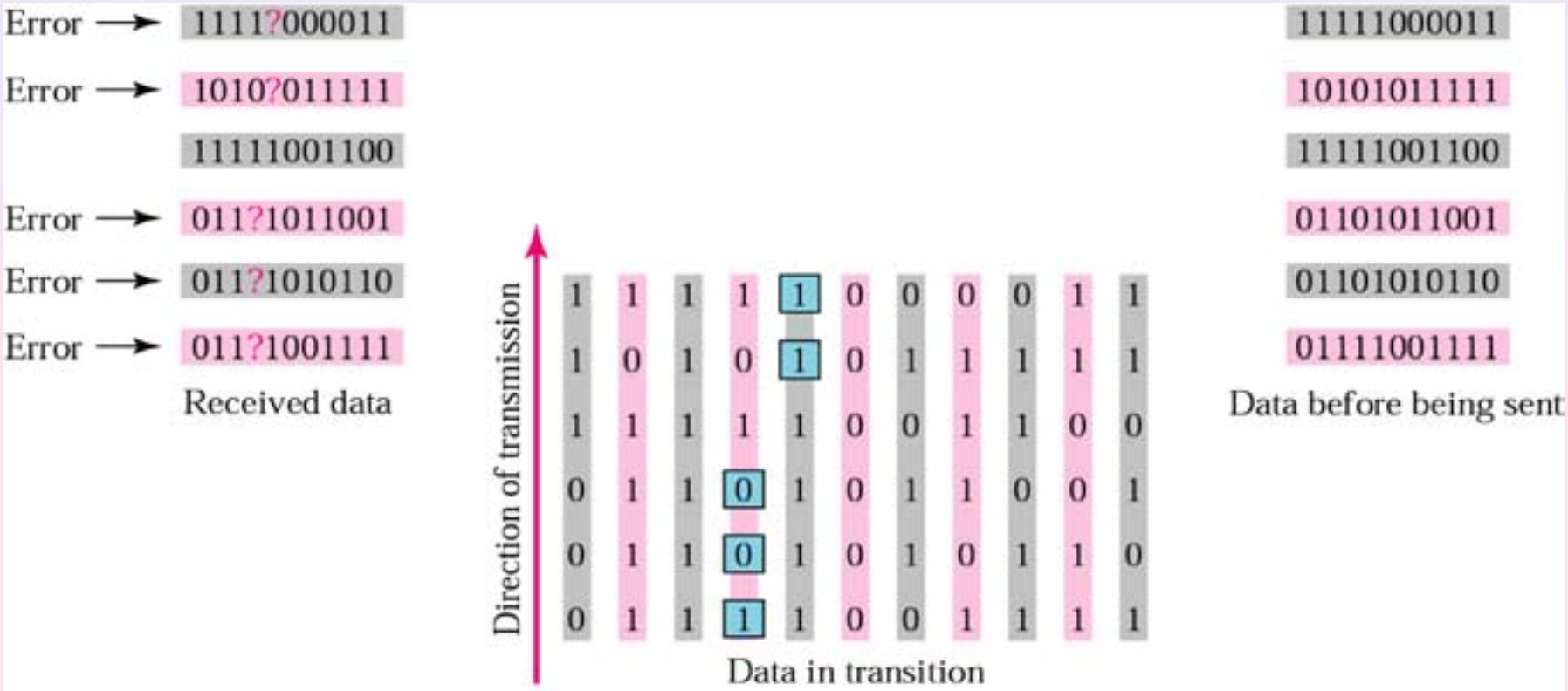


Multiple-bit error correction 1

The Hamming code cannot work for burst errors, but can easily be adapted to this more general case.

Instead of sending all bits in a data unit together, one can organise N units in a column and then send the first bit of each, followed by the second bit of each, aso. If a burst error of $M < N$ bits occurs, then the error does not corrupt M bits of a single unit, it corrupts 1 bit of a unit.

Multiple-bit error correction 2



Multiple-bit error correction 3

In the above figure we exemplify the technique with six data units, each unit being a character with Hamming redundant bits.
The data units have been organised in a table, and transmission is done (in the sense of the arrow) column by column. Corrupted bits are shown in squares.
All five corrupted bits have been detected and corrected at destination.

Multiple-bit error correction 4

There are special codes addressing directly the problem for multiple-bit error correction, in particular, the Reed-Solomon codes. Reed-Solomon coding is a key component of the compact disc. It was the first use of strong error correction coding in a mass-produced consumer product, and DAT and DVD use similar schemes.

Recall from the Hamming code, that to each n-bit data one associates the redundancy (error control) r bits. While the data is completely undetermined, the redundancy is fully determined by the data.

Consequently, out of 2^{n+r} possible strings, only 2^n are valid codewords! For example, if $n = 8$, then $r = 4$, so out of all $2^{12} = 4,096$ 12-bit strings only $2^8 = 256$ are legitimate codewords.

Multiple-bit error correction 5

The **Hamming distance** between two m-bit strings is the number of bits in which the two strings differ.

Given a finite set of codewords one can compute its *minimum distance*, the smallest value of the distance between two codewords in the set.

If d is the minimum distance of a finite set of codewords, then the method can detect any error affecting fewer than d bits (such a change would create an invalid codeword) and correct any error affecting fewer than $d/2$ bits.

Multiple-bit error correction 6

For example, if $d = 10$, and 4 bits of a codeword were damaged, then the string cannot possibly be a valid codeword: at least 10 bits must be changed to create another valid codeword.

If the receiver assumes that any error will affect fewer than 5 bits, then she needs only to find the closest valid codeword to the received damaged one to conclude that it is the correct codeword. Indeed, any other codeword would have had at least 6 bits damaged to resemble the received string.
Selecting the codewords in such a way to maximise the minimum distance is vital.

Error correction on the Internet is performed at multiple levels

- Each Ethernet frame carries a CRC-32 checksum. The receiver discards frames if their checksums do not match.
- The IPv4 header contains a header checksum of the contents of the header (excluding the checksum field). Packets with checksums that don't match are discarded.
- The checksum was omitted from the IPv6 header, because most current link layer protocols have error detection.
- UDP has an optional checksum. Packets with wrong checksums are discarded.
- TCP has a checksum of the payload, TCP header (excluding the checksum field) and source- and destination addresses of the IP header. Packets found to have incorrect checksums are discarded and eventually get retransmitted when the sender receives a triple-ack or a timeout occurs.

What is Skype?

Everybody knows!

Skype is a P2P (peer-to-peer) Voice-Over-IP (VoIP) client founded by Niklas Zennström and Janus Friis—also founders of the file sharing application Kazaa (the most downloaded software ever, Skype's 309 million registered users have made more than 100 billion minutes worth of free Skype-to-Skype calls).

Skype is an application that allows free phone calls between computers, and extremely cheap calls to (practically) everywhere on Earth!

Skype (founded in 2002, acquired by eBay in 2005; on September 1st, 2009, a group of investors led by Silver Lake bought 65% of Skype for $1.9 billion) is the fastest growing service in the history of the Internet.

Why is Skype so successful?

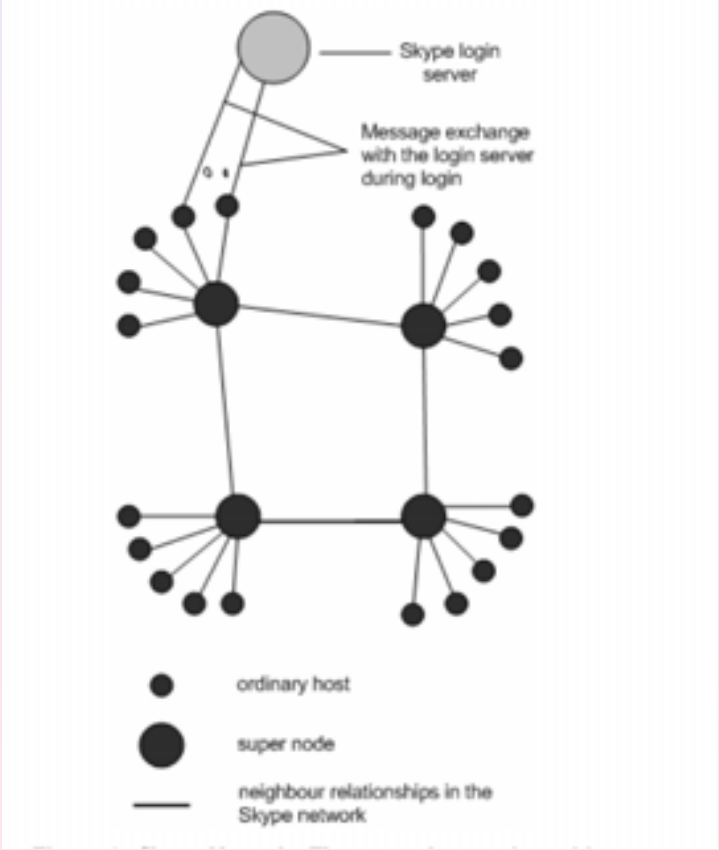
Because:

- it provides better voice quality than anybody else (Skype transmits the full range of human hearing, 20 Hz to 20KHz (compare with the best telephone supporting frequencies in the range 300Hz to 3.4KHz);
- it is reliable and can work almost seamlessly behind NAT's and firewalls;
- it is extremely easy to install and use (probably the most user-friendly application).

How does Skype actually work? 1

Skype network has three types of machines, *all running the same software and treated equally*:

- ordinary host (Skype Client)
- Super Node (SN)
- Skype login server



source: Baset & Schulzrinne, 2004, p.1

How does Skype actually work? 2

A Skype client (SC) (or ordinary host) is the computer of a regular Skype user connected to the network in order to communicate with other users.

An ordinary host connects to a Super Node (SN). Any computer with a public IP and proper hardware configuration can be an SN. An ordinary host must connect to an SN and must register itself with the Skype login server for a successful login.

The Skype login server is the only *central unit* in the whole network. It stores the usernames (Skype Name), e-mail addresses, and respective encrypted representations of passwords of all registered Skype users. There are about 20,000 SN out of many millions of registered Skype users logged on (8,257,048 at 25 May 08 time 09.28; 14, 839,372 at 20 September 2009 time 06.42).

The login process 1

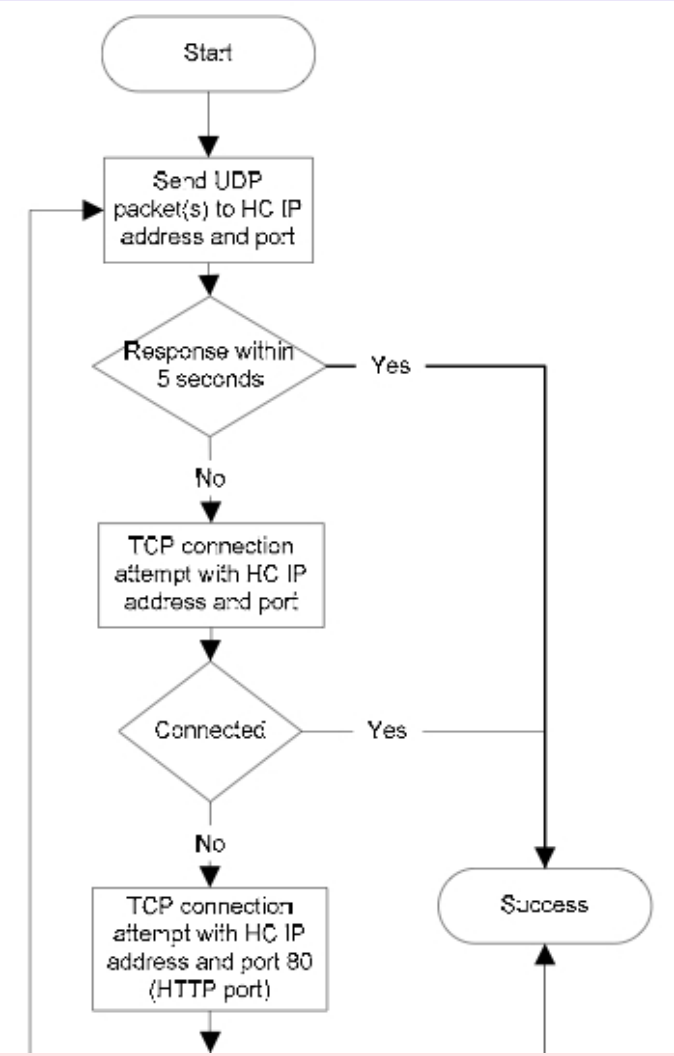
Any SC must connect to an SN, therefore, it maintains a local table that contains the IPs and corresponding ports of SNs—the host cache.

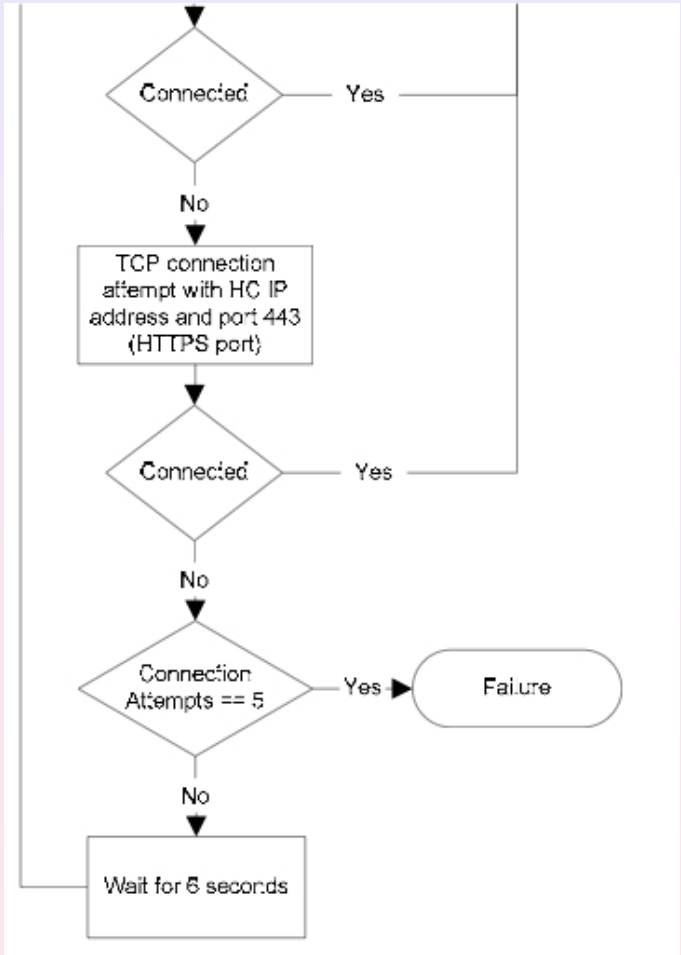
To start-up, SC reads the data from the host cache, takes the first IP (and port) from there, and tries to connect to this SN.

If the connection fails for some reason (e.g. SN is off line), then it reads the next line from the table.

In case it fails to connect to any of the IPs listed, the Skype returns a login error upon start-up. To log in the host cache must contain at least one valid entry—IP address and port number of an online SN.

The login process 2





source: Baset & Schulzrinne, 2004, p.3

Super Nodes 1

SNs (introduced in the third-generation P2P networks) allow

- improved search performance,
- reduced file-transfer latency,
- network scalability,
- and the ability to resume interrupted downloads and simultaneously download segments of one file from multiple peers.

SNs are also responsible for *Global Indexing*, the technology enabling fast searches for other users in the network. Skype guarantees that to find any user registered and logged in during the last 72 hours. Global Indexing allows Skype to build a **reliable** set of services atop a constellation of *unreliable* peers.

Super Nodes 2

The Skype network is self-modifiable.

Almost any computer (which is not behind a firewall) may turn into a SN—*without you even knowing it*. SNs store the *addresses* of up to several hundred SCs, without carrying any voice, text or file-transfer data. In that way, the more SCs come online, the more SNs become available to expand the capacity of the network.

Skype routing

Skype routes the traffic 'intelligently' by choosing the optimum data transfer path. Multiple paths are kept "open" and dynamically Skype chooses the best suited path at the time.

Skype uses either TCP or UDP protocols, hence it *breaks* the whole data stream into separate packets, which can take *different paths* to the end destination where the *final reassembling* is done.

Skype encryption

All Skype communications, voice conversations, text messages, file transfers, are encrypted between the caller and the called party. Encryption is necessary as all calls are routed through the public Internet.

Skype uses Advanced Encryption Standard, known as Rijndel.

Skype uses 256-bit encryption which has a total of 1.1×10^{77} possible keys to encrypt the data in each call or instant message of file transfer. Skype uses 1,536 or 2,048 (for paid services) bit RSA to negotiate symmetric AES keys.

Firewall and NAT 1

Voice-Over-IP (VoIP) was available for years, but has not reached the mainstream market because of the following reasons:

- cheap products do not compare well the standard phones in quality,
- over 50% of residential computers are unable to use the technology because of firewalls and Network Address Translation (NAT) gateway,
- the client needs a lot of technical configuration.

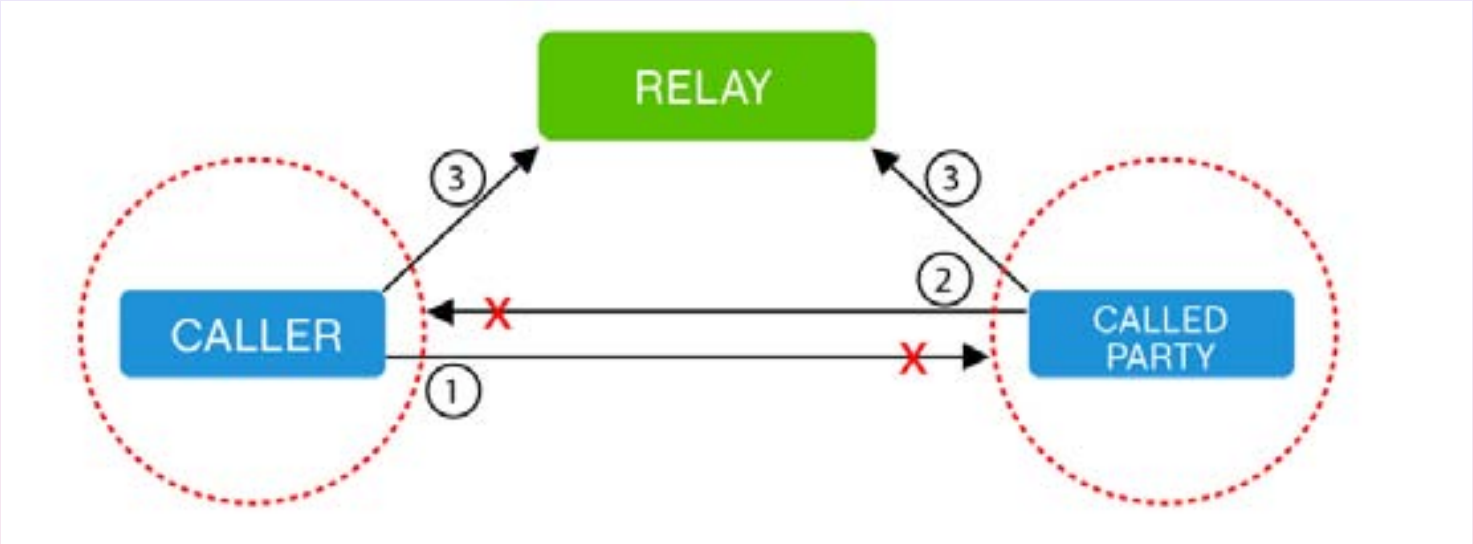
Firewall and NAT 2

If both SCs are on real IP hosts, then the media traffic flows directly between them over UDP. The size of the voice packet is 67B, which is actually the size of UDP payload. One second conversation results in roughly 140 voice packets being exchanged both ways, or 3–16 KBps.

If one of SC or both of them do not have a public IP, then they send voice traffic to another online SN over UDP or TCP.

If both sides are not speaking, voice packets will still be flowing between them. The purpose of these so called silent packages, a form of redundancy, is to keep the connection alive.

Firewall and NAT 3



source: www.skype.com

Skype has implemented a set of NAT traversal techniques which signal to the NAT devices that P2P sessions have been solicited and should be passed. In this way, Skype works behind the majority of firewalls and gateways with no special local configuration.

Not Becoming a Super Node

Three ways to prevent Skype from becoming a SN:

- Beginning with Skype 3.0, an explicit switch is provided in the registry settings to allow the disabling of SN functionality.
- Any computer hosted on a network that is behind a NAT device or restrictive firewall will disable SN functionality.
- Skype clients behind an HTTP or SOCKS5 proxy will not serve as SN.

Skype limitations

Some major limitations of Skype are:

- changing from normal phones to USB phones remains a problem for many users,
- Skype's completely proprietary nature flows against the open source trend (large companies don't appreciate Skype's hidden way of worming through corporate firewalls),
- SNs can generate a significant amount of bandwidth usage, hence some network providers (universities, for example) have banned Skype.

What's next for Skype? 1

- Face competition: VOIP Buster, VOIP Stunt, Gizmo, Free World Dialup, Woize, Xten, Google Talk, etc.
- Continue with improvements in basic technologies and variety and quality of services (fax). For example, screen sharing is getting better making Skype a competitor in the marked o videoconferencing.
- WiFi (short for "wireless fidelity") is becoming more and more popular. The number of cities providing free city-wide WiFi is fast growing. So what?

What's next for Skype? 2

Handheld or other portable wireless devices running Skype become a close substitute for mobile phones: the only difference is that calls are free or at very low rates!
Skype is available on most popular Java-enabled mobile phones from Motorola, Nokia, Samsung and Sony Ericsson: `http://www.skype.com/intl/en/download/skype/mobile`.

Skype is also available in iPhone an iPod touch, directly `http://www.skype.com/download/skype/iphone` or through other applications including Fring, IM+, Nimbuzz, Truphone available from iTunes.

Is growth a problem? Like any P2P network, additional users bring with them their own solution to growth (no more infrastructure is needed).

Fight in White Space 1

With the white-space-sensing technology Microsoft, Google (and others) can provide wireless Internet access at rates of gigabits of data per second instead of Wi-Fi's megabit-per-second speeds. The technology uses white-spaces which are unused bandwidth layered between TV channels that are buffer zones designed to keep broadcast signals from interfering.

Fight in White Space 2

As digital signals take up less airwave space than their analog signals, these spaces got even bigger in June 2009 when TV broadcasts in US went completely digital.
The Federal Communications Commission (FCC) is testing various technologies (including the one by I2R, a research institute in Singapore).

Fight in White Space 3

Adaptrum, Microsoft, Motorola, Philips and Singapore's Institute for Infocomm Research have prototype devices—a form of cognitive radio—which try to identify a slice of airwave space wherein a wireless device could operate without blocking other signals. But none was yet approved by FCC...