Code Division Multiplexing

Direct sequence:

1) To send bit sequence $x = x_1 x_2 \ldots x_n$, use pseudorandom bit sequence $y = y_1 y_2 \ldots y_n$ to compute

$$\begin{aligned} z &= z_1 z_2 \ldots z_n \\ &= (x_1 \oplus y_1)(x_2 \oplus y_2) \ldots (x_n \oplus y_n) \end{aligned}$$

2) Transmit z

3) To decode bit sequence $z = z_1 z_2 \ldots z_n$, compute

$$x = z \oplus y$$

Ex.: $x = 10010$, $y = 01011$

$\longrightarrow \quad z = x \oplus y = 10010 \oplus 01011 = 11001$

$\longrightarrow \quad z \oplus y = 11001 \oplus 01011 = 10010$

Pseudo-random y is called chipping code or pseudo-noise (PN) sequence.

In practice, single data bit encoded using $r > 1$ code bits.

Ex.: Suppose $r = 3$. To send single bit, say $x = 1$, "expand" x to $\tilde{x} = 111$ (r-fold duplication). If $y = 010$ then:

$\longrightarrow$ $z = \tilde{x} \oplus y = 111 \oplus 010 = 101$

$\longrightarrow$ $z \oplus y = 101 \oplus 010 = 111$

$\longrightarrow$ what next?

$\longrightarrow$ why use r-fold duplication?

Data rate usually slower than code rate

$\rightarrow |y| = r \cdot |x|$

$\rightarrow$ more frequent changes: "spreading"

Previous scheme works for single user

$\longrightarrow$ DSSS (direct sequence spread spectrum)

$\longrightarrow$ networking: support multiple users

$\longrightarrow$ how to do it?

Suppose N users, each sending a single bit: $x^1, x^2, \ldots, x^N$. Assume code rate r.

$\longrightarrow$ user i's expanded vector: $\tilde{x}^i = (x^i, x^i, x^i)$

Supposing N random chipping codes (one per user)

$$\{y^1, y^2, \ldots, y^N\}$$

will the encoding

$$z = \tilde{x}^1 \oplus y^1 + \tilde{x}^2 \oplus y^2 + \cdots + \tilde{x}^N \oplus y^N$$

work if user i decodes by EXOR'ing z with y^i ?

$\longrightarrow$ i.e., what is $z \oplus y^i$?

$\longrightarrow$ does it equal $\tilde{x}^i$?

A different twist:

- represent bits as $1, -1$ (not $1, 0$)

  $\rightarrow x^i \in \{1, -1\}$

- assume chipping codes $\{y^1, y^2, \ldots, y^N\}$ are orthonormal

  $\rightarrow$ i.e., $y^i \circ y^j = 0$ $(i \neq j)$ and $y^i \circ y^i = 1$

  $\rightarrow$ "$\circ$" is the dot product

Encoding (combined signal):

$$z = x^1 y^1 + x^2 y^2 + \cdots + x^N y^N$$

$\longrightarrow$ note: x^i is a scalar, y^i is a vector

Decoding (for user i):

$$\begin{aligned} y^i \circ z &= x^1 y^i \circ y^1 + \cdots + x^i y^i \circ y^i + \cdots + x^N y^i \circ y^N \\ &= x^i \end{aligned}$$

$\longrightarrow$ exact recovery

$\longrightarrow$ CDMA (code division multiple access)

Ex.: $N = 4$, $r = 4$, and chipping code y^i's are

$$(1, 1, 1, 1),\ (-1, -1, 1, 1),\ (-1, 1, -1, 1),\ (-1, 1, 1, -1)$$

$\longrightarrow$ note: orthogonal but not orthonormal

$\longrightarrow$ $y^i \circ y^i = 4 \ (= r)$

$\longrightarrow$ hence, $y^i \circ z = 4\, x^i$

$\longrightarrow$ r is also called "gain"

$\longrightarrow$ why useful?

Frequency hopping:

Use pseudorandom number sequence as key to index a set of carrier frequencies $f_1, f_2, \ldots, f_m$.

$\longrightarrow$ frequency spreading

Receiver with access to pseudorandom sequence can decode transmitted signal.

$\longrightarrow$ receiver's tuner must jump around

$\longrightarrow$ code narrowband input as broadband output

$\longrightarrow$ frequency spreading

$\longrightarrow$ FHSS (frequency hopping spread spectrum)

DSSS vs. FHSS?

Benefits of CDMA:

- more secure against eavesdropping

  $\longrightarrow$ confidentiality

- resistant to jamming

  $\longrightarrow$ must jam a wider spectrum: more difficult

  $\longrightarrow$ first introduced in the military context

- noise resistance

  $\longrightarrow$ code rate r

- graceful degradation

  $\longrightarrow$ compared to TDM

Deployment and usage:

$\longrightarrow$ wireless LAN (WLAN): DSSS and FHSS

$\longrightarrow$ cellular (e.g., Sprint PCS, Verizon): CDMA

Competing with CDMA cellular: the rest!

$\longrightarrow$ majority

$\longrightarrow$ AT&T Wireless, Cingular, etc.

$\longrightarrow$ dominant standard: GSM

$\longrightarrow$ uses TDMA (time division multiple access)

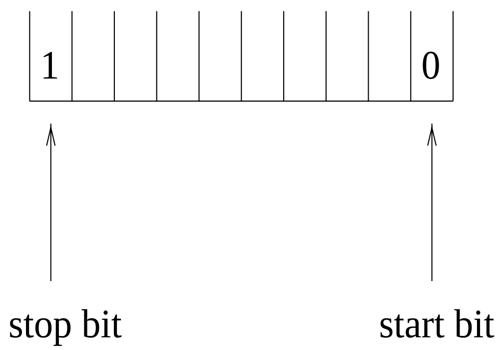
$\longrightarrow$ TDMA: FDM + TDM

Framing

$\longrightarrow$ packet layout

$\longrightarrow$ variety of framing conventions

Asynchronous: e.g., ASCII character transmission between dumb terminal and host computer.



- each character is an independent unit

  $\rightarrow$ "asynchronous"

- receiver needs to know bit duration

  $\rightarrow$ bit rate assumed known between sender/receiver

Overhead problem; assuming 1 start bit, 1 stop bit, 8 data bits, only 80% efficiency.

$\longrightarrow$ inefficient for long messages

iPod & radio example:

$\longrightarrow$ coding used asynchronous?

$\longrightarrow$ clock needed?

Synchronous: "Byte-oriented"; e.g., PPP, BISYNC

SYN	SYN	SOH	Header	STX	Body	ETX	CRC

$\longrightarrow$ `SYN, SOH, STX, ETX, DLE`: sentinels

$\longrightarrow$ variable body size

Two problems:

- How to maintain synchronization if |Body| is large?
- Control characters within body of message.

$\longrightarrow$ inefficient for short messages

$\longrightarrow$ efficiency approaches 1 as $|\text{Body}| \rightarrow \infty$

"Bit-oriented"; e.g., HDLC

$\longrightarrow$ bit is the unit

Use fixed *preamble* and *postamble*; simply a bit pattern.

$\longrightarrow$ 01111110

How to avoid confusing 01111110 in the data part?

$\longrightarrow$ bit stuffing

$\longrightarrow$ for data: stuff 0 after 5 consecutive 1's