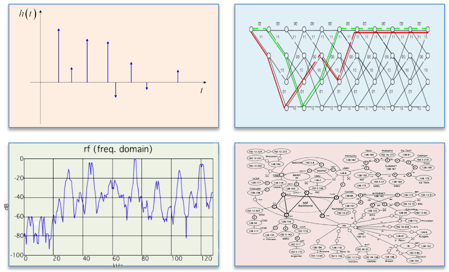


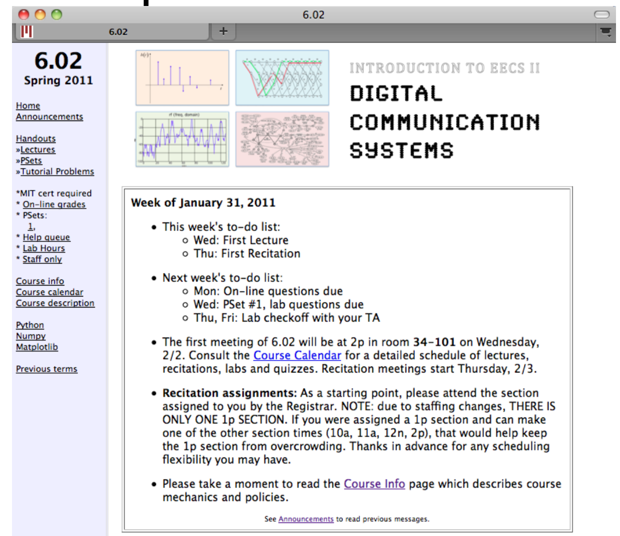
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

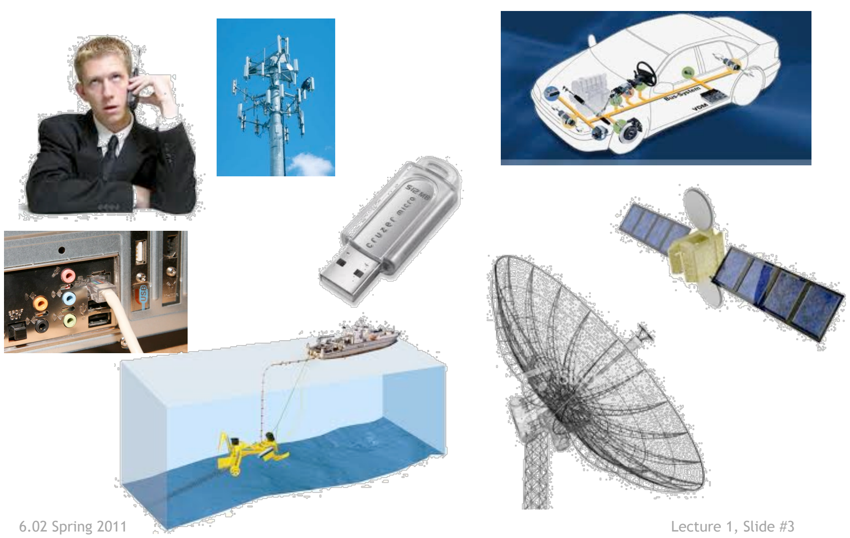
6.02 Spring 2011
Lecture #1

- Engineering goals for communication systems
- Measuring information
- Huffman codes

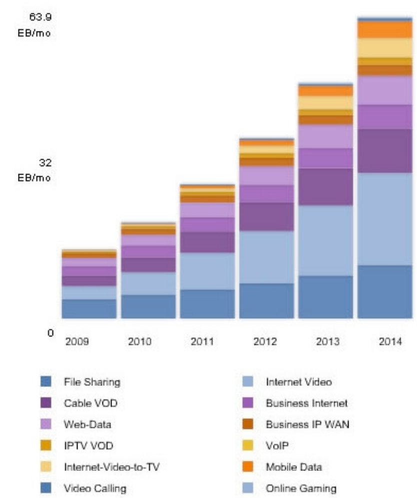
http://web.mit.edu/6.02



Digital Communications

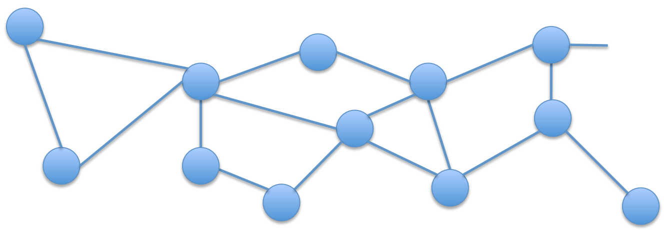


Internet: 10^{21} bytes/year by 2014!



*Cisco VNI June 2010

6.02 Syllabus



Point-to-point communication channels (transmitter→receiver):
- Encoding information
- Models of communication channels
- Noise, bit errors, error correction
- Sharing a channel

Multi-hop networks:
- Packet switching, efficient routing
- Reliable delivery on top of a best-efforts network

Engineering Goals for Networks

(Class discussion)

Information Resolves Uncertainty

In information theory, information is a mathematical quantity expressing the probability of occurrence of a particular sequence of symbols as contrasted with that of alternative sequences.

Information content of a sequence *increases* as the probability of the sequence *decreases* – likely sequences convey less information than unlikely sequences.

We're interested in encoding information efficiently, i.e., trying to *match the data rate to the information rate*. We'll be thinking about:

- Message content (one if by land, two if by sea)
- Message timing (No lanterns? No message!)

Measuring Information Content

Claude Shannon, the father of information theory, defined the information content of a sequence as

$$\log_2\left(\frac{1}{p(seq)}\right)$$

The unit of measurement is the bit (<u>b</u>inary dig<u>it</u>: "0" or "1").

1 bit corresponds to p(seq) = ½, e.g., the probability of a heads or tails when flipping a fair coin.

This lines up with our intuition: we can encode the result of a single coin flip using just 1 bit: say "1" for heads, "0" for tails. Encoding 10 flips requires 10 bits.

Expected Information Content: Entropy

Now consider a message transmitting the outcome of an event that has a set of possible outcomes, where we know the probability of each outcome.

Mathematicians would model the event using a discrete random variable X with possible values $\{x_1, \ldots, x_n\}$ and their associated probabilities $p(x_1), \ldots, p(x_n)$.

The *entropy* H of a discrete random variable X is the expected value of the information content of X:

$$H(X) = E(I(X)) = \sum_{i=1}^{n} p(x_i)\log_2\left(\frac{1}{p(x_i)}\right)$$

Okay, why do we care about entropy?

Entropy tells us the average amount of information that must be delivered in order to resolve all uncertainty. This is a lower bound on the number of bits that must, on the average, be used to encode our messages.

If we send fewer bits on average, the receiver will have some uncertainty about the outcome described by the message.

If we send more bits on average, we're "wasting" the capacity of the communications channel by sending bits we don't have to. "Wasting" is in quotes because, alas, it's not always possible to find an encoding where the data rate matches the information rate.

Achieving the entropy bound is the "gold standard" for an encoding: entropy gives us a metric to measure encoding effectiveness.

Special case: all p_i are equal

Suppose we're in communication about an event where all N outcomes are equally probable, i.e., $p(x_i) = 1/N$ for all i.

$$H_{before}(X) = \sum_{i=1}^{N}\left(\frac{1}{N}\right)\log_2\left(\frac{1}{1/N}\right) = \log_2(N)$$

If you receive a message that reduces the set of possible outcomes to M equally probable choices, the entropy after the receipt of the message is

$$H_{after}(X) = \sum_{i=1}^{M}\left(\frac{1}{M}\right)\log_2\left(\frac{1}{1/M}\right) = \log_2(M)$$

The information content of the received message is given by the change in entropy:

$$H_{message}(X) = H_{before} - H_{after} = \log_2(N) - \log_2(M) = \log_2(N/M)$$

Example

We're drawing cards at random from a standard 52-card deck:

Q. If I tell you the card is a ♤, how many bits of information have you received?

A. We've gone from N=52 possible cards down to M=13 possible cards, so the amount of info received is $\log_2(52/13) = 2$ bits.

This makes sense, we can encode one of the 4 (equally probable) suits using 2 bits, e.g., 00=♡, 01=♢, 10=♧, 11=♤.

Q. If instead I tell you the card is a seven, how much info?

A. N=52, M=4, so info = $\log_2(52/4) = \log_2(13) = 3.7$ bits

Hmm, what does it mean to have a fractional bit?

Example (cont'd.)

Q. If I tell you the card is the 7 of spades, how many bits of information have you received?

A. We've gone from N=52 possible cards down to M=1 possible cards, so the amount of info received is $\log_2(52/1)$ = 5.7 bits.

Note that information is additive (5.7 = 3 + 2.7)!

But this is true only when the separate pieces of information are independent (not redundant in any way).

So if I sent first sent a message the card was black (i.e., a ♤ or ♧) – 1 bit of information since p(♤ or ♧) = ½ – and then sent the message it was a spade, the total information received is *not* the sum of the information content of the two messages since the information in the second message overlaps the information of the first message.

Fixed-length Encodings

An obvious choice for encoding equally probable outcomes is to choose a fixed-length code that has enough sequences to encode the necessary information

- 96 printing characters → 7-bit ASCII
- Unicode characters → UTF-16
- 10 decimal digits → 4-bit BCD (binary coded decimal)

Fixed-length codes have some advantages:

- They are "random access" in the sense that to decode the n^{th} message symbol one can decode the n^{th} fixed-length sequence without decoding sequence 1 through n-1.
- Table lookup suffices for encoding and decoding

Improving on Fixed-length Encodings

$choice_i$	p_i	$\log_2(1/p_i)$
"A"	1/3	1.58 bits
"B"	1/2	1 bit
"C"	1/12	3.58 bits
"D"	1/12	3.58 bits

The expected information content in a choice is given by the entropy:
 = (.333)(1.58) + (.5)(1) + (2)(.083)(3.58) = 1.626 bits

Can we find an encoding where transmitting 1000 choices requires 1626 bits on the average?

The "natural" fixed-length encoding uses two bits for each choice, so transmitting the results of 1000 choices requires 2000 bits.

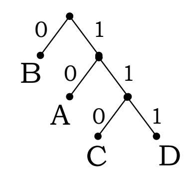
Variable-length encodings

(David Huffman, MIT 1950)

Use shorter bit sequences for high probability choices, longer sequences for less probable choices

$choice_i$	p_i	*encoding*
"A"	1/3	10
"B"	1/2	0
"C"	1/12	110
"D"	1/12	111

BC A BA D
011010010111



Huffman Decoding Tree

Expected length
=(.333)(2)+(.5)(1)+(2)(.083)(3)
= 1.666 bits

Transmitting 1000 choices takes an average of 1666 bits… better but not optimal

Another Variable-length Code (not!)

Here's an alternative variable-length for the example on the previous page:

Letter	*Encoding*
A	0
B	1
C	00
D	01

Why isn't this a workable code?

The expected length of an encoded message is

$$(.333+.5)(1) + (.083 + .083)(2) = 1.22 \text{ bits}$$

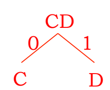
which even beats the entropy bound ☺

Huffman's Coding Algorithm

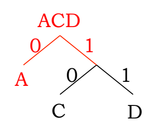
- Begin with the set S of symbols to be encoded as binary strings, together with the probability p(s) for each symbol s in S. The probabilities sum to 1 and measure the frequencies with which each symbol appears in the input stream. In the example from the previous slide, the initial set S contains the four symbols and their associated probabilities from the table.
- Repeat the following steps until there is only 1 symbol left in S:
  - Choose the two members of S having lowest probabilities. Choose arbitrarily to resolve ties.
  - Remove the selected symbols from S, and create a new node of the decoding tree whose children (sub-nodes) are the symbols you've removed. Label the left branch with a "0", and the right branch with a "1".
  - Add to S a new symbol that represents this new node. Assign this new symbol a probability equal to the sum of the probabilities of the two nodes it replaces.

Huffman Coding Example

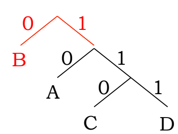
- Initially S = { (A, 1/3) (B, 1/2) (C, 1/12) (D, 1/12) }
- First iteration
  - Symbols in S with lowest probabilities: C and D
  - Create new node
  - Add new symbol to S = { (A, 1/3) (B, 1/2) (CD, 1/6) }



- Second iteration
  - Symbols in S with lowest probabilities: A and CD
  - Create new node
  - Add new symbol to S = { (B, 1/2) (ACD, 1/2) }



- Third iteration
  - Symbols in S with lowest probabilities: B and ACD
  - Create new node
  - Add new symbol to S = { (BACD, 1) }



- Done

Huffman Codes – the final word?

- Given static symbol probabilities, the Huffman algorithm creates an optimal encoding when each symbol is encoded separately. (optimal ≡ no other encoding will have a shorter expected message length)
- Huffman codes have the biggest impact on average message length when some symbols are substantially more likely than other symbols.
- You can improve the results by adding encodings for symbol pairs, triples, quads, etc. From example code:
  Pairs: 1.646 bits/sym, Triples: 1.637, Quads 1.633, …
  But the number of possible encodings quickly becomes intractable.
- Symbol probabilities change message-to-message, or even within a single message.
- Can we do adaptive variable-length encoding?
  - Tune in next time!