Digital Communication

A COURSE MATERIAL FOR

CSC 222

O.D ADENIJI (Ph.d)

Lecturer1

Computer Science Department

University of Ibadan

General Introduction and Course Objective

This textbook presents the fundamental concepts of underlying design of modern digital communication systems. The presentation of topics in this book will provide reader the insight to cutting edge of digital communications research and development. The numerous examples in the self-test question are used to illustrate the key principles, with a view of allowing the reader to perform detailed computations and simulations based on the ideas presented in the text. Generating numerical algorithm, results and plots will provide constructive feedback for student in developing solution to real life problems. However, the course design or self-study will expose and stimulates the reader to discover new idea.

The objective of the book is to develop an understanding to the building blocks of digital communication system. To give the mathematical background for communication signal analysis. To understand and analyze the signal flow in a digital communication system. The analysis of error performance of a digital communication system in presence of noise and other interferences. To understand concept of spread spectrum communication system.

Digital Communication Systems combines theoretical knowledge and practical application that focuses on the rules of functional digital communication. Digital Media such as Web, social and mobile technologies has drastically affected and expanded the ways in which users communicate, including the creation, dissemination and consumption of news and information.

The basic properties of several physical communication channels used in digital communication systems are explained. The basic methods of error correction and detection. The illustration of the theory developed in each chapter provides a concise approach to digital communications, with practical examples and problems.

Course Curriculum Contents

The provision of insightful descriptions and intuitive explanations of all complex concepts in digital communication systems using basic rules such as Overview, Abstraction and Information Compression Errors in communication. The basics Error correction concept on: Linear block codes, Viterbi decoding, Convolution codes, Noise and Transmitting on the physical Channels. The theory and case studies is a unique feature of this book. The discussion on major aspects of communication networks and multiuser communications such as spectral representation of signal, Modulation /Demodulation, Medium Access, contention window, CSMA, TDMA, Packet Switching, Network routing, Reliable Data Transport and sliding window.

Table of Contentent

Study Session *1.*Overview, Abstraction and Information Compression Error in Communication Error Correction

Introduction

In this study session, you will learn about information compression error as an important concept of digital communication. A further explanation on Communication Error correction will be discussed. The objective of the lecture is to introduce information compression Error by highlighting the characteristic of data compression. To ensure that students understand the concept of data compression, coding and information compression Error.

1.1: Concept of Data Compression and Coding

1.1.1: What is Data Compression and Coding

Data compression is the concept of reducing redundancy in stored or communicated data, thereby increasing effective data density. A data compression method has existed for almost forty years from the work of scientists such as Shannon, Fano, and Human in the late '40s to a technique developed in 1986. Data compression can be applied in the area of file storage and distributed systems. Data compression may be viewed as a branch of information theory in which the primary objective is to minimize the amount of data to be transmitted. Information theory is defined to be the study of efficient coding and its concerns, in the form of speed of transmission and probability of error. The characterization of data compression is involved in transforming a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose lengths are small.

Data processing applications require storage of large volumes of data due to increase in number of applications, constant increase of computers algorithm in new disciplines and proliferation of computer communication networks resulting in the massive transfer of data over communication links. However, the advantages of compressing data are to store or transmit by reducing storage and communication costs. When the amount of data to be transmitted is reduced, the effect is increasing the capacity of the communication channel. Compressing a file to 50% or half of its original size is equivalent to doubling the capacity of the storage medium. It may then become viable to store the data at a higher, thus faster method. The level of the storage hierarchy will reduce the load on the input/output channels of the computer system.

1.1.2: Coding

Code: A code is the mapping of source messages for example words from the source alphabet into code words i.e. words of the code alphabet. The source messages are the basic units into which the string to be represented is partitioned. The basic units may be single symbols from the source alphabet or strings of symbols. For string example, α = {a; b; c; d; e; f; g; space}. The explanation will be taken to be β {0; 1}. Codes can be grouped into block-block, block-variable, variable-block or variable-variable. The block-block indicates that the source messages and code words are of fixed length and variable-variable codes map variable-length source messages into variable-length code words.

A block-block code is shown in table 1.1

Source Message	Code word
A	000
B	001
C	010
D	011
E	100
F	101
G	110
Space	111

If the string EXAMPLE were coded using the table 1.1 code, the length of the coded message would be 120 but using table 1.2 the length of coded message would be 30 as shown below.

A variable-variable code is shown in table 1.2

Source Message	Code word
Aa	0
Bbb	1
Cccc	10
Ddddd	11
Eeeeeee	100
Ffffffff	101
Gggggggggg	110
Space	111

A code can also be defined to be a mapping from a source alphabet to a code alphabet; The process of transforming a source ensemble into a coded message is coding or encoding. The encoded message may be referred to as an encoding of the source ensemble. The algorithm which constructs the mapping and uses it to transform the source ensemble is called the **encoder**. The **decoder** performs the inverse operation, restoring the coded message to its original form.

A distinct code is uniquely decodable if every codeword is identifiable when absorbed in a sequence of codewords. Clearly, each of these features is desirable. The codes of table 1.1 and table 1.2 are both distinctive, but the code of Figure 1.2 is not uniquely decodable. For example, the coded message 11 could be decoded as either \ddddd" or \bbbbbb". A uniquely decodable code is a prefix code (or prefix free code) if it has the prefix property, which requires that no codeword is a proper prefix of any other codeword. All uniquely decodable block-block and variable-block codes are prefix codes. The code with code words {1; 100000; 00} is an example of a code which is uniquely decodable but which does not have the prefix property. **Prefix codes** are instantaneously decodable; that is, they have the desirable property that the coded message can be parsed into codewords without the need for a look ahead. In order to decode a message encoded using the codeword set {1; 100000; 00}, look ahead is required. For example, the first codeword of the message 1000000001 is 1, but this cannot be determined until the last (tenth) symbol of the message is read (if the string of zeros had been of odd length, then the first codeword would have been 100000).

A code of a discrete information source is said to be non-singular when different source symbols map to different codewords, A code of a discrete source is said to be non-ambiguous if and only if each (finite length) sequence of code words uniquely corresponds to a single source message. Consider the source consisting of the three symbols a, b and c. Its messages can be any sequence of these symbols; for instance “aabca” is a message of this source.
Then, the following encoding of this source:

a 7→ 1 b 7→ 00 c 7→ 11 is ambiguous.

For instance, there is no way, once encoded, to distinguish the message “aaaa” from the message “cc”. Indeed, both are encoded as “1111”
A code of a discrete source is said to be prefix-free when no codeword is the prefix of another codeword
The three general reasons for coding are basically for different purposes:

1. **Coding for compressing data**: reducing (on average) the length of the messages. This means trying to remove as much redundancy in the messages as possible.
2. **Coding for ensuring the quality of transmission in noisy conditions.** This requires adding redundancy in order to be able to correct messages in the presence of noise.
3. **Coding for secrecy**: making the message impossible (or hard) to read for unauthorized readers. This requires making access to the information content of the message hard.

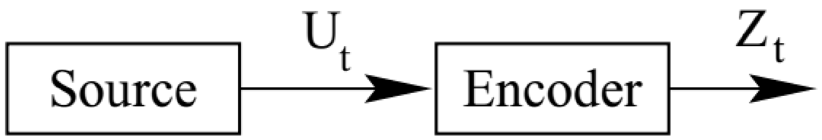


Figure 1: Basic schema for source coding: the source symbol Ut at time t is transformed into the corresponding codeword Zt.

An information source is a message generator. These information sources will generate sequences of symbols. A symbol is simply an element of a set, called an alphabet. Newspaper can be regarded as a discrete information source whose messages are the texts contained in the newspaper, the symbols simply being the letters of the usual alphabet (including the whitespace and other punctuation marks). The messages from the source then come into a encoder which transforms them into a sequence of codewords as shown in figure 1 above.

An information source Ut, and more generally U when the time index is not pertinent, is a random process on a given alphabet VU; i.e. a sequence of random variables on VU, each symbol as a probability P (Ut = ui) to be emitted by the source at time t. The source is said to be memoryless if the probability for a symbol ui to be emitted does not depend on the past emitted values; i.e. if

$\forall t \geq 1\ \forall ui \in VU\ P(Ut = ui|U1...Ut\text{-}1) = P(Ut = ui)$

1.2: what "efficient" means for a compression code

Our goal is to code the information source so as to minimize the average code length; i.e. the average length of a sequence of codewords. If we considered that source symbol ui (1 ≤ i ≤ N) has a probability pi to be emitted, and denoting li the length of the corresponding codeword, the expected code length∗ E [L] is the expected value of the length of a codeword, i.e.

$$E[L] = \sum_{i=1}^{N} p_i\ l_i$$

Equation ---------------1.1

$$\text{if } p_i > p_j \text{ and } l \geq l' \text{ then } p_i\, l + p_j\, l' \geq p_i\, l' + p_j\, l.$$

When precision is required, the expected length of the code Z will be denoted by E [LZ]. For any discrete memory less information source of entropy H(U), the expected code length E [L] of any D-ary prefix-free code for this source satisfies:

$$E\,[L] \geq \frac{H(U)}{\log D}$$ Equation---------------1.2

The above equation is known as Shannon coding theorem 1

1.2.2: Huffman Coding Algorithm

Huffman Coding Algorithm is used to construct an optimum D-ary prefix-free code for a discrete memoryless information source U with n symbols. Consider the unfair dice with the following probability distribution:

1	2	3	4	5	6
0.17	0.12	0.10	0.27	0.18	0.16

For this code, we will use the convention to always give the label 0 to the least probable branch and the label 1 to the most probable branch. The new nodes introduced will be named 7, 8, etc..., in that order. The following steps are used in constructing the algorithm.

1. What are the first two nodes to be regrouped? What is the corresponding probability?
2. What are then the next two nodes that are regrouped? What is the corresponding probability?
3. Keep on giving the names of the two nodes to be regrouped and the corresponding probability.
4. Give the Huffman code found for this source

Huffman coding consists in iteratively regrouping the least two probable values. Let us then first order the source messages by increasing probability:

3	2	6	1	5	4
0.10	0.12	0.16	0.17	0.18	0.27

1) The first two values to be regrouped are then 3 and 2, leading to a new node ”7” whose probability is 0.10 + 0.12 = 0.22.
2) The next iteration of the algorithm now regroups 6 and 1: probability is 0.33. The new set of values is therefore:

5	7	4	8
0.18	0.22	0.27	0.33

3. 9 is made with 5 and 7, its probability is 0.4

   10 is made with 4 and 8, its probability is 0.6

   11 is made with 9 and 10, its probability is 1.0

1.3: Information compression Error

Information compression Error is a process of using less memory or use less network bandwidth to encode the same information in fewer bits. Error detection and correction is the process of adding redundancy to detect and fix up loss or damage. *Information compression **Error*** can detect damaged bits and can correct errors. To encode frequent letters with fewer bits, less frequent things with more bits are required in order to trades complexity against space e.g Huffman code. Also encode runs of identical things with a count e.g World Wide Web => WWWC => W3C. So, words do not occur equally often. compression adapts to properties of particular input while the algorithm is independent of nature of the input. A good example of compression algorithm is **Lempel-Ziv coding**. **A** dictionary is included in the compressed data. Lempel-Ziv is the basis of PKZip, Winzip, gzip, GIF. Compresses Bible from 4.1 MB to 1.2 MB (typical for text).

The following provides examples of how information is compressed:

JPEG (Joint Photographic Experts Group) picture compression: The following steps can be used.

1. Use lossy compression scheme based on file complexity
2. Digitize picture into pixels
3. Discard some color information (use fewer distinct colors)
4. Adopt less sensitive color variation than brightness
5. Discard some fine detail
6. Originality provide a decompressed image as sharp as the original image
7. Discard some fine gradations of color and brightness
8. use Huffman code, run-length encoding, etc., to compress the resulting stream of numeric values
9. Compression is usually 10:1 to 20:1 for pictures but for it can vary. Used in web pages, digital cameras.

MPEG (Moving Picture Experts Group) movie compression: **The following steps can be used.**

1. Use a lossy compression scheme, based on human perceptions
2. use JPEG for individual frames (spatial redundancy)
3. Develop a compression of temporal redundancy
4. Segment the image in blocks as required
5. Transmit the block if there is no change but not the content
6. Transmit the block with the amount of motion
7. Predict the motion by encoding the expected differences plus correction.
8. separate moving parts from the static background.

It is correct that some common numbers have no redundancy thus can't detect when an error might have occurred. – e.g., SSN -- any 9-digit number is potentially valid. If some extra data is added or if some possible values are excluded, this can be used to detect and even correct errors. Common examples include ATM & credit card numbers, ISBN for books and bar codes.

Let consider ATM card checksum credit card / ATM card checksum:

1. Starting at rightmost digit:

2. Multiply digit alternately by 1 or 2

3.If result is > 9 subtract 9

4.Add the resulting digits

5. Sum should be divisible by 10

- ✓ e.g., 12345678 is invalid
- ✓ 8 + (14-9) + 6 + (10-9) + 4 + 6 + 2 + 2 = 34
- ✓ but 42345678 is valid
- ✓ 8 + (14-9) + 6 + (10-9) + 4 + 6 + 2 + 8 = 40.

Summary of lecture 1

1. Data compression may be viewed as a branch of information theory in which the primary objective is to minimize the amount of data to be transmitted.
2. A code can also be defined to be a mapping from a source alphabet to a code alphabet; The process of transforming a source ensemble into a coded message is coding or encoding.
3. Prefix-Free Codes:

   • no codeword is a prefix of another

   • prefix-free =⇒ non-ambiguous =⇒ non-singular

   • prefix-free ≡ instantaneously decodable
4. Huffman code:

   1. introduce 1 - n mod (D - 1) unused leaves with probability 0

   2. recursively regroup least probable nodes

   3. is optimal (regarding the expected code length) among non-ambiguous codes.
5. ***Information compression Error is a process of using*** less memory or use less network bandwidth to encode the same information in fewer bits. Error detection and correction is the process of adding redundancy to detect and fix up loss or damage.

Self-Assessment Question (SAQs) for lecture 1

The Self-Assessment of this lecture 1 can be assessed by answering the following question as discussed in the lecture note.

SAQ 1.1

Consider some information source U, the symbols of which are $u_1 = 1$, $u_2 = 2$, $u_3 = 3$,and $u_4 = 4$, with the following probability distribution:

u_i	1	2	3	4
$P(U = u_i)$	0.5	0.25	0.125	0.125

Consider then the following encoding of it (where zi is the codeword for ui):

z_1	z_2	z_3	z_4
0	10	110	111

1. Is this code non-ambiguous?

2. Encode the message 1423312.

3. Decode the sequence 1001101010

SAQ 1.2

Consider some information source U, the symbols of which are u1 = 1, u2 = 2, u3 = 3, and u4 = 4, with the following probability distribution: What is the expected code length?

u_i	1	2	3	4	5
$P(U = u_i)$	0.125	0.3	0.125	0.25	0.2

Consider then the following encoding of it (where zi is the codeword for ui):

z_1	z_2	z_3	z_4	z_5
1110	110	10	1111	0

SAQ 1.3

1. Compresses Bible from 4.1 MB to 1.2 MB (typical for text).
2. Encode 2 stereo channels as 1 plus difference

3.use a dictionary of encoded things, and refer to it (Lempel-Ziv)

Study Session 2: Basic Error Correction.

Introduction

This lecture will focus on how coding can help in to transmit information in a reliable way, even in the presence of noise during the transmission. The basic idea of such coding is to try to add enough redundancy in the coded message so that transmitting it in "reasonably" noisy conditions leaves enough information undisturbed for the receiver to be able to reconstruct the original message without distortion. Basics Error correction and types of error correction codes.

2.1: Basics Error Correction

The two fundamental notions ruling noisy transmissions are the channel capacity and the rate used for transmitting the symbols of the messages. A communication channel is a shorter channel that represents all that could happen to the transmitted messages between their emission and their reception

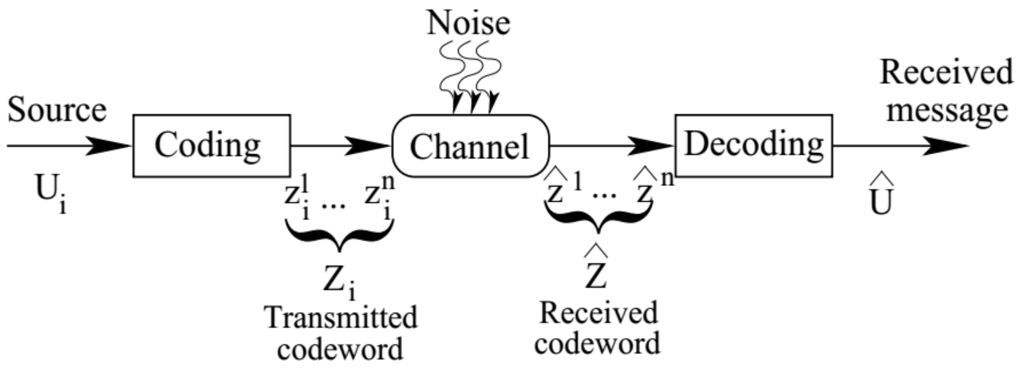


Fig 2.1: Error Correcting Communication over a Noisy Channel

Let transmit the 8 following messages: 000, 001, 010, 011, 100, 101, 110 and 111. Suppose that the channel used for transmission is noisy in such a way that it changes one symbol over ten, regardless of everything else; i.e. each symbol has a probability p = 0.1 to be" flipped" (0 into 1, and 1 into 0). Such a channel is a BSC with an error rate equal to p = 0.1, What is the probability to transmit one of our messages correctly? Regardless of which message is sent, this probability is $(1 - p)^3 = 0.9^3 = 0.719$. It shows the corresponding to the probability to transmit 3 times one bit without error. Therefore, the probability to receive and the erroneous message is 0.281, i.e 28%; which is quite high. If we decide to code each symbol of the message by twice itself, the table below will be formed

message	000	001	010	011	100	...	111
code	000000	000011	001100	001111	110000	...	111111

What is now the probability to have a message sent correctly? In the same way, this is (1 - p)6 = 0.531. And the probability to receive and the erroneous message is now 0.469... ...worse than previously, it seems. Not detecting an erroneous message means that two corresponding symbols have both been changed. If for instance we sent 000000, but 110000 is received, there is no way to see that some errors occurred. However, if 010000 is received, clearly at least one error occurred. The drive of a channel is to transmit messages ("information") from one point (the input) to another (the output). The channel capacity∗ precisely measures this ability: it is the maximum average amount of information the output of the channel can bring on the input.

2.2: Linear Block Codes

Linear codes are block-codes on which an algebraic structure is added to help to decode: the vector space structure. Transmission errors can occur, when 1's become 0's and 0's become 1's when transmitted through a noisy channel. To correct the errors, redundancy of bits is added to the information sequence, at the receiver end in order to locate transmission errors. The code {1101, 0110, 1110} is not a linear code since 0000 is not part of it (it could therefore not be vector space). The (binary) code {0000, 1101, 0110, 1011} is a linear code since any (binary) linear combination of codewords is also a codeword.

Definition of block coding:

1. A message **m** is a binary sequence of size k: **m** $2\ A^k$.

2. To each message, **m** corresponds a codeword which is a binary sequence **c** of size n: $n > k$.

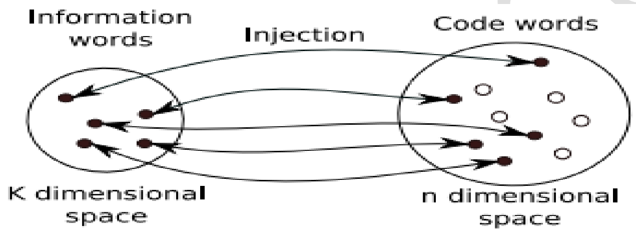


Fig 2.2: K dimensional space and n dimension space

The code space contains 2^n points but only 2^k of them are valid codewords. Code must be a one-to-one relation (injection). For example:

$$\mathbf{m} \in \{00, 01, 10, 11\}$$
$$\mathbf{c} \in \{000, 011, 101, 110\}$$

The transmission rate of a code is defined as R= k/n-------------------- *2.1*.

For example, for the previous code $n = 3$, $k = 2$ so the rate is 2/3.

Minimum Weight and Minimum Distance Equivalence For every linear code C $d_{min}(C) = w_{min}(C)$

where $w_{min}(C)$is the minimum weight of the code, i.e. the smallest weight of non-zero codewords:

$$d_{min} = \min_{\mathbf{v},\mathbf{w}} \{ d(\mathbf{v},\mathbf{w}) : \mathbf{v},\mathbf{w} \in \mathcal{C} \quad \text{and} \quad \mathbf{v} \neq \mathbf{w} \}$$

If the code is linear:

$$\begin{aligned} d_{min} &= \min\{ w(\mathbf{v}+\mathbf{w}) : \mathbf{v},\mathbf{w} \in \mathcal{C} \quad \text{and} \quad \mathbf{v} \neq \mathbf{w} \} \\ &= \min\{ w(\mathbf{x}) : \mathbf{x} \in \mathcal{C} \quad \text{and} \quad \mathbf{x} \neq \mathbf{0} \} \\ &= w_{min} \end{aligned}$$

Theorem: The minimum distance of a linear block code is equal to the minimum weight of its non zero codeword.

Summary of lecture 2

Self-Assessment Question (SAQs) for lecture 2

SAQ 2.1

On Trinity's advice, Morpheus decides to augment each codeword in C from the previous problem with an overall parity bit, so that each codeword has an even number of ones. Call the resulting code C^+.

(a) Explain whether it is True or False that C^+ is a linear code.

(b) What is the minimum Hamming distance of C^+?

(c) Write down the generator matrix, G^+, of code C^+.

Express your answer as a concatenation (or stacking) of G (the generator for code C) and another matrix (which you should specify). Explain your answer

SAQ 2.2

The Matrix Reloaded. Neo receives a 7-bit string, $D_1D_2D_3D_4P_1P_2P_3$ from Morpheus, sent using a code, C, with parity equations

$P_1 = D_1 + D_2 + D_3$

$P_2 = D_1 + D_2 + D_4$

$P_3 = D_1 + D_3 + D_4$

(a) Write down the generator matrix, G, for C.

(b) Write down the parity check matrix, H, for C.

(c) If Neo receives 1000010 and does maximum-likelihood decoding on it, what would his estimate of the data transmission $D_1D_2D_3D_4$ from Morpheus be? For your convenience, the syndrome s_i corresponding to data bit D_i being wrong are given below, for i = 1, 2, 3, 4:

$$s_1 = (111)^T, s_2 = (110)^T, s_3 = (101)^T, s_4 = (011)^T.$$

(d) If Neo uses syndrome decoding for error correction, how many syndromes does he need to compute and store for this code, including the syndrome with no errors?

*Study Session 3: L*inear block codes, convolutional coding, Viterbi decoding of convolutional codes.

3.1 Introduction to Noise

Noise or Error is the main problem in the signal, which disturbs the reliability of the communication system. Error control coding is the coding procedure done to control the occurrences of errors. These techniques help in Error Detection and Error Correction. There are many different error correcting codes depending upon the mathematical principles applied to them. But, historically, these codes have been classified into Linear block codes and Convolution codes.

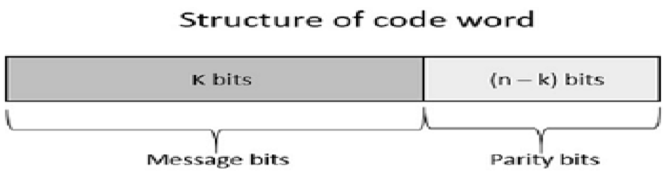
3.1.1 Linear Block Codes

In the linear block codes, the parity bits and message bits have a linear combination, which means that the resultant code word is the linear combination of any two code words. Let us consider some blocks of data, which contains k bits in each block. These bits are mapped with the blocks which has n bits in each block. Here n is greater than k. The transmitter adds redundant bits which are (n-k) bits. The ratio k/n is the **code rate**. It is denoted by **r** and the value **of r is r < 1**.

The (n-k) bits added here, are parity bits. Parity bits help in error detection and error correction, and also in locating the data. In the data being transmitted, the left most bits of the code word correspond to the message bits, and the right most bits of the code word correspond to the parity bits.

3.1.2 Systematic Code

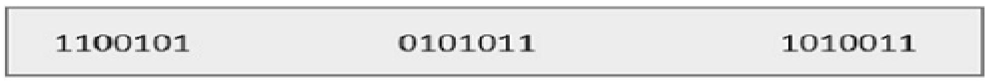
Any linear block code can be a systematic code, until it is altered. Hence, an unaltered block code is called as a systematic code. Following is the representation of the structure of code word, according to their allocation.



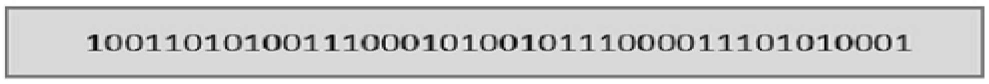
3.1.3 Convolution Codes

So far, in the linear codes, we have discussed that systematic unaltered code is preferred. Here, the data of total n bits if transmitted, k bits are message bits and (n-k) bits are parity bits. In the process of encoding, the parity bits are subtracted from the whole data and the message bits are encoded. Now, the parity bits are again added and the whole data is again encoded.

The following figure quotes an example for blocks of data and stream of data, used for transmission of information.



Example for blocks of data



Example for stream of data

The whole process, stated above is tedious which has drawbacks. The allotment of buffer is a main problem here, when the system is busy. This drawback is cleared in convolution codes. Where the whole stream of data is assigned symbols and then transmitted. As the data is a stream of bits, there is no need of buffer for storage.

3.1.4 Hamming Codes

The linearity property of the code word is that the sum of two code words is also a code word.

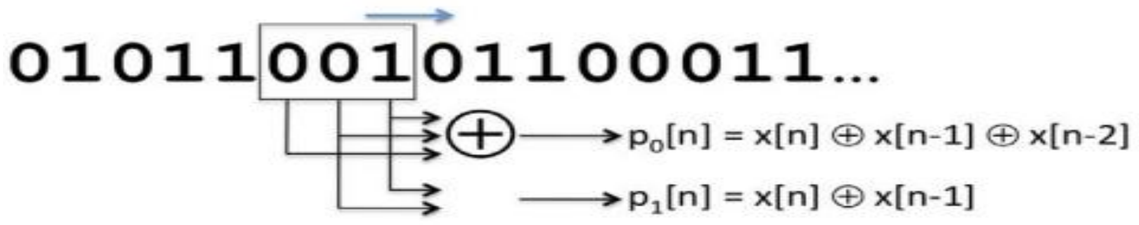
Hamming codes are the type of linear error correcting codes, which can detect up to two-bit errors or they can correct one-bit errors without the detection of uncorrected errors. While using the hamming codes, extra parity bits are used to identify a single bit error. To get from one-bit pattern to the other, few bits are to be changed in the data. Such number of bits can be termed **as Hamming distance**. If the parity has a distance of 2, one-bit flip can be detected. But this can't be corrected. Also, any two-bit flips cannot be detected. However, Hamming code is a better procedure than the previously discussed ones in error detection and correction.

3.2 Convolutional Codes: Construction and Encoding

This is the introduction of a widely used class of codes, called **convolutional codes,** which are used in a variety of systems including today's popular wireless standards (such as 802.11) and in satellite communications. They are also used as a building block in more powerful modern codes, such as turbo codes, which are used in wide-area cellular wireless network standards such as 3G, LTE, and 4G. Convolutional codes are beautiful because they are intuitive, one can understand them in many different ways, and there is a way to decode them so as to recover the most likely message from among the set of all possible transmitted messages. Like the block codes, convolutional codes involve the computation of parity bits from message bits and their transmission, and they are also linear codes. Unlike block codes in systematic form, however, the sender does not send the message bits followed by (or interspersed with) the parity bits; in a convolutional code, the sender sends only the parity bits. These codes were invented by Peter Elias '44, an MIT EECS faculty member, in the mid-1950s. For several years, it was not known just how powerful these codes are and how best to decode them. The answers to these questions started emerging in the 1960s, with the work of people like John Wozencraft (Sc.D. '57 and former MIT EECS professor), Robert Fano ('41, Sc.D. '47, MIT EECS professor), Andrew Viterbi '57, G. David Forney (SM '65, Sc.D. '67, and MIT EECS professor), Jim Omura SB '63, and many others.

3.2.1 Convolutional Code Construction

The encoder uses a sliding window to calculate r > 1 parity bits by combining various subsets of bits in the window. Unlike a block code, however, the windows overlap and slide by 1, as shown in Figure. The size of the window, in bits, is called the code's **constraint length.** The longer the constraint length, the larger the number of parity bits that are influenced by any given message bit.



01011001011000011...
$p_0[n] = x[n] \oplus x[n-1] \oplus x[n-2]$
$p_1[n] = x[n] \oplus x[n-1]$

1: An example of a convolutional code with two parity bits per message bit and a constraint length n the rectangular window) of three. I.e., $r = 2, K = 3$.

Because the parity bits are the only bits sent over the channel, a larger constraint length generally implies a greater resilience to bit errors. The trade-off, though, is that it will take considerably longer to decode codes of long constraint length, so one cannot increase the constraint length arbitrarily and expects fast decoding. If a convolutional code produces r parity bits per window and slides the window forward by one bit at a time, its rate (when calculated over long messages) is 1/r. The greater the value of r, the higher the resilience of bit errors, but the trade-off is that a proportionally higher amount of communication bandwidth is devoted to coding overhead. In practice, we would like to pick r and the constraint length to be as small as possible while providing a low enough resulting probability of a bit error.

We will use K (upper case) to refer to the constraint length, a somewhat unfortunate choice because we have used k (lower case) to refer to the number of message bits that get encoded to produce coded bits. Although "L" might be a better way to refer to the constraint length, we'll use K because many papers and documents in the field use K (in fact, many papers use k in lower case, which is especially confusing). Because we will rarely refer to a "block" of size k while talking about convolutional codes, we hope that this notation won't cause confusion.

Armed with this notation, we can describe the encoding process succinctly. The encoder looks at K bits at a time and produces r parity bits according to carefully chosen functions that operate over various subsets of the K bits. One example is shown in Figure above, which shows a scheme with K = 3 and r = 2 (the rate of this code, 1/r = 1/2). The encoder splits out r bits, which are sent sequentially, slides the window by 1 to the right, and then repeats the process. That's essentially it.

At the transmitter, the two principal remaining details that we must describe are:

1. What are good parity functions and how can we represent them conveniently?

2. How can we implement the encoder efficiently?

The rest of this chapter will discuss these issues, and also explain why these codes are called "convolutional".

3.3 Parity Equations

The example in Figure above shows one example of a set of parity equations, which govern the way in which parity bits are produced from the sequence of message bits, X. In this example, the equations are as follows:

$$p0[n] = x[n] + x[n-1] + x[n-2]$$
$$p1[n] = x[n] + x[n-1]$$

The rate of this code is 1/2.

An example of parity equations for a rate 1/3 code is

$$p0[n] = x[n] + x[n-1] + x[n-2]$$
$$p1[n] = x[n] + x[n-1]$$
$$p2[n] = x[n] + x[n-2]$$

In general, one can view each parity equation as being produced by combining the message bits, X, and a generator polynomial, g. In the first example above, the generator polynomial coefficients are (1, 1, 1) and (1, 1, 0), while in the second, they are (1, 1, 1), (1, 1, 0), and (1, 0, 1).

We denote by g_i the K-element generator polynomial for parity bit p_i. We can then write $p_i[n]$ as follows:

$$p_i[n] = (\sum_{j=0}^{k-1} g_i[j]x[n-j]).$$

The form of the above equation is a convolution of g and x (modulo 2)—hence the term "convolutional code". The number of generator polynomials is equal to the number of generated parity bits, r, in each sliding window. The rate of the code is 1/r if the encoder slides the window one bit at a time.

An Example

Let's consider the two generator polynomials. Here, the generator polynomials are:

g_0 = 1, 1, 1

g_1 = 1, 1, 0

If the message sequence, X = [1, 0, 1, 1,...] (as usual, x[n]=0 For all n < 0), then the parity

$p_0[0] = (1 + 0 + 0) = 1$

$p_1[0] = (1 + 0) = 1$

$p_0[1] = (0 + 1 + 0) = 1$

$p_1[1] = (0 + 1) = 1$

$p_0[2] = (1 + 0 + 1) = 0$

$p_1[2] = (1 + 0) = 1$

$p_0[3] = (1 + 1 + 0) = 0$

$p_1[3] = (1 + 1) = 0$.

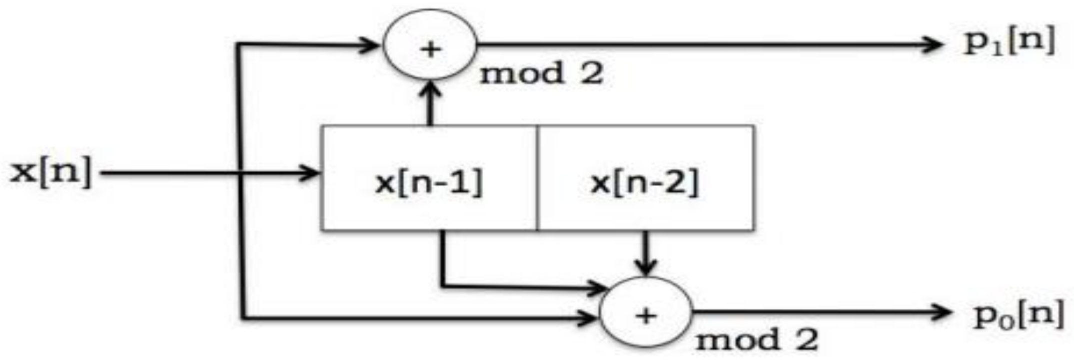
Therefore, the bits transmitted over the channel are [1, 1, 1, 1, 0, 0, 0, 0,...]. There are several generator polynomials, but understanding how to construct good ones is outside the scope Some examples, found originally by J. Bussgang are

Constraint length	g_0	g_1
3	110	111
4	1101	1110
5	11010	11101
6	110101	111011
7	110101	110101
8	110111	1110011
9	110111	111001101
10	110111001	1110011001

: Examples of generator polynomials for rate 1/2 convolutional codes with different constraint

3.4 Two Views of the Convolutional Encoder

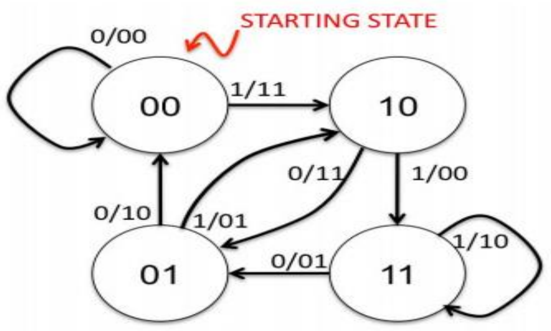
We now describe two views of the convolutional encoder, which we will find useful in better understanding convolutional codes and in implementing the encoding and decoding procedures. The first view is in terms of shift registers, where one can construct the mechanism using shift registers that are connected together. This view is useful in developing hardware encoders. The second is in terms of a state machine, which corresponds to a view of the encoder as a set of states with well-defined transitions between them. The state machine view will turn out to be extremely useful in figuring out how to decode a set of parity bits to reconstruct the original message bits.



3.5 Shift-Register View

Figure above shows the same encoder as in Equations above in the form of a block diagram made up of shift registers. The *x[n - i]* values (here there are two) are referred to as the state of the encoder. This block diagram takes message bits in one bit at a time, and spits out parity bits (two per input bit, in this case). Input message bits, x[n], arrive from the left. The block diagram calculates the parity bits using the incoming bits and the state of the encoder (the k - 1 previous bits; two in this example). After the r parity bits are produced, the state of the encoder shifts by 1, with x[n] taking the place of x[n-1], x[n-1] taking the place of x[n-2], and so on, with x[n- K +1] being discarded. This block diagram is directly amenable to a hardware implementation using shift registers.

3.6 State-Machine View



Another useful view of convolutional codes is as a state machine, which is shown in Figure for the same example that we have used throughout this chapter. An important point to note: the state machine for a convolutional code is identical for all codes with a given constraint length, K, and the number of states is always 2K-1. Only the p_i labels change depending on the number of generator polynomials and the values of their coefficients. Each state is labeled with x[n - 1]x[n - 2] ...x[n - K + 1]. Each arc is labeled with x[n]/p_0p_1 .... In this example, if the message is 101100, the transmitted bits are 11 11 01 00 01 10.

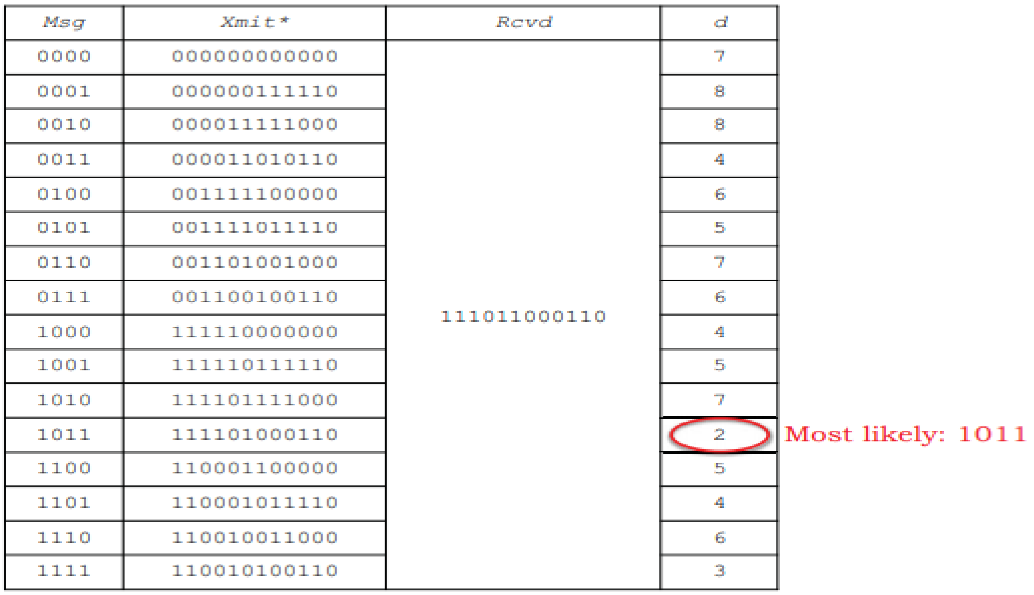
This state-machine view is an elegant way to explain what the transmitter does, and also what the receiver ought to do to decode the message, as we now explain. The transmitter begins in the initial state (labeled “STARTING STATE”) and processes the message one bit at a time. For each message bit, it makes the state transition from the current state to the new one depending on the value of the input bit, and sends the parity bits that are on the corresponding arc.

The receiver, of course, does not have direct knowledge of the transmitter’s state transitions. It only sees the received sequence of parity bits, with possible bit errors. Its task is to determine the best possible sequence of transmitter states that could have produced the parity bit sequence. This task is the essence of the decoding process, which we introduce next, and study

3.7 The Decoding Problem

As mentioned above, the receiver should determine the “best possible” sequence of transmitter states. There are many ways of defining “best”, but one that is especially appealing is the most likely sequence of states (i.e., message bits) that must have been traversed (sent) by the transmitter. A decoder that is able to infer the most likely sequence the maximum-likelihood (ML) decoder for the convolutional code. We previously established that the ML decoder for “hard decoding”, in which the distance between the received word and each valid codeword is the Hamming distance, may be found by computing the valid codeword with smallest Hamming distance, and returning the message that would have generated that codeword. The same idea holds for convolutional codes. (Note that this property holds whether the code is either block or convolutional, and whether it is linear or not.).

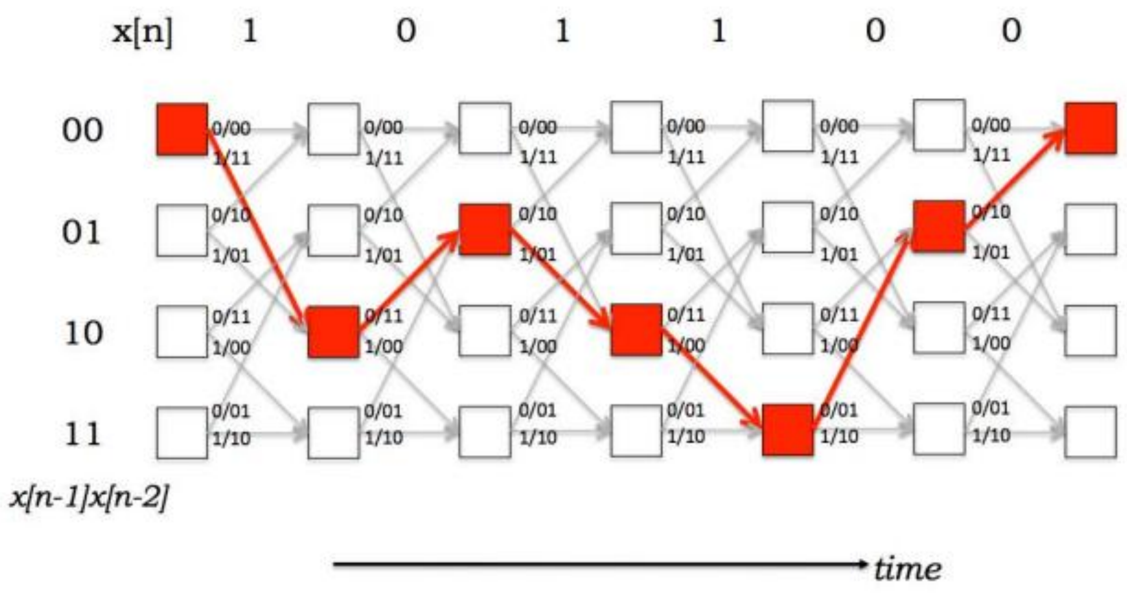
A simple numerical example may be useful. Suppose that bit errors are independent and identically distributed with an error probability of 0.001 (i.e., the channel is a BSC with " = 0.001), and that the receiver digitizes a sequence of analog samples into the bits 1101001. Is the sender more likely to have sent 1100111 or 1100001? The first has a Hamming distance of 3, and the probability of receiving that sequence is (0.999)4(0.001)3 = 9.9 ⇥ 10-10. The second choice has a Hamming distance of 1 and a probability of (0.999)6(0.001)1 = 9.9 ⇥ 10-4, which is six orders of magnitude higher and is overwhelmingly more likely. Thus, the most likely sequence of parity bits that was transmitted must be the one with



Msg	Xmit*	Rcvd	d
0000	000000000000	111011000110	7
0001	000000111110		8
0010	000011111000		8
0011	000011010110		4
0100	001111100000		6
0101	001111011110		5
0110	001101001000		7
0111	001100100110		6
1000	111110000000		4
1001	111110111110		5
1010	111101111000		7
1011	111101000110		2
1100	110001100000		5
1101	110001011110		4
1110	110010011000		6
1111	110010100110		3

Most likely: 1011

When the probability of bit error is less than 1/2, maximum-likelihood decoding boils down to finding the message whose parity bit sequence, when transmitted, has the smallest Hamming distance to the received sequence. Ties may be broken arbitrarily. Unfortunately, for an N-bit transmit sequence, there are 2N possibilities, which makes it hugely intractable to simply go through in sequence because of the sheer number. For instance, when N = 256 bits (a really small packet), the number of possibilities rivals the number of atoms in the universe the smallest Hamming distance from the sequence of parity bits received. Given a choice of possible transmitted messages, the decoder should pick the one with the smallest such Hamming distance. For example, which shows a convolutional code with K = 3 and rate 1/2. If the receiver gets 111011000110, then some errors have occurred, because no valid transmitted sequence matches the received one. The last column in the example shows d, the Hamming distance to all the possible transmitted sequences, with the smallest one circled. To determine the most-likely 4-bit message that led to the parity sequence received, the receiver could look for the message whose transmitted parity bits have smallest Hamming distance from the received bits. (If there are ties for the smallest, we can break them arbitrarily, because all these possibilities have the same resulting post coded BER.) Determining the nearest valid codeword to a received word is easier said than done for convolutional codes. For block codes, we found that comparing against each valid codeword would take time exponential in k, the number of valid codewords for an (n, k) block



code. We then showed how syndrome decoding takes advantage of the linearity property to devise an efficient polynomial-time decoder for block codes, whose time complexity was roughly O (nt), where t is the number of errors that the linear block code can correct. For convolutional codes, syndrome decoding in the form we described is impossible because n is infinite (or at least as long as the number of parity streams times the length of the entire message times, which could be arbitrarily long)! The straightforward approach of simply going through the list of possible transmit sequences and comparing Hamming distances is horribly intractable. We need a better plan for the receiver to navigate this unbelievable large space of possibilities and quickly determine the valid message with smallest Hamming distance. We will study a powerful and widely applicable method for solving this problem, called Viterbi decoding, in the next chapter. This decoding method uses a special structure called the trellis.

3.7.1 The Trellis

The trellis is a structure derived from the state machine that will allow us to develop an efficient way to decode convolutional codes. The state machine view shows what happens at each instant when the sender has a message bit to process, but doesn't show how the system evolves in time. The trellis is a structure that makes the time evolution explicit. An example is shown in Figure. Each column of the trellis has the set of states; each state in a column is connected to two states in the next column—the same two states in the state diagram. The top link from each state in a column of the trellis shows what gets transmitted on a "0", while the bottom shows what gets transmitted on a "1". The picture shows the links between states that are traversed in the trellis given the message 101100. We can now think about what the decoder needs to do in terms of this trellis. It gets a sequence of parity bits, and needs to determine the best path through the trellis—that is, the sequence of states in the trellis that can explain the observed, and possibly corrupted, sequence of received parity bits. The Viterbi decoder finds a maximum-likelihood path through the trellis.

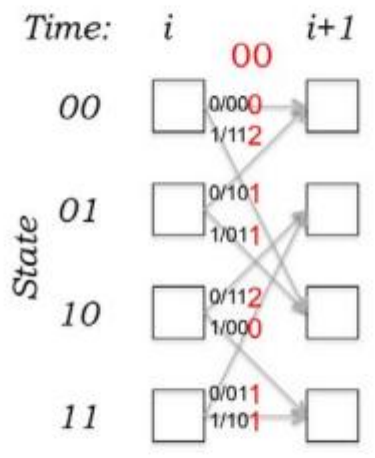
3.8 Viterbi Decoding of Convolutional Codes

This chapter describes an elegant and efficient method to decode convolutional codes, whose construction and encoding we described in the previous chapter. This decoding method avoids explicitly enumerating the 2N possible combinations of N-bit parity bit sequences. This method was invented by Andrew Viterbi '57 and bears his name. The Problem At the receiver, we have a sequence of voltage samples corresponding to the parity bits that the transmitter has sent. For simplicity, and without loss of generality, we will assume that the receiver picks a suitable sample for the bit, or averages the set of samples corresponding to a bit, digitizes that value to a "0" or "1" by comparing to the threshold voltage (the de-mapping step), and propagates that bit decision to the decoder. Thus, we have a received bit sequence, which for a convolutionally-coded stream corresponds to the stream of parity bits. If we decode this received bit sequence with no other information from the receiver's sampling and de-mapper, then the decoding process is termed hard-decision decoding ("hard decoding"). If, instead (or in addition), the decoder is given the stream of voltage samples and uses that "analog" information (in digitized form, using an analog-to-digital conversion) in decoding the data, we term the process soft-decision decoding ("soft decoding"). The Viterbi decoder can be used in either case. Intuitively, because hard-decision decoding makes an early decision regarding whether a bit is 0 or 1, it throws away information in the digitizing process. It might make a wrong decision, especially for voltages near the threshold, introducing a greater number of bit errors in the received bit sequence.

Although it still produces the most likely transmitted sequence given the received bit sequence, by introducing additional errors in the early digitization, the overall reduction in the probability of bit error will be smaller than with soft decision decoding. But it is conceptually easier to understand hard decoding, so we will start with that, before going on to soft decoding. As mentioned in the previous chapter, the trellis provides a good framework for understanding the decoding procedure for convolutional codes (Figure 8-1). Suppose we have the entire trellis in front of us for a code, and now receive a sequence of digitized bits (or voltage samples). If there are no errors, then there will be some path through the states of the trellis that would exactly match the received sequence. That path (specifically, the concatenation of the parity bits "spit out" on the traversed edges) corresponds to the transmitted parity bits. From there, getting to the original encoded message is easy because the top arc emanating from each node in the trellis corresponds to a "0" bit and the bottom arrow corresponds to a "1" bit.

When there are bit errors, what can we do? As explained earlier, finding the most likely transmitted message sequence is appealing because it minimizes the probability of a bit error in the decoding. If we can come up with a way to capture the errors introduced by going from one state to the next, then we can accumulate those errors along a path and come up with an estimate of the total number of errors along the path. Then, the path with the smallest such accumulation of errors is the path we want, and the transmitted message sequence can be easily determined by the concatenation of states explained above.

To solve this problem, we need a way to capture any errors that occur in going through the states of the trellis, and a way to navigate the trellis without actually materializing the entire trellis (i.e., without enumerating all possible paths through it and then finding the one with smallest accumulated error). The Viterbi decoder solves these problems. It is an example of a more general approach to solving optimization problems, called dynamic programming. Later in the course, we will apply similar concepts in network routing, an unrelated problem, to find good paths in multi-hop networks.



The branch metric for hard decision decoding. In this example, the receiver gets the parity bits

3.9 The Viterbi Decoder

The decoding algorithm uses two metrics: the branch metric (BM) and the path metric (PM). The branch metric is a measure of the "distance" between what was transmitted and what was received, and is defined for each arc in the trellis. In hard decision decoding, where we are given a sequence of digitized parity bits, the branch metric is the Hamming distance between the expected parity bits and the received ones. An example is shown in Figure, where the received bits are 00. For each state transition, the number on the arc shows the branch metric for that transition. Two of the branch metrics are 0, corresponding to the only states and transitions where the corresponding Hamming distance is 0. The other non-zero branch metrics correspond to cases when there are bit errors.

The path metric is a value associated with a state in the trellis (i.e., a value associated with each node). For hard decision decoding, it corresponds to the Hamming distance with respect to the received parity bit sequence over the most likely path from the initial state to the current state in the trellis. By "most likely", we mean the path with smallest Hamming distance between the initial state and the current state, measured over all possible paths between the two states. The path with the smallest Hamming distance minimizes the total number of bit errors, and is most likely when the BER is low.

The key insight in the Viterbi algorithm is that the receiver can compute the path metric for a (state, time) pair incrementally using the path metrics of previously computed states and the branch metrics.

3.9.1 Computing the Path Metric

Suppose the receiver has computed the path metric PM[s,i] for each states at time step i (recall that there are 2K-1 states, where K is the constraint length of the convolutional code). In hard decision decoding, the value of PM[s,i] is the total number of bit errors detected when comparing the received parity bits to the most likely transmitted message, considering all messages that could have been sent by the transmitter until time step I (starting from state “00”, which we will take to be the starting state always, by convention).

Among all the possible states at time step i, the most likely state is the one with the smallest path metric. If there is more than one such state, they are all equally good possibilities. Now, how do we determine the path metric at time step i + 1, PM[s,i + 1], for each state s? To answer this question, first observe that if the transmitter is at state s at time step i+ 1, then it must have been in only one of two possible states at time step i. These two predecessor states, labeled ↵ and (, are always the same for a given state. In fact, they depend only on the constraint length of the code and not on the parity functions. The predecessor states for each state (the other end of each arrow). For instance, for state 00, ↵ = 00 and (= 01; for state 01, ↵ = 10 and (= 11. Any message sequence that leaves the transmitter in states at time i + 1 must have left the transmitter in state ↵ or state (at time i. For example, to arrive in state ’01’ at time i + 1, one of the following two properties must hold:

1. The transmitter was in state ‘10’ at time i and the ith message bit was a 0. If that is the case, then the transmitter sent ‘11’ as the parity bits and there were two bit errors, because we received the bits 00. Then, the path metric of the new state, PM[‘01’, i+ 1] is equal to PM[‘10’, i] + 2, because the new state is ‘01’ and the corresponding path metric is larger by 2 because there are 2 errors.

2. The other (mutually exclusive) possibility is that the transmitter was in state ‘11’ at time i

and the ith message bit was a 0. If that is the case, then the transmitter sent 01 as the parity bits and there was one bit error, because we received 00. The path metric of the new state, PM['01', i + 1] is equal to PM['11', i] + 1.

Formalizing the above intuition, we can see that

PM[s,i + 1] = min(PM[↲, i] + BM[↲ ! s],PM[(, i] + BM[(! s]), (8.1)
where ↲ and (are the two predecessor states.

In the decoding algorithm, it is important to remember which arc corresponds to the minimum, because we need to traverse this path from the final state to the initial one keeping track of the arcs we used, and then finally reverse the order of the bits to produce the most likely message.

3.9.2 Finding the Most Likely Path

We can now describe how the decoder finds the maximum-likelihood path. Initially, state '00' has a cost of 0 and the other 2_K-1 - 1 states have a cost of 1. The main loop of the algorithm consists of two main steps: first, calculating the branch metric for the next set of parity bits, and second, computing the path metric for the next column. The path metric computation may be thought of as an add-compare-select procedure:

1. Add the branch metric to the path metric for the old state.
2. Compare the sums for paths arriving at the new state (there are only two such paths to compare at each new state because there are only two incoming arcs from the previous column).
3. Select the path with the smallest value, breaking ties arbitrarily. This path corresponds to the one with fewest errors.

Figure shows the decoding algorithm in action from one-time step to the next. This example shows a received bit sequence of 11 10 11 00 01 10 and how the receiver processes it. The fourth picture from the top shows all four states with the same path metric. At this stage, any of these four states and the paths leading up to them are most likely transmitted bit sequences (they all have a Hamming distance of 2). The bottom-most picture shows the same situation with only the survivor paths shown. A survivor path is one that has a chance of being the maximum-likelihood path; there are many other paths that can be pruned away because there is no way in which they can be most likely. The reason why the Viterbi decoder is practical is that the number of survivor paths is much, much smaller than the total number of paths in the trellis.

Another important point about the Viterbi decoder is that future knowledge will help it break any ties, and in fact may even cause paths that were considered "most likely" at a certain time step to change. proceeding until all the received parity bits are decoded to produce the most likely transmitted message, which has two-bit errors.

3.9.3 Soft-Decision Decoding

Hard decision decoding digitizes the received voltage signals by comparing it to a threshold, before passing it to the decoder. As a result, we lose information: if the voltage was 0.500001, the confidence in the digitization is surely much lower than if the voltage was 0.999999. Both are treated as "1", and the decoder now treats them the same way, even though it is overwhelmingly more likely that 0.999999 is a "1" compared to the other value. Soft-decision decoding (also sometimes known as "soft input Viterbi decoding") builds on this observation. It does not digitize the incoming samples prior to decoding. Rather, it uses a continuous function of the analog sample as the input to the decoder. For example, if the expected parity bit is 0 and the received voltage is 0.3 V, we might use 0.3 (or 0.32, or some such function) as the value of the "bit" instead of digitizing it. For technical reasons that will become apparent later, an attractive soft decision metric is the square of the difference between the received voltage and the expected one. If the convolutional code produces p parity bits, and the p corresponding analog samples are $v = v_1, v_2,...,v_p$, one can construct a soft decision branch metric as follows

$$\text{BM}_{\text{soft}}[u, v] = \sum_{i=1}^{p} (u_i - v_i)^2,$$

where u = u_1, u_2,...,up are the expected p parity bits (each a 0 or 1). The soft decision branch metric for p = 2 when u is 00. With soft decision decoding, the decoding algorithm is identical to the one previously described for hard decision decoding, except that the branch metric is no longer an integer Hamming distance but a positive real number (if the voltages are all between 0 and 1, then the branch metric is between 0 and 1 as well).

It turns out that this soft decision metric is closely related to the probability of the decoding being correct when the channel experiences additive Gaussian noise. First, let's look at the simple case of 1 parity bit (the more general case is a straightforward extension). Suppose the receiver gets the ith parity bit as vi volts. (In hard decision decoding, it would decode - as 0 or 1 depending on whether vi was smaller or larger than 0.5.) What is the probability that vi would have been received given that bit ui (either 0 or 1) was sent? With zero-mean additive Gaussian noise, the PDF of this event is given by

$$f(v_i|u_i) = \frac{e^{-d_i^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}},$$

where $d_i = v_i^2$ if $u_i = 0$ and $d_i = (v_i - 1)^2$ if $u_i = 1$.

Minimizing this path metric is identical to maximizing the log likelihood along the different paths, implying that the soft decision decoder produces the most likely path that is consistent with the received voltage sequence. This direct relationship with the logarithm of the probability is the reason why we chose the sum of squares as the branch metric in equation. A different noise distribution (other than Gaussian) may entail a different soft decoding branch metric to obtain an analogous connection to the PDF of a correct decoding.

Summary of Study 3

From its relatively modest, though hugely impactful, beginnings as a method to decode convolutional codes, Viterbi decoding has become one of the most widely used algorithms in a wide range of fields and engineering systems. Modern disk drives with "PRML" technology to speed-up accesses, speech recognition systems, natural language systems, and a variety of communication networks use this scheme or its variants.

In fact, a more modern view of the soft decision decoding technique described in this lecture is to think of the procedure as finding the most likely set of traversed states in a Hidden Markov Model (HMM). Some underlying phenomenon is modeled as a Markov state machine with probabilistic transitions between its states; we see noisy observations from each state, and would like to piece together the observations to determine the most likely sequence of states traversed. It turns out that the Viterbi decoder is an excellent starting point to solve this class of problems (and sometimes the complete solution). On the other hand, despite its undeniable success, Viterbi decoding isn't the only way to decode convolutional codes. For one thing, its computational complexity is exponential in the constraint length, K, because it does require each of these states to be enumerated. When K is large, one may use other decoding methods such as BCJR or Fano's sequential decoding scheme, for instance. Convolutional codes themselves are very popular over both wired and wireless links. They are sometimes used as the "inner code" with an outer block error correcting code, but they may also be used with just an outer error detection code. They are also used as a component in more powerful codes like turbo codes, which are currently one of the highest-performing codes used in practice.

Self-Assessment Question (SAQs) for lecture 3

SAQ 3.1

1. Consider a convolutional code whose parity equations are

$p_0[n] = x[n] + x[n - 1] + x[n - 3]$

$p_1[n] = x[n] + x[n - 1] + x[n - 2]$

$p_2[n] = x[n] + x[n - 2] + x[n - 3]$

(a) What is the rate of this code? How many states are in the state machine representation of this code?

(b) Suppose the decoder reaches the state "110" during the forward pass of the Viterbi algorithm with this convolutional code.

i. How many predecessor states (i.e., immediately preceding states) does state “110” have?

ii. What are the bit-sequence representations of the predecessor states of state “110”?

iii. What are the expected parity bits for the transitions from each of these predecessor states to state “110”? Specify each predecessor state and the expected parity bits associated with the corresponding transition below.

(c) To increase the rate of the given code, Lem E. Tweakit punctures the p_0 parity stream using the vector (1 0 1 1 0), which means that every second and fifth bit produced on the stream are not sent. In addition, she punctures the p_1 parity stream using the vector (1 1 0 1 1). She sends the p_2 parity stream unchanged. What is the rate of the punctured code?

SAQ 3.2

2. Let conv encode(x) be the resulting bit-stream after encoding bit-string x with a convolutional code, C. Similarly, let conv decode(y) be the result of decoding y to produce the maximum-likelihood estimate of the encoded message. Suppose we send a message M using code C over some channel. Let P = conv encode (M) and let R be the result of sending P over the channel and digitizing the received samples at the receiver (i.e., R is another bit-stream). Suppose we use Viterbi decoding on R, knowing C, and find that the maximum-likelihood estimate of M is Mˆ . During the decoding, we find that the minimum path metric among all the states in the final

stage of the trellis is D_{min}.

D_{min} is the Hamming distance between………………………. and………………………… . Fill in the blanks, explaining your answer.

SAQ 3.3

Consider the trellis in Figure 8-9 showing the operation of the Viterbi algorithm using a hard branch metric at the receiver as it processes a message encoded with a convolutional code, C. Most of the path metrics have been filled in for each state at each time and the predecessor states determined by the Viterbi algorithm are shown by a solid transition arrow.

(a) What is the code rate of C?

(b) What is the constraint length of C?

(c) What bits would be transmitted if the message 1011 were encoded using C?

Note this is not the message being decoding in the example above.

(d) The received parity bits for time 5 are missing from the trellis diagram. What values for the parity bits are consistent with the other information in the trellis? Note that there may be more than one set of such values.

(f) In the trellis diagram shown , circle the states along the most-likely path through the trellis. Determine the decoded message that corresponds to that most-likely path.

(g) Based on your answer to the previous part, how many bit errors were detected in the received transmission and at what time(s) did those error(s) occur?

Study Session 4

Introduction to Noise in Communication

Noise is a variation of voltage which has entered the communication system and which is undesired or not required. The source generating the noise may be located inside or outside the communication system.

Noise can enter the communication system at any stage such as:

- Transmission
- Reception or
- When the data is moving through the channel.

A transmitter combines the incoming message signal i.e. low frequency with carrier signal i.e. high frequency, so as to make it suitable for transmission through a channel and subsequent reception. The combination of message signal and carrier signal is given amplification, it is filtered for frequencies, noise, etc. and then it is transmitted on wired or wireless system. When the carrier signal of high frequency containing the message signal travel through the channel and reach the receiver, then receiver performs the following tasks −

Detect required frequency

Detection of frequency is done by resonance and this process is **known tuning**

Amplify

As the incoming signal is weak, receiver gives strength to the signal and this process is called amplification.

Filter

Filter for undesired noise in the signal received.

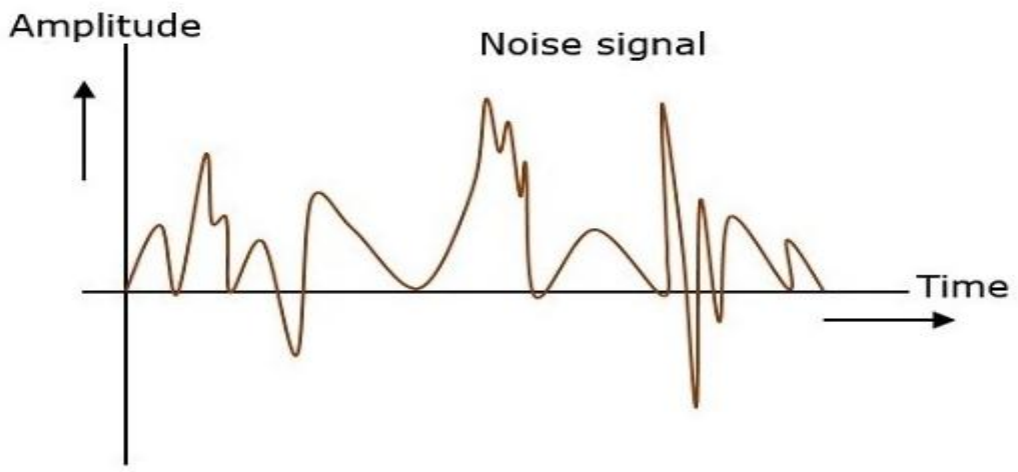
Demodulation

Demodulation is the process of separating carrier wave (high frequency) and carry low frequency message signal.

In any communication system, during the transmission of the signal or while receiving the signal, some unwanted signal gets introduced into the communication, making it unpleasant for the receiver, and questioning the quality of the communication. Such a disturbance is called **as Noise**.

4.1 What is Noise?

Noise is an unwanted signal, which interferes with the original message signal and corrupts the parameters of the message signal. This alteration in the communication process, leads to the message getting altered. It most likely enters at the channel or the receiver. The noise signal can be understood by taking a look at the following figure.



Hence, it is understood that the noise is some signal which has no pattern and no constant frequency or amplitude. It is quite random and unpredictable. Measures are usually taken to reduce it, though it can’t be completely eliminated. Most common examples of noise are :

- Hiss sound in radio receivers
- Buzz sound amidst of telephone conversations
- Flicker in television receivers, etc

4.1.1 Types of Noise

The classification of noise is done depending on the type of the source, the effect it shows or the relation it has with the receiver, etc. There are two main ways in which noise is produced. One is through some external source while the other is created by an internal source, within the receiver section.

4.1.2 External Source

This noise is produced by the external sources, which may occur in the medium or channel of communication usually. This noise cannot be completely eliminated. The best way is to avoid the noise from affecting the signal. Most common examples of this type of noise are:

- Atmospheric noise (due to irregularities in the atmosphere).
- Extra-terrestrial noise, such as solar noise and cosmic noise.
- Industrial noise.

4.1.3 Internal Source

This noise is produced by the receiver components while functioning. The components in the circuits, due to continuous functioning, may produce few types of noise. This noise is quantifiable. A proper receiver design may lower the effect of this internal noise. Most common examples of this type of noise are:

- Thermal agitation noise (Johnson noise or Electrical noise)
- Shot noise (due to the random movement of electrons and holes)
- Transit-time noise (during transition)

4.1.4 Miscellaneous noise is another type of noise which includes flicker, resistance effect and mixer generated noise, etc.

4.1.5 Effects of Noise

Noise is an inconvenient feature, which affects the system performance. Following are the effects of noise.

Noise limits the operating range of the systems

Noise indirectly places a limit on the weakest signal that can be amplified by an amplifier. The oscillator in the mixer circuit may limit its frequency because of noise. A system's operation depends on the operation of its circuits. Noise limits the smallest signal that a receiver is capable of processing.

4.1.6 Noise affects the sensitivity of receivers

Sensitivity is the minimum amount of input signal necessary to obtain the specified quality output. Noise affects the sensitivity of a receiver system, which eventually affects the output.

4.2 Signal to Noise Ratio

Calculate Signal to Noise Ratios and Figure of Merits of various modulated waves, which are demodulated at the receiver. Signal-to-Noise Ratio (SNR) is the ratio of the signal power to noise power. The higher the value of SNR, the greater will be the quality of the received output. Signal-to-Noise Ratio at different points can be calculated using the following formulas.

$$\textbf{Input SNR} = (SNR)_I = \frac{Average\ power\ of\ modulating\ signal}{Average\ power\ of\ noise\ at\ input}$$

$$\textbf{Output SNR} = (SNR)_O = \frac{Average\ power\ of\ demodulated\ signal}{Average\ power\ of\ noise\ at\ output}$$

$$\textbf{Channel SNR} = (SNR)_C = \frac{Average\ power\ of\ modulated\ signal}{Average\ power\ of\ noise\ in\ message\ bandwidth}$$

Figure of Merit

The ratio of output SNR and input SNR can be termed as **Figure of Merit**. It is denoted by F. It describes the performance of a device.

$$F = \frac{(SNR)_O}{(SNR)_I}$$

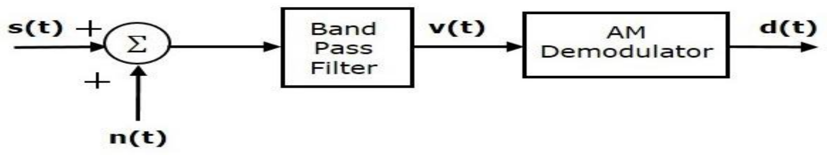
Figure of merit of a receiver is

$$F = \frac{(SNR)_O}{(SNR)_C}$$

It is so because for a receiver, the channel is the input.

4.2.1 SNR Calculations in AM System

Consider the following receiver model of AM system to analyze noise.



We know that the Amplitude Modulated (AM) wave is

$$s(t) = A_c\left[1 + k_a m(t)\right]\cos(2\pi f_c t)$$

$$\Rightarrow s(t) = A_c\cos(2\pi f_c t) + A_c k_a m(t)\cos(2\pi f_c t)$$

Average power of AM wave is

$$P_s = \left(\frac{A_c}{\sqrt{2}}\right)^2 + \left(\frac{A_c k_a m(t)}{\sqrt{2}}\right)^2 = \frac{A_c{}^2}{2} + \frac{A_c{}^2 k_a{}^2 P}{2}$$

$$\Rightarrow P_s = \frac{A_c{}^2\left(1 + k_a{}^2 P\right)}{2}$$

Average power of noise in the message bandwidth is $P_{nc} = WN_0$

Substitute, these values in channel SNR formula

$$(SNR)_{C,AM} = \frac{Average\ Power\ of\ AM\ Wave}{Average\ Power\ of\ noise\ in\ message\ bandwidth}$$

$$\Rightarrow (SNR)_{C,AM} = \frac{A_c^2\left(1+k_a^2\right)P}{2WN_0}$$

Where,

- **P** is the power of the message signal$=\frac{A_m{}^2}{2}$
- **W** is the message bandwidth

Assume the band pass noise is mixed with AM wave in the channel as shown in the above figure. This combination is applied at the input of AM demodulator. Hence, the input of AM demodulator is.

$$v(t) = s(t) + n(t)$$

$$\Rightarrow v(t) = A_c\left[1 + k_a m(t)\right]\cos(2\pi f_c t) + \left[n_1(t)\cos(2\pi f_c t) - n_Q(t)\sin(2\pi f_c t)\right]$$

$$\Rightarrow v(t) = \left[A_c + A_c k_a m(t) + n_1(t)\right]\cos(2\pi f_c t) - n_Q(t)\sin(2\pi f_c t)$$

Where $n_I(t)$ and $n_Q(t)$ are in phase and quadrature phase components of noise.

The output of AM demodulator is nothing but the envelope of the above signal.

$$d(t) = \sqrt{\left[A_c + A_c K_a m(t) + n_I(t)\right]^2 + \left(n_Q(t)\right)^2}$$

$$\Rightarrow d(t) \approx A_c + A_c k_a m(t) + n_1(t)$$

The output of AM demodulator is nothing but the envelope of the above signal.

Substitute, these values in **output SNR** formula.

$$(SNR)_{O,AM} = \frac{Average\ Power\ of\ demodulated\ signal}{Average\ Power\ of\ noise\ at\ Output}$$

$$\Rightarrow (SNR)_{O,AM} = \frac{A_c^2 k_a^2 P}{2WN_0}$$

Substitute, the values in **Figure of merit** of AM receiver formula.

$$F = \frac{(SNR)_{O,AM}}{(SNR)_{C,AM}}$$

$$\Rightarrow F = \left(\frac{A_c^2 k_a^2 P}{2WN_0}\right) / \left(\frac{A_c^2\left(1+k_a^2\right)P}{2WN_0}\right)$$

$$\Rightarrow F = \frac{K_a^2 P}{1 + K_a^2 P}$$

Therefore, the Figure of merit of AM receiver is less than one.

In general, many independent factors affect a signal received over a channel. Those that have a repeatable, deterministic effect from one transmission to another are generally referred to as distortion. Other factors have effects that are better modeled as random, and we collectively refer to them as noise. Communication systems are no exception to the general rule that any system in the physical world must contend with noise. In fact, noise is a fundamental aspect of all communication systems.

In the simplest binary signaling scheme which we will invoke for most of our purposes in this course- a communication system transmits one of two voltages, mapping a “0” to the voltage V_o and mapping a “1” to V_1. The appropriate voltage is held steady over a fixed-duration time slot that is reserved for transmission of this bit, then moved to the appropriate voltage for the bit associated with the next time slot, and so on. We assume that any distortion has been compensated for at the receiver, so that in an ideal noise-free case the receiver ends up measuring V0 in any time slot corresponding to a “0”, and V_1 in any slot corresponding to a “1”.

In this chapter we focus on the case where $V_1 = V_p > 0$ and $V_0 = -V_p$, where V_p is some fixed positive voltage, typically the peak voltage magnitude that the transmitter is capable of imposing on the communication channel. This scheme is sometimes referred to as bipolar signaling or bipolar keying. In the presence of noise, the receiver measures a sequence of voltage samples y[k]that is unlikely to be exactly V_0 or V_1. To deal with this variation, we described a simple and intuitively reasonable decision rule, for the receiver to infer whether the bit transmitted in a particular time slot was a “0” or a “1”. The receiver first chooses a single voltage sample from the sequence of received samples within the appropriate time slot, and then compares this sample to a threshold voltage V_t. Provided “0” and “1” are equally likely to occur in the sender’s binary stream, it seems reasonable that we should pick as our threshold the voltage that “splits the difference”, i.e., use $V_t = (V_0 + V_1)/2$. Then, assuming $V_0 < V_1$, return “0” as the decision if the received voltage sample is smaller than V_t, otherwise return “1”.

The receiver could also do more complicated things; for example, it could form an average or a weighted average of all the voltage samples in the appropriate time slot, and then compare this average with the threshold voltage V_t. Though such averaging leads in general to improved performance, we focus on the simpler scheme, where a single well-selected sample in the time slot is compared with V_t.

The key points of this chapter are as follows:

A simple model and often a good model for the net effect at the receiver of noise in the communication system is to assume additive, Gaussian noise. In this model, each received signal sample is the sum of two components. The first component is the deterministic function of the transmitted signal that would be obtained in the absence of noise. (Throughout this chapter, we will assume no distortion in the channel, so the deterministic function referred to here will actually produce at the receiver exactly the same sample value transmitted by the sender, under the assumption of no noise.) The second component is the noise term, and is a quantity drawn from a Gaussian probability distribution with mean 0 and some variance, independent of the transmitted signal. The Gaussian distribution is described in more detail in this chapter.

If this Gaussian noise variable is also independent from one sample to another, we describe the underlying noise process as **white Gaussian noise**, and refer to the noise as **additive white Gaussian noise (AWGN);** this is the case we will consider. The origin of the term "white" will become clearer when we examine signals in the frequency domain. The variance of the zero-mean Gaussian noise variable at any sample time for this AWGN case reflects the power or intensity of the underlying white-noise process. (By analogy with what is done with electrical circuits or mechanical systems, the term "power" is generally used for the square of a signal magnitude. In the case of a random signal, the term generally denotes the expected or mean value of the squared magnitude.)

If the sender transmitted a signal corresponding to some bit, b, and the receiver measured its voltage as being on the correct side of the threshold voltage V_t, then the bit would be received correctly. Otherwise, the result is a bit error. The probability of a bit error is an important quantity, which we will analyze. This probability, typically called the **bit error rate (BER),** is related to the probability that a Gaussian random variable exceeds some level; we will calculate it using the probability density function (PDF) and cumulative distribution function (CDF) of a Gaussian random variable. We will find that, for the bipolar keying scheme described above, when used with the simple threshold decision rule that was also specified above, the BER is determined by the ratio of two quantities: (i) the power or squared magnitude, V_p^2, of the received sample voltage in the noise-free case; and (ii) the power of the noise process. This ratio is an instance of a signal-to-noise ratio (SNR), and such ratios are of fundamental importance in understanding the performance of a communication system.

3. At the signal abstraction, additive white Gaussian noise is often a good noise model. At the bit abstraction, this model is inconvenient because we would have to keep going to the signal level to figure out exactly how it affects every bit. Fortunately, the BER allows us to think about the impact of noise in terms of how it affects bits. In particular, a simple, but powerful, model at the bit level is that of a binary symmetric channel (BSC). Here, a transmitted bit b (0 or 1) is interpreted by the receiver as 1 − b with probability p_e and interpreted as b with probability 1 − p_e, where p_e is the probability of a bit error (i.e., the bit error rate). In this model, each bit is corrupted independently of the others, and the probability of corruption is the same for all bits (so the noise process is an example of an "iid" random process: "independent and identically distributed").

4.3 Origins of noise

A common source of noise in radio and acoustic communications arises from interferers who might individually or collectively make it harder to pick out the communication that the receiver is primarily interested in. For example, the quality of Wi-Fi communication is affected by other Wi-Fi communications in the same frequency band, an example of interference from other users or nodes in the same network. In addition, interference could be caused by sources external to the network of interest; Wi-Fi, for example, if affected by cordless phones, microwave ovens, Bluetooth devices, and so on that operate at similar radio frequencies. Microwave ovens are doubly troublesome if you're streaming music over Wi-Fi, which in the most common mode runs in the 2.4 GHz frequency band today—not only do microwave ovens create audible disturbances that affect your ability to listen to music, but they also radiate power in the 2.4 GHz frequency band. This absorption is good for heating food, but leakage from ovens interferes with Wi-Fi receptions. In addition, wireless communication networks like Wi-Fi, long-range cellular networks, short-range Bluetooth radio links, and cordless phones all suffer from fading, because users often move around and signals undergo a variety of reflections that interfere with each other (a phenomenon known as “multipath fading”). All these factors cause the received signal to be different from what was sent.

If the communication channel is a wire on an integrated circuit, the primary source of noise is capacitive coupling between signals on neighboring wires. If the channel is a wire on a printed circuit board, signal coupling is still the primary source of noise, but coupling between wires is largely inductive or carried by unintended electromagnetic radiation.

In both these cases, one might argue that the noise is not truly random, as the signals generating the noise are under the designer’s control. However, a signal on a wire in an integrated circuit or on a printed circuit board will frequently be affected by signals on thousands of other wires, so approximating the interference using a random noise model turns out to work very well. Noise may also arise from truly random physical phenomena. For example, electric current in an integrated circuit is generated by electrons moving through wires and across transistors. The electrons must navigate a sea of obstacles (atomic nuclei), and behave much like marbles traveling through a Pachinko machine. They collide randomly with nuclei and have transit times that vary randomly. The result is that electric currents have random noise. In practice, however, the amplitude of the noise is typically several orders of magnitude smaller than the nominal current. Even in the interior of an integrated circuit, where digital information is transported on micron-wide wires, the impact of electron transit time fluctuations is negligible. By contrast, in optical communication channels, fluctuations in electron transit times in circuits used to convert between optical and electronic signals at the ends of the fiber are the dominant source of noise.

To summarize: there is a wide variety of mechanisms that can be the source of noise; as a result, the bottom line is that it is physically impossible to construct a noise-free channel. By understanding noise and analyzing its effects (bit errors), we can develop approaches to reducing the probability of errors caused by noise and to combat the errors that will inevitably occur despite our best efforts. We will also learn in a later chapter about a celebrated and important result of Shannon: provided the information transmission rate over a channel is kept below a limit referred to as the channel capacity (determined solely by the distortion and noise characteristics of the channel), we can transmit in a way that makes the probability of error in decoding the sender's message vanish asymptotically as the message size goes to ∞. This asymptotic performance is attained at the cost of increasing computational burden and increasing delay in deducing the sender's message at the receiver. Much research and commercial development has gone into designing practical methods to come close to this "gold standard".

4.3.1 Additive White Gaussian Noise: A Simple but Powerful Model

A simple model for how noise affects the reception of a signal sent over a channel and processed by the receiver. In this model, noise is:

1. Additive: Given a received sample value y[k] at the kth sample time, the receiver interprets it as the sum of two components: the first is the noise-free component y0[k], i.e., the sample value that would have been received at the kth sample time in the absence of noise, as a result of the input waveform being passed through the channel with only distortion present; and the second is the noise component w[k], assumed independent of the input waveform. We can thus write

$y[k]=y_0[k]+w[k]$.

In the absence of distortion, which is what we are assuming here, y0[k]will be either V_0 or V_1.

2. Gaussian: The noise component w[k]is random, but we assume it is drawn at each sample time from a fixed Gaussian distribution; for concreteness, we take this to be the distribution of a Gaussian random variable W, so that each w[k] is distributed exactly as W is. The reason why a Gaussian makes sense is because noise is often the result of summing a large number of different and independent factors, which allows us to apply an important result from probability and statistics, called the **central limit theorem**. This states that the sum of independent random variables is well approximated (under rather mild conditions) by a Gaussian random variable, with the approximation improving as more variables are summed in.

The Gaussian distribution is beautiful from several viewpoints, not least because it is characterized by just two numbers: its mean μ, and its variance σ^2 or standard deviation σ. In our noise model, we will assume that the mean of the noise distribution is 0. This assumption is not a huge concession: any consistent non-zero perturbation is easy to compensate for. For zero-mean Gaussian noise, the variance, or equivalently the standard deviation, completely characterizes the noise. The standard deviation σ may be thought of as a measure of the expected "amplitude" of the noise; its square captures the expected power.

For noise not to corrupt the digitization of a bit detection sample, the distance between the noise-free value of the sample and the digitizing threshold should be sufficiently larger than the expected amplitude or standard deviation of the noise.

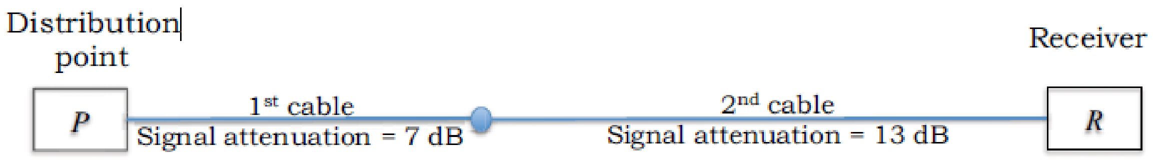
3. White: This property concerns the temporal variation in the individual noise samples that affect the signal. If these Gaussian noise samples are independent from one sample to another, the underlying noise process is referred to as white Gaussian noise. "White" refers to the frequency decomposition of the sequence of noise samples, and essentially says that the noise signal contains components of equal expected power at all frequencies. This statement will become clearer later in the course when we talk about the frequency content of signals. This noise model is generally given the term AWGN, for additive white Gaussian noise.

Summary of Study 4

Self-Assessment Question (SAQs) for lecture 4

SAQ 4.1.

The cable television signal in your home is poor. The receiver in your home is connected to the distribution point outside your home using two coaxial cables in series, as shown in the picture below. The power of the cable signal at the distribution point is P. The power of the signal at the receiver is R.



The first cable attenuates (i.e., reduces) the signal power by 7 db. The second cable attenuates the signal power by an additional 13 dB Calculate P/R as a numeric ratio.

SAQ 4.2

Bit samples are transmitted with amplitude ATX=±1(i.e. bipolar signaling). The channel attenuation is 20dB, so the power of any transmitted signal is reduced by this factor when it arrives at the receiver.

(a)What receiver noise standard deviation value(σ)corresponds to a signal-to-noise ratio (SNR) of 20dB at the receiver? (Note that the SNR at the receiver is defined as the ratio of the received signal power to σ^2.)

(b)Express the bit error rate at the receiver in terms of the erfc () function when the SNR at the receiver is 20dB.

(c)Under the conditions of the previous parts of this question, suppose an amplifier with gain of 10dB is added to the receiver after the signal has been corrupted with noise. Explain how this amplification affects the bit error rate.

Study Session5: Transmitting on a Physical Channel, linear Time-Invariant (LTI) Systems

Introduction

This present chapter begins the process and continued through several subsequent process of representing, modeling, analyzing, and exploiting the characteristics of the physical transmission channel. This is the channel seen between the signal transmitted from the source and the signal captured at the receiver. our intent is to study in more detail the portion of the communication channel represented by the connection between "Mapper + Xmit samples" at the source side, and "Recv samples + Demapper" at the receiver side.

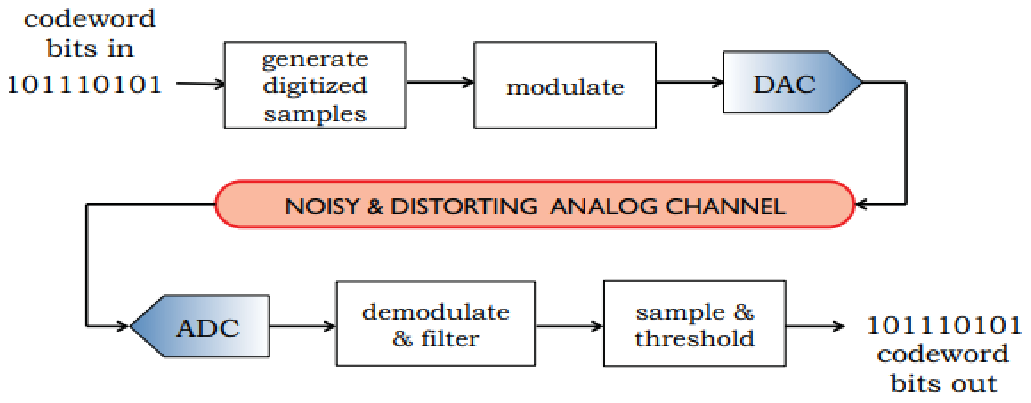

codeword bits in
101110101
generate digitized samples
modulate
DAC
NOISY & DISTORTING ANALOG CHANNEL
ADC
demodulate & filter
sample & threshold
101110101 codeword bits out

Figure 5-1: Elements of a communication channel between the channel coding step at the transmitter and channel decoding at the receiver.

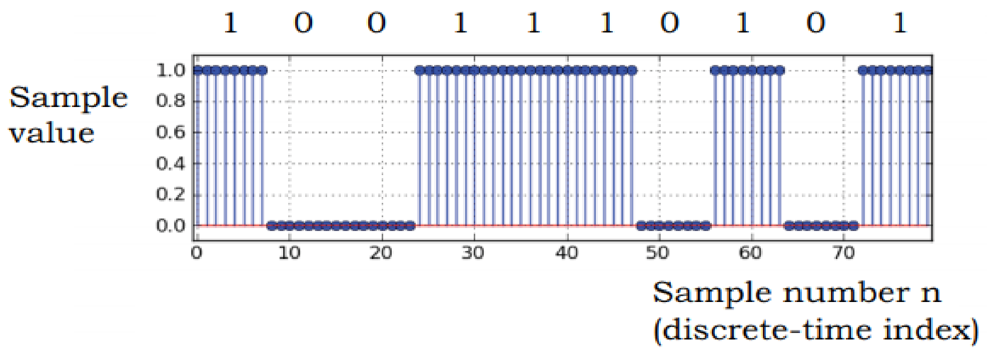


Figure 5-2: Digitized samples of the baseband signal

5.1 Getting the Message Across

5.1.1 The Baseband Signal

In Figure 5-1 we see an expanded version of what might come between the channel coding operation at the transmitter and the channel decoding operation at the receiver. At the source, the first stage is to convert the input bit stream to a digitized and discrete-time (DT) signal, represented by samples produced at a certain sample rate f_s samples/s. We denote this signal by x[n], where n is the integer-valued discrete-time index, ranging in the most general case from $-\infty$ to $+\infty$. In the simplest case, which we will continue to use for illustration, each bit is represented by a signal level held for a certain number of samples, for instance a voltage level of $V_0 = 0$ held for 8 samples to indicate a 0 bit, and a voltage level of $V_1 = 1$ held for 8 samples to indicate a 1 bit, as in Figure 5-2. The sample clock in this example operates at 8 times the rate of the bit clock, so the bit rate is fs/8 bits/s. Such a signal is usually referred to as the **baseband signal.**

5.1.2 Modulation

The DT baseband signal shown in Figure 5-2 is typically not ready to be transmitted on the physical transmission channel. For one thing, physical channels typically operate in continuous-time (CT) analog fashion, so at the very least one needs a digital-to-analog converter (DAC) to produce a continuous-time signal that can be applied to the channel. The DAC is usually a simple zero-order hold, which maintains or holds the most recent sample value for a time interval of 1/fs. With such a DAC conversion, the DT “rectangular wave” in Figure 5-2 becomes a CT rectangular wave, each bit now corresponding to a signal value that is held for 8/fs seconds.

Conversion to an analog CT signal will not suffice in general, because the physical channel is usually not well suited to the transmission of rectangular waves of this sort. For instance, a speech signal from a microphone may, after appropriate coding for digital transmission, result in 64 kilobits of data per second (a consequence of sampling the microphone waveform at 8 kHz and 8-bit resolution), but a rectangular wave switching between two levels at this rate is not adapted to direct radio transmission. The reasons include the fact that efficient projection of wave energy requires antennas of dimension comparable with the wavelength of the signal, typically a quarter wavelength in the case of a tower antenna. At 32 kHz, corresponding to the waveform associated with alternating 1’s and 0’s in the coded microphone output, and with the electromagnetic waves propagating at 3 × 108meters/s (the speed of light), a quarter-wavelength antenna would be a rather unwieldy 3 × 108/(4 × 32 × 103) = 2344 meters long! Even if we could arrange for such direct transmission of the baseband signal (after digital-to-analog conversion), there would be issues related to the required transmitter power, the attenuation caused by the atmosphere at this frequency, interference between this transmission and everyone else’s, and so on. Regulatory organizations such as the U.S. Federal Communications Commission (FCC), and equivalent bodies in other countries, impose constraints on transmissions, which further restrict what sort of signal can be applied to a physical channel. In order to match the baseband signal to the physical and regulatory specifications of a transmission channel, one typically has to go through a modulation process. This process converts the digitized samples to a form better suited for transmission on the available channel. Consider, for example, the case of direct transmission of digital information on an acoustic channel, from the speaker on your computer to the microphone on your computer (or another computer within “hearing” distance). The speaker does not respond effectively to the piecewise-constant voltages that arise from our baseband signal. It is instead designed to respond to oscillatory voltages at frequencies in the appropriate range, producing and projecting a wave of oscillatory acoustic pressure. Excitation by a sinusoidal wave produces a pure acoustic tone. With a speaker aperture dimension of about 5cm (0.05 meters), and a sound speed of around 340 meters/s, we anticipate effective projection of tones with frequencies in the low kilohertz range, which is indeed in (the high end of) the audible range. A simple way to accomplish the desired modulation in the acoustic wave example above is to apply at the output of the digital-to-analog converter, which feeds the loudspeaker—a voltage V0 cos(2πfct) for some duration of time to signal a 0 bit, and a voltage of the form V1 cos(2πfct) for the same duration of time to signal a 1 bit. Here cos(2πfct) is referred to as the carrier signal and fc is the carrier frequency, chosen to be appropriately matched to the channel characteristics. This particular way of imprinting the baseband signal on a carrier by varying its amplitude is referred to as amplitude modulation (AM), which we will study in more detail. The choice V_0 = 0 and V_1 = 1 is also referred to as on-off keying, with a burst of pure tone (“on”) signaling a 1 bit, and an interval of silence (“off”) signaling a 0. One could also choose V_0 = −1 and V_1 = +1, which would result in a sinusoidal voltage that switches phase by π/2 each time the bit stream goes from 0 to 1 or from 1 to 0. This approach may be referred to as polar keying (particularly when it is thought of as an instance of amplitude modulation), but is more commonly termed binary phase-shift keying (BPSK).

Yet another modulation possibility for this acoustic example is frequency modulation (FM), where a tone burst of a particular frequency in the neighborhood of fc is used to signal a 0 bit, and a tone burst at another frequency to signal a 1 bit. All these schemes are applicable to radio frequency (RF) transmissions as well, not just acoustic transmissions, and are in fact commonly used in practice for RF communication.

5.1.3 Demodulation

We shall have more to say about demodulation later, so for now it suffices to think of it as a process that is inverse to modulation, aimed at extracting the baseband signal from the received signal. While part of this process could act directly on the received CT analog signal, the block diagram in Figure 10-1 shows it all happening in DT, following conversion of the received signal using an analog-to-digital converter (ADC). The block diagram also indicates that a filtering step may be involved, to separate the channel noise as much as possible from the signal component of the received signal, as well as to compensate for deterministic distortion introduced by the channel. These ideas will be explored further in later chapters.

5.1.4 The Baseband Channel

The result of the demodulation step and any associated filtering is a DT signal y[n], comprising samples arriving at the rate fs used for transmission at the source. We assume issues of clock synchronization are taken care of separately. We also neglect the effects of any signal attenuation, as this can be simply compensated for at the receiver by choosing an appropriate amplifier gain. In the ideal case of no distortion, no noise on the channel, and insignificant propagation delay, y[n] would exactly equal the modulating baseband signal x[n] used at the source, for all n. If there is a fixed and known propagation delay on the channel, it can be convenient to simply set the clock at the receiver that much later than the clock at the sender. If this is done, then again, we would find that in the absence of distortion and random noise, we

get y[n] = x[n] for all n.

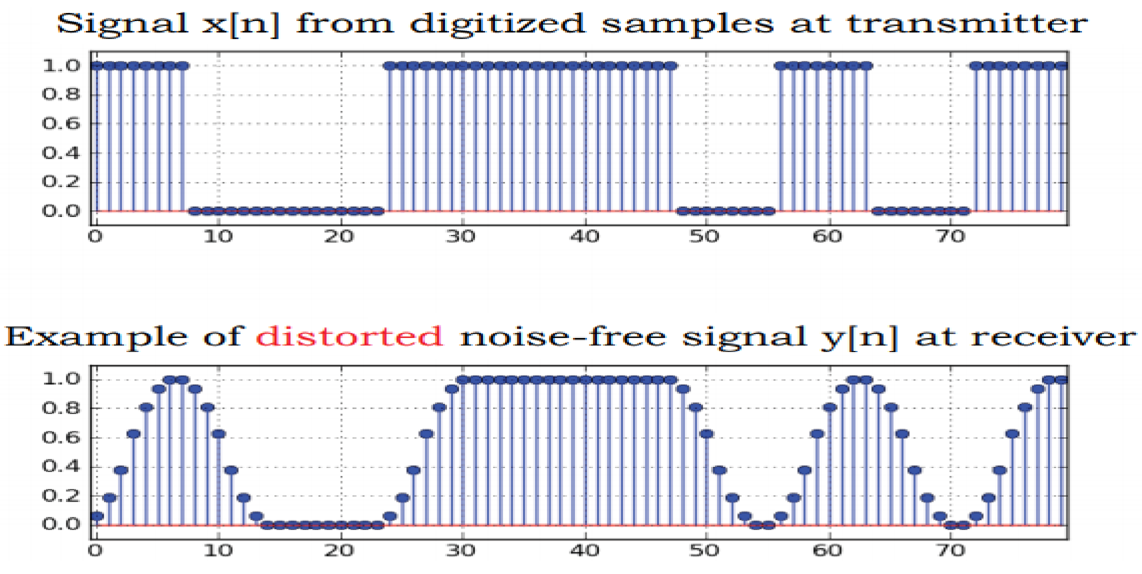


Figure 5-3: Channel distortion example. The distortion is deterministic

More realistically, the channel does distort the baseband signal, so the output DT signal may look (in the noise-free case) as the lower waveform in Figure 5-3. Our objective in what follows is to develop and analyze an important class of models, namely linear and time-invariant (LTI) models, that are quite effective in accounting for such distortion, in a vast variety of settings. The models would be used to represent the end-to-end behavior of what might be called the baseband channel, whose input is x[n] and output is y[n], as in Figure 5-3.

5.2 Linear Time-Invariant (LTI) Models

5.2.1 Baseband Channel Model

Our baseband channel model, as represented in the block diagram in Figure 5-4 takes the DT sequence or signal x[.] as input and produces the sequence or signal y[.] as output. We will often use the notation x[.]—or even simply just x—to indicate the entire DT signal or function. Another way to point to the entire signal, though more cumbersome, is by referring to "x[n] for −∞ <n< ∞"; this often gets abbreviated to just "the signal x[n]", at the risk of being misinterpreted as referring to just the value at a single time n. Figure 5-4 shows x[n] at the input and y[n] at the output, but that is only to indicate that this is a snapshot of the system at time n, so indeed we see x[n] at the input and y[n].

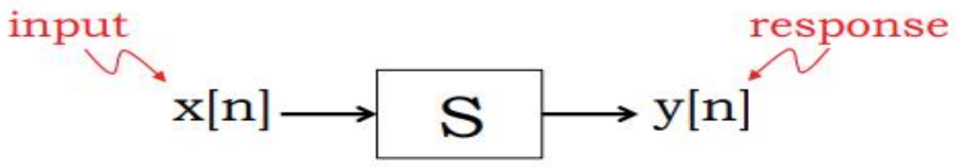


Figure 10-4: Input and output of baseband channel.

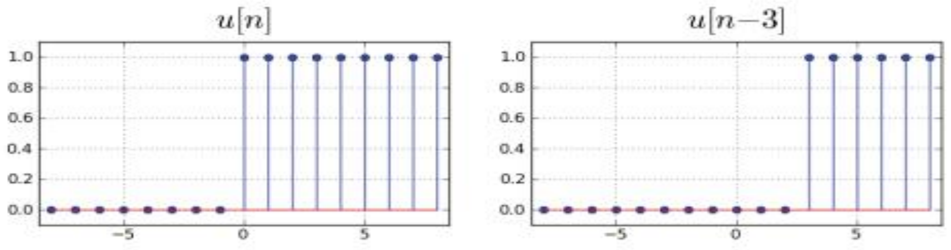


Figure 5-5: A unit step. In the picture on the left the unit step is unshifted, switching from 0 to 1 at index (time) 0. On the right, the unit step is shifted forward in time by 3 units (shifting forward in time means that we use the − sign in the argument because we want the switch to occur with n − 3=0).

at the output of the system. What the diagram should not be interpreted as indicating is that the value of the output signal y[.] at time n depends exclusively on the value of the input signal at that same time n. In general, the value of the output y[.] at time n, namely y[n], could depend on the values of the input x[.] at all times. We are most often interested in causal models, however, and those are characterized by y[n] only depending on past and present values of x[.], i.e., x[k] for k ≤ n.

5.2.2 Linear, Time-Invariant (LTI) Models

Models that are both linear and time-invariant, or LTI models, are hugely important in engineering and other domains. We will mention some of the reasons in the next chapter. We will develop insights into their behavior and tools for their analysis, and then return to apply what we have learned, to better understand signal transmission on physical channels. In the context of audio communication using a computer's speaker and microphone, transmissions are done using bursts at the loudspeaker of a computer, and receptions by detecting the response at a microphone. The input x[n] in this case is a baseband signal of the form in Figure 10-3, but alternating regularly between high and low values. This was converted through a modulation process into the tone bursts that you heard. The signal received at the microphone is then demodulated to reconstruct an estimate y[n] of the baseband input. With the microphone in a fixed position, responses have some consistency from one transition to the next (between tone and no-tone), despite the presence of random fluctuations riding on top of things. The deterministic or repeatable part of the response y[n] does show distortion, i.e., deviation from x[n], though more "real-world" than what is shown in the synthetic example in Figure 5-3. However, when the microphone is very close to the speaker, the distortion is low. There were features of the system response in this communication system to suggest that it may not be unreasonable to model the baseband acoustic channel as LTI. Time invariance (at least over the time-horizon of the demo!) is suggested by the repeatability of the transient responses to the various transitions. Linearity is suggested by the fact that the downward transients caused by negative (i.e., downward) steps at the input look like reflections of the upward transients caused by positive (i.e., upward) steps of the same magnitude at the input, and is also suggested by the appropriate scaling of the response when the input is scaled.

Summary of lecture 5

Self-Assessment Question (SAQs) for lecture 5

SAQ 5.1.

Study Session 6: LTI channel, inter symbol Interference and Convolution

Introduction

This chapter will help us understand what else besides noise perturbs or distorts a signal transmitted over a communication channel, such as a voltage waveform on a wire, a radio wave through the atmosphere, or a pressure wave in an acoustic medium. The most significant feature of the distortion introduced by a channel is that the output of the channel typically does not instantaneously respond to or follow the input. The physical reason is ultimately some sort of inertia effect in the channel, requiring the input to supply energy in order to generate a response, and therefore requiring some time to respond, because the input power is limited. Thus, a step change in the signal at the input of the channel typically causes a channel output that rises more gradually to its final value, and perhaps with a transient that oscillates or "rings" around its final value before settling. A succession of alternating steps at the input, as would be produced by on-off signaling at the transmitter, may therefore produce an output that displays intersymbol interference (ISI): the response to the portion of the input signal associated with a particular bit slot spills over into other bit slots at the output, making the output waveform only a feeble representation of the input, and thereby complicating the determination of what the input message was.

To understand channel response and ISI better, we will use linear, time-invariant (LTI) models of the channel, which we introduced in the previous chapter. Such models provide very good approximations of channel behavior in a range of applications. In an LTI model, the response (i.e., output) of the channel to any input depends only on one function, h[·], called the unit sample response function. Given any input signal sequence, x[·], the output y[·] of an LTI channel can be computed by combining h[·] and x[·] through an operation known as convolution. Knowledge of h[·] will give us guidance on choosing the number of samples to associate with a bit slot in order to overcome ISI, and will thereby determine the maximum bit

rate associated with the channel. In simple on-off signaling, the number of samples that need to be allotted to a bit slot in order to mitigate the effects of ISI will be approximately the number of samples required for the unit sample response h[·] to essentially settle to 0. In this connection, we will also introduce a tool called an eye diagram, which allows a communication engineer to determine whether the number of samples per bit is large enough to permit reasonable communication quality in the face of ISI.

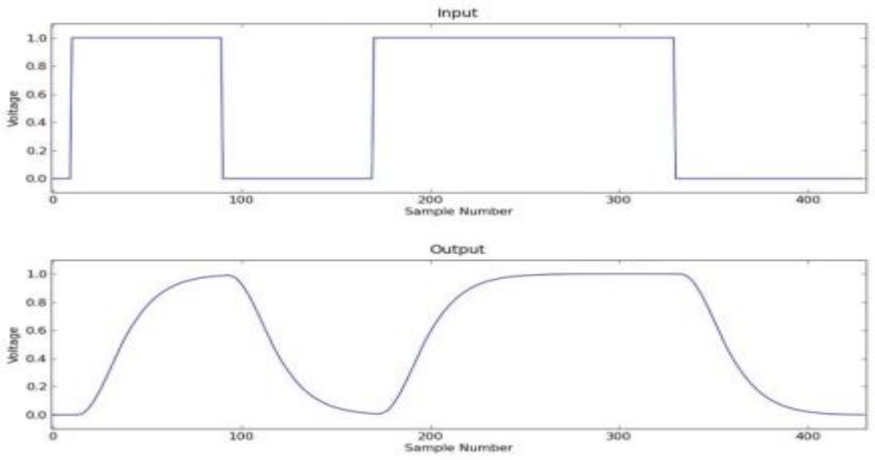


Figure 6-1: On-off signaling at the channel input produces a channel output that takes a non-zero time to rise or fall, and to settle at 1 or 0.

6.1 Distortions on a Channel

Even though communication technologies come in enormous variety, they generally all exhibit similar types of distorting behavior in response to inputs. To gain some intuition on the basic nature of the problem, we first look at some simple examples. Consider a transmitter that does on-off signaling, sending voltage samples that are either set to V_0 = 0 volts or to V_1 = 1 volt for all the samples in a bit period. Let us assume an LTI channel, so that

1. the superposition property applies, and

2. if the response to a unit step u[n] at the input is unit-step response s[n] at the output, then the response to a shifted unit step u[n − D] at the input is the identically shifted unit-step response s[n − D], for any (integer) D. Let us also assume that the channel and receiver gain are such that the unit step response s[n] eventually settles to 1. In this setting, the output waveform will typically have two notable deviations from the input waveform:

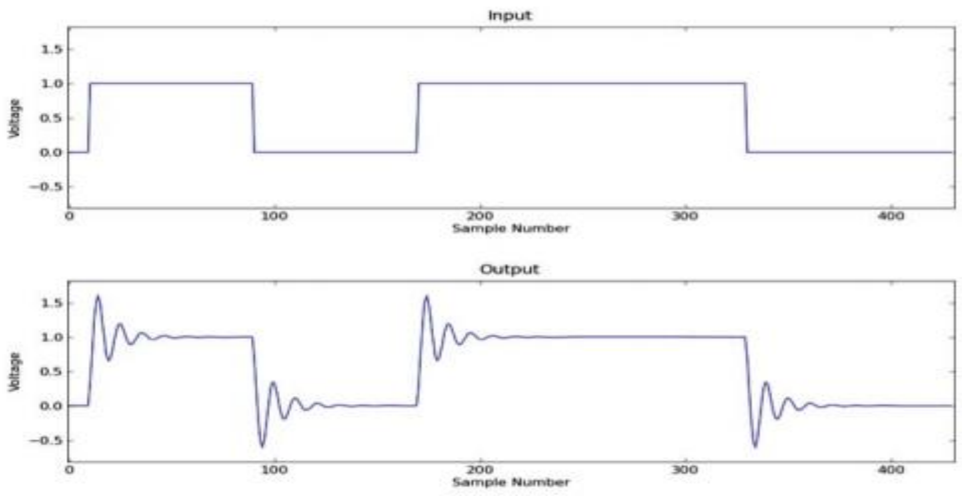


Figure 6-2: A channel showing "ringing"

1. A slower rise and fall. Ideally, the voltage samples at the receiver should be identical to the voltage samples at the transmitter. Instead, as shown in Figure 6-1, one usually finds that the nearly instantaneous transition from V_0 volts to V_1 volts at the transmitter results in an output voltage at the receiver that takes longer to rise from V_0 volts to V_1 volts. Similarly, when there is a nearly instantaneous transition from V_1 volts to V_0 volts at the transmitter, the voltage at the receiver takes longer to fall. It is important to note that if the time between transitions at the

transmitter is shorter than the rise and fall times at the receiver, the receiver will struggle (and/or fail!) to correctly infer the value of the transmitted bits using the voltage samples from the output.

2. Oscillatory settling, or “ringing”. In some cases, voltage samples at the receiver will oscillate before settling to a steady value. In cables, for example, this effect can be due to a “sloshing” back and forth of the energy stored in electric and magnetic fields, or it can be the result of signal reflections at discontinuities. Over radio and acoustic channels, this behavior arises usually from signal reflections. We will not try to determine the physical source of ringing on a channel, but will instead observe that it happens and deal with it. Figure 6-2 shows an example of ringing. Figure 6-3 shows an example of non-ideal channel distortions. In the example, the transmitter converted the bit sequence ...0101110... to voltage samples using ten 1volt samples to represent a “1” and ten 0volt samples to represent a “0” (with sample values of 0 before and after). In the example, the settling time at the receiver is longer than the reciprocal of the bit period, and therefore bit sequences with frequent transitions, like 010,

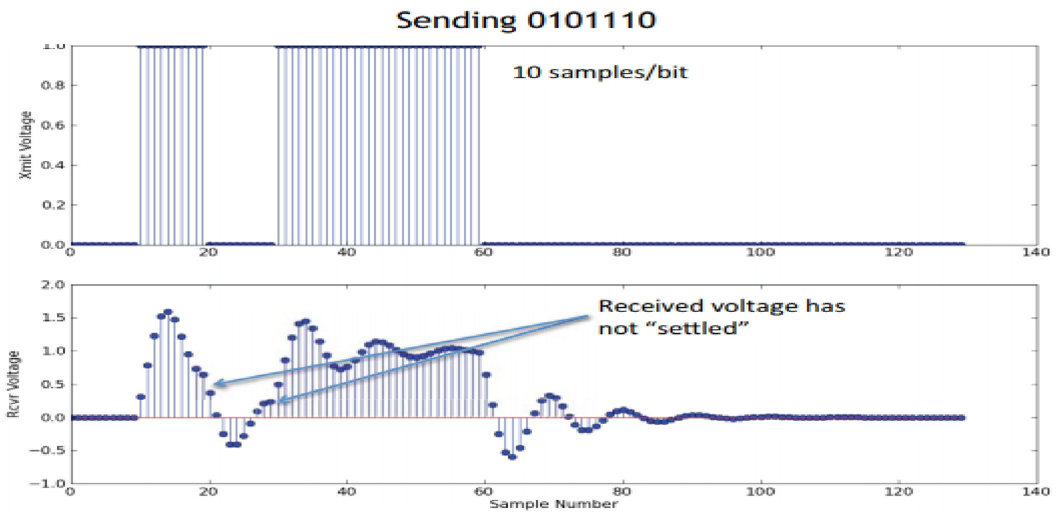


Figure 6-3: The effects of rise/fall time and ringing on the received signal

are not received faithfully. In Figure 6-3, at sample number 21, the output voltage is still ringing in response to the rising input transition at sample number 10, and is also responding to the input falling transition at sample number 20. The result is that the receiver may misidentify the value of one of the transmitted bits. Note also that the receiver will certainly correctly determine that the fifth and sixth bits have the value ’1’, as there is no transition between the fourth and fifth, or fifth and sixth, bit. As this example demonstrates, the slow settling of the channel output implies that the receiver is more likely to wrongly identify a bit that differs in value from its immediate predecessors. This example should also provide the intuition that if the number of samples per bit is large enough, then it becomes easier for the receiver to correctly identify bits because each sequence of samples has enough time to settle to the correct value (in the absence of noise, which is of course a random phenomenon that can still confound the receiver).

There is a formal name given to the impact of rise/fall times and settling times that are long compared to a bit slot: we say that the channel output displays inter-symbol interference, or ISI. ISI is a fancy way of saying that the received samples corresponding to the current bit depend on the values of samples corresponding to preceding bits. Figure 6-4 shows four examples: two for channels with a fast rise/fall compared to the duration of the bit slot, and two for channels with a slower rise/fall.

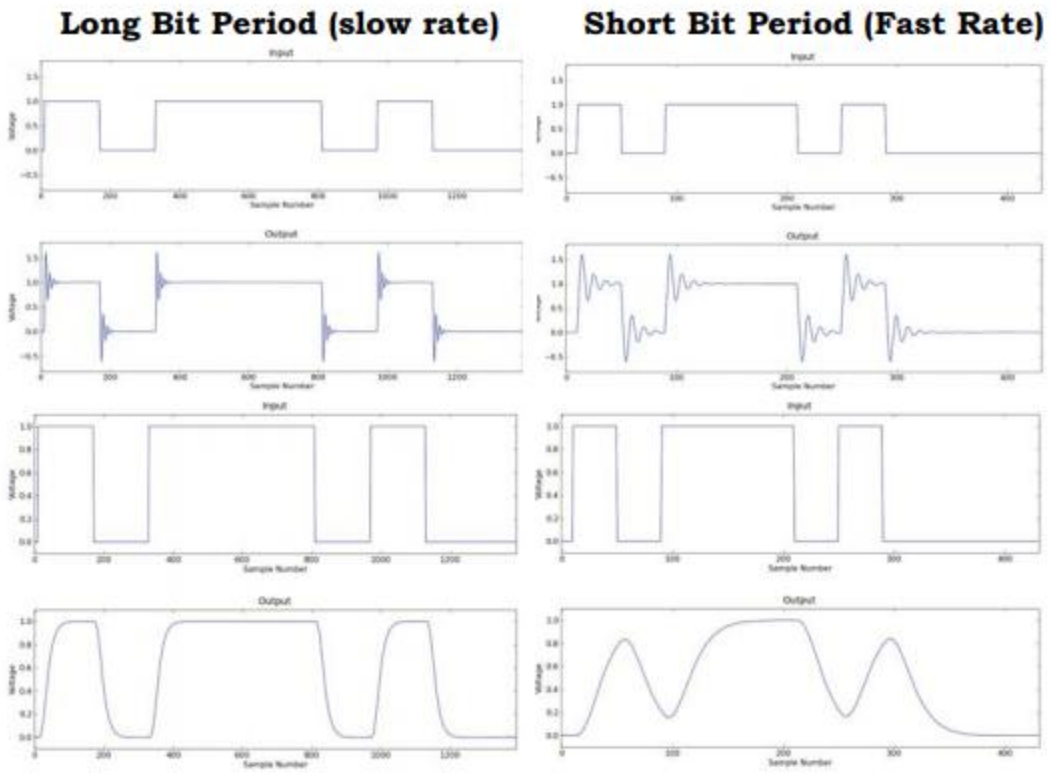
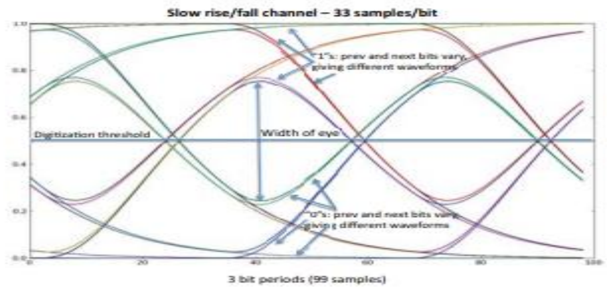


Figure 6-4: Examples of ISI.

We now turn to a more detailed study of LTI models, which will allow us to understand channel distortion and ISI more fundamentally.

6.4 Eye Diagrams

On the face of it, ISI is a complicated effect because the magnitude of bit interference and the number of interfering bits depend both on the channel properties and on how bits are represented on the channel. Figure 6-6 shows an example of what the receiver sees (bottom) in response to what the transmitter sent (top) over a channel with ISI but no noise. Eye diagrams (or "eye patterns") are a useful graphical tool in the toolkit of a communications system designer or engineer to understand ISI. We will use this tool to determine whether the number of samples per bit is large enough to enable the receiver to determine "0"s and "1"s reliably from the demodulated (and filtered) sequence of received voltage samples. To produce an eye diagram, one begins with the channel output that results from a long stretch of on-off signaling, as in the bottom part of Figure 6-6, then essentially slices this up into smaller segments, say 3 bit-slots long, and overlays all the resulting segments. The result spans the range of waveform variations one is likely to see over any 3 bit-slots at the output. A more detailed prescription follows. Take all the received samples and put them in an array of lists, where the number of lists in the array is equal to the number of samples in k bit periods. In practice, we want k to be a small positive integer like 3. If there are s samples per bit, the array is of size k · s. Each element of this array is a list, and element i of the array is a list of the received



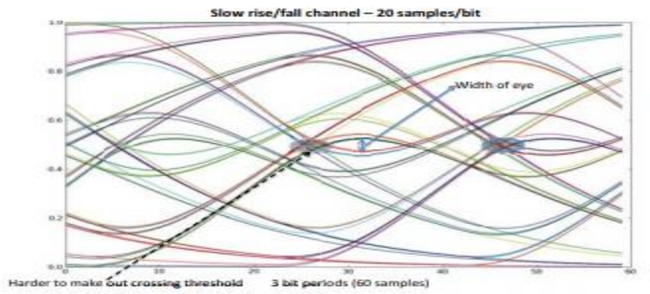


Figure 6-5: Eye diagrams for a channel with a slow rise/fall for 33 (top) and 20 (bottom) samples per bit. Notice how the eye is wider when the number of samples per bit is large, because each step response has time to settle before the response to the next step appears.

samples y[i], y[i + ks], y[i + 2ks],.... Now suppose there were no ISI at all (and no noise). Then all the samples in the Ith list corresponding to a transmitted "0" bit would have the same voltage value, and all the samples in the Ith list corresponding to a transmitted "1" would have the same value. Consider the simple case of just a little ISI, where the previous bit interferes with the current bit, and there's no further impact from the past. Then the samples in the ith list corresponding to a transmitted "0" bit would have two distinct possible values, one value associated with the transmission of a "10" bit sequence, and one value associated with a "00" bit sequence. A similar story applies to the samples in the ith list corresponding to a transmitted

"1" bit, for a total of four distinct values for the samples in the ith list. If there is more ISI, there will be more distinct values in the ith list of samples. For example, if two previous bits interfere, then there will be eight distinct values for the samples in the ith list. If three bits interfere, then the ith list will have 16

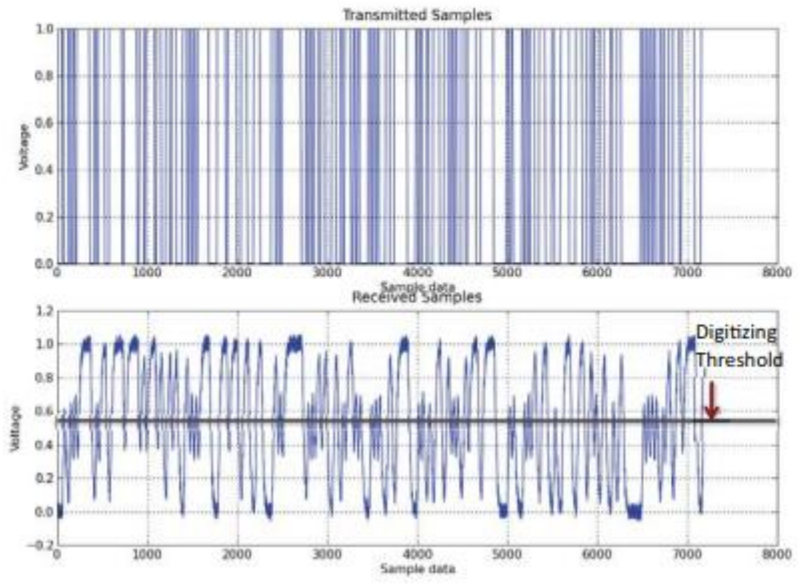


Figure 11-11: Received signals in the presence of ISI. Is the number of samples per bit "just right"? And what threshold should be used to determine the transmitted bit? It's hard to answer this question from this picture. An eye diagram sheds better light.

distinct values, and so on.

Without knowing the number of interfering bits, to capture all the possible interactions, we must produce the above array of lists for every possible combination of bit sequence that can ever be observed. If we were to plot this array on a graph, we will see a picture like the one shown in Figure 6-5. This picture is an eye diagram. In practice, we can't produce every possible combination of bits, but what we can do is use a long random sequence of bits. We can take the random bit sequence, convert it in to a long sequence of voltage samples, transmit the samples through the channel, collect the received samples, pack the received samples in to the array of lists described above, and then plot the result. If the sequence is long enough, and the number of interfering bits is small, we should get an accurate approximation of the eye diagram. But what is "long enough"? We can answer this question and develop a less ad hoc procedure by using the properties of the unit sample response, h[n]. The idea is that the sequence h[0], h[1],...,h[n],... captures the complete noise-free response of the channel. If h[k] ≈ 0 for k>¥, then we don't have to worry about samples more than ¥ in the past. Now, if the number of samples per bit is s, then the number of bits in the past that can affect the present bit is no larger than ¥/s, where ¥ is the length of the non-zero part of h[·]. Hence, it is enough to generate all bit patterns of length B = ¥/s, and send them through the channel to produce the eye diagram. In practice, because noise can never be eliminated, one might be a little conservative and pick B = (¥/n)+2, slightly bigger than what a noise-free calculation would indicate. Because this approach requires 2B bit patterns to be sent, it might be unreasonable for large values of B; in those cases, it is likely that s is too small, and one can find whether that is so by sending a random subset of the 2B possible bit patterns through the channel. Figure 6-5 shows the width of the eye, the place where the diagram has the largest distinction between voltage samples associated with the transmission of a '0' bit and those associated with the transmission of a '1' bit. Another point to note about the diagrams is the "zero crossing", the place where the upward rising and downward falling curves cross. Typically, as the degree of ISI increases (i.e., the number of samples per bit is reduced), there is a greater degree of "fuzziness" and ambiguity about the location of this zero crossing. The eye diagram is an important tool, useful for verifying some key design and operational decisions:

1. Is the number of samples per bit large enough? If it is large enough, then at the center of the eye, the voltage samples associated with transmission of a '1' are clearly above the digitization threshold and the voltage samples associated with the transmission of a '0' are clearly below. In addition, the eye must be "open" enough that small amounts of noise will not lead to errors in converting bit detection samples to bits. As will become clear later, it is impossible to guarantee that noise will never cause errors, but we can reduce the likelihood of error.

2. Has the value of the digitization threshold been set correctly? The digitization threshold should be set to the voltage value that evenly divides the upper and lower halves of the eye, if 0's and 1's is equally likely. We didn't study this use of eye diagrams, but mention it because it is used in practice for this purpose as well.

3. Is the sampling instant at the receiver within each bit slot appropriately picked? This sampling instant should line up with where the eye is most open, for robust detection of the received bits.

Summary of Study Session 6

Self-Assessment Question (SAQs) for lecture 6

SAQ 6.1.

1. Each of the following equations describes the relationship that holds between the input signal x[.] and output signal y[.] of an associated discrete-time system, for all integers n. In each case, explain whether or not the system is (i) causal , (ii) linear, (iii) time-invariant.

(a) $y[n] = 0.5x[n] + 0.5x[n-1]$.

(b) $y[n] = x[n+1] + 7$.

(c) $y[n] = \cos(3n)\, x[n-2]$.

(d) $y[n] = x[n]\, x[n-1]$.

(e) $y[n] = \sum_{k=13}^{n} x[k]$ for $n \geq 13$, otherwise $y[n] = 0$.

(f) $y[n] = x[-n]$.

SAQ 6.1.

(By Yury Polyanskiy.) Explain whether each of the following statements is true or false.

(a) Let S be the LTI system that delays signal by D. Then h ∗ S(x) = S(h) ∗ x for any signals h and x.

(b) Adding a delay by D after LTI system h[n] is equivalent to replacing h[n] with h[n − D].

(c) if h ∗ x[n]=0 for all n then necessarily one of signals h[·] or x[·] is zero.

(d) LTI system is causal if and only if h[n]=0 for n < 0.

(e) LTI system is causal if and only if u[n]=0 for n < 0.

(f) For causal LTI h[n] is zero for all large enough n if and only if u[n] becomes constant for all large enough n.

(g) s[n] is zero for all n ≤ n0 and then monotonically grows for n>n0 if and only if h[n] is zero for all n ≤ n0 and then non-negative for n>n0.

Study Session 7: Frequency Response of LTI systems

Introduction

Sinusoids—and their close relatives, the complex exponentials—play a distinguished role in the study of LTI systems. The reason is that, for an LTI system, a sinusoidal input gives rise to a sinusoidal output again, and at the same frequency as the input. This property is not obvious from anything we have said so far about LTI systems. Only the amplitude and phase of the sinusoid might be, and generally are, modified from input to output, in a way that is captured by the frequency response of the system, which we introduce in this chapter.

7.1 Sinusoidal Inputs

Before focusing on sinusoidal inputs, consider an input that is periodic but not necessarily sinusoidal. A signal x[n] is periodic if

$x[n + P] = x[n]$ for all n ,

where P is some fixed positive integer. The smallest positive integer P for which this condition holds is referred to as the period of the signal (though the term is also used at times for positive integer multiples of P), and the signal is called P-periodic. While it may not be obvious that sinusoidal inputs to LTI systems give rise to sinusoidal outputs, it’s not hard to see that periodic inputs to LTI systems give rise to periodic outputs of the same period (or an integral fraction of the input period). The reason is that if the P-periodic input x[.] produces the output y[.], then time-invariance of the system means that shifting the input by P will shift the output by P. But shifting the input by P leaves the input unchanged, because it is P-periodic, and therefore must leave the output unchanged, which means the output must be P-periodic. (This argument actually leaves open the possibility that the period of the output is P/K for some integer K, rather than actually P-periodic, but in any case we will have $y[n + P] = y[n]$ for all n.)

7.1.1 Discrete-Time Sinusoids

A discrete-time (DT) sinusoid takes the form

$x[n] = \cos(\Omega0n + \theta0)$, eq(7.1)

We refer to Ω0 as the angular frequency of the sinusoid, measured in radians/sample; Ω0 is the number of radians by which the argument of the cosine increases when n increases by 1. (It should be clear that we can replace the cos with a sin in Equation (7.1), because cos and sin are essentially equivalent except for a pi/2 phase shift.) Note that the lowest rate of variation possible for a DT signal is when it is constant, and this corresponds, in the case of a sinusoidal signal, to setting the frequency Ω_0 to 0. At the other extreme, the highest rate of variation possible for a DT signal is when it alternates signs at each time step, as in $(-1)n$. A sinusoid with this property is obtained by taking $\Omega0 = \pm\pi$, because $\cos(\pm\pi n)=(-1)n$. Thus all the action of interest with DT sinusoids happens in the frequency range $[-\pi,\pi]$. Outside of this interval, everything repeats periodically in Ω0, precisely because adding any integer multiple of 2π to Ω0 does not change the value of the cosine in Equation (7.1).

It can be helpful to consider this DT sinusoid as derived from an underlying continuous time (CT) sinusoid $\cos(\omega0t + \theta0)$ of period $2\pi/\omega0$, by sampling it at times $t = nT$ that are integer multiples of some sampling interval T. Writing $\cos(\Omega0n + \theta0) = \cos(\omega0nT + \theta0)$ then yields the relation $\Omega0 = \omega0T$ (with the constraint $|\omega0| \leq \pi/T$, to reflect $|\Omega0| \leq \pi$). It is now natural to think of $2\pi/(\omega0T)=2\pi/\Omega0$ as the period of the DT sinusoid, measured in samples. However, $2\pi/\Omega0$ may not be an integer!

Nevertheless, if $2\pi/\Omega0 = P/Q$ for some integers P and Q, i.e., if $2\pi/\Omega0$ is rational, then indeed $x[n + P] = x[n]$ for the signal in Equation (7.1), as you can verify quite easily. On the other hand, if $2\pi/\Omega0$ is irrational, the DT sequence in Equation (7.1) will not actually be periodic: there will be no integer P such that $x[n + P] = x[n]$ for all n. For example, $\cos(3\pi n/4)$ has frequency $3\pi/4$ radians/sample and a period of 8, because $2\pi/3\pi/4=8/3 = P/Q$, so the period, P, is 8. On the other hand, $\cos(3n/4)$ has frequency 3/4 radians/sample, and is not periodic as a discrete-time sequence because $2\pi/3/4=8\pi/3$ is irrational. We could still refer to $8\pi/3$ as its "period", because we can think of the sequence as arising from sampling the periodic continuous-time signal cos(3t/4) at integral values of t. With all that said, it turns out that the response of an LTI system to a sinusoid of the form in Equation (7.1) is a sinusoid of the same (angular) frequency Ω0, whether or not the sinusoid is actually DT periodic. The easiest way to demonstrate this fact is to rewrite sinusoids in terms of complex exponentials.

7.2 Frequency Response

We are now in a position to determine what an LTI system does to a sinusoidal input. The streamlined approach to this analysis involves considering a complex input of the form

$x[n] = e^{j(\Omega_{0n} + \theta_0)}$ rather than $x[n] = \cos(\Omega_{0n} + \theta_0)$. The reasoning and mathematical calculations associated with convolution work as well for complex signals as they do for real signals, but the complex exponential turns out to be somewhat easier to work with (once you are comfortable working with complex numbers)—and the results for the real sinusoidal signals we are interested in can then be extracted using identities such as those in Equation (7.3). It may be helpful, however, to first just plough in and do the computations directly, substituting the real sinusoidal x[n] from Equation (7.1) into the convolution expression from the previous chapter, and making use of Equation (7.4). The purpose of doing this is to (i) convince you that it can be done entirely with calculations involving real signals; and (ii) help you appreciate the efficiency of the calculations with complex exponentials when we get to them. The direct approach mentioned above yields

$$\begin{aligned}
y[n] &= \sum_{m=-\infty}^{\infty} h[m]x[n-m] \\
&= \sum_{m=-\infty}^{\infty} h[m]\cos\Big(\Omega_0(n-m)+\theta_0\Big) \\
&= \Big(\sum_{m=-\infty}^{\infty} h[m]\cos(\Omega_0 m)\Big)\cos(\Omega_0 n+\theta_0) \\
&\quad +\Big(\sum_{m=-\infty}^{\infty} h[m]\sin(\Omega_0 m)\Big)\sin(\Omega_0 n+\theta_0) \\
&= \mathcal{C}(\Omega_0)\cos(\Omega_0 n+\theta_0)+\mathcal{S}(\Omega_0)\sin(\Omega_0 n+\theta_0)\ ,
\end{aligned}$$

where we have introduced the notation

$$\mathcal{C}(\Omega) = \sum_{m=-\infty}^{\infty} h[m]\cos(\Omega m)\ , \qquad \mathcal{S}(\Omega) = \sum_{m=-\infty}^{\infty} h[m]\sin(\Omega m)\ .$$

Now define the complex quantity

$$H(\Omega) = \mathcal{C}(\Omega) - j\mathcal{S}(\Omega) = |H(\Omega)|.\exp\{j\angle H(\Omega)\}\ ,$$

The result in Equation is fundamental and important! It states that the entire effect of an LTI system on a sinusoidal input at frequency Ω0 can be deduced from the (complex) frequency response evaluated at the frequency Ω0. The amplitude or magnitude of the sinusoidal input gets scaled by the magnitude of the frequency response at the input frequency, and the phase gets augmented by the angle or phase of the frequency response at this frequency.

Now consider the same calculation as earlier, but this time with complex exponentials. Suppose

$$x[n] = A_0 e^{j(\Omega_0 n + \theta_0)} \quad \text{for all } n \,.$$

eq 7.9

Convolution then yields

$$\begin{aligned} y[n] &= \sum_{m=-\infty}^{\infty} h[m]x[n-m] \\ &= \sum_{m=-\infty}^{\infty} h[m]A_0 e^{j\left(\Omega_0(n-m)+\theta_0\right)} \\ &= \left(\sum_{m=-\infty}^{\infty} h[m]e^{-j\Omega_0 m}\right) A_0 e^{j(\Omega_0 n+\theta_0)} \,. \end{aligned}$$

eq 7.10

Thus the output of the system, when the input is the (everlasting) exponential in Equation (7.9), is the same exponential, except multiplied by the following quantity evaluated at Ω=Ω0:

$$\sum_{m=-\infty}^{\infty} h[m]e^{-j\Omega m} = \mathcal{C}(\Omega) - j\mathcal{S}(\Omega) = H(\Omega) \,.$$

eq 7.11

Although we have introduced the notion of a frequency response in the context of what an LTI system does to a single sinusoidal input, superposition will now allow us to use the frequency response to describe what an LTI system does to any input made up of a linear combination of sinusoids at different frequencies. You compute the (sinusoidal) response to each sinusoid in the input, using the frequency response at the frequency of that sinusoid. The system output will then be the same linear combination of the individual sinusoidal responses. As we shall see in the next chapter, when we use Fourier analysis to introduce the notion of the spectral content or frequency content of a signal, the class of signals that can be represented as a linear combination of sinusoids at assorted frequencies is very large. So this superposition idea ends up being extremely powerful.

Summary of Study Session 7

Self-Assessment Question (SAQs) for lecture 7

SAQ 7.1

A student designs a simple causal LTI system characterized by the following unit sample response:

$h[0] = 1$

$h[1] = 2$

$h[2] = 1$

$h[n]=0 \ \forall n > 2$

(a) What is the frequency response, $H(\Omega)$?

(b) What is the magnitude of H at $\Omega=0,\pi/2, \pi$?

(c) If this LTI system is used as a filter, what is the set of frequencies that are removed?

SAQ 7.2

A wireline channel has unit sample response $h1[n] = e−an$ for $n \geq 0$, and 0 otherwise, where $a > 0$ is a real number. (As an aside, $a = Ts/\tau$, where Ts is the sampling rate and τ is the wire time constant. The wire resistance and capacitance prevent fast changes at the end of the wire regardless of how fast the input is changing. We capture this decay in time with exponential unit sample response e−an). A student who recently got a job at WireSpeed Inc., is trying to convince his manager that he can significantly improve the signaling speed (and hence transfer the bits faster) over this wireline channel, by placing a filter with unit sample response

$h2[n] = A\delta[n] + B\delta[n - D]$, at the receiver, so that $(h1 * h2)[n] = \delta[n]$.

(a) Derive the values of A, B and D that satisfy Ben's goal.

(b) Sketch the frequency response of H2(Ω) and mark the values at 0 and ±π.

(c) Suppose a = 0.1. Then, does H2(Ω) behave like a (1) low-pass filter, (2) highpass filter, (3) all-pass filter? Explain your answer.

Study Session 8: Spectral Representation of Signals

8.1 Introduction

The process of (electronic) communication involves the generation, transmission, and reception of various types of signals. The communication process becomes fairly difficult, because:

a) the transmitted signals may have to travel long distances (there by undergoing severe attenuation) before they can reach the destination i.e., the receiver.

b) of imperfections of the channel over which the signals have to travel

c) of interference due to other signals sharing the same channel and

d) of noise at the receiver input.

In quite a few situations, the desired signal strength at the receiver input may not be significantly stronger than the disturbance component present at that point in the communication chain. (But for the above causes, the process of communication would have been quite easy, if not trivial). In order to come up with appropriate signal processing techniques, which enable us to extract the desired signal from a distorted and noisy version of the transmitted signal, we must clearly understand the nature and properties of the desired and undesired signals present at various stages of a communication system by studying the aspect of communication theory

Signals physically exist in the time domain and are usually expressed as a function of the time parameter Because of this feature, it is not too difficult, at least in the majority of the situations of interest to us, to visualize the signal behaviour in the **Time Domain**. In fact, it may even be possible to view the signals on an **oscilloscope.** But equally important is the characterization of the signals in **the Frequency Domain or Spectral Domain**. That is, we characterize the signal in terms of its various frequency components (or its spectrum). Fourier analysis (Fourier Series and Fourier Transform) helps us in arriving at the spectral description of the pertinent signals.

-8.1.1 Periodic Signals and Fourier Series

Signals can be classified in various ways such as:

a) Power or Energy

b) Deterministic or Random

c) Real or Complex

d) Periodic or Aperiodic etc.

Our immediate concern is with periodic signals and how to develop the spectral description of these signal.

8.2 Signal Spectra

By Fourier theory, any waveform can be represented by a summation of a (possibly infinite) number of sinusoids, each with a particular amplitude and phase. Such a representation is referred to as the signal's spectrum (or it's frequency-domain representation). It is often easier to analyze signals and signal networks in terms of their spectral representations. As well, there are many instances where it is easier to synthesize a signal using a frequency-domain approach.

8.2.1 Why is the spectral view useful?

It is often the case that the spectrum of a signal can indicate aspects of the signal that would otherwise not be obvious by looking only at its time-domain representation.

8.2.2 How is a spectrum obtained?

- Given a discrete-time signal x[n], we can determine its frequency response using the Discrete Fourier Transform (DFT):

$$X[k] \triangleq \sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N}, \qquad k = 0, 1, 2, \ldots, N-1.$$

- The DFT determines sinusoidal ``weights'' via the inner product of sinusoids and the signal.

- The DFT can be interpreted as the sum of projections of x[n] onto a set of k sampled complex sinusoids or sinusoidal basis functions at (normalized) radian frequencies given by $\omega_k = 2\pi k/N$
- In this way, the DFT and its inverse provide a ``recipe'' for reconstructing a given discrete-time signal in terms of sampled complex sinusoids.

- If the signal x[n] consists of N samples, X[k] will consist of k=N frequency weights (assuming no zero-padding). Based on the sampling theorem, however, only the first half of these frequency components are unique.

- The DFT coefficients are complex values. To plot the magnitude response of a signal's spectrum, we calculate the magnitude of each coefficient.

- The phase response of a signal is given by the ``angles'' of its complex DFT coefficients.

The action of an LTI system on a sinusoidal or complex exponential input signal can be represented effectively by the frequency response H(Ω) of the system. By superposition, it then becomes easy again using the frequency response to determine the action of an LTI system on a weighted linear combination of sinusoids or complex exponentials. The natural question now is how large a class of signals can be represented in this manner. The short answer to this question: most signals you are likely to be interested in! The tool for exposing the decomposition of a signal into a weighted sum of sinusoids or complex exponentials is **Fourier analysis**. The Discrete-Time Fourier Transform (DTFT), which we have actually seen hints of already and which applies to the most general classes of signals. The Discrete-Time Fourier Series (DTFS), which constructs a similar representation for the special case of periodic signals, or for signals of finite duration. The DTFT development provides some useful background, context and intuition for the more special DTFS development, but may be skimmed over on an initial reading (i.e., understand the logical flow of the development, but don't struggle too much with the mathematical details).

8.3 The Discrete-Time Fourier Transform

We have in fact already derived an expression in the previous chapter that has the flavor of what we are looking for. Recall that we obtained the following representation for the unit sample response h[n] of an LTI system:

$$h[n] = \frac{1}{2\pi} \int_{<2\pi>} H(\Omega) e^{j\Omega n} \, d\Omega \, .$$

eq. 8.1

where the frequency response, H(Ω), was defined by

$$H(\Omega) = \sum_{m=-\infty}^{\infty} h[m] e^{-j\Omega m} \, .$$

eq. 8.2

Equation (8.1) can be interpreted as representing the signal h[n] by a weighted combination of a continuum of exponentials, of the form $e^{j\Omega n}$, with frequencies Ω in a 2π-range, and associated weights H(Ω) dΩ. As far as these expressions are concerned, the signal h[n] is fairly arbitrary; the fact that we were considering it as the unit sample response of a system was quite incidental. We only required it to be a signal for which the infinite sum on the right of Equation (8.2) was well-defined. We shall accordingly rewrite the preceding equations in a more neutral notation, using x[n] instead of h[n]:

$$x[n] = \frac{1}{2\pi} \int_{<2\pi>} X(\Omega) e^{j\Omega n} \, d\Omega \, ,$$

eq. 8.3

where X(Ω) is defined by

$$X(\Omega) = \sum_{m=-\infty}^{\infty} x[m]e^{-j\Omega m} \, .$$

eq. 8.4

For a general signal x[·], we refer to the 2π-periodic quantity X(Ω) as the discrete-time Fourier transform (DTFT) of x[·]; it would no longer make sense to call it a frequency response. Even when the signal is real, the DTFT will in general be complex at each Ω. The DTFT synthesis equation, Equation (8.3), shows how to synthesize x[n] as a weighted combination of a continuum of exponentials, of the form ejΩn, with frequencies Ω in a 2π-range, and associated weights X(Ω) dΩ. From now on, unless mentioned otherwise, we shall take Ω to lie in the range [−π,π]. The DTFT analysis equation, Equation (8.4), shows how the weights are determined. We also refer to X(Ω) as the spectrum or spectral distribution or spectral content of x[·].

Example 1 (Spectrum of Unit Sample Function) Consider the signal x[n] = δ[n], the unit sample function. From the definition in Equation (13.4), the spectral distribution is given by X(Ω) = 1, because x[n]=0 for all n = 0, and x[0] = 1. The spectral distribution is thus constant at the value 1 in the entire frequency range [−π,π]. What this means is that it takes the addition of equal amounts of complex exponentials at all frequencies in a 2π-range to synthesize a unit sample function, a perhaps surprising result. What's happening here is that all the complex exponentials reinforce each other at time n = 0, but effectively cancel each other out at every other time instant.

Example 2 (Phase Matters) What if X(Ω) has the same magnitude as in the previous example, so |X(Ω)| = 1, but has a nonzero phase characteristic, ∠X(Ω) = −αΩ for some α = 0? This phase characteristic is linear in Ω. With this,

$$X(\Omega) = 1.e^{-j\alpha\Omega} = e^{-j\alpha\Omega} \, .$$

To find the corresponding time signal, we simply carry out the integration in Equation (8.3). If α is an integer, the integral

$$x[n] = \frac{1}{2\pi}\int_{<2\pi>} e^{-j\alpha\Omega} e^{j\Omega n}\, d\Omega = \frac{1}{2\pi}\int_{<2\pi>} e^{j(n-\alpha)\Omega}\, d\Omega$$

yields the value 0 for all $n \neq \alpha$. To see this, note that

$$e^{j(n-\alpha)\Omega} = \cos\Big((n-\alpha)\Omega\Big) + j\sin\Big((n-\alpha)\Omega\Big) ,$$

and the integral of this expression over any 2π-interval is 0, when n − α is a nonzero integer. However, if n − α = 0, i.e., if n = α, the cosine evaluates to 1, the sine evaluates to 0, and the integral above evaluates to 1. We therefore conclude that when α is an integer,

$$x[n] = \delta[n - \alpha] .$$

The signal is just a shifted unit sample (delayed by α if α > 0, and advanced by |α| otherwise). The effect of adding the phase characteristic to the case in Example 1 has been to just shift the unit sample in time. For non-integer α, the answer is a little more intricate:

$$\begin{aligned} x[n] &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{-j\alpha\Omega} e^{j\Omega n}\, d\Omega \\ &= \frac{1}{2\pi}\frac{e^{j(n-\alpha)\Omega}}{j(n-\alpha)}\Big|_{-\pi}^{\pi} \\ &= \frac{\sin\Big(\pi(n-\alpha)\Big)}{\pi(n-\alpha)} \end{aligned}$$

This time-function is referred to as a "sinc" function.

Example 3 (A Bandlimited Signal) Consider now a signal whose spectrum is flat but band-limited:

$$X(\Omega) = \begin{cases} 1 & \text{for} \quad |\Omega| < \Omega_c \\ 0 & \text{for} \quad \Omega_c \leq |\Omega| \leq \pi \end{cases}$$

The corresponding signal is again found directly from Equation (8.3). For n = 0, we get

$$\begin{aligned} x[n] &= \frac{1}{2\pi}\int_{-\Omega_c}^{\Omega_c} e^{j\Omega n}\, d\Omega \\ &= \frac{1}{2\pi}\frac{e^{j\Omega}}{jn}\Big|_{-\Omega_c}^{\Omega_c} \\ &= \frac{\sin(\Omega_c n)}{\pi n} , \end{aligned}$$

which is again a sinc function. For n = 0, Equation (8.3) yields

$$x[n] = \frac{1}{2\pi}\int_{-\Omega}^{\Omega_c} 1\, d\Omega = \frac{\Omega_c}{\pi} .$$

(This is exactly what we would get from Equation (8.5) if n was treated as a continuous variable, and the limit of the sinc function as n → 0 was evaluated by L'Hopital's rule—a useful mnemonic, but not a derivation!)

The sinc function in the examples above is actually not absolutely summable because it follows off too slowly—only as 1/n—as |n|→∞. However, it is square summable. A digression: One can also define the DTFT for signals x[n] that do not converge to 0 as |n|→∞, provided they grow no faster than polynomial in n as |n|→∞. An example of such a signal of slow growth would be x[n] = $e^{j\Omega 0n}$ for all n, whose spectrum must be concentrated at $\Omega=\Omega_0$. However, the corresponding X(Ω) turns out to no longer be an ordinary function, but is a (scaled) Dirac impulse in frequency, located at $\Omega=\Omega_0$:

$X(\Omega) = 2\pi\delta(\Omega - \Omega_0)$

You may have encountered the Dirac impulse in other settings. The unit impulse at $\Omega=\Omega_0$ can be thought of as a "function" that has the value 0 at all points except at $\Omega=\Omega_0$, and has unit area. This is an instance of a broader result, namely that signals of slow growth possess transforms that are generalized functions (e.g., impulses), which have to be interpreted in terms of what they do under an integral sign, rather than as ordinary functions. It is partly in order to avoid having to deal with impulses and generalized functions in treating sinusoidal and periodic signals that we shall turn to the **Discrete-Time Fourier Series** rather than the DTFT. If the input x[n] to an LTI system with frequency response H(Ω) is the (everlasting) exponential signal ejΩn, then the output is y[n] = H(Ω)ejΩn. By superposition, if the input is instead the weighted linear combination of such exponentials that is given in Equation (8.3), then the corresponding output must be the same weighted combination of responses, so

$$y[n] = \frac{1}{2\pi}\int_{<2\pi>} H(\Omega)X(\Omega)e^{j\Omega n}\, d\Omega .$$

eq 8.6

However, we also know that the term $H(\Omega)X(\Omega)$ multiplying the complex exponential in this expression must be the DTFT of $y[\cdot]$, so

$$Y(\Omega) = H(\Omega)X(\Omega). \qquad \text{eq. 8.7}$$

Thus, the time-domain convolution relation $y[n] = (h * x)[n]$ has been converted to a simple multiplication in the frequency domain. This is a result we saw in the previous chapter too, when discussing the frequency response of a series or cascade combination of two LTI systems: the relation $h[n] = (h1 * h2)[n]$ in the time domain mapped to an overall frequency response of $H(\Omega) = H1(\Omega)H2(\Omega)$ that was simply the product of the individual frequency responses. This is a major reason for the power of frequency-domain analysis; the more involved operation of convolution in time is replaced by multiplication in frequency.

8.4 The Discrete-Time Fourier Series

The DTFT synthesis expression in Equation (13.3) expressed $x[n]$ as a weighted sum of a continuum of complex exponentials, involving all frequencies Ω in $[-\pi,\pi]$. Suppose now that $x[n]$ is a periodic signal of (integer) period P, so

$$x[n + P] = x[n]$$

for all n. This signal is completely specified by the P values it takes in a single period, for instance the values $x[0], x[1],\ldots,x[P - 1]$. It would seem in this case as though we should be able to get away with using a smaller number of complex exponentials to construct $x[n]$ on the interval $[0, P - 1]$ and thereby for all n. The discrete-time Fourier series (DTFS) shows that this is indeed the case. Before we write down the DTFS, a few words of reassurance are warranted. The expressions below may seem somewhat bewildering at first, with a profusion of symbols and subscripts, but once we get comfortable with what the expressions are saying, interpret them in different ways, and do some examples, they end up being quite straightforward. So, don't worry if you don't get it all during the first pass through this material—allow yourself some time, and a few visits, to get comfortable!

8.4.1 The Synthesis Equation

The essence of the DTFS is the following statement:

Any P-periodic signal x[n] can be represented (or synthesized) as a weighted linear combination of P complex exponentials (or spectral components), where the frequencies of the exponentials are located evenly in the interval [−π,π], starting in the middle at the frequency Ω0 = 0 and increasing outwards in both directions in steps of

$$\Omega_1 = 2\pi/P.$$

More concretely, the claim is that any P-periodic DT signal x[n] can be represented in the form

$$x[n] = \sum_{k=\langle P\rangle} A_k e^{j\Omega_k n} ,$$

eq 8.8

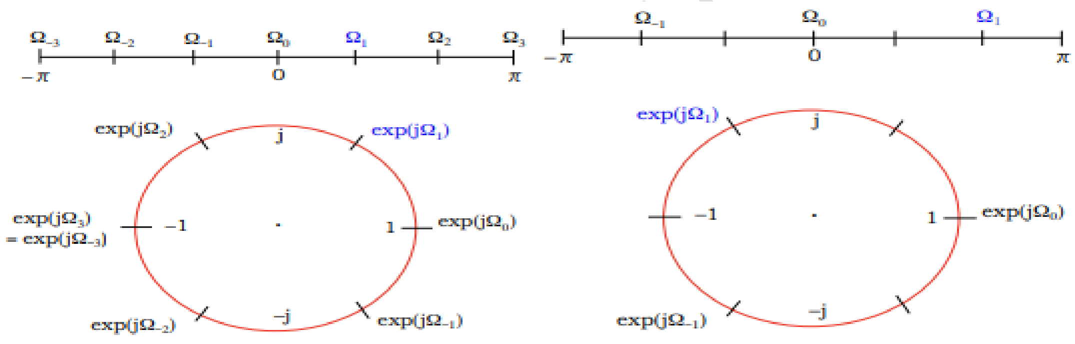


Figure 8-1: When P is even, the end frequencies are at ±π and the Ω_k values are as shown in the pictures on the left for P = 6. When P is odd, the end frequencies are at ±(π − Ω 2 1), as shown on the right for P = 3.

where we write k = (P) to indicate that k runs over any set of P consecutive integers. The Fourier series coefficients or spectral weights A_k in this expression are complex numbers in general, and the spectral frequencies Ω_k are defined by

$$\Omega_k = k\Omega_1 , \quad \text{where} \quad \Omega_1 = \frac{2\pi}{P} .$$

eq. 8.9

We refer to Ω_1 as the fundamental frequency of the periodic signal, and to Ω_k as the k-th harmonic. Note that $\Omega_0 = 0$.

Note that the expression on the right side of Equation (8.8) does indeed repeat periodically every P time steps, because each of the constituent exponentials

$$e^{j\Omega_k n} = e^{jk\Omega_1 n} = e^{jkn(2\pi/P)} = \cos(k\frac{2\pi}{P}n) + j\sin(k\frac{2\pi}{P}n)$$

eq. 8.10

repeats when n changes by an integer multiple of P.

It also follows from Equation (8.10) that changing the frequency index k by P — or more generally by any positive or negative integer multiple of P — brings the exponential in that equation back to the same point on the unit circle, because the corresponding frequency Ω_k has then changed by an integer multiple of 2π. This is why it suffices to choose k = (P) in the DTFS representation.

Putting all this together, it follows that the frequencies of the complex exponentials used to synthesize a P-periodic signal x[n] via the DTFS are located evenly in the interval[−π,π], starting in the middle at the frequency Ω_0 = 0 and increasing outwards in both directions in steps of Ω_1 = 2π/P. In the case of an even value of P, such as the case P = 6 in Figure 8-1 (left), the end frequencies will be at ±π (we need only one of these frequencies, or both, as they translate to the same point on the unit circle when we write $e^{j\Omega kn}$). In the case of an odd value of P, such as the case P = 3 shown in Figure 8-1 (right), the end points are ± (π − Ω_1/2). The weights {Ak} collectively constitute the spectrum of the periodic signal, and we typically plot them as a function of the frequency index k, as in Figure 8-2. The spectral weights in these simple sinusoidal examples have been determined by inspection, through direct application of Euler's identity. We turn next to a more general and systematic way of determining the spectrum for an arbitrary real P-periodic signal.

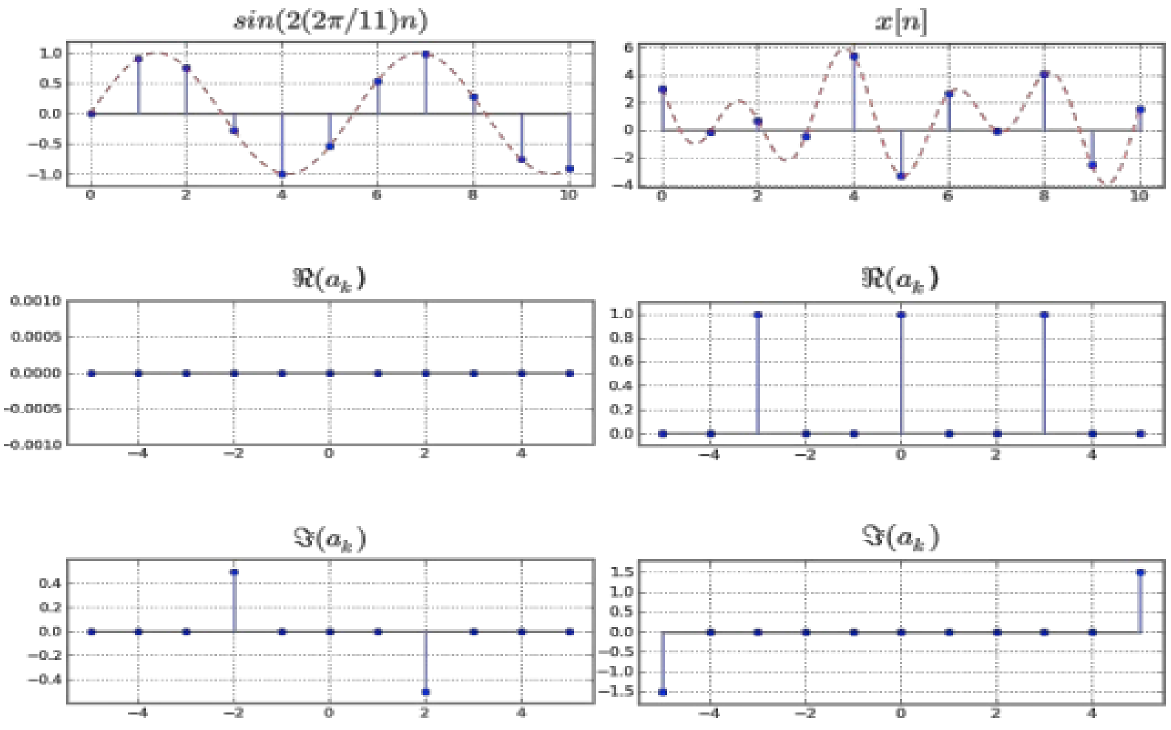


Figure 8-2: The spectrum of two periodic signals, plotted as a function of the frequency index, k, showing the real and imaginary parts for each case. P = 11 (odd).

8.2.2 The Analysis Equation

We now address the task of computing the spectrum of a P-periodic x[n], i.e., determining the Fourier coefficients Ak. Note first that the {Ak} comprise P coefficients that in general can be complex numbers, so in principle we have 2P real numbers that we can choose to match the P real values that a P-periodic real signal x[n] takes in a period. It would therefore seem that we have more than enough degrees of freedom to choose the Fourier coefficients to match a P-periodic real signal. (If the signal x[n] was an arbitrary complex P-periodic signal, hence specified by 2P real numbers, we would have exactly the right number of degrees of freedom.) It turns out—and we shall prove this shortly—that for a real signal x[n] the Fourier coefficients satisfy certain symmetry properties, which end up reducing our degrees of freedom to precisely P rather than 2P. Specifically, we can show that

$$A_k = A^*_{-k} \ , \qquad \text{eq. 8.11}$$

so, the real part of Ak is an even function of k, while the imaginary part of A_k is an odd function of k. This also implies that A0 is purely real, and also that in the case of even P, the values $A_{P/2} = A_{-P/2}$ are purely real. Making a careful count now of the actual degrees of freedom, we find that it takes precisely P real parameters to specify the spectrum {Ak} for a real P-periodic signal. So, given the P real values that x[n] takes over a single period, we expect that Equation (8.8) will give us precisely P equations in P unknowns. (For the case of a complex signal, we will get 2P equations in 2P unknowns.) To determine the mth Fourier coefficient A_m in the expression in Equation (8.8), where m is one of the values that k can take, we first multiply both sides of Equation (8.8) by $e^{-j\Omega mn}$ and sum over P consecutive values of n. This results in the equality

$$\begin{aligned}
\sum_{n=\langle P\rangle} x[n]e^{-j\Omega_m n} &= \sum_{n=\langle P\rangle}\sum_{k=\langle P\rangle} A_k e^{j(\Omega_k-\Omega_m)n} \\
&= \sum_{k=\langle P\rangle} A_k \sum_{n=\langle P\rangle} e^{j\Omega_1(k-m)n} \\
&= \sum_{k=\langle P\rangle} A_k \sum_{n=\langle P\rangle} e^{j2\pi(k-m)n/P} .
\end{aligned}$$

The summation over n in the last equality involves summing P consecutive terms of a geometric series. Using the fact that for $r = 1$

$$1 + r + r^2 + \cdots + r^{P-1} = \frac{1-r^P}{1-r} ,$$

it is not hard to show that the above summation over n ends up evaluating to 0 for $k = m$. The only value of k for which the summation over n survives is the case $k = m$, for which each term in the summation reduces to 1, and the sum ends up equal to P. We therefore arrive at

$$\sum_{n=\langle P\rangle} x[n]e^{-j\Omega_m n} = A_m P$$

or, rearranging and going back to writing k instead of m,

$$A_k = \frac{1}{P}\sum_{n=\langle P\rangle} x[n]e^{-j\Omega_k n} .$$

eq. 8.12

This DTFS analysis equation which holds whether x[n] is real or complex looks very similar to the DTFS synthesis equation, Equation (8.8), apart from $e^{-j\Omega kn}$ replacing $e^{j\Omega kn}$, and the scaling by P.

Two particular observations that follow directly from the analysis formula:

$$A_0 = \frac{1}{P} \sum_{n=\langle P \rangle} x[n] \, ,$$

eq. 8.13

and, for the case of even P, where $\Omega_{P/2} = \pi$,

$$A_{P/2} = A_{-P/2} = \frac{1}{P} \sum_{n=\langle P \rangle} (-1)^n x[n] \, .$$

eq. 8.14

The symmetry properties of A_k that we stated earlier in the case of a real signal follow directly from this analysis equation, as we leave you to verify. Also, since $A_{-k} = A_k^*$ for a real signal, we can combine the terms

$$A_k e^{-j\Omega_k n} + A_k e^{j\Omega_k n}$$

into the single term

$$2|A_k| \cos(\Omega_k n + \angle A_k) \, .$$

Thus, for even P,

$$x[n] = A_0 + \sum_{k=1}^{P/2} 2|A_k| \cos(\Omega_k n + \angle A_k) \, ,$$

while for odd P the only change is that the upper limit becomes $(P-1)/2$.

8.2.4 Action of an LTI System on a Periodic Input

Suppose the input x[·] to an LTI system with frequency response H(Ω) is P-periodic. This signal can be represented as a weighted sum of exponentials, by the DTFS in Equation

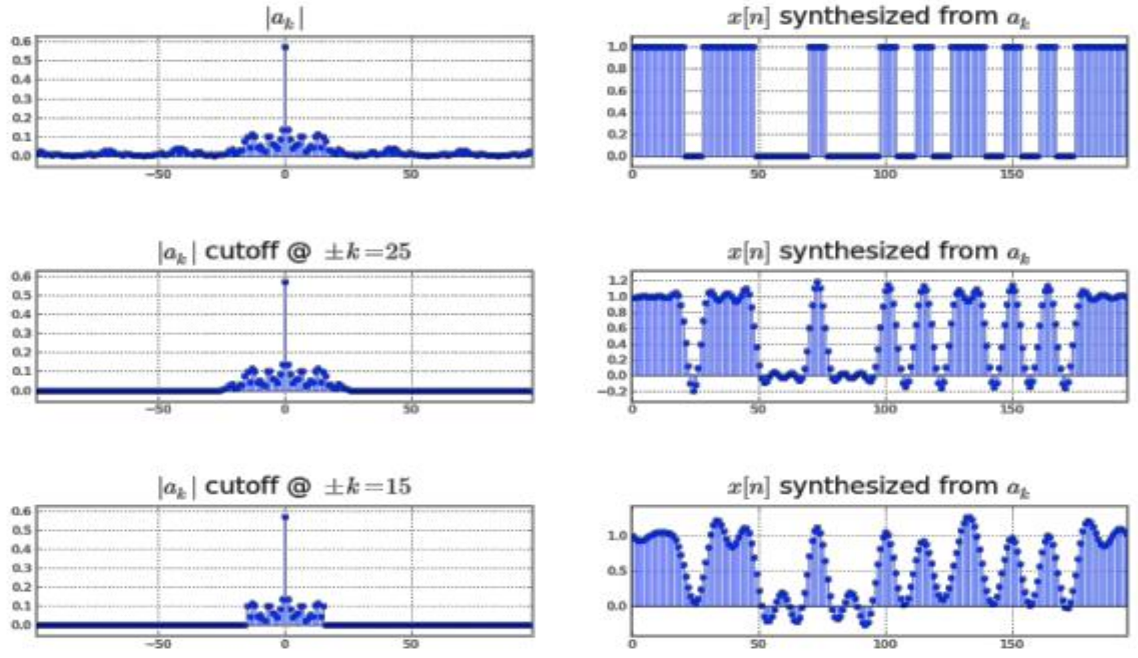


Figure 8-3: Effect of band-limiting a transmission, showing what happens when a periodic signal goes through a lowpass filter.

(8.8). It follows immediately that the output of the system is given by

$$y[n] = \sum_{k=\langle P\rangle} H(\Omega_k)A_k e^{j\Omega_k n} = \frac{1}{P}\sum_{k=\langle P\rangle} H(\Omega_k)X_k e^{j\Omega_k n} .$$

This immediately shows that the output y[·] is again P-periodic, with (scaled) spectral coefficients given by

$$Y_k = H(\Omega_k)X_k \qquad \text{eq.8.17}$$

So, knowledge of the input spectrum and of the system's frequency response suffices to determine the output spectrum. This is precisely the DTFS version of the DTFT result in Equation (8.7). As an illustration of the application of this result, Figure 8-3 shows what happens when a periodic signal goes through an ideal lowpass filter, for which H(Ω) = 1 only for $|\Omega| < \Omega_c < \pi$, with H(Ω) = 0 everywhere else in [−π,π]. The result is that all spectral components of the input at frequencies above the cutoff frequency Ωc are no longer present in the output. The corresponding output signal is thus more slowly varying—a “blurred” version of the input because it does not have the higher-frequency components that allow it to vary more rapidly.

8.2.5 Application of the DTFS to Finite-Duration Signals

The DTFS turns out to be useful in settings that do not involve periodic signals, but rather signals of finite duration. Suppose a signal x[n] takes nonzero values only on some finite interval, say [0, P − 1] for example. We are not forbidding x[n] from taking the value 0 for n within this interval, but are saying that x[n]=0 for all n outside this interval. If we now compute the DT Fourier transform of this signal, according to the definition in Equation (8.4), we get

$$X(\Omega) = \sum_{n=0}^{P-1} x[n] e^{-j\Omega n} \,.$$

eq. 8.18

The corresponding representation of x[n] by a weighted combination of complex exponentials would then be the expression in Equation (8.3), involving a continuum of frequencies. However, it is possible to get a more economical representation of x[n] by using the DT Fourier series. In order to do this, consider the new signal $_{xP}$ [·] obtained by taking the portion of x[·] that lies in the interval [0, P − 1] and replicating it periodically outside this interval, with period P. This results in $_{xP}$ [n + P] = $_{xP}$ [n] for all n, with $_{xP}$ [n] = x[n] for n in the interval [0, P − 1]. We can represent this periodic signal by its DTFS:

$$x_P[n] = \frac{1}{P} \sum_{k=<P>} X_k e^{j\Omega_k n} \,,$$

eq. 8.19

Where

$$X_k = \sum_{n=<P>} x_P[n] e^{-j\Omega_k n} = \sum_{n=0}^{P-1} x[n] e^{-j\Omega_k n}$$

eq. 8.20

(For consistency, we should perhaps have used the notation X_{Pk} instead of X_k, but we are trying to keep our notation uncluttered.) We can now represent x[n] by the expression in Equation

(8.19), in terms of just P complex exponentials at the frequencies Ω_k defined earlier (in our development of the DTFS), rather than complex exponentials at a continuum of frequencies. However, this representation only captures x[n] in the interval [0, P − 1]. Outside of this interval, we have to ignore the expression, instead invoking our knowledge that x[n] is actually 0 outside. Another observation worth making from Equations (8.18) and (8.20) is that the (scaled) DTFS coefficients X_k are actually simply related to the DTFT X(Ω) of the finite duration signal x[n]:

$$X_k = X(\Omega_k) \qquad \text{eq.8.21}$$

so the (scaled) DTFS coefficients X_k are just P samples of the DTFT X(Ω). Thus, any method for computing the DTFS for (the periodic extension of) a finite duration signal will yield samples of the DTFT of this finite-duration signal (keep track of our use of DTFS versus DTFT here). And if one wants to evaluate the DTFT of this finite-duration signal at a larger number of sample points, all that needs to be done is to consider x[n] to be of finite duration on a larger interval, of length $P^i > P$, where of course the additional signal values in the larger interval will all be 0; this is referred to a zero-padding. Then computing the DTFS of (the periodic extension of) x[n] for this longer interval will yield P^i samples of the underlying DTFT of the signal. As an application of the above results on finite-duration signals, consider the case of an LTI system whose unit sample response h[n] is known to be 0 for all n outside of some interval [0, n_h], and whose input x[n] is known to be 0 for all n outside some interval [0, n_x]. It follows that the earliest time instant at which a nonzero output value can appear is n = 0, while the latest such time instant is $n = n_x + n_h$. In other words, the response $y[n]= (h * x) [n]$ is guaranteed to be 0 for all n outside of the interval [0, $n_x + n_h$]. All the interesting input/output action of the system therefore takes place for n in this interval. Outside of this interval we know that x[·] and y[·] are both 0. We can therefore take all the signals of interest to have finite duration, being 0 outside of the interval [0, P − 1], where $P = n_x + n_h + 1$. A DTFS representation of x[·] and y[·] on this interval, with this choice of P, can then be used to carry out a frequency-domain analysis of the system. In particular, the kth (scaled) Fourier coefficients of the input and output will be related as in Equation (8.17).

8.2.6 The FFT

Implementing either the DTFS synthesis computation or the DTFS analysis computation, as defined earlier, would seem to require on the order of P^2 multiply/add operations: we have to do P multiply/adds for each of P frequencies. This can quickly lead to prohibitively expensive computations in large problems. Happily, in 1965 Cooley and Tukey published a fast method for computing these DTFS expressions (rediscovering a technique known to Gauss!). Their algorithm is termed the Fast Fourier Transform or FFT, and takes on the order of P log P operations, which is a big saving. (Note that the FFT is not a new kind of transform, despite its name it's a fast algorithm for computing a familiar transform, namely the DTFS.)

The essence of the idea is to recursively split the computation into a DTFS computation involving the signal values at the even time instants and another DTFS computation involving the signal values at the odd time instants. One then cleverly uses the nice algebraic properties of the P complex exponentials involved in these computations to stitch things back together and obtain the desired DTFS. The FFT has become a (or maybe the) workhorse of practical numerical computation. Its most common application is to computing samples of the DTFT of finite-duration signals, as described in the previous subsection. It can also be applied, of course, to computing the DTFS of a periodic signal.

Summary of Study Session 8

Self-Assessment Question (SAQs) for lecture 8

SAQ 8.1

Let x[·] be a signal that is periodic with period P = 12. For each of the following x[.], give the corresponding spectral coefficients Ak for the discrete-time Fourier series for x[·], for k in the range $-6 \leq k \leq 6$. (Hint: In most of the following cases, all you need to do is express the signal as the sum of appropriate complex exponentials, by inspection—this is much easier than cranking through the formal definition of the spectral coefficient.)

(a) Determine A_k when x[0 : 11] = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0].

(b) Determine A_k when x[n]=1 for all n.

(c) Determine A_k when x[n] = sin(r(2π/12)n) for the following two choices of r:

i. r = 3; and

ii. r = 8.

(d) Determine A_k when x[n] = sin(3(2π/12)n + ϕ) where ϕ is some specified phase offset.

SAQ 8.2

Consider an audio channel with a sampling rate of 8000 samples/second.

(a) What is the angular frequency of the piano note A (in radians/sample), given that its continuous time frequency is 880 Hz?

(b) What is the smallest number of samples, P, needed to represent the note A as a 2π spectral component at Ω_k =2π/p k, for integer k? And what is the value of k?

Study Session 9: Modulation/Demodulation

9.1 Introduction

The purpose of a communication system is to transfer information from a source to a destination. To transmit data over a communication link, which can be achieved by mapping the bit stream we wish to transmit onto analog signals because most communication links, at the lowest layer, are able to transmit analog signals, not binary digits. The signals that most simply and directly represent the bit stream are called the **baseband signals**.

In practice, problems arise in baseband transmissions, the major cases being:

- Noise in the system – external noise and circuit noise reduces the signal-to-noise (S/N) ratio at the receiver (Rx) input and hence reduces the quality of the output.
- Such a system is not able to fully utilise the available bandwidth, for example, telephone quality speech has a bandwidth $\simeq$ 3kHz, a co-axial cable has a bandwidth of 100's of Mhz.
- Radio systems operating at baseband frequencies are very difficult.
- Not easy to network.

9.1.1 What is Modulation?

In modulation, a message signal, which contains the information is used to control the parameters of a carrier signal, so as to impress the information onto the carrier.

The Messages

The message or modulating signal may be either:

analogue – denoted by m(t)

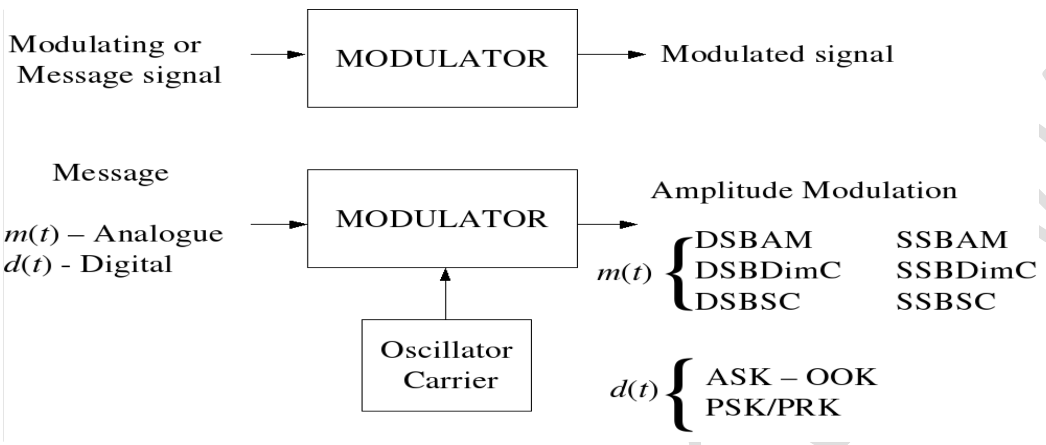
digital – denoted by d(t) – i.e. sequences of 1's and 0's

The message signal could also be a multilevel signal, rather than binary

The Carrier

The carrier could be a 'sine wave' or a 'pulse train'.

- If the message signal m(t) controls amplitude – gives **AMPLITUDE MODULATION AM**
- If the message signal m(t) controls frequency – gives **FREQUENCY MODULATION FM**
- If the message signal m(t) controls phase- gives **PHASE MODULATION PM**



9.1.2 Why Modulation?

There are two principal motivating reasons for modulation: matching the transmission characteristics of the medium, and considerations of power and antenna size, which impact portability. The second is the desire to multiplex, or share, a communication medium among many concurrently active users.

9.1.3 What is Demodulation?

Demodulation is the reverse process (to modulation) to recover the message signal m(t) or d(t) at the receiver.

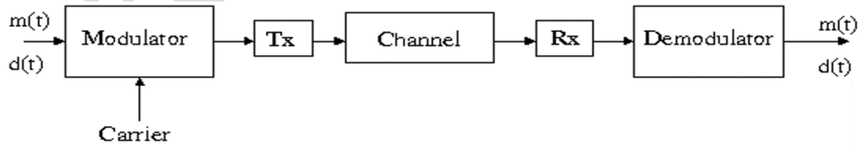


Figure 9 -2

Demodulation is the process by which the original information bearing signal, i.e. the modulation is extracted from the incoming overall received signal. The process of demodulation for signals using amplitude modulation can be achieved in a number of different techniques, each of which has its own advantage. The demodulator is the circuit, or for a software defined radio, the software that is used to recover the information content from the overall incoming modulated signal.

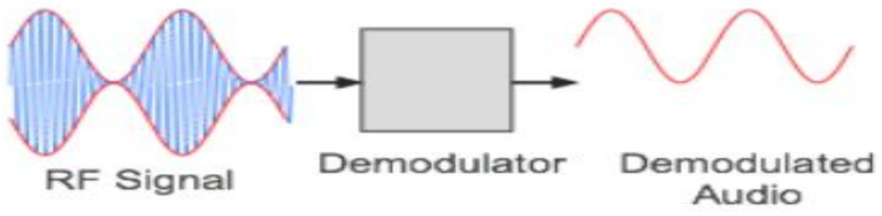


Figure 9-3

9.2 CW – Contention Window

This is a term used in IEEE (Institute of Electrical and Electronics Engineers) 802.11 networks which are supporting the QoS (Quality of Service) enhancements originally defined in the 802.11e standard. It defines a period of time in which the network is operating in contention mode. The IEEE 802.11 standard defines a detailed medium access control (MAC) and physical layer (PHY) specification for wireless local area networks (WLANs). WLANs are growing in popularity because of the convenience offered in terms of supporting mobility while providing flexibility. In the IEEE 802.11 MAC layer protocol, the basic access method is the distributed coordination function (DCF) which is based on the mechanism of carrier sense multiple access with collision avoidance (CSMA/CA). The standard also defines an optional point coordination function (PCF), which is a centralized MAC protocol supporting collision free and time bounded services. In this paper, we limit our interest to DCF. The DCF has two schemes for packet transmission. The basic scheme is a two-way handshaking technique. In this scheme, if a station has a packet to transmit, it waits for a DIFS idle duration of the medium and then transmits its packet. When the packet is received successfully, the destination station sends a positive acknowledgement (ACK) to the sending station after a short interframe space (SIFS). The second scheme is based on four-way handshaking that avoids the "hidden terminals" problem. In this scheme, whenever a packet is to be transmitted, the transmitting station first sends out a short request-to-send (RTS) packet containing information on the length of the packet. If the receiving station hears the RTS, it responds with a short clear-to-send (CTS) packet. After this exchange, the transmitting station sends its packet. When the packet is received successfully, the thereceiving station transmits an ACK packet. This back-and-forth exchange is necessary to avoid the "hidden terminals" problem. In this paper, we assume that each station can hear each other so that there is no hidden terminal problem in the system and we need only focus on the basic CSMA/CA.

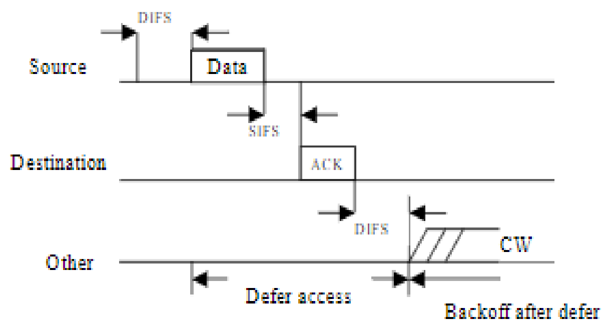


Figure 9-4

the protocol works on a “listen before talk” scheme. To transmit a packet, a station must sense the medium and must ensure that the medium is idle for the specified DCF interframe space (DIFS) duration before transmitting. If a station having a packet to transmit initially senses the medium to be busy; then the station waits until the medium becomes idle for DIFS period, and then chooses a random "backoff counter" which determines the amount of time the station must wait until it is allowed to transmit. During the period in which the medium is idle, the transmitting station decreases its backoff counter. (When the medium becomes busy, its backoff counter is frozen. It can decrease its backoff counter again only after the medium is idle for DIFS). This process is repeated until the backoff counter reaches to zero and the station is allowed to transmit. The idle period after a DIFS period is referred to as Contention Window. The IEEE 802.11 MAC layer protocol adopts exponential backoff. Contention Window is initially assigned the minimum contention window size CWmin. Then, the CW is doubled each time the station experiences a collision until the CW reaches to CWmax which is the maximum contention window size. When the CW is increased to CWmax, it remains the same even if there are more collisions. After every successful transmission, CW is reset to the initial value CWmin. A packet will be discarded if it cannot be successfully transmitted after it is retransmitted for a specific retry time.

9.3 Portability

Mobile phones and other wireless devices send information across free space using electromagnetic waves. To send these electromagnetic waves across long distances in free space, the frequency of the transmitted signal must be quite high compared to the frequency of the information signal. For example, the signal in a cell phone is a voice signal with a bandwidth of about 4 kHz. The typical frequency of the transmitted and received signal is several hundreds of megahertz to a few gigahertz (for example, the popular Wi-Fi standard is in the 2.4 GHz or 5+ GHz range).

One important reason why high-frequency transmission is attractive is that the size of the antenna required for efficient transmission is roughly one-quarter the wavelength of the propagating wave. Since the wavelength of the (electromagnetic) wave is inversely proportional to the frequency, the higher the frequency, the smaller the antenna. For example, the wavelength of a 1 GHz electromagnetic wave in free space is 30 cm, whereas a 1 kHz electromagnetic wave is one million times larger, 300 km, which would make for an impractically huge antenna and transmitter power to transmit signals of that frequency.

There are two reasons why frequency-division multiplexing works:

1. Any baseband signal can be broken up into a weighted sum of sinusoids using Fourier decomposition. If the baseband signal is band-limited, then there is a finite maximum frequency of the corresponding sinusoids. One can take this sum and modulate it on a carrier signal of

some other frequency in a simple way: by just multiplying the baseband and carrier signal (also called "mixing"). The result of modulating a band-limited baseband signal on to a carrier is a signal that is band-limited around the carrier, i.e., limited to some maximum frequency deviation from the carrier frequency.

2. When transmitted over a linear, time-invariant (LTI) channel, and if noise is negligible, each sinusoid shows up at the receiver as a sinusoid of the same frequency. The reason is that an LTI system preserves the sinusoids. If we were to send a baseband signal composed of a sum of sinusoids over the channel, the output will be the sum of sinusoids of the same frequencies. Each receiver can then apply a suitable filter to extract the baseband signal of interest to it. This insight is useful because the noise-free behavior of real-world communication channels is often well-characterized as an LTI system.

9.4 Amplitude Modulation with the Heterodyne Principle

The heterodyne principle is the basic idea governing several different modulation schemes. The idea is simple, though the notion that it can be used to modulate signals for transmission was hardly obvious before its discovery.

Heterodyne principle: The multiplication of two sinusoidal waveforms may be written as the sum of two sinusoidal waveforms, whose frequencies are given by the sum and the difference of the frequencies of the sinusoids being multiplied. This result may be seen from standard high-school trigonometric identities, or by (perhaps more readily) writing the sinusoids as complex exponentials and performing the multiplication. For example, using trigonometry,

$$\cos(\Omega_s n) \cdot \cos(\Omega_c n) = ½(\cos(\Omega_s + \Omega_c)n + \cos(\Omega_s - \Omega_c)n) \qquad \text{eq. 9.1}$$

We apply the heterodyne principle by treating the baseband signal, think of it as periodic with period 2π Ω for now, as the sum of different sinusoids of frequencies $\Omega_{s1} = k_1\Omega_1$,$\Omega_{s2} = k_2\Omega_1$,$\Omega_{s3} = k_3\Omega_1$ ... and treating the carrier as a sinusoid of frequency $\Omega_c = k_c\Omega_1$. Here, Ω_1 is the fundamental frequency of the baseband signal.

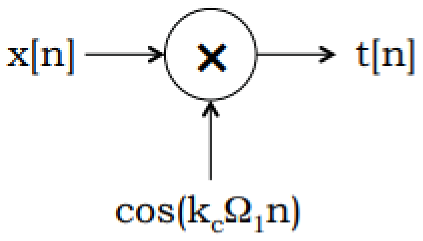


Figure 9-5: Modulation involved "mixing", or multiplying, the input signal x[n] with a carrier signal ($\cos(\Omega_c n) = \cos(k_c\Omega_1 n)$ here) to produce t[n], the transmitted signal.

The application of the heterodyne principle to modulation is shown schematically in Figure 9-5. Mathematically, we will find it convenient to use complex exponentials; with that notation, the process of modulation involves two important steps:

1. **Shape the input to band-limit it**. Take the input baseband signal and apply a lowpass filter to band-limit it. There are multiple good reasons for this input filter, but the main one is that we are interested in frequency division multiplexing and wish to make sure that there is no interference between concurrent transmissions. Hence, if we limit the discrete-time Fourier series (DTFS) coefficients to some range, call it $[-k_x, -k_x]$, then we can divide the frequency spectrum into non-overlapping ranges of size $2k_x$ to ensure that no two transmissions interfere. Without such a filter, the baseband could have arbitrarily high frequencies, making it hard to limit interference in general. Denote the result of shaping the original input by x[n]; in effect, that is the baseband signal we wish to transmit. An example of the original baseband signal and its shaped version is shown in Figure 9-6. We may express x[n] in terms of its discrete-time Fourier series (DTFS) representation as follows,

$$x[n] = \sum_{k=-k_x}^{k_x} A_k e^{jk\Omega_1 n}. \qquad \text{eq. 9.2}$$

Notice how applying the input filter ensures that high-frequency components are zero; the frequency range of the baseband is now $[-k_x\Omega_1, k_x\Omega_1]$ radians/sample.

2. Mixing step. Multiply x[n] (called the baseband modulating signal) by a carrier, $\cos(k_c\Omega_1 n)$, to produce the signal ready for transmission, t[n]. Using the DTFS form,

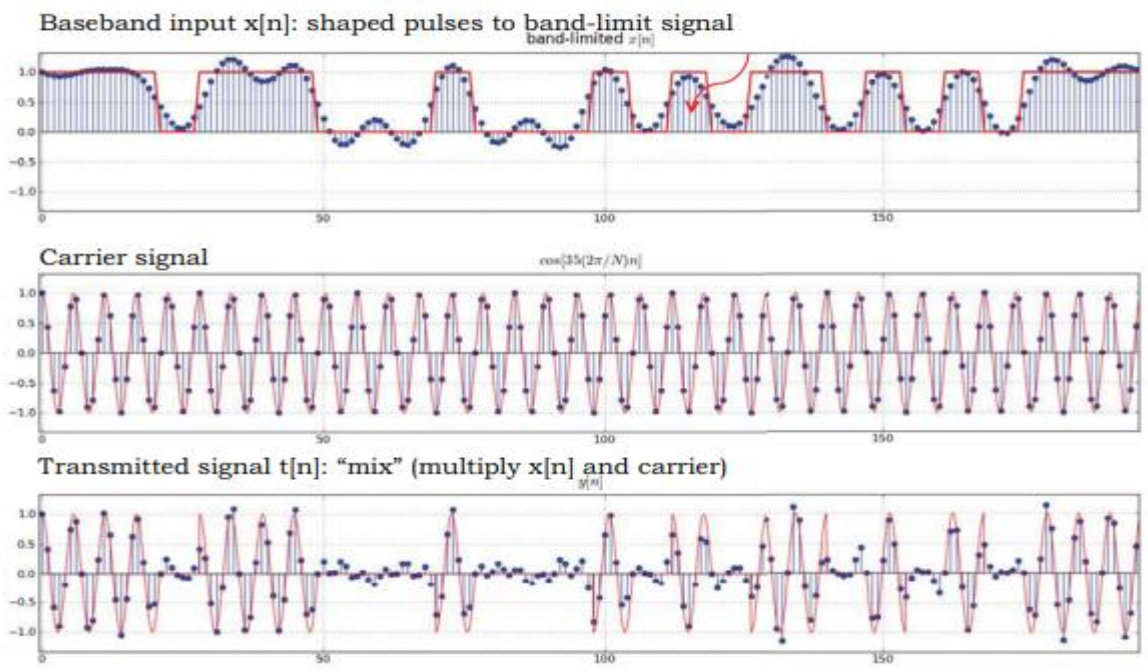


Figure 9-6: The two modulation steps, input filtering (shaping) and mixing, on an example signal

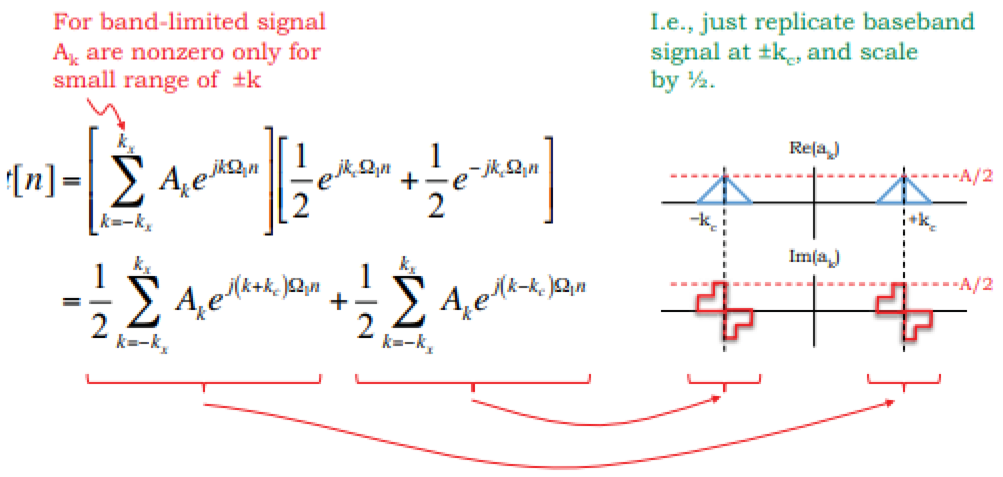
We get

$$\begin{aligned} t[n] &= \left(\sum_{k=-k_x}^{k_x} A_k e^{jk\Omega_1 n}\right)\left(\frac{1}{2}(e^{jk_c\Omega_1 n} + e^{-jk_c\Omega_1 n})\right) \\ &= \frac{1}{2}\sum_{k=-k_x}^{k_x} A_k e^{j(\underline{k+k_c})\Omega_1 n} + \frac{1}{2}\sum_{k=-k_x}^{k_x} A_k e^{j(\underline{k-k_c})\Omega_1 n}. \end{aligned}$$

eq. 9.3

Equation (9.3) makes it apparent (see the underlined terms) that the process of mixing produces, for each DTFS component, two frequencies of interest: one at the sum and the other at the difference of the mixed (multiplied) frequencies, each scaled to be one-half in amplitude compared to the original.

We transmit t[n] over the channel. The heterodyne mixing step may be explained mathematically using Equation (9.3), but you will rarely need to work out the math from scratch in any given problem: all you need to know and appreciate is that the (shaped) baseband signal is simply replicated in the frequency domain at two different frequencies, ±k_c, which are the nonzero DTFS coefficients of the carrier sinusoidal signal, and scaled by 1/2. We show this outcome schematically in Figure 9-7. The time-domain representation shown in Figure 9-6 is not as instructive as the frequency-domain picture to gain intuition about what modulation does and why frequency division multiplexing avoids interference.



$$[n] = \left[\sum_{k=-k_x}^{k_x} A_k e^{jk\Omega_1 n}\right]\left[\frac{1}{2}e^{jk_c\Omega_1 n} + \frac{1}{2}e^{-jk_c\Omega_1 n}\right]$$

$$= \underbrace{\frac{1}{2}\sum_{k=-k_x}^{k_x} A_k e^{j(k+k_c)\Omega_1 n}}_{} + \underbrace{\frac{1}{2}\sum_{k=-k_x}^{k_x} A_k e^{j(k-k_c)\Omega_1 n}}_{}$$

For band-limited signal
A_k are nonzero only for
small range of ±k
I.e., just replicate baseband
signal at ±k_c, and scale
by ½.
Re(a_k)
A/2
−k_c
+k_c
Im(a_k)
A/2

Figure 9-7: Illustrating the heterodyne principle.

9.5 Demodulation

The Simple No-Delay Case Assume for simplicity that the receiver captures the transmitted signal, t[n], with no distortion, noise, or delay; that's about as perfect as things can get. Let's see how to demodulate the received signal, r[n] = t[n], to extract x[n], the shaped baseband signal. The trick is to apply the heterodyne principle once again: multiply the received signal by a local sinusoidal signal that is identical to the carrier. An elegant way to see what would happen is to start with Figure 9-8, rather than the time-domain representation. We now can pretend that we have a "baseband" signal whose frequency components are as shown in Figure 9-8, and what we're doing now is to "mix" (i.e., multiply) that with the carrier. We can accordingly take each of the two (i.e., real and imaginary) pieces in the right-most column of Figure 9-8 and treat each in turn. The result is shown in Figure 9-9. The left column shows the frequency components of the original (shaped) baseband signal, x[n]. The middle column shows the frequency components of the modulated signal, t[n], which is the same as the right-most column of Figure 9-8. The carrier (cos($35\Omega_1$n), so the DTFS coefficients of t[n] are centered around k = −35 and k = 35 in the middle column. Now, when we mix that with a local signal identical to the carrier, we will shift each of these two groups of coefficients by ±35 once again, to see a cluster of coefficients at −70 and 0 (from the −35 group) and at 0 and +70 (from the +35 group). Each piece will be scaled by a further factor of 1/2, so the left and right clusters on the right-most column in Figure 14-7 will be 1/4 as large as the original baseband components, while the middle cluster centered at 0, with the same spectrum as the original baseband signal, will be scaled by 1/2. What we are interested in recovering is precisely this middle portion, centered at 0, because in the absence of any distortion, it is exactly the same as the original (shaped) baseband, except that is scaled by 1/2.

How would we recover this middle piece alone and ignore the left and right clusters, which are centered at frequencies that are at twice the carrier frequency in the positive and negative directions? We have already studied a technique: a low-pass filter. By applying a low-pass filter whose cut-off frequency lies between k_x and $2k_c - k_x$, we can recover the original signal faithfully.

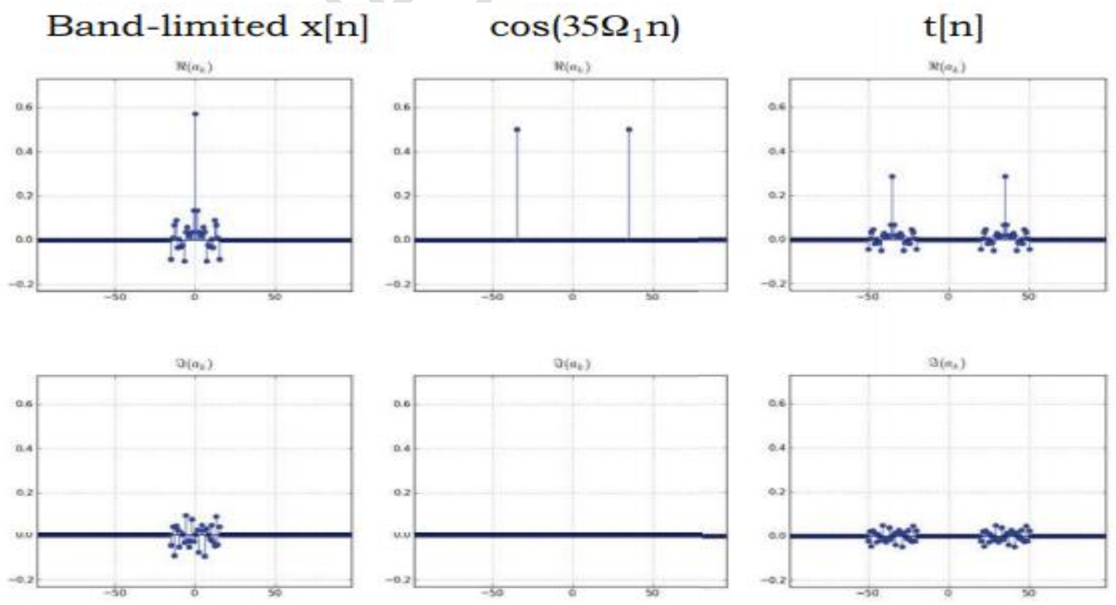


Figure 9-8: Frequency-domain representation, showing how the DTFS components (real and imaginary) of the real-valued band-limited signal x[n] after input filtering to produce shaped pulses (left), the purely cosine sinusoidal carrier signal (middle), and the heterodyned (mixed) baseband and carrier at two frequency ranges whose widths are the same as the baseband signal, but that have been shifted ±k_c in frequency, and scaled by 1/2 each (right). We can avoid interference with another signal whose baseband overlaps in frequency, by using a carrier for the other signal sufficiently far away in frequency from k_c.

We can reach the same conclusions by doing a more painstaking calculation, similar to the calculations we did for the modulation, leading to Equation (9.3). Let z[n] be the signal obtained by multiplying (mixing) the local replica of the carrier $\cos(k_c\Omega_1 n)$ and the received signal, r[n] = t[n], which is of course equal to x[n] $\cos(k_c\Omega_1 n)$. Using Equation 9.3, we can express z[n] in terms of its DTFS coefficients as follows:

$$
\begin{aligned}
z[n] &= t[n]\left(\frac{1}{2}e^{jk_c\Omega_1 n} + \frac{1}{2}e^{-jk_c\Omega_1 n}\right) \\
&= \left(\frac{1}{2}\sum_{k=-k_x}^{k_x} A_k e^{j(k+k_c)\Omega_1 n} + \frac{1}{2}\sum_{k=-k_x}^{k_x} A_k e^{j(k-k_c)\Omega_1 n}\right)\left(\frac{1}{2}e^{jk_c\Omega_1 n} + \frac{1}{2}e^{-jk_c\Omega_1 n}\right) \\
&= \frac{1}{4}\sum_{k=-k_x}^{k_x} A_k e^{j(k+2k_c)\Omega_1 n} + \underline{\frac{1}{2}\sum_{k=-k_x}^{k_x} A_k e^{jk\Omega_1 n}} + \frac{1}{4}\sum_{k=-k_x}^{k_x} A_k e^{j(k-2k_c)\Omega_1 n} \qquad (
\end{aligned}
$$

eq. 9.4

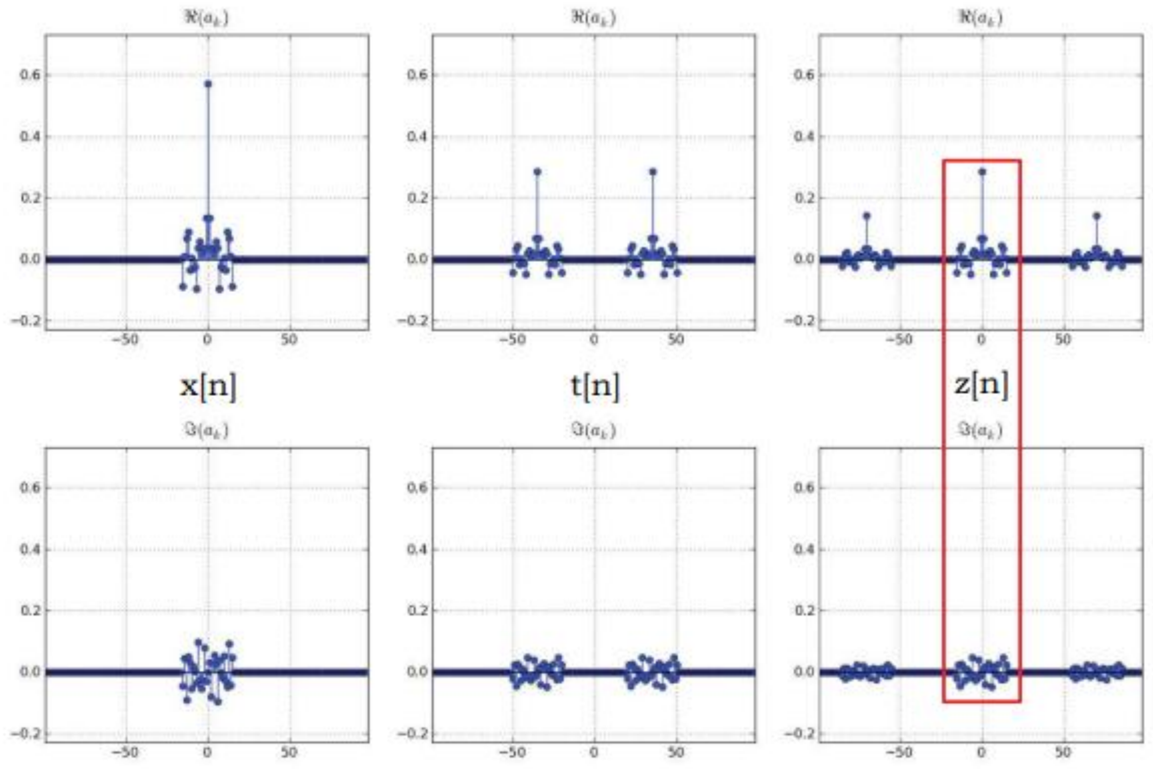


Figure 9-9: Applying the heterodyne principle in demodulation: frequency-domain explanation. The left column is the (shaped) baseband signal spectrum, and the middle column is the spectrum of the modulated signal that is transmitted and received. The portion shown in the vertical rectangle in the right-most column has the DTFS coefficients of the (shaped) baseband signal, x[n], scaled by a factor of 1/2, and may be recovered faithfully using a low-pass filter. This picture shows the simplified ideal case when there is no channel distortion or delay between the sender and receiver.

9.6 Handling Channel Distortions

Thus far, we have considered the ideal case of no channel distortions or delays. We relax this idealization and consider channel distortions now. If the channel is LTI (which is very often the case), then one can extend the approach described above.

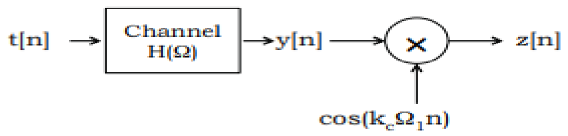


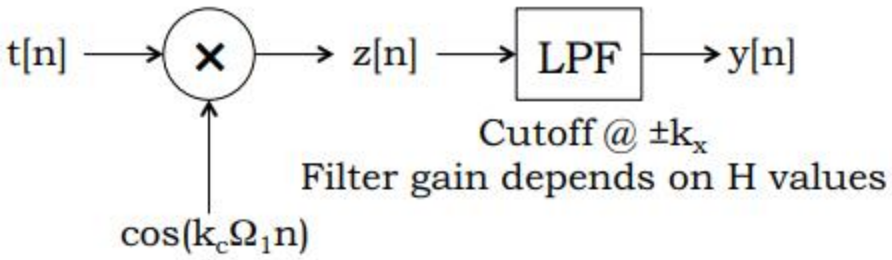
Figure 9-10: Demodulation in the presence of channel distortion characterized by the frequency response of the channel.

The difference is that each of the Ak terms in Equation (9.4), as well as Figure 9-9 will be multiplied by the frequency response of the channel, H(Ω), evaluated at a frequency of $k\Omega_1$. So, each DTFS coefficient will be scaled further by the value of this frequency response at the relevant frequency. Figure 9-10 shows the model of the system now. The modulated input, t[n], traverses the channel enroot to the demodulator at the receiver. The result, z[n], may be written as follows:

$$
\begin{aligned}
z[n] &= y[n]\cos(k_c\Omega_1 n) \\
&= y[n]\left(\frac{1}{2}e^{jk_c\Omega_1 n} + \frac{1}{2}e^{-jk_c\Omega_1 n}\right) \\
&= \Big(\frac{1}{2}\sum_{k=-k_x}^{k_x} H((k+k_c)\Omega_1)A_k e^{j(k+k_c)\Omega_1 n} + \frac{1}{2}\sum_{k=-k_x}^{k_x} H((k-k_c)\Omega_1)A_k e^{j(k-k_c)\Omega_1 n} \\
&\quad \Big(\frac{1}{2}e^{jk_c\Omega_1 n} + \frac{1}{2}e^{-jk_c\Omega_1 n}\Big) \\
&= \underline{\frac{1}{4}\sum_{k=-k_x}^{k_x} A_k e^{jk\Omega_1 n}\Big(H((k+k_c)\Omega_1) + H((k-k_c)\Omega_1)\Big)} + \\
&\quad \frac{1}{4}\sum_{k=-k_x}^{k_x} A_k e^{j(k+2k_c)\Omega_1 n}\Big(H((k+k_c)\Omega_1) + H((k-k_c)\Omega_1)\Big) + \\
&\quad \frac{1}{4}\sum_{k=-k_x}^{k_x} A_k e^{j(k-2k_c)\Omega_1 n}\Big(H((k+k_c)\Omega_1) + H((k-k_c)\Omega_1)\Big) \qquad (1
\end{aligned}
$$

eq. 9.5

Of these three terms in the RHS of Equation (9.5), the first term contains the baseband signal that we want to extract. We can do that as before by applying a lowpass filter to get rid of the $\pm 2k_c$ components. To then recover each A_k, we need to pass the output of the lowpass filter to another LTI filter that undoes the distortion by multiplying the kth Fourier coefficient by the inverse of $H((k + k_c)\Omega_1) + H((k - k_c)\Omega_1)$. Doing so, however, will also amplify any noise at frequencies where the channel attenuated the input signal t[n], so a better solution is obtained by omitting the inversion at such frequencies. For this procedure to work, the channel must be relatively low-noise, and the receiver needs to know the frequency response, $H(\Omega)$, at all the frequencies of interest in Equation (9.5); i.e., in the range $[-k_c - k_x, -k_c + k_x]$ and $[k_c - k_x, k_c + k_x]$. To estimate $H(\Omega)$, a common approach is to send a known preamble at the beginning of each packet (or frame)



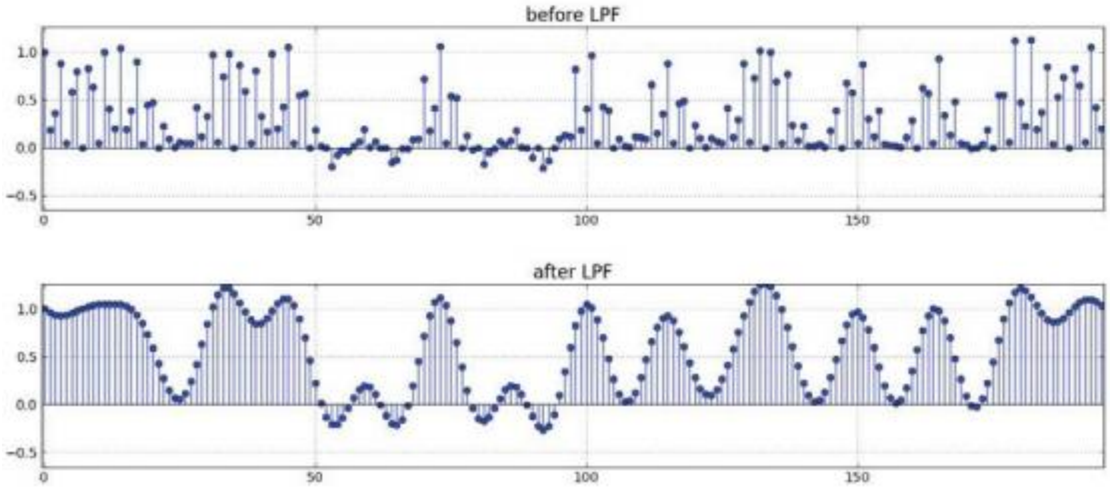


Figure 9-11: Demodulation steps: the no-delay case (top). LPF is a lowpass filter. The graphs show the time-domain representations before and after the LPF.

of transmission. The receiver looks for this known preamble to synchronize the start of reception, and because the transmitted signal pattern is known, the receiver can deduce channel's the unit sample response, h[·], from it. One can then apply the frequency response equation, to estimate H(Ω) and use it to approximately undo the distortion introduced by the channel. Ultimately, however, our interest is not in accurately recovering x[n], but rather the underlying bit stream. For this task, what is required is typically not an inverse filtering operation. We instead require a filtering that produces a signal whose samples, obtained at the bit rate, allow reliable decisions regarding the corresponding bits, despite the presence of noise. The optimal filter for this task is called the **matched filter.** We leave the discussion of the matched filter to more advanced courses in communication.

9.7 More Sophisticated (De)Modulation Schemes

9.8 We conclude this chapter by briefly outlining three more sophisticated (de)modulation schemes.

9.7.1 Binary Phase Shift Keying (BPSK)

In BPSK, as shown in Figure 14-14, the transmitter selects one of two phases for the carrier, e.g. −π/2 for “0” and π/2 for “1”. The transmitter does the same mixing with a sinusoid as explained earlier. The receiver computes the I and Q components from its received waveform, as before. This approach “almost” works, but in the presence of channel delays or phase errors, the previous strategy to recover the input does not work because we had assumed that x[n] ≥ 0. With BPSK, x[n] is either +1 or −1, and the two levels we wish to distinguish have the same magnitude on the complex plane after quadrature demodulation. The solution is to think of the phase encoding as a differential, not absolute: a change in phase corresponds to a change in bit value. Assume that every message starts with a “0” bit. Then, the first phase change represents a 0 → 1 transition, the second phase change a 1 → 0 transition, and so on. One can then recover all the bits correctly in the demodulator using this idea, assuming no intermediate glitches (we will not worry about such glitches here, which do occur in practice and must be dealt with).

9.7.2 Quadrature Phase Shift Keying (QPSK)

Quadrature Phase Shift Keying is a clever idea to add a “degree of freedom” to the system (and thereby extracting higher performance). This method, uses a quadrature scheme at both the transmitter and the receiver. When mapping bits to voltage values in QPSK, we would choose the values so that the amplitude of t[n] is constant. Moreover, because the constellation now involves four symbols, we map two bits to each symbol. So 00 might map to (A,A), 01 to (−A,A), 11 to (−A,−A), and 10 to (A,−A) (the amplitude is therefore $\sqrt{2A}$). There is some flexibility in this mapping, but it is not completely arbitrary; for example, we were careful here to not map 11 to (A,−A) and 00 to (A,A). The reason is that any noise is more likely to cause (A,A) to be confused fo (A,−A), compared to (−A,−A), so we would like a symbol error to corrupt as few bits as possible.

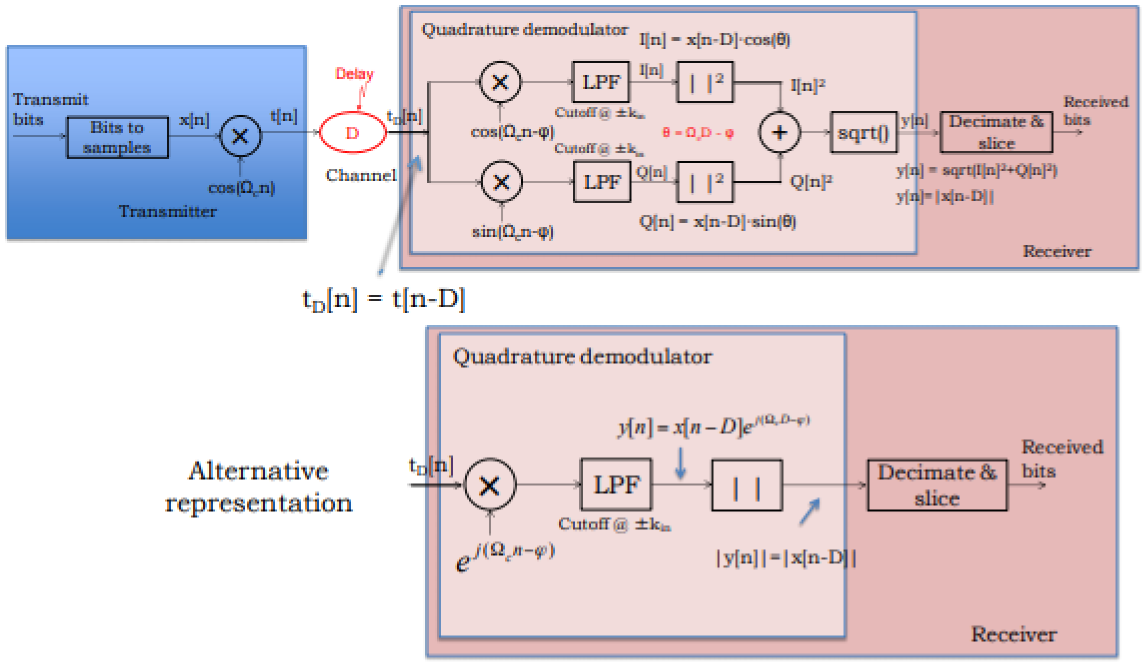
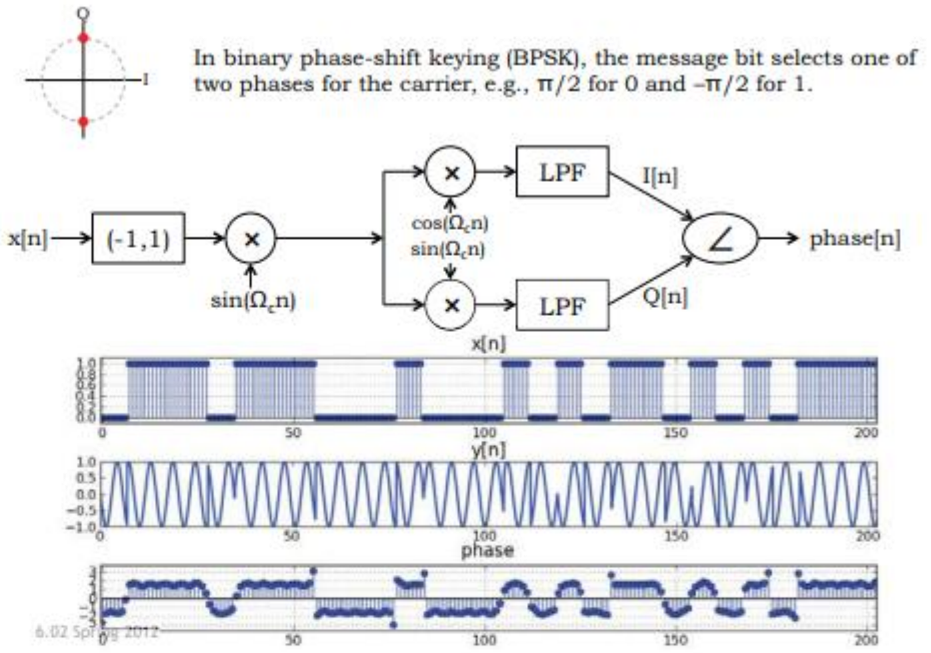


Figure 9-12: Quadrature demodulation: overall system view. The "alternative representation" shown implements the quadrature demodulator using a single complex exponential multiplication, which is a more compact representation and description.

9.7.3 Quadrature Amplitude Modulation (QAM)

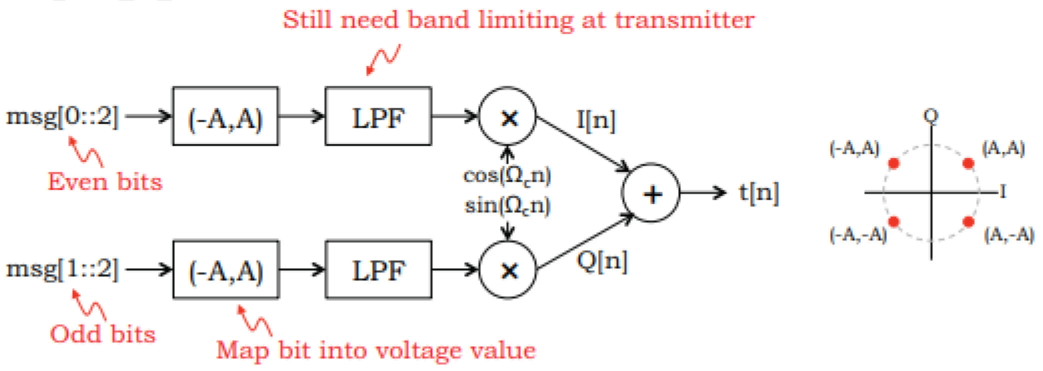
QAM may be viewed as a generalization of QPSK (in fact, QPSK is sometimes called QAM4). One picks additional points in the constellation, varying both the amplitude and the phase. In QAM-16, we map four bits per symbol. Denser QAM constellations are also possible; practical systems today use QAM-4 (QPSK), QAM-16, and QAM64. Quadrature demodulation with the adjustment for phase is the demodulation scheme used at the receiver with QAM. For a given transmitter power, the signal levels corresponding to different bits at the input get squeezed closer together in amplitude as one goes to constellations with more points. The resilience to noise reduces because of this reduced separation, but sophisticated coding and signal processing techniques may be brought to bear to deal with the effects of noise to achieve higher communication bit rates. In many real-world communication systems, the physical layer provides multiple possible constellations and choice of codes; for any given set of channel conditions (e.g., the noise variance, if the channel is well-described using the AWGN model), there is some combination of constellation, coding scheme, and code rate, which maximizes the rate at which bits can be received and decoded reliably. Higher-layer “bit rate selection” protocols use information about the channel quality (signal-to-noise ratio, packet loss rate, or bit error rate) to make this decision.

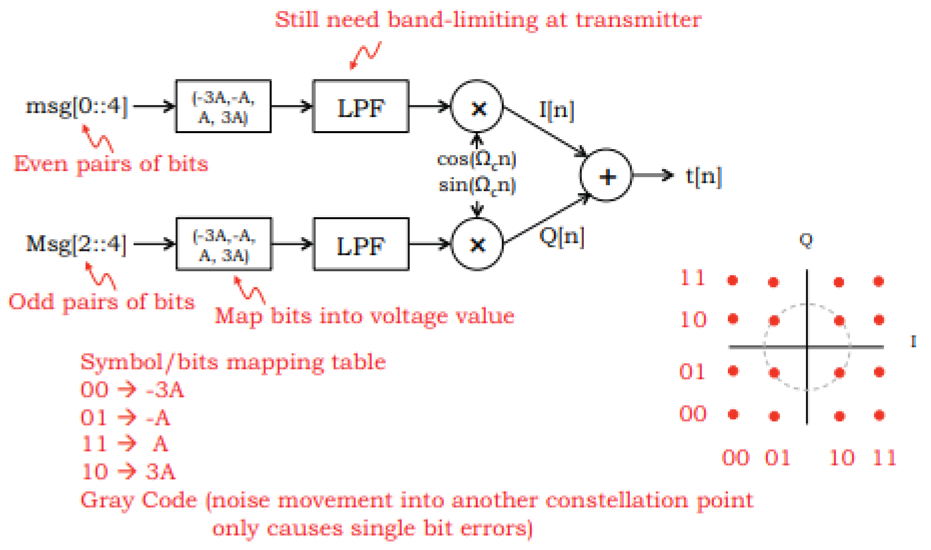
Summary of Study Session 9



In binary phase-shift keying (BPSK), the message bit selects one of two phases for the carrier, e.g., π/2 for 0 and −π/2 for 1.
x[n]
(-1,1)
sin(Ωcn)
cos(Ωcn)
sin(Ωcn)
LPF
LPF
I[n]
Q[n]
phase[n]
x[n]
y[n]
phase
6.02 Spring 2012

Figure 9-13: Binary Phase Shift Keying (BPSK).




Still need band-limiting at transmitter
msg[0::4]
(-3A,-A, A, 3A)
LPF
I[n]
Even pairs of bits
cos(Ω_c n)
sin(Ω_c n)
t[n]
Msg[2::4]
(-3A,-A, A, 3A)
LPF
Q[n]
Odd pairs of bits
Map bits into voltage value
Q
I
11
10
01
00
00 01 10 11
Symbol/bits mapping table
00 → -3A
01 → -A
11 → A
10 → 3A
Gray Code (noise movement into another constellation point only causes single bit errors)

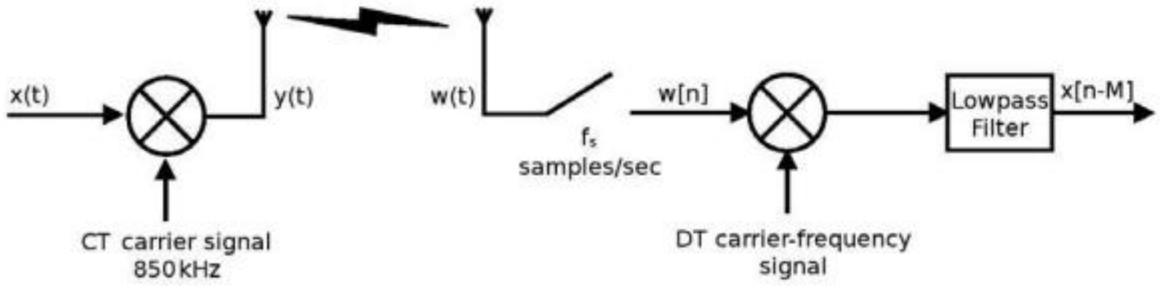
Self-Assessment Question (SAQs) for Study 9

SAQ 9.1

The University sports radio station WEEI AM ("amplitude modulation") broadcasts on a carrier frequency of 850 kHz, so its continuous-time (CT) carrier signal can be taken to be cos(2π × 850 × 103t), where t is measured in seconds. Denote the CT audio signal that's modulated onto this carrier by x(t), so that the CT signal transmitted by the radio station is

y(t) = x(t) cos(2π × 850 × 103t) ,

as indicated schematically on the left side of the figure below



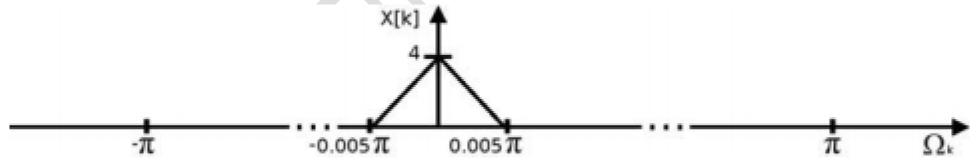
We use the symbols y[n] and x[n] to denote the discrete-time (DT) signals that would have been obtained by respectively sampling y(t) and x(t) in Equation (9.9) at fs samples/sec; more specifically, the signals are sampled at the discrete time instants t = n(1/fs). Thus

$$y[n] = x[n]\cos(\Omega_c n) \qquad \text{eq. 9.10}$$

for an appropriately chosen value of the angular frequency Ω_c. Assume that x[n] is periodic with some period N, and that fs = 2 × 106 samples/sec. Answer the following questions, explaining your answers in the space provided.

(a) Determine the value of Ω_c in Equation (9.10), restricting your answer to a value in the range [−π,π]. (You can assume in what follows that the period N of x[n] is such that $\Omega_c = 2k_c\pi/N$ for some integer k_c; this is a detail, and needn't concern you unduly.)

(b) Suppose the Fourier series coefficients X[k] of the DT signal x[n] in Equation (9.10) are purely real, and are as shown in the figure below, plotted as a function of $\Omega k = 2k\pi/N$. (Note that the figure is not drawn to scale. Also, the different values of Ω_k are so close to each other that we have just interpolated adjacent values of X[k] with a straight line, rather than showing you a discrete "stem" plot.) Observe that the Fourier series coefficients are non-zero for frequencies Ω_k in the interval [−.005π,.005π], and 0 at all other Ω_k in the interval [−π,π].



Draw a carefully labeled sketch below (though not necessarily to scale) to show the Fourier series coefficients of the DT modulated signal y[n]. However, rather than labeling your horizontal axis with the Ω_k, as we have done above, you should label the axis with the appropriate frequency f_k in Hz.

Assume now that the receiver detects the CT signal w(t) = 10−3y(t − t_0), where $t_0 = 3 \times 10^{-6}$ sec, and that it samples this signal at fs samples/sec, thereby obtaining the DT signal

$w[n] = 10^{-3}y[n-M] = 10^{-3}\, x[n -M\,] \cos(\Omega_c(n- M)$ eq. 9.11

for an appropriately chosen integer M.

C. Determine the value of M in Equation (9.11).

D. Noting your answer from part B, determine for precisely which intervals of the frequency axis the Fourier series coefficients of the signal y[n − M] in Equation (9.11) are non-zero. You need not find the actual coefficients, only the frequency range over which these coefficients will be non-zero. Also state whether or not the Fourier coefficients will be real. Explain your answer.

E. The demodulation step to obtain the DT signal x[n−M] from the received signal w[n] now involves multiplying w[n] by a DT carrier-frequency signal, followed by appropriate low-pass filtering (with the gain of the low-pass filter in its passband being chosen to scale the signal to whatever amplitude is desired). Which one of the following six DT carrier-frequency signals would you choose to multiply the received signal by? Circle your choice and give a brief explanation.

i. $\cos\Big(\Omega_c n\Big)$.
ii. $\cos\Big(\Omega_c(n-M)\Big)$
iii. $\cos\Big(\Omega_c(n+M)\Big)$
iv. $\sin\Big(\Omega_c n\Big)$.
v. $\sin\Big(\Omega_c(n-M)\Big)$.
vi. $\sin\Big(\Omega_c(n+M)\Big)$.

Study Session 10: Sharing a Channel: Media Access (MAC) Protocols

10.1 Introduction

There are many communication channels, including radio and acoustic channels, and certain kinds of wired links (coaxial cables), where multiple nodes can all be connected and hear each other's transmissions (either perfectly or with some non-zero probability). This chapter addresses the fundamental question of how such a common communication channel also called a shared medium can be shared between the different nodes.

There are two fundamental ways of sharing such channels (or media): time sharing and frequency sharing. The idea in time sharing is to have the nodes coordinate with each other to divide up the access to the medium one at a time, in some fashion. The idea in frequency sharing is to divide up the frequency range available between the different transmitting nodes in a way that there is little or no interference between concurrently transmitting nodes. The methods used here are the same as in frequency division multiplexing. This chapter focuses on time sharing. We will investigate two common ways: time division multiple access, or TDMA, and contention protocols. Both approaches are used in networks today. These schemes for time and frequency sharing are usually implemented as communication protocols. The term **protocol** refers to the rules that govern what each node is allowed to do and how it should operate. Protocols capture the "rules of engagement" that nodes must follow, so that they can collectively obtain good performance. Because these sharing schemes define how multiple nodes should control their access to a shared medium, they are termed **media access (MAC) protocols or multiple access protocols**. Of particular interest to us are contention protocols, so called because the nodes contend with each other for the medium without pre-arranging a schedule that determines who should transmit when, or a frequency reservation that guarantees little or no interference. These protocols operate in laissez faire fashion: nodes get to send according to their own volition without any external agent telling them what to do. These contention protocols are well-suited for data networks, which are characterized by nodes transmitting data in bursts and at variable rates (we will describe the properties of data networks in more detail in a later chapter on packet switching). In this chapter and the subsequent ones, we will assume that any message is broken up into a set of one or more packets, and a node attempts to send each packet separately over the shared medium.

10.1.1 Examples of Shared Media Satellite communications.

Satellite communications

Perhaps the first example of a shared-medium network deployed for data communication was a satellite network: the ALOHAnet in Hawaii. The ALOHAnet was designed by a team led by Norm Abramson in the 1960s at the University of Hawaii as a way to connect computers in the different islands together (Figure 10-1). A computer on the satellite functioned as a switch to provide connectivity between the nodes on the islands; any packet between the islands had to be first sent over the uplink to the switch, and from there over the downlink to the desired destination. Both directions used radio communication and the medium was shared. Eventually, this satellite network was connected to the ARPANET (the precursor to today’s Internet). Such satellite networks continue to be used today in various parts of the world, and they are perhaps the most common (though expensive) way to obtain connectivity in the high seas and other remote regions of the world. Figure 10.1 shows the schematic of such a network connecting islands in the Pacific Ocean and used for teleconferencing. In these satellite networks, the downlink usually runs over a different frequency band from the uplinks, which all share the same frequency band. The different uplinks, however, need to be shared by different concurrent communications from the ground stations to the satellite.

Marshall Is.
Nauru
Kiribati
Solomon
Tuvalu
Tokelau
Vanuatu
Fiji
Samoa
Niue
Cook Is.
Tonga

Figure 10-1: A satellite network. The "uplinks" from the ground stations to the satellite form a shared medium.

Wireless networks.

The most common example of a shared communication medium today, and one that is only increasing in popularity, uses radio. Examples include cellular wireless networks (including standards like EDGE, 3G, and 4G), wireless LANs (such as 802.11, the WiFi standard), and various other forms of radio-based communication. Another example of a communication medium with similar properties is the acoustic channel explored in the 6.02 labs. Broadcast is an inherent property of radio and acoustic communication, especially with so-called omni-directional antennas, which radiate energy in all (or many) different directions. However, radio and acoustic broadcasts are not perfect because of interference and the presence of obstacles on certain paths, so different nodes may correctly receive different parts of any given transmission. This reception is probabilistic and the underlying random processes that generate bit errors are hard to model.

Shared bus networks.

An example of a wired shared medium is Ethernet, which when it was first developed (and for many years after) used a shared cable to which multiple nodes could be connected. Any packet sent over the Ethernet could be heard by all stations connected physically to the network, forming a perfect shared broadcast medium. If two or more nodes send packets that overlap in time, both packets ended up being garbled and received in error.

Over-the-air radio and television.

Even before data communication, many countries in the world had (and still have) radio and television broadcast stations. Here, a relatively small number of transmitters share a frequency range to deliver radio or television content. Because each station was assumed to be active most of the time, the natural approach to sharing is to divide up the frequency range into smaller sub-ranges and allocate each subrange to a station (frequency division multiplexing). Given the practical significance of these examples, and the sea change in network access brought about by wireless technologies, developing methods to share a common medium is an important problem.

10.2 Model and Goals

Before diving into the protocols, let's first develop a simple abstraction for the shared medium and more rigorously model the problem we're trying to solve. This abstraction is a reasonable first-order approximation of reality. We are given a set of N nodes sharing a communication medium. We will assume N is fixed, but the protocols we develop will either continue to work when N varies, or can be made to work with some more effort. Depending on the context, the N nodes may or may not be able to hear each other; in some cases, they may not be able to at all, in some cases, they may, with some probability, and in some cases, they will always hear each other. Each node has some source of data that produces packets. Each packet may be destined for some other node in the network. For now, we will assume that every node has packets destined to one given "master" node in the network. Of course, the master must be capable of hearing every other node, and receiving packets from those nodes. We will assume that the master perfectly receives packets from each node as long as there are no "collisions" (we explain what a "collision" is below).

The model we consider has the following rules:

1. Time is divided into slots of equal length, τ .

2. Each node can send a packet only at the beginning of a slot.

3. All packets are of the same size, and equal to an integral multiple of the slot length. In practice, packets will of course be of varying lengths, but this assumption simplifies our analysis and does not affect the correctness of any of the protocols we study.

4. Packets arrive for transmission according to some random process; the protocol should work correctly regardless of the process governing packet arrivals. If two or more nodes send a packet in the same time slot, they are said to collide, and none of the packets are received successfully. Note that even if only part of a packet encounters a collision, the entire packet is

assumed to be lost. This “perfect collision” assumption is an accurate model for wired shared media like Ethernet, but is only a crude approximation of wireless (radio) communication.

The reason is that it might be possible for multiple nodes to concurrently transmit data over radio, and depending on the positions of the receivers and the techniques used to decode packets, for the concurrent transmissions to be received successfully.

5. The sending node can discover that a packet transmission collided and may choose to retransmit such a packet.

6. Each node has a queue; any packets waiting to be sent are in the queue. A node with a non-empty queue is said to be backlogged.

Performance goals.

An important goal is to provide high throughput, i.e., to deliver packets successfully at as high a rate as possible, as measured in bits per second. A mea-sure of throughput that is independent of the rate of the channel is the utilization, which is defined as follows:

Definition. The utilization that a protocol achieves is defined as the ratio of the total throughput to the maximum data rate of the channel. For example, if there are 4 nodes sharing a channel whose maximum bit rate is 10 Megabits/s,3 and they get throughputs of 1, 2, 2, and 3 Megabits/s, then the utilization is (1 + 2 + 2 + 3)/10 = 0.8. Obviously, the utilization is always between 0 and 1. Note that the utilization may be smaller than 1 either because the nodes have enough offered load and the protocol is inefficient, or because there isn’t enough offered load. By offered load, we mean the load presented to the network by a node, or the aggregate load presented to the network by all the nodes. It is measured in bits per second as well.

But utilization alone isn’t sufficient: we need to worry about fairness as well. If we weren’t concerned about fairness, the problem would be quite easy because we could arrange for a particular backlogged node to always send data. If all nodes have enough load to offer to the network, this approach would get high utilization. But it isn’t too useful in practice because it would also starve one or more other nodes. A number of notions of fairness have been developed in the literature, and it’s a topic that continues to generate activity and interest. For our purposes, we will use a simple, standard definition of fairness: we will measure the throughput achieved by each node over some time period, T, and say that an allocation with lower standard deviation is “fairer” than one with higher standard deviation. Of course, we want the notion to work properly when the number of nodes varies, so some normalization is needed. We will use the following simplified fairness index:

$$F = \frac{(\sum_{i=1}^{N} x_i)^2}{N \sum x_i^2},$$

eq. 10.1

where xi is the throughput achieved by node i and there are N backlogged nodes in all. Clearly, 1/N ≤ F ≤ 1; F = 1/N implies that a single node gets all the throughput, while F = 1 implies perfect fairness. We will consider fairness over both the long-term (many thousands of “time slots”) and over the short term (tens of slots). It will turn out that in the schemes we study, some schemes will achieve high utilization but poor fairness, and that as we improve fairness, the overall utilization will drop. The next section discusses Time Division Multiple Access, or TDMA, a scheme that achieves high fairness, but whose utilization may be low when the offered load is nonuniform between the nodes, and is not easy to implement in a fully distributed way without a central coordinator when nodes join and leave dynamically. However, there are practical situations when TDMA works well, and such protocols are used in some cellular wireless networks. Then, we will discuss a variant of the Aloha protocol, the first contention MAC protocol that was invented. Aloha forms the basis for many widely used contention protocols, including the ones used in the IEEE 802.11 (Wi-Fi) standard.

10.3 Time Division Multiple Access (TDMA)

If one had a centralized resource allocator, such as a base station in a cellular network, and a way to ensure some sort of time synchronization between nodes, then a “time division” is not hard to develop. As the name suggests, the goal is to divide time evenly between the N nodes. One way to achieve this goal is to divide time into slots starting from 0 and incrementing by 1, and for each node to be given a unique identifier (ID) in the range [0, N − 1].

A simple TDMA protocol uses the following rule:

If the current time slot is t, then the node with ID i transmits if, and only if, it is backlogged and tmodN = i. If the node whose turn it is to transmit in time slot t is not backlogged, then that time slot is “wasted”. This TDMA scheme has some good properties. First, it is fair: each node gets the same number of transmission attempts because the protocol provides access to the medium in round-robin fashion among the nodes. The protocol also incurs no packet collisions (assuming it is correctly implemented!): exactly one node is allowed to transmit in any time slot. And if the number of nodes is static, and there is a central coordinator (e.g., a master nodes), this TDMA protocol is simple to implement. This TDMA protocol does have some drawbacks. First and foremost, if the nodes send data in bursts, alternating between periods when they are backlogged and when they are not, or if the amount of data sent by each node is different, then TDMA under-utilizes the medium. The degree of under-utilization depends on how skewed the traffic pattern; the more the imbalance, the lower the utilization. An “ideal” TDMA scheme would provide equal access to the medium only among currently backlogged nodes, but even in a system with a central master, knowing which nodes are currently backlogged is somewhat challenging. Second, if each node sends packets that are of different sizes (as is the case in practice, though the model we specified above did not have this wrinkle), making TDMA work correctly is more involved. It can still be made to work, but it takes more effort. An important special case is when each node sends packets of the same size, but the size is bigger than a single time slot. This case is not hard to handle, though it requires a little more thinking, and is left as an exercise for the reader.) Third, making TDMA work in a fully distributed way in a system without a central master, and in cases when the number of nodes changes dynamically, is tricky. It can be done, but the protocol quickly becomes more complex than the simple rule stated above.

Contention protocols like Aloha and CSMA don’t suffer from these problems, but unlike TDMA, they encounter packet collisions. In general, burst data and skewed workloads favor contention protocols over TDMA. The intuition in these protocols is that we somehow would like to allocate access to the medium fairly, but only among the backlogged nodes. Unfortunately, only each node knows with certainty if it is backlogged or not. Our solution is to use randomization, a simple but extremely powerful idea; if each backlogged node transmits data with some probability, perhaps we can arrange for the nodes to pick their transmission probabilities to engineer an outcome that has reasonable utilization (throughput) and fairness! The rest of this chapter describes such randomized contention protocols, starting with the ancestor of them all, Aloha.

10.3.1 Definition -Time Division Multiple Access (TDMA)

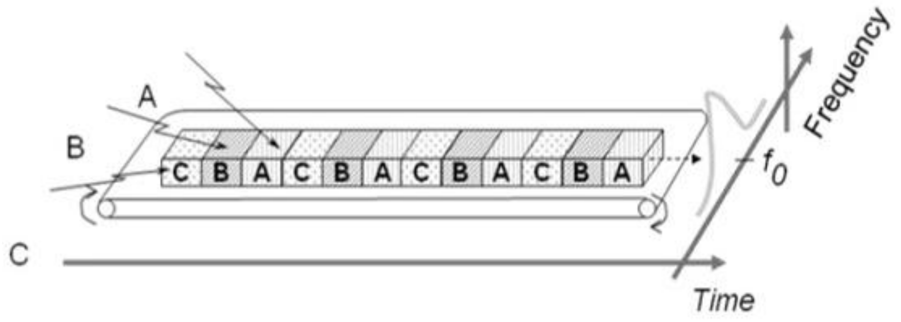
Time division multiple access (TDMA) is a channel access method (CAM) used to facilitate channel sharing without interference. TDMA allows multiple stations to share and use the same transmission channel by dividing signals into different time slots. Users transmit in rapid succession, and each one uses its own time slot. Thus, multiple stations (like mobiles) may share the same frequency channel but only use part of its capacity.

Time Division Multiple Access (TDMA) is a digital cellular telephone communication technology. It facilitates many users to share the same frequency without interference. Its technology divides a signal into different timeslots and increases the data carrying capacity.Time Division Multiple Access (TDMA) is a complex technology because it requires an accurate synchronization between the transmitter and the receiver. TDMA is used in digital mobile radio systems. The individual mobile stations cyclically assign a frequency for the exclusive use of a time interval.

In most of the cases, the entire system bandwidth for an interval of time is not assigned to a station. However, the frequency of the system is divided into sub-bands, and TDMA is used for the multiple access in each sub-band. Sub-bands are known as carrier frequencies. The mobile system that uses this technique is referred to as the multi-carrier systems. Examples of TDMA include IS-136, personal digital cellular (PDC), integrated digital enhanced network (iDEN) and the second generation (2G) Global System for Mobile Communications (GSM).

TDMA allows a mobile station's radio component to listen and broadcast only in its assigned time slot. During the remaining time period, the mobile station may apply network measurements by detecting surrounding transmitters in different frequencies. This feature allows interfrequency handover, which differs from a code division multiple access (CDMA), where frequency handover is difficult to achieve. However, CDMA allows handoffs, which enable mobile stations to simultaneously communicate with up to six base stations. TDMA is used in most 2G cellular systems, while 3G systems are based on CDMA. However, TDMA remains relevant to modern systems. For example, combined TDMA, CDMA and time division duplex (TDD) are universal terrestrial radio access (UTRA) systems that allow multiple users to share one time slot.

In the following example, the frequency band has been shared by three users. Each user is assigned definite timeslots to send and receive data. In this example, user 'B' sends after user 'A,' and user 'C' sends thereafter. In this way, the peak power becomes a problem and larger by the burst communication.



FDMA and TDMA

This is a multi-carrier TDMA system. A 25 MHz frequency range holds 124 single chains (carrier frequencies 200) bandwidth of each kHz; each of these frequency channels contains 8 TDMA conversation channels. Thus, the sequence of timeslots and frequencies assigned to a mobile station is the physical channels of a TDMA system. In each timeslot, the mobile station transmits a data packet. The period of time assigned to a timeslot for a mobile station also determines the number of TDMA channels on a carrier frequency. The period of timeslots is combined in a so-called TDMA frame. TDMA signal transmitted on a carrier frequency usually requires more bandwidth than FDMA signal. Due to the use of multiple times, the gross data rate should be even higher.

10.3.2 Advantages of TDMA

Here is a list of few notable advantages of TDMA −

- Permits flexible rates (i.e. several slots can be assigned to a user, for example, each time interval translates 32Kbps, a user is assigned two 64 Kbps slots per frame).
- Can withstand gusty or variable bit rate traffic. A number of slots allocated to a user can be changed frame by frame (for example, two slots in the frame 1, three slots in the frame 2, one slot in the frame 3, frame 0 of the notches 4, etc.).
- No guard band required for the wideband system.
- No narrowband filter required for the wideband system.

10.3.3 Disadvantages of TDMA

The disadvantages of TDMA are as follow

- High data rates of broadband systems require complex equalization.
- Due to the burst mode, a large number of additional bits are required for synchronization and supervision.
- Call time is needed in each slot to accommodate time to inaccuracies (due to clock instability).
- Electronics operating at high bit rates increase energy consumption.
- Complex signal processing is required to synchronize within short slots.

10.4 Aloha

ALOHAnet, also known as the ALOHA System, or simply ALOHA, was a pioneering computer networking system developed at the University of Hawaii. ALOHAnet became operational in June 1971, providing the first public demonstration of a wireless packet data network. ALOHA originally stood for Additive Links On-line Hawaii Area. The ALOHAnet used a new method of medium access (ALOHA random access) and experimental ultra-high frequency (UHF) for its operation since frequency assignments for communications to and from a computer were not available for commercial applications in the 1970s. But even before such frequencies were assigned there were two other media available for the application of an ALOHA channel –

cables & satellites. In the 1970s ALOHA, random access was employed in the nascent Ethernet cable-based network and then in the Marisat (now Inmarsat) satellite network.

In the early 1980s frequencies for mobile networks became available, and in 1985 frequencies suitable for what became known as Wi-Fi was allocated in the US. These regulatory developments made it possible to use the ALOHA random-access techniques in both Wi-Fi and in mobile telephone networks. ALOHA channels were used in a limited way in the 1980s in 1G mobile phones for signaling and control purposes. In the late 1980s, the European standardization group GSM who worked on the Pan-European Digital mobile communication system GSM greatly expanded the use of ALOHA channels for access to radio channels in mobile telephony. In addition, SMS message texting was implemented in 2G mobile phones. In the early 2000s, additional ALOHA channels were added to 2.5G and 3G mobile phones with the widespread introduction of GPRS, using a slotted-ALOHA random-access channel combined with a version of the Reservation ALOHA scheme.

The original version of ALOHA used two distinct frequencies in a hub configuration, with the hub machine broadcasting packets to everyone on the "outbound" channel, and the various client machines sending data packets to the hub on the "inbound" channel. If data was received correctly at the hub, a short acknowledgment packet was sent to the client; if an acknowledgment was not received by a client machine after a short wait time, it would automatically retransmit the data packet after waiting a randomly selected time interval. This acknowledgment mechanism was used to detect and correct for "collisions" created when two client machines both attempted to send a packet at the same time.

ALOHAnet's primary importance was its use of a shared medium for client transmissions. Unlike the ARPANET where each node could only talk directly to a node at the other end of a wire or satellite circuit, in ALOHAnet all client nodes communicated with the hub on the same frequency. This meant that some sort of mechanism was needed to control who could talk at what time. The ALOHAnet solution was to allow each client to send its data without controlling when it was sent, with an acknowledgment/retransmission scheme used to deal with collisions.

This approach radically reduced the complexity of the protocol and the networking hardware, since nodes do not need to negotiate "who" is allowed to speak.

This solution became known as a pure ALOHA, or random-access channel, and was the basis for subsequent Ethernet development and later Wi-Fi networks. Various versions of the ALOHA protocol (such as Slotted ALOHA) also appeared later in satellite communications and were used in wireless data networks such as ARDIS, Mobitex, CDPD, and GSM. Also important was ALOHAnet's use of the outgoing hub channel to broadcast packets directly to all clients on a second shared frequency, using an address in each packet to allow selective receipt at each client node. Two frequencies were used so that a device could both receive acknowledgments regardless of transmissions. The Aloha network introduced the mechanism of randomized multiple access, which resolved device transmission collisions by transmitting a package immediately if no acknowledgement is present, and if no acknowledgment was received, the transmission was repeated after a random waiting time.

10.4.1 Pure ALOHA

If you have data to send, send the data If, while you are transmitting data, you receive any data from another station, there has been a message collision. All transmitting stations will need to try resending "later". Note that the first step implies that Pure ALOHA does not check whether the channel is busy before transmitting. Since collisions can occur and data may have to be sent again, ALOHA cannot use 100% of the capacity of the communications channel. How long a station waits until it transmits, and the likelihood a collision occurs are interrelated, and both affect how efficiently the channel can be used. This means that the concept of "transmit later" is a critical aspect: the quality of the backoff scheme chosen significantly influences the efficiency of the protocol, the ultimate channel capacity, and the predictability of its behaviour.

To assess Pure ALOHA, there is a need to predict its throughput, the rate of (successful) transmission of frames. All frames have the same length. Stations cannot generate a frame while transmitting or trying to transmit. (That is, if a station keeps trying to send a frame, it cannot be allowed to generate more frames to send). The population of stations attempts to transmit (both new frames and old frames that collided) according to a Poisson distribution. Let

"T" refers to the time needed to transmit one frame on the channel, and let's define "frame-time" as a unit of time equal to T. Let "G" refer to the mean used in the Poisson distribution over transmission-attempt amounts: that is, on average, there are G transmission-attempts per frame-time.

Consider what needs to happen for a frame to be transmitted successfully. Let "t" refer to the time at which it is intended to send a frame. It is preferable to use the channel for one frame-time beginning at t, and all other stations to refrain from transmitting during this time.

Disadvantages of Pure ALOHA:

1) Time is wasted

2) Data is lost

10.4.2 Slotted ALOHA

An improvement to the original ALOHA protocol was "Slotted ALOHA", which introduced discrete timeslots and increased the maximum throughput. A station can start a transmission only at the beginning of a timeslot, and thus collisions are reduced. In this case, only transmission-attempts within 1 frame-time and not 2 consecutive frame-times need to be considered, since collisions can only occur during each timeslot. Slotted ALOHA is used in low-data-rate tactical satellite communications networks by military forces, in subscriber-based satellite communications networks, mobile telephony call setup, set-top box communications and in the contactless RFID technologies. The basic variant of the Aloha protocol that we're going to start with is simple, and as follows:

> If a node is backlogged, it sends a packet from its queue with probability p. we will assume that each packet is exactly one slot in length. Such a system is also called slotted Aloha. We have not specified what p is; we will figure that out later, once we analyze the protocol as a function of p. Suppose there are N backlogged nodes and each node uses the same value of p. We can then calculate the utilization of the shared medium as a function of N and p by simply counting the number of slots in which exactly one node sends a packet. By definition,

a slot with 0 or greater than 1 transmission does not correspond to a successfully delivered packet, and therefore does not contribute toward the utilization.

If each node sends with probability p, then the probability that exactly one node sends in any given slot is $N_p(1 - p)^{N-1}$. The reason is that the probability that a specific node sends in the time slot is p, and for its transmission to be successful, all the other nodes should not send. That combined probability is $p(1 - p)^{N-1}$. Now, we can pick the successfully transmitting node in N ways, so the probability of exactly one node sending in a slot is $N_p(1 - p)^{N-1}$.

This quantity is the utilization achieved by the protocol because it is the fraction of slots that count toward useful throughput. Hence,

$$^{U}\text{Slotted Aloha}(p) = N_p(1 - p)^{N-1} \qquad \text{eq. 10.2}$$

for N = 8 as a function of p. The maximum value of U occurs when p = 1/N, and is equal to $(1 - 1\ )^{N-1}$. As N → ∞, U → 1/e ≈ 37%. This N result is an important one: the maximum utilization of slotted Aloha for a large number of backlogged nodes is roughly 1/e. 37% might seem like a small value (after all, the majority of the slots are being wasted), but notice that the protocol is extremely simple and has the virtue that it is hard to botch its implementation! It is fully distributed and requires no coordination or other specific communication between the nodes. That simplicity in system design is worth a lot oftentimes, it's a very good idea to trade simplicity off for high performance, and worry about optimization only when a specific part of the system is likely to become (or already has become) a bottleneck. That said, the protocol as described thus far requires a way to set p. Ideally, if each node knew the value of N, setting p = 1/N achieves the maximum. Unfortunately, this isn't as simple as it sounds because N here is the number of backlogged nodes that currently have data in their queues. The question then is: how can the nodes pick the best p? We turn to this important question next, because without such a mechanism, the protocol is impractical.

10.5 Carrier Sense Multiple Access (CSMA)

So far, we have assumed that no two nodes using the shared medium can hear each other. This assumption is true in some networks, notably the satellite network example mentioned here. Over a wired Ethernet, it is decidedly not true, while over wireless networks, the assumption is sometimes true and sometimes not (if there are three nodes A, B, and C, such that A and C can't usually hear each other, but B can usually hear both A and C, then A and C are said to be hidden terminals). The ability to first listen on the medium before attempting a transmission can be used to reduce the number of collisions and improve utilization. The technical term given for this capability is called carrier sense: a node, before it attempts a transmission, can listen to the medium to see if the analog voltage or signal level is higher than if the medium were unused, or even attempt to detect if a packet transmission is in progress by processing ("demodulating", a concept we will see in later lectures) a set of samples. Then, if it determines that another packet transmission is in progress, it considers the medium to be busy, and defers its own transmission attempt until the node considers the medium to be idle. The idea is for a node to send only when it believes the medium to be idle. One can modify the stabilized version of Aloha described above to use CSMA. One advantage of CSMA is that it no longer requires each packet to be one time slot long to achieve good utilization; packets can be larger than a slot duration, and can also vary in length. Note, however, that in any practical implementation, it will take some time for a node to detect that the medium is idle after the previous transmission ends, because it takes time to integrate the signal or sample information received and determine that the medium is indeed idle. This duration is called the detection time for the protocol. Moreover, multiple backlogged nodes might discover an "idle" medium at the same time; if they both send data, a collision ensues. For both these reasons, CSMA does not achieve 100% utilization, and needs a backoff scheme, though it usually achieves higher utilization than stabilized slotted Aloha over a single shared medium. You will investigate this protocol in the lab.

Carrier-sense multiple access (CSMA) is a media access control (MAC) protocol in which a node verifies the absence of other traffic before transmitting on a shared transmission medium, such as an electrical bus or a band of the electromagnetic spectrum. A transmitter attempts to determine whether another transmission is in progress before initiating a transmission using a carrier-sense mechanism. That is, it tries to detect the presence of a carrier signal from another node before attempting to transmit. If a carrier is sensed, the node waits for the transmission in progress to end before initiating its own transmission. Using CSMA, multiple nodes may, in turn, send and receive on the same medium. Transmissions by one node are generally received by all other nodes connected to the medium. Variations on basic CSMA include the addition of collision-avoidance, collision-detection, and collision-resolution techniques

10.5.1 CSMA access modes

1-persistent

1-persistent CSMA is an aggressive transmission algorithm. When the transmitting node is ready to transmit, it senses the transmission medium for idle or busy. If idle, then it transmits immediately. If busy, then it senses the transmission medium continuously until it becomes idle, then transmits the message (a frame) unconditionally (i.e. with probability=1). In the case of a collision, the sender waits for a random period of time and attempts the same procedure again. 1-persistent CSMA is used in CSMA/CD systems including Ethernet.

Non-persistent

Non-persistent CSMA is a non-aggressive transmission algorithm. When the transmitting node is ready to transmit data, it senses the transmission medium for idle or busy. If idle, then it transmits immediately. If busy, then it waits for a random period of time (during which it does not sense the transmission medium) before repeating the whole logic cycle (which started with sensing the transmission medium for idle or busy) again. This approach reduces collision, results in overall higher medium throughput but with a penalty of longer initial delay compared to 1–persistent.

P-persistent

This is an approach between 1-persistent and non-persistent CSMA access modes. When the transmitting node is ready to transmit data, it senses the transmission medium for idle or busy. If idle, then it transmits immediately. If busy, then it senses the transmission medium continuously until it becomes idle, then transmits with probability p. If the node does not transmit (the probability of this event is 1-p), it waits until the next available time slot. If the transmission medium is not busy, it transmits again with the same probability p. This probabilistic hold-off repeats until the frame is finally transmitted or when the medium is found to become busy again (i.e. some other node has already started transmitting). In the latter case, the node repeats the whole logic cycle (which started with sensing the transmission medium for idle or busy) again. p-persistent CSMA is used in CSMA/CA systems including Wi-Fi and other packet radio systems.

O-persistent

Each node is assigned a transmission order by a supervisory node. When the transmission medium goes idle, nodes wait for their time slot in accordance with their assigned transmission order. The node assigned to transmit first transmits immediately. The node assigned to transmit second waits one time slot (but by that time the first node has already started transmitting). Nodes monitor the medium for transmissions from other nodes and update their assigned order with each detected transmission (i.e. they move one position closer to the front of the queue). O-persistent CSMA is used by CobraNet, LonWorks and the controller area network. When broadcasting over vehicular ad hoc networks, the original 1-persistence and p-persistence strategies often cause the broadcast storm problem. To improve performance, engineers developed three modified techniques: weighted p-persistence, slotted 1-persistence, and slotted p-persistence.

10.5.2 Carrier-sense multiple access with collision detection

CSMA/CD is used to improve CSMA performance by terminating transmission as soon as a collision is detected, thus shortening the time required before a retry can be attempted. CSMA/CD is used by Ethernet.

10.5.3 Carrier-sense multiple access with collision avoidance

In CSMA/CA collision avoidance is used to improve the performance of CSMA. If the transmission medium is sensed busy before transmission, then the transmission is deferred for a random interval. This random interval reduces the likelihood that two or more nodes waiting to transmit will simultaneously begin transmission upon termination of the detected transmission, thus reducing the incidence of collision. CSMA/CA is used by Wi-Fi.

10.5.4 CSMA with Collision Resolution

CSMA/CR uses priorities in the frame header to avoid collisions. It is used in the Controller Area Network

10.5.5 Virtual time CSMA

VTCSMA is designed to avoid collision generated by nodes transmitting signals simultaneously, used mostly in hard real-time systems. It uses two clocks to prioritize messages based on their deadline. Reduce collisions with on-going transmissions by transmitting only if a channel appears not to be busy.

• For large T (slots/packet) if the channel is busy this cycle, the same sender will probably be transmitting more of their packet next cycle

• When the channel is idle, there's no chance of interrupting an on-going transmission.

• That leaves the possibility of colliding with another transmission that starts at the same time – a one slot window of vulnerability, not 2T-1 slots.

• Expect collisions to drop dramatically, utilization to be quite a bit better, although a "wasted" slot is now necessary

• Busy = detect energy on the channel. On wireless channels, transmitters turn on the carrier to transmit (we'll learn more about this after the break), hence the term "carrier sense.

10.6 A Note on Implementation: Contention Windows

In the protocols described so far, each backlogged node sends a packet with probability p, and the job of the protocol is to adapt p in the best possible way. With CSMA, the idea is to send with this probability but only when the medium is idle. In practice, many contention protocols such as the IEEE 802.3 (Ethernet) and 802.11 (Wi-Fi) standards do something a little different: rather than each node transmitting with a probability in each time slot, they use the concept of a contention window. A contention window scheme works as follows. Each node maintains its own current value of the window, which we call CW. CW can vary between CWmin and CWmax; CWmin may be 1 and CWmax may be a number like 1024. When a node decides to transmit, it does so by picking a random number r uniformly in [1, CW] and sends in time slot C + r, where C is the current time slot. If a collision occurs, the node doubles CW; on a successful transmission, a node halves CW (or, as is often the case in practice, directly resets it to CWmin). You should note that this scheme is similar to the one we studied and analyzed above. The doubling of CW is analogous to halving the transmission probability, and the halving of CW is analogous to doubling the probability (CW has a lower bound; the transmission probability has an upper bound).

But there are two crucial differences:

1. Transmissions with a contention window are done according to a uniform probability distribution and not a geometrically distributed one. In the previous case, the a priori probability that the first transmission occurs t slots from now is geometrically distributed; it is p (1 − p)t−1, while with a contention window, it is equal to 1/CW for t ∈ [1,CW] and 0 otherwise. This means that each node is guaranteed to attempt a transmission within CW slots, while that is not the case in the previous scheme, where there is always a chance, though exponentially decreasing, that a node may not transmit within any fixed number of slots.

2. The second difference is more minor: each node can avoid generating a random number in each slot; instead, it can generate a random number once per packet transmission attempt. In the lab, you will implement the key parts of the contention window protocol and experiment with it in conjunction with CSMA. There is one important subtlety to keep in mind while doing this implementation. The issue has to do with how to count the slots before a node decides to

transmit. Suppose a node decides that it will transmit x slots from now as long as the medium is idle after x slots; if x includes the busy slots when another node transmits, then multiple nodes may end up trying to transmit in the same time slot after the ending of a long packet transmission from another node, leading to excessive collisions. So, it is important to only count down the idle slots; i.e., x should be the number of idle slots before the node attempts to transmit its packet (and of course, a node should try to send a packet in a slot only if it believes the medium to be idle in that slot).

Summary of Study 10

This lecture discussed the issues involved in sharing a communication medium amongst multiple nodes. We focused on contention protocols, developing ways to make them provide reasonable utilization and fairness. This is what we learned:

1. Good MAC protocols optimize utilization (throughput) and fairness, but must be able to solve the problem in a distributed way. In most cases, the overhead of a central controller node knowing which nodes have packets to send is too high. These protocols must also provide good utilization and fairness under dynamic load.

2. TDMA provides high throughput when all (or most of) the nodes are backlogged and the offered loads is evenly distributed amongst the nodes. When per-node loads are busty or when different nodes send different amounts of data, TDMA is a poor choice.

3. Slotted Aloha has surprisingly high utilization for such a simple protocol, if one can pick the transmission probability correctly. The probability that maximizes throughput is 1/N, where N is the number of backlogged nodes, the resulting utilization tends toward $1/e \approx 37\%$, and the fairness is close to 1 if all nodes present the same load. The utilization does remains high even when the nodes present different loads, in contrast to TDMA. It is also worth calculating (and noting) how many slots are left idle and how many slots have more than one node transmitting at the same time in slotted Aloha with $p = 1/N$. When N is large, these numbers are $1/e$ and $1 - 2/e \approx 26\%$, respectively. It is interesting that the number of idle slots is the same as the utilization: if we increase p to reduce the number of idle slots, we don't increase the utilization but actually increase the collision rate.

4. Stabilization is crucial to making Aloha practical. We studied a scheme that adjusts the transmission probability, reducing it multiplicatively when a collision occurs and increasing it (either multiplicatively or to a fixed maximum value) when a successful transmission occurs. The idea is to try to converge to the optimum value.

5. A non-zero lower bound on the transmission probability is important if we want to improve fairness, in particular to prevent some nodes from being starved. An upper bound smaller than 1 improves fairness over shorter time scales by alleviating the capture effect, a situation where one or a small number of nodes capture all the transmission attempts for many time slots in succession.

6. Slotted Aloha has double the utilization of unslotted Aloha when the number of backlogged nodes grows. The intuitive reason is that if two packets are destined to collide, the "window of vulnerability" is larger in the unslotted case by a factor of two.

7. A broadcast network that uses packets that are multiple slots in length (i.e., mimicking the unslotted case) can use carrier sense if the medium is a true broadcast medium (or approximately so). In a true broadcast medium, all nodes can hear each other reliably, so they can sense the carrier before transmitting their own packets. By "listening before transmitting" and setting the transmission probability using stabilization, they can reduce the number of collisions and increase utilization, but it is hard (if not impossible) to eliminate all collisions. Fairness still requires bounds on the transmission probability as before.

8. With a contention window, one can make the transmissions from backlogged nodes occur according to a uniform distribution, instead of the geometric distribution imposed by the "send with probability p" schemes. A uniform distribution in a finite window guarantees that each node will attempt a transmission within some fixed number of slots, which is not true of the geometric distribution.

Self-Assessment Question (SAQs) for lecture 10

SAQ 10.1

We studied TDMA, (stabilized) Aloha, and CSMA protocols in this chapter. In each statement below, assume that the protocols are implemented correctly. Which of these statements is true (more than might be)?

(a) TDMA may have collisions when the size of a packet exceeds one time slot.

(b) There exists some offered load for which TDMA has lower throughput than slotted Aloha.

(c) In stabilized Aloha, two nodes have a certain probability of colliding in a time slot. If they actually collide in that slot, then they will experience a lower probability of colliding with each other when they each retry.

(d) There is no workload for which stabilized Aloha achieves a utilization greater that (1 − $1/N^{)N-1}$ (≈ 1/e for large N) when run for a long period of time.

(e) In slotted Aloha with stabilization, each node's transmission probability converges to 1/N, where N is the number of backlogged nodes.

(f) In a network in which all nodes can hear each other, CSMA will have no collisions when the packet size is larger than one time slot.

SAQ 10.2

In the Aloha stabilization protocols, we studied, when a node experiences a collision, it decreases its transmission probability, but sets a lower bound, p_{min}. When it transmits successfully, it increases its transmission probability, but sets an upper bound, p_{max}.

(a) Why would we set a lower bound on pmin that is not too close to 0?

(b) Why would we set pmax to be significantly smaller than 1?

(c) Let N be the average number of backlogged nodes. What happens if we set $p_{min} >> 1/N$?

SAQ 10.3

Alyssa and Ben are all on a shared medium wireless network running a variant of slotted Aloha (all packets are the same size and each packet fits in one slot). Their computers are configured such that Alyssa is 1.5 times as likely to send a packet as Ben. Assume that both computers are backlogged.

(a) For Alyssa and Ben, what is their probability of transmission such that the utilization of their network is maximized?

SAQ 10.4

You have two computers, A and B, sharing a wireless network in your room. The network runs the slotted Aloha protocol with equal-sized packets. You want B to get twice the throughput over the wireless network as A whenever both nodes are backlogged. You configure A to send packets with probability p. What should you set the transmission probability of B to, in order to achieve your throughput goal?

SAQ 10.5

Which of the following statements are always true for networks with $N > 1$ nodes using correctly implemented versions of unslotted Aloha, slotted Aloha, Time Division Multiple Access (TDMA) and Carrier Sense Multiple Access (CSMA)? Unless otherwise stated, assume that the slotted and unslotted versions of Aloha are stabilized and use the same stabilization method and parameters. Explain your answer for each statement.

(a) There exists some offered load pattern for which TDMA has lower throughput than slotted Aloha.

(b) Suppose nodes I, II and III use a fixed probability of $p = 1/3$ when transmitting on a 3-node slotted Aloha network (i.e., $N = 3$). If all the nodes are backlogged then over time the utilization averages out to $1/e$.

(c) When the number of nodes, N, is large in a stabilized slotted Aloha network, setting $p_{max} = p_{min} = 1/N$ will achieve the same utilization as a TDMA network if all the nodes are backlogged.

(d) Using contention windows with a CSMA implementation guarantees that a packet will be transmitted successfully within some bounded time.

(b) What is the maximum utilization?

Study Session 11

11.1 Introduction

Thus far we have studied techniques to engineer a point-to-point communication link to send messages between two directly connected devices. These techniques give us a communication link between two devices that, in general, has a certain error rate and a corresponding message loss rate. Message losses occur when the error correction mechanism is unable to correct all the errors that occur due to noise or interference from other concurrent transmissions in a contention MAC protocol. We now turn to the study of multi-hop communication networks systems that connect three or more devices together. The key idea that we will use to engineer communication networks is composition: we will build small networks by composing links together, and build larger networks by composing smaller networks together. The fundamental challenges in the design of a communication network are the same as those that face the designer of a communication link: sharing for efficiency and reliability. The big difference is that the sharing problem has different challenges because the system is now distributed, spread across a geographic span that is much larger than even the biggest shared medium we can practically build. Moreover, as we will see, many more things can go wrong in a network in addition to just bit errors on the point-to-point links, making communication more unreliable than a single link's unreliability. We will discuss these two challenges and the key principles to overcome them. In addition to sharing and reliability, an important and difficult problem that many communication networks (such as the Internet) face is scalability: how to engineer a very large, global system. We won't say very much about scalability in this book, leaving this important topic for more advanced courses. This chapter focuses on the sharing problem and discusses the following concepts:

1. Switches and how they enable multiplexing of different communications on individual links and over the network. Two forms of switching: circuit switching and packet switching.

2. Understanding the role of queues to absorb bursts of traffic in packet-switched networks.

3. Understanding the factors that contribute to delays in networks: three largely fixed delays (propagation, processing, and transmission delays), and one significant variable source of delays (queueing delays).

4. Little's law, relating the average delay to the average rate of arrivals and the average queue size.

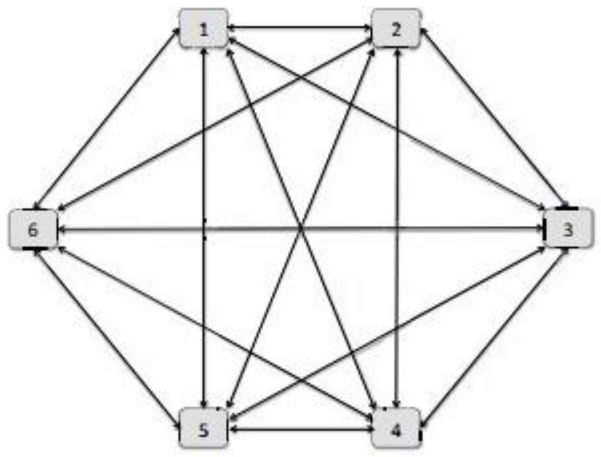


Figure 11-1: A communication network with a link between every pair of devices has a quadratic number of links. Such topologies are generally too expensive, and are especially untenable when the devices are far from each other.

11.1 Sharing with Switches

The collection of techniques used to design a communication link, including modulation and error-correcting channel coding, is usually implemented in a module called the physical layer (or "PHY" for short). The sending PHY takes a stream of bits and arranges to send it across the link to the receiver; the receiving PHY provides its best estimate of the stream of bits sent from the other end. On the face of it, once we know how to develop a communication link, connecting a collection of N devices together is ostensibly quite straightforward: one could simply connect each pair of devices with a wire and use the physical layer running over the wire to communicate between the two devices. This picture for a small 5-node network is shown in Figure 11-1. This simple strawman using dedicated pairwise links has two severe problems. First, it is extremely expensive. The reason is that the number of distinct communication links that one needs to build scales quadratically with N—there are N/2 = N(N −1)/2 bi-directional links in this design (a bi-directional link is one that can transmit data in both directions, as

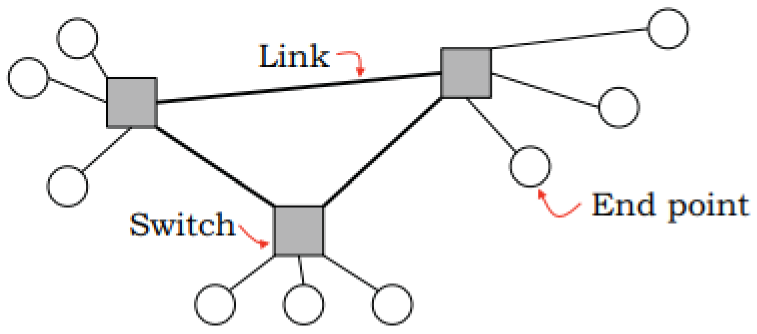


Figure 11-2: A simple network topology showing communicating end points, links, and switches.

opposed to a uni-directional link). The cost of operating such a network would be prohibitively expensive, and each additional node added to the network would incur a cost proportional to the size of the network. Second, some of these links would have to span an enormous distance. Such "long-haul" links are difficult to engineer, so one can't assume that they will be available in abundance. Clearly, we need a better design, one that can "do for a dime what any fool can do for a dollar". The key to a practical design of a communication network is a special computing device called **a switch**. A switch has multiple "interfaces" (often also called "ports") on it; a link (wire or radio) can be connected to each interface. The switch allows multiple different communications between different pairs of devices to run over each individual link that is, it arranges for the network's links to be shared by different communications. In addition to the links, the switches themselves have some resources (memory and computation) that will be shared by all the communicating devices. Figure 11-2 shows the general idea. A switch receives bits that are encapsulated in data frames arriving over its links, processes them (in a way that we will make precise later), and forwards them (again, in a way that we will make precise later) over one or more other links. In the most common kind of network, these frames are called packets, as explained below. We will use the term end points to refer to the communicating devices, and call the switches and links over which they communicate the network infrastructure. The resulting structure is termed the network topology, and consists of nodes (the switches and end points) and links. A simple network topology is shown in Figure 11-2. We will model the network topology as a graph, consisting of a set of nodes and a set of links (edges) connecting various nodes together, to solve various problems.

11.1.1 Three Problems That Switches Solve

The fundamental functions performed by switches are to multiplex and demultiplex data frames belonging to different device-to-device information transfer sessions, and to determine the link(s) along which to forward any given data frame. This task is essential because a given physical link will usually be shared by several concurrent sessions between different devices. We break these functions into three problems:

1. **Forwarding:** When a data frame arrives at a switch, the switch needs to process it, determine the correct outgoing link, and decide when to send the frame on that link.

2**. Routing**: Each switch somehow needs to determine the topology of the network, so that it can correctly construct the data structures required for proper forwarding. The process by which the switches in a network collaboratively compute the network topology, adapting to various kinds of failures, is called routing. It does not happen on each data frame, but occurs in the "background". The next two chapters will discuss forwarding and routing in more detail.

3. **Resource allocation:** Switches allocate their resources access to the link and local memory to the different communications that are in progress.

Over time, two radically different methods have been developed for solving these problems. These techniques differ in the way the switches forward data and allocate resources (there are also some differences in routing, but they are less significant). The first method, used by networks like the telephone network, is called circuit switching. The second method, used by networks like the Internet, is called packet switching. There are two crucial differences between the two methods, one philosophical and the other mechanistic. The mechanistic difference is the easier one to understand, so we'll talk about it first. In a circuit-switched network, the frames do not (need to) carry any special information that tells the switches how to forward information, while in packet-switched networks, they do. The philosophical difference is more substantive: a circuit-switched network provides the abstraction of a dedicated link of some bit rate to the communicating entities, whereas a packet switched network does not. Of course, this dedicated link traverse multiple physical links and at least one switch, so the end points and switches must do some additional work to provide the illusion of a dedicated link. A packet-switched network, in contrast, provides no such illusion; once again, the end points and switches must do some work to provide reliable and efficient communication service to the applications running on the end points.

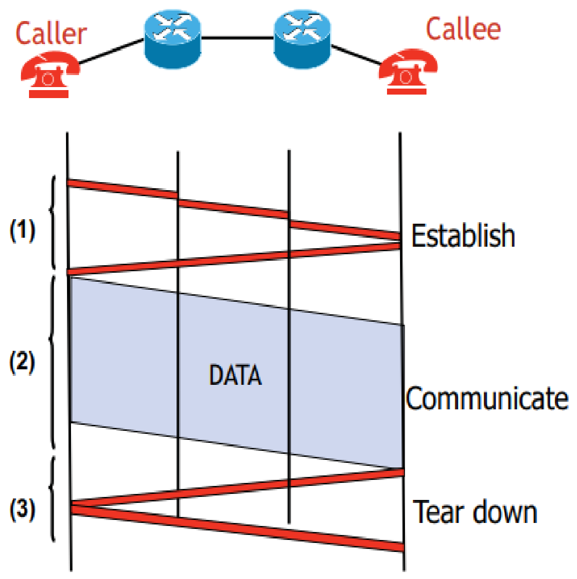


Figure 11-3: Circuit switching requires setup and teardown phases.

11.2 Circuit Switching

The transmission of information in circuit-switched networks usually occurs in three phases (see Figure 11-3):

1. The setup phase, in which some state is configured at each switch along a path from source to destination,

2. The data transfer phase when the communication of interest occurs, and

3. The teardown phase that cleans up the state in the switches after the data transfer ends.

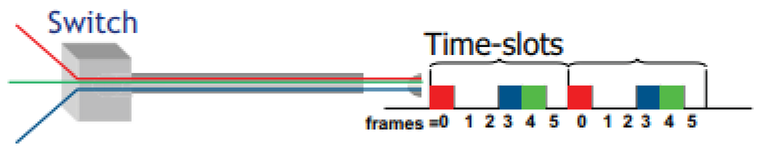
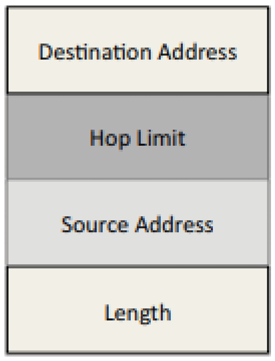


Figure 11-4: Circuit switching with Time Division Multiplexing (TDM). Each color is a different conversation and there is a maximum of N = 6 concurrent communications on the link in this picture. Each communication (color) is sent in a fixed time-slot, modulo N

Because the frames themselves contain no information about where they should go, the setup phase needs to take care of this task, and also configure (reserve) any resources needed for the communication so that the illusion of a dedicated link is provided. The teardown phase is needed to release any reserved resources.

11.3 Packet Switching

An attractive way to overcome the inefficiencies of circuit switching is to permit any sender to transmit data at any time, but yet allow the link to be shared. Packet switching is a way to accomplish this task, and uses a tantalizingly simple idea: add to each frame of data a little bit of information that tells the switch how to forward the frame. This information is usually added inside a header immediately before the payload of the frame, and the resulting frame is called a packet. In the most common form of packet switching, the header of each packet contains the address of the destination, which uniquely identifies the destination of data. The switches use this information to process and forward each packet. Packets usually also include the sender's address to help the receiver send messages back to the sender. A simple example of a packet header is shown in Figure 11-5. In addition to the destination and source addresses, this header shows a checksum that can be used for error detection at the receiver. The figure also shows the packet header used by IPv6 (the Internet Protocol version 6), which is increasingly used on the Internet today. The Internet is the most prominent and successful example of a packet-switched network. The job of the switch is to use the destination address as a key and perform a lookup on



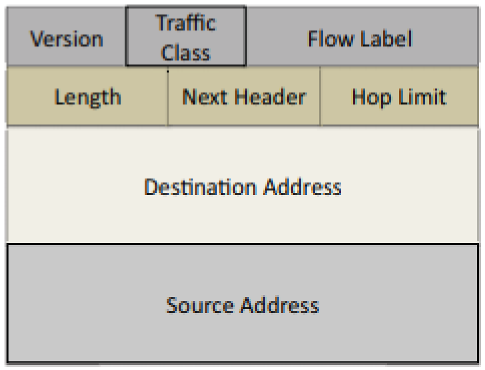


Figure 11-5: LEFT: A simple and basic example of a packet header for a packet-switched network. The destination address is used by switches in the forwarding process. The hop limit field will be explained in the chapter on network routing; it is used to discard packets that have been forwarded in the network for more than a certain number of hops, because it's likely that those packets are simply stuck in a loop. Following the header is the payload (or data) associated with the packet, which we haven't shown in this picture. RIGHT: For comparison, the format of the IPv6 ("IP version 6") packet header is shown. Four of the eight fields are similar to our simple header format. The additional fields are the version number, which specifies the version of IP, such as "6" or "4" (the current version that version 6 seeks to replace) and fields that specify, or hint at, how switches must prioritize or provide other traffic management features for the packet.

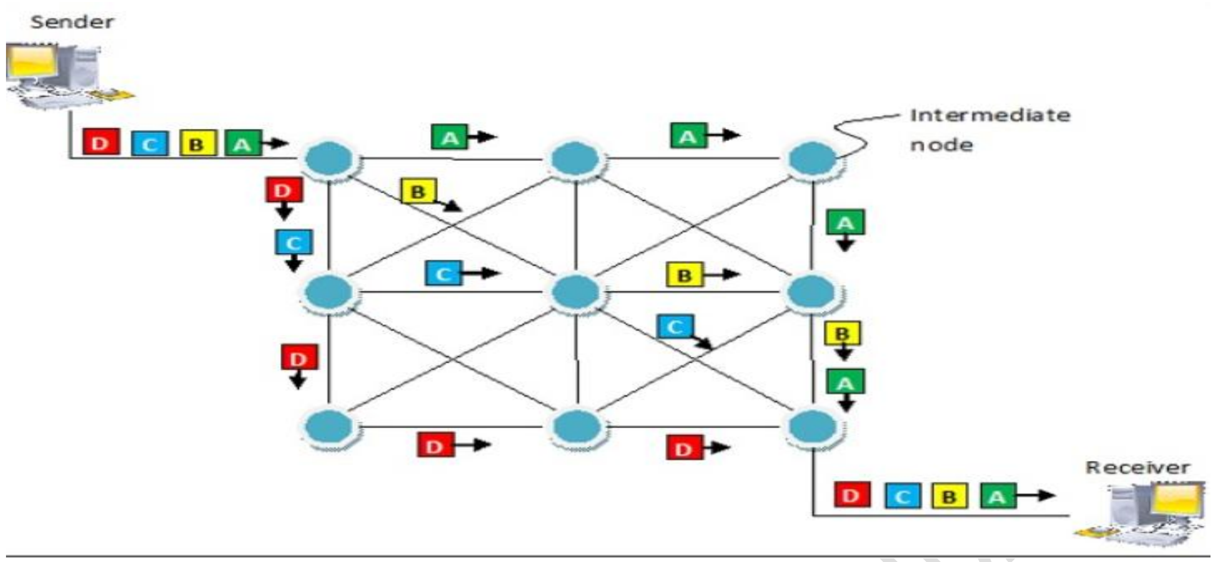
a data structure called a routing table. This lookup returns an outgoing link to forward the packet on its way toward the intended destination. There are many ways to implement the lookup operation on a routing table, but for our purposes we can consider the routing table to be a dictionary mapping each destination to one of the links on the switch. While forwarding is a relatively simple lookup in a data structure, the trickier question that we will spend time on is determining how the entries in the routing table are obtained. The plan is to use a background process called a routing protocol, which is typically implemented in a distributed manner by the switches. There are two common classes of routing protocols, which we will study in later chapters. For now, it is enough to understand that if the routing protocol works as expected, each switch obtains a route to every destination. Each switch participates in the routing protocol, dynamically constructing and updating its routing table in response to information received from its neighbors, and providing information to each neighbor to help them construct their own routing tables. Switches in packet-switched networks that implement the functions described in this section are also known as routers, and we will use the terms “switch” and “router” interchangeably when talking about packet-switched networks.

Packet switching is a connectionless network switching technique. Here, the message is divided and grouped into a number of units called packets that are individually routed from the source to the destination. There is no need to establish a dedicated circuit for communication.

11.3.1 Process

Each packet in a packet switching technique has two parts: a header and a payload. The header contains the addressing information of the packet and is used by the intermediate routers to direct it towards its destination. The payload carries the actual data. A packet is transmitted as soon as it is available in a node, based upon its header information. The packets of a message are not routed via the same path. So, the packets in the message arrive in the destination out of order. It is the responsibility of the destination to reorder the packets in order to retrieve the original message.

The process is diagrammatically represented in the following figure. Here the message comprises four packets, A, B, C, and D, which may follow different routes from the sender to the receiver.



11.3.2 Advantages and Disadvantages of Packet Switching

Advantages

- Delay in the delivery of packets is less since packets are sent as soon as they are available.
- Switching devices don't require massive storage since they don't have to store the entire messages before forwarding them to the next node.
- Data delivery can continue even if some parts of the network faces link failure. Packets can be routed via other paths.
- It allows simultaneous usage of the same channel by multiple users.
- It ensures better bandwidth usage as a number of packets from multiple sources can be transferred via the same link.

Disadvantages

- They are unsuitable for applications that cannot afford delays in communication like high quality voice calls.
- Packet switching high installation costs.
- They require complex protocols for delivery.

- Network problems may introduce errors in packets, delay in delivery of packets or loss of packets. If not properly handled, this may lead to loss of critical information.

11.4 Little's Law

A common method used by engineers to analyze network performance, particularly delay and throughput (the rate at which packets are delivered), is queueing theory. In this course, we will use an important, widely applicable result from queueing theory, called Little's law (or Little's theorem). It's used widely in the performance evaluation of systems ranging from communication networks to factory floors to manufacturing systems. For any stable (i.e., where the queues aren't growing without bound) queueing system, little's law relates the average arrival rate of items (e.g., packets), λ, the average delay experienced by an item in the queue, D, and the average number of items in the queue, N. The formula is simple and intuitive:

$$N = \lambda \times D \qquad \text{eq.11.1}$$

Note that if the queue is stable, then the departure rate is equal to the arrival rate.

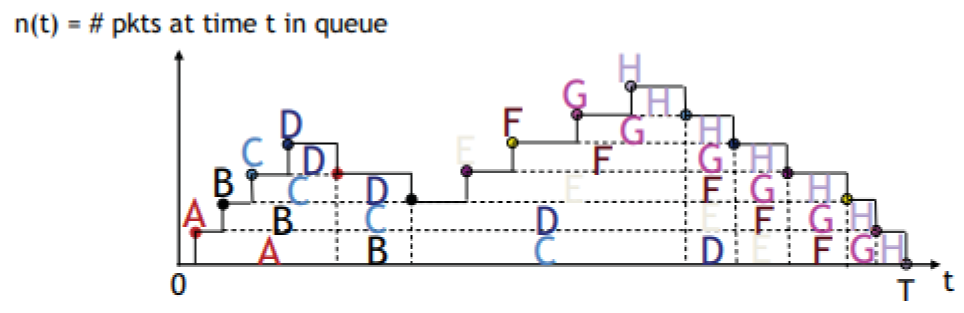


Figure 11-6: Packet arrivals into a queue, illustrating Little's law.

Example.

Suppose packets arrive at an average rate of 1000 packets per second into a switch, and the rate of the outgoing link is larger than this number. (If the outgoing rate is smaller, then the queue will grow unbounded.) It doesn't matter how inter-packet arrivals are distributed; packets could arrive in weird bursts according to complicated distributions. Now, suppose there are 50 packets in the queue on average. That is, if we sample the queue size at random points in time and take the average, the number is 50 packets. Then, from Little's law, we can conclude that the average queueing delay experienced by a packet is 50/1000 seconds = 50 milliseconds. Little's law is quite remarkable because it is independent of how items (packets) arrive or are serviced by the queue. Packets could arrive according to any distribution. They can be serviced in any order, not just first-in-first-out (FIFO). They can be of any size. In fact, about the only practical requirement is that the queueing system be stable. It's a useful result that can be used profitably in back-of-the-envelope calculations to assess the performance of real systems. Why does this result hold? Proving the result in its full generality is beyond the scope of this course, but we can show it quite easily with a few simplifying assumptions using an essentially pictorial argument. The argument is instructive and sheds some light into the dynamics of packets in a queue. Figure 11-6 shows n(t), the number of packets in a queue, as a function of time t. Each time a packet enters the queue, n(t) increases by 1. Each time the packet leaves, n(t) decreases by 1. The result is the step-wise curve like the one shown in the picture. For simplicity, we will assume that the queue size is 0 at time 0 and that there is some time T >> 0 at which the queue empties to 0. We will also assume that the queue services jobs in FIFO order (note that the formula holds whether these assumptions are true or not). Let P be the total number of packets forwarded by the switch in time T (obviously, in our special case when the queue fully empties, this number is the same as the number that entered the system). Now, we need to define N, λ, and D. One can think of N as the time average of the number of packets in the queue; i.e.,

$$N = \sum_{t=0}^{T} n(t)/T.$$

The rate λ is simply equal to P/T, for the system processed P packets in time T. D, the average delay, can be calculated with a little trick. Imagine taking the total area under the n(t) curve and assigning it to packets as shown in Figure 11-6. That is, packets A, B, C, ... each are assigned the different rectangles shown. The height of each rectangle is 1 (i.e., one packet) and the length is the time until some packet leaves the system. Each packet's rectangle(s) last until the packet itself leaves the system. Now, it should be clear that the time spent by any given packet is just the sum of the areas of the rectangles labeled by that packet. Therefore, the average delay experienced by a packet, D, is simply the area under the n(t) curve divided by the number of packets. That's because the total area under the curve, which is s ∑n(t), is the total delay experienced by all the packets.

Hence,

$$D = \sum_{t=0}^{T} n(t)/P.$$

From the above expressions, little's law follows: N = λ × D.

Little's law is useful in the analysis of networked systems because, depending on the context, one usually knows some two of the three quantities in Eq. (11.1), and is interested in the third. It is a statement about averages, and is remarkable in how little it assumes about the way in which packets arrive and are processed.

Self-Assessment Question (SAQs) for lecture 11

SAQ 11.1

Under what conditions would circuit switching be a better network design than packet switching?

SAQ 11.2

2. Which of these statements are correct?

(a) Switches in a circuit-switched network process connection establishment and tear-down messages, whereas switches in a packet-switched network do not.

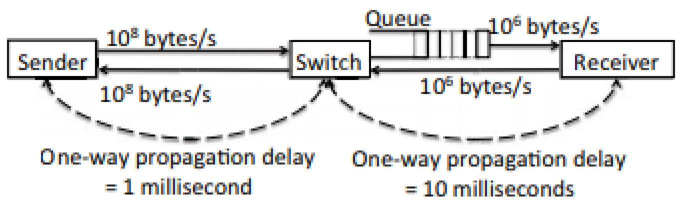
(b) Under some circumstances, a circuit-switched network may prevent some senders from starting new conversations.

(c) Once a connection is correctly established, a switch in a circuit-switched network can forward data correctly without requiring data frames to include a destination address.

(d) Unlike in packet switching, switches in circuit-switched networks do not need any information about the network topology to function correctly.

SAQ 11.3

Consider the network topology shown below. Assume that the processing delay at all the nodes is negligible


Sender
10^8 bytes/s
10^8 bytes/s
Switch
Queue
10^6 bytes/s
10^6 bytes/s
Receiver
One-way propagation delay = 1 millisecond
One-way propagation delay = 10 milliseconds

(a) The sender sends two 1000-byte data packets back-to-back with a negligible inter-packet delay. The queue has no other packets. What is the time delay between the arrival of the first bit of the second packet and the first bit of the first packet at the receiver?

(b) The receiver acknowledges each 1000-byte data packet to the sender, and each acknowledgment has a size A = 100 bytes. What is the minimum possible round-trip time between the sender and receiver? The round-trip time is defined as the duration between the transmission of a packet and the receipt of an acknowledgment for it.

Study Session 12:Network Routing Without Any Failures

12.1 Introduction

This chapter and the next one discusses the key technical ideas in network routing. We start by describing the problem, and break it down into a set of sub-problems and solve them. The key ideas that you should understand by the end are:

1. Addressing and forwarding.

2. Distributed routing protocols: distance-vector and link-state protocols.

3. How routing protocols handle failures and find usable paths.

The problem of finding paths in the network is challenging for the following reasons:

1. **Distributed information**: Each node only knows about its local connectivity, i.e., its immediate neighbors in the topology (and even determining that reliably needs a little bit of work, as we'll see). The network has to come up with a way to provide network-wide connectivity starting from this distributed information.

2. **Efficiency:** The paths found by the network should be reasonably "good"; they shouldn't be inordinately long in length, for that will increase the latency (delay) experienced by packets. For concreteness, we will assume that links have costs (these costs could model link latency, for example), and that we are interested in finding a path between any source and destination that minimizes the total cost. We will assume that all link costs are non-negative. Another aspect of efficiency that we must pay attention to is the extra network bandwidth consumed by the network in finding good paths.

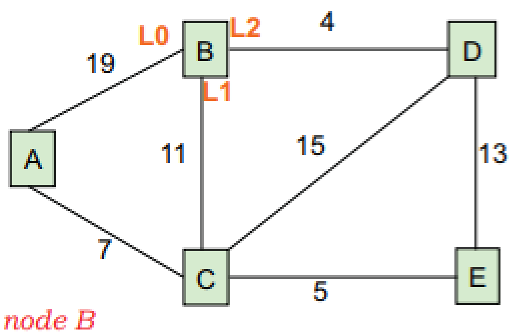
3. **Failures:** Links and nodes may fail and recover arbitrarily. The network should be able to find a path if one exists, without having packets get "stuck" in the network forever because of glitches. To cope with the churn caused by the failure and recovery of links and switches, as

well as by new nodes and links being set up or removed, any solution to this problem must be dynamic and continually adapt to changing conditions. In this description of the problem, we have used the term "network" several times while referring to the entity that solves the problem. The most common solution is for the network's switches to collectively solve the problem of finding paths that the end points' packets take. Although network designs where end points take a more active role in determining the paths for their packets have been proposed and are sometimes used, even those designs require the switches to do the hard work of finding a usable set of paths. Hence, we will focus on how switches can solve this problem. Clearly, because the information required for solving the problem is spread across different switches, the solution involves the switches cooperating with each other. Such methods are examples of distributed computation.

Our solution will be in three parts: first, we need a way to name the different nodes in the network. This task is called addressing. Second, given a packet with the name of a destination in its header we need a way for a switch to send the packet on the correct outgoing link. This task is called forwarding. Finally, we need a way by which the switches can determine how to send a packet to any destination, should one arrive. This task is done in the background, and continuously, building and updating the data structures required for forwarding to work properly. This background task, which will occupy most of our time, is called routing.

12.2 Addressing and Forwarding

Clearly, to send packets to some end point, we need a way to uniquely identify the end point. Such identifiers are examples of names, a concept commonly used in computer systems: names provide a handle that can be used to refer to various objects. In our context, we want to name end points and switches. We will use the term address to refer to the name of a switch or an end point. For our purposes, the only requirement is that addresses refer to end points and switches uniquely. In large networks, we will want to constrain how addresses are assigned, and distinguish between the unique identifier of a node and its addresses. The distinction will allow us to use an address to refer to each distinct network link (aka "interface") available on a node; because a node may have multiple links connected to it, the unique name for a node is distinct from the addresses of its interfaces (if you have a computer with multiple active network interfaces, say a wireless link and an Ethernet, then that computer will have multiple addresses, one for each active interface). In a packet-switched network, each packet sent by a sender contains the address of the destination. It also usually contains the address of the sender, which allows applications and other protocols running at the destination to send packets back. All this information is in the packet's header, which also may include some other useful fields. When a switch gets a packet, it consults a table keyed by the destination address to determine which link to send the packet on in order to reach the destination. This process is a table lookup, and the table in question is called the routing table. 2 The selected link is called the outgoing link.



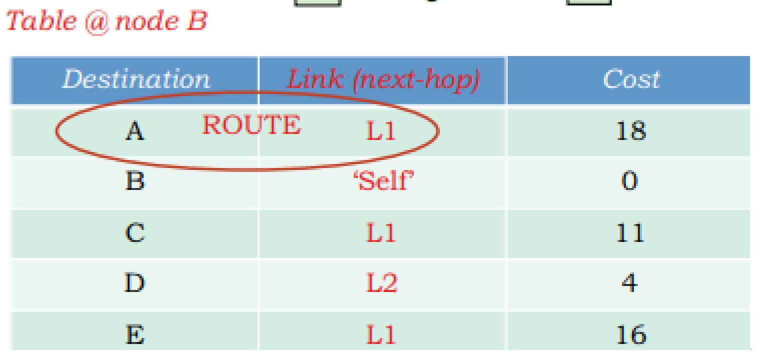


Table @ node B

Destination	Link (next-hop)	Cost
A	ROUTE L1	18
B	'Self'	0
C	L1	11
D	L2	4
E	L1	16

Figure 12-1: A simple network topology showing the routing table at node B. The route for a destination is marked with an oval. The three links at node B are L0, L1, and L2; these names aren't visible at the other nodes but are internal to node B.

The combination of the destination address and outgoing link is called the route used by the switch for the destination. Note that the route is different from the path between source and destination in the topology; the sequence of routes at individual switches produces a sequence of links, which in turn leads to a path (assuming that the routing and forwarding procedures are working correctly). Figure 12-1 shows a routing table and routes at a node in a simple network. Because data may be corrupted when sent over a link (uncorrected bit errors) or because of bugs in switch implementations, it is customary to include a checksum that covers the packet's header, and possibly also the data being sent. These steps for forwarding work as long as there are no failures in the network. We will use a "hop limit" field in the packet header to detect and discard packets that are being repeatedly forwarded by the nodes without finding their way to the intended destination.

12.3 Overview of Routing

If you don't know where you are going, any road will take you there. —Lewis Carroll Routing is the process by which the switches construct their routing tables. At a high level, most routing protocols have three components:

1. **Determining neighbors:** For each node, which directly linked nodes are currently both reachable and running? We call such nodes neighbors of the node in the topology. A node may not be able to reach a directly linked node either because the link has failed or because the node itself has failed for some reason. A link may fail to deliver all packets (e.g., because a backhoe cuts cables), or may exhibit a high packet loss rate that prevents all or most of its packets from being delivered. For now, we will assume that each node knows who its neighbors are. In the next chapter, we will discuss a common approach, called the HELLO protocol, by which each node determines who its current neighbors are. The basic idea if for each node to send periodic "HELLO" messages on all its live links; any node receiving a HELLO knows that the sender of the message is currently alive and a valid neighbor.

2. **Sending advertisements:** Each node sends routing advertisements to its neighbors. These advertisements summarize useful information about the network topology. Each node sends these advertisements periodically, for two reasons. First, in vector protocols, periodic advertisements ensure that over time the nodes all have all the information necessary to compute correct routes. Second, in both vector and link-state protocols, periodic

advertisements are the fundamental mechanism used to overcome the effects of link and node failures (as well as packet losses).

3. **Integrating advertisements:** In this step, a node processes all the advertisements it has recently heard and uses that information to produce its version of the routing table. Because the network topology can change and because new information can become available, these three steps must run continuously, discovering the current set of neighbors, disseminating advertisements to neighbors, and adjusting the routing tables. This continual operation implies that the state maintained by the network switches is soft: that is, it refreshes periodically as updates arrive, and adapts to changes that are represented in these updates. This soft state means that the path used to reach some destination could change at any time, potentially causing a stream of packets from a source to destination to arrive reordered; on the positive side, however, the ability to refresh the route means that the system can adapt by "routing around" link and node failures. A variety of routing protocols have been developed in the literature and several different ones are used in practice. Broadly speaking, protocols fall into one of two categories depending on what they send in the advertisements and how they integrate advertisements to compute the routing table. Protocols in the first category are called vector protocols because each node, n, advertises to its neighbors a vector, with one component per destination, of information that tells the neighbors about n's route to the corresponding destination.

For example, in the simplest form of a vector protocol, n advertises its cost to reach each destination as a vector of destination: cost tuples. In the integration step, each recipient of the advertisement can use the advertised cost from each neighbor, together with some other information (the cost of the link from the node to the neighbor) known to the recipient, to calculate its own cost to the destination. A vector protocol that advertises such costs is also called a distance-vector protocol. Routing protocols in the second category are called link-state protocols. Here, each node advertises information about the link to its current neighbors on all its links, and each recipient re-sends this information on all of its links, flooding the information about the links through the network. Eventually, all nodes know about all the links and nodes in the topology. Then, in the integration step, each node uses an algorithm to compute the minimum-cost path to every destination in the network. We will compare and contrast distance-vector and link-state routing protocols at the end of the next chapter, after we study how they work in detail. For now, keep in mind the following key distinction: in a distance-vector protocol (in fact, in any vector protocol), the route computation is itself distributed, while in a link-state protocol, the route computation process is done independently at each node and the dissemination of the topology of the network is done using distributed flooding. The next two sections discuss the essential details of distance-vector and link-state protocols. In this chapter, we will assume that there are no failures of nodes or links in the network; we will assume that the only changes that can occur in the network are additions of either nodes or links. We will relax this assumption in the next chapter. We will assume that all links in the network are bi-directional and that the costs in each direction are symmetric (i.e., the cost of a link from A to B is the same as the cost of the link from B to A, for any two directly connected nodes A and B).

12.3.1 Network Layer Routing

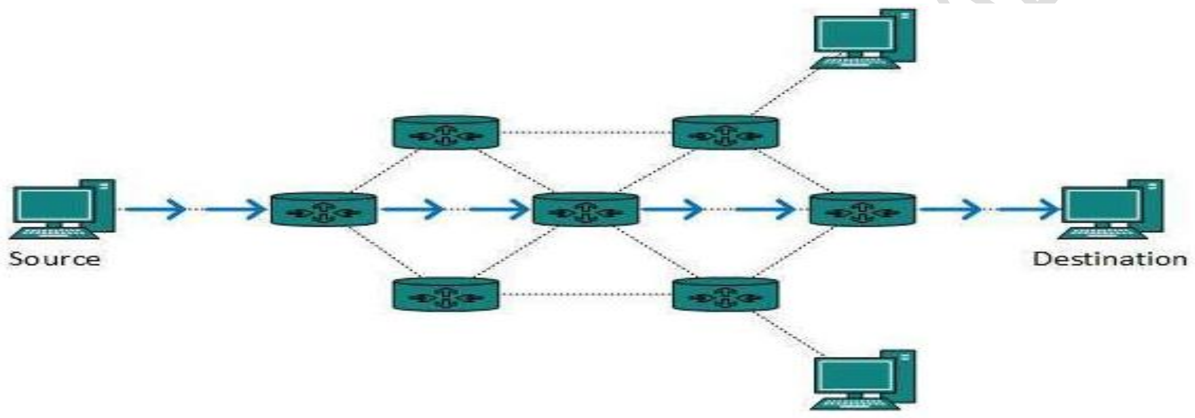
When a device has multiple paths to reach a destination, it always selects one path by preferring it over others. This selection process is termed as Routing. Routing is done by special network devices called routers or it can be done by means of software processes. The software-based routers have limited functionality and limited scope. A router is always configured with some default route. A default route tells the router where to forward a packet if there is no route found for a specific destination. In case there are multiple paths existing to reach the same destination, the router can make a decision based on the following information:

- Hop Count
- Bandwidth
- Metric
- Prefix-length
- Delay

Routes can be statically configured or dynamically learnt. One route can be configured to be preferred over others.

12.3.2 Unicast routing

Most of the traffic on the internet and intranets knew as unicast data or unicast traffic is sent with the specified destination. Routing unicast data over the internet is called unicast routing. It is the simplest form of routing because the destination is already known. Hence the router just has to look up the routing table and forward the packet to next hop.



12.3.3 Broadcast routing

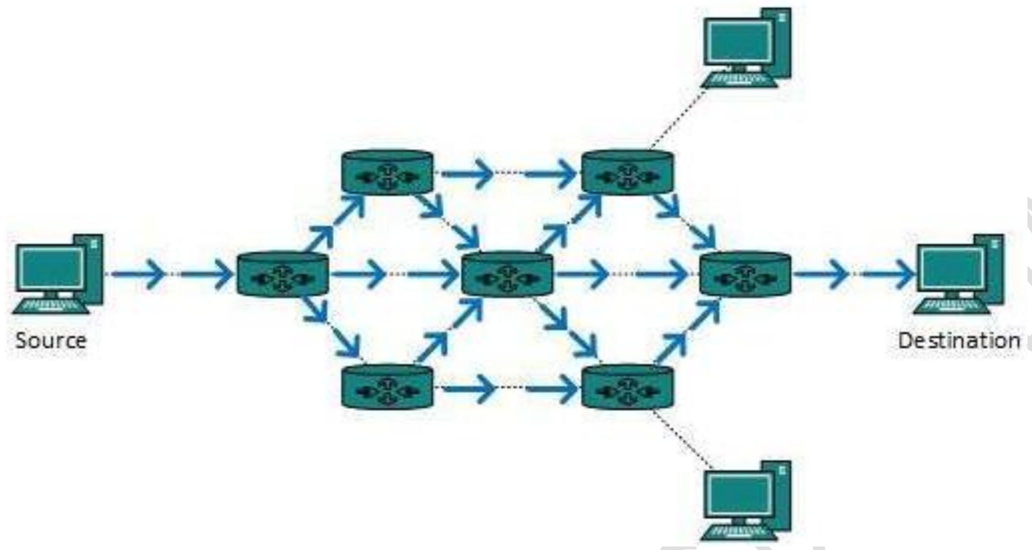
By default, the broadcast packets are not routed and forwarded by the routers on any network. Routers create broadcast domains. But it can be configured to forward broadcasts in some special cases. A broadcast message is destined for all network devices.

12.3.3.1 Broadcast routing can be done in two ways (algorithm):

A router creates a data packet and then sends it to each host one by one. In this case, the router creates multiple copies of the single data packet with different destination addresses. All packets are sent as unicast but because they are sent to all, it simulates as if the router is broadcasting.

This method consumes lots of bandwidth and router must destination address of each node.

Secondly, when the router receives a packet that is to be broadcasted, it simply floods those packets out of all interfaces. All routers are configured in the same way.

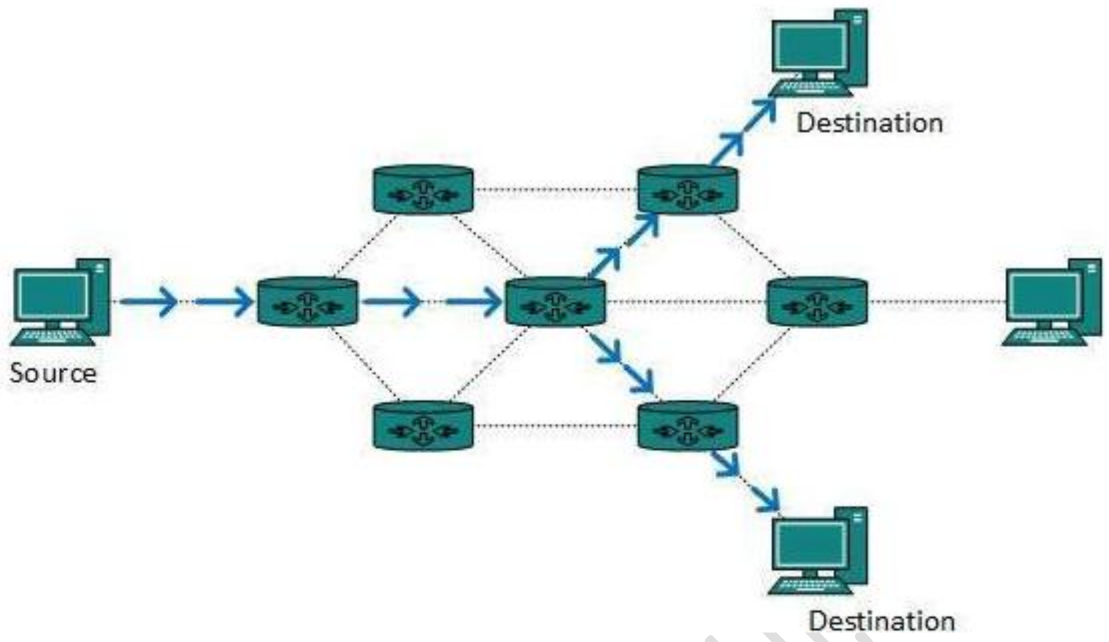


- This method is easy on router's CPU but may cause the problem of duplicate packets received from peer routers.

  Reverse path forwarding is a technique, in which router knows in advance about its predecessor from where it should receive the broadcast. This technique is used to detect and discard duplicates.

12.3.4 Multicast Routing

Multicast routing is a special case of broadcast routing with significance difference and challenges. In broadcast routing, packets are sent to all nodes even if they do not want it. But in Multicast routing, the data is sent to only nodes which want to receive the packets.

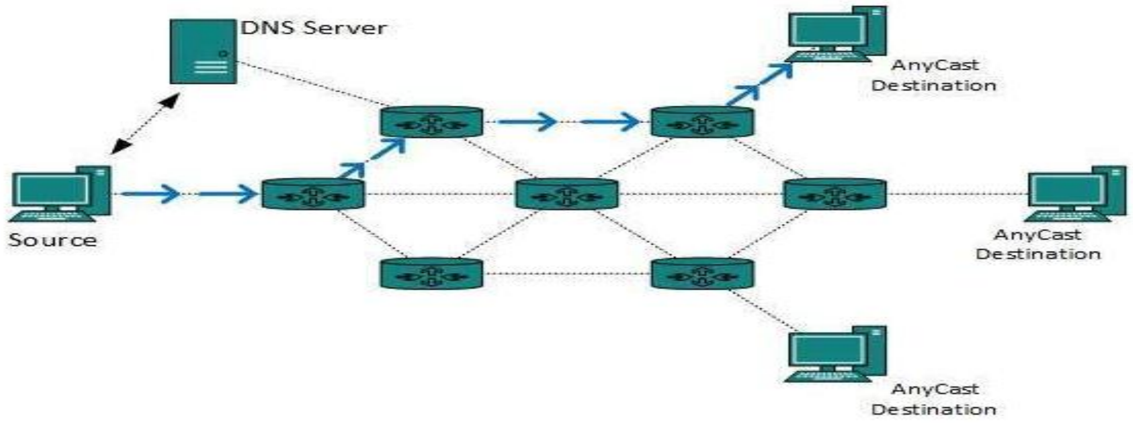


The router must know that there are nodes, which wish to receive multicast packets (or stream) then only it should forward. Multicast routing works spanning tree protocol to avoid looping.

Multicast routing also uses reverse path Forwarding technique, to detect and discard duplicates and loops.

12.3.5 Anycast Routing

Anycast packet forwarding is a mechanism where multiple hosts can have a same logical address. When a packet destined to this logical address is received, it is sent to the host which is nearest in routing topology.



Anycast routing is done with help of DNS server. Whenever an Anycast packet is received it is enquired with DNS to where to send it. DNS provides the IP address which is the nearest IP configured on it.

12.4 Unicast Routing Protocols

There are two kinds of routing protocols available to route unicast packets:

12.4.1 Distance Vector Routing Protocol

Distance Vector is a simple routing protocol which takes routing decision on the number of hops between source and destination. A route with less number of hops is considered as the best route. Every router advertises its set best routes to other routers. Ultimately, all routers build up their network topology based on the advertisements of their peer routers, For example Routing Information Protocol (RIP).

12.4.2 Link State Routing Protocol

Link State protocol is slightly complicated protocol than Distance Vector. It takes into account the states of links of all the routers in a network. This technique helps routes build a common graph of the entire network. All routers then calculate their best path for routing purposes. for example, Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (ISIS).

12.4.3 Multicast Routing Protocols

Unicast routing protocols use graphs while Multicast routing protocols use trees, i.e. spanning tree to avoid loops. The optimal tree is called shortest path spanning tree.

DVMRP - Distance Vector Multicast Routing Protocol

MOSPF - Multicast Open Shortest Path First

CBT - Core Based Tree

PIM - Protocol independent Multicast

Protocol Independent Multicast is commonly used now. It has two flavors:

PIM Dense Mode

This mode uses source-based trees. It is used in a dense environment such as LAN.

PIM Sparse Mode

This mode uses shared trees. It is used in a sparse environment such as WAN.

Routing Algorithms

The routing algorithms are as follows:

12.4.4 Flooding

Flooding is the simplest method packet forwarding. When a packet is received, the routers send it to all the interfaces except the one on which it was received. This creates too much burden on the network and lots of duplicate packets wandering in the network.

Time to Live (TTL) can be used to avoid infinite looping of packets. There exists another approach for flooding, which is called Selective Flooding to reduce the overhead on the network. In this method, the router does not flood out on all the interfaces, but selective ones.

12.4.5 Shortest Path

Routing decision in networks are mostly taken on the basis of cost between source and destination. Hop count plays a major role here. The shortest path is a technique which uses various algorithms to decide a path with a minimum number of hops.

Common shortest path algorithms are:

- Dijkstra's algorithm
- Bellman Ford algorithm
- Floyd Warshall algorithm.

In a real world scenario, networks under the same administration are generally scattered geographically. There may exist requirement of connecting two different networks of the same kind as well as of different kinds. Routing between two networks is called internetworking.

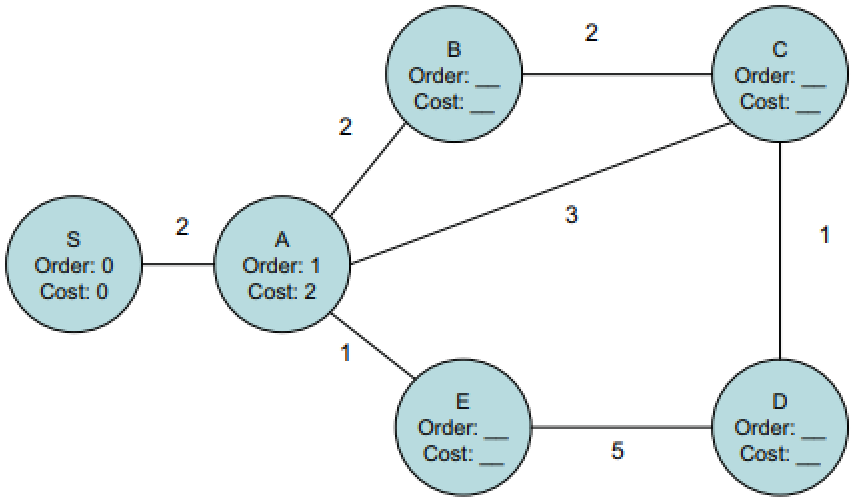
Networks can be considered different based on various parameters such as, Protocol, topology, Layer-2 network and addressing scheme.

In internetworking, routers have knowledge of each other's address and address beyond them. They can be statically configured go on a different network or they can learn by using an internetworking routing protocol.

Routing protocols which are used within an organization or administration are called Interior Gateway Protocols or IGP. RIP, OSPF are examples of IGP. Routing between different organizations or administrations may have Exterior Gateway Protocol, and there is only one EGP i.e. Border Gateway Protocol.

Self-Assessment Question (SAQs) for lecture 11

SAQ 12.1



Consider the network shown above.

The number near each link is its cost. We're interested in finding the shortest paths (taking costs into account) from S to every other node in the network. What is the result of running Dijkstra's shortest path algorithm on this network? To answer this question, near each node, list a pair of numbers: The first element of the pair should be the order, or the iteration of the algorithm in which the node is picked. The second element of each pair should be the shortest path cost from S to that node.

SAQ 12.2

Eaver implements distance vector routing in his network in which the links all have arbitrary positive costs. In addition, there are at least two paths between any two nodes in the network. One node, u, has an erroneous implementation of the integration step: it takes the advertised costs from each neighbor and picks the route corresponding to the minimum advertised cost to each destination as its route to that destination, without adding the link cost to the neighbor. It breaks any ties arbitrarily. All the other nodes are implemented correctly. Let’s use the term “correct route” to mean the route that corresponds to the minimum-cost path. Which of the following statements are true of Eaver’s network?

(a) Only u may have incorrect routes to any other node.

(b) Only u and u’s neighbors may have incorrect routes to any other node.

(c) In some topologies, all nodes may have correct routes.

(d) Even if no HELLO or advertisements packets are lost and no link or node failures occur, a routing loop may occur.

Study Session 13: Reliable Data Transport Protocols

13.1 Introduction

Packets in a best-effort network can be lost for any number of reasons, including queue overflows at switches because of congestion, repeated collisions over shared media, routing failures, and uncorrectable bit errors. In addition, packets can arrive out-of-order at the destination because different packets sent in sequence take different paths or because some switch enroute reorders packets for some reason. They usually experience variable delays, especially whenever they encounter a queue. In some cases, the underlying network may even duplicate packets. Many applications, such as Web page downloads, file transfers, and interactive terminal sessions would like a reliable, in-order stream of data, receiving exactly one copy of each byte in the same order in which it was sent. A reliable transport protocol does the job of hiding the vagaries of a best-effort network packet losses, reordered packets, and duplicate packets—from the application, and provides it the abstraction of a reliable packet stream. We will develop protocols that also provide in-order delivery. A large number of protocols have been developed that various applications use, and there are several ways to provide a reliable, in-order abstraction. This chapter will not discuss them all, but will instead discuss two protocols in some detail. The first protocol, called stop-and-wait, will solve the problem in perhaps the simplest possible way that works, but do so somewhat inefficiently. The second protocol will augment the first one with a sliding window to significantly improve performance. All reliable transport protocols use the same powerful ideas: redundancy to cope with packet losses and receiver buffering to cope with reordering, and most use adaptive timers. The tricky part is figuring out exactly how to apply redundancy in the form of packet retransmissions, in working out exactly when retransmissions should be done, and in achieving good performance. This chapter will study these issues, and discuss ways in which a reliable transport protocol can achieve high throughput.

13.2 The Problem

The problem we're going to solve is relatively easy to state. A sender application wants to send a stream of packets to a receiver application over a best-effort network, which can drop packets arbitrarily, reorder them arbitrarily, delay them arbitrarily, and possibly even duplicate packets. The receiver wants the packets in exactly the same order in which the sender sent them, and wants exactly one copy of each packet. Our goal is to devise mechanisms at the sending and receiving nodes to achieve what the receiver wants. These mechanisms involve rules between the sender and receiver, which constitute the protocol. In addition to correctness, we will be interested in calculating the throughput of our protocols, and in coming up with ways to maximize it. All mechanisms to recover from losses, whether they are caused by packet drops or corrupted bits, employ redundancy. We have already studied error-correcting codes such as linear block codes and convolutional codes to mitigate the effect of bit errors. In principle, one could apply similar coding techniques over packets (rather than over bits) to recover from packet losses (as opposed to bit corruption). We are, however, interested not just in a scheme to reduce the effective packet loss rate, but to eliminate their effects altogether, and recover all lost packets. We are also able to rely on feedback from the receiver that can help the sender determine what to send at any point in time, in order to achieve that goal. Therefore, we will focus on carefully using **retransmissions** to recover from packet losses; one may combine retransmissions and error-correcting codes to produce a protocol that can further improve throughput under certain conditions. In general, experience has shown that if packet losses are not persistent and occur in bursts, and if latencies are not excessively long (i.e., not multiple seconds long), retransmissions by themselves are enough to recover from losses and achieve good throughput. Most practical reliable data transport protocols running over Internet paths today use only retransmissions on packets (individual links usually use the error correction methods, such as the ones we studied earlier, and may also augment them with a limited number of retransmissions to reduce the link-level packet loss rate. We will develop the key ideas for two kinds of reliable data transport protocols: **stop and-wait and sliding window** with a fixed window size. We will use the word "sender" to refer to the sending side of the transport protocol and the word "receiver" to refer to the receiving side. We will use “sender application” and “receiver application” to refer to the processes (applications) that would like to send and receive data in a reliable, in-order manner.

13.3 Stop-and-Wait Protocol

The high-level idea in this protocol is simple. The sender attaches a transport-layer header to every data packet, which includes a unique identifier for the data packet (the transport layer header is distinct from the network-layer packet header that contains the destination address, hop limit, and header checksum). Ideally, this unique identifier will never be reused for two different packets on the same stream. The receiver, upon receiving the data packet with identifier k, will send an acknowledgment (ACK) to the sender; the header of this ACK contains k, so the receiver communicates “I got data packet k” to the sender. Both data packets and ACKs may get lost in the network. In the stop-and-wait protocol, the sender sends the next data packet on the stream if, and only if, it receives an ACK for k. If it does not get an ACK within some period of time, called the timeout, the sender retransmits data packet k.

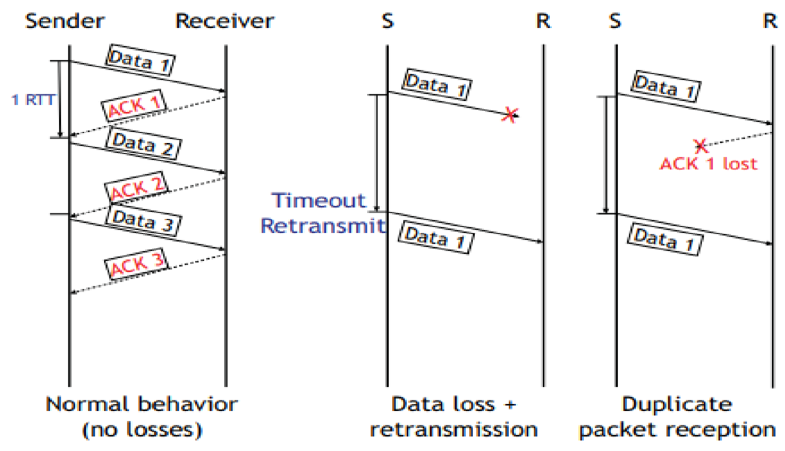


Figure 13-1: The stop-and-wait protocol. Each picture has a sender timeline and a receiver timeline. Time starts at the top of each vertical line and increases moving downward. The picture on the left shows what happens when there are no losses; the middle shows what happens on a data packet loss; and the right shows how duplicate packets may arrive at the receiver because of an ACK loss.

The receiver’s job is to deliver each data packet it receives to the receiver application. Figure 13-1 shows the basic operation of the protocol when packets are not lost (left) and when data packets are lost (right). Three properties of this protocol bear some discussion:

1. how to pick unique identifiers,

2. why this protocol may deliver duplicate data packets to the receiver application, and how the receiver can prevent that from occurring, and

3. how to pick the timeout. We discuss each of these in turn below

13.3.1 Selecting Unique Identifiers Sequence Numbers

The sender may pick any unique identifier for a data packet. In most transport protocols, a convenient and effective choice of unique identifier is to use an incrementing sequence number. The simplest way to achieve this goal is for the sender and receiver to agree on the initial value of the identifier (which for our purposes will be taken to be 1), and then increment the identifier by 1 for each subsequent new data packet sent. Thus, the data packet sent after the ACK for k is received by the sender will have identifier k + 1. These incrementing identifiers are called sequence numbers. In practice, transport protocols like TCP (Transmission Control Protocol), the standard Internet protocol for reliable data delivery, devote considerable effort to picking a good initial sequence number to avoid overlaps with previous instantiations of reliable streams between the same communicating processes. We won't worry about these complications in this chapter, except to note that establishing and properly terminating these streams (aka connections) reliably is a non-trivial problem. TCP also uses a sequence number that identifies the starting byte offset of the packet in the stream, to handle variable packet sizes.

13.3.2 Semantics of this Stop-and-Wait Protocol

It is easy to see that the stop-and-wait protocol achieves reliable data delivery as long as each of the links along the path have a non-zero packet delivery probability. However, it does not achieve exactly once semantics; its semantics are at least once—i.e., each packet will be delivered to the receiver application either once or more than once. One reason is that the network could drop ACKs, as shown in Figure 13-1 (right). A data packet may have reached the receiver, but the ACK doesn't reach the sender, and the sender will then timeout and retransmit the data packet. The receiver will get multiple copies of the data packet, and deliver both to the receiver application. Another reason is that the sender might have timed out, but the original data packet may not actually have been lost. Such a retransmission is called a spurious retransmission, and is a waste of bandwidth. The sender may strive to reduce the number of spurious retransmissions, but it is impossible to eliminate them in general.

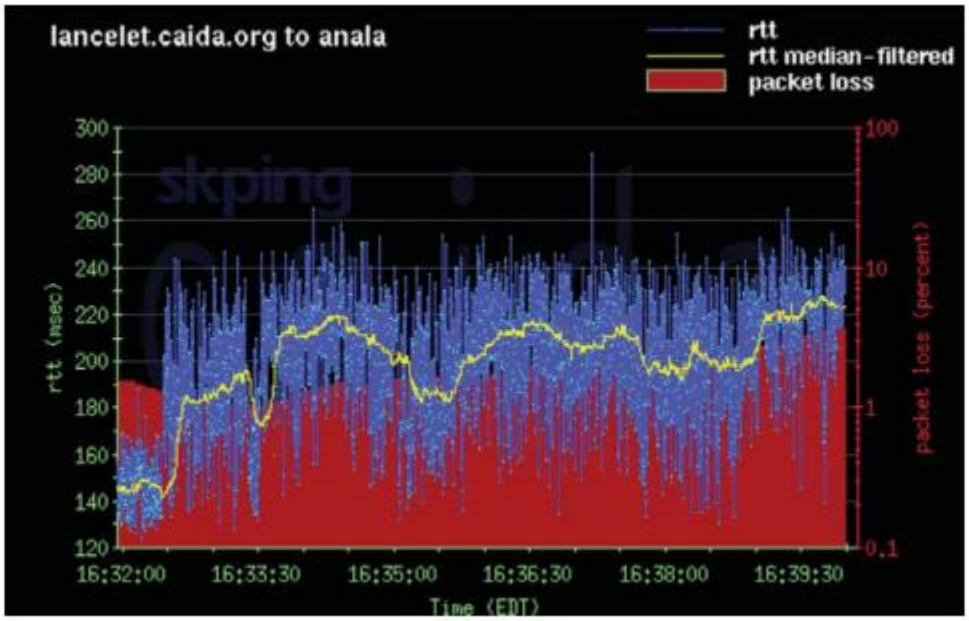
Preventing duplicates: The solution to the problem of duplicate data packets arriving at the receiver is for the receiver to keep track of the last in-sequence data packet it has delivered to the application. At the receiver, let us maintain the sequence number of the last in-sequence data packet in the variable rcv_seqnum. If a data packet with sequence number less than or equal to rcv_seqnum arrives, then the receiver sends an ACK for the packet and discards it. Note that the only way a data packet with sequence number smaller than rcv_seqnum can arrive is if there were reordering in the network and the receiver gets an old data packet; for such packets, the receiver can safely not send an ACK because it knows that the sender knows about the receipt of the packet and has sent subsequent packets. This method prevents duplicate packets from being delivered to the receiving application. If a data packet with sequence number rcv_seqnum + 1 arrives, then the receiver sends an ACK to the sender, delivers the data packet to the application, and increments rcv_seqnum. Note that a data packet with sequence number greater than rcv_seqnum + 1 should never arrive in this stop-and-wait protocol because that would imply that the sender got an ACK for rcv_seqnum + 1, but such an ACK would have been sent only if the receiver got the corresponding data packet. So, if such a data packet were to arrive, then there must be a bug in the implementation of either the sender or the receiver in this stop-and-wait protocol. With this modification, the stop-and-wait protocol guarantees exactly-once delivery to the application.

13.3.3 Setting Timeouts

The final design issue that we need to nail down in our stop-and-wait protocol is setting the value of the timeout. How soon after the transmission of a packet should the sender conclude that the data packet (or the ACK) was lost, and go ahead and retransmit? One approach might be to use some constant, but then the question is what it should be set to. Too small, and the sender may end up retransmitting data packets before giving enough time for the ACK for the original transmission to arrive, wasting network bandwidth. Too large, and one ends up wasting network bandwidth and simply idling before retransmitting. The natural time-scale in the protocol is the time between the transmission of a data packet and the arrival of the ACK for the packet. This time is called the round-trip time, or RTT, and plays a crucial role in all reliable transport protocols. A good value of the timeout must clearly depend on the RTT; it makes no sense to use a timeout that is not bigger than the mean RTT (and in fact, it must be quite a bit bigger than the average, as we'll see). The other reason the RTT is an important concept is that the throughput (in packets per second) achieved by the stop-and-wait protocol is inversely proportional to the RTT. In fact, the throughput of the sliding window protocol also depends on the RTT, as we will see. The next section describes a procedure to estimate the RTT and set sender timeouts. This technique is general and applies to a variety of protocols, including both stop-and-wait and sliding window.

13.4 Adaptive RTT Estimation and Setting Timeouts

The RTT experienced by packets is variable because the delays in a best-effort network are variable. An example is shown in Figure 13-2, which shows the RTT of an Internet path between two hosts (blue) and the packet loss rate (red), both as a function of the time-of-day. The "rtt median-filtered" curve is the median RTT computed over a recent window of samples, and you can see that even that varies quite a bit. Picking a timeout equal to simply the mean or median RTT is not a good idea because there will be many RTT samples that are larger than the mean (or median), and we don't want to timeout prematurely and send spurious retransmissions. A good solution to the problem of picking the timeout value uses two tools we have seen earlier in the course: probability distributions (in our case, of the RTT estimates) and a simple filter design. Suppose we are interested in estimating a good timeout post facto: i.e., suppose we run the protocol and collect a sequence of RTT samples, how would one use these values to pick a good timeout? We can take all the RTT samples and plot them as a probability distribution, and then see how any given timeout value will have performed in terms of the probability of a spurious retransmission. If the timeout value is T, then this probability may be estimated as the area under the curve to the right of "T" in the picture on the left of Figure 13-3, which shows the histogram of RTT samples. Equivalently, if we look at the cumulative distribution function of the RTT samples (the picture on the right of Figure 13-3, the probability of a spurious retransmission may be assumed to be the value of the y-axis corresponding to a value of T on the x-axis.



Courtesy of the Cooperative Association for Internet Data Analysis. Used with permission.

Figure 13-2: RTT variations are pronounced in many networks

Real-world distributions of RTT are not actually Gaussian, but an interesting property of all distributions is that if you pick a threshold that is a sufficient number of standard deviations greater than the mean, the tail probability of a sample exceeding that threshold can be made arbitrarily small. (For the mathematically inclined, a useful result for arbitrary distributions is Chebyshev's inequality, which you might have seen in other courses already (or soon will): $P(|X - \mu| \geq k\sigma) \leq 1/k^2$, where μ is mean and σ the standard deviation of the distribution. For Gaussians, the tail probability falls off much faster than $1/k^2$; for instance, when $k = 2$, the Gaussian tail probability is only about 0.05 and when $k = 3$, the tail probability is about 0.003.) The protocol designer can use past RTT samples to determine an RTT cut-off so that only a small fraction f of the samples are larger. The choice of f depends on what spurious retransmission rate one is willing to tolerate, and depending on the protocol, the cost of such an action might be small or large. Empirically, Internet transport protocols tend to

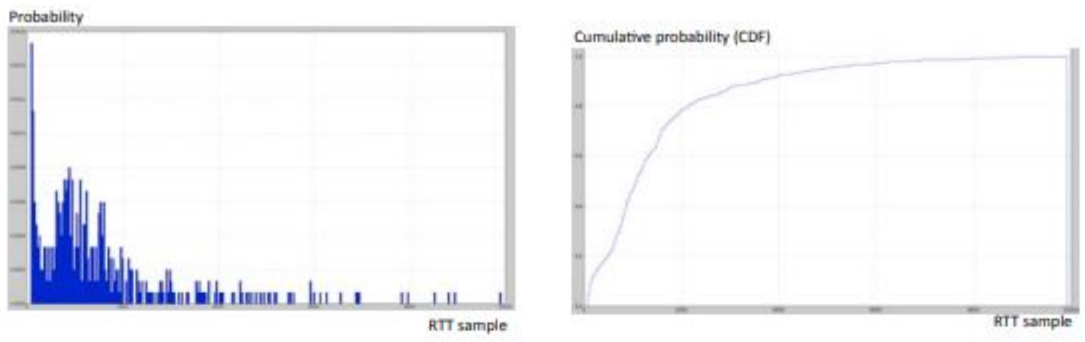


Figure 13-3: RTT variations on a wide-area cellular wireless network (Verizon Wireless's 3G CDMA Rev A service) across both idle periods and when data transfers are in progress, showing extremely high RTT values and high variability. The x-axis in both pictures is the RTT in milliseconds. The picture on the left shows the histogram (each bin plots the total probability of the RTT value falling within that bin), while the picture on the right is the cumulative distribution function (CDF). These delays suggest a poor network design with excessively long queues that do nothing more than cause delays to be very large. Of course, it means that the timeout method must adapt to these variations to the extent possible. (Data collected in November 2009 in Cambridge, MA, and Belmont, MA.)

be conservative and use k = 4, in an attempt to make the likelihood of spurious retransmission very small, because it turns out that the cost of doing one on an already congested network is rather large. Notice that this approach is similar to something we did earlier in the course when we estimated the bit-error rate from the probability density function of voltage samples, where values above (or below) a threshold would correspond to a bit error. In our case, the "error" is *spurious retransmission*. So far, we have discussed how to set the timeout in a post-facto way, assuming we knew what the RTT samples were. We now need to talk about two important issues to complete the story:

1. How can the sender obtain RTT estimates?

2. How should the sender estimate the mean and deviation and pick a suitable timeout?

Obtaining RTT estimates. If the sender keeps track of when it sent each data packet, then it can obtain a sample of the RTT when it gets an ACK for the packet. The RTT sample is simply the difference in time between when the ACK arrived and when the data packet was sent. An elegant way to keep track of this information in a protocol is for the sender to include the current time in the header of each data packet that it sends in a "timestamp" field. The receiver then simply echoes this time in its ACK. When the sender gets an ACK, it just has to consult the clock for the current time and subtract the echoed timestamp to obtain an RTT sample.

13.5 Throughput of Stop-and-Wait

We now show how to calculate the throughput of the stop-and-wait protocol. Clearly, the maximum throughput occurs when there are no packet losses. The sender sends one packet every RTT, so the maximum throughput is exactly that. We can also calculate the throughput of stop-and-wait when the network has a packet loss rate of C. For convenience, we will treat C as the bi-directional loss rate; i.e., the probability of any given packet or its ACK getting lost is C. We will assume that the packet loss distribution is independent and identically distributed. What is the throughput of the stop-and-wait protocol in this case? The answer clearly depends on the timeout that's used. Let's assume that the retransmission timeout is RTO, which we will assume to be a constant for simplicity (i.e., it is the same throughout the connection and the sender doesn't use any exponential back-off). These assumptions mean that the calculation below may be viewed as a (good) upper bound on the throughput. Let T denote the expected time taken to send a data packet and get an ACK for it. Observe that with probability 1 − C, the data packet reaches the receiver and its ACK reaches the sender. On the other hand, with probability C, the sender needs to time out and retransmit a data packet. We can use this property to write an expression for T:

$$T = (1 - C) \cdot RTT + C(RTO + T), \qquad \text{eq. 13.1}$$

because once the sender times out, the expected time to send a data packet and get an ACK is exactly T, the number we want to calculate. Solving Equation (13.1), we find that

$T = RTT + C/1-C.RTO.$

The expected throughput of the protocol is then equal to 1/T packets per second. The good thing about the stop-and-wait protocol is that it is very simple, and should be used under two circumstances: first, when throughput isn't a concern and one wants good reliability, and second, when the network path has a small RTT such that sending one data packet every RTT is enough to saturate the bandwidth of the link or path between sender and receiver. On the other hand, a typical Internet path between Boston and San Francisco might have an RTT of about 100 milliseconds. If the network path has a bit rate of 1 megabit/s, and we use a data packet size of 10,000 bits, then the maximum throughput of stop-and-wait would be only 10% of the possible rate. And in the face of packet loss, it would be much lower than that. The next section describes a protocol that provides considerably higher throughput. It builds on all the mechanisms used in the stop-and-wait protocol.

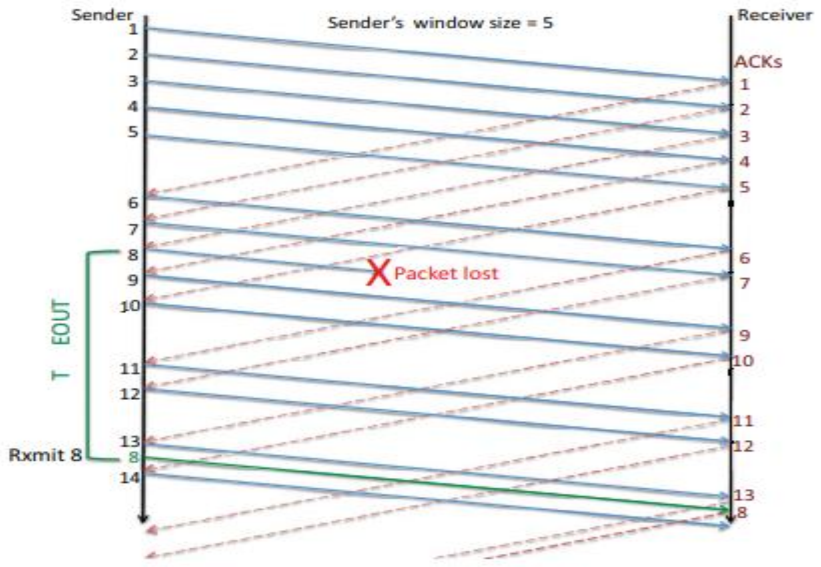


Figure 13-4: The sliding window protocol in action (W = 5 here).

13.6 Sliding Window Protocol

The idea is to use a window of data packets that are outstanding along with the path between sender and receiver. By "outstanding", we mean "unacknowledged". The idea then is to overlap data packet transmissions with ACK receptions. For our purposes, a window size of W data packets means that the sender has at most W outstanding data packets at any time. Our protocol will allow the sender to pick W, and the sender will try to have W outstanding data packets in the network at all times. The receiver is almost exactly the same as in the stop-and-wait case, except that it must also buffer data packets that might arrive out-of-order so that it can deliver them in order to the receiving application. This enhancement makes the receiver more complicated than before, but this complexity is worth the improvement in throughput in most situations. The key idea in the protocol is that the window slides every time the sender gets an ACK. The reason is that the receipt of an ACK is a positive signal that one data packet left the network, and so the sender can add another to replenish the window. This plan is shown in Figure 13-4 that shows a sender (top line) with W = 5 and the receiver (bottom line) sending ACKs (dotted arrows) whenever it gets a data packet (solid arrow). Time moves from left to right here. There are at least two different ways of defining a window in a reliable transport protocol. Here, we will use the following:

A window size of W means that the maximum number of outstanding (unacknowledged) data packets between sender and receiver is W.

When there are no packet losses, the operation of the sliding window protocol is fairly straightforward. The sender transmits the next in-sequence data packet every time an ACK arrives; if the ACK is for data packet k and the window is W, the data packet sent out has sequence number k + W. The receiver ACKs each data packet echoing the sender's timestamp and delivers packets in sequence number order to the receiving application. The sender uses the ACKs to estimate the smoothed RTT and linear deviations and sets a timeout. Of course, the timeout will only be used if an ACK doesn't arrive for a data packet within that duration. We now consider what happens when a packet is lost. Suppose the receiver has received data packets 0 through k − 1 and the sender doesn't get an ACK for data packet k. If the subsequent data packets in the window reach the receiver, then each of those packets triggers an ACK. So the sender will have the following ACKs assuming no further packets are lost: k + 1, k + 2,...,k + W − 1. Moreover, upon the receipt of each of these ACKs, an additional new data packet will get sent with an even higher sequence number. But somewhere in the midst of these new data packet transmissions, the sender's timeout for data packet k will occur, and the sender will retransmit that packet. If that data packet reaches, then it will trigger an ACK, and if that ACK reaches the sender, yet another new data packet with a new sequence number one larger than the last sent so far will be sent. Hence, this protocol tries hard to keep as many data packets outstanding as possible, but not exceeding the window size, W. If C data packets or ACKs get lost, then the effective number of outstanding data packets reduces to W − C, until one of them times out, is retransmitted and received successfully by the receiver, and its ACK received successfully at the sender. We will use a fixed size window in our discussion in this chapter. The sender picks a maximum window size and does not change that during a stream. In practice, most practical transport protocols on the Internet should implement a congestion control strategy to adjust the window size to prevailing network conditions (level of congestion, the rate of data delivery, packet loss rates, round-trip times, etc.)

13.6.1 Sliding Window Sender

We now describe the salient features of the sender side of this fixed-size sliding window protocol. The sender maintains un_acked pkts, a buffer of unacknowledged data packets. Every time the sender is called (by a fine-grained timer, which we assume fires each slot), it first checks to see whether any data packets were sent greater than "timeout" seconds ago (assuming time is maintained in seconds). If so, the sender retransmits each of these data packets and takes care to change the packet transmission time of each of these packets to be the current time. For convenience, we usually maintain the time at which each packet was last sent in the packet data structure, though other ways of keeping track of this information are also possible.

After checking for retransmissions, the sender proceeds to see whether any new data packets can be sent. To properly check if any new packets can be sent, the sender maintains a variable, outstanding, which keeps track of the current number of outstanding data packets. If this value is smaller than the maximum window size, the sender sends a new data packet, setting the sequence number to be max seq + 1, where max seq is the highest sequence number sent so far. Of course, we should remember to update max seq as well, and increment outstanding by. Whenever the sender gets an ACK, it should remove the acknowledged data packet from un_acked pkts (assuming it hasn't already been removed), decrement outstanding, and call the procedure to calculate the timeout (which will use the timestamp echoed in the current ACK to update the EWMA filters and update the timeout value). We would like outstanding to keep track of the number of unacknowledged data packets between sender and receiver. We have described the method to do this task as follows:

increment it by 1 on each new data packet transmission, and decrement it by 1 on each ACK that was not previously seen by the sender, corresponding to a packet the sender had previously sent that is being acknowledged (as far as the sender is concerned) for the first time. The question now is whether outstanding should be adjusted when retransmission is done. A little thought will show that it should not be. The reason is that it is precisely on a timeout of a data packet that the sender believes that the packet was actually lost, and in the sender's view, the packet has left the network. But the retransmission immediately adds a data packet to the network, so the effect is that the number of outstanding packets is exactly the same. Hence, no change is required in the code. Implementing a sliding window protocol is sometimes error-prone even when one completely understands the protocol in one's mind. Three kinds of errors are common. First, the timeouts are set too low because of an error in the EWMA estimators, and data packets end up being retransmitted too early, leading to spurious retransmissions. In addition to keeping track of the sender's smoothed round-trip time (srtt), RTT deviation, and timeout estimates, it is a good idea to maintain a counter for the number of retransmissions done for each data packet. If the network has a certain total loss rate between sender and receiver and back (i.e., the bi-directional loss rate), pl, the number of retransmissions should 1 be on the order of 1−pl − 1, assuming that each packet is lost independently and with the same probability. (It is a useful exercise to work out why this formula holds.) If your implementation shows a much larger number than this prediction, it is very likely that there's a bug in it. Second, the number of outstanding data packets might be larger than the configured window, which is an error. If that occurs, and especially if a bug causes the number of outstanding packets to grow unbounded, delays will increase and it is also possible that packet loss rates caused by congestion will increase. It is useful to place an assertion or two that checks that the outstanding number of data packets does not exceed the configured window. Third, when retransmitting a data packet, the sender must take care to modify the time at which the packet is sent. Otherwise, that packet will end up getting retransmitted repeatedly, a pretty serious bug that will cause the throughput to diminish.

13.6.2 Sliding Window Receiver

At the receiver, the biggest change to the stop-and-wait case is to maintain a list of received data packets that are out-of-order. Call this list rcv_buf. Each data packet that arrives is added to this list, assuming it is not already on the list. It's convenient to store this list in increasing sequence order. Then, check to see whether one or more contiguous data packets starting from rcv_seqnum + 1 are in rcv_buf. If they are, deliver them to the application, remove them from rcv_buf, and remember to update rcv seq_num.

13.6.3 Throughput

What is the throughput of the sliding window protocol we just developed? Clearly, we send W data packets per RTT when there are no data packet or ACK losses, so the throughput in the absence of losses is W/RTT packets per second. So the question one should ask is, what should we set W to in order to maximize throughput, at least when there are no data packet or ACK losses? After answering this question, we will provide a simple formula for the throughput of the protocol in the absence of losses, and then finally consider acket losses.

Setting W

One can address the question of how to choose W using Little's law. Think of the entire bi-directional path between the sender and receiver as a single queue (in reality it's more complicated than a single queue, but the abstraction of a single queue still holds). W is the number of (unacknowledged) packets in the system and RT T is the mean delay between the transmission of a data packet and the receipt of its ACK at the sender (upon which the sender transmits a new data packet). We would like to maximize the processing rate of this system. Note that this rate cannot exceed the bit rate of the slowest, or bottleneck, the link between the sender and receiver (i.e., the rate of the bottleneck link). If that rate is B packets per second, then by Little's law, setting W = B × RTT will ensure that the protocol comes close to achieving a throughput equal to the available bit rate. But what should the RTT be in the above formula? After all, the definition of a "RTT sample" is the time that elapses between the transmission of a data packet and the receipt of an ACK for it. As such, it depends on other data using the path. Moreover, if one looks at the formula B = W/ RTT, it suggests that one can simply increase the window size W to any value and B may correspondingly just increase. Clearly, that can't be right. Consider the simple case when there is only one connection active over a network path. Observe that the RTT experienced by a packet P sent on the connection may be broken into two parts: one part that does not depend on any queueing delay (i.e., the sum of the propagation, transmission, and processing delays of the packet and its ACK), and one part that depends on how many other packets were ahead of P in the bottleneck queue. (Here we are assuming that ACKs experience no queueing, for simplicity.) Denote the RTT in the absence of queuing as RTT_{min}, the minimum possible round-trip time that the connection can experience. Now, suppose the RTT of the connection is equal to RTT_{min}. That is, there is no queue building up at the bottleneck link. Then, the throughput of the connection is $W/RTT = W/RTT_{min}$. We would like this throughput to be the bottleneck link rate, B. Setting $W/RTT_{min} = B$, we find that W should be equal to $B \cdot RTT_{min}$.

This quantity $B \cdot RTT_{min}$ is an important concept for sliding window protocols (all sliding window protocols, not just the one we have studied). It is called the bandwidth-delay product of the connection and is a property of the bi-directional network path between the sender and receiver. When the window size is strictly smaller than the bandwidth delay product, the throughput will be strictly smaller than the bottleneck rate, B, and the queueing delay will be non-existent. In this phase, the connection's throughput linearly increases as we increase the window size, W, assuming no other traffic intervenes. The smallest window size for which the throughput will be equal to B is the bandwidth-delay product.

This discussion shows that for our sliding window protocol, setting W = B × RTTmin achieves the maximum possible throughput, B, in the absence of any data packet or ACK losses. When packet losses occur, the window size will need to be higher to get maximum throughput (utilization), because we need a sufficient number of unacknowledged data packets to keep a B × RTTmin worth of packets even when losses occur. A smaller window size will achieve sub-optimal throughput, linear in the window size, and inversely proportional to RTTmin. But once W exceeds B × RTTmin, the RTT experienced by the connection includes queueing as well, and the RTT will no longer be a constant independent of W! That is, increasing W will cause RTT to also increase, but the rate, B, will no longer increase.

What is the throughput in this case?

We can answer this question by applying Little’s law twice. Once at the bottleneck link’s queue, and once on the entire network path. We will show the intuitive result that if W >B × RTTmin, then the throughput is B packets per second. First, let the average number of packets at the queue of the bottleneck link be Q. By Little’s law applied to this queue, we know that Q = B · τ, where B is the rate at which the queue drains (i.e., the bottleneck link rate), and τ is the average delay in the queue, so τ = Q/B.

We also know that

RTT = RTTmin + τ = RTTmin + Q/B. eq.13.2

Now, consider the window size, W, which is the number of unacknowledged packets. We know that all these packets, by conservation of packets, must either be in the bottleneck queue, or in the non-queueing part of the system. That is,

W = Q + B · RTTmin. eq. 19.3

Finally, from Little’s law applied to the entire bi-directional network path, W Throughput = (13.3) RTT

B · RTTmin + Q = RTTmin + (Q/B) = B eq. 13.4

Thus, we can conclude that, in the absence of any data packet or ACK losses, the connection’s throughput is as shown schematically in Figure 13-5.

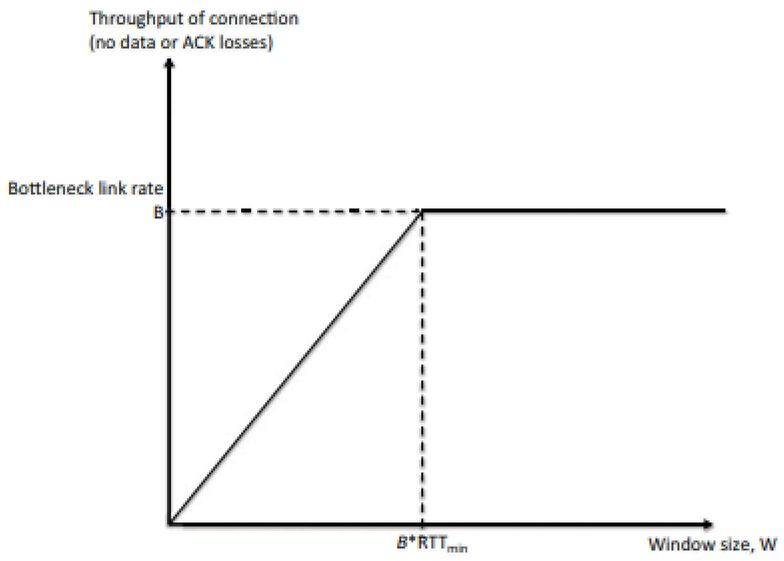


***Figure 13-5: Throughput of the sliding window protocol as a function of the window size in a network with no other traffic. The bottleneck link rate is B packets per second and the RTT without any queueing is RTTmin. The product of these two quantities is the bandwidth-delay product*.**

Throughput of the sliding window protocol with packet losses Assuming that one sets the window size properly, i.e., to be large enough so that W ≥ B × RTTmin always, even in the presence of data or ACK losses, what is the maximum throughput of our sliding window protocol if the network has a certain probability of packet loss?

Consider a simple model in which the network path loses any packet—data or ACK— such that the probability of either a data packet being lost or its ACK being lost is equal to C, and the packet loss random process is independent and identically distributed (the same model as in our analysis of stop-and-wait). Then, the utilization achieved by our sliding window reliable transport protocol is at most 1 − C. Moreover, for large-enough window size, W, our sliding window protocol comes close to achieving it. The reason for the upper bound on utilization is that in this protocol, a data packet is acknowledged only when the sender gets an ACK explicitly for that packet. Now consider the number of transmissions that any given data packet must incur before its ACK is received by the sender. With probability 1 − C, we need one transmission, with probability C(1 − C), we need two transmissions, and so on, giving us an expected number of transmissions of. If we make this number of transmissions, one data packet is successfully sent 1−£ 1 and acknowledged. Hence, the utilization of the protocol can be at most 1 = 1 − C. In 1−£ fact, it turns out the 1 − C is the capacity (i.e., upper-bound on throughput) for any channel (network path) with packet loss rate C. If the sender picks a window size sufficiently larger than the bandwidth-$_{\text{minimum}}$RTT product, so that at least bandwidth-minimum-RTT packets are in transit (unacknowledged) even in the face of data and ACK losses, then the protocol's utilization will be close to the maximum value of 1 − C.

Is a good timeout important for the sliding window protocol?

Given that our sliding window protocol always sends a data packet every time the sender gets an ACK, one might reasonably ask whether setting a good timeout value, which under even the best of conditions involves a hard trade-off, is essential. The answer turns out to be subtle: it's true that the timeout can be quite large because data packets will continue to flow as long as some ACKs are arriving. However, as data packets (or ACKs) get lost, the effective window size keeps falling, and eventually, the protocol will stall until the sender retransmits. So, one can't ignore the task of picking a timeout altogether, but one can pick a more conservative (longer) timeout than in the stop-and-wait protocol. However, the longer the timeout, the bigger the stalls experienced by the receiver application even though the receiver's transport protocol would have received the data packets, they can't be delivered to the application because it wants the data to be delivered in order. Therefore, a good timeout is still quite useful, and the principles discussed in setting it are widely useful.

Secondly, we note that the longer the timeout, the bigger the receiver's buffer has to be when there are losses; in fact, in the worst case, there is no bound on how big the receiver's buffer can get. To see why think about what happens if we were unlucky and a data packet with a particular sequence number kept getting lost, but everything else got through. The two actors mentioned above affect the throughput of the transport protocol, but the biggest consequence of a long timeout is the effect on the latency perceived by applications (and users). The reason is that data packets are delivered in-order by the protocol to the application, which means that a missing packet with sequence number k will cause the application to stall, even though data packets with sequence numbers larger than k have arrived and are in the transport protocol's receiver buffer. Hence, an excessively long timeout hurts interactivity and degrades the user's experience.

Summary of Study 13

This chapter described the key concepts in the design on a reliable data transport protocol. The big idea is to use redundancy in the form of careful retransmissions, for which we developed the idea of using sequence numbers to uniquely identify data packets and acknowledgments for the receiver to signal the successful reception of a data packet to the sender. We discussed how the sender can set a good timeout, balancing between the ability to track a persistent change of the round-trip times against the ability to ignore nonpersistent glitches. The method to calculate the timeout involved estimating a smoothed mean and linear deviation using an exponential weighted moving average, which is a single real-zero low-pass filter. The timeout itself is set at the mean + 4 times the deviation to ensure that the tail probability of spurious retransmission is small. We used these ideas in developing the simple stop-and-wait protocol.

We then developed the idea of a sliding window to improve performance and showed how to modify the sender and receiver to use this concept. Both the sender and receiver are now more complicated than in the stop-and-wait protocol, but when there are no losses, one can set the window size to the bandwidth-delay product and achieve high throughput in this protocol. We also studied how increasing the window size increases the throughput linearly up to a point, after only the (queueing) delay increases, and not the throughput of the connection.

Self-Assessment Question (SAQs) for lecture 13

SAQ 13.1

1. Consider a best-effort network with variable delays and losses. In such a network, Louis suggests that the receiver does not need to send the sequence number in the ACK in a correctly implemented stop-and-wait protocol, where the sender sends data packet k + 1 only after the ACK for data packet k is received. Explain whether he is correct or not.

SAQ 13.2

The 802.11 (Wi-Fi) link-layer uses a stop-and-wait protocol to improve link reliability. The protocol works as follows:

(a) The sender transmits data packet k + 1 to the receiver as soon as it receives an ACK for the data packet k.

(b) After the receiver gets the entire data packet, it computes a checksum (CRC). The processing time to compute the CRC is T_p and you may assume that it does not depend on the packet size.

(c) If the CRC is correct, the receiver sends a link-layer ACK to the sender. The ACK has negligible size and reaches the sender instantaneously. The sender and receiver are near each other, so you can ignore the propagation delay. The bit rate is R = 54 Megabits/s, the smallest data packet size is 540 bits, and the largest data packet size is 5,400 bits.

What is the maximum processing time T_p that ensures that the protocol will achieve a throughput of at least 50% of the bit rate of the link in the absence of data packet and ACK losses, for any data packet size?

SAQ 13.3

Suppose the sender in a reliable transport protocol uses an EWMA filter to estimate the smoothed round trip time, srtt, every time it gets an ACK with an RTT sample r. $srtt \rightarrow \alpha \cdot r + (1 - \alpha) \cdot srtt$ We would like every data packet in a window to contribute a weight of at least 1% to the srtt calculation. As the window size increases, should α increase, decrease, or remain the same, to achieve this goal? (You should be able to answer this question without writing any equations.)