

## Lecture 1: Introduction

August 27, 2007

Lecturer: Atri Rudra

Scribe: Atri Rudra

## 1 Overview

Error-correcting codes (or just codes) are clever ways of representing data so that one can recover the original information even if parts of it are corrupted. The basic idea is to judiciously introduce redundancy so that the original information can be recovered even when parts of the (redundant) data have been corrupted.

Perhaps the most natural and common application of error correcting codes is for communication. For example, when packets are transmitted over the Internet, some of the packets get corrupted or dropped. To deal with this, multiple layers of the TCP/IP stack use a form of error correction called CRC Checksum [4]. Codes are used when transmitting data over the telephone line or via cell phones. They are also used in deep space communication and in satellite broadcast (for example, TV signals are transmitted via satellite). Codes also have applications in areas not directly related to communication. For example, codes are used heavily in data storage. CDs and DVDs work fine even in presence of scratches precisely because they use codes. Codes are used in Redundant Array of Inexpensive Disks (RAID) [3] and error correcting memory [2]. Codes are also deployed in other applications such as paper bar codes, for example, the bar code used by UPS called MaxiCode [1].

In this course, we will mainly think of codes in the communication scenario. In this framework, there is a sender who wants to send (say)  $k$  message symbols over a noisy channel. The sender first *encodes* the  $k$  message symbols into  $n$  symbols (called a *codeword*) and then sends it over the *channel*. The receiver gets a *received word* consisting of  $n$  symbols. The receiver then tries to *decode* and recover the original  $k$  message symbols.

The fundamental tradeoff that will occupy our attention for almost all of the course is that between the amount of redundancy used and the number of errors that can be corrected by a code. We would like to correct as many errors as possible while using the minimum amount of redundancy. Intuitively, these are contradictory goals: a code with higher redundancy should be able to tolerate more number of errors. In the next few weeks, we will see a formalization of this notion.

## 2 Some definitions and codes

We begin with the definition of a code.

**Definition 2.1 (Code).** A code of block length  $n$  over an alphabet  $\Sigma$  is a subset of  $\Sigma^n$ .

We will also frequently use the following alternate way of looking at a code. Given a code  $C \subseteq \Sigma^n$ , with  $|C| = M$ , we will think of  $C$  as a mapping of the following form:

$$C : [M] \rightarrow \Sigma^n,$$

where  $[x]$  for any integer  $x \geq 1$  denotes the set  $\{1, 2, \dots, x\}$ .

We will also need the notion of *dimension* of a code.

**Definition 2.2 (Dimension of a code).** Given a code  $C \subset \Sigma^n$ , its dimension is given by

$$k \stackrel{\text{def}}{=} \log_{|\Sigma|} |C|.$$

Let us begin by looking at two specific codes. Both codes are defined over  $\Sigma = \{0, 1\}$  (also known as *binary codes*). In both cases  $|C| = 2^4$  and we will think of each of the 16 messages as a 4 bit vector.

We first look at the so called *parity code*, which we will denote by  $C_{\oplus}$ . Given a message  $(x_1, x_2, x_3, x_4) \in \{0, 1\}^4$ , its corresponding codeword is given by

$$C_{\oplus}(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4, x_1 \oplus x_2 \oplus x_3 \oplus x_4),$$

where the  $\oplus$  denotes the EXOR operator. In other words, the parity code appends the parity of the message bits at the end of the message.

The second code we will look at is the so called *repetition code*. This is a very natural code (and perhaps the first code one might think of). The idea is to repeat every message bit a fixed number of times. For our example say we repeat each of the 4 message bits 3 times and let us use  $C_{3,rep}$  to denote such a code.

Let us now try to look at the tradeoff between the amount of redundancy and the number of errors each of these codes can correct. Even before we begin to answer the question, we need to define how we are going to measure the amount of redundancy. One natural way to define redundancy for a code with dimension  $k$  and block length  $n$  is by their difference  $n - k$ . By this definition, the parity code uses the least amount of redundancy. However, one “pitfall” of such a definition is that it does not distinguish between a code with  $k = 100$  and  $n = 102$  and another code with dimension and block length 2 and 4 respectively. Intuitively the latter code is using more redundancy. This motivates the following notion of measuring redundancy.

**Definition 2.3 (Rate of a code).** The rate of a code with dimension  $k$  and block length  $n$  is given by

$$R \stackrel{\text{def}}{=} \frac{k}{n}.$$

Note that higher the rate, lesser the amount of redundancy in the code. Also note that as  $k \leq n$ ,  $R \leq 1$ . Given the above definition,  $C_{\oplus}$  and  $C_{3,rep}$  have rates of  $\frac{4}{5}$  and  $\frac{1}{3}$ . As was to be expected, the parity code has a higher rate than the repetition code.

In the next lecture, we will look at how many errors these codes can correct. This will lead to the definition of some key concepts. We will also look at a more sophisticated code called the *Hamming code*.

## References

- [1] Donald G. Chandler, Eric P. Batterman, and Govind Shah. Hexagonal, information encoding article, process and system. *US Patent Number 4,874,936*, October 1989.
- [2] C. L. Chen and M. Y. Hsiao. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM Journal of Research and Development*, 28(2):124–134, 1984.
- [3] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, 1994.
- [4] Larry L. Peterson and Bruce S. Davis. *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, San Francisco, 1996.