COSC 264
Data Communications and Networking
Assignment

Andreas Willig
Dept. of Computer Science and Software Engineering
University of Canterbury
email: andreas.willig@canterbury.ac.nz

July 31, 2014

---

1 Administrivia

This assignment is part of the COSC 264 assessment process. It is worth 30% of the final marks. It is a programming project in which you combine properties of error-correcting codes with retransmission protocols and determine the efficiency of different protocol configurations. You will work in groups of two persons, it is not an option to work alone or in larger groups, unless you have **very** convincing reasons (mere convenience on your side won't convince me) and have obtained **explicit approval** by me (normally by e-mail). Submissions by individuals or larger groups without such approval will not be marked.

You can use Python, Java, Matlab or C. Your program should be a text-mode program, no extra marks will be given for graphical interfaces.

It is quite likely that the problem description below leaves a lot of things unclear to you. Please do not hesitate to use the "Question and Answer Forum" on the Learn platform for raising and discussing any unclear issues.

2 Problem Description

2.1 BSC Channel

We are given one sender, one receiver and a channel in between them. The channel is a so-called **binary symmetric channel** (BSC): in such a channel each transmitted bit is erroneous independently of all other bits, and each bit is erroneous with a fixed probability $p \in (0, 1)$ – this probability is called the

bit error probability or **bit error rate**. We are only interested in channel bit errors and do not care for the delay and capacity properties of the channel.

The transmitter has an infinite supply of data that should be transmitted to the receiver. It subdivides its data into **packets**. More precisely, one packet consists of $o = 100$ bits of overhead (header, checksum) and u bits of user data, summing up to a total packet length of $k = o + u$ bits. The overhead includes a packet checksum which for simplicity we assume to be perfect, i.e. any error introduced by the channel into the k bits of the packet is reliably discovered. After formatting the packet, the transmitter **encodes** the packet using an error-correcting code (forward error correction, FEC) that adds $n - k \geq 0$ redundant bits to the k bits of the packet. We do not consider any particular code. Instead, we assume that for our chosen values of n and k a block code exists which achieves the so-called **Hamming bound**: such a code can correct up to t bit errors in an encoded packet of n bits length, provided the relationship

$$2^{n-k} \geq \sum_{i=0}^{t} \binom{n}{i}$$

is fulfilled (where $\binom{n}{i}$ is the binomial coefficient). For given n and k we then clearly are interested in finding the maximum t satisfying the above relation, and use this as the error-correction capability of our code. Let $t^* = t^*(n, k)$ be the maximum number of wrong bits that can be corrected by the code for the given values of n and k. We then make the simplifying assumption that any t^* or fewer bit errors in the n-bit encoded packet can be reliably corrected, whereas any number of bit errors larger than t^* can never be corrected by the FEC scheme and the packet checksum check at the receiver will fail. (Can you identify a case where this assumption is overly conservative? Consider for example systematic codes.)

A simple protocol is running between transmitter and receiver to ensure reliable data delivery. It operates as follows, starting with a new packet to transmit:

- The sender fetches a new chunk of u user data bits, formats and encodes the packet as above and then transmits it for the first time to the receiver.
- The channel introduces a random number s of bit errors into the packet, following the BSC model.
- The receiver first applies the decoding algorithm and subsequently checks whether the checksum of the decoded packet is correct. If it is correct, the receiver sends a **positive acknowledgement** (ACK) back to the transmitter, otherwise the receiver sends back a **negative acknowledgement** (NACK). We assume that ACK or NACK messages are always reliably received by the transmitter.
- When the transmitter receives a NACK, it re-transmits the same packet again. If it receives an ACK, it stops retransmissions, throws away the

old packet and prepares a new packet for transmission, i.e. the protocol starts over. Note that by this rule there are no bounds on the number of retransmissions the transmitter performs.

You should develop a small simulation tool for this protocol. This tool should allow users to specify values for the user data size $u > 0$, the number $n-k \geq 0$ of redundant bits (note that 0 is a valid value!), and the bit error probability p. For each such parameter combination your program should simulate $N = 1,000,000$ new packets sent by the transmitter, where each new packet might require zero or more retransmissions as described above. For each of these packets you should record its **efficiency value**: if the transmitter needs x_i trials for the i-th packet, it consequently transmits $x_i \cdot n$ bits until this packet is successfully received. The efficiency value for the i-th packet is defined as:

$$\eta_i = \frac{u}{x_i \cdot n}$$

At the end, your simulation should output the average efficiency, calculated as :

$$\eta = \frac{1}{N} \sum_{i=1}^{N} \eta_i$$

In order to simulate the channel it is sufficient to generate a random value for the number s of bit errors in the n-bit packet (of course you need to do this for each packet transmission trial) and compare this against the threshold value t^* (which you need to determine at the start of your simulation, after you have read the parameter values for u and $n-k$). You need to work out which probability distribution s has and you need to identify how in your chosen programming language you can generate random numbers that have the right probability distribution.

2.2 A Few Hints

- There is absolutely no need to generate / prepare actual packets, nor does it really matter how the header is structured.
- Suppose you are given a packet of n bits in total and have a bit error rate of p. You have to figure out the discrete random distribution for the number of bit errors in the n-bit packet when the bit-error rate is p. Let us just for the sake of example assume this is a geometric distribution (which it is not!), which depends on a parameter. Once you have worked out the right parameter value you need to check your preferred computer language for a facility to generate random numbers with geometric distribution. To give this random number generator a figurative name, let us call it the oracle. In your simulation then, for one packet of n bits length you should do the following:
  - Ask the oracle to give you a number s, and interpret this number as the number of bit errors that the current packet has.

  - Compare s against the threshold t^*, which you need to have determined at the start of your program:
    * If s is smaller than or equal to t^*, the packet is correctly transmitted, otherwise it is assumed to be wrong. We call this the outcome.
    * Update your statistics for the efficiency according to the outcome.

- For properly evaluating t^* (especially the binomial coefficients) you will need to carry out calculations with arbitrary-precision integers. Pythons NumPy package provides such a facility. Furthermore, you can speed up computations of binomial coefficients by using a technique called **memoization**.

- To produce graphs, the tool `gnuplot` can be useful. Its main advantage is that it allows for script-based (i.e. non-interactive) creation of graphs, but admittedly its command syntax needs some getting used to. However, you are free to use any tool you like (including Excel, Matlab, etc.) for producing graphs. However, **under all circumstances you need to make sure that axes and curves are properly labeled**. You will loose marks otherwise.

2.3 Two-State Channel

The second scenario to be considered is very similar to the first scenario and you will be able to re-use most of your code. The major change concerns the channel model, which will now become "stateful".

Suppose the channel itself can be in one of two states, which we figuratively call "good" and "bad". When the channel is in the good state, it acts like a BSC with bit error rate p_g, when it is in the bad state it acts like a BSC with bit error rate $p_b \gg p_g$. The channel switches its state in a random fashion. More precisely:

- We consider time to be split up into contiguous time slots, which have the same size. A new time slot starts with each new transmission of the source node, independent of whether the transmission is an initial transmission of a packet or a re-transmission.

- The channel state for a slot is decided at the start of the time slot, immediately before this slots packet transmission commences. When the chosen state for a slot is the good state, then packet transmission in this slot will use bit error rate p_g, otherwise bit error rate p_b is used.

- The channel state s_n that the system enters at beginning of time slot n is determined based on the previous state s_{n-1} and a random experiment as follows:
  - Draw a fresh uniform random number q between 0 and 1.

  - If s_{n-1} == good then:
    - If $q < p_{g,g}$ then $s_n :=$ good
    - If $q \geq p_{g,g}$ then $s_n :=$ bad

    where $p_{g,g} \in [0, 1]$ is a parameter.
  - If s_{n-1} == bad then:
    - If $q < p_{b,b}$ then $s_n :=$ bad
    - If $q \geq p_{b,b}$ then $s_n :=$ good

    where $p_{b,b} \in [0, 1]$ is a parameter.

The initial state s_0 can be defined arbitrarily. The average time that the channel stays in state good or bad is determined by the two parameters $p_{g,g}$ and $p_{b,b}$, respectively. For example, with $p_{g,g} = p_{b,b} = 0.9$ the channel would stay for an average of ten time slots (i.e. ten packet transmission attempts) in the good state / bad state before it switches to the other state.

This model is a very simple model of channels with "bursty" error characteristics.

3 Deliverables

Each student pair has to submit **a single pdf/a file** (a pdf file with all fonts embedded, other formats will not be accepted) which includes the following items:

- The names and student-id's of both group members. If you have received approval to work as an individual, then please state this and also give the date on which you have received this approval.
- You need to give an **agreed-upon** percentage contribution for each partner, reflecting how much work either partner has spent. The relative weights will influence marking.
- Responses to **all** questions given below.
- A listing of your source code. While not strictly necessary, when marking we would appreciate if you use a pretty printer for your source code.

The pdf file has to be submitted to the departmental coursework dropbox, see `http://cdb.cosc.canterbury.ac.nz`. Please submit no later than **Friday, September 26, 2014, 11.59pm**. Late submissions are **not** accepted. Note that you can submit several versions of the pdf file to CDB, you do not have to wait until the last minute.

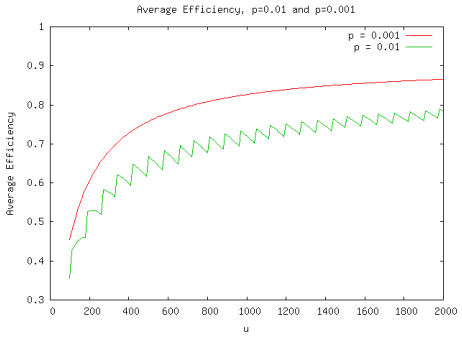

Figure 1: Average Efficiency for $p = 0.01$, $p = 0.001$ and varying u (actually, $n \approx 1.1 \cdot (u + o)$ is used on the x-axis).

4 Questions and Marking

Each pair of students should submit answers to all of the following questions.

1. Please explain your choice of probability distribution for the number s of erroneous bits in an encoded packet of n bits length. Please make all your assumptions clear and also give a brief description of how you generated those random numbers in your chosen programming language.

2. Please give a formula for the probability that a packet with n bits transmitted over a BSC with bit error probability p has at least one bit error. Please explain how you have arrived at this formula.

3. For the BSC channel model we consider two values for p, namely $p = 0.001$ and $p = 0.01$

   - Compute the average efficiency for $u = 100, 110, 120, \ldots, 2000$ and plot them in a graph with u or n on the x-axis and the average efficiency on the y-axis. You should assume that the number $n - k$ of check bits is always about ten percent of the packet size k, i.e. $n - k = \lceil 0.1 \cdot k \rceil$ where $\lceil x \rceil$ is the smallest integer that is larger than or equal to x. What can you observe? Please explain your results. For comparison, in Figure 1 the average efficiency for $p = 0.01$ and $p = 0.001$ for varying u (or n) is shown, based on analytical calculations. You can compare your results against these curves.
   - (For fun, will not affect marking): Can you find the optimal u for given p and ten percent check bits analytically, ignoring integer constraints?

4. Again for the BSC channel, we fix $u = 512$ and assume that we use ten percent check bits as above. Compute the average efficiency for $p =$

$0.0001, \ldots, p = 0.01$ and plot them in a graph with p on the x-axis and the average efficiency on the y-axis. What can you observe? Please explain your results.

5. Now consider the two-state channel with bit error rates $p_g = 10^{-5}$ and $p_b = 1.99 \cdot 10^{-3}$ which leads to a "long-term average" bit error rate of 10^{-3}. Fix $p_{g,g} = p_{b,b} = 0.9$ and $u = 1024$. The number $n-k$ of check bits is to be varied as $n - k \in \{0, 10, 20, 30, \ldots, 500\}$. Produce a graph showing $n - k$ on the x-axis and the average efficiency on the y-axis. Produce the same graph for a BSC channel with a bit error rate $p = 10^{-3}$, plot both graphs into one figure. What can you observe? Please explain your results.

Marking will mostly be based on these answers and to a small degree on the source code. We will not mark the source code for style (only really ugly or messy source code will get deductions) but for its ability to produce the right results.