EEEN 20060 – Communication Systems

Solutions to Problem Set 1

1. Explain the terms *noise*, *interference* and *distortion*, in the context of the physical layer of a communication system. Show that you understand the difference between these terms.

Noise is a random addition to the signal. An example is thermal noise, caused by the random motion of electrons in resistive elements in the receiver circuit and also in the cable (if used).

Interference is another unwanted addition to the signal, generated by something outside the channel of interest. On a cable, it could arise from leakage of signals from another pair of wires in the same cable, or effects of other electrical systems nearby. In a radio-link, it could arise due to other transmissions on (or near) the frequency being used.

Distortion is a change to the signal, caused by the characteristics of the channel. If the channel does not attenuate all frequency components of the signal equally, the signal will be distorted. If the channel does not delay all frequency components of the signal equally, the signal will be distorted.

2. You want to transmit data at 10 Mbit/s along a cable, using NRZ signals with average value 0 V. The voltage range at the transmitter is limited to ±8 V (you do not have to use all of this range, but you should aim to use most of it). From the viewpoint of the transmitter, the channel looks like a resistance of 600 Ω. The bandwidth is limited to 2.5 MHz.

   Design a suitable set of signals. Define the mapping from bits to signals. Calculate the average power sent into the channel by the transmitter, assuming a random sequence of 1s and 0s is being transmitted.

Using a set of 2 signals would require signals to be sent at 10 million per second, needing absolute minimum bandwidth 5 MHz. Using a set of 4 signals would allow each signal to carry 2 bits, so would only need 5 million signals per second, and could just about fit in a bandwidth of 2.5 MHz. Most practical solution would be to use a set of 8 signals, each carrying 3 bits. Then only need to send 3.33 million signals per second, and this fits easily in the 2.5 MHz bandwidth available.

Next choose signal voltages. These should be evenly spaced (for equal probability of error for all signals) and symmetrical about 0 V (to ensure average value 0 V). A simple set of voltages meeting these requirements is -7 V, -5 V, -3 V, -1 V, +1 V, +3 V, +5V, +7 V. (Note 8 values, all separated by 2 V, and symmetrical about 0 V.) This set of voltages could be used, or all the values could be scaled up by 8/7 to make full use of the allowed voltage range.

Errors will occur at the receiver when the noise that has been added to the signal brings the total voltage the wrong side of a threshold, so the receiver decides a different signal was most

likely to have been sent. By far the most common errors will be to adjacent signals (a much larger noise voltage would be needed to make the -5 V signal look like the +3 V signal). So choose the mapping so that adjacent signals differ in only one bit, so errors to adjacent signals will cause only one bit error. This requires a Gray code, for example:

Signal voltage:	-7 V	-5 V	-3 V	-1 V	+1 V	+3 V	+5V	+7 V
Bits	000	001	011	010	110	111	101	100

Assume each signal is transmitted as a constant voltage for a fixed time (300 ns), then the transmitted switches to the next signal. Power in each time interval $P_i = \frac{V_i^2}{R}$. With a random sequence of bits, each of the 8 signals is equally likely to be sent, so average power will be the average of all of these: $P = \frac{1}{8}\sum_{i=0}^{7} P_i = \frac{1}{8R}\sum_{i=0}^{7} V_i^2$

Using the ±7 V signal set above gives average power 35 mW.

3. Write a function to transmit data at 100 kbit/s using Manchester coding with signal values of ±2.5 V. The function should have two arguments:

bool *data	pointer to array of bits to be sent
int nBit	number of bits to be sent

You can use a function void output (float v, float t)
where v is the voltage to be transmitted, in volts,
and t is the time for which it is to be transmitted, in microseconds.

At 100 kbit/s, the bit time is 10 µs. In Manchester coding, each signal value is output for half the bit time, or 5 µs. An example function is given below:

```
#define LOW -2.5          // output voltage for low
#define HIGH 2.5          // output voltage for high
#define HALFBIT 5         // time for half bit (us)

void manchester(bool *data, int nBit)
{
    for (int k = 0; k < nBit, k++)      // for each bit
    {
        if (data[k])    // send a 1 bit
        {
            output(LOW, HALFBIT);
            output(HIGH, HALFBIT);      // rising edge
        }
        else            // send a 0 bit
        {
            output(HIGH, HALFBIT);
            output(LOW, HALFBIT);       // falling edge
        }
    }   // end for loop
}   // end function
```

4. A simple error detecting code adds one check bit at the end of a group of 18 data bits, so that the number of 1 bits in the new group of 19 bits is always an even number. The physical layer is such that each received bit has probability of error 5×10^{-3}. The receiver checks for an even number of 1 bits in each group of 19. If this test is passed, it removes the check bit and delivers the 18 data bits. What is the probability of these delivered data bits being incorrect? You may make approximations to simplify the calculation, but you must show clearly that you are doing this deliberately!

The error detection will fail if there are 2, 4, or any even number of errors in the group of 19 bits. Then at least one of the delivered data bits will be incorrect.

If each bit has probability p of being in error, then the probability of k errors in a group of n bits is $P(k) = \binom{n}{k}(1-p)^{n-k}p^k$. Thus the probability of 2 errors in the group of 19 bits is 3.926×10^{-3}. The probability of 4 errors is 2.247×10^{-6}. The sum of these is 3.928×10^{-3}.

An exact answer would involve adding the probability of all the even numbers of errors, but the probability of the higher numbers of errors will be negligible. The approximate answer is 3.93×10^{-3}.

5. To improve the reliability of the system in problem 4, it is proposed to add 3 check bits to each group of 18 data bits. Propose a method of doing this, based on the same parity check idea as in problem 4. Calculate the probability of an un-detected error.

Example: propose associating each check bit with a group of 6 data bits. Then an un-detected error would require 2, 4 or 6 errors in a group of 7 bits, but there are 3 such groups at risk.

Probability of 2 errors in a group of 7 is 5.12×10^{-4}. Probability of 4 errors is 2.15×10^{-8}. Probability of 6 errors is 1.09×10^{-13}. Total is 5.12×10^{-4}. This is the probability of undetected error in a group of 7 bits, call it P_1. Note that it is essentially the same as the probability of 2 errors in the group of 7 bits – the more complicated error combinations could have been ignored.

Then the probability of undetected error in any of the 3 groups is $1-(1-P_1)^3$. This gives the result 1.535×10^{-3}. This is a small improvement over the system in problem 4.

6. A cyclic redundancy check (CRC), with divisor 1101, is used to add check bits to the end of a group of 18 data bits. For the data bits 101001110011010011 (leftmost bit sent first), calculate the transmitted bits.

With a 4-bit divisor, the remainder will have 3 bits. So add 3 zero bits to the end of the data. Then divide by the divisor as shown below. The remainder is 110, so the transmitted bits are 101001110011010011110 (leftmost bit sent first).

								quotient																
divisor							1	1	0	1	0	1	1	0	1	0	1	1	1	1	0	1	1	0
1	1	0	1	1	0	1	0	0	1	1	1	0	0	1	1	0	1	0	0	1	1	0	0	0
				1	1	0	1																	
					1	1	1	0																
					1	1	0	1																
						0	1	1	1															
						0	0	0	0															
							1	1	1	1														
							1	1	0	1														
								0	1	0	1													
								0	0	0	0													
									1	0	1	0												
									1	1	0	1												
										1	1	1	0											
										1	1	0	1											
											0	1	1	1										
											0	0	0	0										
												1	1	1	1									
												1	1	0	1									
													0	1	0	0								
													0	0	0	0								
														1	0	0	1							
														1	1	0	1							
															1	0	0	0						
															1	1	0	1						
																1	0	1	0					
																1	1	0	1					
																	1	1	1	1				
																	1	1	0	1				
																		0	1	0	1			
																		0	0	0	0			
																			1	0	1	0		
																			1	1	0	1		
																				1	1	1	0	
																				1	1	0	1	
																					0	1	1	0
																					0	0	0	0
																						1	1	0

7. A link layer protocol uses a flag consisting of exactly 6 consecutive 0 bits to mark the start and end of a frame. The frame header includes a 4-bit type code (to identify data, ACK, NAK, etc.), a 4-bit sequence number and an 8-bit address. This is followed by a variable number of data bits, up to a maximum of 1500 bits. The trailer includes a 16-bit CRC, calculated over all the data bits and the header bits (excluding the flag). Bit stuffing is used on all the bits between the flags, and is applied after the CRC is calculated. Explain clearly how the receiver operates: How does it detect the start and end of the frame? How does it know if the frame has errors? How does it know how many data bits it has received?

Receiver detects start of frame if it sees exactly six 0 bits, followed by a 1 bit. (That means it has to receive 10000001.) It detects the end of a frame the same way, so it has to remember whether it is inside a frame or outside a frame at all times. While inside a frame, it has to apply the rule to reverse the bit stuffing: if it receives 5 consecutive 0 bits, followed by a 1, remove the 1 (as it was stuffed in at the transmitter).

When it detects the start flag, the receiver initialises its CRC hardware, and starts putting the received bits into it. When it detects the end flag, it has to look back 8 bits, to see if the remainder was zero. (Remainder should have been zero after processing the last bit of the 16-bit CRC.)

When the receiver detects the start flag, it begins counting bits. When it detects the end flag, it records the bit count. As the header and trailer are a fixed number of bits, it can subtract a constant to get the number of data bits in the frame. (Note that the start flag is not included in the bit count, so should not be included in the subtracted value.)

8. A link layer protocol uses the data frame described in problem 7, and transfers data using a simple stop and wait mechanism with both positive and negative acknowledgements, but without enquiries or flow control. Design the response frame (ACK or NAK). Calculate the throughput that could be achieved using frames of maximum size on a link operating at 100 kbit/s, with probability of bit error 2×10^{-5} and propagation delay 30 ms.

The response frame could be similar to a data frame, but with no data. The type code already includes provision for ACK and NAK. The sequence number would refer to the data block being acknowledged. The CRC would provide very good protection against undetected errors.

Frame overhead, H, is 48 bits. ACK size, A, proposed is also 48 bits. With max frame size, number of data bits, D, is 1500. Probability of bit error is p, and the fact that such a figure is given means that the errors are independent. Propagation delay τ and bit rate R are also given.

Throughput is then given by $U = \frac{DR(1-p)^{(D+H+A)}}{D+H+A+2\tau R}$ = 19.1 kbit/s.

(The probability of successful transmission is 0.969. The main reason for the poor throughput is the large wait to receive acknowledgements – the round trip delay of 60 ms would allow 6000 bits to be transmitted.)

9. Find the optimum number of data bits per frame for the protocol in problems 7 and 8. Calculate the maximum throughput that could be achieved.

Applying the result in the lecture notes gives maximum throughput at 14675 data bits per frame. With this frame size, the throughput is 52.6 kbit/s. (The probability of success is 0.744, so there will be significant numbers of re-transmissions.)

10. Re-design the protocol in problems 7 and 8 to allow almost continuous transmission. Increase the number of bits used to represent the sequence number if necessary. Find the throughput that your new design can provide, using 1500 data bits per frame.

A sliding-window protocol with selective reject would meet the requirements. With frames of length 1548 bits, frame transmission time is 15.48 ms. Almost 4 frames would be transmitted during the round-trip delay of 60 ms. With a 4-bit sequence number, there are 16 possible sequence numbers, so this should easily allow non-stop transmission after a NAK has been received.

Throughput is given by $U = \frac{DP_S R}{D+H}$ = 93.9 kbit/s.