CptS/EE 555
Homework 2 Solutions
September 14, 2001

I. Prove claims from p.100 regarding CRCs.
   a. CRC can detect single bit errors: proved in text. Observe that the requirement on the divisor is actually that it have two or more non-zero terms. However, the requirement that x^k and x^0 terms be non-zero is quite natural. If the x^k term is zero the divisor polynomial isn't of degree k. If the x^0 term is zero then the divisor polynomial is divisible by x^n for some $n > 0$, so the low-order bits of the error code are wasted.
   b. CRC can detect double-bit errors provided the divisor polynomial has a factor with at least three terms. Students observed in class that they had found a counterexample. Suppose the error is $x^3 + 1$ and the divisor is $x^2 + x + 1$. The latter divides the former evenly. It was suggested that perhaps the definition of double-bit error encompassed only errors that were at most the degree of the divisor polynomial apart. But those fall in the category of "burst error" below, so I'm left wondering what was intended here. Further investigation reveals that there is a class of divisor polynomials called "primitive polynomials" that are able to detect double bit errors quite far apart – far enough apart that they could detect all double bit errors in messages of practical lengths.
   c. CRC can detect any odd number of errors as long as C contains the factor $(x + 1)$. Suppose the error polynomial, E, is divisible by C. (That is E mod C = 0.) Then for some polynomial, D, E=$(x + 1)$ D. Now we use the observation that if two functions are equal they must evaluate to the same thing given the same argument. So E(1) = (1+1)D(1) = 0, which can only be true if E has an even number of terms. Thus C will detect any odd number of errors.
   d. CRC can detect any burst error as long as the burst is of less than k bits. Such an error polynomial is expressible as E=D x^n where the degree of D is less than k. Since from part a above, x^n is not a factor of the divisor polynomial, C, E mod C = 0 implies D mod C = 0, which requires that the degree of D be at least that of C. Hence all burst errors of less than k bits will be detected.

II. Design a finite state machine implementing bit stuffing as described on page 88.
   a. We designed the transmitter's FSM in class.
   b. Unlike the transmitter's FSM, the receiver's needs to detect the end-of-frame marker (01111110) and the illegal sequence (01111111). The transmitter presumably produces these by a mechanism independent of its bit-stuffing FSM. But the receiver's FSM is responsible for detecting these sequences.

III. The shift-register implementation of CRC obviously has only finitely many states. Thus CRC is implementable by a FSM. However, the shift register for

a degree k CRC divisor has k bits, hence 2^k different states. For a degree 32 divisor such as CRC-32 this is in excess of 4 billion states.

IV. Problems 2.9, 2.10, and 2.12

a. 2.9 Show that two-dimensional parity allows detection of all 3-bit errors. There are several approaches to this proof. You could observe that any odd number of bit errors change the overall parity of the message which is detectable. Indeed you only need a single parity bit for the whole message to detect all errors consisting of an odd number of bits. You might be more explicit in examining the cases: if all 3 bits are in a single row, clearly the parity for that row is wrong. If all 3 bits are not in a single row, then at least one row has only 1 wrong bit and the parity for that row is wrong.

b. 2.10 Give an example of a 4-bit error that would not be detected by 2-dimensional parity. What is the general rule for undetectable 4-bit errors? Any 4 errors making up the corners of a rectangle will go undetected.

c. 2.12 Show that the internet checksum will never be 0xFFFF unless every byte in the buffer is 0. That is, show that the ones-complement sum of non-zero quantities is non-zero. Observe that the sum is monotonically increasing unless there is a carry. If there is a carry, the new value will be at least 1.

V. Problems 2.23 and 2.41

a. 2.23 Once triggered by a duplicated data frame or ACK, each data frame and ACK will be sent twice and this will continue until either the end of the transmission or the loss of a data or ACK frame. If the receiver's algorithm includes retransmission of ACKs upon failure to receive the next data frame within a timeout, then the late arrival of a data or ACK frame will trigger the Sorcerer's Apprentice. Even if late arrivals are not possible, the loss of a data or ACK frame will trigger timeouts on both the sender and receiver. If their frames cross in flight the bug has been triggered.

b. 2.41 In this problem we look at the "birthday problem" phenomenon. In the first part we are asked for the probability of a collision between randomly chosen 48 bit addresses on a 1024-host network. (To disambiguate the question, we are looking for 1-P(there are no collisions)). The P(no collisions) = $\prod_{i=1}^{n=1023}\left(1-\frac{i}{2^{48}}\right)$ which for small n is approximated by $1-\sum_{i=1}^{n=1023}\frac{i}{2^{48}}$ or approximately $1-2^{-29}$. Thus, the probability of 1 or more collisions in this case is 2^{-29} or about 1 in 500 million. The second part asks us to consider what happens if we have 2^{20} such networks. What is the probability of a collision on one or more of those networks. In class I asserted we could sum the probabilities for the individual networks and came up with 2^{-9}. Ilia pointed out that this methodology clearly doesn't work if one asks the related question: what is the probability of no collisions on one or more of the networks, because in that case you get a number much bigger than 1 which is not a probability.

The answer to this lies in the fact that my use of the summing technique is only valid if we are considering events with low probability (see the box on p. 96). Since the probability of no collision on a single network is high the approximation isn't valid. Still, we should work it through carefully and see what happens. So, P(one or more collisions on 2^{20} networks) is $1 -$ P(no collisions on 2^{20} networks) is $1-\left(1-2^{-29}\right)^{2^{20}}$. And now we see where the previous approximation on p. 96 came from: for small ε, $(1\pm\varepsilon)^n$ is approximately $(1\pm n\varepsilon)$. Plugging in the numbers we find P(one or more collisions on 2^{20} networks) is $1-(1-2^{20}2^{-29})$ which is 2^{-9} as we had previously calculated. In the third part of this question we are asked to revisit the first part of the problem but with 2^{30} individuals in the population. In this case, the approximation used in the first part of the problem breaks down: it yields a "probability" of 2^{11} which is clearly nonsensical. To address this, following the suggestion of problem 8.18 we can work instead with logarithms of probabilities. log(Probability of no collisions) = $\log\prod_{i=1}^{n=2^{30}}\left(1-\frac{i}{2^{48}}\right)=\sum_{i=1}^{n=2^{30}}\log\left(1-\frac{i}{2^{48}}\right)$. We can make use of the Taylor series approximation for $\log(1-\varepsilon)$ being $-\varepsilon$ to calculate $\log\prod_{i=1}^{n=2^{30}}\left(1-\frac{i}{2^{48}}\right)=\sum_{i=1}^{n=2^{30}}\log\left(1-\frac{i}{2^{48}}\right)\cong-\sum_{i=1}^{n=2^{30}}\frac{i}{2^{48}}\cong-2^{11}$. So, P(no collisions) = $e^{-2048}\cong10^{-890}$. In other words, the certainty of collision extends to almost 900 decimal places. Whew!