TCOM 370 NOTES 99-8

ERROR CONTROL: BLOCK CODES

THE NEED FOR ERROR CONTROL

The physical link is always subject to imperfections (noise/interference, limited bandwidth/distortion, timing errors) so that individual bits sent over the physical link cannot be received with zero error probability. A bit error rate (BER) of 10^{-6}, which may sound quite low and very good, actually leads on the average to an error every $\frac{1}{10}$-th second for transmission at 10 Mbps. Even with better links, say BER=10^{-7}, one would make on the average one error in transferring a binary file of size 1.25 Mbytes. This is not acceptable for "reliable" data transmission.

We need to provide in the *data link control protocols* (Layer 2 of the ISO 7-layer OSI protocol architecture) a means for obtaining better reliability than can be guaranteed by the physical link itself.

Note: Error control can be (and is) also incorporated at a higher layer, the transport layer.

ERROR CONTROL TECHNIQUES

Error Detection and Automatic Request for Retransmission (ARQ)

This is a **"feedback"** mode of operation and depends on the receiver being able to **detect** that an error has occurred. (Error *detection* is easier than error *correction* at the receiver). Upon detecting an error in a frame of transmitted bits, the receiver asks for a retransmission of the frame. This may happen at the data link layer or at the transport layer. The characteristics of ARQ techniques will be discussed in more detail in a subsequent set of notes, where the delays introduced by the ARQ process will be considered explicitly. A very simple example of coding to allow error detection is the addition of a *parity bit* to 7-bit ASCII characters; by examining the parity of

the received 8 bits, one can decide if a single error occurred during transmission of the 8-bit byte.

Forward Error Correction (FEC)

FEC improves reliability by sending enough extra (redundant) bits that allow data bits to be received correctly even though some transmission errors may have occurred. A simple way to add redundancy for error correction is to simply *repeat* bits. Even though this is not an efficient way to use redundancy, it serves as a very simple example. For example, each data bit to be transmitted may be repeated three times, and at the receiver a decision is based on the majority bit-type in each group of three bits. Clearly, with this scheme we are able to recover from one error in each group of three bits, but at a cost of slowing down the useful data rate to one-third of the original rate, or we need three times the bandwidth.

The term "forward" in FEC signifies that the transmitter does the work of encoding for error control before forwarding bits to the receiver. The receiver does the best it can do to correct any errors. Thus the receiver does not rely only on ARQ to obtain frames correctly (there may be little or no "feedback" from the receiver in these cases.) FEC may be used when an ARQ feedback channel is not available, or when ARQ can only be used to a limited extent (e.g. only one or two ARQ's permitted per frame), or when ARQ is not desirable in the application, e.g video (where real-time operation is needed).

Generally we may employ both error correction and error detection techniques in transmitting frames of binary digits. In a typical scenario we may encode the data with enough redundancy or overhead to allow the receiver to *correct* certain types of errors. (For illustration, consider a repetition code.) The decoded bits may then be checked to see if they are free from error (the error detection part). For example, we may have applied the repetition code to data bits that already had a parity bit added, and we can check the parity of the decoded bits. If errors are detected, the frame may be discarded or an ARQ request may be sent.

Useful techniques for adding redundancy to data bits to enable error detection or error correction are often based on the ideas of **block coding.** (Another type of coding technique is *convolutional coding,* but we will not discuss it here.)

ERROR DETECTION BY PARITY CHECKING

As an example of this, we have seen that in asynchronous transmission of 7-bit ASCII characters an 8-th bit may be added. This is called a "parity" bit and depending on the scheme used, the parity bit may be a bit that makes the total number of 1's in the 8-bit byte an even number ("even parity") or an odd number ("odd parity"). At the receiver, the parity of every 8-bit received byte is checked. If the parity differs from what is expected (odd or even), the receiver detects that an error has occurred.

(Actually it detects that either one, or three, or in general an odd number of errors, have occurred. An even number of errors will leave the parity unchanged! If the BER for the link to begin with is fairly small, the probability of more than one error for the byte is small anyway.)

Parity bits are computed using **modulo-2 (mod-2) arithmetic** on binary digits, for which 0+0=0, 0+1=1, and 1+1=0. [This can be implemented using an "exclusive-or" electronic gate.] Suppose we write each ASCII character in row-form, starting from most significant bit (msb, bit position 6) to least significant bit (lsb, bit position 0). Then for **even** parity the parity bit = [lsb+bit 1 + ... + msb]; if the even parity bit is added mod-2 to this sum, the result is zero. For example, if the character bits are 1110000, then the even parity bit is 1 and the mod-2 sum of all 8 bits is 0. At the receiver the even parity condition is checked by obtaining the mod-2 sum of the 8 bits. This sum will not be 0 if a single error has occurred. A similar consideration applies for odd parity.

Parity checking may be extended to work for a **block of characters** in a frame. For each character there is a parity bit provided, as above. Suppose we write each character in the sequence of M characters in row-form, and precede the character with the character parity bit (bit position 7; here even parity is assumed):

STX w/parity	1	0	0	...	1	0
Byte 1	parity	msb	bit 5	...	bit 1	lsb(bit 0)
Byte 2	parity	msb	bit 5	...	bit 1	lsb(bit 0)
	.					
	.					
	.					
Byte M	parity	msb	bit 5	...	bit 1	lsb(bit 0)
ETX w/parity	0	0	0	...	1	1

Here the STX and ETX characters also have parity bits before them, and the m-th Byte is the m-th Character with a parity bit before it.

We then add a final byte called the block check character (BCC):

BCC	p7	p6	p5	...	p1	p0

It is computed to maintain the parity of each **longitudinal column** from lsb to msb at a specified parity (say even). Thus the BCC is a character for which bit n = pn is a parity bit for the set of bits in position n of the M+2 characters above it, for n=0,.1, ..., 6. The last bit p7 of the BCC is computed to maintain its own parity as for each character in the frame.

It is then clear that if Char-m is the m-th character, and we think of it as a 7-bit row, then the modulo-2 sum of the rows STX + Char-1 + Char-2 + ... + Char-M + ETX = BCC(6:0), the last 7 bits of BCC (for even parity).

The BCC allows error patterns to be detected that would otherwise go undetected with just character parity. For example, an even number of errors in one character is not detected by character parity alone, but will be detected in the block scheme as long as the same pattern of errors does not occur in another character.

BLOCK CODES

The addition of a parity-check bit to a character or a sequence of data bits is an example of a block coding technique, albeit a simple one for error detection only. More generally, block codes can be used for more powerful error detection capability and for error correction capability.

Definition:

> An (N,k) binary block code provides a codeword of N bits for each k-bit data word that is to be transmitted, N>k.

In general, for an (N,k) block code, to transmit k bits of data we convert it to an N-bit string according to a specific rule, and send out the N bits. Thus there is some redundancy added, we now have to transmit N bits instead of only k. The conversion of k bits to N bits can take place in many different ways. A simple example is a (3,1) block code in which each bit to be transmitted is converted to a 3-bit string by repeating it twice.

We see that there are **2^k data words** of length k, and from the set of **2^N possible combinations of N bits**, a particular (N,k) code uses only a **subset** of 2^k codewords of length N and assigns a codeword to each data word.

For example, an (8,7) binary block code is being used whenever we transmit ASCII characters with a single parity bit. For a given character, the even parity bit is a particular one of the two possible (1 or 0) to make the codeword parity turn out even. More generally, N-k can be larger than unity.

Repetition Code

Suppose we have k data bits. To gain resilience against transmission errors we may repeat each bit 2 times, so that the total size of the transmitted word is N=3k. Let the probability be **p_e** that any individual transmitted bit is received in error (i.e., p_e is the BER for the physical link). Now for each bit sent out in this scheme, as long as we have no more than one error in its three positions we are able to receive it correctly (using majority logic). Let us calculate the **probability of data block error**, that is the probability of receiving at least one of the k data bits in error.

We assume that bit errors occur *independently* with the same probability p_e for each bit. For **each bit**, since it is repeated twice, the probability of error is the probability of 2 or 3 errors out of three. This is

$$\binom{3}{2} p_e^2(1-p_e) + \binom{3}{3} p_e^3 = 3p_e^2 - 2p_e^3$$

which for small p_e can be approximated as $3p_e^2$. (Consider for example $p_e=10^{-6}$; then $2p_e^3$ is about 10^{-6} times smaller than the first term $3p_e^2$). Thus the **probability of successful transmission of a bit** is approximately **$1-3p_e^2$**.

Finally, the **probability of successful transmission of all k bits** is **$(1-3p_e^2)^k$**, because we are assuming *independence* of bit errors for the transmitted bits; the probability of no error for k bits is the product of the probabilities of no error for each of the k bits.

Also, $(1-3p_e^2)^k$ is approximately **$1- k3p_e^2$**. [Use the approximation $(1-p)^m$ is approx. $1-mp$ for $p<<1$ and $mp<<1$. Consider typical numbers where k may be of the order of 10 or 100 and p_e is of the order of 10^{-6}.]

Therefore the **probability of receiving a block of k data bits in error** (from the 3k transmitted bits) is **$k3p_e^2$**. For **$p_e=10^{-4}$** and k=4 we find that the probability of block error is approximately **$1.2\text{x}10^{-7}$**. This is **much better than sending k data bits directly without coding.** Without coding, the probability of at least one error for the k-bit word is $1-(1-p_e)^k \approx kp_e$, because the probability of no error is $(1-p_e)^k$. We find that kp_e = **$4\text{x}10^{-4}$** for k=4 and $p_e= 10^{-4}$. Thus the repetition code gives much better performance. However, the **price we have to pay** is that the effective data rate is one-third of the bit rate for the link.

In general, **for an (N,k) block code we define the "rate" to be $\frac{k}{N}$.** This is the factor by which the actual transmission rate is multiplied to get the rate at which the original binary data is transmitted.

- What we would like are codes with good rates (close to unity) that provide good error performance.

LINEAR BLOCK CODES

A **linear (N,k) block code** is one for which a **k-by-N binary generator matrix G** with 0 or 1 entries can be defined, such that a codeword row **c** corresponding to a data word row **d** (the binary words **c** and **d** are rows of N bits and k bits, respectively) is obtained as

$$\mathbf{c} = \mathbf{dG}$$

where the binary addition operations involved are ***modulo-2*** operations.

Hamming Code

As an example, the so-called **Hamming (7,4)** code has the G matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

The codeword corresponding to the data word [1010] is therefore

$$[1010]G = [1010001].$$

(This is an example of a **systematic code**. Here the codeword is always formed with the dataword to which is appended an extra set of bits. The G matrix has the identity matrix forming its first 4 columns).

This Hamming code is a rate $\frac{4}{7}$ code, and it has a very interesting property, as we will see below.

Hamming Distance

- The **Hamming distance between any pair of codewords** is the **number of bit positions at which they differ.**

For example, the Hamming distance between codewords [1010001] and [1000110] is 4. (This is also the same as the number of 1's in the result for the modulo-2 vector sum [1010001]+[1000110].) It turns out that for the Hamming (7,4) code, every pair of codewords has a Hamming distance of 3 or more between them. Thus the **minimum distance d_{min}** between codewords is $d_{min}=3$ for this particular code.

The **minimum distance** is an important parameter of a linear code. Clearly, for good error correction capability, we want the codewords to be as "far apart" from each other as possible, so that an error in transmitting a codeword will still keep it nearest to the transmitted codeword rather than move it closer to some other codeword. (This is an argument based on the geometry of codeword locations in N-space). This also suggests how we should decode; we choose from the received N-bit word that codeword that is closest to it from amongst all the 2^k codewords, and put out the corresponding k-bit data word as the final output. This can be implemented efficiently in a variety of ways, including table look-up.

Since $d_{min}=3$ for the Hamming (7,4), it follows that we can actually **correct** $\frac{(d_{min}-1)}{2}$ =1 error in the transmission of each seven-bit codeword. We can show that in this case the probability of correct reception of the 4-bit data word is approx. $\mathbf{21p_e^2}$, which for $p_e=10^{-4}$ is approximately $2.1\text{x}10^{-7}$. This is very close to the performance of the rate 1/3 repetition code, but here the rate is much better at 4/7.

[More generally, Hamming $(2^m-1, 2^m-m-1)$ block codes with $d_{min}=3$ can be found for all integer m. These allow single errors to be *corrected* in codewords of size 2^m-1, the rate getting very close to 1 as m grows. The (7,4) code above was a special case of this with m=3**]**

Hamming Weight and Hamming Distance

The **minimum Hamming distance** d_{min} is an important parameter of a linear block code. The **Hamming weight** of a codeword is easier to compute, it is simply the number of 1's in the codeword. For a linear block code, if $\mathbf{c_1}$ and $\mathbf{c_2}$ are codewords then so is $\mathbf{c_1} + \mathbf{c_2}$. Also, the Hamming distance between $\mathbf{c_1}$ and $\mathbf{c_2}$ is the Hamming weight of $\mathbf{c_1} + \mathbf{c_2}$. Actually, $\mathbf{c_1} + \mathbf{c_2}$ must also be a valid codeword because it must correspond to the message $\mathbf{d_1} + \mathbf{d_2}$ ($\mathbf{d_1}$ produces codeword $\mathbf{c_1}$ and $\mathbf{d_2}$ produces $\mathbf{c_2}$). To find the Hamming distance d_{min} for the linear block code we should consider all pairwise sums of distinct codewords and look at their Hamming weights.

- Since the pairwise sums themselves form all codewords, the **minimum Hamming distance** is simply the **minimum Hamming weight for all the non-zero codewords** in the code!