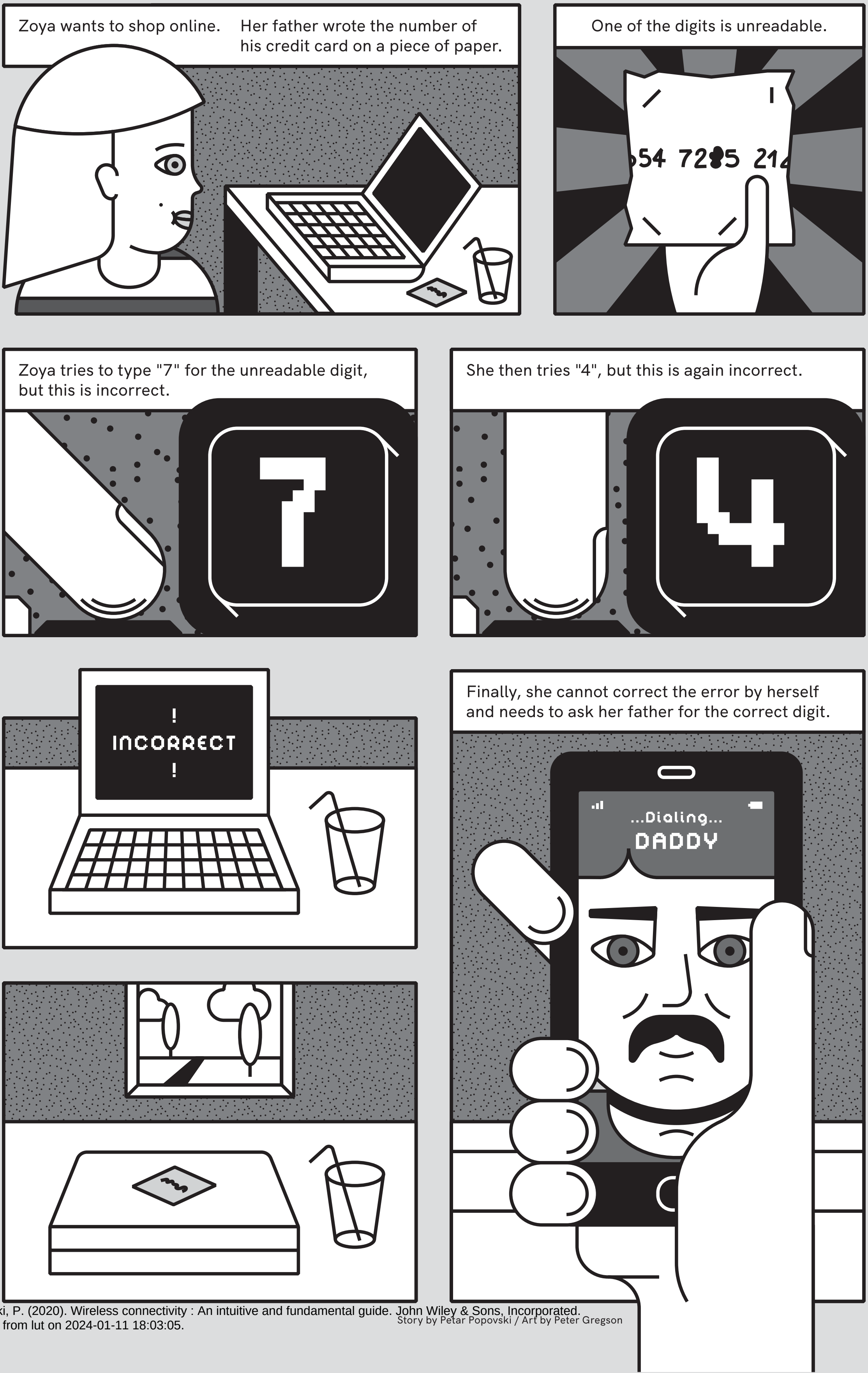

Zoya wants to shop online.
Her father wrote the number of his credit card on a piece of paper.
One of the digits is unreadable.
54 72 5 21
Zoya tries to type "7" for the unreadable digit, but this is incorrect.
7
She then tries "4", but this is again incorrect.
4
!
INCORRECT
!
Finally, she cannot correct the error by herself and needs to ask her father for the correct digit.
...Dialing...
DADDY

Story by Petar Popovski / Art by Peter Gregson

7 Coding for Reliable Communication

The fundamental problem in information and communication theory is how to achieve reliable communication over channels that are originally unreliable in a sense that each individual transmission over the channel has the chance to be received incorrectly. We have seen in the previous chapter that, if a single channel use is unreliable, then non-zero throughput can only be achieved by grouping the data bits into packets, where each packet spans multiple channel uses and contains redundant bits for checking the integrity of the packet.

It is important to note that checking for errors does not directly improve the reliability of the data bits. For example, consider a binary symmetric channel (BSC) where the probability of error is $p = P(1|0) = P(0|1)$. Xia transmits a packet to Yoshi, containing b data bits and c check bits. The error check is assumed ideal: if there is at least one erroneous bit in the packet, Yoshi will detect a packet error and the packet will be considered to be erased. The probability of packet error is given by (6.18) and the average goodput is given by (6.20), and here it is written in a more convenient form:

$$G = \frac{b}{b+c}(1-p)^{b+c}. \tag{7.1}$$

This mode of transmission, where the probability of error for a single data bit remains unchanged, is termed *uncoded transmission*. We remark here that this term is not completely correct, as the addition of the c bits for error detection is a form of coding[1]. The way to improve the reliability of the individual data bits is to use redundancy in a form of a *forward error correction (FEC)* code. We start to introduce the methods for error correction through some simple examples involving the BSC.

7.1 Some Coding Ideas for the Binary Symmetric Channel

7.1.1 A Channel Based on Repetition Coding

A trivial way to improve the reliability is to repeat the same data bit/symbol multiple times. Let us take the BSC with probability of error p, depicted in Figure 7.1(a) and referred to as a p-channel in this discussion. For easier explanation, let us assume that one channel use

1 Strictly speaking, an error detection code can also be used for error correction, but so far we have considered only its error detecting functionality.

Wireless Connectivity: An Intuitive and Fundamental Guide, First Edition. Petar Popovski.
© 2020 John Wiley & Sons Ltd. Published 2020 by John Wiley & Sons Ltd.

takes 1 s. Xia increases the transmission reliability by using the following strategy: if she needs to send the bit value 1, she uses the channel three times and transmits 111. Similarly, she transmits 000 for the data bit with value 0. Since the BSC can flip any of the transmitted bits, Yoshi receives the triplet $y_1y_2y_3$ that can take any of the 8 possible values:

$$y_1y_2y_3 \in \{000, 001, 010, 011, 100, 101, 110, 111\}.$$

Yoshi applies majority voting in order to decide whether 000 or 111 has been transmitted. If there is at most single bit error, the decision of Yoshi is correct; otherwise, it is incorrect. With this type of decoding, the probability that an error will occur is:

$$p_{\mathrm{E}} = 3p^2(1-p) + p^3. \tag{7.2}$$

Once the transmission/decision rules are fixed in the way described above, then it can be said that we have created a *new binary symmetric channel* that has a probability of error equal to p_{E}. This channel is depicted in Figure 7.1(b) and is referred to as the r-channel. It can be easily checked that $p > p_{\mathrm{E}}$ whenever $p \leq 0.5$ [2].

Nevertheless, there is a price to pay for decreasing the probability of error. A single use of the new channel takes three seconds instead of one. Let us now try to find the goodput, in bits per second, that can be achieved by using each of the channels above. As discussed

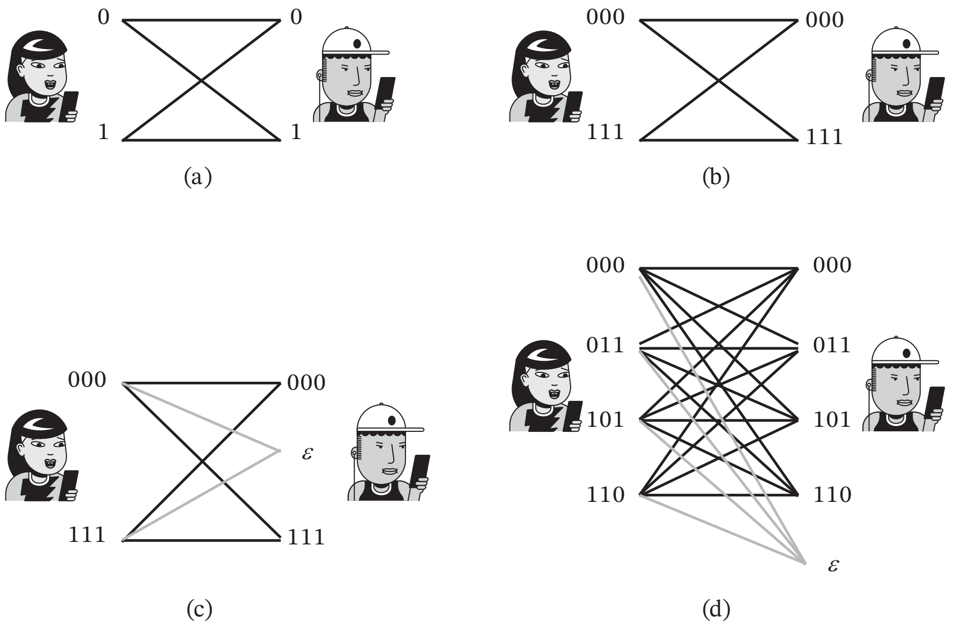


Figure 7.1 Binary symmetric channel (BSC) between Xia and Yoshi and three channels derived from the BSC through repetition/coding. (a) p-channel: BSC with probability of error (p); (b) r-channel: repetition coding with majority voting; (c) ϵ-channel: repetition coding with erasures; (d) c-channel: coding with four inputs.

2 Note that it is sufficient to consider the BSC with $p \leq 0.5$, since if the opposite is true, then the receiver should always interpret the received symbol 0 as 1 and vice versa, which brings us back to the case of a BSC with $p \leq 0.5$.

in the previous section, obtaining non-zero goodput requires grouping the data bits, adding error check bits in order to obtain a packet with checkable integrity, and then sending it by making use of multiple channel uses. Xia takes b data bits and adds c check bits (the error check is assumed perfect), and the packet length is $l = b + c$. If the p-channel is used, then the achieved goodput is:

$$G_\mathrm{p} = \frac{b}{l}(1-p)^l \quad \text{(bit/s)}. \tag{7.3}$$

When the r-channel is used, we still need to make l uses of that channel, which corresponds to $3l$ uses of the p-channel, such that the transmission is three times longer. This leads to the following goodput:

$$G_r = \frac{b}{3l}(1-p_\mathrm{E})^l \quad \text{(bit/s)}. \tag{7.4}$$

We can write G_r in a different way:

$$G_r = G\frac{1}{3}\left(\frac{1-p_\mathrm{E}}{1-p}\right)^l \tag{7.5}$$

from which it can be seen that the repetition coding will lead to benefit in terms of goodput only if

$$\frac{1}{3}\left(\frac{1-p_\mathrm{E}}{1-p}\right)^l > 1. \tag{7.6}$$

As stated above, we only need to consider the case $p \leq 0.5$. It turns out that G_r is not always higher than G, such that the reliability gained in repetition coding does not necessarily translate into a better goodput. For example, for $l = 100$ and $p = 0.495$ it can be calculated that $G_r = 0.546G$. However, if we increase the packet size to $l = 300$ and keep the other parameters unchanged, it follows that $G_r = 1.15G$ and the channel obtained by repetition coding leads to a higher goodput. When the original channel is very reliable and $p = 10^{-6}$, then repetition reveals its inefficiency. The reader can check that even with $l = 1000000$, it is still $G_r < G$, as the improved reliability cannot repair the damage made by decreasing the nominal transmission rate to $\frac{1}{3}$.

7.1.2 Channel Based on Repetition Coding with Erasures

Now we create a different channel by using again three repetitions over the BSC, but change the decision rule at Yoshi's side. If Yoshi receives 000 or 111, then he decides 000 or 111, respectively. Otherwise, if Yoshi receives any of the symbols 001,010,011,100,101,110, he announces an erasure. The channel created in such a way is shown in Figure 7.1(c) and is referred to as the ϵ-channel. The probabilities that define the ϵ-channel can be determined as follows:

- Probability of successful transmission:

$$p_\mathrm{S} = (1-p)^3. \tag{7.7}$$

- Probability that error occurs (all three bits in error):

$$p_\mathrm{E} = p^3. \tag{7.8}$$

- Probability of erasure is:

$$p_{\mathrm{ERS}} = 3p(1-p)^2 + 3p^2(1-p). \tag{7.9}$$

Using the erasures, Yoshi can now apply a different strategy for decoding the packets. Let Yoshi receive a packet over l uses of the ϵ-channel, corresponding to $3l$ BSC channel uses, which has a duration of $3l$ s. Let us assume that p of the BSC channel is small, such that the probabilities of error p_{E} and erasure p_{ERS} are also small. More specifically, assume that l is large and $p_{\mathrm{ERS}} = \frac{1}{l}$, such that only a single erasure is expected in the received packet. Yoshi tries to reconstruct the packet: he puts 0 at the bit position that has been erased and runs the error check again. Note that we are still assuming that the error check is ideal. If no error is indicated, then Yoshi accepts 0 at the erased position and the packet is correctly received. If not, Yoshi puts 1 at the bit position, runs the error check again and, if correct, Yoshi puts 1 and the erased position, thereby recovering the packet correctly. If neither 0 or 1 work, then Yoshi has to conclude that some of the other bits are erroneous and he discards the packet as incorrect.

The strategy based on flipping bit combinations until the error check flags that the packet is correct does not need to be used only with the erasure channel. The same strategy can be used for the p-channel, which is without repetition and erasures. If the received packet is flagged by the error checks as erased, then Yoshi starts to flip the bits one by one and, after each flip, he runs an error check. If no correct packet is obtained, then Yoshi starts to try all possible 2 bit combinations at all possible pairs of positions. Eventually, for some m, Yoshi will try all of the 2^m combinations for all possible sets of m out of l positions and the error check will indicate a correct packet.

The reasoning in the previous paragraphs should imply that Yoshi will eventually find "the correct" bit error pattern and decode the packet correctly, although the packet was originally erased. However, this reasoning is flawed. The problem is that we have assumed that the error check, created by adding c bits to the b information bits, is ideal. However, that assumption may only be approximately correct if the number of errors is very small and becomes completely incorrect as the number of bit errors increases. To see why this is the case, note that when Yoshi receives the packet of $l = b + c$ bits, there are 2^l possible bit sequences that Yoshi can receive and there are 2^b among them that represent the correct codewords. If an erroneous codeword is received and m bits are flipped to look for the right combination, then as m grows it becomes increasingly more likely that by flipping bits Yoshi will obtain one of the 2^b codewords that is different from the one that has originally been sent.

We therefore limit only to flipping the bits that correspond to the erasures and, if the number of erasures is small and the error check is ideal, then Yoshi will obtain the correct packet. Thus the probability that a packet of l bits is received correctly is $(1-p^3)^l$ and the goodput is:

$$G_\epsilon = \frac{b}{3l}(1-p^3)^l = G\frac{1}{3}\left(\frac{1-p^3}{1-p}\right)^l \quad \text{(bit/s)}. \tag{7.10}$$

Taking a packet of length $l = 200$ and $p = 0.006$, one gets $p_{\mathrm{S}} = 0.9821$, $p_{\mathrm{E}} = 2 \cdot 10^{-7} \approx 0$ and $p_{\mathrm{ERS}} = 0.018$, such that the expected number of erasures in a packet is less than 4, i.e. $l \cdot p_{\mathrm{ERS}} = 3.6$. For the chosen values one can calculate:

$$G_\epsilon = 1.11G > G_r > G.$$

One can conclude that there is a region of values for p and l for which repetition with erasures offers the highest goodput. However, can we do even better while still using three BSC channel uses per one data bit?

7.1.3 Coding Beyond Repetition

At first glance, repeating the bit and applying majority voting at the receiver seems to be the only possible way to improve channel reliability when three channel uses are bundled together. Assuming that one symbol is defined through three BSC uses, one can possibly choose from 8 different input symbols 000,001, 010,011, 100,101, 110,111. Let us call them 3-BSC symbols, for brevity. Differently from this, when repetition coding is used, we are restricted to choosing either 000 or 111.

With this insight, we can now try to choose four possible input symbols, for example 000,001, 011,111. For each use of this new channel, Xia picks two new data bits and uniquely assigns one of the input symbols. This is very similar to the transmission over a baseband channel that uses QPSK, except that now the four constellation points are chosen among the eight possible patterns that can be created by three bits. If the packet contains l bits, then it can be sent by using $\frac{l}{2}$ 3-BSC symbols, which corresponds to a total of $\frac{3l}{2}$-BSC channel uses and therefore a duration of $\frac{3l}{2}$ s. If we assume that the probability of error of the original channel is $p = 0$, then the nominal goodput of this channel is:

$$\frac{b}{\frac{3l}{2}} = \frac{2}{3}\frac{b}{l} \quad (\mathrm{s}).$$

The four input 3-BSC symbols can be selected in $\binom{8}{4} = 70$ different ways. Each of the different selections of the set of four 3-BSC symbols defines a different communication channel with four inputs. The number of outputs from this channel depends on how the erasures are treated, but in general, there can be 8 possible output symbols, that is, all possible 3-BSC symbols. The four selected 3-BSC symbols that can act as valid inputs to the newly defined channel can be seen as length 3 *codewords* for the original BSC channels.

Following our discussion in Chapter 5 about selecting the modulation constellations and the assumption that $p \leq 0.5$, we should select the four points that have the highest separation among each other. With that criterion, the best choice of the input symbols is:

$$000 \quad 011 \quad 101 \quad 110 \tag{7.11}$$

where the Hamming distance between each pair of codewords (valid 3-BSC symbols) is two. Similar to the second channel created with repetition coding, we define the channel outputs as follows. Yoshi uses a decision rule that is based on erasures, such that whenever he receives a 3-BSC symbol that is not part of the input symbols, 001,010, 100,111, then Yoshi announces an erasure. The channel created in that way is illustrated in Figure 7.1(d) and will be referred to as the c-channel, defined with the probabilities:

- Probability of successful transmission:

$$p_{\mathrm{S}} = (1-p)^3. \tag{7.12}$$

- The probability of error is more involved, as there are four input symbols. For example, when the symbol 101 is sent, the probability to receive a specific incorrect symbol, say 011, corresponds to the probability that exactly two errors have occurred:

$$p^2(1-p) \tag{7.13}$$

  and this is the probability associated with the edge, in this case 011–101. However, the probability that an error has occurred when a symbol, say 101, is sent is three times higher, as error occurs if either of the three remaining triplets 000,011, 110 is received:

$$p_\text{E} = 3p^2(1-p). \tag{7.14}$$

- Probability of erasure is:

$$p_\text{ERS} = 3p(1-p)^2 + p^3. \tag{7.15}$$

If we assume again that p is small, the number of erasures is small, the error check is ideal, and Yoshi attempts different bit patterns at the erased positions until getting a correct packet, then the goodput can be calculated as:

$$G_\text{c} = \frac{2}{3}\frac{b}{l}(1-p_\text{E})^{\frac{l}{2}} \quad (\text{s}). \tag{7.16}$$

It can be noted that G_c increases the goodput with respect to repetition coding in two ways. First, the factor $\frac{1}{3}$ is increased to $\frac{2}{3}$, as now two data bits per 3-BSC symbol (a triplet) are sent. Second, the number of required triplets is $\frac{l}{2}$, which decreases the packet length measured in number of uses of the c-channel. However, the total duration of the packet in terms of BSC uses is $\frac{3l}{2} > l$, which indicates redundancy. The objective in coding is to reduce this redundancy while still meeting the target error performance.

7.1.4 An Illustrative Comparison of the BSC Based Channels

Figure 7.2 provides illustrative results on the goodput provided by each of the channels defined above: p-, r-, ϵ-, and c-channels. The figure shows the value of the goodput normalized by $\frac{b}{l}$, such that, for example, what is plotted for G_p is $(1-p)^l$. The packet length is fixed to $l = 128$. It can be noticed that when the error probability of the original BSC is very low, then using *uncoded transmission* and sending the bits one by one through the BSC leads to the highest goodput G_p. However, as p increases, G_p rapidly decreases, such that the goodput G_c, that is not based on repetition, leads to the highest goodput. The reason is that the use of code balances the rate loss that occurs due to coding (which is $\frac{1}{3}$ for the repetition code), while still improving the reliability of each individual transmission; yet, less compared to the repetition code.

Figure 7.3 is supplementary and should be used for sanity assessment of the assumption about an ideal error check. It shows the expected number of erased symbols for the channels that feature erasures, which are the ϵ-channel and the c-channel. Note that each symbol is represented by a triplet of transmitted values 0, 1. However, the ϵ-channel uses $l = 128$ triplets, while the c-channels uses 64 triplets. Therefore, the expected number of erased symbols for the c-channel is approximately half that for the ϵ-channel, but otherwise the fraction of erased symbols is almost the same for both in the considered range of p. The reader should also notice that a single erased symbol for the c-channel corresponds

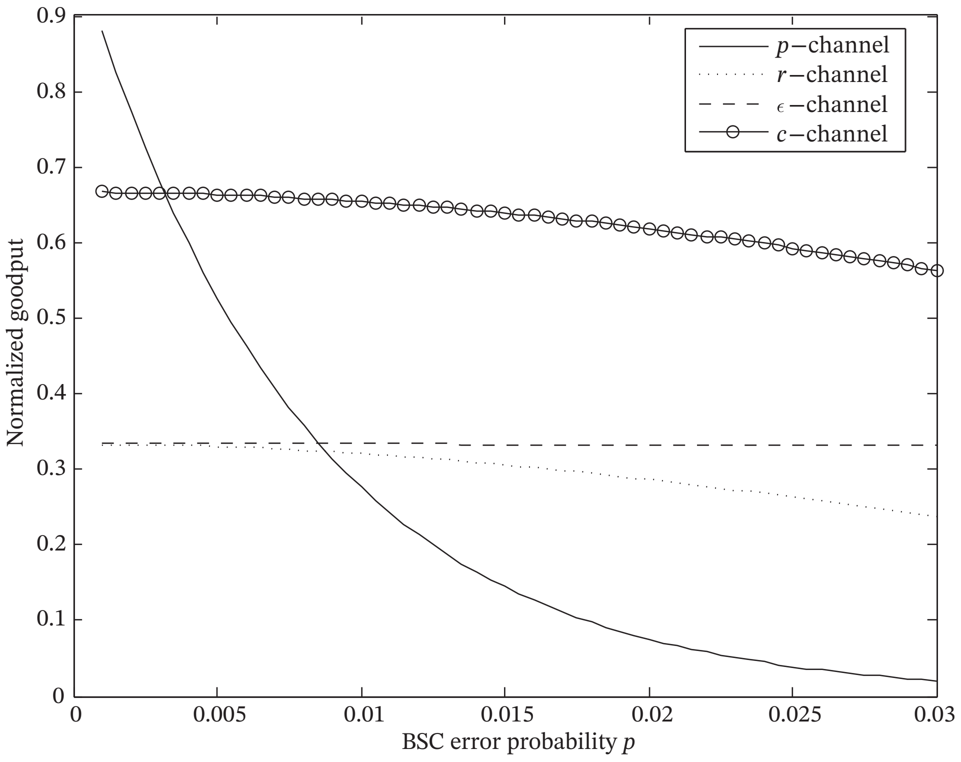


Figure 7.2 Comparison of the normalized goodput $G\frac{l}{b}$ for each of the four communication channels. The packet length is $l = 128$ bits.

to two erased bits in the packet. In any case, the average number of erasures stays low for $p < 0.015$ and we can use the ideal error check assumption to interpret the relation among the goodputs. A realistic comparison would require that, in addition to l, we specify b and the actual used c bit error checking code.

7.2 Generalization of the Coding Idea

Using the simple case of the BSC, we have illustrated the main idea behind FEC or *channel coding*, which can be summarized as follows. Given a channel with an unreliable single channel use, group multiple channel uses to create an expanded version of the channel. Then select a subset of the possible inputs for the expanded channel to represent valid input symbols. Those valid input symbols for the expanded channel are codewords for the original channel. This leads to a channel that is more reliable than the original one. The main tension in the design is explained as follows. On the one hand, how to select as many as possible inputs for the expanded channel and in this way prevent the decrease of the nominal data rate. On the other hand, how to select as few inputs as possible in order to guarantee high reliability for the transmission of a single input in the expanded channel.

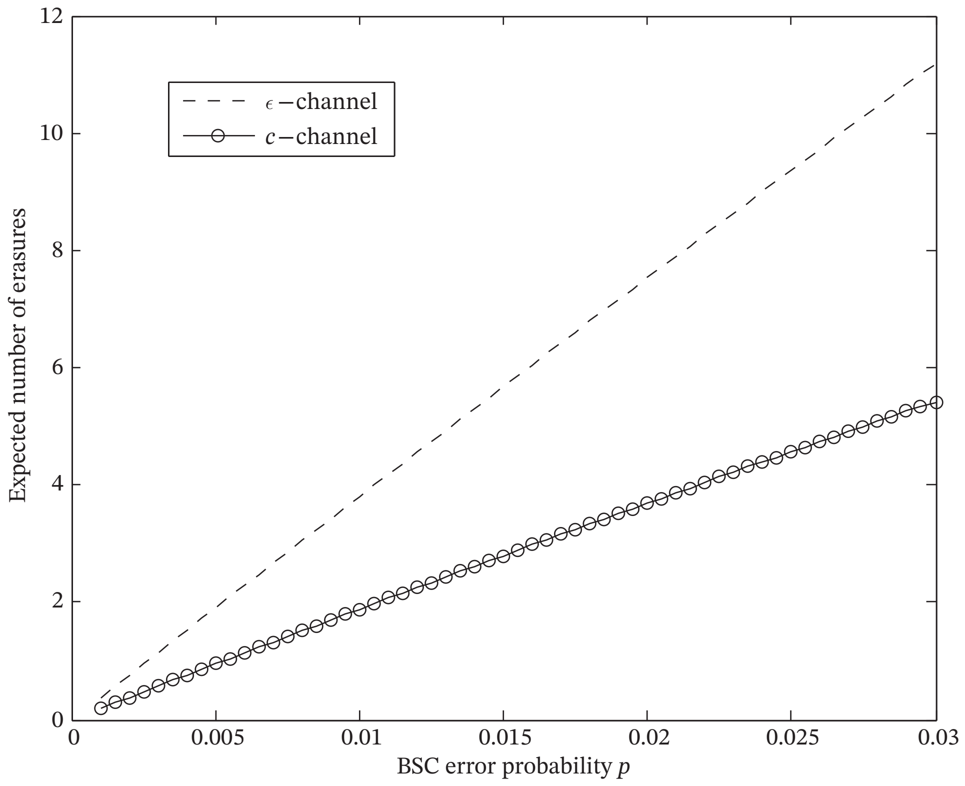


Figure 7.3 Expected number of erased symbols for the ϵ-channel and the c-channel. The number of bits is $l = 128$, which corresponds to 128 uses of the ϵ-channel and 64 uses of the c-channel.

We take now the idea to a discrete (digital) channel that is not the most general one, but it is sufficiently more general than a BSC to show how coding ideas lead to reliable communication. Let the channel between Xia and Yoshi have S input symbols and S output symbols, identical to the input ones. An *uncoded* transmission over that channel is done by taking $\log_2 S$ data bits at a time, mapping them to one of the S symbols and transmitting them. In this way, during the l uses of the channel, the total number of transmitted bits is $l \log_2 S$. The input symbols that Xia sends over l channel uses with a l-dimensional vector are

$$\mathbf{x} = (x_1, x_2, \dots x_l) \tag{7.17}$$

where each x_i is one of the S possible input symbols. In uncoded transmission, any x_i can take any of the S values, such that the total number of possible vectors that can be sent over l channel uses is S^l. In the absence of errors, this is the best possible communication strategy, as in each channel use the transmitter is *multiplexing* the maximal possible number of $\log_2 S$ bits.

However, if there are errors, then the reliability of each of the S^l possible transmitted vectors decreases, in a sense that it can be confused with another vector at the receiver. Figure 7.4(a) depicts the situation in which a symbol transmitted in a single channel use is received correctly with probability $p_1 < 1$ and incorrectly with probability $(1 - p_1)$. Specifically, when the symbol is not correctly received, the receiver Yoshi gets instead any

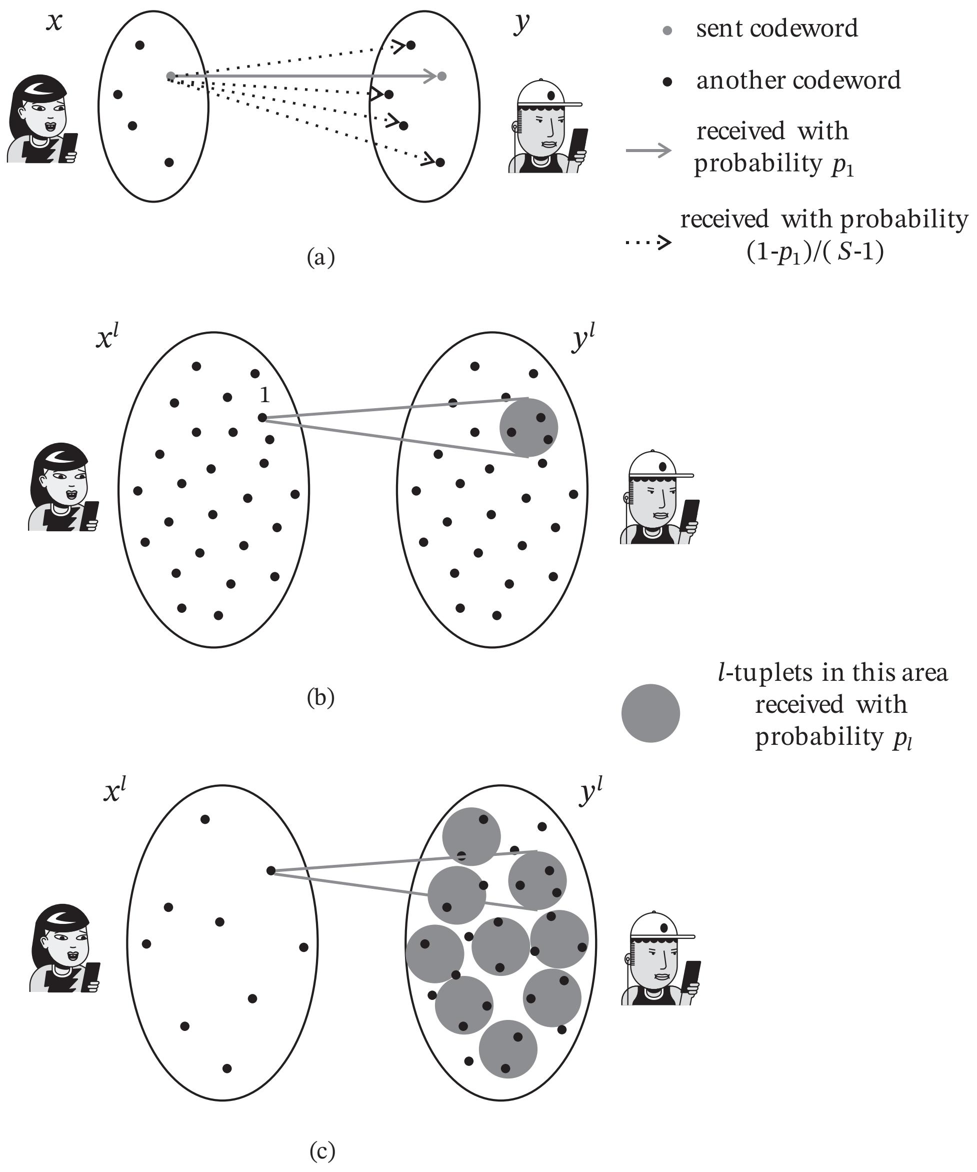

x
y
sent codeword
another codeword
received with probability p_1
received with probability $(1\text{-}p_1)/(S\text{-}1)$
(a)
x^l
y^l
1
(b)
l-tuplets in this area received with probability p_l
x^l
y^l
(c)

Figure 7.4 The main plot in error correction coding. (a) The original communication channel between Xia and Yoshi with S inputs and identical set of S outputs. (b) Probabilistic outcome upon transmission of a given l-tuplet over l uses of the original channel. (c) New channel between Xia and Yoshi created by selection of l-tuplets that are codewords and have non-overlapping shaded areas (where the received signal will lie with high probability).

of the remaining $(S-1)$ symbols. The easiest channel model to think of is the one where the probability that a particular incorrect symbol is received is $\frac{1-p_1}{S-1}$. Hence, considering only a single channel use does not leave us with many options to seek more reliable transmission.

Now let us consider Figure 7.4(b), which is a new communication channel, created by using the original channel l times. Let us call the original channel a 1-channel and the one obtained by l channel uses the l-channel. The l-channel has S^l possible inputs and S^l possible outputs. However, one important thing is different. For many types of 1-channels, the derived l-channel does not have the property that the correct symbol is received with probability q, while *any* of the incorrect symbols is received with equal probability $\frac{1-q}{S^l-1}$. Instead of that, when a symbol over the l-channel is sent, there is a subset of outputs (l-tuplets) that are more likely to be received compared to the rest of the possible outputs. Referring to Figure 7.4(b), if Xia sends the l-tuplet x^l labeled as "1", then Yoshi receives an l-tuple that belongs to the shaded area with probability p_l, where $p_l > p_1$. The shaded area plays the role of a noise cloud, discussed in the previous chapters, and contains the correct l-tuple as well as a subset of l-tuples that are likely to be received. The exact shaded area changes depending on which l-tuple has been sent by Xia. For simplicity, in this discussion we assume that the shaded area has the same shape regardless of which l-tuple has been sent.

Let us assume that Xia can select M different l-tuples whose shaded areas do not overlap, as in Figure 7.4(c). Each l-tuple is called a codeword and can be understood as a *modulation symbol* that can carry $\log_2 M$ bits. Let Yoshi make the following decision rule. If he receives an l-tuple, then he looks for a codeword $\hat{x}^l$, such that the shaded area associated with $\hat{x}^l$ contains the received l-tuple. If such a codeword is not found, then Yoshi announces an erasure. The decision rule will be made more precise in the next section.

The newly created channel, after deciding the codewords and Yoshi's decision rule, has a probability of error at most $(1-p_l)$. On the other hand, in each use of this new channel Xia can send $\log_2 M$ bits and that corresponds to

$$R = \frac{\log_2 M}{l} \tag{7.18}$$

bits per single use of the original channel. How do you select M? What is the probability of error in relation to l? These and similar questions lie at the heart of information theory and will be discussed in Chapter 8. However, what is important is that for many meaningful channels, as l increases, the probability of error $(1-p_l)$ can be made arbitrarily small, while multiplexing $R > 0$ bits per single use of the original channel.

The reader might have noticed the following terminological subtlety. Namely, once we create the new channel that takes l uses of the original channel and has a defined set of inputs/outputs (that we are unwilling or unable to change), then a transmission of $\log_2 M$ bits is an *uncoded transmission* over the new channel, while it is a coded transmission over the original channel.

Finally, in our example channel with S inputs and S outputs, redundancy is reflected in the fact that for sending M different messages we are using l Sary symbols. In the absence of errors only $\log_S M$ Sary symbols are sufficient to represent all M messages and therefore the redundancy is:

$$l - \log_S M$$

channel uses.

7.2.1 Maximum Likelihood (ML) Decoding

In this section we look closely at the decision rule, loosely defined above as the "valid codeword belonging to the shaded area". There are S^l possible channel inputs to the channel expanded over l channel uses, but Xia and Yoshi have agreed that only M of them are valid codewords and they are denoted by $\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_M$. There are S^l possible outputs $\{\mathbf{y}\}$ that can be observed by Yoshi, each consisting of l outputs of the original channel. Given that a specific vector $\mathbf{y}$ has been received, Yoshi makes a decision that $\hat{\mathbf{x}}$ has been sent. Similar to the discussion in the previous chapter, related to the QPSK modulation and Gray mapping, here we would also like to decide $\hat{\mathbf{x}}$ such that it is the most likely candidate that can produce the observed $\mathbf{y}$. A decoding rule is established by specifying which codeword $\hat{\mathbf{x}}$ should be decided by Yoshi for every possible received vector $\mathbf{y}$. If each codeword is sent with equal probability of $\frac{1}{M}$, then for given $\mathbf{y}$ Yoshi needs to find $\mathbf{x}_i$ such that the following probability is maximized:

$$P(\mathbf{x}_i|\mathbf{y}) = \frac{P(\mathbf{y}|\mathbf{x}_i)P(\mathbf{x}_i)}{P(\mathbf{y})} = P(\mathbf{y}|\mathbf{x}_i)\frac{1}{MP(\mathbf{y})}. \tag{7.19}$$

Since $P(\mathbf{y})$ does not depend on the decision we associate with $\mathbf{y}$, the maximization of $P(\mathbf{x}_i|\mathbf{y})$ is equivalent to the maximization of $P(\mathbf{y}|\mathbf{x}_i)$. The latter corresponds to the likelihood of observing $\mathbf{y}$ when $\mathbf{x}_i$ is sent; thereby the term *maximum likelihood (ML) decoding*. Considering a memoryless channel, we get

$$P(\mathbf{y}|\mathbf{x}_i) = \prod_{j=1}^{l} P(y_j|x_{ij}) \tag{7.20}$$

where x_{ij} is the input in the jth channel use of the ith codeword and the probabilities $P(y_j|x_{ij})$ are specified in a single use of the original channel. Instead of maximizing directly (7.20) it is equivalent and more convenient to work with the log-likelihood function:

$$\log P(\mathbf{y}|\mathbf{x}_i) = \sum_{j=1}^{l} \log P(y_j|x_{ij}). \tag{7.21}$$

For the communication channels that are commonly used, such as a BSC, the smaller the Hamming distance between $\mathbf{y}$ and $\mathbf{x}$, the higher the value of the log-likelihood $\log P(\mathbf{y}|\mathbf{x}_i)$. Therefore we can represent the outputs $\mathbf{y}$ that are decoded into a particular $\mathbf{x}_i$ as a shaded area around $\mathbf{x}_i$ in Figure 7.4. This can be directly related to the concept of noise clouds, used in the previous chapters, where upon transmission of a symbol, it is most likely to receive a symbol that lies within a noise cloud that surrounds the transmitted symbol. In other words, the noise cloud around $\mathbf{x}_i$ contains[3] $\mathbf{y} = \mathbf{x}_i$ and all other outputs $\{\mathbf{y}\}$ for which the ML decoding rule outputs $\mathbf{x}_i$.

The major obstacle in building practical codes with error probability that goes to zero is the complexity of decoding. In order to completely specify the ML decoding rule, for each of the S^l possible received symbols we need to compute M different likelihoods and pick $\mathbf{x}$ that leads to the maximal one. Hence, the complexity grows exponentially with l. Indeed, if the output y belongs to a discrete alphabet, then the number of possible outputs is countable and finite, such that the receiver can store the decoding rule as a lookup table for each

3 There can be communication channels in which a given input does not appear as an output; however, for the meaningful channels considered here, the input symbol always appears as a valid output symbol in the absence of errors.

possible $\mathbf{y}$. This somewhat lessens the real-time computation burden, but it does not essentially decrease the need for an exponentially large search. Therefore, the main trade-off in practical coding is to find sufficiently separated codewords, while keeping the decoding complexity at a reasonable level. Coding history has witnessed many outstanding attempts in this direction, including convolutional codes, turbo codes, low-density parity check codes (LDPCs), polar codes, etc. In the next section we illustrate the coding mechanism for different examples of channels.

7.3 Linear Block Codes for the Binary Symmetric Channel

We continue to introduce ideas in error control coding in the context of a BSC. There are many different code constructions and in this section we keep the discussion on the important class of linear block codes. Another important and widely used class of codes is the one of *convolutional codes*, briefly treated in the next section in relation to the *trellis codes*.

A *block code* for a BSC gets b data bits, which constitute a *message*:

$$\mathbf{d} = (d_1, d_2 \dots d_b)$$

and for each message there is a codeword, which is an output l-tuple of binary values:

$$\mathbf{x} = (x_1, x_2, \dots x_l)$$

where $l > b$, and both d_i and x_j are binary values. We thus obtain a (l, b) code with *code rate* equal to:

$$R = \frac{b}{l} \quad \text{(bit/c.u.)} \tag{7.22}$$

where “c.u.” stands for “channel use”, which corresponds to the nominal data rate that is achieved in the absence of errors. Following the terminology from the previous sections, we can say that creating a (l, b) block code corresponds to creating a c-channel by gathering l channel uses to represent a single use of the c-channel. The obtained c-channel has 2^b channel inputs and 2^l channel outputs.

During the transmission of a particular codeword $\mathbf{x}$, there can be some channel uses in which the bit values x_j are flipped from 0 to 1 or vice versa, thereby introducing error in the received codeword. Referring to Figure 7.4(c), Xia hopes that the channel errors will not move the received codeword outside of the shaded area around $\mathbf{x}$, such that Yoshi can correctly decide the codeword that has originally been sent. Refocusing from a codeword to a bit level, the idea behind a good code is that a particular message bit d_i affects multiple coded bits in $\mathbf{x}$. In that way the same bit of information d_i experiences diversity in the transmission and arrives through multiple independent channel uses. The simplest form of diversity is repetition coding, which is fundamentally inefficient, as each transmitted symbol x_j carries information for only one information bit d_i. The art of error-correcting codes is to find smart ways to introduce diversity: each d_i should affect multiple transmitted symbols $\{x_j\}$, while each transmitted symbol x_j should be affected by multiple information bits $\{d_i\}$.

Consider again the example of a c-channel in Figure 7.1(d). The c-channel has four possible inputs and five possible outputs, since whenever an error is detected in the received

Table 7.1 Encoding rule for the simple code of rate 2/3.

Message d	Codeword x
00	000
01	011
10	101
11	110

codeword, the channel output is an erasure. Once the c-channel is specified, and assuming that the input message is specified in quaternary and equally probable symbols, in principle, we do not need to go back to the BSC and deal with bits. However, a bit is the main currency of digital communication, the original messages are specified in bits, and channel coding is about finding systematic ways to map a sequence of bits into a sequence of transmitted symbols. This will lead us to methods for creating a c-channel with 2^b inputs, where b and the code rate are arbitrary. Therefore, we can see the c-channel as a channel that applied an error-correction code that maps 2-bit messages into three transmitted binary symbols and thus has a nominal rate of $R = \frac{2}{3}$ (bit/c.u.). The *encoding* is specified through the mapping of each possible 2 bit sequence into a codeword of three binary transmitted symbols and is depicted in Table 7.1.

If there is a single bit error, Yoshi is capable of detecting it and considers a received triplet from the set $\{001, 010, 100, 111\}$ as being erased. However, Yoshi has no way of correcting a single bit error: for example, when 001 is received and ML decoding is applied, then it is equally likely that one of the following codewords has been sent $\{000, 011, 101\}$. In this simple case each noise cloud contains a single received bit triplet, which is the originally transmitted codeword, while the four received triplets that are erased do not belong to any cloud. Furthermore, if two bit errors occur in the codeword, then the error is *undetectable*. In order to be capable to correct and detect more errors, the code construction needs to have larger b and l, as will be shown later in this section.

The code described in Table 7.1 belongs to the class of *linear block codes*, where linearity is defined with respect to addition and multiplication of binary numbers, that is, within the Galois Field GF(2). These codes have a convenient representation using vectors and matrices. For this particular example, a message is a 1×2 binary vector $\mathbf{d}$, a codeword is a 1×3 binary vector $\mathbf{x}$ and the encoding process is described as:

$$\mathbf{x} = \mathbf{d} \cdot \mathbf{G} \tag{7.23}$$

where $\mathbf{G}$ is a *generator matrix* specified as follows:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{7.24}$$

Linear codes have multiple interesting properties. For example, due to linearity, the sum of two or more codewords is a valid codeword. This means that a sum of a codeword with itself is always a valid codeword. In other words, $\mathbf{x} = [000 \cdots 0]$ is always a valid codeword that is associated with the message $\mathbf{d} = [000 \cdots 0]$.

Perhaps one of the most important property of linear codes is related to the Hamming distance. The *Hamming distance spectrum* for a particular codeword $\mathbf{x}_0$ is represented by the histogram of Hamming distances between $\mathbf{x}_0$ and all the other codewords. The minimal Hamming distance tells us how far away the closest codeword is and the multiplicity of the minimal Hamming distance tells us how many of these closest codewords there are. For linear codes, the Hamming distance spectrum for each codeword is identical, and, specifically, equal to the distance spectrum of the codeword $[000 \cdots 0]$. Therefore, the noise clouds of all codewords are identical, although in the binary case and discrete space in which the codewords are placed it is hard to argue that their geometric shape is a circle cloud.

Linear codes represent a subclass of all possible codes and there are nonlinear codes that are at least as good as the linear ones. For example, by changing the mapping in Table 7.1 into

$$00 \to 011 \qquad 01 \to 000 \qquad 10 \to 101 \qquad 11 \to 110$$

we create a nonlinear code that has identical performance in terms of communication and errors, but cannot be generated using relation (7.23).

An well known example of a code that can correct at most one bit error is the Hamming code with $b = 4$ and $l = 7$, denoted as the $(7, 4)$ Hamming code. The generator matrix of this code is given by:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{7.25}$$

The Hamming code is a systematic code since the original message bits appear in the associated codeword. For the matrix $\mathbf{G}$ in (7.25) the reader can check that the first four bits of the codewords are identical to the 4 bit message. Each of the 16 possible codewords is generated by using (7.23). Conversely, given a 7-tuple of binary values, we can check whether it is a valid codeword by using the *parity-check matrix* $\mathbf{H}$, which is uniquely determined by the generator matrix $\mathbf{G}$. Namely, each codeword $\mathbf{x}$ should satisfy the following:

$$\mathbf{x} \cdot \mathbf{H}^T = \mathbf{0} \tag{7.26}$$

where H^T is a transposed parity-check matrix and the result $\mathbf{0}$ is a row vector of size 3. The parity-check matrix that corresponds to the generator matrix (7.25) is:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{7.27}$$

The parity-check property (7.26) provides a tool for the receiver for checking if errors have occurred. A convenient way to represent the received 7-tuple of binary values $\mathbf{y}$ is:

$$\mathbf{y} = \mathbf{x} \oplus \mathbf{e} \tag{7.28}$$

where $\oplus$ is a binary addition, for each vector component, and the binary 7-tuple is termed *error vector*. If the channel is BSC with probability of error p, then each component of $\mathbf{e}$ takes the value 0 with probability $(1-p)$ and 1 with probability p. Upon receiving $\mathbf{y}$, Yoshi applies (7.26) and gets:

$$\mathbf{s} = \mathbf{y} \cdot \mathbf{H}^T = (\mathbf{x} \oplus \mathbf{e}) \cdot \mathbf{H}^T = \mathbf{e} \cdot \mathbf{H}^T \tag{7.29}$$

where $\mathbf{s}$ is a *syndrome vector*. In (7.29) we have used the linearity and the party-check property (7.26) in order to arrive at the fact that the value of the syndrome vector depends only on the error vector. If $\mathbf{s} \neq \mathbf{0}$, then this is an indication that an error has occurred, that is, at least one of the components of $\mathbf{e}$ is not 0.

The objective of Yoshi is to find the correct transmitted vector $\mathbf{x}_c$, or more precisely, the message $\mathbf{d}_c$ that produced $\mathbf{d}_c$. However, $\mathbf{s}$ does not carry any information about the transmitted vector, but only about the error vector. This means that any of the 2^b possible messages could have been sent, since for a given vector $\mathbf{x}$, we can always find an error vector that will produce a given syndrome vector $\mathbf{s}$. Hence, there are 2^b possible error vectors that result in $\mathbf{s}$. Among all the possible error vectors, Yoshi needs to select the one that has the highest likelihood of occurrence, which in the case that the error probability in the BSC is $p < 0.5$, corresponds to the error vector that has the lowest Hamming weight. In other words, Yoshi selects the error vector that is closest to *some codeword* $\hat{\mathbf{x}}$ in terms of Hamming distance and then decides that Xia has transmitted $\hat{\mathbf{x}}$.

The $(7, 4)$ Hamming code is capable of correctly decoding the transmitted message when the received binary vector has up to one bit error. That means that the noise cloud around a valid codeword contains in total 8 binary vectors: the codeword itself (Hamming distance 0) and 7 binary vectors that correspond to the 7 possible error vectors of Hamming weight 1. Since there are 16 codewords and each of them has a noise cloud of size 8, the total number of binary vectors of size 7 that are covered by the noise clouds is $16 \cdot 8 = 128$. On the other hand, this corresponds to the number of all possible binary vectors of size 7. It can be concluded that any possible received vector is within a Hamming distance of one from exactly one codeword and there are no received vectors that can, with equal likelihood, be associated with two or more codewords, as it was the case for the simple $(3, 2)$ code from Table 7.1. This is a very interesting feature of the Hamming code, which brings it into the class of *perfect codes*; the only other code that belongs to this class, besides the Hamming codes, is the $(23, 12)$ Golay code.

The decoding problem in linear block codes is equivalent to a search for an error vector with minimal Hamming weight and there is a long track of research works aimed at reducing the complexity of that type of search. Hamming codes represent only one possible class of linear bock codes. A detailed discussion on other possible constructions is out of scope for this book; we only remark here that another important class of codes are the *cyclic block codes*, whose members are the cyclic redundancy check (CRC) codes, extensively used for error detection. We have indicated in the previous chapter, Section 6.5.2, that no error detection can be perfect and this can be further clarified by the discussion on the linear block codes and the error vectors. Since the error vector of size l can get any of the 2^l possible values with non-zero probability, it follows that there is a non-zero probability that the error vector shifts a transmitted codeword into the noise cloud of another codeword, thus resulting in an undetected error.

7.4 Coded Modulation as a Layered Subsystem

We shift our focus from BSC to Gaussian channels. A codeword $\mathbf{x}$ of size l in a Gaussian channel is represented by (7.22); however, now each x_i is a complex number rather than a binary value. We recall from Section 6.2.2 that in Gaussian channels there is a restriction imposed by the average power P of the transmitted symbols such that a single transmitted symbol x_i can have an arbitrary amplitude, as long as the average power of all transmitted symbols in a given codeword satisfies the power constraint. Let us call such an approach *analog coding*, to emphasize the fact that each message $\mathbf{d}$ is directly mapped to a codeword with analog values $\mathbf{x}$. In other words, analog coding combines error control coding and modulation in a single operation. In the next chapter we show that analog coding, focused on the full codewords rather than the individual symbols in the codeword, is essential for achieving the highest possible rate for a given Gaussian channel.

In Chapter 5 we introduced the idea of adaptive modulation. This can be generalized to the concept of *adaptive modulation and coding (AMC)* with the same objective: use a higher rate, measured in bits per symbol, when the SNR increases. In order to illustrate this, recall the uncoded adaptive modulation, in which a BPSK transmission sends 1 data bit per symbol and a QPSK transmission sends 2 data bits per symbol. Assume that a rate-1/2 FEC code is available, such that each data bit produces two coded bits. The combination of this code and QPSK modulation results in transmission of 1 data bit per symbol, exactly the same nominal goodput as in uncoded BPSK. However, the reliability of coded QPSK transmission is usually superior to uncoded BPSK transmission. The same nominal rate can be achieved by using a rate-1/4 FEC code and 16-QAM modulation. The important point is that the combination of modulation and coding schemes lead to a larger number of transmission modes to select from and thus a better granularity for adaptation, which would result in smaller “stairs” in Figure 5.8.

In order to design a transmission mode with FEC coding and a specific modulation, we can apply a layered approach for the Gaussian channel, which means that the coding and modulation are designed separately. Refer to Section 6.5.1 for the concept of layering applied to communication channels. In order to avoid confusion, we introduce the following notation. The original message $\mathbf{d}$ of b bits is mapped to a binary codeword

$$\mathbf{x}_d = (x_{d_1}, x_{d_2}, \cdots x_{d_l}) \tag{7.30}$$

which is further mapped onto another codeword of u complex symbols:

$$\mathbf{s} = (s_1, s_2, \cdots s_u). \tag{7.31}$$

In the most common approach, the constellation used for modulation is fixed, such that each s_i gets a value from a set of M predefined complex values and the average power of each s_i is P. In the usual case, M is of a form $M = 2^m$, such that l and u satisfy the following relation:

$$l = m \cdot u. \tag{7.32}$$

The mapping of the original message $\mathbf{d}$ to the transmitted codeword $\mathbf{s}$ can be seen as a *concatenation* of two codes. This is illustrated in Figure 7.5, where the output of code 1 $\mathbf{x}_d$ acts as an input to code 2. Code 1 is a binary code with a rate $R_d = \frac{b}{l}$, as it gets b message

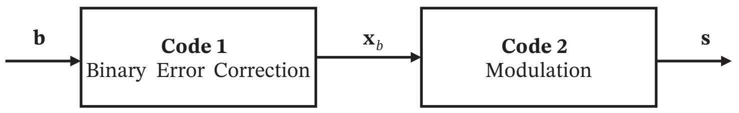


Figure 7.5 Representation of coding and modulation as a concatenation of two codes.

bits and outputs l binary values. Since both the input and the output are binary values, this code clearly introduces redundancy. The output of code 2 is in a form of analog values and one cannot tell easily whether redundancy has been introduced or not. If (7.32) is satisfied, then a possible implementation of code 2 is to take a group of m bits from $\mathbf{x}_d$ and map them to an Mary output symbol using, for example, Gray code. Hence, looking only at code 2, its nominal transmission rate is $m = \frac{l}{u}$ bits per symbol. The overall transmission rate of the concatenated code is

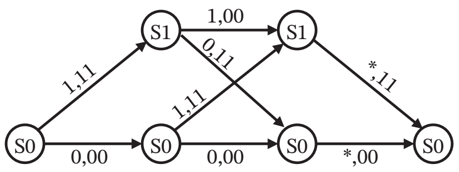
$$R = R_d \cdot m = \frac{b}{l} \cdot \frac{l}{u} = \frac{b}{u} \quad \text{(bit/c.u.)}. \tag{7.33}$$

Figure 7.5 indicates a clear separation between code 1 and code 2, representing coding and modulation, respectively. This is due to the fact that, for example, the Gray coding of code 2 is done independently from the error correction method used in code 1. Having this perspective, it comes natural to ask: what can be attained by not separating code 1 and code 2 and designing them jointly through some form of cross-layer optimization?

This question has motivated *trellis coded modulation*, a shining example of technology breakthroughs in communication engineering. In order to understand the basic idea of it, let us at first look at the concept of a trellis code. Figure 7.6 shows an example of a simple trellis code that encodes 2 bit messages into codewords of length $l = 6$, thus leading to a code rate of $R = \frac{1}{3}$. The codewords of a trellis code are created by a finite-state machine and in this example there are two states, S0 and S1. At the start of the codeword generation, the encoder is in state S0. Given the 2 bit input message $\mathbf{d} = (d_1, d_2)$, the codeword $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$ is generated in the following way. In the first step, Xia takes the first message bit d_1; if $d_1 = 0$ then she sends $(x_1, x_2) = (0, 0)$ and transits to state S0, otherwise, if $d_1 = 1$ she sends $(x_1, x_2) = (1, 1)$ and transits to state S1. The next steps proceed analogously, following the structure of the *trellis graph* in Figure 7.6. Each branch of the graph is labeled by $d_i, x_j x_{j+1}$ and specifies the output binary values that are generated by the jth message bit. If the message bit is set to $*$, then the transition is made with a dummy message bit, which in this example is the "third" message bit.

This is a very simple example of a trellis code and it can be generalized in many ways. For example, the number of branches going out of a state is not necessarily two, as the encoder can take b message bits in a single step and produce l output binary values of the codeword. In that case, each branch is labeled $(d_1 d_2 \cdots d_b, x_1 x_2 \cdots x_l)$. The concept of trellis

Figure 7.6 Example of a trellis code with two states. On the left is the trellis diagram and on the right the code of rate $R = \frac{1}{3}$ produced by the trellis.



coding was originally introduced to describe *convolutional codes*, where the state transitions are implemented by shift registers; however, the trellis can also be used to represent other codes, such as linear block codes. The trellis representation has had widespread usage due to the fact that the decoder can use the seminal Viterbi algorithm for maximum likelihood decoding.

The strength of a trellis code is tightly related to the number of states used. The idea is that, the more states there are, the larger number of output bits become affected by a specific input bit, which is in line with the principles of good code design stated earlier in this chapter. Furthermore, multiple states represent the key to the joint design of coding and modulation in trellis-coded modulation. In order to see this, assume that 8-PSK modulation is used and the symbols are $s_1, s_2, \dots s_8$. Let us assume that there are two data bit sequences $\mathbf{d}_1$ and $\mathbf{d}_2$ that act as inputs to code 1 from Figure 7.5. Let us denote the respective outputs of code 1 as follows:

$$\mathbf{d}_1 \mapsto \mathbf{c}_{11}, 000001, \mathbf{c}_{12}$$
$$\mathbf{d}_2 \mapsto \mathbf{c}_{21}, 000001, \mathbf{c}_{22}$$

where $\mathbf{c}_{ij}$ represent sequences of coded bits associated with the data bits $\mathbf{d}_i$. For the present discussion the actual content of the coded bit sequences $\mathbf{c}_{ij}$ is irrelevant; it is only important that the coded bits of both $\mathbf{d}_1$ and $\mathbf{d}_2$ contain the subsequence 000001.

Let us at first look at the case in which coding and modulation are designed separately. In that case, the mapping of the bit triplets from the code output onto the symbols is fixed: for example, 000 is always mapped to s_1, 001 is always mapped to s_2, etc. Therefore, the mapping of the data bits into modulated symbols looks as follows:

$$\mathbf{d}_1 \mapsto \mathbf{s}_{11}, s_1, s_2, \mathbf{s}_{12}$$
$$\mathbf{d}_2 \mapsto \mathbf{s}_{21}, s_1, s_2, \mathbf{s}_{22}$$

where $\mathbf{s}_{ij}$ represent the other modulation symbols associated with $\mathbf{d}_i$.

In contrast, in trellis coding the system has different states and the mapping of a coded bit sequence onto a modulation symbol depends on the state the system. For example, there can be a trellis-coded modulation where the mappings are made as follows:

$$\mathbf{d}_1 \mapsto \mathbf{t}_{11}, s_1, s_2, \mathbf{t}_{12}$$
$$\mathbf{d}_2 \mapsto \mathbf{t}_{21}, s_3, s_4, \mathbf{t}_{22}.$$

This means that the state the trellis-coded system is in at the step when the modulation sequence $\mathbf{t}_{11}$ has been produced is different from the state the system is in at the step at which the modulation sequence $\mathbf{t}_{21}$ has been produced. Hence, 000001 results in two different pairs of transmitted symbols, s_1, s_2 and s_3, s_4, respectively.

There are certainly other coding methods in which 000001 can result in different modulation symbols, but the important advantage of trellis-coded modulation is that its structure preserves the possibility of using the efficient Viterbi algorithm for decoding.

7.5 Retransmission as a Supplement to Coding

The methods for channel coding or FEC that emerge from the basic model of a communication system by Shannon, depicted in Figure 6.1 do not consider the possibility of feedback from Yoshi to Xia. By contrast, in practice, Xia can believe that Yoshi has received the data packet correctly if she gets an acknowledgement from him. In Chapter 4, when discussing the reliable transmission service provided by a lower layer to the layer above, we have assumed that the lower layer ensures successful transmission of an acknowledgement. In this section we discuss concepts and ideas that Xia can use for retransmission upon receiving feedback from Yoshi.

At this point we are completing the story of the cartoon from the beginning of this chapter. At first Zoya notices that one of the digits is erroneous and her capability for pattern recognition, combined with previous experience, plays the role of FEC decoding. The credit card number has an embedded error checking capability, which helps to detect that the digit that Zoya "decodes" from the piece of paper is incorrect. Finally, after not succeeding to decode the correct credit number by only using FEC, Zoya asks her dad to tell her the correct digit. The call placed to her dad can be understood as a feedback that asks for retransmission. Here it should be noted that Zoya can ask her father to read again all the digits of the credit card (inefficient) or only the digit that was unreadable (efficient).

Nevertheless, receiving an ACK (acknowledgement) from Yoshi is not a guarantee to Xia that he has received the packet correctly. The non-ideality of the error checking code can lead to error patterns that can trick Yoshi into believing that he has received a packet correctly, while in reality he is acknowledging the reception of a wrong packet. However, if receiving an ACK is not sufficient, is there a way for Yoshi to send richer feedback to Xia and guarantee the reception of the correct packet?

To answer this question, assume that Xia sends the data packet $\mathbf{d_1}$. Yoshi decodes the packet

$$\mathbf{d}_2 = \mathbf{d_1} \oplus \mathbf{e} \tag{7.34}$$

where $\mathbf{e}$ is the bit error pattern and $\oplus$ is a binary addition, as in (7.28). Since $\mathbf{e}$ is a product of a random process, assume that it happens to be such that $\mathbf{d}_2$ satisfies the error check and Yoshi believes that this is the correct packet. Now, instead of sending ACK, Yoshi decides to use (very) rich feedback and transmit the whole packet $\mathbf{d}_2$ to Xia. To do that, Yoshi may also use some form of FEC, which does not need to be the same FEC used by Xia to send $\mathbf{d_1}$. Now the random error process happens to produce the same error pattern $\mathbf{e}$. Note that this event can occur, although with very low probability. Then Xia receives:

$$\mathbf{d}_2 \oplus \mathbf{e} = \mathbf{d_1} \oplus \mathbf{e} \oplus \mathbf{e} = \mathbf{d_1} \tag{7.35}$$

since $\mathbf{e} \oplus \mathbf{e} = \mathbf{0}$, which is an all-zero vector. Clearly, $\mathbf{d_1}$ satisfies the error check and Xia believes that this is the packet received and decoded by Yoshi. However, the actual packet decoded and accepted as correct by Yoshi is an incorrect one.

The previous result shows the theoretical impossibility of perfectly reliable communication within finite time; the next chapter will present the conditions under which communication can become perfectly reliable in an asymptotic regime. To facilitate the discussion, for the rest of this section we will neglect the imperfectness of the error check and assume that ACK always represents the correct packet.

7.5.1 Full Packet Retransmission

The most elementary way for Xia to react upon not receiving an ACK from Yoshi is to retransmit the whole packet again. Even when Xia does not use any form of FEC, retransmission will eventually lead to reliable transmission, provided that the error check is perfect. If Xia does use FEC for sending the original packet, then the retransmission protocol is called *hybrid ARQ (HARQ)*. In the simplest HARQ version, Yoshi discards all versions of the packet that were not decoded correctly and waits until he gets a packets that is decoded correctly. An improved version is the one in which Yoshi does not waste the erroneously received copies of the packets and instead combines them with the present version of the packet in order to improve the decoding reliability.

To see how this works, let $\mathbf{s} = (s_1, s_2, \dots s_u)$ represent the packet sent by Xia through the respective baseband symbols. Yoshi does not receive the first packet transmission correctly, Xia does not receive an ACK and she retransmits the same $\mathbf{s}$. Let us look at the same received symbol from both packet transmissions:

$$\begin{aligned} y_{i,1} &= hs_i + n_{i,1} \\ y_{i,2} &= hs_i + n_{i,2} \end{aligned} \tag{7.36}$$

where the index j stands for the jth packet transmission; $y_{i,j}$ is the ith symbol received by Yoshi and $n_{i,j}$ is the ith noise sample. It is assumed that the channel h stays constant during the transmission and the retransmission. Using Chase combining or maximum ratio combining (MRC), Yoshi creates:

$$y_i = y_{i,1} + y_{i,2} = 2hs_i + n_{i,1} + n_{i,2}. \tag{7.37}$$

If the noise samples are independent, then MRC makes the SNR of y_i be double the original SNR under which the data is attempted to be decoded from the individual $y_{i,j}$. Assuming that the feedback from Yoshi is ideal and instantaneous, then L retransmissions will increase the overall transmission time of the packet L times, which decreases the nominal throughput L times. On the other hand, using MRC will increase the decoding SNR L times. These are two opposing mechanisms that are well-sublimed in Shannon's expression for the capacity of a Gaussian channel (see the next chapter).

This is illustrated in Figure 7.7(a), where Xia sends the same packet $\mathbf{s}_1$. The jth transmitted version of the packet is denoted as $\mathbf{S}_{1,j}$. After the first transmission, Yoshi sends NACK. However, Yoshi is not able to detect the packet $\mathbf{S}_{1,2}$, therefore the short reception time at the beginning of the packet. Hence, he is not in a position to send NACK and Xia sends the version $\mathbf{S}_{1,3}$ due to the absence of NACK. Nevertheless, the fact that Yoshi has not detected $\mathbf{S}_{1,2}$ has the consequence that he is not able to combine all three transmitted versions of the packet, that is, the transmission of $\mathbf{S}_{1,2}$ is wasted. This figure also illustrates the fact that retransmissions effectively decrease the throughput by consuming more time to deliver the same portion of data.

As a final remark, there is one statistical subtlety related to the Chase combining in (7.37). Namely, the statistical nature of $n_{i,1}$ and $n_{i,2}$ is different. This is because, after retransmission is requested, then Yoshi knows something about $n_{i,1}$, since the set of all noise instances in the first transmission have been such that they have resulted in an erroneous reception. This creates a dependence among $n_{1,1}, n_{2,1}, n_{3,1}, \dots$, while $\{n_{i,2}\}$ remain independent from each other, as well as from the noise instances from the first transmission. As a result, strictly speaking, the SNR of the combined signal is not doubled, but has a more involved statistical relationship with the first and the second transmissions.

7.5.2 Partial Retransmission and Incremental Redundancy

Instead of retransmitting the same $\mathbf{S}_{1,1} = \mathbf{S}_{1,2} = \mathbf{S}_{1,3} = \mathbf{s}$, Xia can retransmit another set of symbols $\mathbf{R}_{1,2}$ for the first retransmission, $\mathbf{R}_{1,3}$ for the second retransmission, etc. This is depicted in Figure 7.7(b), where it is seen that the retransmission of smaller set of symbols results in a throughput improvement, provided, of course, that the smaller packets $\mathbf{R}_{1,2}, \mathbf{R}_{1,3}$ do not deteriorate the decoding reliability. If one looks at the entirety of $\mathbf{S}_{1,1}, \mathbf{R}_{1,2}, \mathbf{R}_{1,3}, \ldots$, then $\mathbf{R}_{1,2}, \mathbf{R}_{1,3}, \ldots$ can be seen as elements contributing to an additional error correction capability. However, the redundancy of $\mathbf{R}_{1,2}, \mathbf{R}_{1,3}, \ldots$ is not introduced in the first transmission, but upon feedback from the receiver. This is why this method of retransmission is called *incremental redundancy*.

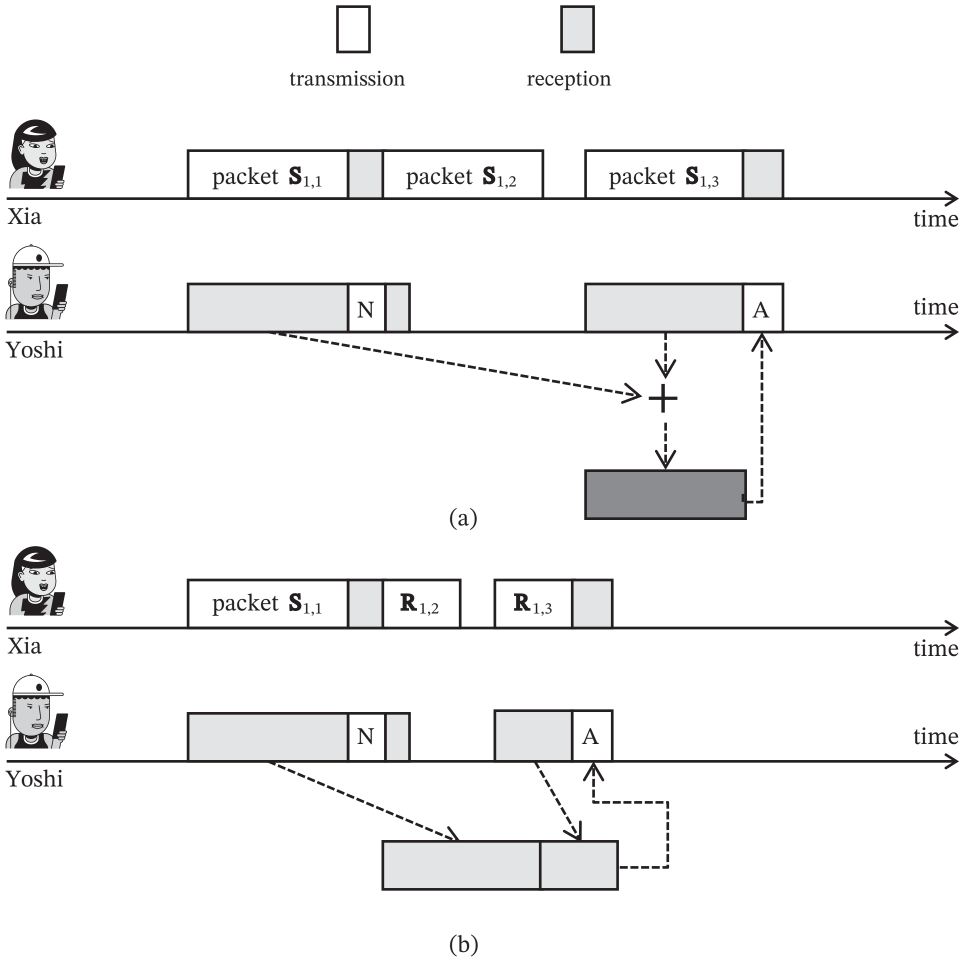

transmission
reception
packet $\mathbf{S}_{1,1}$
packet $\mathbf{S}_{1,2}$
packet $\mathbf{S}_{1,3}$
Xia
time
N
A
time
Yoshi
(a)
packet $\mathbf{S}_{1,1}$
$\mathbf{R}_{1,2}$
$\mathbf{R}_{1,3}$
Xia
time
N
A
time
Yoshi
(b)

Figure 7.7 Comparison of full and partial retransmission. In both cases the packet of the first retransmission is not detected by Yoshi and therefore not used in the coding process. (a) Full retransmission. (b) Partial retransmission and incremental redundancy.

Incremental redundancy utilizes the fact that a full retransmission may introduce excessive redundancy, in a sense that Yoshi could have been able to decode the data with less information received through the retransmission. This is why it can be possible to improve the throughput as in Figure 7.7(b) without decreasing the reliability. The ideal redundancy should be just sufficient to supplement the previously received version of the packet and thus enable Yoshi to decode the data packet correctly.

In principle, this could be achieved by rich feedback instead of a simple NACK: after failing to decode the packet, Yoshi sends feedback to Xia and this feedback contains information "I am missing b' bits of information to be able to decode". The key phrase is "in principle", as there is a major difficulty in finding coding/decoding methods that enable Yoshi to measure how much information he is missing to decode the packet correctly. Despite the potential of the rich feedback and the suitability of some of the modern codes to quantify the amount of information required to decode the packet, the norm in wireless communication is the use of a simple binary ACK/NACK feedback.

The simplest way to use NACK for incremental redundancy is to retransmit only a subset of the symbols $\mathbf{S}_{1,1} = \mathbf{s}$ sent in the original transmission. For example, Xia retransmits s_1, s_2, s_3 only and during the retransmission, Yoshi receives $y_{i,2}$ for $i = 1, 2, 3$, as given by (7.36). Then one option is that Yoshi replaces $y_{i,1}$ from the previously received packet with the respective $y_{i,2}$ for $i = 1, 2, 3$, while he reuses the remaining $y_{i,1}$ for $i > 3$ and attempts to decode the packet again. Another option is that Yoshi attempts partial MRC: he replaces $y_{i,1}$ with $y_{i,1} + y_{i,2}$ for $i = 1, 2, 3$, retains $y_{i,1}$ for $i > 3$ and attempts to decode the packet. The problem with this type of retransmission is that all symbols of $\mathbf{S}_{1,1} = \mathbf{s}$ are statistically identical and there is no special reason to select s_1, s_2, s_3 upon reception of NACK. In other words, the retransmission of s_1, s_2, s_3 is a guess; in a similar way, Xia could have tried a guess by retransmitting, for example, s_2, s_5, s_7, s_9. More generally, Xia could decide to select $l_1 < l$ symbols for retransmission, but Yoshi should either know the positions of those l_1 symbols in advance or Xia should inform him about them in the packet. The latter would represent an additional overhead.

A more general approach would be the one following the very definition of incremental redundancy; Xia applies a FEC code to an input message $\mathbf{d}$ and obtains the codeword $\mathbf{c}$, but does not transmit the whole codeword. Let us assume that the codeword consists of three parts $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3)$. In the first transmission, Xia modulates $\mathbf{c}_1$ and transmits it. After getting NACK, she modulates $\mathbf{c}_2$ and transmits it, and, after the second NACK, she transmits $\mathbf{c}_3$. A common way for doing this when a linear code is used is based on *puncturing*, by which some of the parity bits generated by the code are omitted in the first transmission.

Another generalization step can be seen in the adaptation of the transmit power. For example, Xia can start optimistically, by transmitting with a lower power and, upon receiving NACK, retransmit with a higher power. This is of interest in applications with a constraint on latency. By starting with a low power and gradually increasing it in the retransmissions, Xia keeps the average power acceptable, while investing a significant power as the deadline is approaching; this is what most of us do with project deadlines as well.

Incremental redundancy plays also a role in AMC. Given a set of transmission modes and their associated "stairs" in Figure 5.8, incremental redundancy can act as a bridge between different stairs. In fact, the use of feedback and retransmissions can be interpreted as a method that helps the transmitter to find out the proper combination of modulation and coding that is suitable for the current SNR of the channel.

7.6 Chapter Summary

This chapter has introduced the concepts of error control coding or FEC. The main underlying idea is to use the channel multiple times and thus define a new channel, call it a super-channel, with larger possible number of inputs and outputs. The coding consists of restricting the subset of inputs that are allowed to be used over the super-channel. Those allowed inputs are called codewords. The fact that not all possible inputs can be used over the super-channel can be interpreted as a source of redundancy. The good coding methods are selecting the inputs of the super-channel in such a way as to enable the receiver to be able to correctly guess, with high probability, the actually transmitted channel input. We have also introduced the relation between the coding and modulation and exemplified the cross-layer design principle through trellis-coded modulation. Finally, we discussed the role of feedback and retransmission as a way to improve the communication reliability.

7.7 Further Reading

Error control coding has been one of the areas of major intellectual and technological advances in communication engineering. There are a number of textbooks dedicated to it, such as Shu and Costello [2004] and Richardson and Urbanke [2008], but also the more general book on digital communications Proakis and Salehi [2008]. Two interesting reads by the original authors who made giant intellectual leaps in coding theory are Ungerboeck [1987] in trellis-coded modulation and, more recently, Arıkan [2009] in polar codes. Regarding feedback and retransmissions, a revisionist view on HARQ with a richer feedback is presented in Trillingsgaard and Popovski [2017].

7.8 Problems and Reflections

1. *Machine learning for the coding problem*. The problem of an exhaustive search for good codes of a given length is quickly becoming infeasible due to the vast space of codebooks that should be searched. Try to devise a procedure for searching for good codebooks of a given rate and given (short) length for a BSC. You can use some of the advanced tools from machine learning and algorithms for searching large data sets.
2. *Multiple error checks*. The common way of carrying block coding is the one in which the sender Xia takes a message of b bits, adds c check bits for error detection and then produces an FEC codeword. After FEC decoding, the receiver Yoshi has only one possibility to check, that is, he will either accept all $b + c$ bits as correct or none. As an intermediate step, Xia can add multiple error checks to the original b bit messages. Analyze the costs and the benefits from introducing multiple error checks and make examples for coding/decoding.

3. *Rich feedback with error checks*. The approach from the previous problem can be changed as follows. Xia sends a message $\mathbf{d}$ that consists of b bits and c check bits. Assume for the moment that Xia does not use any FEC coding. Yoshi receives the $b + c$ bits and runs an error check. If it is correct, he sends an ACK. If it is not correct, Yoshi sends a NACK plus an additional feedback created as follows. Yoshi computes error check bits for the first half of the received packet $\mathbf{d}'$, that is, for the first $\frac{b+c}{2}$ bits, and obtains the checksum $\mathbf{c}_1'$. Then he computes the checksum for the second half of the bits and obtains $\mathbf{c}_2'$. Yoshi sends $\mathbf{c}_1', \mathbf{c}_2'$ as feedback to Xia. On her side, Xia knows $\mathbf{d}$ and she can compute $\mathbf{c}_1, \mathbf{c}_2$ for the correct data as well as the checksum. If the error in the packet to Yoshi were only the first half of the packet, then Xia would find out that $\mathbf{c}_1' \neq \mathbf{c}_1$ and would retransmit only the first half of $\mathbf{d}$.
   (a) Discuss the pros and cons of this scheme.
   (b) Generalize the scheme to more than two checksums sent from Yoshi.
   (c) Generalize the scheme to work with a FEC code and discuss the trade-offs.
4. *Interleaving*. An interleaver is a system element that often appears in coding schemes and its role is to permute (shuffle) the message bits at the transmitter side, while the suitable de-interleaver brings them back to the original order. Investigate and compare the use of interleaver in error control coding in three different contexts:
   (a) In fading channels.
   (b) In turbo codes.
   (c) In bit-interleaved coded modulation (BICM).
5. *Superposition coding and retransmission*. In Chapter 5 we introduced the idea of superposition coding. Assume that Zoya transmits data to Yoshi by splitting her power into multiple superposed packets. Design retransmission schemes that utilize the structure of the transmission. *Hint*: Recall that the decoding and canceling of one of the packets makes it easier to decode another of the superposed packets.