Practice Set P2b (Reliability) 2ITX0 2020–2021 Q2

The exercises in this set allow you to practice with the course material. They are particularly suitable to familiarize yourself with the definitions concerning *reliable communication*.

Copy with hints. First, try to solve the problems without hints. Write down your answers. That is what you need to do on the exam as well. Practice this now.

Unless explicitly requested otherwise, answer the following questions with the bit as unit.

Exercise 1: (Effect of variable-length compression on reliability)

Consider an information source with five messages $\{ A, B, C, D, E \}$ having equal probabilities. We use the following (prefix-free) encoding C_1:

$A \to 00$
$B \to 01$
$C \to 100$
$D \to 101$
$E \to 11$

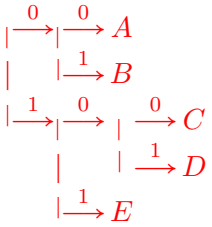
1. Check that C_1 is indeed prefix-free.
2. What is the entropy of this source?
3. What is the average number of bits per message in this code (when used to encode the given information source)?
4. What is the redundancy of this code (for the given information source)? Express the redundancy as the percentage above the entropy.
5. Using C_1, encode the message sequence $m = ABCDE\ldots$.
6. Investigate the consequences of each of the 12 possible single-bit errors in m (Do at least the errors at bit position 1 and at bit position 5.) When a bit is flipped (by noise), the receiver will be decoding a slightly different sequence, resulting in a different decoded sequence of messages. What will that sequence be for each single-bit error? Compare the decoded sequence to the original sequence m, and note extent of the difference.

Hint Look up the definitions of prefix-free and entropy of an information source.

Note that a single-bit error flips exactly one bit in the (encoded) bit sequence on the channel. Flipping amounts to adding a 1, using clock arithmetic modulo 2, where $1 + 1 = 2 = 0$.

Answer

1. No codeword is a prefix of another codeword, because they can be placed in an encoding-decoding tree. (This code was obtained by Huffman's algorithm. You are encouraged to verify this.)



2. $\mathcal{H} = -\sum_{M \in C_1} P(M) \log_2 P(M) = \log_2 5 = \log 5 / \log 2 \approx 2.32$

3.

$$\begin{aligned} \mu &= \sum_{M \in C_1} P(M)\text{length}(\text{codeword}(M)) \\ &= 0.2 \sum_{M \in C_1} \text{length}(\text{codeword}(M)) \\ &= 0.2(2+2+3+3+2) \\ &= 2.4 \end{aligned}$$

4. Observe that this is only about $100(2.4 - 2.32)/2.32 \approx 3.4\%$ above the entropy.

5. $m = ABCDE\ldots \rightarrow 000110010111\ldots$

6. There are 12 possible single-bit errors to investigate:

error	received	decoded
100000000000	$\underline{1}00\,11\,00\,101\,11\ldots$	$\underline{C}E\underline{A}DE\ldots$
010000000000	$0\underline{1}\,01\,100\,101\,11\ldots$	$\underline{B}BCDE\ldots$
001000000000	$00\,\underline{1}1\,100\,101\,11\ldots$	$A\underline{E}CDE\ldots$
000100000000	$00\,0\underline{0}\,100\,101\,11\ldots$	$A\underline{A}CDE\ldots$
000010000000	$00\,01\,\underline{0}0\,01\,01\,11\ldots$	$AB\underline{ABB}E\ldots$
000001000000	$00\,01\,1\underline{1}\,01\,01\,11\ldots$	$AB\underline{EBB}E\ldots$
000000100000	$00\,01\,10\underline{1}\,101\,11\ldots$	$AB\underline{D}DE\ldots$
000000010000	$00\,01\,100\,\underline{0}0\,11\,1\ldots$	$ABC\underline{AE\ldots}$
000000001000	$00\,01\,100\,1\underline{1}\,11\,1\ldots$	$ABC\underline{EE\ldots}$
000000000100	$00\,01\,100\,10\underline{0}\,11\ldots$	$ABC\underline{C}E\ldots$
000000000010	$00\,01\,100\,101\,\underline{0}1\ldots$	$ABCD\underline{B}\ldots$
000000000001	$00\,01\,100\,101\,1\underline{0}\ldots$	$ABCD\underline{\ldots}$

The changes from the original have been underlined.

Note that in some cases, the damage is considerable, whereas in other cases, it is quite limited. This is known as *ripple effect*. Note also that in all cases, the receiver has no indication that something went wrong!

Exercise 2: (Less compression can benefit reliability)

We turn the code C_1 of Exercise 1 into a fixed-length code C_2 by extending the shorter codewords as follows:

$A \to 000$
$B \to 010$
$C \to 100$
$D \to 101$
$E \to 110$

1. Why can a prefix-free code always be made into a fixed-length code by *arbitrarily* extending all codewords to a fixed length? That is, why will there be no problem in decoding the extended code?

2. What is the redundancy in code C_2 for this information source?

3. Fill out the following table that investigates the effect of all single-bit errors in each of the codewords of C_2. Each entry consists of $r \to M$, where r is the bit sequence received (incorporating the error), and M is the decoded message (write ? when decoding fails).

		error sequence		
message	codeword	100	010	001
A	000	$100 \to C$	$\to$	$\to$
B	010	$\to$	$\to$	$\to$
C	100	$\to$	$\to$	$\to$
D	101	$\to$	$\to$	$\to$
E	110	$\to$	$\to$	$\to$

4. Which error(s) can be detected?

5. Which error(s) can even be corrected, when assuming that the error probability is low? (As always, justify your answer.)

Hint Look up the definition of prefix-free code, and of (absolute/relative) redundancy.

Answer

1. All the extended codewords will be unique, because they could already be distinguished without the extension, since we started with a prefix-free code.

2. The efficiency of C_2 is worse than for C_1. The code has more *redundancy*. The relative redundancy is approximately $100(3 - 2.23)/2.23 \approx 35\%$ above the entropy.

3.

		error sequence		
message	codeword	100	010	001
A	000	$100 \to C$	$010 \to B$	$001 \to\ ?$
B	010	$110 \to E$	$000 \to A$	$011 \to\ ?$
C	100	$000 \to A$	$110 \to E$	$101 \to D$
D	101	$001 \to\ ?$	$111 \to\ ?$	$100 \to C$
E	110	$010 \to B$	$100 \to C$	$111 \to\ ?$

   Note that for a fixed-length code, there is no 'rippling' effect of errors.

4. A positive consequence of the higher redundancy is that now the receiver can at least detect some errors. More precisely, 5 of the 15 possible errors are detected, viz. the cases where decoding fails (indicated by ? in the table above).

5. When an error is detected, the receiver could try to make a good guess at what the error was, and then repair it.

   For example, when receiving 001, something must have gone wrong, because this is not a codeword in C_2. However, there are two single-bit errors that explain this received word (see the table above): $A+001$ and $D+100$. The receiver has no clue which one to prefer. (Moreover, there could have been more error bits. But the assumption that the bit error probability is low, makes this much less likely than a single-bit error.)

   This also holds for the non-codeword 111.

   But for the non-codeword 011 there is only one single-bit error that can explain it, viz. $011 = B + 001$. So, in this case, a 'correction' makes sense. We write 'correction' in quotes, because the receiver cannot be 100% certain that this is what happened. But given a low bit error probability, it is very likely an improvement.

Exercise 3: (Less compression can benefit reliability in different ways)
Now consider the following extension of C_1 from Exercise 1 to a fixed-length encoding C_3:

$A \to 000$
$B \to 011$
$C \to 100$
$D \to 101$
$E \to 110$

Again, investigate all single-bit errors, by answering the questions of Exercise 2.

Hint (No hint)

Answer

1. As argued in Exercise 2.1, also in this extension all codewords are unique.
2. The redundancy is the same as in Exercise 2.
3.

		error sequence		
message	codeword	100	010	001
A	000	$100 \to C$	$010 \to$?	$001 \to$?
B	011	$111 \to$?	$001 \to$?	$010 \to$?
C	100	$000 \to A$	$110 \to E$	$101 \to D$
D	101	$001 \to$?	$111 \to$?	$100 \to C$
E	110	$010 \to$?	$100 \to C$	$111 \to$?

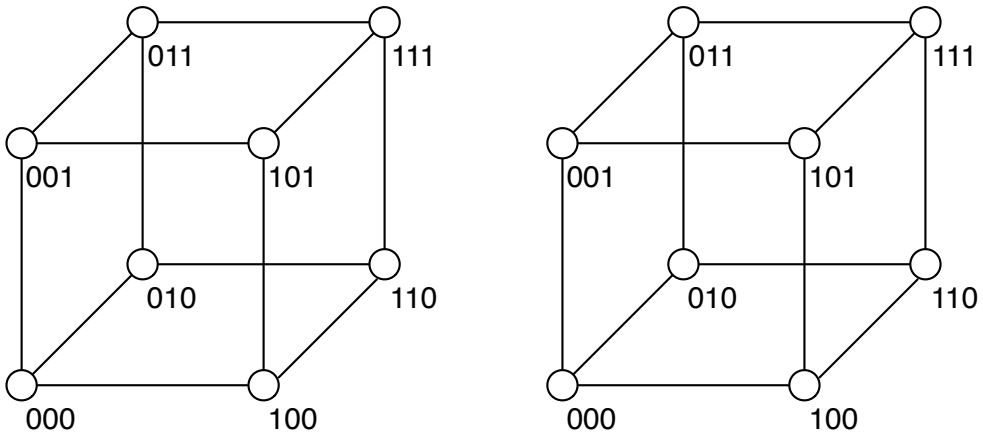
4. With code C_3, the receiver can detect 9 of the 15 single-bit errors. Note that this is a considerable improvement over code C_2 of Exercise 2.
5. Using code C_3, there are 3 non-codewords:

non-codeword	codeword neighbors	#
001	000 (A), 011 (B), 101 (D)	3
010	000 (A), 011 (B), 110 (E)	3
111	011 (B), 101 (D), 110 (E)	3

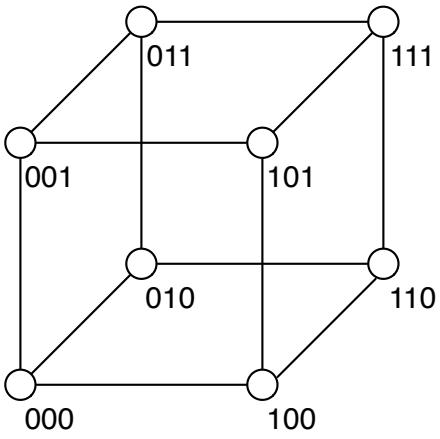
In this case, no error has an 'obvious' correction. (That is the disadvantage of better detection.)

Exercise 4: (Visualize codes as point clouds in a discrete space)

To understand the error-detecting and error-correcting abilities of codes, it helps to visualize codes as a 'point' cloud in some discrete 'space'. Each possible binary block of a given length (in Exercise 2 and 3, that length is 3) is considered a point in that space. The code is then a subset (cloud) of points in that space. Here is the space for block length 3:



1. In the figure above (on the left), mark the codewords of code C_2 from Exercise 2, by filling their corresponding circles.
2. Do the same for the codewords of code C_3 from Exercise 3 (on the right).
3. Can you describe the 'spatial' difference in distribution of codewords in C_2 and in C_3?
4. Suppose there would be only 4 messages (equiprobable) to encode. Using 2-bit code words would be optimal. What would the redundancy be when encoding these messages with 3-bit codewords?

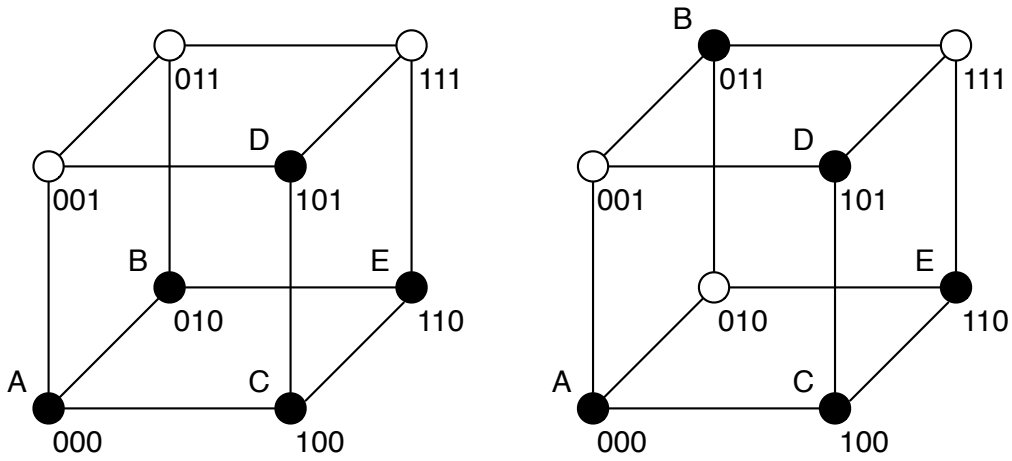


5. With only 4 messages to encode, it is possible to come up with a 'perfect' distribution of codewords in the 3-bit space. What is that distribution? Use the diagram above.
6. Can you explain why every single-bit error is detectable under that 'perfect' distribution?
7. Can any errors be corrected under that 'perfect' distribution?

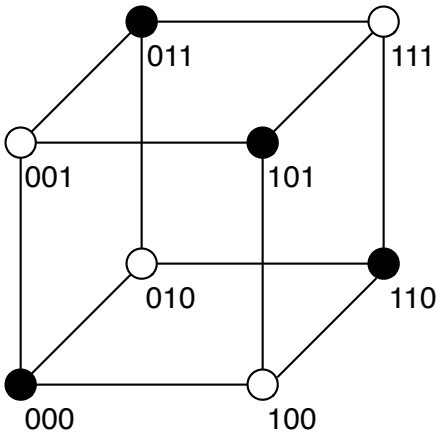
Hint (No hint)

Answer

1. See left in the figure below for the point cloud of code C_2.



2. See right in the figure above for the point cloud of code C_3.
3. The difference in distribution is now easier to 'see'. In C_2, the point cloud is less evenly distributed than in C_3
4. With 5 points in the 8-point space, it is too crowded to detect all errors. Put differently, we don't have sufficient redundancy. When encoding only 4 equiprobable messages in 3-bit codewords, the redundancy becomes $100(3-2)/2 = 50\%$ above the entropy
5. . In the next figure, where the code has only 4 codewords, the distribution is 'perfect': every single-bit error is detectable.



   Note that the code consists of all words with even *parity*, that is, with an even number of 1's.
6. No codeword is adjacent to another codeword. That is, any single-bit error in a codeword will *not* lead to another codeword, but leads to a non-codeword. Hence, it is detectable.
7. This code does not admit any reasonable way of correcting errors, because any non-codeword is adjacent to multiple (in fact, 3) codewords. So, the receiver has no clue which codeword was the most likely codeword sent.

Exercise 5: (Hamming distance)

1. What is the Hamming distance between 000111 and 001011?
2. What is the Hamming distance between 000111 and 100110?
3. What is the Hamming distance between 000111 and 011100?
4. What is the Hamming distance between 101010 and 010101?
5. Consider this set C of 5-bit codewords:

   $$C \;=\; \{\,00000, 00111, 11011, 11100\,\}\;?$$

   (a) What is the rate of code C? Assume that the 4 messages are equally likely and are given by the source as a sequence of 2 bits.

   (b) What is the smallest Hamming distance between pairs of distinct codewords in C?

   (c) How many errors can be *detected* when using code C?

   (d) How many errors can be *corrected* when using code C?

6. (Extra)

   (a) Prove that the Hamming distance between two n-bit words, both having an even number of 1s, is even.

   (b) What is the largest code of 5-bit words (that is, having the largest number of codewords), where the Hamming distance between all pairs of distinct codewords is at least 2?

   (c) What is the rate of that code?

   (d) How many bit errors per codeword can this code *detect*?

   (e) How many bit errors per codeword can this code *correct*?

   (f) Can you generalize this result for codes with n-bit codewords? What is the rate of those codes?

Hint Look up the definition of Hamming distance.

Answer

1. 2
2. 2
3. 4
4. 6
5. (a) The rate is the number of bits on the input divided by the number of bits on the output: $2/5$.

   (b) 3. There are 6 pairs to consider (and only 4, if you take the left-right symmetries into account). The pairwise Hamming distances are 3 or 4.

   (c) Code C can detect 2-bit errors, because its minimum Hamming distance is $3 \geq \underline{2} + 1$.

   (d) Code C can correct single-bit errors, because its minimum Hamming distance is $3 \geq 2 \cdot \underline{1} + 1$.
6. (a) The total number of 1s in the two words together is also even. Consider the process that, one by one, removes bits, appearing at the same position in these words, that have the same value (both 0, or both 1).

   Invariants of this process are:

   - the Hamming distance between the two words is unchanged (the same after as before), because the process only removes bits at positions where they are not different;
   - the total number of 1s in the two words together is even, because the process removes 1s in pairs.

   The process terminates when the words differ in all positions. What remains are two words with an even number of 1s in positions where the other word has a 0. Thus, their Hamming distance is even. By the second invariant this is also the Hamming distance between the original words.

   (b) In total there are $2^5 = 32$ words of 5 bits. Half of these have an even number of 1s (start with all $2^4 = 16$ words of 4 bits, and append a *parity* bit to make the total number of 1s even). In this set of words with even parity, all Hamming distances between distinct words are even and not zero; hence, at least 2. All non-codewords have an odd number of 1s, and they are at Hamming distance 1 of a at least one codeword (flip one bit and the parity will be even). None of them can be added to the code, because then the minimum Hamming distance would go down. to 1.

   You cannot fit more words into such a code, even if you try a different approach for defining the codewords. Each codeword will be at Hamming distance 1 to 5 words, that must be non-codewords (otherwise, the minimum Hamming distance of the code would not be 2). At best each non-codeword is at Hamming distance 1 to 5 codewords. Hence, the number of non-codewords is at least half the number of words.

   You could call this a *perfect* code with minimum Hamming distance 2.

   (c) Its rate is $4/5$, because 4 source bits are encoded as 5 channel bits.

   (d) This code can detect all single-bit errors, because its minimum Hamming distance is $2 \geq \underline{1} + 1$.

(e) The code cannot correct errors, because to correct single-bit errors, its minimum Hamming distance should be at least $2 \cdot \underline{1} + 1 = 3 > 2$.

(f) This type of code can be constructed for any word length n. Take as codewords all words with an even number of 1s. There are 2^{n-1} of these; that is, half of the words will be codewords. And its minimum Hamming distance is 2. It is known as a *parity check code*. Their rate is $(n-1)/n$.

Exercise 6: (Repetition codes)

1. How many errors can be *detected* by a 6-repeat code?
2. How many errors can be *corrected* by a 6-repeat code?
3. How many errors can be *detected* by a k-repeat code ($k \geq 1$)?
4. How many errors can be *corrected* by a k-repeat code ($k \geq 1$)?

Hint Look up the bounds on error detection and error correction.

Answer

1. The 6-repeat code can detect up to 5 bit errors per codeword. Any change of at most 5 bits cannot change one codeword into another codeword. When the receiver receives a non-codeword, it is clear that something went wrong. (Note that there are only two codewords.)

   To detect 5 bit errors per codeword, the minimum Hamming distance between codewords must be at least $5 + 1 = 6$. To detect 6 bit errors per codeword, the minimum Hamming distance between codewords must be at least $6 + 1 = 7$. Hence, the 6-repeat code can detect up 5 bit errors, but not 6 bit errors.

2. The 6-repeat code can correct up to 2 bit errors per codeword.

   To correct 2 bit errors per codeword, the minimum Hamming distance between codewords must be at least $2 \cdot 2 + 1 = 5$. And to correct 3 bit errors, the minimum distance should be at least $2 \cdot 3 + 1 = 7$. Hence, the 6-repeat code can correct all 2 bit errors, but not all 3 bit errors..

3. In the k-repeat code, the Hamming distance between the two codewords is k. Hence, it can detect up to $k - 1$ bit errors per codeword, because $k \geq (k - 1) + 1$.

4. In the k-repeat code, the Hamming distance between the two codewords is k. Hence, it can correct up to $\lfloor (k - 1)/2 \rfloor$ bit errors per codeword, because $k \geq 2 \cdot \lfloor (k - 1)/2 \rfloor + 1$ (verify that $(k - 1)/2 \geq \lfloor (k - 1)/2 \rfloor$).

   Here, the expression $\lfloor x \rfloor$ denotes x rounded down to the nearest integer.

Exercise 7: (Hamming $(7, 4)$ error-correcting code)

1. What do the numbers 7 and 4 signify in 'Hamming $(7, 4)$ error-correcting code'?
2. Write out the all codewords of the Hamming $(7, 4)$ error-correcting code.
3. Look at how the parity bits differ, when the 4 source bits of two codewords differ only in one bit. How much difference (Hamming distance) should there be between the parity bits?
4. This code is perfect, in the sense that every non-codeword is at Hamming distance 1 to exactly one codeword. Verify this property.
5. (Advanced question) What other combinations of code length and number of source bits can give rise to a perfect single-bit error-correcting code?

Hint Look up the definition of Hamming $(7, 4)$ error-correcting code.

How many 7-bit words are there? How many words are adjacent (at Hamming distance 1) to a codeword?

Answer

1. The number 7 is the code length, that is, the number of bits in each codeword. The number 4 is the number of 'source' bits, that is, the number of bits that can be freely chosen to encode a message. The remaining $7 - 4 = 3$ bits are so-called *partiy bits*, that are uniquely determined by source bits. Hence, from a compression point of view, these parity bits are completely redundant.

2. There are 16 codewords in this code:

source bits	*parity bits*
0000	000
0001	111
0010	110
0011	001
0100	101
0101	010
0110	011
0111	100
1000	011
1001	100
1010	101
1011	010
1100	110
1101	001
1110	000
1111	111

3. The Hamming $(7, 4)$ code can correct every single-bit error. Hence, the minimum Hamming distance between pairs of code words must be at least 3. If the distance between the blocks of source bits is only 1, then the distance between the blocks of parity bits must be at least 2. In many cases the distance between the parity bits is in fact 3. Whenever the rightmost source bit changes, all three parity bits change, yielding a total Hamming distance of 4. Whenever any of the other source bits changes, only two parity bits change, yielding a total Hamming distance of 3.

4. Every codeword has exactly 7 non-codewords at Hamming distance 1, that is, where only one bit in the codeword was changed, because each codeword has 7 bits.

   Because the minimum Hamming distance between codewords is at least 3, those 7 non-codewords are more than 2 or more bit changes removed from other codewords. So, those non-codewords belong to a unique codeword.

   Let's count. There are 2^4 codewords (because there are 4 source bits that can be chosen freely). Thus, there are at least $7 \cdot 2^4$ non-codewords. That makes a total of $8 \cdot 2^4 = 2^3 \cdot 2^4 = 2^7$ words.

   Since there are only 2^7 words of 7 bits, these are all the words. Therefore, every non-codeword is at Hamming distance 1 from a codeword. In other words, the code is perfect.

5. Suppose there are m source bits and, hence, $M = 2^m$ codewords. If there are k parity bits, then the length of the codewords is $n = m + k$ bits. Each codeword has n non-codewords at Hamming distance 1. These non-codewords are further away from other codewords (otherwise, the code would not be able to correct all single-bit errors).

   This amounts to a total of $(1 + n)M = (n + 1)2^m$ words. Altogether there are 2^n words of n bits. To be perfect, we must have $2^n = (n + 1)2^m$. Consequently, $n + 1$ must be a power of two; in fact, $n = 2^{n-m} - 1 = 2^k - 1$ and $m = n - k = 2^k - k - 1$. It turns out that for all $k \geq 2$, these perfect codes exist. They are all known as *Hamming codes*.

   A special case is $k = 2$, with $n = 3$ and $m = 2$ (a smaller m is not interesting; why?). This is the 3-repeat code, where each bit is sent three times. The two codewords are $\{\, 000, 111 \,\}$. The minimum Hamming distance is obviously 3.

   For $k = 3$ we get the Hamming $(7, 4)$ code.

   The next one in the family is $k = 4$, $n = 2^4 - 1 = 15$ and $m = 15 - 4 = 11$. It encodes 11 source bits in codewords of 15 bits, by adding 4 parity bits, and it can correct all single-bit errors. Note that the rate of this code is $11/15$, which is better than $4/7$. But given its longer length, it provides less improvement in reliability, since only a single-bit error can be corrected on each block of 15 transmitted bits.

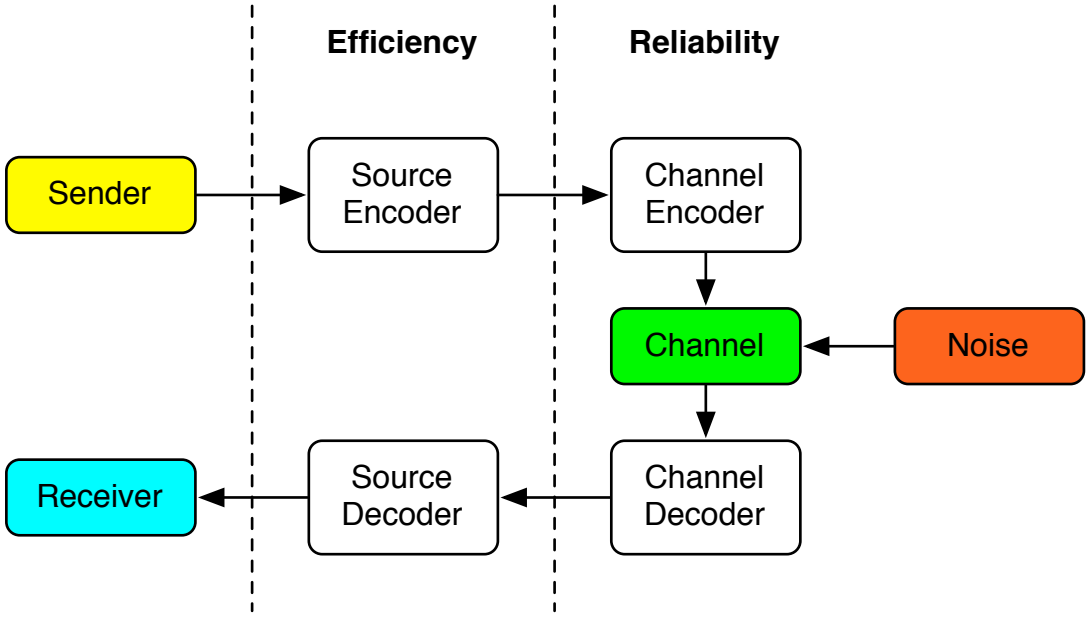
Exercise 8: We have a low-entropy information source, whose messages need to be sent across a secure, noisy channel to a receiver. We have facilities for

- lossless data compression: compress and decompress
- encryption: encrypt and decrypt
- applying an error-correcting code: encode and decode

Connect these nine blocks in the most appropriate order to set up the data stream.

Hint Consider the purpose of *source* coding and *channel* coding.

Answer Source coding is done to remove redundancy from the source signal; channel coding is done to add redundancy to protect against noise on the channel. Decoding must be done in the reverse order.



To the left of the right-most dashed line, you see a 'compressed source' with (almost) no redundancy. That is, the *source together with the source encoder* can be viewed as a new (higher entropy) source. To the right of the right-most dashed line, you see a (almost 100%) 'reliable channel'. That is, the *channel together with the channel encoder and decoder* can be viewed as a new (more reliable) channel.