MIT 6.02 **DRAFT** *Lecture Notes*
Spring 2010 (Last update: March 8, 2010)
Comments, questions or bug reports?
Please contact 6.02-staff@mit.edu

LECTURE 9
Viterbi Decoding of Convolutional Codes

This lecture describes an elegant and efficient method to decode convolutional codes. It avoids the explicit enumeration of the 2^N possible combinations of N-bit parity bit sequences. This method was invented by Andrew Viterbi ('57, SM '57) and bears his name.

9.1 The Problem

At the receiver, we have a sequence of voltage samples corresponding to the parity bits that the transmitter has sent. For simplicity, and without loss of generality, we will assume 1 sample per bit.

In the previous lecture, we assumed that these voltages have been digitized to form a *received bit sequence*. If we decode this received bit sequence, the decoding process is termed **hard decision decoding** (aka "hard decoding"). If we decode the voltage samples directly *before digitizing them*, we term the process **soft decision decoding** (aka "soft decoding"). The Viterbi decoder can be used in either case. Intuitively, because hard decision decoding makes an "early" decision regarding whether a bit is 0 or 1, it throws away information in the digitizing. It might make a wrong digitizing decision, especially for voltages near the threshold, introducing a greater number of bit errors in the received bit sequence. Although it still produces the most likely transmitted sequence, by introducing additional errors in the early digitization, the overall reduction in BER is less than with soft decision decoding. But it is conceptually a bit easier to understand, so we will start with hard decision decoding.

As mentioned in the previous lecture, the trellis provides a good framework for understanding decoding. Suppose we have the entire trellis in front of us for a code, and now receive a sequence of digitized bits (or voltage samples). If there are no errors (or if noise were low), then there will be some path through the states of the trellis that would exactly match up with the received sequence. That path (specifically, the concatenation of the encoding of each state along the path) corresponds to the transmitted parity bits. From there, getting to the original message is easy because the top arc emanating from each node in

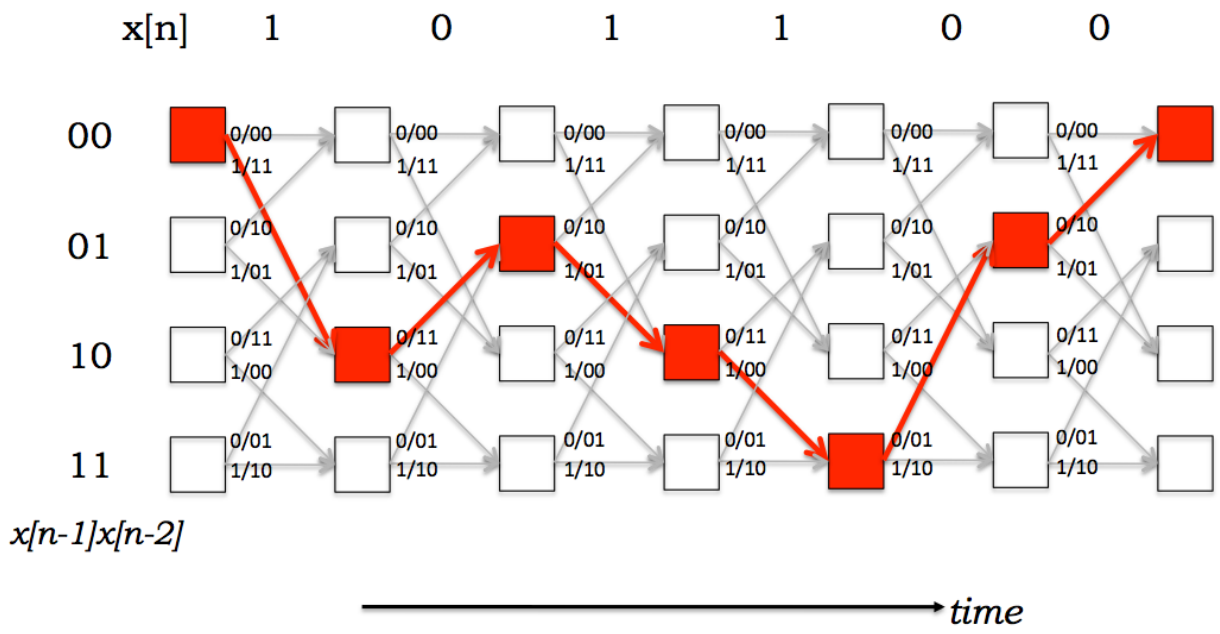

Figure 9-1: The trellis is a convenient way of viewing the decoding task and understanding the time evolution of the state machine.

the trellis corresponds to a "0" bit and the bottom arrow corresponds to a "1" bit.

When there are errors, what can we do? As explained earlier, finding the *most likely* transmitted message sequence is appealing because it minimizes the BER. If we can come up with a way to capture the errors introduced by going from one state to the next, then we can accumulate those errors along a path and come up with an estimate of the total number of errors along the path. Then, the path with the smallest such accumulation of errors is the path we want, and the transmitted message sequence can be easily determined by the concatenation of states explained above.

To solve this problem, we need a way to capture any errors that occur in going through the states of the trellis, and a way to navigate the trellis without actually materializing the entire trellis (i.e., without enumerating all possible paths through it and then finding the one with smallest accumulated error). The Viterbi decoder solves these problems.

■ 9.2 The Viterbi Decoder

The decoding algorithm uses two metrics: the **branch metric** (BM) and the **path metric** (PM). The branch metric is a measure of the "distance" between what was transmitted and what was received, and is defined for each arc in the trellis. In hard decision decoding, where we are given a sequence of digitized parity bits, the branch metric is the *Hamming distance* between the expected parity bits and the received ones. An example is shown in Figure 9-2, where the received bits are 00. For each state transition, the number on the arc shows the branch metric for that transition. Two of the branch metrics are 0, corresponding to the only states and transitions where the corresponding Hamming distance is 0. The other non-zero branch metrics correspond to cases when there are bit errors.

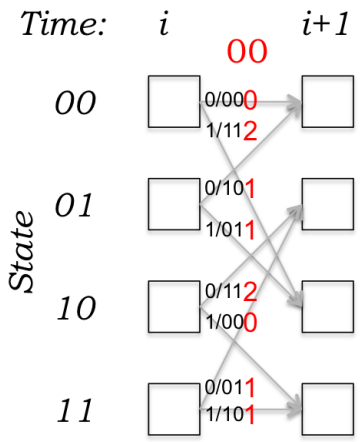

Figure 9-2: The branch metric for hard decision decoding. In this example, the receiver gets the parity bits 00.

The path metric is a value associated with a state in the trellis (i.e., a value associated with each node). For hard decision decoding, it corresponds to the Hamming distance over the most likely path from the initial state to the current state in the trellis. By "most likely", we mean the path with smallest Hamming distance between the initial state and the current state, measured over all possible paths between the two states. The path with the smallest Hamming distance minimizes the total number of bit errors, and is most likely when the BER is low.

The key insight in the Viterbi algorithm is that the receiver can compute the path metric for a (state, time) pair incrementally using the path metrics of previously computed states and the branch metrics.

9.2.1 Computing the Path Metric

Suppose the receiver has computed the path metric PM$[s, i]$ for each state s (of which there are 2^{k-1}, where k is the constraint length) at time step i. The value of PM$[s, i]$ is the total number of bit errors detected when comparing the received parity bits to the most likely transmitted message, considering all messages that could have been sent by the transmitter until time step i (starting from state "00", which we will take by convention to be the starting state always).

Among all the possible states at time step i, the most likely state is the one with the smallest path metric. If there is more than one such state, they are all equally good possibilities.

Now, how do we determine the path metric at time step $i + 1$, PM$[s, i + 1]$, for each state s? To answer this question, first observe that if the transmitter is at state s at time step $i + 1$, then it must have been in only one of two possible states at time step i. These two *predecessor states*, labeled α and β, are always the same for a given state. In fact, they depend only on the constraint length of the code and not on the parity functions. Figure 9-2 shows the predecessor states for each state (the other end of each arrow). For instance, for state 00, $\alpha = 00$ and $\beta = 01$; for state 01, $\alpha = 10$ and $\beta = 11$.

Any message sequence that leaves the transmitter in state s at time $i + 1$ *must have* left

the transmitter in state α or state β at time i. For example, in Figure 9-2, to arrive in state '01' at time $i+1$, one of the following two properties *must hold*:

1. The transmitter was in state '10' at time i and the i^{th} message bit was a 0. If that is the case, then the transmitter sent '11' as the parity bits and there were two bit errors, because we received the bits 00. Then, the path metric of the new state, PM['01', $i+1$] is equal to PM['10', i] + 2, because the new state is '01' and the corresponding path metric is larger by 2 because there are 2 errors.
2. The other (mutually exclusive) possibility is that the transmitter was in state '11' at time i and the i^{th} message bit was a 0. If that is the case, then the transmitter sent 01 as the parity bits and tere was one bit error, because we received 00. The path metric of the new state, PM['01', $i+1$] is equal to PM['11', i] + 1.

Formalizing the above intuition, we can easily see that

$$\text{PM}[s, i+1] = \min(\text{PM}[\alpha, i] + \text{BM}[\alpha \rightarrow s], \text{PM}[\beta, i] + \text{BM}[\beta \rightarrow s]), \tag{9.1}$$

where α and β are the two predecessor states.

In the decoding algorithm, it is important to remember which arc corresponded to the minimum, because we need to traverse this path from the final state to the initial one keeping track of the arcs we used, and then finally reverse the order of the bits to produce the most likely message.

■ 9.2.2 Finding the Most Likely Path

We can now describe how the decoder finds the most likely path. Initially, state '00' has a cost of 0 and the other $2^{k-1}-1$ states have a cost of ∞.

The main loop of the algorithm consists of two main steps: calculating the branch metric for the next set of parity bits, and computing the path metric for the next column. The path metric computation may be thought of as an *add-compare-select* procedure:

1. *Add* the branch metric to the path metric for the old state.
2. *Compare* the sums for paths arriving at the new state (there are only two such paths to compare at each new state because there are only two incoming arcs from the previous column).
3. *Select* the path with the smallest value, breaking ties arbitrarily. This path corresponds to the one with fewest errors.

Figure 9-3 shows the algorithm in action from one time step to the next. This example shows a received bit sequence of 11 10 11 00 01 10 and how the receiver processes it. The fourth picture from the top shows all four states with the same path metric. At this stage, any of these four states and the paths leading up to them are most likely transmitted bit sequences (they all have a Hamming distance of 2). The bottom-most picture shows the same situation with only the *survivor paths* shown. A survivor path is one that has a chance of being the most likely path; there are many other paths that can be pruned away because there is no way in which they can be most likely. The reason why the Viterbi decoder is practical is that the number of survivor paths is much, much smaller than the total number of paths in the trellis.

Another important point about the Viterbi decoder is that *future knowledge* will help it break any ties, and in fact may even cause paths that were considered "most likely" at a

certain time step to change. Figure 9-4 continues the example in Figure 9-3, proceeding until all the received parity bits are decoded to produce the most likely transmitted message, which has two bit errors.

9.3 Soft Decision Decoding

Hard decision decoding digitizes the received voltage signals by comparing it to a threshold, *before* passing it to the decoder. As a result, we lose information: if the voltage was 0.500001, the confidence in the digitization is surely much lower than if the voltage was 0.999999. Both are treated as "1", and the decoder now treats them the same way, even though it is overwhelmingly more likely that 0.999999 is a "1" compared to the other value.

Soft decision decoding builds on this observation. It *does not digitize the incoming samples prior to decoding*. Rather, it uses a continuous function of the analog sample as the input to the decoder. For example, if the expected parity bit is 0 and the received voltage is 0.3 V, we might use 0.3 (or 0.3^2, or some such function) as the value of the "bit" instead of digitizing it.

For technical reasons that will become apparent later, an attractive soft decision metric is the *square* of the difference in analog values between the received voltage and the expected one. If the convolutional code produces p parity bits, and the p corresponding analog samples are $v = v_1, v_2, \ldots, v_p$, one can construct a soft decision branch metric as follows

$$\text{BM}_{\text{soft}}[u, v] = \sum_{i=1}^{p} (u_i - v_i)^2, \tag{9.2}$$

where $u = u_1, u_2, \ldots, u_p$ are the *expected* p parity bits (each a 0 or 1). Figure 9-5 shows the soft decision branch metric for $p = 2$ when u is 00.

With soft decision decoding, the decoding algorithm is identical to the one previously described for hard decision decoding, except that the branch metric is no longer an integer Hamming distance but a positive real number (if the voltages are all between 0 and 1, then the branch metric is between 0 and 1 as well).

It turns out that this soft decision metric is closely related to the *probability of the decoding being correct* when the noise is Gaussian. To keep the math simple, let's look at the simple case of 1 parity bit (the more general case is a straightforward extension). Given a soft decision metric of x_i for the i^{th} bit, the PDF that x_i corresponds to the expected parity is the Gaussian, $f(x) = \frac{e^{-x^2/2\sigma^2}}{\sqrt{2\pi\sigma^2}}$. The logarithm of this quantity, $\log f(x)$, is proportional to $-x^2$.

The probability density of the entire decoding path being correct, assuming that the noise process is independent (and identically distributed) is the product of the individual PDFs, $\Pi_i f(x_i)$. Observe, however, that $\log \Pi_i f(x_i) = \sum_i log f(x_i) \propto \sum_i -x_i^2$.

The branch metric of Equation (9.2) leads to a path metric that is directly proportional to $-\log \Pi_i f(x_i)$. Minimizing this quantity is exactly the same as maximizing the PDF of a correct decoding! Hence, for soft decision decoding with the metric described above, the path metric is proportional to the log of the likelihood of the chosen path being correct when the noise is Gaussian.

This connection with the logarithm of the probability is the reason why we chose the

sum of squares as the branch metric in Eq. (9.2). A different noise distribution (other than Gaussian) may entail a different soft decoding branch metric to obtain an analogous connection to the PDF of a correct decoding.

9.4 Summary

From its relatively modest, though hugely impactful, beginnings as a method to decode convolutional codes, Viterbi decoding has become one of the most widely used algorithms in a wide range of fields and engineering systems. Modern disk drives with "PRML" technology to speed-up accesses, speech recognition systems, natural language systems, and a variety of communication networks use this scheme or its variants.

In fact, a more modern view of the soft decision decoding technique described in this lecture is to think of the procedure as finding the most likely set of traversed states in a *Hidden Markov Model* (HMM). Some underlying phenomenon is modeled as a Markov state machine with probabilistic transitions between its states; we see noisy observations from each state, and would like to piece together the observations to determine the most likely sequence of states traversed. It turns out that the Viterbi decoder is an excellent starting point to solve this class of problems (and sometimes the complete solution).

On the other hand, despite its undeniable success, Viterbi decoding isn't the only way to decode convolutional codes. For one thing, its computational complexity is exponential in the constraint length, k, because it does require each of these states to be enumerated. When k is large, one may use other decoding methods such as BCJR or Fano's sequential decoding scheme, for instance.

Convolutional codes themselves are very popular over both wired and wireless links. They are often used as the "inner code" with an outer block error correcting code, but they may also be used with just an outer error detection code.

We are now at the end of the four lectures on error detection and error correcting codes. Figure 9-6 summarizes the different pieces of what we have learned so far in the course and how they fit together. These pieces are part of the *physical layer* of a communication network. We will get back to the different layers of a network in a few lectures, after studying another physical layer topic: modulation (in the context of frequency division multiplexing).

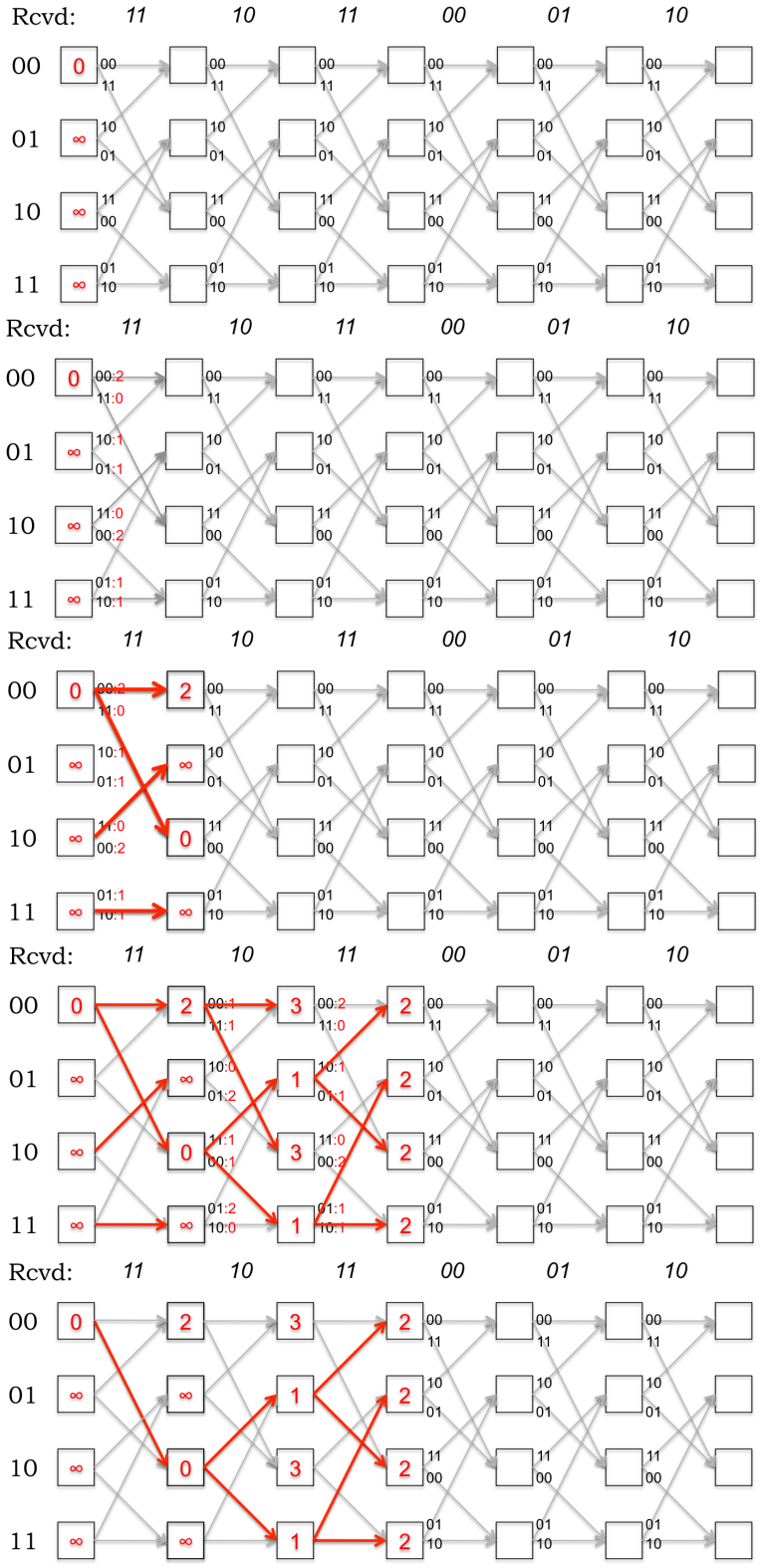

Figure 9-3: The Viterbi decoder in action. This picture shows 4 time steps. The bottom-most picture is the same as the one just before it, but with only the survivor paths shown.

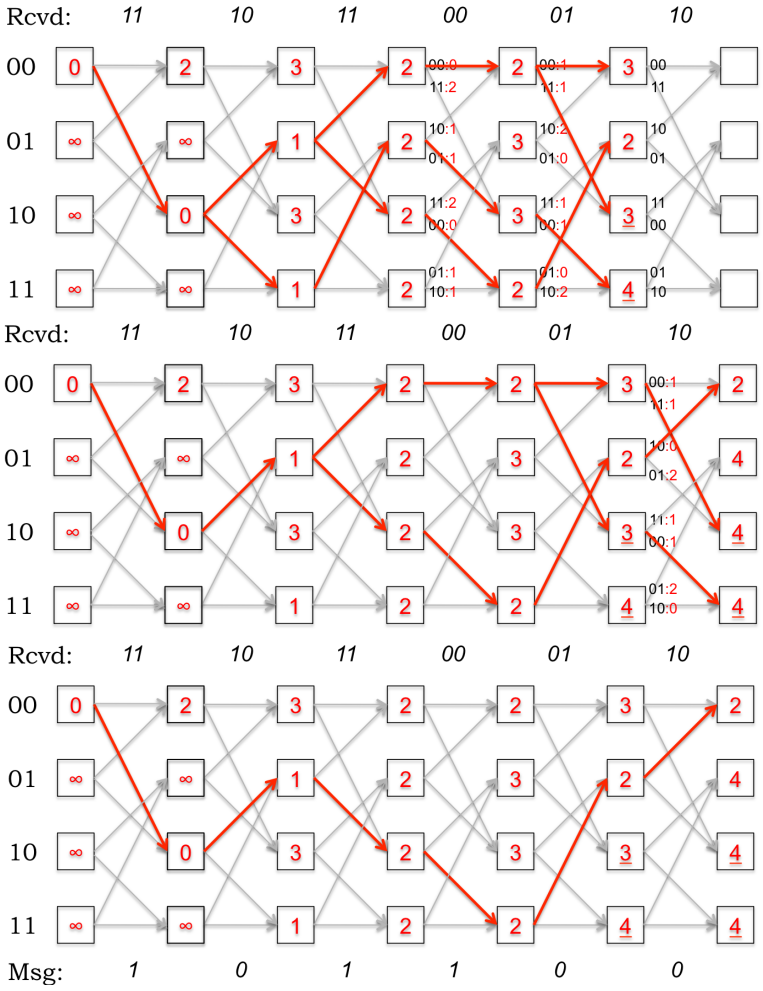

Figure 9-4: The Viterbi decoder in action (continued from Figure 9-3. The decoded message is shown. To produce this message, start from the final state with smallest path metric and work backwards, and then reverse the bits. At each state during the forward pass, it is important to remeber the arc that got us to this state, so that the backward pass can be done properly.

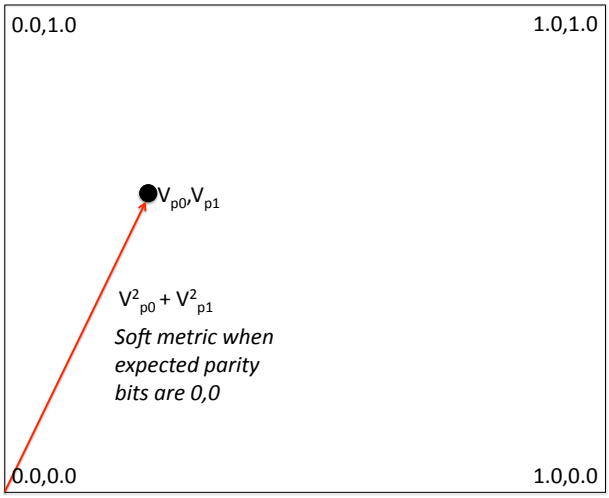

Figure 9-5: Branch metric for soft decision decoding.

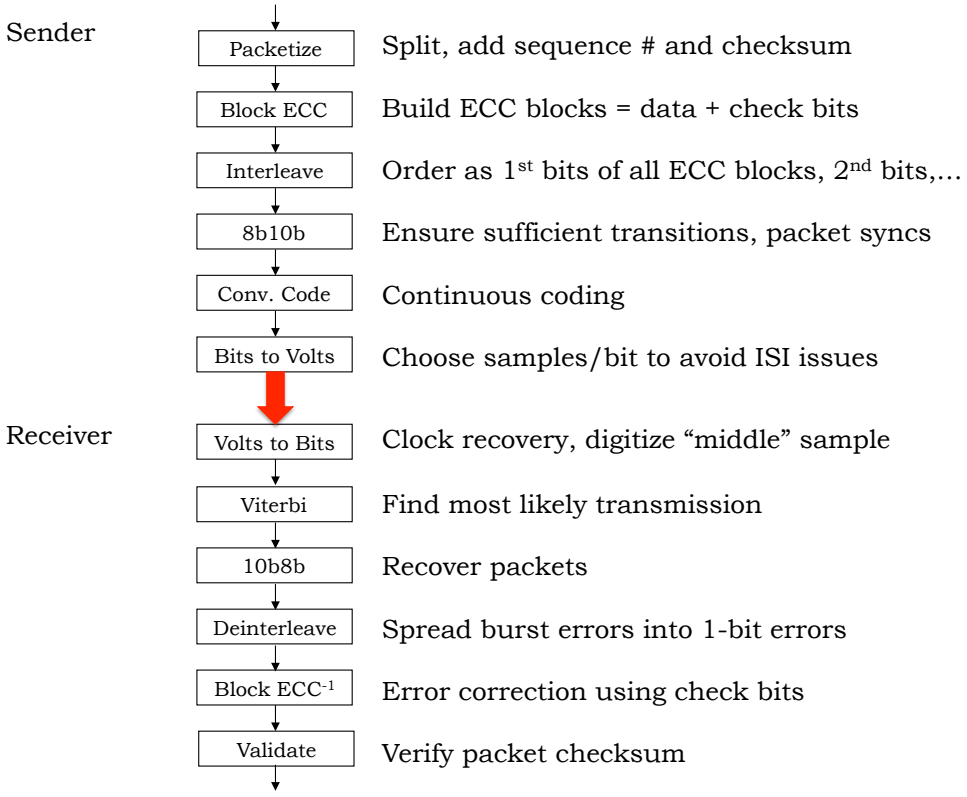

Figure 9-6: How the pieces we have learned so far fit together.