Lecture 15: Physical layer concepts

Anirudh Sivaraman

2020/12/17

This week, we'll talk about the lowest layer of the 5-layer stack: the physical layer. This is the layer that deals with the ground realities of transmitting data on a cable or an antenna. It is also the layer that is closer to electrical engineering than computer science.

Why are we learning about this layer? It's not because it is going to be particularly useful in a job that you take up after you graduate. The physical layer is mostly hidden from view and just gets out of the way, a testament to how well it is engineered. Even for many networking researchers, the layer is just a blackbox with a certain functionality. So why learn about it?

Opening up the physical layer serves to illustrate an important philosophical aspect of computer networks in specific and computer systems in general: you can always open up the blackbox, dig one level deeper, and understand things at one more level of detail. The philosophers call this the infinite regress problem where the validity of any statement can be repeatedly questioned. For this course, however, we'll stop at the physical layer, and not regress any further.

The discussion in this lecture will be quite qualitative. This is not because the physical layer is inherently qualitative. The reality is just the opposite. The physical layer is backed by some really beautiful mathematical concepts. However, these concepts require a familiarity with proabability, random processes, and signal processing, which I am not assuming for this class.

The central question for today's class is: how do we transmit bits on a wire? Next class, we'll see how we transmit bits on an antenna and what distinguishes an antenna from a wire.

We'll deal with transmitting bits as electromagnetic (EM) waves—as opposed to (say) sound waves. While there are examples of transmission media that use sound as a communication medium (e.g., SONAR and human communication), the bulk of digital communication today occurs using electromagnetic waves as the communication medium. So, we'll focus on this alone. The general concepts discussed here only rely on the "wave" part of EM waves, and they can be generalized to sound waves as well. The major difference is quantitative: communication over a sound medium has lower bandwidth (and hence capacity) than communication over an EM wave medium.

We'll deal with three primary concerns in this lecture:

1. How do we translate bits into voltages on the wire?
2. How do we transmit these voltages from one end of the wire to the other?
3. How do we check that the bits/voltages sent at one end are received correctly at the other end?

There are other concerns in the physical layer, which we won't be dealing with at all, such as the detailed implementation of carrier sense, the ability to carry multiple voltage signals on the same wire without interference (multiplexing), and the ability to automatically detect *and* correct a limited number of transmission errors (channel coding).

1 Translating bits into voltages

The simplest method to translate bits into voltages is to use a high voltage for a 1 and a low voltage for a 0. But this doesn't make the best use of the voltage range. Instead, we can *quantize* the voltage range into more than 2 values and represent each of these values using a multi-bit pattern. For instance if the high voltage is 5 Volts, and the low voltage is 0 Volts, we can divide it up into 4 levels/values, which can be represented using 2 bits. For example, any voltage between 0 and 1.25 Volts can be represented by the value 0 (the bit pattern 00), any voltage between 1.25 and 2.5 Volts by the value 1 (the bit pattern 01), and so on.

Why does each value (or bit pattern) correspond to a range of voltages as opposed to a specific voltage? This is to provide a certain level of robustness to *noise*, which can modify the voltage while it is being transmitted from the sender to the receiver end of a wire. Noise is an unavoidable reality of any analog system, and is the key difference between the analog and digital worlds. Noise occurs for many reasons, e.g., manufacturing defects in the cables and the transmitter/receiver, imperfect contact between the transmitter and the cable, etc. Noise is what you are hearing when you hear static on your radio.

The extent of noise depends on the operating conditions for the wire, but in general the noise levels are lower (relative to the signal/voltage levels) in a wire compared with a wireless medium. We'll see how this affects the design of the wireless physical layer in the next lecture. Hence, a good figure of merit for the quality of a communication medium at the analog level is the signal to noise ratio, i.e., how dominant is the noise level relative to the voltage level. For a single point-to-point communication link, the Shannon-Hartley theorem that we briefly mentioned in a footnote earlier relates a channel's bandwidth and its signal-to-noise ratio (SNR) to the channel's capacity as follows:

$$C = B.log_2(1 + \frac{S}{N}) \tag{1}$$

In the equation above, C is the capacity in bits/second. B is the bandwidth in Hertz. S is the signal power in Watts (which varies as the square of the voltage). N is the noise power in Watts. The quantity S/N is called the signal-to-noise ratio. Because capacity depends on the log of the SNR, it is typical to measure SNR on a log scale.

Back to quantization, how does SNR affect quantization? The more quantized a voltage range, the more number of values (bit patterns) we can pack into a limited voltage range. But, the voltage range available to each bit pattern shrinks in the process, providing a lower and lower level of robustness against noise. In our 4-level example, each level has a 1.25 V range, which (very roughly speaking) translates to a tolerance to at most 1.25 V of noise.

What happens when the noise exceeds the tolerance level of a communication link? The bit patterns get mixed up: a value of 0, which belongs to the range of 0–1.25 V might get interpreted as a value of 1 if 1.25 V of noise were added to it. This leads to *bit-flip errors*, where a 0 gets flipped to a 1 and vice versa. The probability of a bit flip is related to the noise level in a system and hence the SNR. A large number of bit flips in a packet can lead to the packet itself being corrupted—and hence discarded.

To summarize, we want to pack in as many different levels as possible into a limited voltage range to maximize its utility, but how many levels we can pack depends on the extent of noise we expect to observe. Hence, communication links (especially the wireless ones) adaptively pick the number of quantization levels based on an estimate of the current SNR.

Most of this discussion carries over to the wireless physical layer as well. The difference between the wired and wireless physical layer have to do with *attenuation*, *multipath*, and *interference* in a wireless medium. Attenuation refers to how the voltage level of the transmitted bits degrades as we get further and further from the transmitter (this is just a consequence of the EM wave being an electric field). Multipath refers to how the EM waves from the transmitter to the receiver can take different paths. The waves arriving at the receiver from different paths can constructively or destructively interfere at the receiver depending on small differences in the receiver's position. One simplified view of both attentuation and multipath is that it reduces the SNR at the receiver in a wireless link. Interference refers to how EM waves from multiple transmitters can arrive at the receiver at the same time, destroying the reception of the signal for all transmitters in the process. Interference at the physical layer is what shows up as a collision in the MAC/link layer.

These 3 problems typically don't show up on a wired link, which is why wired links have higher SNRs (and hence higher capacity) than wireless links. Attenuation is not as much of a concern because the distance between the transmitter and receiver in a wire is typically less than the transmitter-to-receiver distance on a wireless channel. Further, the wire is a much more shielded medium than the air that carries EM waves. Multipath is not as much of a concern because there are only a few paths between the sender and the receiver on a wire because the paths are all confined to the volume within the wire. Similarly, the wire isolates one sender from another; hence, interference is not a concern. The consequence of a higher SNR on a wire relative to a wireless link is that the incidence of bit flips is much lower on a wire relative to a wireless link.

2 Transmitting voltages

The next question is: how do we actually transmit the voltages from one end of the wire to the other? This is accomplished by a process called *modulation*. The basic idea is to take a sequence of quantized voltage levels representing the bit patterns that make up the packet (this sequence is called the message signal) and overlay it on top of a *carrier signal*: an EM wave that is suitable for transmission over a wired or a wireless medium. As an example, on a WiFi network, the carrier signal has frequency around 2.4 GHz, while the modulation scheme that takes WiFi frames and overlays on the carrier is called Orthogonal Frequence Division Multiplexing (OFDM).[1]

In general, different properties of the carrier signal can be varied based on the bit pattern in the message signal at any given instant. In *frequency modulation* (FM), the frequency of the carrier signal is increased for higher bit patterns and decreased for lower bit patterns. This is the modulation scheme used by most radio stations around you, which have carrier signals around the 100 MHz range. For instance, ESPN New York 98.7 FM has a carrier signal of frequency 98.7 MHz.

In *amplitude modulation* (AM), the amplitude of the carrier signal is varied in response to the current bit pattern/voltage level in the message signal. WCBS Newsradio 880 is an example of a radio station that employs AM with a carrier signal of frequency 880 KHz. While AM and FM might seem like ancient voice-only technologies compared with the Internet of today (which offers data services as well), FM is finding new uses as a means of communicating low amounts of data to and from Internet-connected sensors over long distances.

Some versions of Ethernet use a form of modulation called Pulse Amplitude Modulation (PAM). While in AM and FM, the carrier signal is an analog sinusoidal signal, in PAM, the carrier signal is a rectangular wave of pulses that are equally spaced apart in time. The amplitude of the pulse is varied in response to the bit pattern (or voltage level) in the message signal.

Regardless of the modulation scheme, the sender and receiver need a way to *synchronize* so that they can agree on what constitutes the beginning or the end of a packet. Otherwise, the receiver is at risk of misconstruing the transmitter's transmissions. Synchronization is a primitive that recurs at many layers of the communication stack: for instance, TCP needs a synchronization step to coordinate between the sender and the receiver so that they can agree on what constitutes the beginning of an in-order bytestream.

At the physical layer, synchronization is typically achieved by using a preamble: a known sequence of bits that the receiver looks for in order to synchronize itself with the transmitter. For Etherent, this sequence if 56 bits long. The sequence needs to be long enough to prevent the receiver from latching onto noise on the medium: for instance, with a 1-bit preamble, there is a 50-50 chance that the receiver is just latching onto noise. This synchronization is implemented by a digital circuit that continuosly looks at incoming voltage samples at the receiver and correlates them against the know bit pattern to see if there is a sudden increase in the correlation.

3 Checking for transmission errors

As we discussed earlier, the presence of noise on a wired medium, and in addition the presence of multipath, interference, and attenuation on a wireless medium causes bit flip errors: a 0 being interpreted as a 1 at the receiver and vice versa. Hence, at the very least we need a way to detect these errors so that they are not propagated up the networking layers. Most communication links today also include a way to correct a certain number of these errors (called channel coding or forward error correction). In this lecture, we'll just look at the simplest way to check, but not correct, for a 1-bit error in a packet.

Error checking is accomplished using a checksum. An example of a checksum you may have encountered in practice is the md5sum program on most UNIX-like OSes. This is a program that you use to verify the integrity

[1] We won't be studying OFDM in this course.

of a large file that you download from a web site. This program goes over all the bits in the large file and computes a single scalar *sum* that is unique (with high probability) to each file.

We won't describe md5sum here, but instead describe the simplest checksum algorithm: a parity bit. A parity bit is a bit that detects whether the number of 1s (or 0s) in a packet is even (or odd). The transmitter appends this bit at the end of the packet. When the receiver receives the packet, it checks if the parity bit in the received packet is consistent with the number of 1s (or 0s) in the received packet. The parity bit can only detect 1-bit errors, and can incorrectly report that a transmission is error-free for a two-bit error. Parity bits are used in some transmission channels seen in microcontrollers or embedded systems, such as the UART. Here, a 7-bit data packet is used to represent all the ASCII characters and a single parity bit is added to the data packet before transmission. Parity bits are also used in some storage technologies.