5 BLINKENLIGHTS

Where we conclude that pretty can also be deadly, and we learn to read from LEDs

The first part of this book focused on various problems related to the design of the data entry point system. Those problems were limited to deducing input by observing seemingly unrelated behavioral patterns by a user with local access to a system. But as information moves farther down its path to the addressee and leaves this system, its exposure broadens, and problems become more tangible.

The second part of this book focuses on some of the problems that occur while the data remains within reach, but just after it leaves the originating system—moments before it enters the Internet. The exposure discussed here is limited to roughly the physical footprint of a local area network with its direct surroundings. An attack at this level requires an observation point that is local to the origin, but it does not require system-level access.

The specific problem discussed in this chapter is somewhat different from those discussed previously: the exposure now manifests at the hardware level, much like in TEMPEST, but is different. The beauty of this phenomenon, and the ease of observing it with no specialized equipment, more than justify giving it a closer look.

The Art of Transmitting Data

The need for computers to communicate with other electronic devices has been apparent since the beginning of practical computing, as has the difficulty of achieving this task reliably and on a budget. We can control the machine's internal communication by providing generous and custom-fit interfaces among all major components with a desired capacity, maintaining precise signal characteristics, and using a common reference clock for all operations, so that the recipient always knows when to listen, and the sender always knows when to transmit data. But communication over longer distances or to devices equipped with nonspecialized, cheap interfaces is a different challenge: the computer is forced to communicate over a medium that usually does not allow for the degree of freedom we have grown accustomed to working with on the insides of a single machine.

In fact, the situation is quite the opposite. The customer expects simple, convenient, practical, and cheap solutions, and requiring computers to be connected through a $100, 3-inch, 100-wire cable didn't seem like a winning solution. Simplicity is a necessity. The core of any external communication channel almost always relies on the serial transmission of subsequent bits that only when reassembled and grouped together produce numeric values, text strings, or other pieces of data native to the machine environment of the sender or recipient. In the most seemingly trivial and obvious scenario, when two machines or devices connected only by a pair of wires need to exchange information, they do so by setting one of the wires to high or low voltage in relation to the other (reference) line—or by using any other differing signals or states, for that matter. They do so in order to send subsequent bits of data at a given frequency—a frequency that must be kept reasonably close and in sync on both devices.

Even in such a trivial design, a number of problems immediately arise. First, the devices do not share a reference clock. Although both have internal quartz-based clocks, no two affordable clocks are ever accurate enough to maintain reliable and fast communications over an extended period of time due to slight manufacturing imperfections, interference, and other physical conditions. And serial communications demand precise synchronization. The straightforward bit-encoding scheme, usually referred to as Non-Return to Zero (NRZ), simply outputs one signal (voltage) for 0 and another signal for 1. In such a system, it is easy to keep both endpoints synchronized when values change on a regular basis—the system simply needs to detect a falling or rising edge, use it as a rough reference, and adjust its own clock accordingly. But given a longer sequence of 1s or 0s, it becomes difficult for the receiving side to accurately determine how many bits are being sent. In fact, even a small clock drift can cause problems, and there is no way to compensate for this during the exchange of a constant sequence of bits.

The obvious solution, to simply interleave the data with a separate, distinguishable timing signal, is not always the most convenient and efficient method; increased complexity and reduced throughput is often perceived as a nuisance.

To effectively address this problem, many systems use a scheme called *Manchester encoding*, also known as *biphase code*. The algorithm for Manchester coding, shown along with NRZ in Figure 5-1, encodes data using signal edges, as opposed to signal levels. The original, aforementioned NRZ encoding uses an internal clock to measure voltage levels at a constant pace, interpreting low voltage as binary 0 and high voltage as 1. Manchester encoding, on the other hand, carries data in transitions from low to high voltage or vice versa. In such a design, the signal is switched to high to denote binary 1 and to low to indicate 0.[*]

Although such encoding does not require the clocks to be kept synchronized, it is also not quite enough as it is: there is no way to encode two binary 0s or 1s, because it is not possible to go from low to high voltage twice without returning to low halfway down the road (and vice versa). To allow this type of information to be encoded, transitions that occur shortly after a falling or rising signal edge are ignored, thus allowing the system to encode multiple occurrences of 0 and 1 by returning to the same voltage midcycle. To manage the "blackout" period after a transition, a simple one-shot interval clock is necessary.

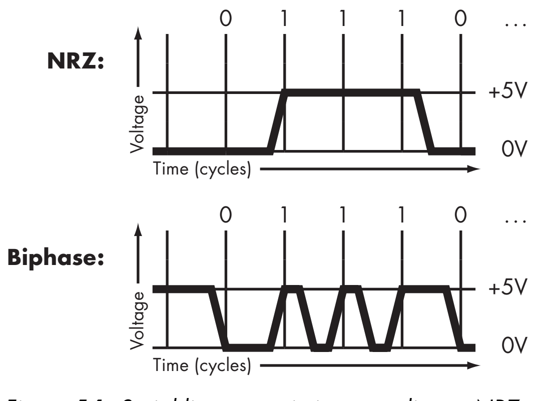


Figure 5-1: Serial line transmission encodings—NRZ and biphase (Manchester)

The design of a serial line based on the self-synchronizing scheme discussed above is often extended to provide full-duplex communications in which both parties can talk at once, either by using two separate lines (transmit and receive, Tx and Rx for short) or by using advanced echo detection and cancellation tricks to differentiate between its own signal and the data sent from the other side. Some mediums require or allow for more sophisticated signaling schemes, for example sending more than just one bit in every cycle;

[*] Or the other way around, depending on the transmitter design.

yet the principle of communications remains essentially the same, and Manchester encoding over the lowest possible number of wires—often two—is prevalent across the entire domain.

Equipped with a knowledge of the basics of "wire pair" serial communications, let's take a peak at two prominent examples of serial communications in the world of networking, see how they exchange data internally, and look at how this information can leak to third parties without the user noticing.

From Your Email to Loud Noises . . . Back and Forth

The most popular long-distance computer communications device is, hands down, a modem. Initially introduced in the 1950s for the maintenance and control of certain types of military equipment at remote locations, the modem brought the Internet to the masses. Although today often considered somewhat obsolete, the modem has given birth to many advanced technologies, such as affordable high-speed DSL (Digital Subscriber Line) systems or cable modems. These devices all use clever variations of the same set of techniques to communicate over phone lines or other nondedicated analog media using either audible or inaudible signals. The research invested in improving modems also contributed to our understanding of numerous large-scale design problems in electronics in general and computer and network design in particular. Thus, an understanding of how modems work is key to exploring other, perhaps more up-to-date, methods of long-distance data transmission.

The universality of the telephone line makes it a natural medium for computers to use for communication. Phone lines can be found almost anywhere, and phone systems provide excellent call-routing capabilities, making it possible to reach just about any location with little if any effort. There is a tiny caveat, though: phone lines were meant to carry the human voice, transmitted as a waveform, within narrow-frequency response range (usually not exceeding several kHz). Because these frequencies were recorded as voltage changes over a pair of wires and relayed through a number of analog repeaters and amplifiers, the standard of quality for the transmission wasn't particularly high. It had to be just good enough for people to hear and understand each other, and because the human brain is a superb signal filtering and processing system, occasional noise or sound-level fluctuations were not much of a concern—not until much later on, when customers grew a bit picky.

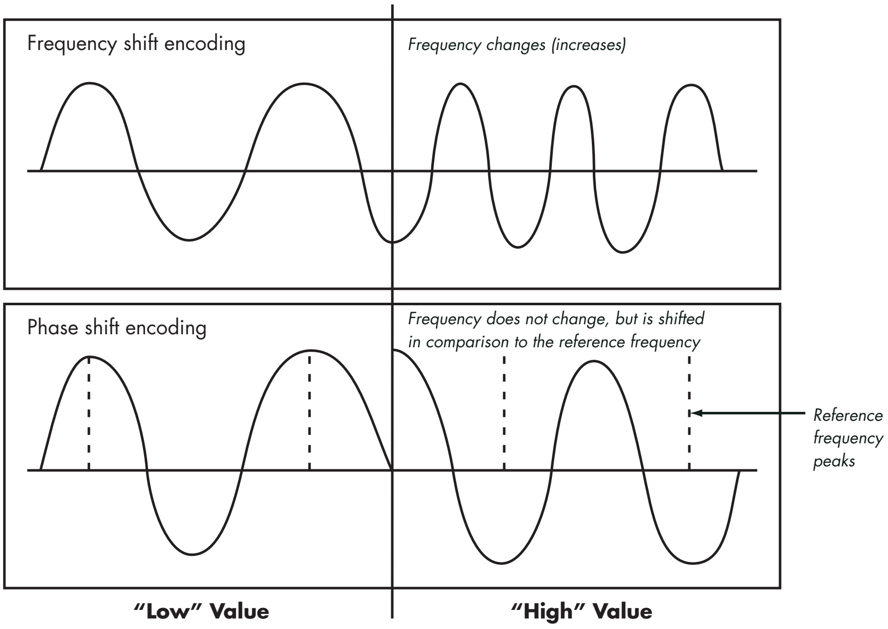
Computers, on the other hand, are generally engineered to exchange binary information, which is encoded using fairly precise voltage levels over well-designed, short lines with good signal characteristics and low capacitance—an exact opposite of long-distance, poorly shielded telephone lines with inadequate signal characteristics. Computers also need to talk much faster and much more than humans usually do. As such, modem designers had (huge understatement here) a difficult challenge to solve: They had to determine a way to encode bits of data not only in a manner that could be efficiently transmitted to a remote system over the wire (something that Manchester encoding made a bit easier), but also as audible signals that could be accurately distinguished at the other end of the line regardless of often entirely unpredictable voltage changes and other transmission artifacts. They had to employ robust error-correction algorithms and variable transmission speeds to compensate for poor line quality, occasional cross talk, trucks going over a buried phone line, birds building a nest on a pole, and so forth. The designers nodded, scratched their heads, and after perhaps just 40 years brought us an affordable and fairly fast method for computer-to-computer communication. Let's take an abbreviated look at how this developed and how the technology matured—yet essentially stayed the same—over the decades that followed.

The history of commercial modem development and standardization began in the 1960s when two standards, Bell 103/113 and V.21, were conceived. Both standards provided an amazing (for the time) 300-baud (bits per second) full-duplex connectivity using a technique called *frequency shift keying* (FSK). FSK is a mysterious-sounding term that happens to stand for a rather trivial signal-encoding scheme: it uses two different tones to denote different values, one frequency for "low," and another frequency for "high." The advantage of using audible frequencies over other types of signaling is rather significant: this is the only type of signal that can be relayed through the phone system fairly well—after all, this is what the system was designed for. All other signals are more or less destined to be trashed beyond recognition before reaching the other end of the wire, in the best-case scenario, or being immediately filtered out by bandpass filters somewhere down the line in the worst case.

In addition to FSK encoding, the aforementioned Bell 103/113 and V.21 standards split the frequency range that could be transmitted over phone lines in two: one of the modems, the caller, used a frequency of 980 Hz to encode low and 1,180 Hz to encode high. The other end, the answerer, used the higher part of the spectrum: 1,650 Hz and 1,850 Hz, respectively. Why split the frequency in this way? Because a phone line is essentially just a pair of wires, which can be used for transmission by two devices simultaneously (full duplex), but only if they are capable of dealing with the fact that their respective transmissions would superimpose on each other. In full-duplex communication, each device must be able to distinguish its own signal from the data it's receiving and filter it out. If it cannot do so successfully, each device would have to pause while the other end is talking (simplex mode), severely impairing the already sort of unimpressive throughput. By splitting the frequency, the phone line is essentially made to carry what it sees as two different "voices," thus ensuring that simultaneous communication can occur with no collisions.

It took 25 more years for modems to take another step in the right direction. The next major set of standards, Bell 212A and V.22, took a big leap forward and dropped frequency shift keying in favor of *differential phase shift keying* (DPSK). Rather than change the frequency of a wave, DPSK shifts its phase to signal different values.

The phase shift technique essentially introduces a minimal time shift, or delay, that causes the output audio signal to be slightly out of sync with the original reference wave, while maintaining exactly the same shape (see Figure 5-2).



Frequency shift encoding
Frequency changes (increases)
Phase shift encoding
Frequency does not change, but is shifted in comparison to the reference frequency
Reference frequency peaks
"Low" Value
"High" Value

Figure 5-2: Frequency shift versus phase shift

The value of the phase shift, also called the *shift value*, is expressed in degrees (a reference to its effect on trigonometric functions: y = sin(x) shifted by 90° is exactly the same as y = sin(90° + x). A shift value of 360° denotes a shift by the entire wavelength, which simply puts the waves right back in sync and has no effect on the waveform. The correspondence of various phase shifts is shown in Figure 5-3, on the left.

Once both parties are synchronized and have a way to compare the signal received over the cable with the expected waveform, the actual encoded data can be easily retrieved. A differential circuit can compare two signals, subtract them, and easily determine the exact phase shift of the signal, by comparing it to a reference signal, as shown in Figure 5-3, on the right.

The new standard also took advantage of a more advanced data-encoding method. Instead of simply using two alternating signals to transmit 0s and 1s, as was the case previously, V.22 encodes whole *dibits*—slang for pairs of bits. Encoding two bits at once can be achieved using four phase shift values, with the amount of shift used to denote each of the possible values chosen so that values are uniformly and possibly farthest spaced through the entire 360° spectrum—and thus easily distinguishable from each other (see Table 5-1).

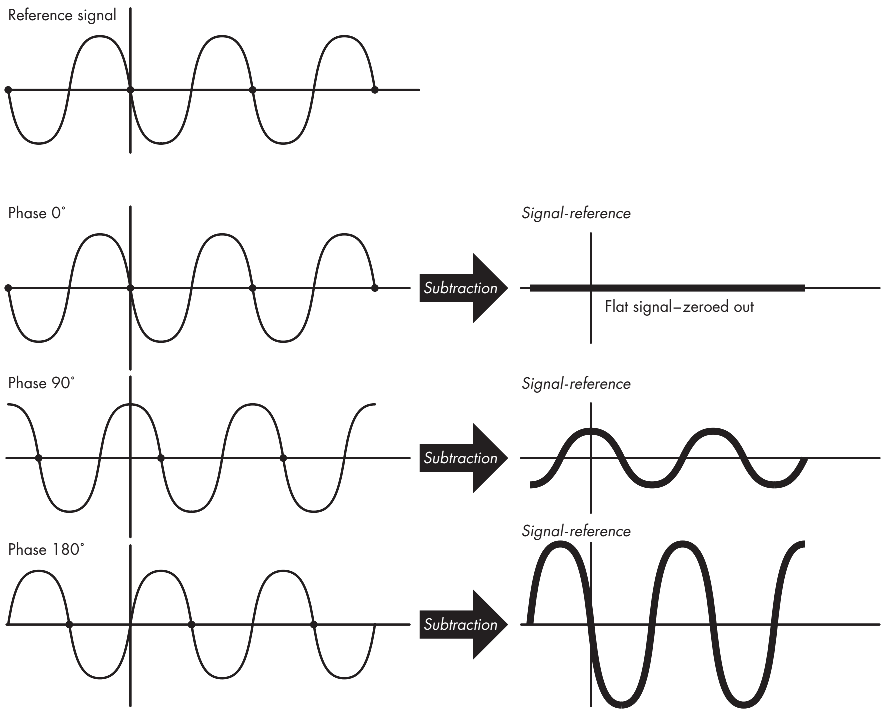


Figure 5-3: Phase shifted signals (left) and a result of subtracting a reference waveform to more easily distinguish between phases (right)

The use of dibits allowed for significantly faster transfer speed (1,200 baud) without the need to increase the physical rate with which the actual signal was modulated. Twice as much information—twice as many bits—was carried within every single beep.

Table 5-1: Using phase shifts to encode two bits of data (dibit)

Dibit	Phase Shift
00	90°
01	0°
10	180°
11	270°

NOTE *Although it is theoretically possible to use such an extended alphabet—that is, composite signal units similar to dibits (that have more than two states and thus encode more than one bit at once)—with FSK encoding as well, it is a bit more problematic to do so. FSK signals must avoid subharmonics and other frequencies that are particularly prone to distortion when sent through phone systems, thus severely limiting the set of possible states. The advantage of DPSK over FSK is that it uses a fixed frequency that is known to cause the fewest transmission problems and, hence, can be used more reliably at higher transmission rates.*

In the next few years, the pace of research accelerated a bit, and a number of new standards surfaced. The V.22bis standard took the concept of wide alphabet signaling a bit further, combining DPSK with signal amplitude (loudness) modulation to build a two-dimensional set of 16 possible values. The transition from a measured signal to binary values was expressed using a two-dimensional table. The value to which a signal corresponds is obtained by first looking up the column, based on the measured phase-shift value, and then the row is looked up based on the amplitude measurement. A simplified but analogous two-by-four example is shown in Table 5-2.

Table 5-2: Two-dimensional encoding of three bits using two distinct signal parameters

	Phase 0°	**Phase 90°**	**Phase 180°**	**Phase 270°**
Low amplitude	000 (0)	001 (1)	010 (2)	011 (3)
High amplitude	100 (4)	101 (5)	110 (6)	111 (7)

To add to the confusion, this new approach was called *quadrature amplitude modulation* (QAM). QAM once again made it possible to go from 1,200 to 2,400 bps without actually improving signal modulation speed, but by extending the number of meanings a single atom of signal can have.

The next major evolutionary step was V.32. V.32 was the first design to introduce a novel concept: instead of splitting frequencies, it used advanced echo cancellation circuitry* to detect and subtract the signal transmitted by the device itself from the data received over the wire. This technique allowed both devices (sender and receiver) to use the entire frequency spectrum, instead of just half of it, while still doing full-duplex.

Development continued, and the V.34 protocol soon appeared. Although the rate at which the signal could safely alternate before introducing excessive distortion did not noticeably change over the years, the standard was considerably faster than its predecessors. V.34 achieves a throughput of 28,800 baud, sometimes pushed a bit further by manufacturers to a unofficial speed of 33,600 baud (33.6 Kbps) by sending only about 2,500 to 3,500 signal samples (alphabet symbols) per second; however, it combines four different encoding schemes to build a four-dimensional structure with 1,664 possible states, making it possible to send as many as 41 bits at once. As it turns out, it's not about raw speed but how you use what you've got.

It is widely believed that the V.34 standard and its derivatives approach the theoretical limit for transmission of data via the voice-oriented telephone system. Although this may seem an odd statement given the prevalence of 56 Kbps modems, there is a catch: 56 Kbps devices achieve this transmission rate in a wholly different way than in analog solutions. Given that most phone systems have migrated from analog to digital since modems were first

* Echo cancellation circuits attempt to distinguish signals being sent by the device itself from those coming from the other party, and to eliminate or significantly reduce the former. Various types of such devices are commonly used not only in digital data transfer, but also to improve phone call quality, eliminate microphone feedback during public events, and solve many other everyday problems.

developed, and because most dial-up providers can now interface their systems directly with digital telecommunication systems, service providers can return to the most obvious but, until recently, impossible solution: changing line voltages instead of shifting frequencies when sending data to a subscriber. Because the signal is carried as digital data from the beginning—and can travel over buried copper lines only till the nearest telco facility—there are virtually no signal quality problems, and the only limit is the voice-carrying capacity designed into the phone system hardware. Working at 8,000 symbols per second, but operating with a considerably smaller alphabet (usually about 128 symbols, or voltage levels), it is possible to send data to a subscriber who is connected to a digital phone system with high-quality wire using a 56 Kbps modem at a higher speed than usual. The upstream transfer is still implemented the old-fashioned way, though, and is considerably slower; as such, the modem is only partly 56 Kbps, and only when conditions permit.

The Day Today

Not much has changed since the conception of modem technology. As transmission protocols advanced, so did the error-correction and fallback mechanisms needed to ensure reliable transmission when your favorite quadruped decides to chew the phone cable. A jungle of standards were spawned: V.42 provided a basic CRC (cyclic redundancy check) implementation, MNP-1 to MNP-4 provided proprietary error-correction algorithms, V.42bis and MNP-5 provided integrity checking combined with compression, and so on. But the real revolution is yet to come.

Or is it? You might argue that DSL and cable modems are a revolutionary technology that has changed the world. I am willing to argue: in fact, they are quite similar to their older cousins, modems. The only significant difference between the two is that the other endpoint—the server that handles all connections—has moved from a distant city where the service provider is located to the nearest local telco facility, and the connection to it can be made directly using the copper wire coming from the customer's residence or business. Because that direct connection again does not go through any other equipment, these devices can use high, inaudible frequencies and subtler signals that would otherwise be distorted or not relayed at all over the telephone network. In contrast, the good old modem was strictly limited to a narrow range of audible frequencies and signals that the phone system was intended to carry and that it could carry well. In many ways, DSL devices have it much easier than the old modem.

As we see, designing a modem is actually quite a complex and difficult task; that's why it took us decades to advance from bulky and expensive 300-baud devices to where we are now.[1] Surprisingly, all these devices can talk to one another, even to devices ten years older, even at the lowest speeds we long forgot about. Too, all are usually aware of the standards known to date, including the dozens of alternatives and forks of each. Doesn't that make modems even more a marvel of computer engineering?

But who pulls the strings?

Sometimes, a Modem Is Just a Modem

Modem-to-modem communications is, of course, not where the story starts or ends. The modem is just a piece of fairly inert middleware that's hardly even a good paperweight. For a modem to be of any use, it must be able to communicate with a computer to receive commands and exchange data, even when it's only being used for something as feeble as random web browsing. Internal modems have it easy: ISA (Integrated Systems Architecture), PCI (Peripheral Component Interconnect), PCMCIA (PC Memory Card International Association), and some other dedicated buses provide high-speed and fairly generous parallel interfaces that make the communication process almost trivial.

External modems (of the analog or DSL kind), however, have to do things the hard way, with a serial link. Most analog modems use the well-known serial protocol RS-232 (renamed in the '90s to the much more descriptive EIA/TIA-232-E[2]); many newer ones use USB (Universal Serial Bus). As we get close to the information disclosure scenarios in those devices, we want to get a glimpse of what happens to the data on its way between the modem and the computer, too, because that plays a crucial role in the attack.

Although external modems have to use inhumane means of communicating not only with a remote system, but also with the local machine itself, thanks to the proximity to the computer and the fact that interfaces such as RS-232 are digital and were designed for use by computers to start with, this stage is still much simpler than the phone line modulation and demodulation for which bit modems became famous.

RS-232 uses a fairly straightforward implementation of bipolar encoding for the data exchanged over two separate lines and backs this with a set of NRZ control lines. To make life a bit more interesting, RS-232 comes with a multitude of link or protocol features that make it fairly difficult to implement from scratch: its asynchronous nature, a wide array of possible settings and speeds, and unusual voltage levels. But with all this, RS-232 still does not even come close to a real challenge for an implementator who had dealt with signal modulation over phone lines.

USB, on the other hand, attempts to standardize and unify the serial interface. Although USB requires higher-end circuitry than RS-232 in order to interface a computer with a device (because of, among other things, a higher level of abstraction and higher supported transmission speeds), the USB is universal (hence its name) and has fewer oddities and legacy features.

Last but not least, a common method of communicating with local devices is the use of Ethernet, a mechanism somewhat similar to, but predating, USB. Let us look at Ethernet for a while now, and I am sure all those communication protocols will eventually meet in one place.

Collisions Under Control

Ethernet networks are, in essence, an advanced type of a multiparty serial link.[3] An Ethernet network is composed of a number of computers connected by a shared medium—nothing particularly complex, in its most basic form, just a pair of fairly regular wires. When a device on the network uses the medium, it applies a specific voltage to the wire, and all other connected systems can interpret the data by measuring the voltages. A set of checks ensures that devices do not try to use the link at the same time and that recovery is smooth if an accident happens. Still, even considering this possibility, the basic design is unbelievably trivial, compared with modems.

To work around the problem of two parties talking at once, a standard named Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is used as the core mechanism controlling all communication via Ethernet. Before sending any data, every device connected to Ethernet follows a CSMA procedure to see if another device is using the cable by checking the modem's electrical properties. If no other transmission is occurring, the device enters the transmission phase and beams its data out to the masses.

In this phase, the data is sent on the wire as a sequence of bits using *bipolar encoding*; the traffic contains a header with all the necessary sender and recipient information and a proper checksum intended to protect the integrity of the data in case of external or internal interference, quadruped or not. A network interface that considers itself to be acting on behalf of a recipient, presumably by comparing the observed destination address provided in the packet with its unique MAC (hardware) address stored on the card, should accept this traffic and verify the checksum. At the same time, all other parties should ignore this frame; naturally, if they do not (and almost every card can be instructed not to), the user can view or react to traffic addressed to others. (You can see how Ethernet was designed in the spirit of far-fetched trust and altruism—a noble but risky approach.)

It is possible (and not very unlikely) for two devices on an Ethernet network to start sending at exactly the same moment, even though both checked just microseconds or nanoseconds ago for another party transmitting. And, if they do transmit at exactly the same moment, a disaster is bound to happen. Two transmissions are mixed up and mangled, and the sent data should fail the checksum test at the destination . . . or should it?

Although the use of a checksum implemented within the Ethernet frame specification is typically sufficient to verify data transmission accuracy, it may not be particularly effective if the link is saturated and hundreds or thousands of collisions occur in a short period of time; it is just small enough to accidentally come out correct from time to time. The law of probabilities tells us that some damaged packets will—just by chance—have the same checksum as an original packet. Furthermore, even if we ignore the problem of checksum deficiencies, we still want to stop collisions as soon as possible—by just letting collisions run rampant, you might find that you are no longer able to ensure the timely retransmissions of mangled and dropped frames in your network. After all, the sender sent it with no indication of a problem, and the recipient did not receive anything even remotely resembling a useful packet.

The solution comes with the latter part of the standard: collision detection (CD). The specification calls for the sender to monitor the network link while explaining their business to others. If another party is caught trying to talk at the same time, that should be detected (again, with a simple measurement of the electrical properties of the line), and the transmission should be immediately aborted. The device should also send a special jam code to ensure that both frames (the one being sent and the one that interfered with it) will be unconditionally dropped, without even getting to the checksum verification; the recipient should be able to spot the jam code and stop the reception of data being processed. The device then idles for a gradually increasing and preferably (initially) random period of time after every attempt (called retransmission backoff), to minimize the likelihood of a subsequent collision.

NOTE *A fun fact: The jam code mechanism imposes an unusual requirement on the protocol. All frames must have a minimum (!) length, with the value calculated such that it allows the jam code to be generated and propagated to all machines before the transmission is completed. With very short frames, there may not be enough time to achieve this. Hence, the sender is required to artificially pad all their outgoing transmissions.*

Figure 5-4 shows the exact sequence of events in a typical collision scenario. As you can see, Sender A hopes to send data to the recipient but notices another transmission occurring, at which point they decide to wait until that transmission stops. Sender A then prepares to send the data but, unfortunately, Sender B does the same, and both conclude that it is safe to send data at nearly the same time.

Both attempt to transmit, data gets mangled, and at that point both detect the other transmission and quickly send a jam code to instruct the recipient to disregard this frame. Finally, both senders back off for a random amount of time and hopefully manage not to start simultaneously the next time around.

Behind the Scenes: Wiring Soup and How We Dealt with It

Although not an example of a particularly scalable or elegant design, the Ethernet protocol is amazingly powerful and easy to deploy; it enabled the building of cheap peer-structure networks using coaxial cables just about anywhere. As such, it has become a de facto standard, replacing many other (and sometimes superior, but more expensive or proprietary) networking architectures.

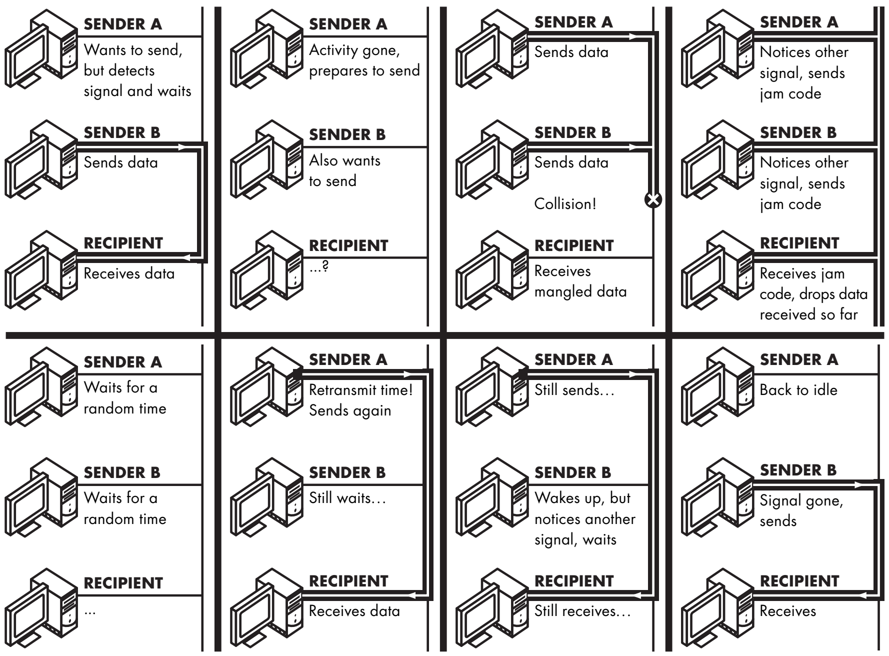


Figure 5-4: The stages of a typical Ethernet conversation

Naturally, simple Ethernet over coaxial cable had its limits and disadvantages; it was essentially based on a long piece of wire with devices hooked up to it at various locations, and with resistors on both ends, not something you'd want to be responsible for maintaining in a large office. A simple and difficult-to-debug mishap, such as a shorted terminal, could bring the entire infrastructure down. A more advanced—but only marginally more expensive—replacement was warmly welcomed.

Electronic multiport repeaters (hubs) made it possible to run wiring without much effort using twisted pair wiring (Cat-3 and Cat-5 cables with RJ-45 connectors). To use them, you simply plugged a piece of wire from your machine into a black box, and all other devices connected to this black box could communicate with it without much consideration of electrical problems or the risk that a single cable failure would bring down the entire network.

Hubs are, in essence, simple repeaters that broadcast all traffic received on one port to all other ports. They make it possible to build easily reconfigurable and more reliable star-type networks, but they do little else. As the network grows, the cost of broadcasting every bit of information to all locations, and the fact that only one party can talk at once across the entire network, makes it all too evident that the simplicity of this design is its major weakness.

Switches turned out to be the solution. Switches are the next generation of hubs. Equipped with a decent processor and some memory, they're a more expensive alternative to hubs that provide, under normal circumstances, additional high-level analysis of Ethernet frames. This analysis associates hardware addresses with specific ports and optimizes frame routing by delivering certain packets directly to the appropriate port (in unicast mode), instead of broadcasting them to all parties (see Figure 5-5). This greatly improves performance in more extensive networks.

NOTE *Another fun fact: Real hubs are almost extinct nowadays. Almost all 10/100 Mb devices marketed as hubs actually use basic switch chipsets; it is simply cheaper to repackage the chip than to develop and maintain several variants.*

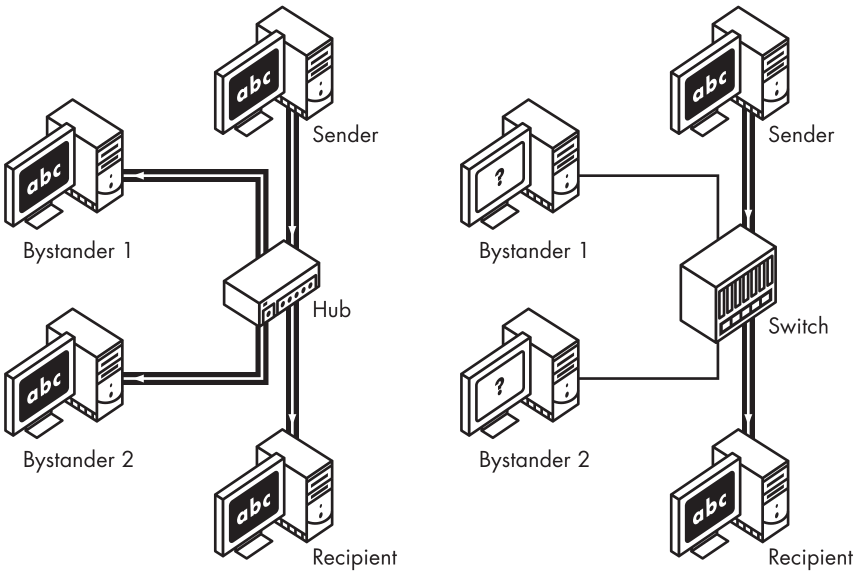


Figure 5-5: Hubs versus switches in local networks

I'm guessing that at this point you're asking yourself, Where the heck are you going with all this? What do modems have to do with information disclosure? What significance do serial links have in this context? How do Ethernet networks fit in? And what the heck are blinkenlights?

Glad you asked. I am about to get there—to the last question, that is.

Blinkenlights in Communications

Historically, almost all refrigerator-sized computers were equipped with numerous prominently exposed diagnostic interfaces. These included arrays of tiny lights that displayed, among other things, certain arcane properties of the internal state of a machine, such as internal registers or flags of the core processing unit or an indication of whether the cat living underneath had been fed today. As computers became more reliable and compact, and an average user no longer had to understand the machine's internals in order to use it efficiently, the lights started to disappear from many devices. Ever-increasing clock speeds also contributed to the decline—most of the time it was no longer possible for humans to get any meaningful information from such a visual signal that would change thousands or millions of times every second.

Yet, the lights prevailed in some applications; for example, almost all networking devices feature light-emitting diodes (LEDs) on their front or back panel. These provide link diagnostics, such as an indication of whether a particular module or socket is functioning properly, a party is connected, data is being transferred, and so on. The lights are not merely a diagnostic tool either; their hypnotic patterns have strange appeal, and their mystery plants seeds of uncertainty, fear, and respect in the hearts of lay people who enter the realm of the server room.

The term *blinkenlights* or *blinkenlichten* has been used to describe the much-adored institution of diagnostic LEDs on computer equipment ever since the dark ages of computing, bathing the computer geek in the soothing green light during those long, lonely nights spent at the terminal. It came from an amusing prank note in mock German (itself a spoof of another, noncomputer joke from WWII), displayed some time in the 1950s at IBM laboratories. The note later propagated into a majority of server rooms and computer science laboratories across the world and went like this (as quoted from Eric S. Raymond's *Hacker's Dictionary*):

```
ACHTUNG!

ALLES LOOKENSPEEPERS!

Alles touristen und non-technischen
looken peepers! Das computermachine
ist nicht fuer gefingerpoken und
mittengrabben. Ist easy schnappen
der springenwerk, blowenfusen und
poppencorken mit spitzensparken.
Ist nicht fuer gewerken bei das
dumpkopfen. Das rubbernecken
sichtseeren keepen das cotton-pickenen
hans in das pockets muss; relaxen und
watchen das blinkenlichten.
```

Communications equipment is one of the last domains in which blinkenlights prevail and prosper. But that's not all. Almost all these devices use serial lines for communications. And, for the sake of simplicity and aesthetics, "activity" LEDs are sometimes wired almost directly, through a simple driver circuit, to the transmit or receive line of the device. Curtain falls.

The Implications of Aesthetics

It took decades for the problem to be discovered, and once it happened (in 2002), it struck us all as so obvious and trivial we wanted to bang our heads on the keyboard a couple of times.

Joe Lughry and David A. Umphress, in a research paper titled "Information Leakage from Optical Emanations,"[4] discovered a new type of signal-disclosure scenario in certain types of network equipment, most often modems. They concluded that someone observing these lights could go beyond simply watching the magic lights with the naked eye.

LEDs, unlike incandescent bulbs, usually have short rise and fall times, meaning that they turn on and off almost instantly. That's not surprising; after all, high-end LEDs are used to control fiber-optic links and some other optoelectronic communication channels. As such, the blinking of an LED hooked up to a serial data transmission line can actually often mirror single bits of the transmission as it occurs on the wire. Given a way to record this activity at a sufficient speed, it should be possible to retrieve this information, from at least as far as you can see the tiny blinking light on a device with the naked eye (or with a telephoto lens).

This research caused some stir in the industry; it was eventually also both downplayed and overhyped, and hence a great deal of confusion ensued, and very little has changed. The paper resulted in many conflicting reports, but its basic premise is simple and truly beautiful. The beauty of this technique is that it is trivial to devise such a device to receive the signal: the equally cheap and popular counterparts of LEDs—photodiodes and phototransistors—are easy to acquire and equally easy to interface with the computer. And the exposure zone, unlike most of the TEMPEST activity we discussed in Chapter 3, is not merely the subject of urban legends and pure laboratory results, but can be directly observed and measured.

In the course of their research, the authors performed a set of experiments to verify that the signal could be successfully acquired from as far away as 20 meters (just under 100 feet) without the need for additional digital signal conditioning. And common sense suggests that this might actually be an understatement, especially when good optics are used. (The authors used a 100 mm focal length, f/2.0 lens for the test, but a much better telephoto lens is commonly available to many midrange SLR (single lens reflex) photography amateurs. Those who are willing to part with their money can buy a superb-quality lens with a focal length of as much as 1,200 mm.)

The paper takes a defensive stance in several cases, and a careful reader might be tempted to conclude that some of the devices classified are not vulnerable to the problem. In particular, some of the Ethernet devices may exhibit a more subtle variant of the vulnerability, as you'll see in the prevention section later in this chapter. But first let's peek at the problem with our own (computerized) eyes, shall we?

Building Your Own Spy Gear . . .

The simplicity of building a snooping device makes it quite tempting to do so. This section contains several suggestions and rough schematics on how to build and connect such a device to an ordinary computer. Although the circuit is not particularly complex and does not require a master's degree in soldering and a printed board circuit design software, a minimum level of proficiency in electronics is desirable, as is a dose of common sense. Although external interfaces of today's computers are fairly robust and foolproof, there is always the risk of damaging equipment when attaching home-brew devices in a really innovative way, in a brief moment of insanity. It's happened to the best of us.

The baseline design is extremely trivial. It calls for a single phototransistor (a component consisting of a transistor driven by a built-in photodiode), a regular low-power NPN (Negative-Positive-Negative) transistor to amplify the signal a bit further (not always necessary), and a set of potentiometers (perhaps in the range of 10 kΩ just to have enough flexibility) to experimentally pull down the voltage and control the circuit's sensitivity and threshold points. There are no particular requirements for the components, although your mileage will vary depending on which ones you use. Be sure to select a phototransistor that has a decent response in the visible light range, though all cheap ones should work. (For reference, a green LED emits a wavelength of approximately 520 nm.)

A sample circuit design is shown in Figure 5-6.

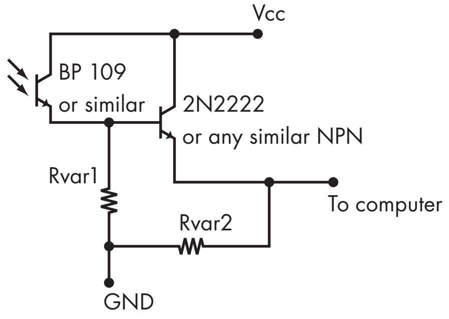


Figure 5-6: A simple receiver circuit

The circuit has an optimal running voltage of approximately 5V and a low maximum current: a power supply capable of delivering perhaps 10 to 50 mA is more than enough. A word of warning: If you use a supply capable of delivering a higher voltage, you will risk damaging the port or the computer; likewise, if you use a more powerful supply and do not prevent higher current from flowing through the circuit.

NOTE *Setting Rvar1 or Rvar2 to a very low resistance may short the circuit. If you want to fiddle with the knobs mindlessly, it might be a good idea to add a fixed resistor to limit the current drain.*

You must shield the phototransistor from external light sources—for example, by enclosing it in an opaque tube. Because the phototransistor has no focusing mechanism, it is not likely to pick up more distant signals (other than ambient light). Thus, for initial tests, it is a good idea to cover it entirely to simulate darkness and then put it by an LED to excite the circuit. You can also connect another LED temporarily between the GND and the output line to test the circuit. The test LED should light up when the sensor is directed at a light source, but otherwise be fairly dark.

. . . And Using It with a Computer

If the circuit with a test LED hookup works so far, well done; you have built a fancy TV remote tester. Because generic, cheap phototransistors are eager to pick up infrared light, your creation should "translate" IR (infrared) into visible light, but that's about all the fun stuff it will do. To make it a bit more useful, you need to interface the circuit with the computer. A good way to do so is through a line printer interface, LPT, if your computer has one. Unfortunately, this wonderful hardware hacker's tool is being dropped from some of the more compact and fancy designs.

Although initially designed to be unidirectional (for output only), the LPT interface provides a number of status feedback lines, such as "paper out," "busy," and "acknowledgment," that were intended to provide a means for the printer to complain about problems. You can easily read the data that issues through this interface by accessing port 0x379 (the LPT1 status register) on a PC-compatible system. By hooking the circuit to a parallel port, you can easily transmit information back to the computer. Although you might want to connect the circuit to a different interface, LPT is much faster than, say, RS-232, and you won't have to cope with any mundane protocols, signaling schemes, or unusual voltage levels. Too, unlike USB and some other current solutions, you do not need special controllers to implement a fairly complex protocol to even be able to talk to your PC.

NOTE *Although LPT also offers bi-directional operation modes (ECP or EPP), it is usually pointless to attempt to use this functionality for such a simple task. In the unidirectional mode, four bits are available for input, more than enough for this application; switching to bi-directional modes such as EPP or ESP provides an extra four bits.*

It is up to you to choose the status line to use. Table 5-3 shows a pin layout of the DB25 connector used for a printer port. The rows shaded gray can be used for input.

To interface the circuit with this port, you can simply connect the ground reference point on the connector with the one used in your circuit and then hook up the output line to any of the five pins. (Remember to disconnect the LED used for diagnostics first.) Next, monitor the status port as you first expose it to light and then cover the sensor. In either case, the value read depends on how you hooked up the circuit; the exact value does not matter, as long as the two values are different.

Table 5-3: LPT pinout

LPT Port: DB25 Pinout (Standard Mode)		
Pin	**Name**	**Function**
1	$\overline{\text{Strobe}}$	Control output bit 0
2	D0	Data output bit 0
3	D1	Data output bit 1
4	D2	Data output bit 2
5	D3	Data output bit 3
6	D4	Data output bit 4
7	D5	Data output bit 5
8	D6	Data output bit 6
9	D7	Data output bit 7
10	$\overline{\text{ACK}}$	Status input bit 2
11	Busy	Status input bit 3
12	Paper Out	Status input bit 1
13	Select In	Status input bit 0
14	$\overline{\text{Autofeed}}$	Control output bit 1
15	$\overline{\text{Error}}$	Status input (unused)
16	$\overline{\text{Init}}$	Control output bit 2
17	$\overline{\text{Select}}$	Control output bit 3
18	GND	Ground (0V)
19	GND	Ground (0V)
20	GND	Ground (0V)
21	GND	Ground (0V)
22	GND	Ground (0V)
23	GND	Ground (0V)
24	GND	Ground (0V)
25	GND	Ground (0V)

Because chip logic requires somewhat different input levels than your test LED diode, you might have to tweak the `Rvar2` until you get distinct readings from the port when you cover the sensor, and when you expose it to light. To accomplish this, it is best to be able to monitor the port in real time on the computer itself.

The way you can monitor the state of the port will depend on the operating system and the programming language you are using. If you're using C, the function used to read the value off a port is `inb(port)`, so in this particular case you would issue `inb(0x379)` and check the return value. In other languages, it is likely to have a similar name. (Try looking for `in`, `inport`, `readport`, and so forth.) Also, Windows users may find the built-in "debug" utility and its "i" (port read) function quite handy.

NOTE *On some systems, such as Linux, you might need to request that the system give you permission to access a specific port first. Consult the documentation for* `iopl(3)` *or a similar call for more information.*

At this point, you are ready to go. You can choose to point your probe at any LED on a device, adjust the sensor based on its brightness, and start reading alternating patterns of light and dark signals, as you discover how they correspond to the exchanged information, if at all.

NOTE *If you're curious, you might try to examine the brightness of the indicator diode, not only a binary representation of its state. It might turn out that even though a specific LED is not intended to directly map a signal on the serial line to its blink patterns, there is some analog cross talk between circuits, and the serial line signal will have some influence on the brightness. A cheap analog-to-digital converter such as TLV571 from Texas Instruments is just asking to be used this way.*

You can use this approach to sample the frequency of less than 1 million bits per second, which should suffice for capturing transmission on many interfaces, but not necessarily on Ethernet ports (which transmit at least 10 million bits per second). Past this capture capacity, your LPT port will likely reach its physical throughput limits, but do not despair: as long as the sensor (phototransistor) can switch at the rate sufficient to capture communications in question, you still have an option. Remember that LPT is a parallel port. To reach faster capture speeds, such as the one needed for Ethernet, combine a trivial clock, a counter circuit, and a set of sample-and-hold latches (such as 74LS377) to sequentially store data between the port read attempts on the computer side. You can accumulate this information for a short period of time and then, by using more than just one status pin (or by switching the port to bidirectional mode), easily send several bits—samples—to the computer, in a single burst, in one read cycle, thus improving the read rate four- or eightfold.

I'll spare you a further, perhaps needless, excursion into the world of electronics. If you want to toy with the idea of high-speed or analog sampling, or perhaps just get your kicks from soldiering stuff together and hooking it to a computer, you might want to take a look at my fairly comprehensive introductory tutorial under the thin disguise of a computer-controlled robot design project. You should be able to find it at http://lcamtuf.coredump.cx/robot.txt.

And now, for those with interests that lean more toward practical security: a brief discussion of how to address the issue, short of covering all LEDs in the office with duct tape.

Preventing Blinkenlights Data Disclosure—and Why It Will Fail

The easiest solution to the problem, and one suggested by the original research, is *pulse stretching*—a practice intended to distort the blinks on an indicator by prolonging some of them, thus making any practical data recovery seemingly not feasible. *Pulse stretching circuits* are a group of fairly trivial devices that extend the duration of an encountered "high" input signal for an additional period of time. Most basic pulse stretcher design relies on a capacitor that charges in the presence of an input signal and then discharges slowly. This capacitor is connected to a *binary discriminator,* which is not a nickname for a vicious wrestling champion, but rather a device that converts analog data into binary output by applying a particular threshold (outputting a voltage for logical 1 for all input voltages above *n*, and 0 for all input voltages below). In this case, it uses a certain capacitor charge level as the discrimination point.

More advanced and reliable designs, including purely digital circuitry, are also common, and all can be used in hubs and switches to make LEDs nice to look at. Without them, the high-speed blinking at way more than 50 cycles per second (considered the limit on our ability to perceive flicker), would usually result in our seeing the lights as dim but seemingly constant. A discriminator causes the LED to be driven by 1 more often than by 0 by extending the duration of each 1 pulse. This makes the LED light brighter and blink less often. Figure 5-7 shows the behavior of such a pulse stretcher: a single spike (single 1) is stretched to last three times as long, whereas all 0s are left as they are.

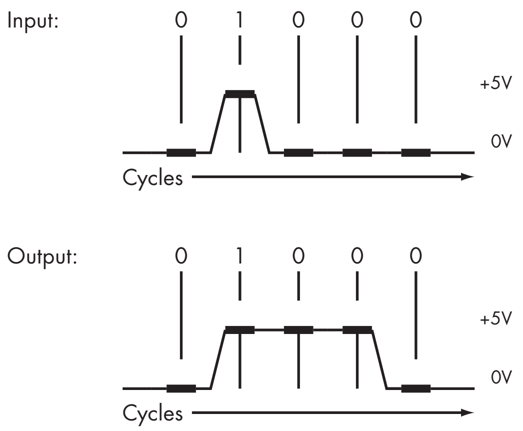


Figure 5-7: Pulse stretcher behavior, 3x

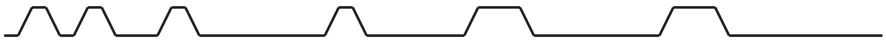
While their primary purpose is aesthetic, as I have mentioned, this also seems to be a good way to solve the problem of light emissions information disclosure, by letting the attacker deduce only certain general properties of the traffic. Thus, at best, the attacker can figure out only general properties of the traffic, such as when something is being sent and when it is not.*

What seems to be a good solution, however, is not always. Consider the following sample data and the corresponding serial line signal:

Data:

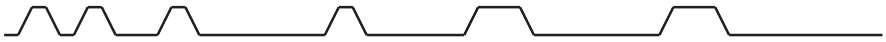
0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0

NRZ signal:

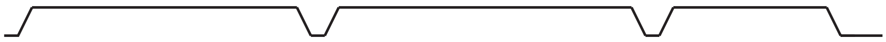


Assume the signal is processed using a 5x pulse stretcher that makes every 1 last for five additional cycles. (The original paper suggests a safe limit of 2x, but we'll exaggerate to make a point.)

Original signal:



Level activated stretcher (5x):



Data we can read out of LED after stretching:

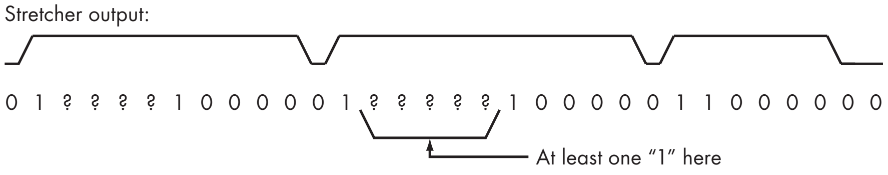
0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0

Although it might appear that almost all important information has been lost when compared with the input signal we want to intercept, it is possible to recover much of it by making four important observations:

- Obviously, all areas where the stretcher output is zero must have been zero in the original signal.
- Each stretched run of 1s must have been triggered by 1 at the starting location in the original stream.
- Each run of L 1s must have originally contained at least one 1 for every *N* cycles, where *N* is the stretch factor for this circuit; otherwise, there would be gaps in the run. The count of 1s in a block of data represented under a single stretch of 1s in output is greater than or equal to L/*N* rounded up.
- Every run ends after exactly *N*-1 zeros in the original stream. We know that these zeros must have been preceded with 1; otherwise the run would have ended sooner.

* This, technically speaking, is still an attack venue, per the discussion in Chapter 1, yet it is considerably less effective and practical, for we only get a rough idea of what is going on, not a copy of the data.

By applying this knowledge to the previous example, we can reconstruct most of the original data, as follows:



In the previous fairly realistic example, fewer than 9 out of 32 bits of data were lost due to pulse stretching and cannot be conclusively reconstructed (marked with question marks in the graphic). Thus, we recovered 99.999988% of the potential search space. We must guess at the remaining data, which (especially if the data snooped is regular English text, such as email) is rather trivial to reconstruct compared to the starting point. The authors of the research suggest that even N = 1.5 or N = 2 "on" time pulse stretching is sufficient to obfuscate the data, but this is not necessarily so.

The previous reconstruction scheme works with stretches of 0s or 1s. Some links use return-to-zero (RZ) encodings (such as the Manchester scheme mentioned earlier), and because the signal is constantly alternating there, the 2x stretching might indeed be sufficient to obfuscate all data. However, this is only true if the LED is driven by a signal prior to initial internal decoding to NRZ—which, in most situations, is not the case. In fact, applying pulse stretching to RZ-encoded signal is often a silly idea in that the LED would be on all the time; hence there seems to be no point in doing that in the first place.

As noted previously, an additional problem stems from the quality of the pulse stretcher and its susceptibility to interference from other internal circuits: LED voltage fluctuations that result in slight brightness changes during a "stretch" period might disclose some information. Capacitor-based solutions, in particular, can fall into this category.

Thus, some systems, particularly Ethernet devices known to deploy pulse stretching, can be partly vulnerable to attack, even though the original paper discussed earlier concluded that there is no direct correlation between the transmitted data and the behavior of an LED, based on the observation of a recorded blinking pattern using an oscilloscope.

The optimal solution, particularly with other types of encoding, or when pulse stretching is not desirable for some other reason (for example, if the designer wants to avoid making the LED light appear constantly for the time of a transmission) is to sample the line at a fairly low frequency (for example, 20 Hz) and latch it to a register that holds it until the next sample and that also controls the LED.

And, now, back to plain English.

Food for Thought

Other than network device LEDs, plenty of other, equally interesting light emissions leak scenarios can be found, although the amount of information disclosed can be significantly lower. For example, consider disk activity LEDs. Of course, disk communication is not using serial signaling; instead, portions of data, ranging from bytes to 32-bit words, are sent simultaneously using a set of signal lines. And, although the LED is usually attached to indicate only a state of a specific control line, it is still possible to deduce many aspects of system activity by measuring seek times or the amount of data stored and read. (Depending on what the LED is actually attached to, it may be possible to measure either or both.) Although it's unlikely that this information would give an attacker any immediate advantage, certain induced I/O activities can be combined with hard-disk drive LED observation to draw interesting conclusions, although I am unaware of any research in this area.

Other potential attack venues involve many USB devices and other proprietary interfaces. As mentioned earlier, USB is a serial bus, and some USB appliances do have activity indicators.

Various other unusual and arcane information-disclosure venues have also been proposed, partly researched or at least toyed with. These include measuring the acoustic effects of recharging capacitors as the CPU consumes various levels of power depending on the executed instruction[5] or measuring a black box device by analyzing its power consumption with the help of statistical analysis.[6] Once again, no truly comprehensive research has been done in the area of disclosure channels other than classic EMF (electromagnetic field) emanations—and it appears to be a good idea to investigate. Best of luck. :-)