Introduction To Computer Networking

Alex S.*

1 Introduction

1.1 Serial Lines

Serial lines are generally the most basic and most common communication medium you can have between computers and/or equipment. They generally consist of relatively cheap wire sending bits across at relatively slow speeds. That's all there is to them.

Because of their simplicity, serial ports are available on pretty much all sorts of hardware (except many new laptops). Many port and jack configurations are supported (there are round jacks (DIN-8), 9 pin jacks (DB-9), 25 pin jacks (DB-25), etc.) Most times, only a few of the wires are actually used. In fact, the most basic serial port communication between two computers uses only 3 wires: (transmit, receive, and ground). Sometimes an RJ-45 is used for serial communications—and there are various standards on how pins map to which wires.

The transmission can use hardware flow control or software flow control. The hardware flow control generally requires more wires to be connected. Software flow control is harder on the CPU.

1.2 Null Modem

There is a concept of a 'null' modem. This is basically when the serial cable isn't connected to a device, but to another computer. It is the same wire as a regular serial cable, except a few wires have been crossed over... namely the send/receive wires—along with a few hardware flow control wires if those are used).

1.3 Terminal Devices

In the UNIX world, there is a thing called a 'terminal'. It is basically a screen/keyboard connected to the 'computer' via a serial cable. You've seen a similar idea when you use Telnet (or SSH) to log into Solaris/Linux computers (except connection doesn't travel over serial cables—but it still uses terminal devices).

*alex@theparticle.com

In fact, there is a program called `getty` which presents a login prompt on a particular serial port. Something you could try: connect two computers via a null-modem cable, do a `getty` on the serial port on one computer, and on another computer, you can cat the serial port device to read/write it and actually be logged into the other computer through a serial port (and obviously use all the command line programs).

UNIX has a bunch of *virtual* terminals, and when the system starts up, `getty` is executed on a bunch of those terminals. That's why you see a login prompt on the screen (your screen/keyboard are just treated as command line terminals as far as the computer are concerned).

There is a program called `stty` which you can use to adjust the properties of your terminals. For example, to change the speed of a serial port, or to enable/disable control flow options, specify special characters, number of rows/columns, etc.

1.4 IP over Serial Lines

It turns out, you can use TCP/IP over serial lines. This shouldn't come as a surprise, as many of you might still be using dial-up Internet. Basically, there are two protocols that do the job: SLIP and PPP.

Both have the basic idea of encapsulating IP packets, and transmitting them (with some flow control) over the serial line. The process would work something like this: the dial-up server is setup to pickup a phone line when it rings and start a `getty` on it. You dial up (ATDT), see a terminal (or your dial-up networking software sees the login), it proceeds to login (via the simple UNIX like login prompt), it then runs PPP or SLIP to start a remote protocol engine—which then configures the already established serial link for transfer of IP packets.

1.4.1 SLIP

SLIP (Serial Line Internet Protocol) is probably the simplest one, and at one point, you'd use it if you wanted to setup a dial-up connection to the Internet from your Linux box.

1.4.2 PPP

PPP (Point-to-Point Protocol) is a huge bloated standard that is capable of quite a bit more than just sending packets. When you dial into the your ISP, this is what you're using.

1.4.3 Others

There may be other such encapsulation protocols used—so you can't just dial up and expect things to work (via standard networking configuration). Some ISPs require you to have special "dialer" software that enables your Windows box to use *other* encapsulation protocol. AOL was notorious for that.

1.5 TCP/IP

TCP/IP is a whole bunch of various protocols stuck into one:

1.5.1 IP

Internet Protocol, or IP, is used to transport RAW data from one machine to the next. This involves hopping from computer to computer until the data gets there.

1.5.2 ICMP

Internet Control Message Protocol, or ICMP, is used as a helper protocol for IP. It provides assistance with error messages, routing, and echo requests.

1.5.3 ARP

Address Resolution Protocol, or ARP, provides services to resolve logical network addresses to hardware network addresses. Generally, figuring out which IP addresses correspond to which MAC addresses.

It basically screams out to the network "does anyone know the hardware address for this IP". Then some machine (if it is alive) will respond, "Yes, here I am."

1.5.4 UDP

User Datagram Protocol, or UDP, sends a packet from one *program* to another *program*. It's unreliable, and connectionless.

1.5.5 TCP

Transmission Control Protocol, or TCP, sends a stream of data from one *program* to another *program*. It is reliable, and connection oriented (establishes a 'session').

1.6 Layered Model, ISO/OSI, TCP/IP

A bit unrelated, but has to be mentioned when discussing protocols. Theer are 7 layers of the OSI model:

1. Physical Layer (Physical cable, medium, air)
2. Data Link Layer (Transmit/recieve packets, resolve hardware addresses)
3. Network Layer (Routing, accounting).
4. Transport Layer (Guarantee end-to-end data transfer—from machine to machine)
5. Session Layer (Authentication/Authorization)

6. Presentation Layer (Data compression, and other data conversion)

7. Application Layer (Provide end-user services, like e-mail)

There is an interesting quote from *UNIX System Administration Handbook*: "Some think a financial layer and a political layer should be added to these. There is no magic about the number seven; seven committees were involved in the specification, and one layer was created for each."

In any case, there have been some attempts to implement the OSI model, and some sorta actually worked, etc., but it never managed to catch on.

The TCP/IP can be thought of consisting of four layers:

1. Link Layer (Network hardware and device level)

2. Network Layer (Communication, addressing, and routing)

3. Transport Layer (Communication among programs on a network)

4. Application Layer (End-user application programs)

The TCP/IP does most of the things the OSI model does, except it does it in four layers, and is actually a lot more robust and real than the OSI idea.

TCP/IP is setup in such a way that it doesn't really matter which network is used underneath. The Link Layer can use physical wire, wireless, serial lines, etc.

1.7 Segmentation and Packets

TCP/IP breaks up all traffic into packets. A packet is just a structure with a header, and some data. When referring to a packet on a LAN, we often use the term "frame" (even though when a packet travels in a frame it has a bit more info attached to it).

Every network has MTU, or Maximum Transfer Unit. This is the maximum size of a packet that can traverse the network. The IP layer, breaks up your data into packets of MTU size, transfers those to the destination machine, where they are reassembled.

Every packet, obviously has to have source IP, destination IP, size of data, etc. When it is encapsulated in some other protocol, those are also included. For example, a UDP packet carrying 100 bytes of data on an Ethernet would have:

- 14 bytes Ethernet Header (includes things like MAC addresses)
- 20 bytes IP Header (includes things like source/destination IP addresses)
- 8 bytes UDP Header (includes things like source/destination port addresses)
- 100 bytes of user data
- 4 bytes Ethernet Trailer (terminates the frame)

So while we were just sending 100 bytes via UDP, over the Ethernet, that sent 146 bytes.

1.8 Internet Addresses

Since we are primarily concerned with TCP/IP, let's discuss network addresses. An IP addresses is just a 32 bit number, written out with each byte (0-255) separated by dots. For example, 192.168.0.1 is an IP address.

There are several classes of networks (and thus addresses). Each address belongs to a class. Part of the IP address indicates the network address. For example, N.N.N.H has 3 bytes dedicated to the network address, and 1 byte to the host address. Various classes of networks:

- Class A: Starts with 1-126. Major networks. N.H.H.H
- Class B: Starts with 128-191. Large networks. N.N.H.H
- Class C: Starts with 192-223. Small networks (easy to get). N.N.N.H
- Class D: Starts with 224-239. Multi-cast addresses.
- Class E: Starts with 240-254. Experimental addresses.

In any case, most networks that most people are concerned with are B and C.

1.9 Routing

Routing is the process of directing a packet through the network to the destination computer. You can view it as asking for directions. You are sent to deliver something to an unknown city. You ask for directions, and someone may point you towards a general direction (or towards someone who may known the general direction). You soon get to the city, and you start asking for more specific information concerning the street address, etc. Same idea works for routing.

For routing to work, the operating system maintains a routing table. It is just a table of network masks and destinations. Whenever the kernel sees a packet, it figures out which network it corresponds to, and sends the packet to that network—that's all there is to it. There are also 'default' routes; when everything else fails (you don't know what to do with a packet) you send it off to the default route.

There is static routing and dynamic routing. Static means that you setup the routing table manually, at boot-time, etc., and it stays around. For small networks that is quite efficient.

Dynamic routing involves the use of routing daemons like `routed` or `gated` to discover the network topology and how to reach distant networks. There are many approaches to this, which we won't get to at this time.