Crossroads
The ACM Student Magazine

Under the hood of the Internet:

An overview of the TCP/IP Protocol Suite

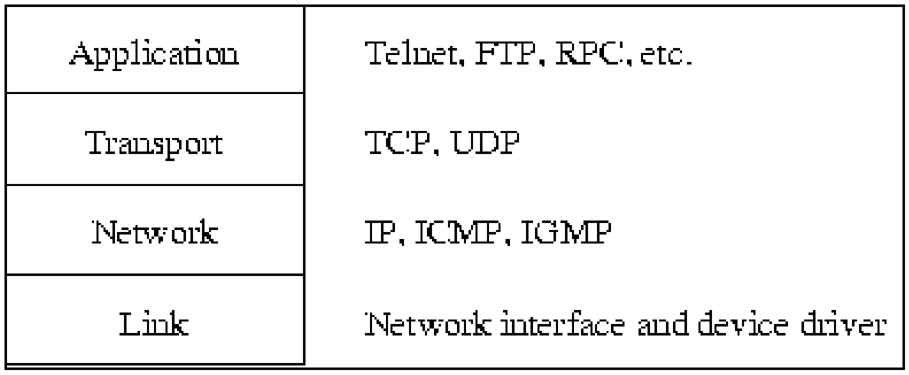
by ***Jason Yanowitz***

Introduction

Most people use the Internet with little idea of what is actually happening when they ftp a file, read a news article, or telnet to another machine. Vague notions like ``bandwidth'' circulate, but there is little concrete knowledge. The goal of this article is to provide the reader with an introduction to TCP/IP, the protocols used on the Internet. We start with the typical overview of the protocol stack and then provide more detail on each layer. This article is equivalent to the first day of a course on TCP/IP networking - it provides a broad overview of the issues with few of the specifics. We leave out whole chunks of the protocol suite, most of the theory behind them, and discussions of their implementation. In the conclusion, we suggest some books for interested readers.

The Protocol Stack

Before we examine each layer of TCP/IP, we should cover a few basic concepts. At its most basic level, a computer network is simply a series of connections between computers which allow them to communicate. The content, scope, size, speed, and reliability of the network varies depending on its protocols and implementation. Protocols are pre-established means of communication. The term TCP/IP (Transmission Control Protocol/Internet Protocol) actually refers to a whole family of protocols, of which TCP and IP are just two. Figure 1 contains the standard ``stack'' diagram of TCP/IP. Rather than make protocols monolithic (which would mean ftp, telnet, and gopher would each have a full network protocol implementation, including separate copies of kernel code for the devices each protocol uses), the designers of TCP/IP broke the job of a full network protocol suite into a number of tasks. Each layer corresponds to a different facet of communication. Conceptually, it is useful to envision TCP/IP as a stack. In implementations, programmers often blur the layers for increased performance.



Application	Telnet, FTP, RPC, etc.
Transport	TCP, UDP
Network	IP, ICMP, IGMP
Link	Network interface and device driver

Figure 1. The Layers of the TCP/IP Protocol Suite

The first, the link layer, is responsible for communicating with the actual network hardware (e.g., the Ethernet card). Data it receives off the network wire it hands to the network layer; data it receives from the network layer it puts on the network wire. This is where device drivers for different interfaces reside.

The second, the network layer, is responsible for figuring out how to get data to its destination. Making no guarantee about whether data will reach its destination, it just decides where the data should be sent.

The third, the transport layer, provides data flows for the application layer. It is at the transport layer where guarantees of reliability may be made.

The fourth, the application layer, is where users typically interact with the network. This is where telnet, ftp, email, IRC, etc. reside.

Packets are the basic unit of transmission on the Internet. They contain both data and header information. Simply put, headers generally consist of some combination of checksums, protocol identifiers, destination and source addresses, and state information. Each layer may add its own header information, so it can interpret the data the lower layer is handing it. In Figure 2, we see a sample Ethernet frame. This is the product of a packet which has gone from that application layer all the way to the link layer. Each layer takes the previous layer's packet, viewing almost all of it as data, and puts its own header on it.

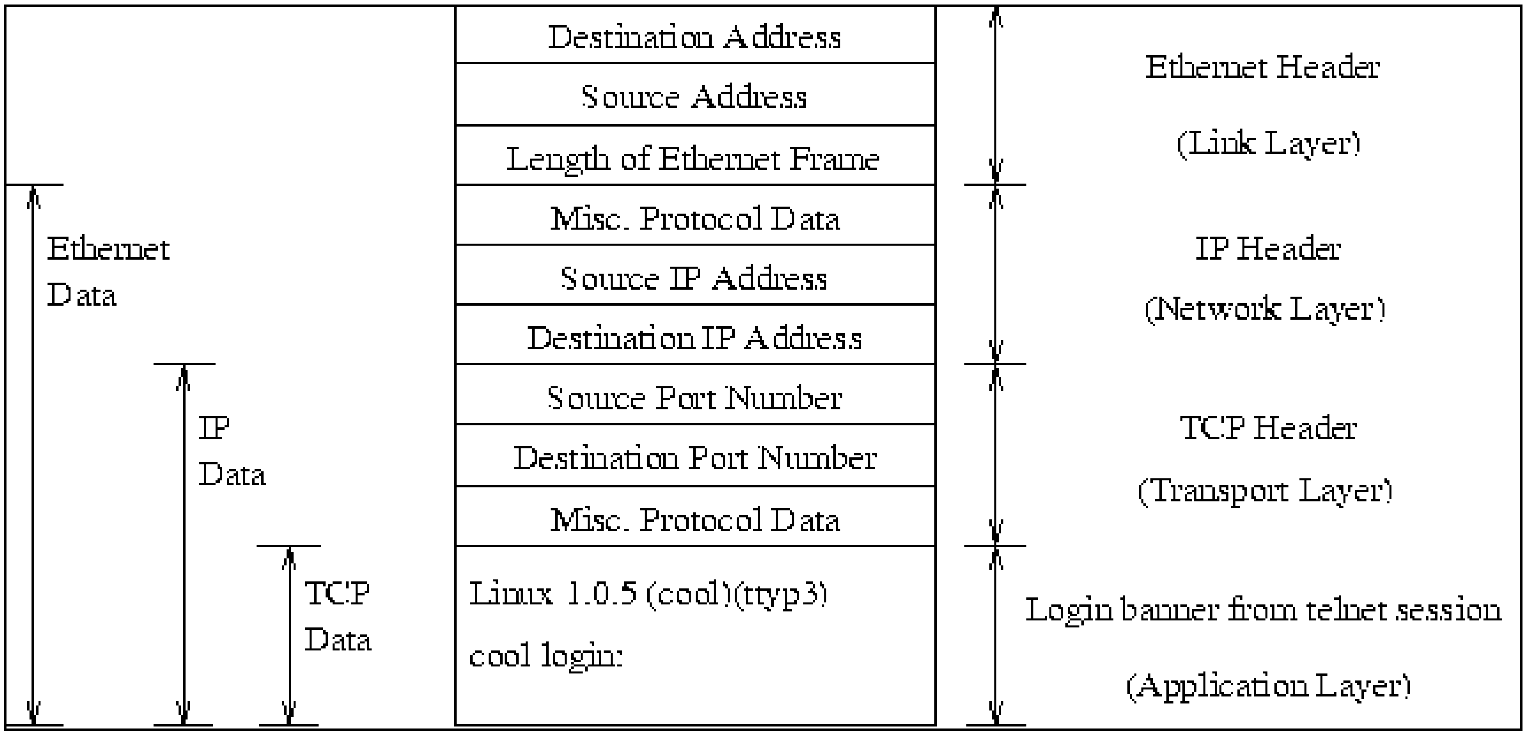


Figure 2. A Sample Ethernet Frame

We will now examine each part in turn, with a particular emphasis on the network and transport layers. In examples that follow, we'll refer to two machines: swell.cs.umass.edu and cool.alaska.edu. swell is the machine we are on, cool is the destination computer. We assume cool and swell are on Ethernets at their respective organizations. Most of our examples assume an Ethernet, but could work with any kind of network (e.g., token-ring).

The Link Layer

The link layer is the simplest layer to understand. Composed of the network hardware and the device drivers, the link layer is the lowest level of the protocol stack. When receiving data from the network, it takes packets from the network wire, strips away any link layer header information, and hands it off to the network layer. When transmitting data onto the network, it takes packets from the network layer, sticks a link layer header on them, and send them out over the wire.

The benefit of separating out the hardware layer is that protocol implementors only have to write the network layer once. Then they provide a common interface to the network layer by writing different device drivers for each kind of network interface.

The Network Layer

This is where the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP), among others, reside. ICMP is used both to provide network reliability information and by utilities like ping and traceroute. IP is used for almost all other Internet communication. When sending packets, it is figures out how to get them to their destination; when receiving packets, it figures out where they belong. Because it does not worry about whether packets get to where they are going nor whether they arrive in the order sent, its job is greatly simplified. If a packet arrives with any problems (e.g., corruption), IP silently discards it. Upper layers are responsible for insuring reliable reception of packets. We refer to IP's behavior as ``stateless'' or ``connectionless'' because the existence of previous or future packets is irrelevant when processing the current packet. We could unplug the network wire, wait a minute, plug it back in, and IP would never know the difference.

IP is able to get packets to their destinations because every network interface on the Internet has a unique, numeric address. Oddly enough, these numbers are called IP addresses. Notice, every *interface* has its own address. If a machine has multiple interfaces (as is the case with a router), each one has its own IP address. The Internic is responsible for assigning sets of addresses to organizations, thereby insuring uniqueness.

Because it's a pain to refer to machines with strings of numbers, the designers of TCP/IP allowed network administrators to associate names with IP addresses. Although this has nothing to do with the IP layer per se, we feel this is useful material. Originally, every host on the Internet maintained its own complete copy of this database (on Unix systems, it's in /etc/hosts). However, as the Internet reached its current size, this soon became unwieldly -- both in terms of raw size and the administrative nightmare of updating it. And so was born the domain name system (DNS). It is a distributed database of IP addresses and their natural language names, called host names. In fact one IP address can have multiple names associated with it. When a network administrator adds a new machine to her network, she is responsible for updating her organization's nameserver table. Her changes quickly propagate. All communication with a machine is done via IP numeric addresses, so the hostname for a machine is only used at the beginning of a connection.

The steps IP takes to send a packet are simple: based on its IP address, figure out how to get it there and send it on its way.

Deciding out how to get the packet there, aka routing, is the critical task for IP. Fortunately, swell doesn't have to know how to get a packet all the way to Alaska, it just needs to figure out which local router is responsible for getting packets to Alaska. A router differs from a typical machine on the net because it has at least two network interfaces -- this allows it to connect to two or more networks. For a small organization, there will typically be a local network (e.g., Ethernet) and then a leased-line link to the Internet. The organization's router is connected to both the local network and the Internet link. All packets bound for the Internet are sent to the router, which then puts it on the leased line, bound for the next router.

Each router only needs to know about the routers to which it is connected. Those routers then know about all the routers to which they are connected. This allows swell's local router to say, ``Well, all packets bound for the West go to MIT, so I'll just send it there and let MIT figure out what to do next.'' MIT puts it on a T3 line to Cleveland, from there it goes to Chicago, San Francisco, Seattle, and into Alaska, where it goes from the organization's router to the Ethernet interface on swell. The router at each hop is only concerned with where to send it next. It doesn't try to determine the full path which the packet will take.

To determine where a given packet will go next, machines on the Internet maintain routing tables. They consist of three major items: addresses of routers, addresses they can handle, and the interface to which they are connected. In the case of a machine on a local net (like cool.cs.umass.edu), it probably has three entries: one for the loopback interface (which allows a host to connect to itself), one for the local network and a default entry.

The local network entry lets IP know that the machine is directly connected to a certain set of IP addresses. Rather than try and route those packets, IP figures out the hardware address of the Ethernet interface to which the IP address corresponds and sends the packet there. With this entry, cool is essentially the router, the addresses it can handle are all the IP addresses on the local net, and the destination interface is an Ethernet card on the local net.

The default entry says ``for all other addresses, send it to this router.'' Instead of trying to deliver a packet for cool.alaska.edu to a machine on the local net, cool sends it to the router's interface, saying, ``Here, I don't know where this goes, you figure it out.'' The router then looks at its table, sees it doesn't have a direct connection to cool, so sends it to its default destination, MIT. And so the process continues.

At this point, the reader should have a rough idea of how packets are transmitted on the Internet. When receiving data, IP takes the packet from the link layer, checks for any blatant corruption, and hands the packet to the proper process at the transport layer. If there is any problem with the packet, IP silently discards it because it doesn't have to worry about whether a packet reaches its destination.

We have left a huge amount out of this picture. Here are just some of the issues we're ignoring: packet fragmentation, netmasks and other routing tricks, network error handling, and the interactions between the network and transport layer.

The Transport Layer

There are two protocols at the transport layer: the transmission control protocol (TCP) and the user datagram protocol (UDP). TCP provides end-to-end reliable communication and UDP doesn't. UDP is as unreliable as IP, but allows people to write user level software that creates its own packet formats, which is particularly helpful if you want to write new protocols, don't have the kernel sources, and don't want the overhead of TCP.

TCP creates a ``virtual circuit'' between two processes. It insures that packets are received in the order they are sent and that lost packets are retransmitted. We won't go into the details of how it works, but interactive programs like ftp and telnet use it.

So far we have discussed addressing on the host level -- how to identify a particular machine. But once at a machine, we need a way to identify a particular service (e.g., mail). This is the function of ports -- identification numbers included with every UDP or TCP packet. TCP/IP ports are not hardware-based. They are a just a way of labeling packets. A process on a machine ``listens'' on a particular port. When the transport layer receives a packet, it checks the port number and sends the data to the corresponding process. When a process starts up, it registers a port number with the TCP/IP stack. Only one process per protocol can listen on a given port. So while a process using UDP and one using TCP can both listen on port 111, two processes that both used TCP could not. There are a number of ports which are reserved for standard services. For example, SMTP, the mail protocol, is always on port 25, and telnetd is always on port 23. To see a list of the reserved ports on a Unix system, look at /etc/services.

We've examined how ports work on the server end -- specific ports are reserved for set tasks. On the initiator end, port assignment is dynamic. When a telnet client on swell starts up, it gets a new port number (e.g., 1066). This is the source port which swell's TCP layer puts on every packet. This allows the telnet daemon (telnetd) on cool to responds to the correct telnet process on swell. The combination of source/destination IP addresses and ports provides a unique conversation identifier. Each conversation is called a flow.

UDP is essentially IP with port numbers (flows). It gives the user access to IP-style datagrams. The network file system (NFS) and talk are two examples of UDP-based protocols.

This has been an extremely cursory exploration of TCP and UDP. At this point, you should have a decent understanding of how the network (IP) and transport (TCP/UDP) layers interact. We now turn to the final layer.

The Application Layer

This is where the user interacts with the network. All network programs like telnet, ftp, mail, news, and WWW clients are at the application layer. They then use either TCP or UDP to communicate with other machines. To provide a clearer picture, I'll examine telnet in a little detail.

Telnet is used for remote login. It removes the need for hardwired terminals. A user on swell types ``telnet cool.alaska.edu'' and he is rapidly connected to cool.alaska.edu, which asks him to login. He can then interact with cool. Here's a breakdown of the process:

1. The name address is turned into a numeric one (137.229.18.103), via a domain name server.
2. Telnet tells the transport layer it wants to start a TCP connection with 137.229.18.103, at port 23
3. TCP initiates a conversation with cool. IP is used to route packets. The telnet process on swell gets a port number of, say, 1096. TCP places the source and destination port numbers in its packet header, IP the IP address.
4. Packets are now handed to the IP layer, which sends them to the link layer. They proceed from the user's machine to his organization's router and out onto the Internet. They make their way to cool, one router at a time.
5. cool's TCP layer replies in a similar fashion.
6. The telnet daemon on cool and the telnet client on swell exchange terminal information and other parameters necessary for an interactive session. Control messages are sent in-band, as an escape byte of 255, followed by the control byte. Control messages include: echo, status,

terminal type, terminal speed, flow control, linemode, and environmental variables.

7. The user sees a login prompt. After logging in, data is sent back and forth.

Once the task is broken into a number of steps, we see it's relatively simply. Because the transport layer provides a standard interface, network applications do not need to be rewritten or even recompiled if the transport layer code changes.

Conclusion

We have now covered the basic four layers of TCP/IP. The reader should have a basic idea of how the different parts of TCP/IP interact. We have deliberately left out most of the detail. There are plenty of good books written on the subject, particularly [1]. In addition, there are the Request for Comments (RFCs), the defining documents of the Internet (don't let the name fool you). RFCs are issued by the Internet Engineering Task Force (IETF), the standards body for the Internet and available, among other places, at ftp.internic.net:/rfc.

References

1. Stevens, W. R. ***TCP/IP Illustrated, Volume 1.*** Addison-Wesley, Reading, Massachusetts, 1994.