TCP/IP

Alex S.*

TCP/IP is a collection of protocols, each designed for a particular task. They are: IP, ICMP, ARP, UDP, TCP. IP is responsible for sending data between computers. ICMP is not used for sending data, but is used for communicating things like errors, etc. ARP is used to figure out lower level addresses—such as MAC addresses. UDP is used to send messages between applications—it is basically the IP protocol with a port address. TCP is the key protocol of the whole package—it provides a reliable data stream between two network nodes.

1 TCP, Transmission Control Protocol

1.1 Connection Oriented

One of the key features of TCP is that it's connection oriented. Meaning that the two parties need to establish a 'connection' (or a session) before they can send data—and that data *appears* to be coming from a stream (one byte at a time), as opposed to discrete chunks.

In TCP, a connection is established via an opening hand-shake. This consists of SYN packet sent from the initiating computer. This is acknowledged by the recipient computer with a SYN & ACK packets, which the initiating computer acknowledges (ACK) back.

It goes something like this:

C1: Howdy, you wanna talk?

C2: Ok, What's up?

C1: Ok, Here's the data.

Once the connection is established, the two begin sending each other data... and that's pretty much it.

The receiving computer sends its buffer size (window size) as part of the ACK packet. The sending computer is then free to send data packets upto the size of that buffer. The receiving computer, then acknowledges those packets (and possibly moves the buffer window a bit).

*alex@theparticle.com

Just like there is a connection start, there is also a connection 'end'. To close a connection, one of the computers sends a FIN & ACK packets, and wait for the FIN & ACK back, which they then ACK, and the connection is closed. Basically this ensures that everyone has received everyone's data and that the connection is properly closed. Of course, the connection can also time-out, and that would be the end.

That's pretty much it for TCP.

1.2 Sockets

All communication happens through an imaginary thing called a 'socket'. This is nothing more than an abstraction for a file, which is itself an abstraction of some other concept.

You create sockets using system calls (most operating systems, languages, systems, etc., provide functions to create and manipulate sockets).

1.3 Client/Server

The term client/server just refers to the fact that there is some client and some server. A client is generally a computer that needs services. A server is generally a computer that provides some service. Although the terms can be used in many domains, they are very often applied to TCP/IP networks.

1.3.1 The Server

The server generally begins it's life by binding to a particular port, and listening (accepting connections) on that port. When a connection arrives, it returns with a socket that belongs to the client that just connected. If you read from that socket, you'll be reading exactly what the client wrote. And if you write[1] to that socket, you'll be writing to the client.

While the service is going on, the server can do anything it wants with these sockets. The server can manage a whole collection of sockets in some data structure. It can also create a 'client' object for each accepted connection, and use that client object to manage each individual socket (client connection).

When the service is over, the server can close the socket, which closes the connection.

If working in Java, look into the `java.net.ServerSocket` object documentation. It is simply a matter of creating an instance of that object, and you have a server socket that's ready to accept connections.

1.3.2 The Client

Client connects to a server using its own socket. Usually, you don't care what port the client is using (so you rarely bind a client to a particular port—just let the system pick next free port).

[1] The term 'write' is used loosely—very often, you setup your system so that you 'print' to sockets; just like you would if you were writing to a file.

The client generally only needs to know the host-name of the server, and the port to which to connect. The client then creates a socket to that port host:port. Whatever the client then writes to that socket is sent to the server. Whatever it reads from that socket, comes from the server. The client can close the connection, etc., when it no longer needs the socket.

If working in Java, there is a very convenient `java.net.Socket` class that accepts a host-name and port address.

1.3.3 Vague Terms

The terms client/server are used loosely. Often, a server acts like a client for some other services. For example, it is not uncommon for web-server code to act as a client to the database. So the term 'server' and 'client' are just names for ideas, not necessarily real-life clients.