Having spent the last two chapters learning the basics of UDP and TCP, the two major data transports available on IP networks, it is time for us to step back and talk about two larger issues that need to be tackled regardless of which transport you are using. In this chapter, we will discuss the topic of network addresses and will describe the distributed service that allows names to be resolved to raw IP addresses.

Hostname and Domain Names

Before we plunge into this topic, we should get a few terms straight that will play a big role in the discussion that follows.

- **Top-level domain (TLD):** These are the few hundred strings like `com`, `net`, `org`, `gov`, and `mil` that, together with country codes like `de` and `uk`, form the set of possible suffixes for valid domain names. Typically, each TLD has its own set of servers and its own organization that is in charge of granting ownership to domains beneath the TLD.

- **Domain name:** This is the name that a business or organization appends as a suffix to its sites and hosts on the Internet, like `python.org`, `imdb.com`, or `bbc.co.uk`. It typically costs an annual fee to own a domain name, but owning one gives you the right to create as many hostnames beneath it as you want.

- **Fully qualified domain name:** The FQDN names an Internet site or host by appending its organization’s full domain name to the name of a particular machine in that organization. Example FQDNs are `gnu.org` and `asaph.rhodesmill.org`. Whether a domain name is “fully qualified” does not depend on its having any specific number of components—it may have two, three, four, or more dot-separated names. What makes it a FQDN is that it ends with a TLD and therefore will work from anywhere. You can often use just the hostname `athena` if you are connected to an MIT network, but from anywhere else in the world, you have to fully qualify the name and specify `athena.mit.edu`.

- **Hostname:** This term, unfortunately, is ambiguous! Sometimes it means the bare, unqualified name that a machine might print when you connect to it, like `asaph` or `athena`. But sometimes people instead mean the FQDN when they say “the hostname.”
In general, an FQDN may be used to identify a host from anywhere else on the Internet. Bare hostnames, by contrast, work as relative names only if you are already inside the organization and using their own nameservers (a concept we discuss later in this chapter) to resolve names on your desktop, laptop, or server. Thus athena should work as an abbreviation for athena.mit.edu if you are actually on the MIT campus, but it will not work if you are anywhere else in the world—unless you have configured your system to always try MIT hostnames first, which would be unusual, but maybe you are on their staff or something.

Socket Names

The last two chapters have already introduced you to the fact that sockets cannot be named with a single primitive Python value like a number or string. Instead, both TCP and UDP use integer port numbers to share a single machine's IP address among the many different applications that might be running there, and so the address and port number have to be combined in order to produce a socket name, like this:

('18.9.22.69', 80)

While you might have been able to pick up some scattered facts about socket names from the last few chapters—like the fact that the first item can be either a hostname or a dotted IP address—it is time for us to approach the whole subject in more depth.

You will recall that socket names are important at several points in the creation and use of sockets. For your reference, here are all of the major socket methods that demand of you some sort of socket name as an argument:

- `mysocket.accept()`: Each time this is called on a listening TCP stream socket that has incoming connections ready to hand off to the application, it returns a tuple whose second item is the remote address that has connected (the first item in the tuple is the net socket connected to that remote address).
- `mysocket.bind(address)`: Assigns the socket the local address so that outgoing packets have an address from which to originate, and so that any incoming connections from other machines have a name that they can use to connect.
- `mysocket.connect(address)`: Establishes that data sent through this socket will be directed to the given remote address. For UDP sockets, this simply sets the default address used if the caller uses `send()` rather than `sendto()`; for TCP sockets, this actually negotiates a new stream with another machine using a three-way handshake, and raises an exception if the negotiation fails.
- `mysocket.getpeernam()`: Returns the remote address to which this socket is connected.
- `mysocket.getsockname()`: Returns the address of this socket's own local endpoint.
- `mysocket.recvfrom(...)`: For UDP sockets, this returns a tuple that pairs a string of returned data with the address from which it was just sent.
- `mysocket.sendto(data, address)`: An unconnected UDP port uses this method to fire off a data packet at a particular remote address.

So, there you have it! Those are the major socket operations that care about socket addresses, all in one place, so that you have some context for the remarks that follow. In general, any of the foregoing methods can receive or return any of the sorts of addresses that follow, meaning that they will work