Assessing Your Networking Needs

Before you can choose a networking API, you need to know a little bit about the networking API families that OS X and iOS offer.

OS X and iOS provide three main user-space networking API layers. The first two—Foundation and CFNetwork (based on Core Foundation)—are frameworks specific to OS X and iOS. The lowest layer, POSIX, is the same as on any other UNIX- or Linux-based operating system.

Within each layer, there are functions or classes that support common networking tasks, such as connecting to remote hosts (protocol streams), downloading the contents of URLs, and discovering services on your local network. These layers are shown in Table 2-1.

Table 2-1 The layers and families of OS X and iOS networking APIs

Layer	Protocol streams	URL loading	Service discovery
Foundation layer	`NSStream`	`NSURLConnection` and `NSURLRequest`	`NSNetService`
Core Foundation layer	`CFStream`	`CFHTTPMessage`	`CFNetService`
POSIX layer	`kqueue`	`libcurl` (for example. Note that this is a third-party API)	DNS Service Discovery
	Built on top of BSD sockets (directly or indirectly)		

You can easily accomplish most client-side networking tasks using only Foundation classes. If you are writing server code or you have specialized needs, you may want to use lower-level frameworks instead. However, as a general rule, you should always choose the highest-level API that meets your needs.

Common Networking Tasks

Before you can decide which specific API to use, you must first assess what networking tasks your program needs to perform.

Support peer-to-peer networking for games. In iOS, the Game Kit framework provides support for peer-to-peer communication, both globally (over the Internet) and locally (using a Bluetooth personal area network or a Wi-Fi LAN).

You can use Game Kit in your app to simplify the following peer-to-peer networking tasks:

- Providing network communication for a multiplayer game
- Providing voice communication

Any peer-to-peer communication that is not covered by the tasks above should be accomplished with lower level networking APIs described later in this section.

To learn more about Game Kit, read *Game Center Programming Guide*.

Support peer-to-peer networking for other apps. In iOS, the Multipeer Connectivity framework provides support for peer-to-peer communication over infrastructure Wi-Fi, peer-to-peer Wi-Fi, and

Bluetooth. To learn more, read Discovering and Advertising Network Services.

Connect to a web server. The preferred way to send and receive short pieces of information is over a standard protocol such as HTTP or HTTPS. By using these existing protocols, you minimize the amount of work needed to support the connection on both the client and server side. HTTP also makes it easy to move to a secure (HTTPS) connection—you just add a certificate on your server and then add a single letter to the first part of the URL.

To learn more about APIs for making HTTP and HTTPS requests, read Making HTTP and HTTPS Requests.

To learn how to display a web page in your application, read Displaying Web and Multimedia Content.

Connect to an FTP server. Unless you must do so to maintain compatibility with existing servers, the use of FTP is discouraged. FTP is an old protocol with serious limitations and no real security (data and passwords are sent in cleartext).

With that in mind, if you just need to download a file over FTP, you should use the `NSURLConnection` API and pass it the appropriate URL. This API is described in Making HTTP and HTTPS Requests, but can also be used with `ftp://` URLs.

For more complex requests, the CFNetwork framework (based on Core Foundation) provides the `CFFTPStream` API for communicating with FTP servers. CFNetwork also provides the `CFURLAccess` API, which can be used for deleting files on an FTP server. The details of these APIs are outside the scope of this document. To learn more, read *CFNetwork Programming Guide*.

Discover and advertise network services. OS X and iOS provide support for DNS Service Discovery, which allows you to describe what services your program provides and discover other services on the user's machine, on nearby machines, or on one of the user's home machines using Back to My Mac. You can then use that information to communicate with other copies of your program or other programs that your program knows how to talk to. For example, OS X uses DNS Service Discovery to let users find nearby printers, stream music in iTunes from nearby computers, share screens in Finder, and so on.

To learn how to discover services on your local area network or on remote servers using DNS Service Discovery, read Discovering and Advertising Network Services.

Resolve DNS hostnames. OS X and iOS provide Core-Foundation-layer and POSIX-layer name resolver APIs for obtaining IP addresses for a hostname. To learn about these APIs, read Designing for Real-World Networks. However, if you are resolving hosts because you want to connect to them, you should generally connect by name instead. Read Avoid Resolving DNS Names Before Connecting to a Host in Avoiding Common Networking Mistakes to learn why.

Use sockets or socket streams. If you need to make network requests in ways that are not supported by higher-level APIs, you can use sockets (at both the POSIX layer and the Core Foundation layer) or socket streams (at the Core Foundation layer). For more information, read Using Sockets and Socket Streams.

Communicate securely. OS X and iOS support the Transport Layer Security (TLS) protocol and its predecessor, the Secure Sockets Layer (SSL), for encrypted communication and server trust determination. To learn more, read Using Networking Securely.

Next Steps

Now that you have decided what you want to do, you can easily accomplish a wide array of networking tasks in OS X and iOS, with little or no configuration. Many of the most common networking tasks and the recommended methods for accomplishing them are briefly described in the chapters that follow. Keep in mind that these are not comprehensive API discussions; each chapter ends with links to other documents that provide in-depth information about these tasks.