Network Interface Concepts

This thread has been locked by a moderator.

40

For important background information, read Extra-ordinary Networking before reading this.

Share and Enjoy
—
Quinn “The Eskimo!” @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Network Interface Concepts

Most developers don’t care about how the system manages network interfaces and their addresses. However, for some apps these details matter. This post gives an overview of that process.

IMPORTANT This post covers a lot of ground, and its focus is on concepts. I’ve skipped over a bunch of details, and it’s also quite possible that I’ve got some things wrong. Lemme know if you find any particularly egregious problems.

Terminological Inexactitude

The term *interface* is overloaded. In general terms we say things like “connect the Ethernet interface”, which refers to a **hardware interface**. However, at the API level a **network interface** represents a source and sink of packets for a particular network protocol. That interface may run over a hardware interface, or it may be entirely virtual, like VPN or loopback.

Your Network Interface Fallacy Is…

Networking on Apple devices is way more complicated than you might think. Consider:

- A device may have 0 or more network interfaces [1].
- A device may have 0 or more network interfaces of a given type. Historically there was an iPhone with no Wi-Fi interface! And an iPhone today might have multiple WWAN interfaces.
- No single network interface best represents the device as a whole.
- BSD interface names are not considered API. There’s no guarantee, for example, that an iPhone’s Wi-Fi interface is `en0`.
- A device may have 0 or more IP addresses.
- Which can be an arbitrary mix of IPv4 and IPv6.
- Each address may or may not be globally reachable. For example, a 192.168.0.0/16 private-use address is not globally reachable.
- A hardware interface may have 0 or more IP addresses.
- These can be an arbitrary mix of IPv4 and IPv6.
- And each one may or may not be link-local. It’s common for a hardware interface to have multiple IPv6 link-local addresses.
- No single IP address best represents the device as a whole.
- Even within the scope of a single hardware interface, no single IP address best represents that interface.
- There’s no correlation between an interface’s type and the presence of a globally reachable IP address. For example, most cellular carriers use NAT, so the device’s WWAN interface gets a private-use address, but some carriers don’t, so it gets a globally reachable IP address!
- The list of interfaces can change at any time.
- As can the list of IP addresses.

[1] For this discussion I’m ignoring loopback interfaces and their loopback IP addresses. You can rely on those always being present.

Address Acquisition

When an interface comes up — for example, the user joins a Wi-Fi network or plugs in an Ethernet cable — the system tries to acquire IP addresses for that interface. The exact mechanism for this varies by IP version and interface type. For example, for Ethernet-like interfaces the system typically gets an IPv4 address using DHCP. In contrast, for IPv6 the system starts by assigning at least one link-local address and then uses that to request addresses from the router.

There are three common behaviours with respect to IPv4:

- In some cases the system won’t even try to acquire an IPv4 address for the interface. Apple platforms often have private, internal interfaces. They assign such an interface a link-local IPv6 address. The internal subsystems that use them are expected to support IPv6.
- In others it’ll try to acquire an IPv4 address but, if that fails, it falls back to a link-local IPv4 address. For more on IPv4 link-local addresses, see RFC 3927 Dynamic Configuration of IPv4 Link-Local Addresses.
- In other cases it’ll try to acquire an IPv4 address but, if that fails, it’ll take the interface down.

Interfaces and the Default Route

Apple platforms support multiple network interfaces. One of those interfaces is special in that it holds the *default route*. That route is used when the system needs to send traffic and there’s no other information about where to send it. In practice, the default route is effectively the path to the wider Internet.

As interfaces come up and go down, the system re-evaluates which one should be the default route. This is complicated by two factors:

- An interface can come up in two stages. In the first stage, the system brings up the interface to evaluate whether it’s useful. A classic example of this is the captive network support on iOS. If that evaluation completes successfully, the system makes the interface eligible to become the default route.
- In the presence of VPN there may be multiple default routes! If a VPN interface claims the default route, it’ll be the default route for most programs. However, inside the VPN interface’s implementation, the original default route takes precedence.

Note Thinking about that second point makes my brain hurt. If you want to join me in the pain, hop on over to A Peek Behind the NECP Curtain.

iOS has a general policy of preferring Wi-Fi to WWAN. If the Wi-Fi interface is eligible to the become the default route, iOS makes it so. Many iPhone users think of this as “switching to Wi-Fi”. That’s the general effect, but its not how it actually works. When iOS switches the default route to the Wi-Fi interface, the WWAN interface is still available, it’s just not the default route.

In contrast, macOS gives the user control over this. In System Settings > Network there’s an action menu with a Set Service Order command. The resulting window shows the list of all the network interfaces. The user can reorder interfaces in that list. As interfaces come up and down, macOS ensures that the default route is always the first interface that’s eligible.

Routing Information

The previous section says “and there’s no other information about where to send it”. What other information might there be? This brings us to the system’s routing table. It’s a set of rules that lets the system map traffic to a destination interface. The routing table is directly exposed on macOS, but you can see its effect on all platforms. For example:

- For each interface there’s a route to that interface’s network. So, imagine your local Wi-Fi network is 192.168.1.0/24. Any traffic destined for that network will be routed directly to that interface, ignoring the default route.
- You can bind a network operation to a specific interface, ensuring that its traffic will only go out over that interface.
- Packet tunnel providers can add routes for their VPN interface.

On macOS you can dig into the routing table with with tools like:

- `netstat` — See its man page.
- `route` — See its man page, in section 8.

There’s even a routing socket API; see the `route` man page in section 4.

WARNING Routing sockets are not considered API on other Apple platforms. The declarations needed to use them are not part of our other platform SDKs. I’ve seen folks copy the declarations from the macOS SDK and go to town. That is *so* not supported.

Network

Reply

Posted 2 days ago by eskimo

Add a Comment