Article

Connecting devices for peer-to-peer Wi-Fi

Make outgoing and accept incoming secure connections with paired devices.

Overview

With the Wi-Fi Aware™ technology, your app can securely establish peer-to-peer (P2P) connections between Wi-Fi devices. By using the Wi-Fi Aware framework, devices can discover, pair, and communicate with other nearby devices without an internet connection or access point.

For your app to pair devices, or ask the person using your app for access to existing devices, you can use either of the following two methods:

- `DeviceDiscoveryUI`: Pair any kind of device for device-to-device and app-to-app use cases, such as file transfer and media streaming.
- `AccessorySetupKit`: Pair a person's personal hardware accessory with the accessory's companion app, such as for setup and accessory management.

After a person pairs a device or grants your app access to a device, the system adds the device to your app's `WAPairedDevice.Devices` collection, which is accessible through the `allDevices` APIs.

Connect paired devices

After your app pairs devices, it can initiate secure peer-to-peer Wi-Fi connections between those devices on demand. The Wi-Fi Aware framework integrates with the Network framework to provide Wi-Fi Aware connections and related functionality, extending several common networking primitives:

- `NetworkBrowser` subscribes to Wi-Fi Aware services on paired devices, and makes outgoing secure connections to them.

- `NetworkListener` publishes Wi-Fi Aware services to paired devices, and accepts incoming secure connections from them.
- `NetworkConnection` opens a secure, high-performance data connection to Wi-Fi Aware devices.
- `NWParameters` configures Wi-Fi Aware parameters.
- `NWPath` fetches Wi-Fi Aware connection statuses and performance metrics.
- `NWError` fetches Wi-Fi Aware error statuses.

Create a listener to publish

The following code example creates a `NetworkListener` that publishes the `_example-service._tcp` service over Wi-Fi Aware, and accepts incoming connections from the specified paired devices:

```
// Specifies a service.
extension WAPublishableService {
    public static var exampleService: WAPublishableService {
        allServices["_example-service._tcp"]!
    }
}

// Selects devices from the list of the example app's paired devices.
let devices = #Predicate<WAPairedDevice> { $0.name?.starts(with: "My Device") ?? fal

// Constructs a `NetworkListener` to publish the service and accept connections fron
try await NetworkListener(for:
    .wifiAware(.connecting(to: .exampleService, from: .matching(devices))),
using: {
    TLS()
})
.onStateUpdate { listener, state in
    // Processes state updates.
}
.run { connection in
    // Handle incoming connections.
}
```

Create a browser to subscribe

The following code example creates a `NetworkBrowser` that subscribes for the `_example-service._tcp` service over Wi-Fi Aware, and provides browse results that the system can use to make outgoing connections to the specified paired devices:

```
// Specifies a service.
extension WASubscribableService {
    public static var exampleService: WASubscribableService {
        allServices["_example-service._tcp"]!
    }
}

// Selects `devices` from the list of all paired devices.
let devices = #Predicate<WAPairedDevice> { $0.name?.starts(with: "My Device") ?? fal

// Constructs a `NetworkBrowser` to subscribe for the service on the selected device
let browser = NetworkBrowser(
    for:
        .wifiAware( .connecting(to: .matching(devices),  from: .exampleService))
)
.onStateUpdate { browser, state in
    // Processes state updates.
}

// Run the browser and get discovered endpoints
let endpoint = try await browser.run { waEndpoints in
    // Handle discovered endpoints.
    // Return .finish(endpoint) once the desired endpoint(s) are discovered or .cont
}
```

Make a connection

The following code example creates a `NetworkConnection` to a discovered Wi-Fi Aware endpoint:

```
// On the browser device, construct a `NetworkConnection` to make a connection to or
let connection = NetworkConnection(
    to:
        endpoint,
    using: {
        TLS()
    }
).onStateUpdate { connection, state in
    // Processes state updates.
}

// The run closure on the `NetworkListener` device receives the incoming connection
```

Monitor connection performance

The following code example gets the Wi-Fi Aware performance report from the current path of a connection.

```
if let wifiAwarePath = try await connection.currentPath?.wifiAware {
    let performanceReport = wifiAwarePath.performance

    // Use the performance report.
}
```

Get Wi-Fi Aware errors

The following code example gets the Wi-Fi Aware specific error code from the `NWError` reported by the Network framework as part of the state updates provided for the `NetworkBrowser`, `NetworkListener` and `NetworkConnection` instances.

```
.onStateUpdate { _, state in
    switch state {
    ...

    case .failed(let nwError): let wifiAwareError = error.wifiAware
    default: break
    }
}
```

See Also

Essentials

{} Building peer-to-peer apps

Communicate with nearby devices over a secure, high-throughput, low-latency connection by using Wi-Fi Aware.

Adopting Wi-Fi Aware

Add entitlements and declare your app’s services.

`com.apple.developer.wifi-aware`

The entitlement the system requires for an app to use the Wi-Fi Aware framework.

`WiFiAwareServices`

Dictionaries of Wi-Fi Aware services that the app can publish or subscribe to.