

Bonjourとネットワーク通信を通して学ぶ Network.framework実践ガイド

Yutaro Muta (@yutailang0119)

<https://fortee.jp/iosdc-japan-2024/proposal/38a4ff87-2cc1-4494-b8e6-63fcf506430f>

Network.framework^{※1}は、2018年6月の「World Wide Developers Conference 2018」(以下、WWDC18)で発表されました。このフレームワークは、iOS 12.0をはじめとしたWWDC18発表以降のAppleプラットフォームで使用できます。また、visionOSもサポートしています。

本稿ではNetwork.frameworkを活用し、NWBrowserでのネットワークサービスの検出と、NWListener^{※2}およびNWConnection^{※3}を用いた通信について解説します。また、Swift Concurrencyと組み合わせて、2024年時点のアプローチでサンプルコードを提供します。Network.frameworkがサポートする範囲は膨大で、すべてを網羅はできませんが、取り組むきっかけとなる内容を扱います。

はじめに

本稿を読む上での注意事項を説明します。

GitHubリポジトリ

本稿はgithub.com/yutailang0119/iosdc-2024-pamphlet^{※4}でも閲覧可能です。サンプルコードは、リポジトリの /ExampleApp を参照してください。

内容に訂正がある場合、GitHubリポジトリで訂正します。また、質問などはissues^{※5}から連絡ください。

開発環境

本稿の内容は、開発環境としてmacOS Sonoma 14.5 (23F79)とXcode 15.4 (15F31d)を使用して動作を確認しています。開発言語にはXcodeに付属のSwift 5.10を利用します。サンプルコードの実行は、iOS 17.5 (21F79)を対象とします。

権利・ライセンス

本稿の全文、GitHubリポジトリ /pamphlet 以下は、クリエイティブ・コモンズのCC-BY 4.0^{※6}で公開します。サンプルコード、GitHubリポジトリ /ExampleApp 以下は、MITライセンスで利用できます。

ローカルネットワークのプライバシー

本稿で扱うローカルネットワークアクセスは、ユーザーの許可が必要です。Info.plistに対応するキーを指定します。ローカルネットワークの使用には、NSLocalNetworkUsageDescription^{※7}に用途を記載します。Bonjourでの検出には、NSBonjourServices^{※8}にサービスタイプを宣言します。詳しくはWWDC20の動画“Support local network privacy in your app”^{※9}を参照してください。

iPhoneやMacのネットワーク設定によって、ローカルネットワークの通信は制限されることがあります。Firewallの設定を確認し、一時的なオフを検討してください。

※1 : <https://developer.apple.com/documentation/network>
※2 : <https://developer.apple.com/documentation/network/nwlistener>
※3 : <https://developer.apple.com/documentation/network/nwconnection>
※4 : <https://github.com/yutailang0119/iosdc-2024-pamphlet>
※5 : <https://github.com/yutailang0119/iosdc-2024-pamphlet/issues>
※6 : <https://creativecommons.org/licenses/by/4.0/>
※7 : https://developer.apple.com/documentation/bundleresources/information_property_list/nslocalnetworkusagedescription
※8 : https://developer.apple.com/documentation/bundleresources/information_property_list/nsbonjournservices
※9 : <https://developer.apple.com/videos/play/wwdc2020/10110/>

Network.frameworkの概観

Network.frameworkは、データ送受信のためのネットワーク接続用フレームワークです。URLSession^{※10}の内部でも使用されており、多くのアプリを支えています。

サポートする通信プロトコル

Network.frameworkを使用することで、TLS、TCP、UDPなどの主要な通信プロトコルに直接アクセスが可能です。UDPは NWProtocolUDP^{※11}といった、プロトコルに対応するclassがそれぞれ用意されています。iOS 13では NWProtocolWebSocket^{※12}でWebSocket、iOS 15では NWProtocolQUIC^{※13}でQUICのサポートが追加されました。

さらに NWProtocolFramer^{※14}を使って、独自プロトコルでの通信も実装できます。従来のソケット通信の複雑なバイナリ列を扱うことなく、構造化されたメッセージの読み取りが可能です。詳しくはWWDC19の動画“Advances in Networking, Part 2”^{※15}を参照してください。

通信状況のモニタリング——NWPathMonitor

Network.frameworkの機能の一つは、ネットワーク状況の監視です。監視には NWPathMonitor^{※16}を使用します。具体的な使用方法は、サンプルコードを参照してください。

NWPathMonitor は、実際に通信が成功するかを正確に反映するものではないことに注意します。ネットワークが利用可能かは推測せず、ユーザーによって行われた通信は常に接続を試みるべきです。

Network.frameworkの登場以前

Network.frameworkの機能は、CFNetwork.framework^{※17}やFoundation.framework^{※18}のNS prefixなAPIなどを駆使して、実装可能でした。しかし、C言語との相互運用やポインタを意識した実装が必要で、Swiftに慣れ親しんだ開発者には難解です。Network.frameworkの利点は、簡単な接続の確立、データ転送の最適化、組込みのセキュリティ、シームレスなモビリティ、そしてSwiftのフレームワークであることです。

Bonjourを用いたネットワーク上サービスの検出

ネットワークはインターネットに接続するだけではありません。ローカルネットワーク上のほかのデバイスとやりとりすることができます。Appleデバイスでローカルネットワークを利用するには、Bonjour(ボンジュール)が最適です。Network.frameworkを使ったBonjourサービスの検出について解説します。

Bonjour

Bonjourは、Appleが開発したゼロコンフィギュレーションプロトコルで、IPアドレスやホスト名を入力せずに接続する方法を提供します。ビデオやオーディオのストリーミング、peer-to-peerゲーム、プリンター、カメラ、ホームデバイスとの通信の基盤となります。たとえば、同じネットワーク上のAirPrintのプリンター自動検出やHomeKitへの接続に使用されます。

現在ではAppleデバイスに限らず、LinuxやAndroid、Windowsなどの主要プラットフォームでも利用できます。プラットフォームの垣根を超えた接続、連携が可能です。

※10 : <https://developer.apple.com/documentation/foundation/urlsession>
※11 : <https://developer.apple.com/documentation/network/nwprotocoludp>
※12 : <https://developer.apple.com/documentation/network/nwprotocolwebsocket>
※13 : <https://developer.apple.com/documentation/network/nwprotocolquic>
※14 : <https://developer.apple.com/documentation/network/nwprotocolframer>
※15 : <https://developer.apple.com/videos/play/wwdc2019/713/>
※16 : <https://developer.apple.com/documentation/network/nwpathmonitor>
※17 : <https://developer.apple.com/documentation/cfnetwork>
※18 : <https://developer.apple.com/documentation/foundation>

Network.frameworkを用いたAirPlayの検出

Appleデバイスに囲まれた生活を送る私たちの周りには、Apple TVなどのAirPlay対応デバイスが溢れています。Network.frameworkを使って、同じネットワーク上のAirPlayを検出してみます。

▼ AirPlay

Bonjourの検出には、告知(アドバタイズ)しているサービス名が必要です。AirPlayは“_airplay._tcp”でTCPサービスを告知しています。より詳しい解説は“AppleデバイスでAirPlayを使用する”^{※19}を参照してください。

▼ NWBrowserでのAirPlay検出

Network.frameworkでのBonjourサービスの検出は、NWBrowser^{※20}を使用します。サービスタイプ“_airplay._tcp”を指定してNWBrowserを初期化し、browseResultsChangedHandler^{※21}で検出結果の変化を受け取ります。start(queue:)^{※22}で、検出を開始します(図1)。

```
func browse() -> AsyncStream<Set<NWBrowser.Result>> {
    AsyncStream { continuation in
        let browser = NWBrowser(
            for: .bonjour(
                type: "_airplay._tcp",
                domain: nil
            ),
            using: .tcp
        )
        browser.browseResultsChangedHandler = { results, changes in
            continuation.yield(results)
        }
        continuation.onTermination = { _ in
            browser.cancel()
        }
        browser.start(queue: .main)
    }
}
```



(図1) “_airplay._tcp”の検出

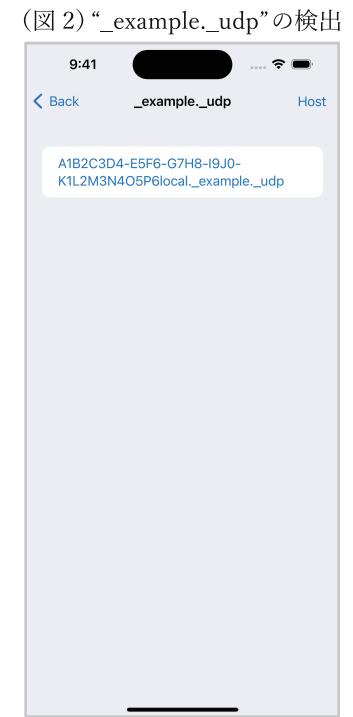
```
func host() -> AsyncThrowingStream<NWConnection, Error> {
    AsyncThrowingStream { continuation in
        do {
            let listener = try NWListener(using: .udp)
            listener.service = NWListener.Service(name: uuid, type: "_example._udp")
            continuation.onTermination = { _ in
                listener.cancel()
            }
            listener.start(queue: .main)
        } catch {
            continuation.finish(throwing: error)
        }
    }
}
```

Bonjourサービスの検出(ディスカバー) —— NWBrowser

NWListenerが告知しているBonjourサービスの検出は、前述のNWBrowserを使用します。サービスタイプを“_example._udp”、NWParameters^{※25}の指定を.udp^{※26}に変更します。

```
let browser = NWBrowser(
    for: .bonjour(
        type: "_example._udp",
        domain: nil
    ),
    using: .udp
)
```

NWListenerが告知しているサービスの一覧を検出します(図2)。



(図2) “_example._udp”的検出

Network.frameworkでのUDP送受信

UDPでのコネクション、データの送受信を解説します。例は“_example._udp”で接続し、パネルの状態を相互に送り合うアプリです。

Bonjourサービスの告知(アドバタイズ) —— NWListener

サービスの告知は、NWListenerを使用します。初期化したNWListenerにサービスタイプ“_example._udp”を指定したNWListener.Service^{※23}を設定します。start(queue:)^{※24}で、告知を開始します。

コネクションの確立 —— NWConnection

告知、検出がそれぞれできたので、コネクションを確立します。Network.frameworkでは、NWConnectionを使用して接続します。

例にするUDPでは、TCPのようなハンドシェイクを省略して、コネクションが確立されます。

▼ Bonjourサービス検出側のコネクション

Bonjourサービスを検出する側から、コネクションの確立を試みます。検出したNWBrowser.Result^{※27}からNWEndpoint^{※28}を取得し、NWConnectionを初期化します。start(queue:)^{※29}で、コネクションを確立します。

※19 : <https://support.apple.com/ja-jp/guide/deployment/dep9151c4ace/web>

※20 : <https://developer.apple.com/documentation/network/nwbrowser>

※21 : <https://developer.apple.com/documentation/network/nwbrowser/3200395-browserresultchangedhandler>

※22 : <https://developer.apple.com/documentation/network/nwbrowser/3200402-start>

※23 : <https://developer.apple.com/documentation/network/nwlistener/service>

※24 : <https://developer.apple.com/documentation/network/nwlistener/2998669-start>

※25 : <https://developer.apple.com/documentation/network/nwparameters>

※26 : <https://developer.apple.com/documentation/network/nwparameters/2998711-udp>

※27 : <https://developer.apple.com/documentation/network/nwbrowser/result>

※28 : <https://developer.apple.com/documentation/network/nwendpoint>

※29 : <https://developer.apple.com/documentation/network/nwconnection/2998575-start>

```
let connection = NWConnection(to: result.endpoint, using: .udp)
connection.start(queue: .main)
```

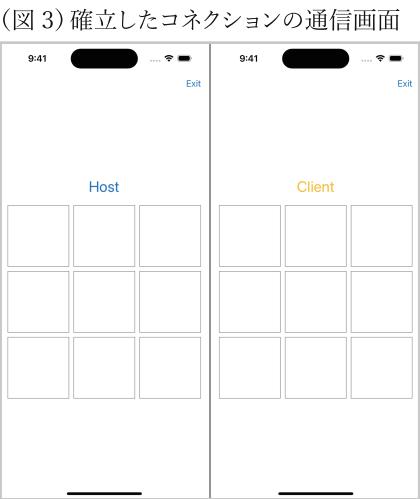
UDPの場合、`send(content:contentContext:isComplete:completion:)`^{※30} で一度データを送信することで、`NWListener` 側のコネクションも起動します。

▼ Bonjourサービス告知側のコネクション

`NWListener` で告知しているサービスにコネクションが確立すると、`newConnectionHandler`^{※31} に通知されます。

```
func host() -> AsyncThrowingStream<NWConnection, Error> {
    AsyncThrowingStream { continuation in
        (省略)
        listener.newConnectionHandler = { connection in
            continuation.yield(connection)
        }
        (省略)
    }
}
```

`NWListener` では、通知された `NWConnection` を使って通信を開始します(図 3)。



データの送受信

コネクションの確立ができたので、`NWConnection` を通して双方向にデータの送受信を行います。例は `Codable`^{※32} に準拠した `ConnectionData` を `Data` に変換してやりとりします。

▼ データ送信の実装

データの送信は、前述の `send(content:contentContext:isComplete:completion:)` を使用します。UDPのようなデータグラムプロトコルでは、`isComplete` はコンテンツが完全なデータグラムを表していることを示します。例では、常に完結したデータを送信するため、`isComplete: true` としています。

```
let content = try? encoder.encode(connectionData)
connection.send(
    content: content,
    contentContext: .defaultMessage,
    isComplete: true,
    completion: .contentProcessed { error in
        print(error)
    }
)
```

▼ データ受信の実装

受信したデータは、`receiveMessage(completion:)`^{※33} で取得できます。これまで登場したハンドラと異なり、呼び出しごとに一度だけ実行されるため、複数回データを受信するには再度実行する必要があります。この実装でも、`AsyncThrowingStream`^{※34} は便利です。

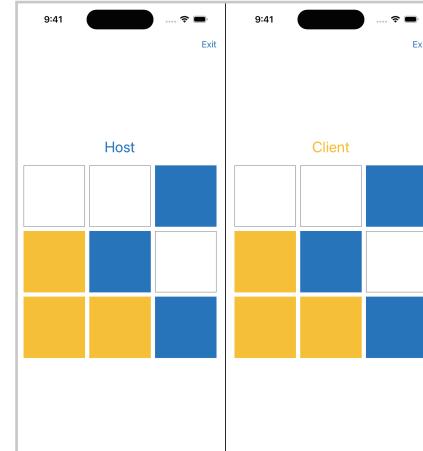
```
func receiveMessages() -> AsyncThrowingStream<Data, Error> {
    AsyncThrowingStream { continuation in
        func receiveMessage() {
            connection.receiveMessage { content, contentContext, isComplete, error in
                if let content {
                    continuation.yield(content)
                    receiveMessage()
                } else if let error {
                    print(error)
                    connection.cancel()
                }
            }
        }

        continuation.onTermination = { _ in
            connection.cancel()
        }
        receiveMessage()
        connection.start(queue: .main)
    }
}
```

パネルをタップするごとに、画面の状態を送信します(図 4)。

簡略化しましたが、`Codable` の `decode`、`encode` は実行コストが高く、高頻度の通信には向きません。また、実用にはパスコードの入力を挟むなど、コネクションの安全性も考慮する必要があるでしょう。GitHubリポジトリには TCPでの通信も用意したので、参照してください。

(図 4) 送受信したデータの表示



まとめ

Bonjourとローカルネットワーク通信の実装を通して、`Network.framework`について解説しました。ローカルネットワークでの通信には、`Core Bluetooth`^{※35} や `Multipeer Connectivity`^{※36} もあります。状況に合わせた選択肢を持っているとよいでしょう。

`Network.framework` が `URLSession` の内部で使用されていると説明した通り、インターネットへの接続も実装できます。HTTP通信を再実装してみるのも、よい題材になりそうです。

※30 : <https://developer.apple.com/documentation/network/nwconnection/3003626-send>

※31 : <https://developer.apple.com/documentation/network/nwlistener/2998663-newconnectionhandler>

※32 : <https://developer.apple.com/documentation/swift/codable>

※33 : <https://developer.apple.com/documentation/network/nwconnection/3020638-receivemessage>

※34 : <https://developer.apple.com/documentation/swift/asynctrueowingstream>

※35 : <https://developer.apple.com/documentation/corebluetooth>

※36 : <https://developer.apple.com/documentation/multipeerconnectivity>