How To Set Up a Connection Over LAN in SwiftUI

Bonjour over a network framework in iOS

Mark Lucking Follow
Mar 21 · 3 min read ★

Photo by Faisal Waheed on Unsplash.

I'm sure you've heard the saying "The secret to success is networking." I suspect it's also the secret to building killer apps. Let's take a journey through the tech you can use to connect your apps (aka sockets).

OK, there are two flavours: TCP and UDP. In short, TCP sockets guarantee delivery, whereas UDP sockets do not. TCP sockets are useful when you need to be sure the message you sent arrived. UDP sockets are useful when you're more concerned with speed.

On to the code then. Now, as with all conversations, we have somebody talking and somebody listening. This communication takes place over a UDP connection using a socket. Indeed, you need a pair. You listen on one device and talk on the other.

A Simple Listening Socket

Let's start with a simple socket app that listens on port 1984 and prints out what we send to it to the screen (assuming, of course, that we send a printable string):

```
1  import Network
2  import Combine
3
4  let sendToScreen = PassthroughSubject <String,Never>()
5
6  class Connect: NSObject {
7    private var talking: NWConnection?
8    private var listening: NWListener?
9    func listenUDP(port: NWEndpoint.Port) {
10     do {
11       self.listening = try NWListener(using: .udp, on: port)
12       self.listening?.stateUpdateHandler = {(newState) in
13         switch newState {
14         case .ready:
15           print("ready")
16         default:
17           break
18         }
19       }
20       self.listening?.newConnectionHandler = {(newConnection) in
```

```
21          newConnection.stateUpdateHandler = {newState in
22            switch newState {
23            case .ready:
24              print("new connection")
25              self.receive(on: newConnection)
26            default:
27              break
28            }
29          }
30          newConnection.start(queue: DispatchQueue(label: "new client"))
31        }
32      } catch {
33        print("unable to create listener")
34      }
35      self.listening?.start(queue: .main)
36    }
37
38    func receive(on connection: NWConnection) {
39      connection.receiveMessage { (data, context, isComplete, error) in
40        if let error = error {
41          print(error)
42          return
43        }
44        if let data = data, !data.isEmpty {
45          let backToString = String(decoding: data, as: UTF8.self)
46          print("b2S",backToString)
47          DispatchQueue.main.async {
48            sendToScreen.send(backToString)
49          }
50        }
51      }
52    }
53  }
```

sock1.swift hosted with ❤️ by GitHub view raw

This is a class I called `connect` in which I defined a listener that watches the port supplied when called. When you send data through said port, it prints it out. To make use of said code, you just need to call it in your SwiftUI definition:

```
1  Text("Hello, world!")
2    .padding()
3    .onAppear {
```

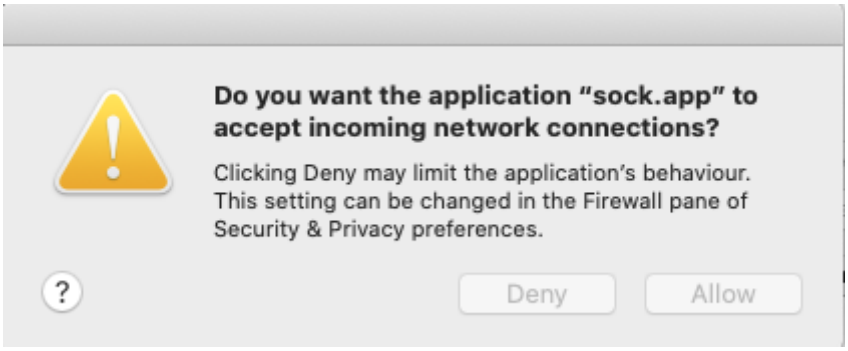
```
    let communication = Connect()
    let port2U = NWEndpoint.Port.init(integerLiteral: 1984)
    communication.listenUDP(port: port2U)
}
.onReceive(sendToScreen, perform: { value in
  comeback = value
})
Text(comeback)
```

listenudp.swift hosted with ❤️ by GitHub view raw

Of course, this is only the listening part. Fortunately, since OS X is UNIX, we can use a command-line tool to test it. This line:

```
echo -n "ping" | nc -4u -w1 192.168.1.110 1984
```

Note that the IP address I'm using here is the IP address of the computer running the simulator (it would be the IP address of your iPhone/iPad if you were running it on a real device). Also, when you run this code, you may get a popup looking like this:



The answer to this question is "Allow."

So go ahead and run it and try the echo. You should see the word "ping" appear on the debug screen (assuming that's what you sent to it, of course).

A Simple Talking Socket

Obviously, you need to be able to talk to your socket through iOS. Doing so is reasonably easy too. Add this code to your `connect` class:

```
func connectToUDP(hostUDP:NWEndpoint.Host,portUDP:NWEndpoint.Port) {
    self.talking = NWConnection(host: hostUDP, port: portUDP, using: .udp)
    self.talking?.stateUpdateHandler = { (newState) in
      switch (newState) {
      case .ready:
        break
      default:
        break
      }
    }
    self.talking?.start(queue: .main)
  }

  func sendUDP(_ content: String) {
    let contentToSendUDP = content.data(using: String.Encoding.utf8)
    self.talking?.send(content: contentToSendUDP, completion: NWConnection.SendCompletic
      if (NWError == nil) {
        // code
      } else {
        print("ERROR! Error when data (Type: String) sending. NWError: \n \(NWError!) ")
      }
    }))
  }
```

sock2.swift hosted with ❤️ by GitHub — view raw

And this code to your SwiftUI file to complete the job:

```
.onTapGesture {
  let communication = Connect()
  let host = NWEndpoint.Host.init("192.168.1.110")
  let port = NWEndpoint.Port.init("1984")
  communication.connectToUDP(hostUDP: host, portUDP: port!)
  communication.sendUDP("pong\n")
}
```

connecttoudp.swift hosted with ❤️ by GitHub — view raw

This sends the word “pong” when you tap on the “Hello World” label. Obviously, the IP address and port (1984) we hardcoded into the app here are again from our simulator/real device.

You’re done… only you’re not because we’re very much in prototype mode right now. You could arguably ship something with a fixed port number for communicating. A fixed IP address is out of the question, though. You most certainly need more.

Networking Using the Bonjour Protocol

Fortunately for us, there is the tried and tested solution that been working for almost 20 years now: a protocol developed by Apple in 2002 (initially, it was called rendezvous). Let’s change the code in our `connect` class to make use of it and banish those hardcoded port numbers/IP addresses into the process (well, almost).

First, we need to rework the listening method to launch a service to query. We obviously also need to redo the connection method to use the newly advertised service:

```
func bonjourUDP(name: String) {
   do {
     self.listening = try NWListener(using: .udp)
     self.listening?.service = NWListener.Service(name:name, type: "_whack._udp", domai
     self.listening?.stateUpdateHandler = {(newState) in
       switch newState {
       case .ready:
         print("ready")
       default:
         break
       }
     }
     self.listening?.serviceRegistrationUpdateHandler = { (serviceChange) in
       switch(serviceChange) {
       case .add(let endpoint):
         switch endpoint {
         case let .service(name, _, _, _):
           print("Service ",name)
         default:
           break
```

```
            }
          default:
            break
          }
        }
        self.listening?.newConnectionHandler = {(newConnection) in
          newConnection.stateUpdateHandler = {newState in
            switch newState {
            case .ready:
              print("new connection")
              self.receive(on: newConnection)
            default:
              break
            }
          }
          newConnection.start(queue: .main)
        }
      } catch {
        print("unable to create listener")
      }
      self.listening?.start(queue: .main)
    }

    func bonjourToUDP(name: String) {
      self.talking = NWConnection(to: .service(name: name, type: "_whack._udp", domain: "l
      self.talking?.stateUpdateHandler = { (newState) in
        switch (newState) {
        case .ready:
          print("ready to send")
        default:
          break
        }
      }
      self.talking?.start(queue: .main)
    }
```

sock3.swift hosted with ❤️ by GitHub view raw

This code registers a bonjour service calling `_whack` that will run over UDP. We leave the IP address and port number problems with the bonjour service.

Having reworked these, we will also need to change the calls to them in the SwiftUI code:

```
.onAppear {
  let communication = Connect()
  communication.bonjourUDP(name: "whack")
}
.onReceive(sendToScreen, perform: { value in
  comeback = value
})
.onTapGesture {
  let communication = Connect()
  communication.bonjourToUDP(name: "whack")
  communication.sendUDP("pong\n")
}
```

bonjourtoudp.swift hosted with ❤️ by GitHub view raw

This creates a service called `"whack"` that runs when we launch the app. It's a service that we subsequently query with the `tap` gesture to make a socket connection.

Note that you can run the following command in the UNIX shell to check if the `whack` service is running (a useful debugging tip):

```
dns-sd -B _whack._udp
```

This brings me to the end of this short article about setting up a connection across a local area network.

Keep calm, keep coding.