Advances in Networking
Part 2

Eric Kinnear, Internet Technologies
Tommy Pauly, Internet Technologies
Stuart Cheshire, Internet Technologies
Part 2

Bonjour

Building Framing Protocols

Collecting Metrics

Best Practices and Status Updates
URLSession
Network.framework
Bonjour
Bonjour Support

iOS  
macOS  
watchOS  
tvOS
Wide-Area Service Discovery

https://github.com/IETF-Hackathon/mDNSResponder
Wide-Area Service Discovery
What this means for your app

When browsing, specify `nil` for domain

Specifying "local" will explicitly prevent any non-local discovery
Bonjour in Network.framework

Server

Advertise a Service
*.myapp._tcp.*
NWListener

Discover Service Endpoints
*.myapp._tcp.*
NWBrowser

Client

Connect to an Endpoint
MyPhone._myapp._tcp.example.com
NWConnection
Service Discovery in Network.framework

Browse for Bonjour service types

Optimized for Swift

Optionally access TXT record details

// Create a browser for "_myapp._tcp" on all domains
let browser = NWBrowser(for: .bonjour(type: "_myapp._tcp", domain: nil),
                        using: NWParameters())

browser.browseResultsChangedHandler = {...}

// Start browsing
browser.start(queue: myQueue)
browser.browseResultsChangedHandler = { __, changes in
    for change in changes {
        switch change {
            case .added(let browseResult):
                print("Added \(browseResult.endpoint)")
            case .removed(let browseResult):
                print("Removed \(browseResult.endpoint)")
            case .changed(_, let browseResult, let flags):
                if flags.contains(.interfaceAdded) {
                    print("\(browseResult.endpoint) added interfaces")
                }
                if flags.contains(.interfaceRemoved) {
                    print("\(browseResult.endpoint) removed interfaces")
                }
            default:
                print("No change")
        }
    }
}
browser.browseResultsChangedHandler = { browseResults, _ in
    // Up-to-date and complete list of browse results
    for browseResult in browseResults {
        print("Discovered \(browseResult.endpoint) over \(browseResult.interfaces)")
    }
}
Tic-Tac-Toe

Advertise games to nearby players
Browse for available games
Use browse results to connect to game
Tic-Tac-Toe

Advertise
- NWListener

Discover
- NWBrowser

Connect
- NWConnection
Building Framing Protocols

Tommy Pauly, Internet Technologies
### Byte-stream

<table>
<thead>
<tr>
<th>type</th>
<th>len</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>&quot;สอน,1,2&quot;</td>
</tr>
</tbody>
</table>

`move` 8

---
Messages, not byte-streams
Application queue

- Application
- Framing
- Security
- Transport

Shared networking workflow

Memory-mapped channel

Interface

Driver

NEW
Message Framing

Application queue

receiveMessage() receiveMessage() receiveMessage()

body body body

Application

Shared networking workloop

Framing

Security

Transport
Framing protocols encapsulate or encode application messages
1. Implement a reusable framing protocol

2. Add the framing protocol to a connection
Implementing Framing Protocols

Define a class that implements `NWProtocolFramerImplementation`

- Encapsulate messages in `handleOutput()`
- Parse bytes into messages in `handleInput()`

Optionally store metadata in `NWProtocolFramer.Message`
// Define a framing protocol

class MyProtocol: NWProtocolFramerImplementation {

    static let definition = NWProtocolFramer.Definition(implementation: MyProtocol.self)

    required init(framer: NWProtocolFramer.Instance) { }
    func start(framer: NWProtocolFramer.Instance) -> NWProtocolFramer.StartResult {
        return .ready
    }
    func wakeup(framer: NWProtocolFramer.Instance) { }
    func stop(framer: NWProtocolFramer.Instance) -> Bool {
        return true
    }
    func cleanup(framer: NWProtocolFramer.Instance) { }

    func handleOutput(...) {
    }
    func handleInput(...) {
        ...
    }
}
// Header-prefixed output handler

func handleOutput(framer: NWProtocolFramer.Instance,
                 message: NWProtocolFramer.Message,
                 messageLength: Int,
                 isComplete: Bool) {

    // Initialize and write the header
    var header: MyHeader = MyHeader(...)
    framer.writeOutput(data: header.serializedData)

    // Write the body
    do {
        try framer.writeOutputNoCopy(length: messageLength)
    } catch let error {
        print("Error writing output: \(error)")
    }
}
```swift
func handleInput(framer: NWProtocolFramer.Instance) -> Int {
    while true {
        // Read out the header
        var header: MyHeader
        let headerSize = MyHeader.serializedLength
        let parsed = framer.parseInput(minimumIncompleteLength: headerSize,
                                        maximumLength: headerSize) { (buffer, isComplete) -> Int in
            // Try to parse header, and move the cursor forward by headerSize
            return headerSize
        }
        // If we couldn’t read, return and wait for more bytes
        if !parsed { return headerSize }
        // Create a message object
        let message = NWProtocolFramer.Message(instance: framer)
        // Deliver the body and message to application
        if !framer.deliverInputNoCopy(length: length, message: message, isComplete: true) {
            return 0 // Waiting to deliver message, stop looping
        }
    }
    return 0 // Waiting to deliver message, stop looping
}
```
Using Framing Protocols

Add one or more framers to your `NWProtocolStack`

```swift
let framerOptions = NWProtocolFramer.Options(definition: MyProtocol.definition)

// Add the framer to NWParameters above TLS
let parameters = NWParameters.tls
parameters.defaultProtocolStack.applicationProtocols.insert(framerOptions, at: 0)
```

Add `NWProtocolWebSocket.Options()` in the same way
Using Framing Protocols

Use framing protocols to write common code across TCP and UDP transports.
Using Framing Protocols

Send message values

```swift
let message = NWProtocolFramer.Message(definition: MyProtocol.definition)
message["key"] = "value"

let sendContext = NWConnection.ContentContext(identifier: "custom", metadata: [message])
connection.send(content: data, contentContext: sendContext,
               isComplete: true, completion: .contentProcessed { error in
               // Check for errors
               })
```
Using Framing Protocols

Receive message values

```swift
connection.receiveMessage { (content, context, isComplete, error) in
    if let context = context,
        let message = context.protocolMetadata(definition: MyProtocol.definition) as?
            NWProtocolFramer.Message,
        let value = message["key"] {
            // Use value
        }
}
```
Use `.willMarkReady` to implement a protocol handshake.

Call `prependApplicationProtocol()` to dynamically build a protocol stack.
Dynamic Protocol Stacks

Use `.willMarkReady` to implement a protocol handshake.

Call `prependApplicationProtocol()` to dynamically build a protocol stack.

Diagram:
- Application
- TLS
- STARTTLS
- TCP
- `ready`
Collecting Metrics
Collecting Metrics

- URLSession
  - Connection Properties
  - Request/Response Metrics

- Network.framework
  - Establishment Report
  - Data Transfer Report
URLSession Task Metrics

Connection Properties

- DNS
- TCP
- TLS

Response and Request Metrics

- fetchStart
- requestStart
- responseStart
- GET
- 200 OK
- Response Body
Network.framework Metrics

Establishment Report

DNS  TCP  TLS

.ready
Calls to send() and receive()

.cancelled

Data Transfer Reports
Connection Properties
URLSession

Available in the **didFinishCollectingMetrics** delegate method

```swift
if let transactionMetrics = metrics.transactionMetrics.last {
    // Access local and remote endpoints
    print("From \(transactionMetrics.localAddress) Port \(transactionMetrics.localPort)")
    print("To \(transactionMetrics.remoteAddress) Port \(transactionMetrics.remotePort)")

    // Check security properties
    if transactionMetrics.negotiatedTLSProtocolVersion == .TLSv13 {
        print("Used TLS 1.3")
    }

    // Inspect path properties
    print("Used constrained path: \(transactionMetrics.isConstrained)")
}
```
Establishment Report
Network.framework

Available after connection moves to the .ready state

DNS resolution source, latency, and address racing result

Protocol handshake duration and round-trip-time for TCP and TLS

Proxy information
// Request establishment report on a connection
connection.requestEstablishmentReport(queue: .main) { report in

    // Ensure that the report is ready
    if let report = report {
        print("Connection took \(report.duration) seconds to establish")

        for resolution in report.resolutions {
            print("Resolved \(resolution.endpointCount) endpoints from \\
            \(resolution.source) in \(resolution.duration) seconds")
        }

        for handshake in report.handshakes {
            print("Handshake for \(handshake.definition) took \\
            \(handshake.handshakeDuration) seconds")
        }
    }
}
Optimistic DNS

Optimistic DNS is now enabled by default

Improves performance for answers with short times-to-live

```java
// NWConnection.EstablishmentReport.Resolution
public enum Source {
    case query
    case cache
    case expiredCache
}
```
Demo
Access header and body bytes counts

```swift
if let transactionMetrics = metrics.transactionMetrics.last {
    print("Sent \(transactionMetrics.countOfRequestHeaderBytesSent) header bytes")
    print("Sent \(transactionMetrics.countOfRequestBodyBytesSent) encoded body bytes")
    print("Sent \(transactionMetrics.countOfRequestBodyBytesBeforeEncoding) body bytes")
    print("Read \(transactionMetrics.countOfResponseHeaderBytesReceived) header bytes")
    print("Read \(transactionMetrics.countOfResponseBodyBytesReceived) encoded body bytes")
    print("Read \(transactionMetrics.countOfResponseBodyBytesAfterDecoding) body bytes")
}
```
Data Transfer Report
Network.framework

Retrieve byte counts, packet counts, and round-trip-times
Start and end reports to correspond to application activity
Multiple reports supported simultaneously
Per-path breakdown for multi-path protocols
// Start a data transfer report
let pendingReport = connection.startDataTransferReport()

// Later, collect the report
pendingReport.collect(queue: .main) { report in
    let aggregateReport = report.aggregatePathReport

    print("Sent \(aggregateReport.sentIPPacketCount) packets")
    print("Received \(aggregateReport.receivedIPPacketCount) packets")

    print("Sent \(aggregateReport.sentTransportByteCount) bytes")
    print("Received \(aggregateReport.receivedTransportByteCount) bytes")

    print("RTT is \(aggregateReport.transportSmoothedRTT) seconds")
}
Best Practices and Status Updates

Stuart Cheshire, Internet Technologies
iPad Apps for Mac
iPad Apps for Mac
Networking on watchOS

Audio streaming apps that use `AVFoundation` can use direct networking. Use `URLSessionStreamTask` or `NWConnection` for TLS, TCP, and UDP. Traffic will either proxy through the phone, or use Wi-Fi or Cellular directly.
# Performance and Privacy Improvements with TLS 1.3

IETF RFC 8446, August 2018

<table>
<thead>
<tr>
<th></th>
<th>TLS 1.2</th>
<th>TLS 1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection setup performance</strong></td>
<td>Two round trips</td>
<td>One round trip (almost always)</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Weak cryptographic algorithms</td>
<td>AEAD with Forward Secrecy</td>
</tr>
<tr>
<td><strong>Privacy</strong></td>
<td>Certificates, most handshake fields, and Server Name Indication are cleartext</td>
<td>Certificates and most handshake fields are encrypted. Work on Encrypted SNI underway</td>
</tr>
</tbody>
</table>
CNCopyCurrentNetworkInfo

Requires Capability: Access Wi-Fi Information

Must also meet at least one of criteria below

• Apps with permission to access location
• Currently enabled VPN app
• NEHotspotConfiguration (only Wi-Fi networks that the app configured)

Otherwise, returns nil
Network Link Conditioner
Alternatives to Networking Pre-Flight Checks

Pre-flight checks have inherent race conditions

We still see many apps that do this

Instead of pre-flight checks, attach constraints to your operations
Get on Wi-Fi and then click Connect.
allowsExpensiveNetworkAccess = false

waitsForConnectivity = true

taskIsWaitingForConnectivity
Deprecations

PAC files for schemes `file://` and `ftp://`

SPDY
• Replaced by HTTP/2

Secure Transport
• Does not support TLS 1.3
• Use URLSession or Network.framework instead
Summary — Part 1

Low Data Mode
Combine in URLSession
WebSocket
Mobility Improvements
Summary — Part 2

Bonjour

Building Framing Protocols

Collecting Metrics

Best Practices and Status Updates
## More Information


<table>
<thead>
<tr>
<th>Event</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advances in Networking, Part 1</td>
<td>Thursday, 11:00</td>
</tr>
<tr>
<td>Network Extensions for Modern macOS</td>
<td>Friday, 9:00</td>
</tr>
<tr>
<td>Networking Lab</td>
<td>Friday, 9:00</td>
</tr>
</tbody>
</table>