#WWDC18

Introducing Network.framework

A modern alternative to sockets

Session 715

Josh Graessley, Networking
Tommy Pauly, Networking
Eric Kinnear, Networking

Modernizing transport APIs

Making your first connections

Optimizing data transfer

Solving network mobility

Getting involved

Modernizing Transport APIs

Josh Graessley, Networking

Using sockets to write apps for today’s Internet is hard.

Connection Establishment

Hostname Resolution

NAT64

Dual-Stack Hosts

Bonjour Service Resolution

Optimistic DNS

Connection Establishment

PAC Evaluation

Cellular Radio Bringup

Proxies

VPN Configurations

Data Transfer

Transport Layer Security

Writable Events

Backpressure

Data Transfer

Datagram Batches

Low Water Marks

Sending Early Data

Reading Complete Headers

Mobility

Wait for Reachability

Network Transitions

Trigger VPN

Mobility

Avoid Cellular Networks

Wi-Fi Assist

Multipath TCP

`getaddrinfo`

`SecureTransport`

`socket`

`SCNetworkReachability`

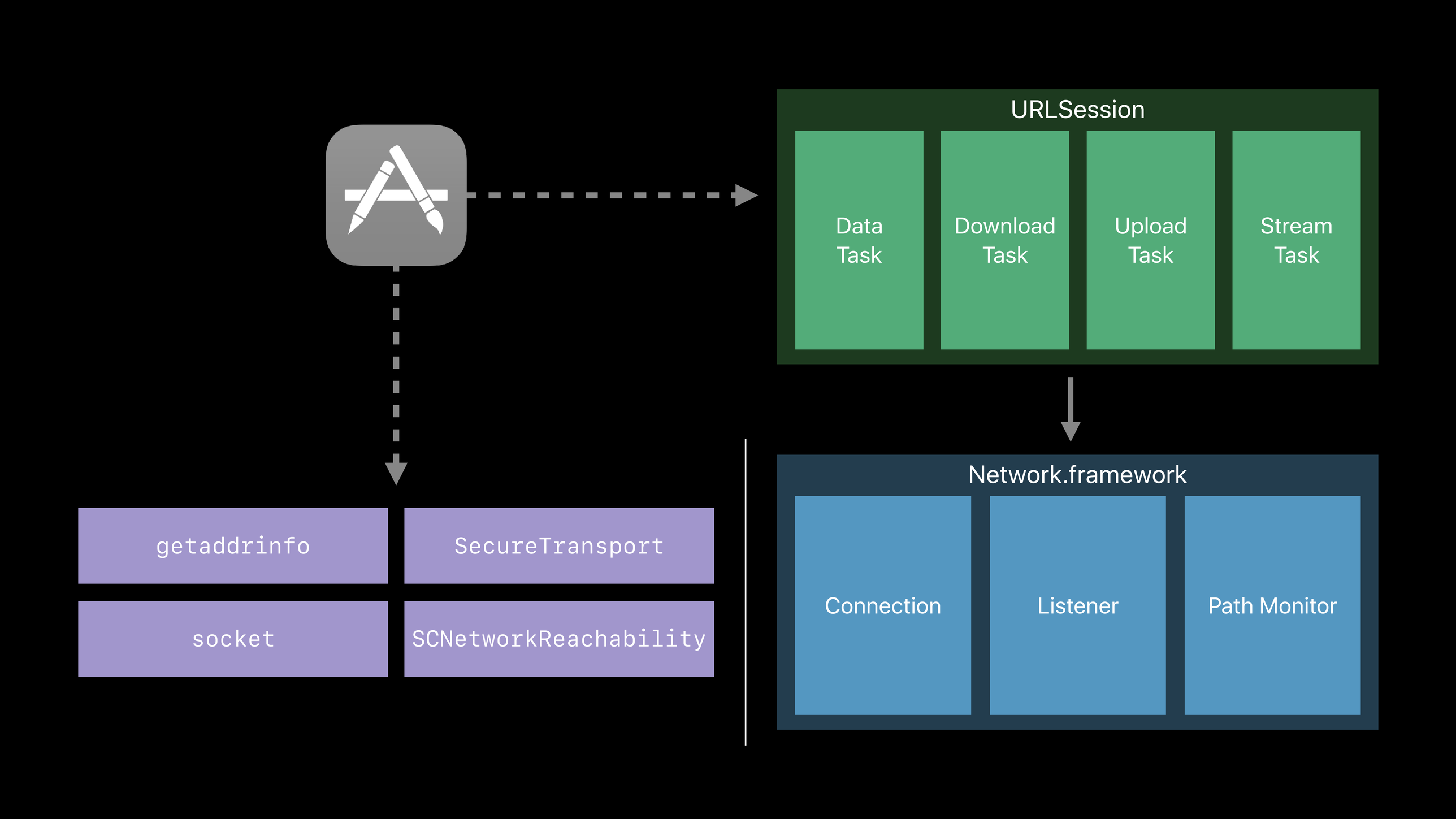
URLSession

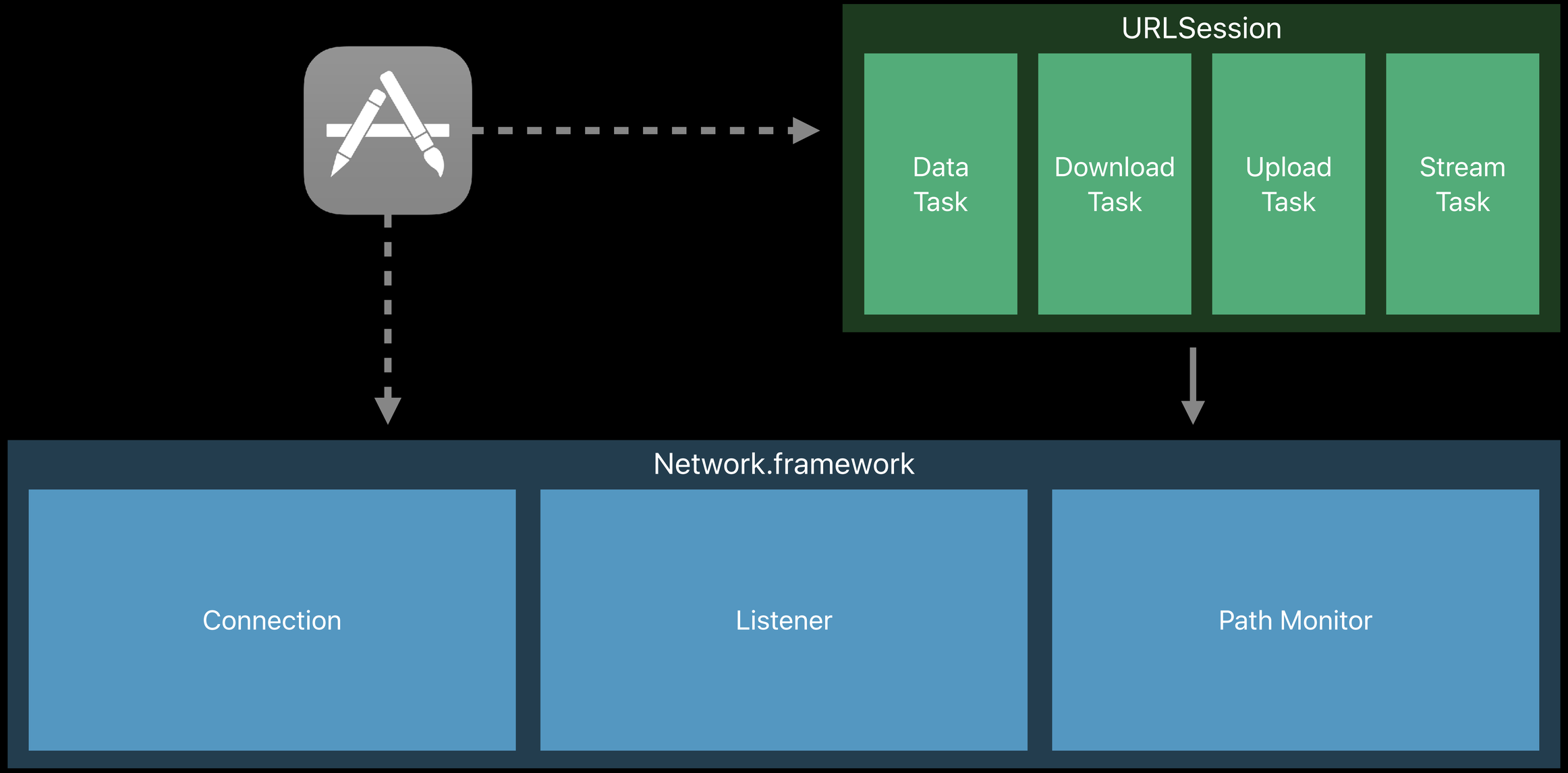
Data Task

Download Task

Upload Task

Stream Task


URLSession
Data Task
Download Task
Upload Task
Stream Task
Network.framework
Connection
Listener
Path Monitor
getaddrinfo
SecureTransport
socket
SCNetworkReachability


URLSession
Data Task
Download Task
Upload Task
Stream Task
Network.framework
Connection
Listener
Path Monitor

Introducing Network.framework

NEW

Smart connection establishment

Optimized data transfer

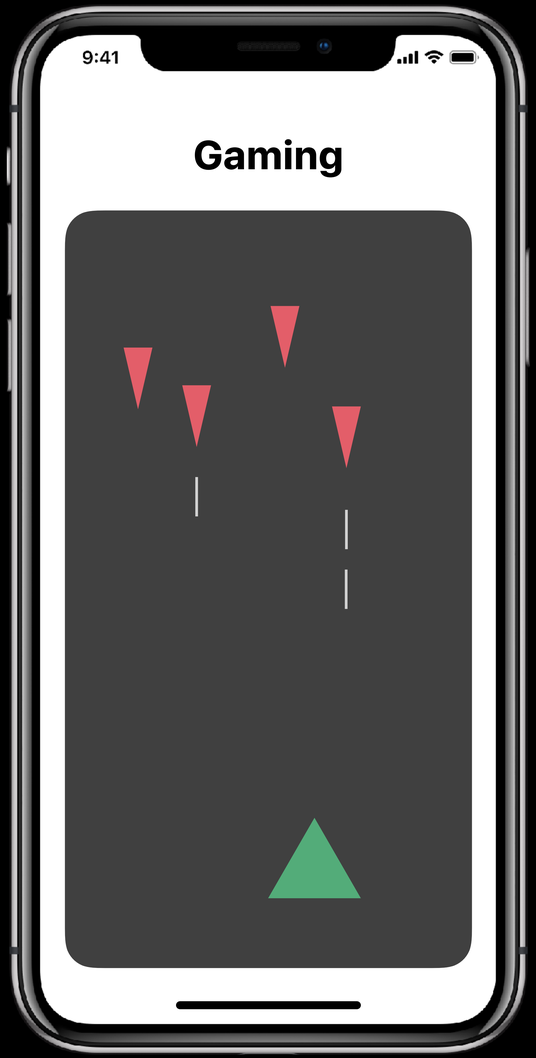
Built-in security

Seamless mobility

Native Swift support

Making Your First Connections

Tommy Pauly, Networking


9:41
Gaming


9:41
Live Streaming


9:41
Mail and
Messaging

Connection Setup

Hostname: mail.example.com

Port: 993

Protocol: TLS/TCP

Connection Setup

Sockets

1. Perform DNS resolution with `getaddrinfo()`
2. Call `socket()` with the correct address family
3. Set socket options with `setsockopt()`
4. Call `connect()` to start TCP
5. Wait for a writable event

Connection Setup

Network.framework

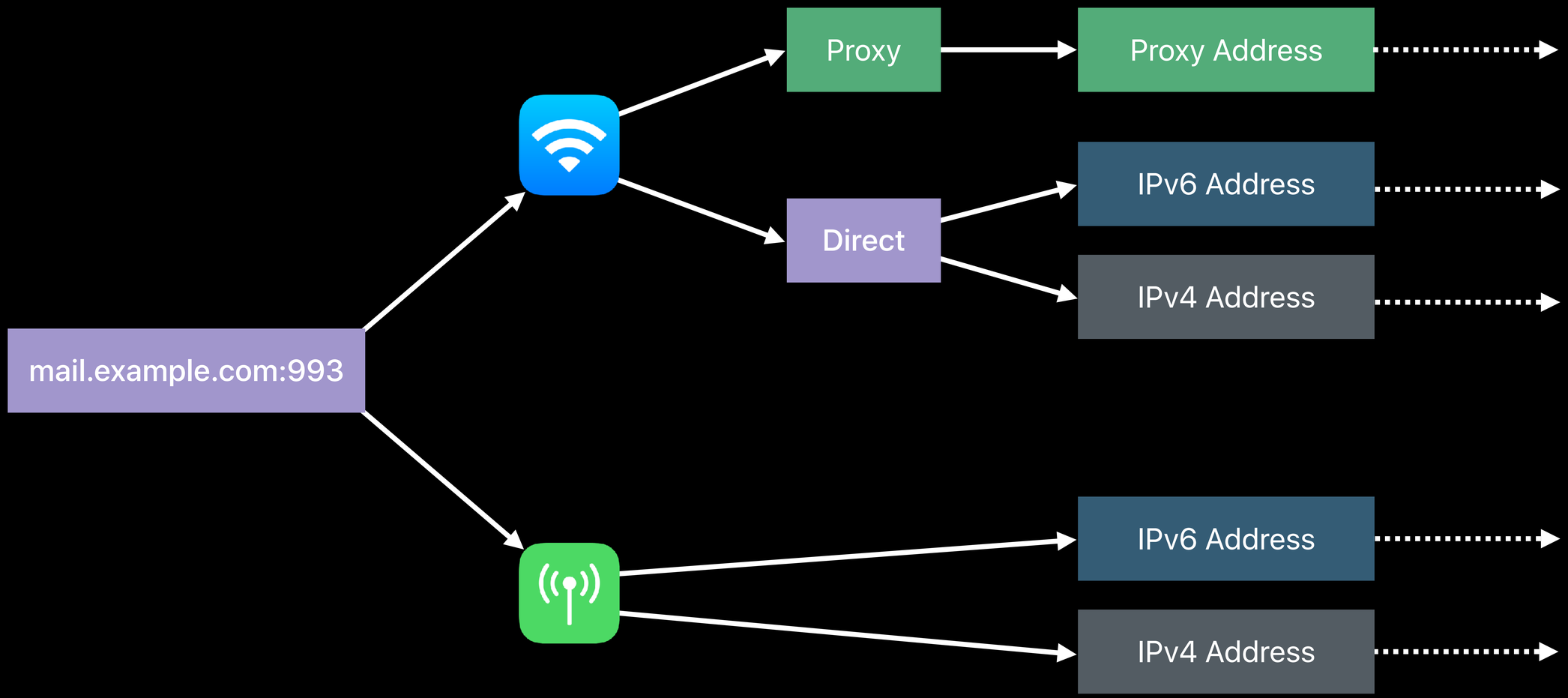
1. Create a connection to an `NWEndpoint` and `NWParameters`
2. Call `connection.start()`
3. Wait for connection to move to the `.ready` state

```
// Create an outbound connection
import Network
let connection = NWConnection(host: "mail.example.com", port: .imaps, using: .tls)

connection.stateUpdateHandler = { (newState) in
    switch(newState) {
    case .ready:
        // Handle connection established
    case .waiting(let error):
        // Handle connection waiting for network
    case .failed(let error):
        // Handle fatal connection error
    default:
        break
    }
}

connection.start(queue: myQueue)
```

Connection Lifecycle

Connection Lifecycle

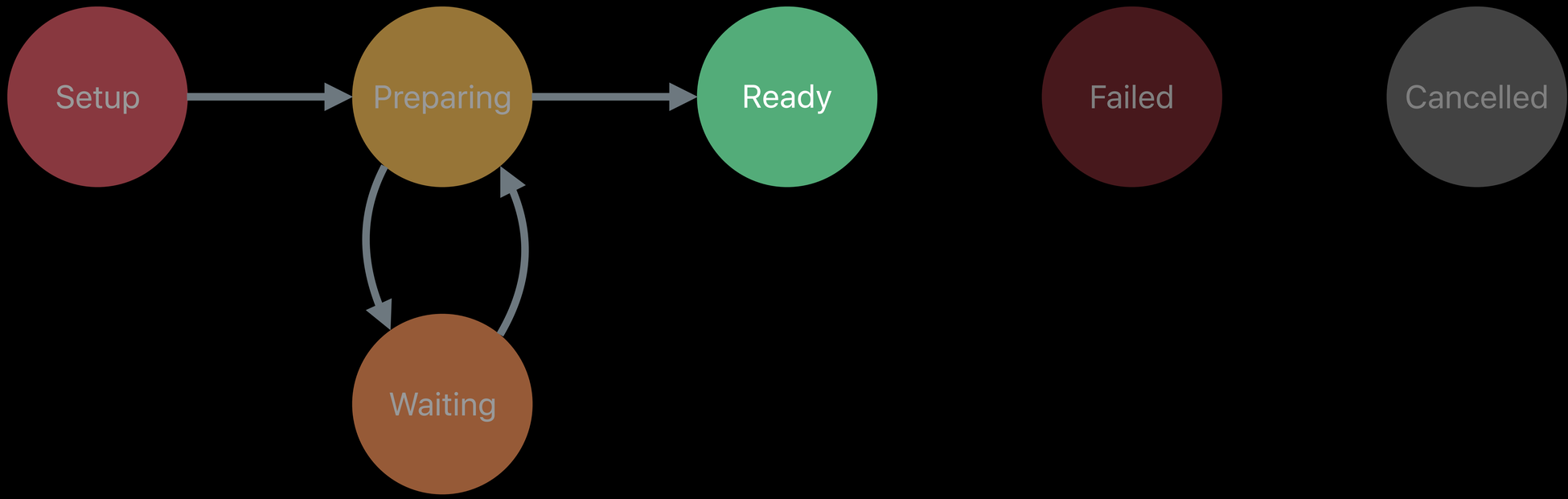
Smart Connection Establishment



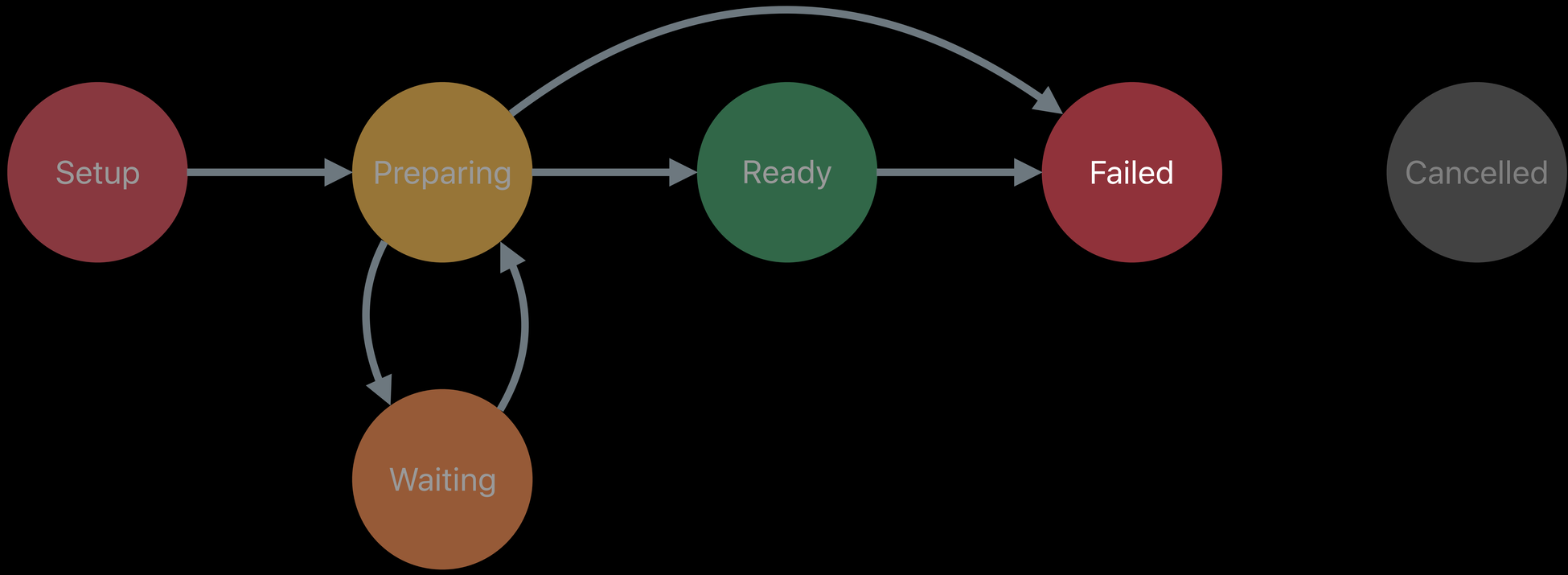
```
// Limiting Connection Establishment

// Restrict connections based on interface types
parameters.prohibitedInterfaceTypes = [ .cellular ]

// Restrict connections based on address family
if let ipOptions = parameters.defaultProtocolStack.internetProtocol as? NWProtocolIP.Options {
    ipOptions.version = .v6
}

// Avoid proxies
parameters.preferNoProxies = true
```

Connection Lifecycle

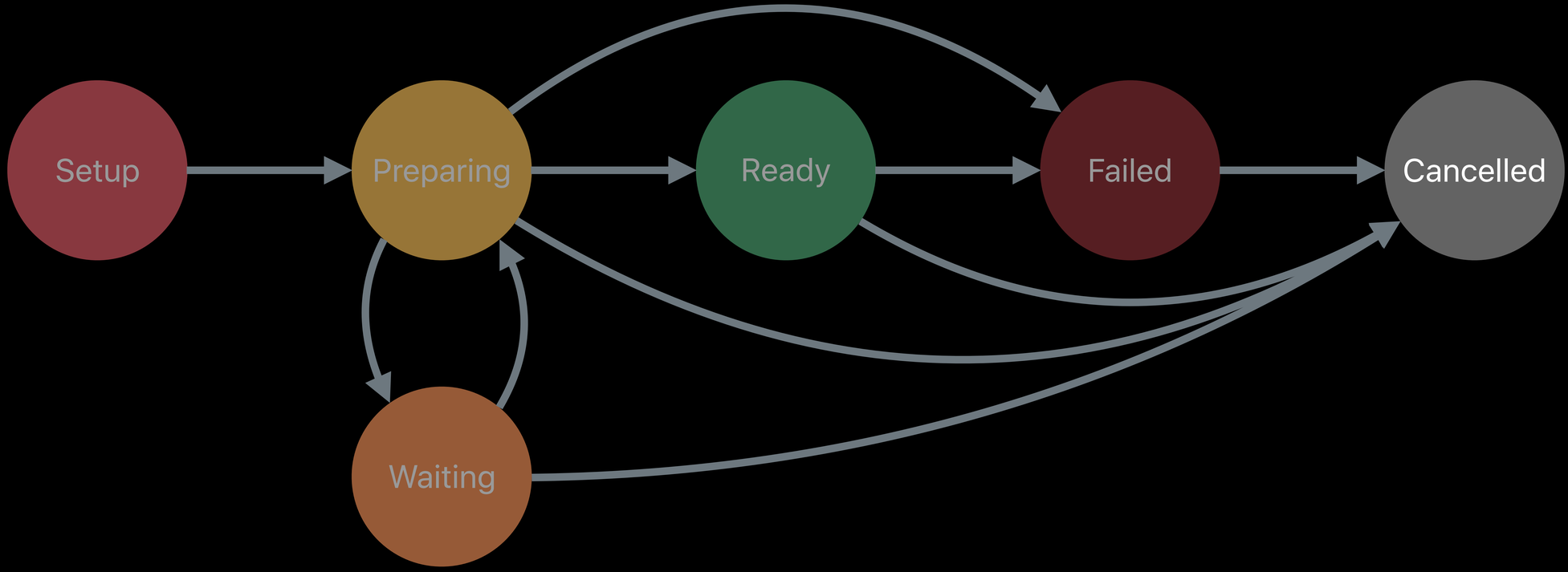
Connection Lifecycle



Connection Lifecycle



Connection Lifecycle



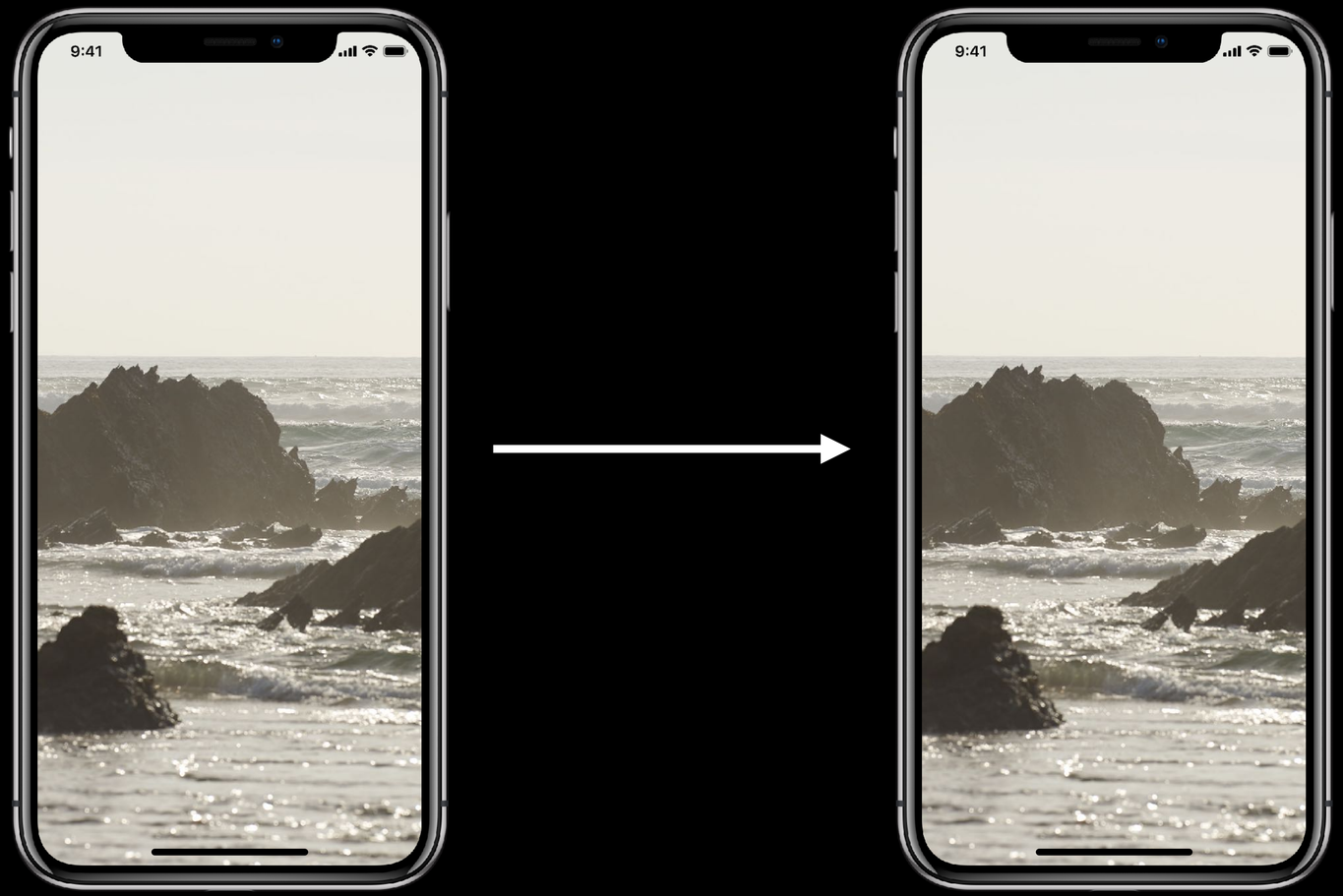
Example

Streaming Video

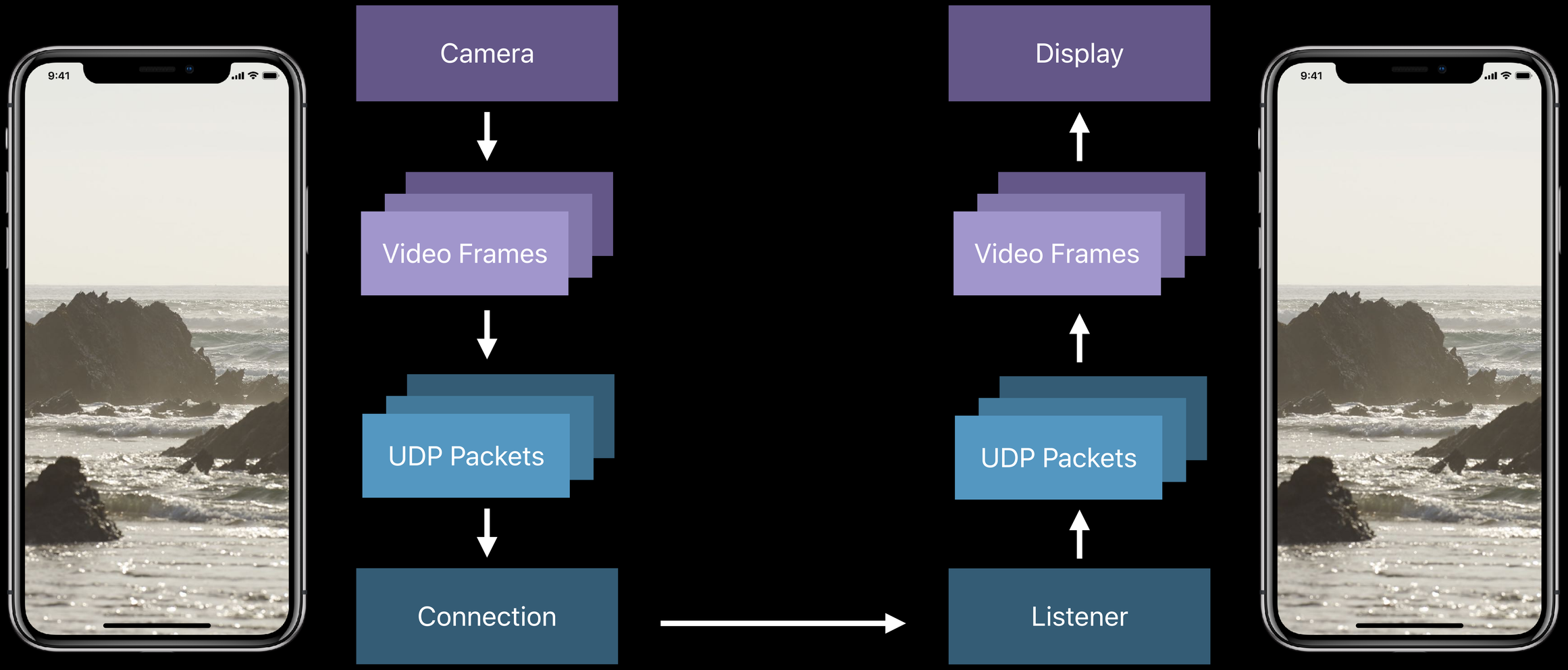
Eric Kinnear, Networking


9:41
Live Streaming

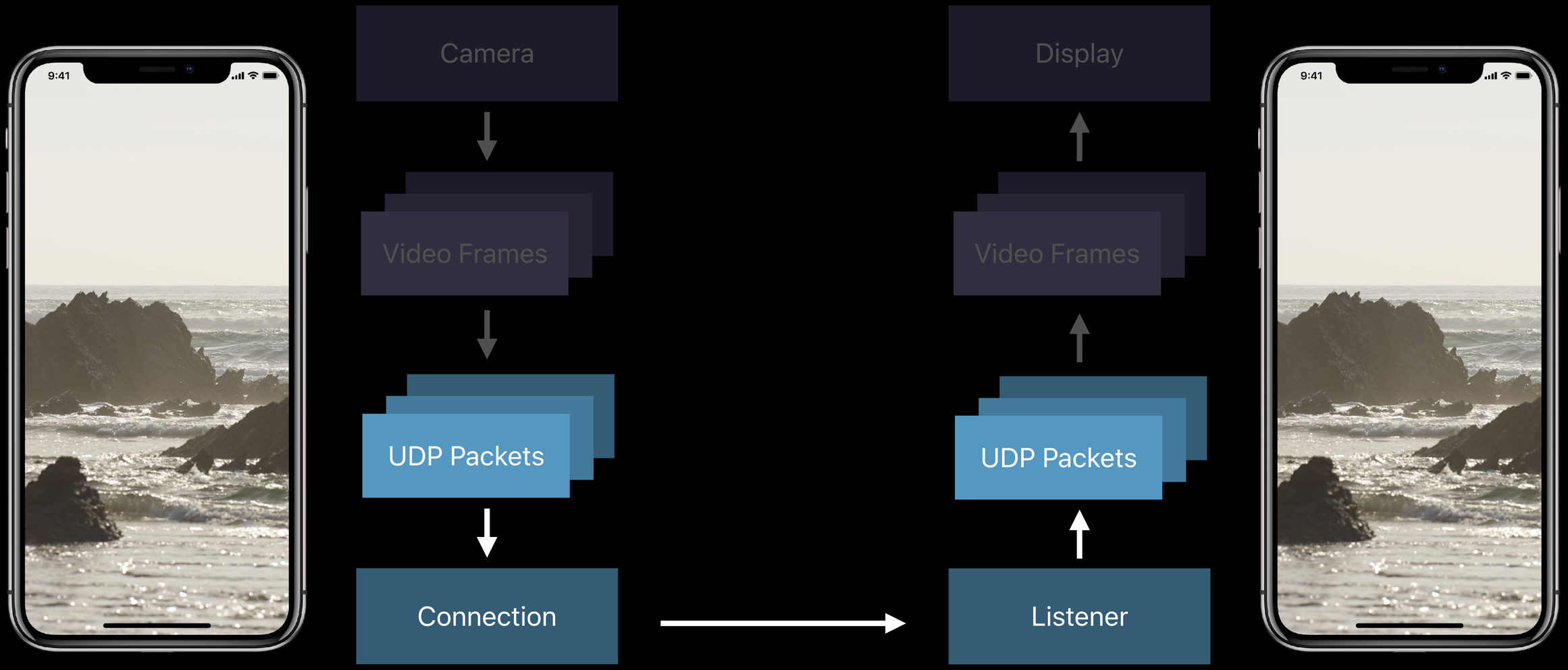
Streaming Live Video with UDP



Streaming Live Video with UDP



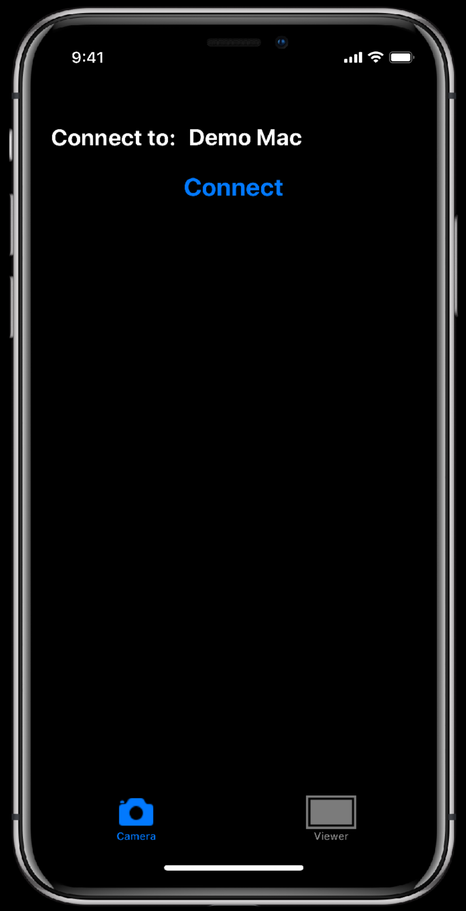
Streaming Live Video with UDP

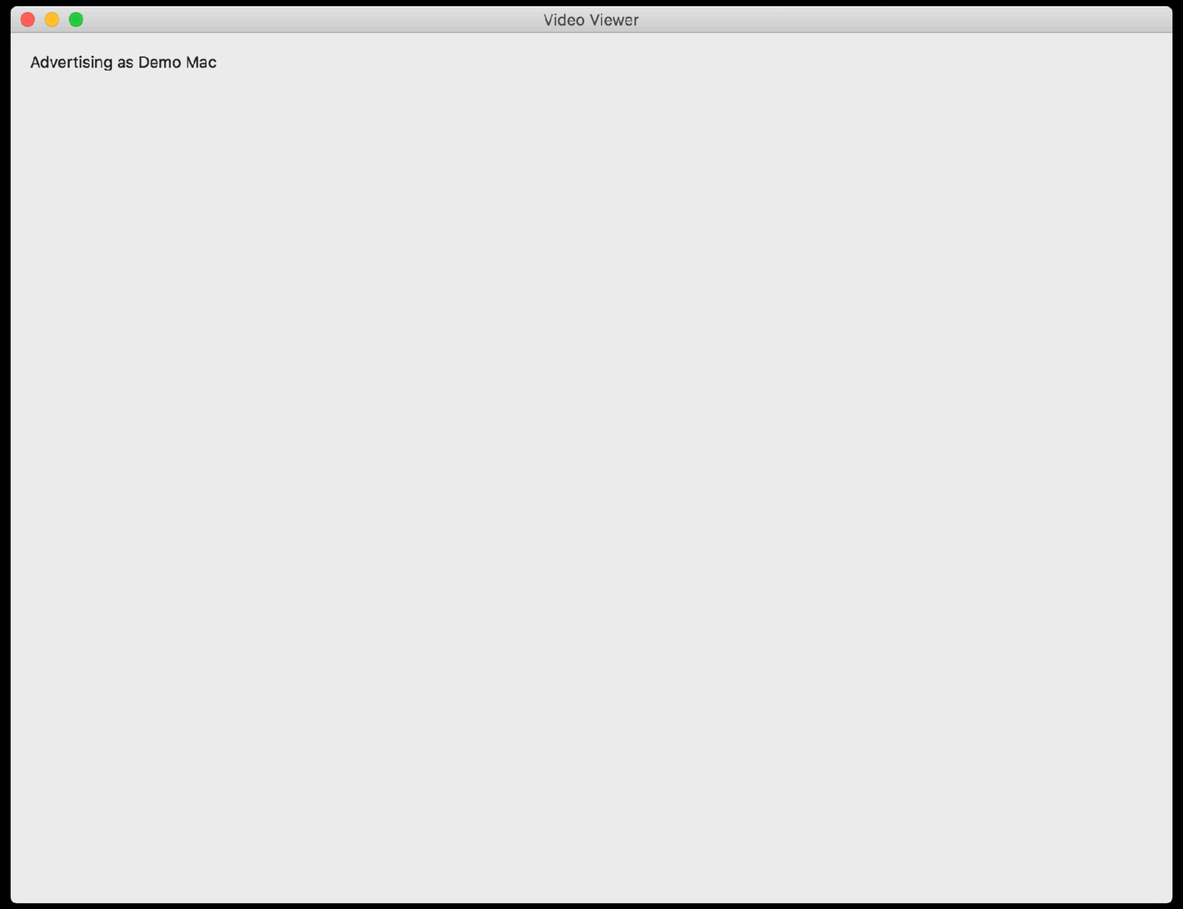


```
// UDP Bonjour listener

do {
    if let listener = try NWListener(parameters: .udp) {

        // Advertise a Bonjour service
        listener.service = NWListener.Service(type: "_camera._udp")

        listener.newConnectionHandler = { (newConnection) in
            // Handle inbound connections
            newConnection.start(queue: myQueue)
        }

        listener.start(queue: myQueue)
    }
} catch {
    // Handle listener creation error
}
```

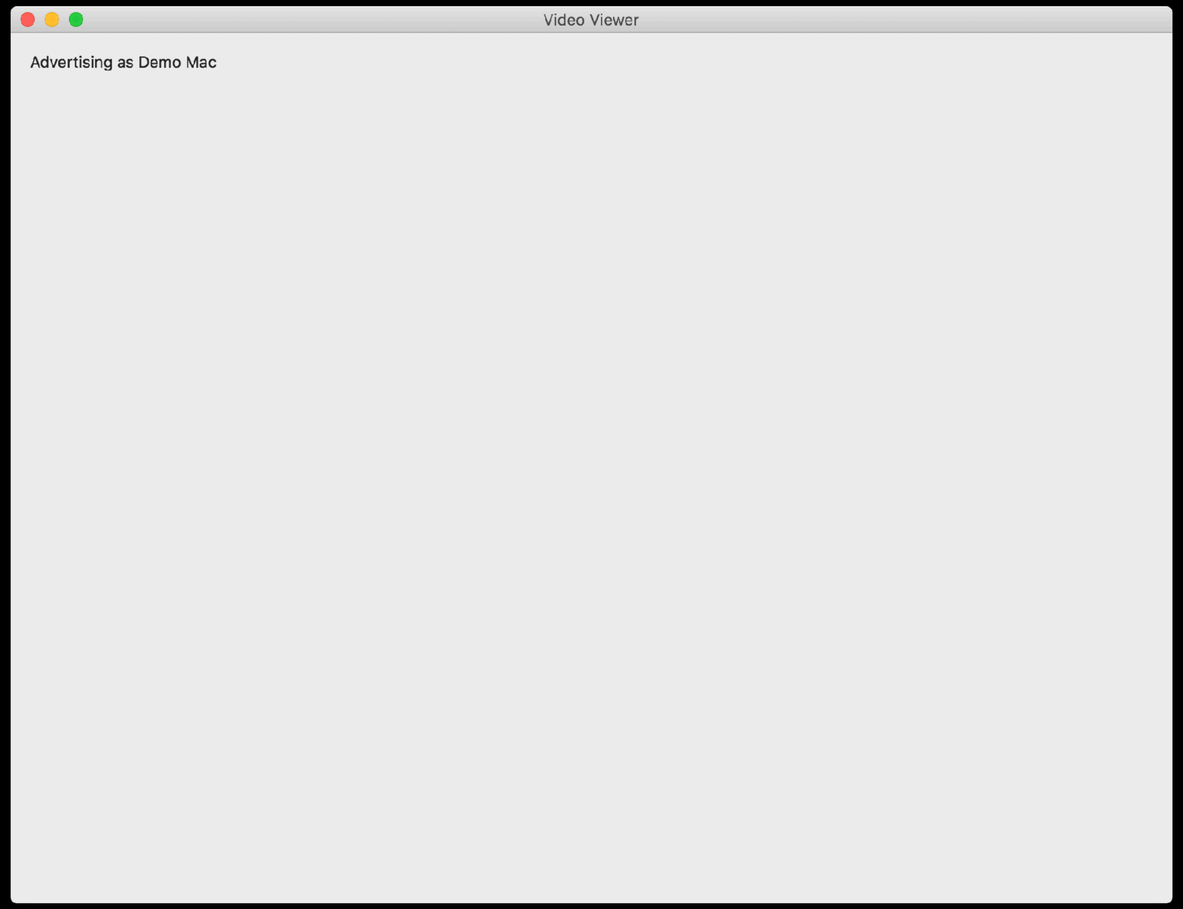
Demo

Streaming Live Video with UDP

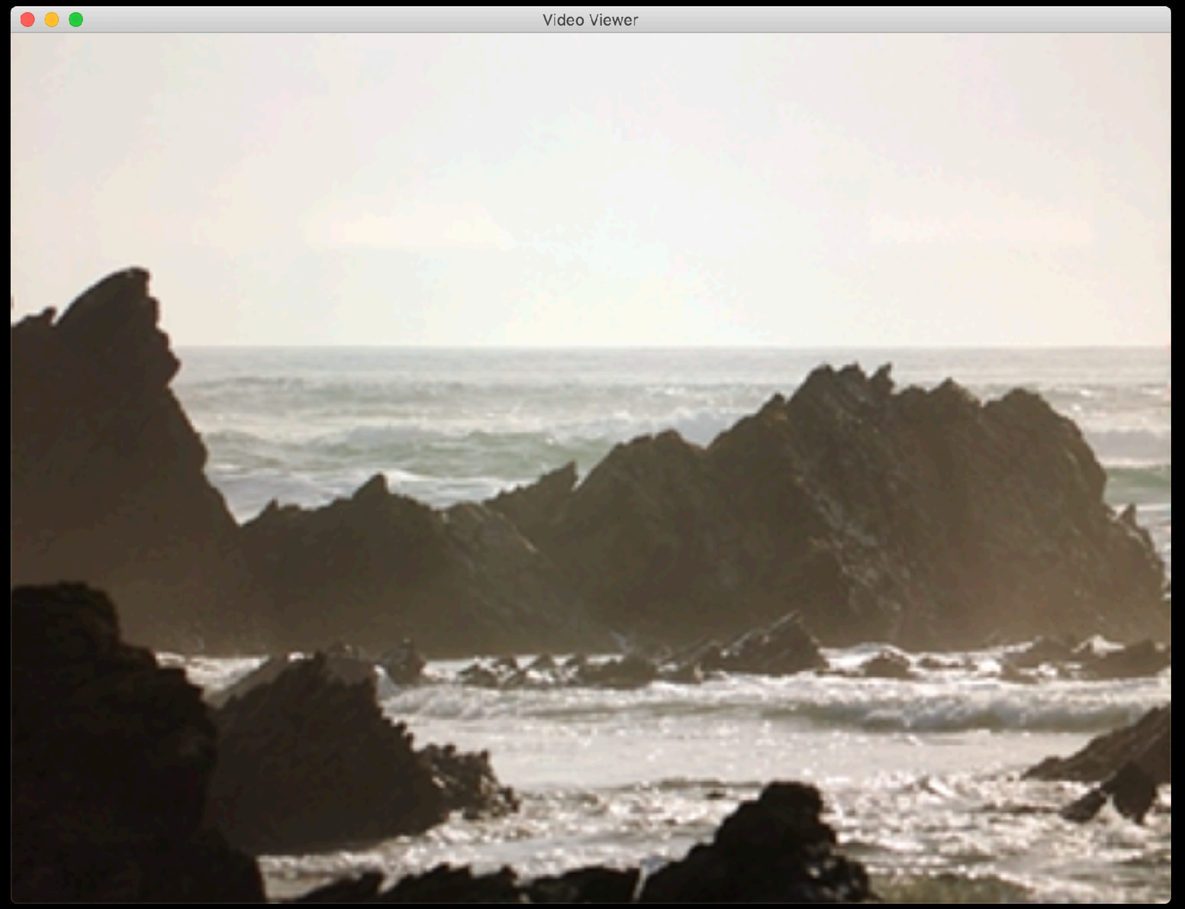




Streaming Live Video with UDP



Streaming Live Video with UDP



Optimizing Data Transfer

Tommy Pauly, Networking

Send and Receive

```swift
// Send a single frame
func sendFrame(_ connection: NWConnection, frame: Data) {

    // The .contentProcessed completion provides sender-side back-pressure
    connection.send(content: frame, completion: .contentProcessed { (sendError) in

        if let sendError = sendError {
            // Handle error in sending

        } else {
            // Send has been processed, send the next frame
            let nextFrame = generateNextFrame()
            sendFrame(connection, frame: nextFrame)
        }
    })
}
```

```
// Hint that multiple datagrams should be sent as one batch
connection.batch {

    for datagram in datagramArray {
        connection.send(content: datagramArray, completion: .contentProcessed { (error) in
            // Handle error in sending
        })
    }
}
```

```swift
// Read one header from the connection
func readHeader(connection: NWConnection) {
    // Read exactly the length of the header
    let headerLength: Int = 10
    connection.receive(minimumIncompleteLength: headerLength, maximumLength: headerLength)
            { (content, contentContext, isComplete, error) in
        if let error = error {
            // Handle error in reading

        } else {
            // Parse out body length
            readBody(connection, bodyLength: bodyLength)
        }
    }
}

// Follow the same pattern as readHeader() to read exactly the body length
func readBody(_ connection: NWConnection, bodyLength: Int) { ... }
```

Advanced Options

Explicit Congestion Notification

ECN negotiation is enabled by default on TCP connections

Mark ECN flags per packet with UDP

```
let ipMetadata = NWProtocolIP.Metadata()
ipMetadata.ecn = .ect0

let context = NWConnection.ContentContext(identifier: "ECN", metadata: [ ipMetadata ])

connection.send(content: datagram, contentContext: context, completion: .contentProcessed{..})
```

Service Class

Interface queuing and Cisco Fastlane

Mark service class on parameters to apply to an entire connection

```
let parameters = NWParameters.tls
parameters.serviceClass = .background
```

Mark service class per-packet for UDP

```
let ipMetadata = NWProtocolIP.Metadata()
ipMetadata.serviceClass = .signaling

let context = NWConnection.ContentContext(identifier: "Signaling", metadata: [ ipMetadata ])
connection.send(content: datagram, contentContext: context, completion: .contentProcessed{..})
```

Fast Open Connections

Zero round trip data

Allowing fast open on a connection requires sending idempotent data

```
parameters.allowFastOpen = true
let connection = NWConnection(to: endpoint, using: parameters)

connection.send(content: initialData, completion: .idempotent)

connection.start(queue: myQueue)
```

Fast Open Connections

Zero round trip data

Allowing fast open on a connection requires sending idempotent data

```
parameters.allowFastOpen = true
let connection = NWConnection(to: endpoint, using: parameters)

connection.send(content: initialData, completion: .idempotent)

connection.start(queue: myQueue)
```

TCP Fast Open may be manually enabled to run TLS over TFO

```
let tcpOptions = NWProtocolTCP.Options()
tcpOptions.enableFastOpen = true
```

Allow Expired DNS Answers

Remove DNS round trip time

Optimistically try expired DNS answers

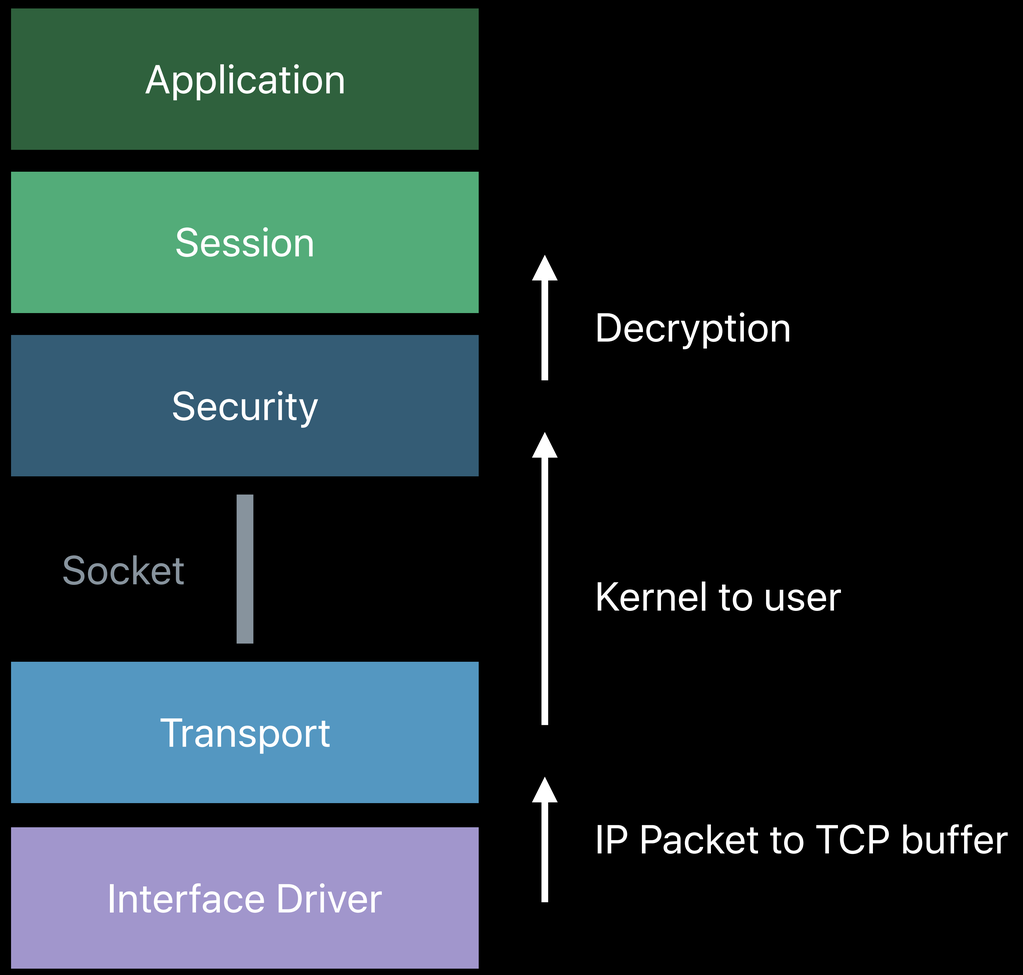
```
parameters.expiredDNSBehavior = .allow
let connection = NWConnection(to: endpoint, using: parameters)

connection.start(queue: myQueue)
```

A DNS query for a new answer will run in parallel

User-Space Networking

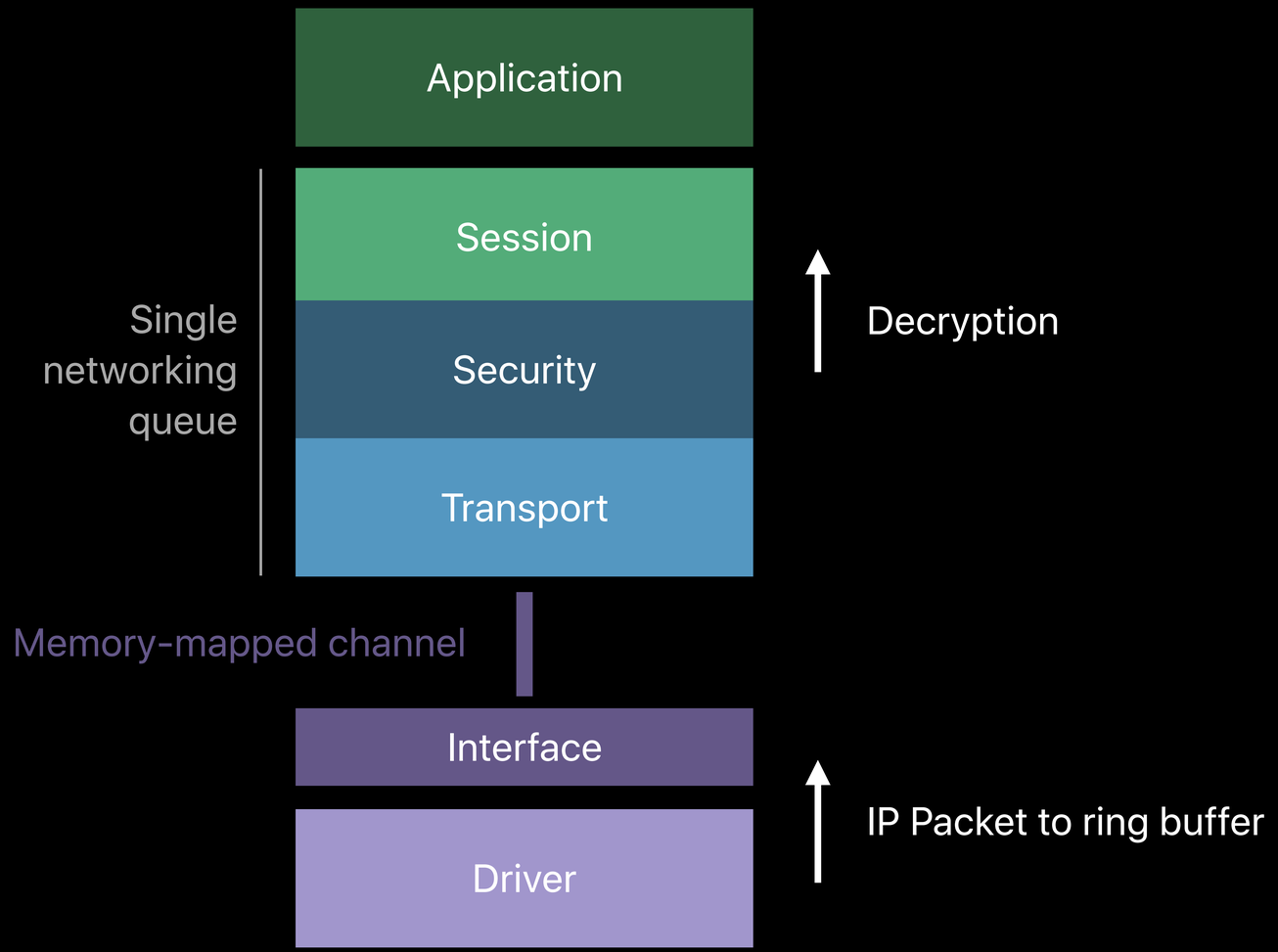
User-Space Networking

Legacy socket model



User-Space Networking

URLSession and Network.framework



Demo

UDP performance

Sockets
UDP

User-Space
UDP

~30% less overhead

Solving Network Mobility

Starting Connections

`.waiting` state indicates lack of connectivity

Avoid checking reachability before starting a connection

Restrict interface types in `NWParameters` if necessary

Reacting to Network Transitions

Connection viability

Current Path

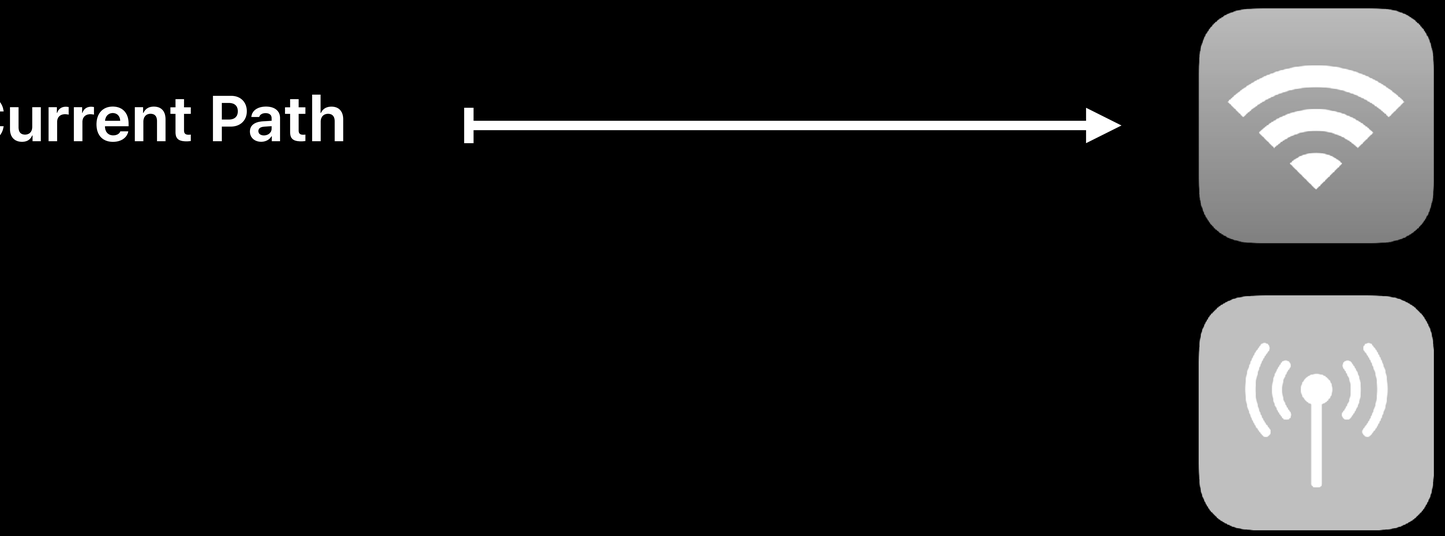
```
isViable = true
```

```
betterPathAvailable = false
```

Reacting to Network Transitions

Connection viability

Current Path



`isViable = false`

`betterPathAvailable = false`

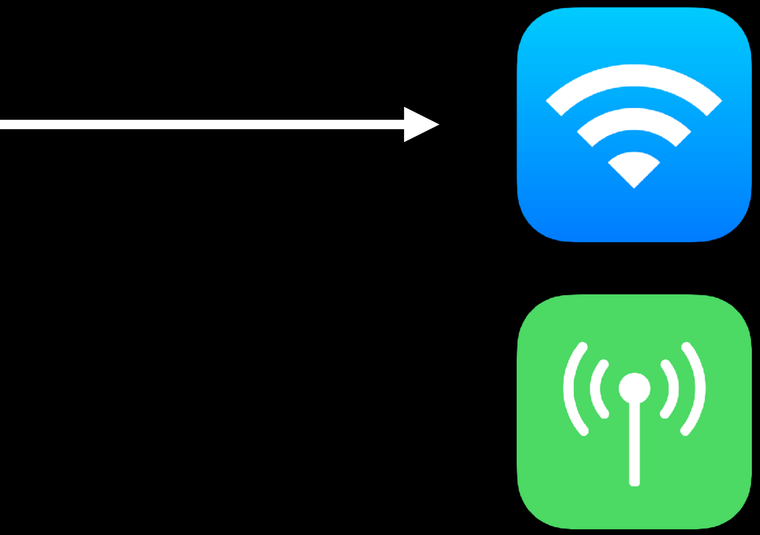
Inform user about no connectivity

Do not close connection

Reacting to Network Transitions

Better path

Current Path



`isViable = true`

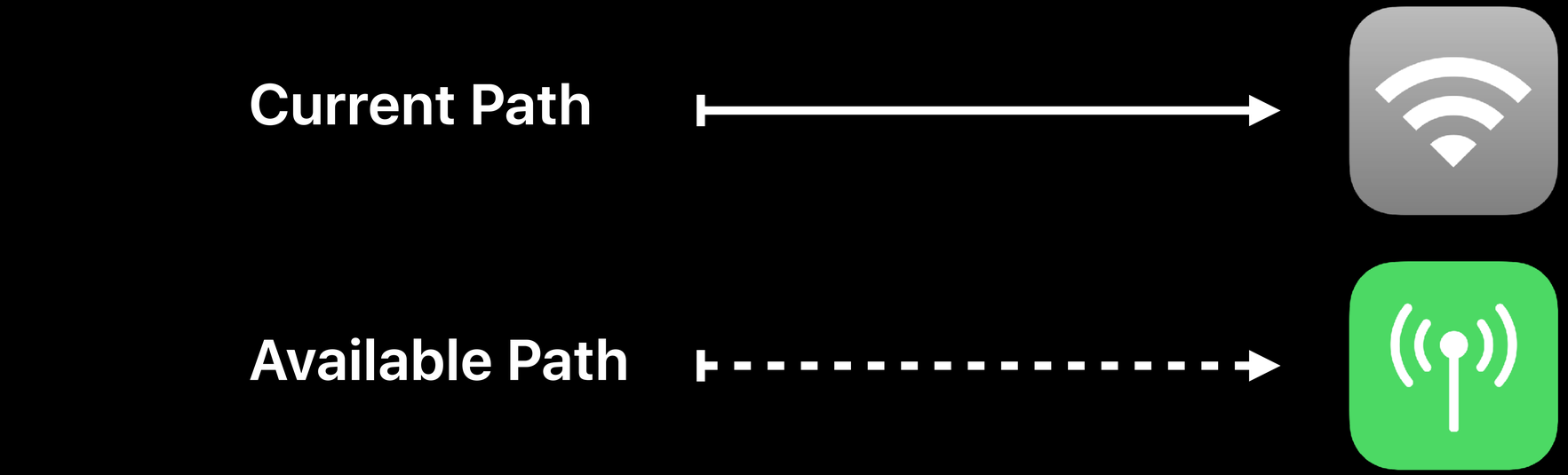
`betterPathAvailable = false`

Reacting to Network Transitions

Better path

Current Path

Available Path



`isViable = false`

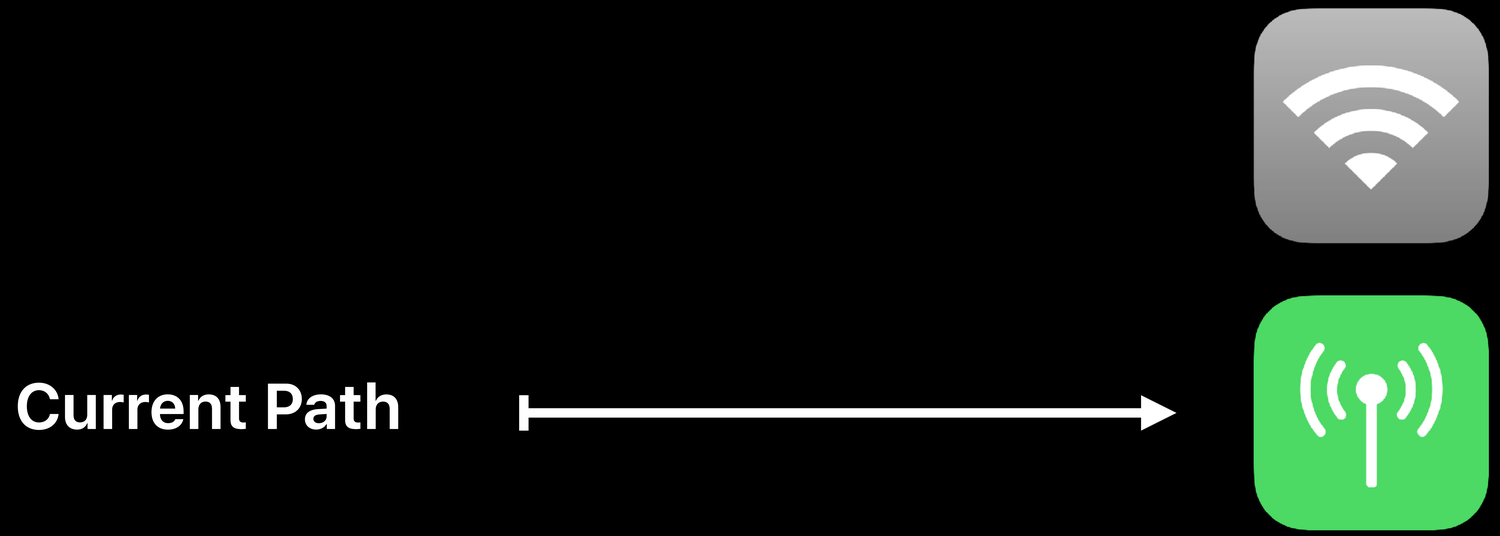
`betterPathAvailable = true`

Attempt new connection

Close original connection once new connection is ready

Reacting to Network Transitions

Better path

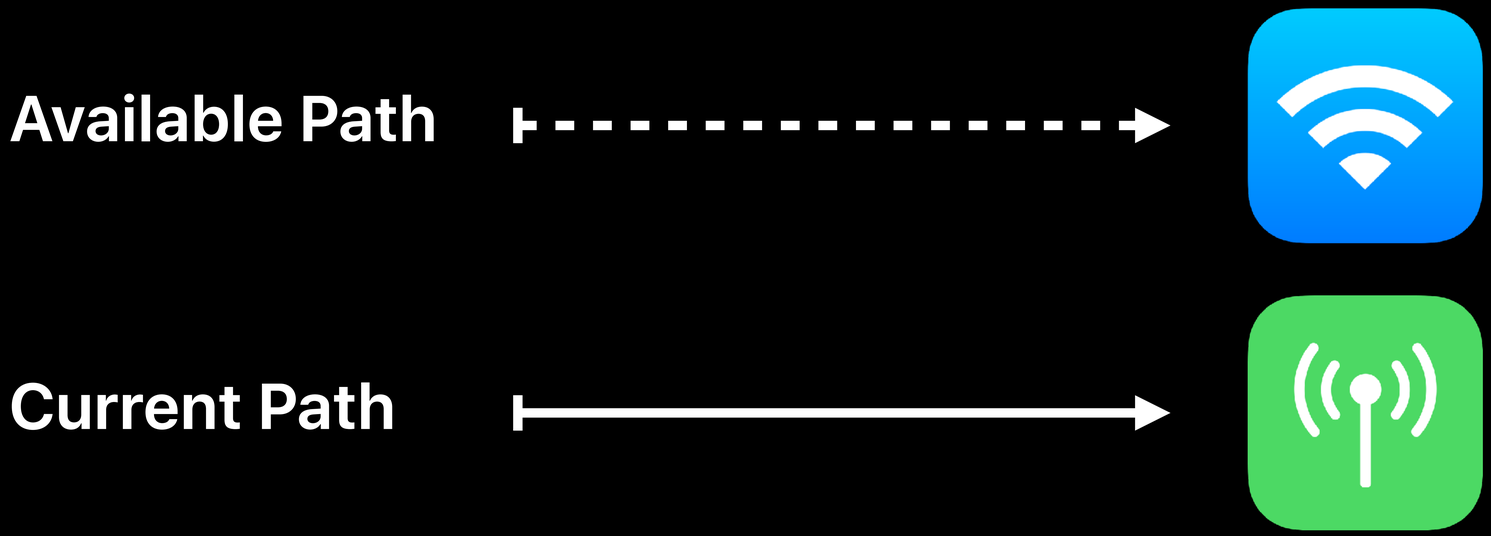



```
isViable = true
```

```
betterPathAvailable = false
```

Reacting to Network Transitions

Better path



- `isViable = true`
- `betterPathAvailable = true`

Attempt to migrate to new connection

Continue to use original connection until a new connection is ready

```
// Handle connection viability
connection.viabilityUpdateHandler = { (isViable) in
    if (!isViable) {
        // Handle connection temporarily losing connectivity
    } else {
        // Handle connection return to connectivity
    }
}

// Handle better paths
connection.betterPathUpdateHandler = { (betterPathAvailable) in
    if (betterPathAvailable) {
        // Start a new connection if migration is possible
    } else {
        // Stop any attempts to migrate
    }
}
```

Multipath Connections

Achieving ideal mobility

Enable Multipath TCP with `NWParameters.multipathServiceType`

Also available in URLSession

Restricting interface types in `NWParameters` limits paths for Multipath TCP

Connection viability for Multipath TCP indicates the presence of active subflows

Watching Interface Changes

Monitor network state, not host reachability

Use `NWPathMonitor` to iterate the current available network interfaces

Updates notify network changes

Useful for updating UI or opening connections per-interface

Along with connection state `.waiting`, replaces `SCNetworkReachability`

Getting Involved

Josh Graessley, Networking

Discouraged Practices

Network Kernel Extensions

FTP and File URLs for Proxy Automatic Configuration (PAC)

Discouraged APIs

CoreFoundation

```
CFStreamCreatePairWithPeerSocketSignature
CFStreamCreatePairWithSocketToHost
CFStreamCreatePairWithSocket
CFStreamCreatePairWithSocketToCFHost
CFStreamCreatePairWithSocketToNetService
CFSocket
```

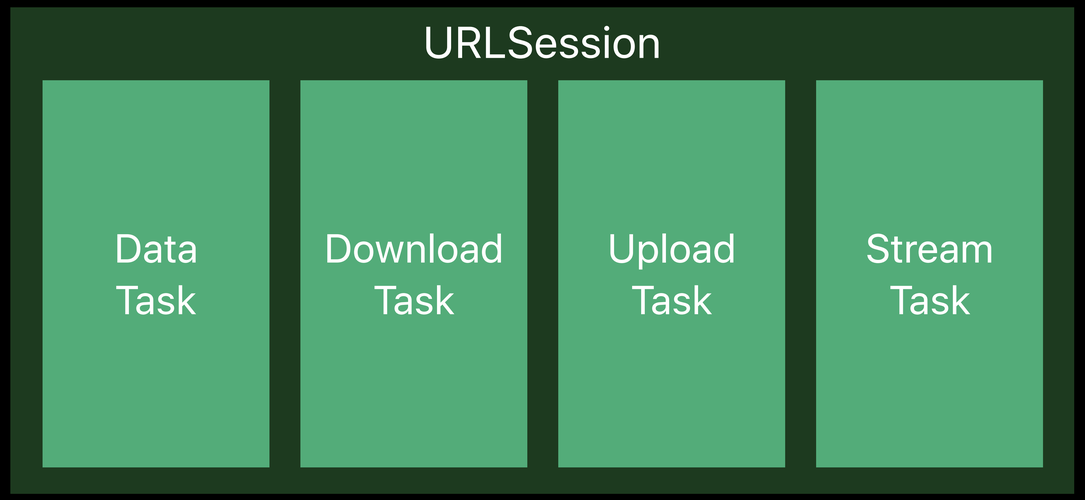
Discouraged APIs

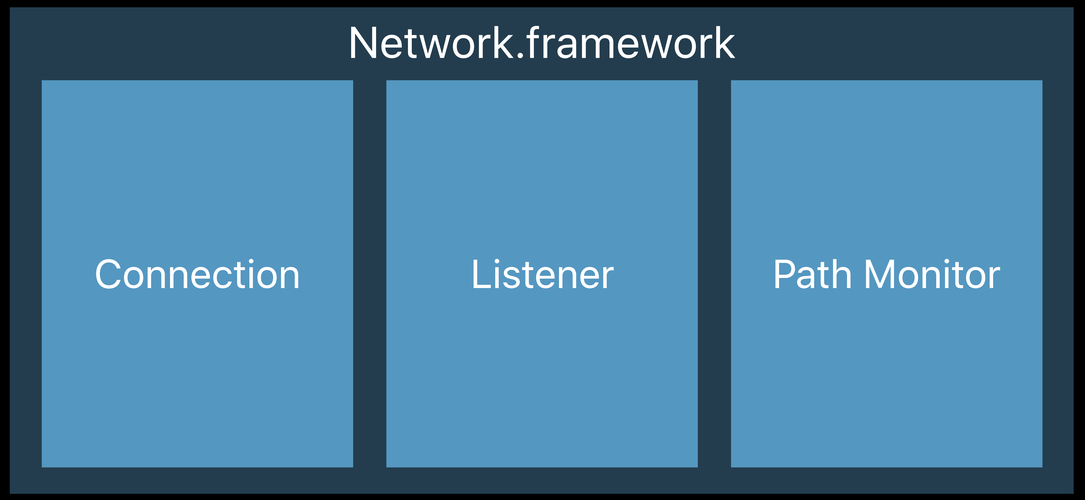
Foundation and SCNetworkReachability

```
+[NSStream getStreamsToHostWithName:port:inputStream:outputStream:]
+[NSStream getStreamsToHost:port:inputStream:outputStream:]
-[NSNetService getInputStream:outputStream:]
NSNetServiceListenForConnections
NSSocketPort
```

```
SCNetworkReachability
```

Preferred APIs





Next Steps

Adopt Network.framework

Optimize sending and receiving

Handle network mobility gracefully

Contact Developer Support with questions and enhancement requests

More Information

https://developer.apple.com/wwdc18/715

Networking Lab	Technology Lab 1	Thursday 2:00PM
Networking Lab	Technology Lab 2	Friday 9:00AM


WWDC18