Implementation Example: Network.framework on macOS and iOS

Tommy Pauly
TAPS
IETF 102, July 2018, Montreal

Network.framework

Public framework available on iOS 12 and macOS Mojave betas

Preconnection objects:

`NWEndpoint`
`NWParameters (ProtocolStack, NWProtocol.Options)`

Active objects:

`NWConnection`
`NWListener`
`NWPathMonitor`

Introspection objects:

`NWPath`
`NWProtocol.Metadata`

Connection Establishment

```
let dest = NWEndpoint.hostPort(host: "www.example.com", port:.https)
let connection = NWConnection(to: dest, using:.tls)
connection.stateUpdateHandler = { [weak self] (newState) in
   switch (newState) {
   case .ready:
      // Handle connection established
   case .waiting(let error):
      // Handle network availability error
   case .failed(let error):
      // Handle error
   default:
      break
   }
}
connection.start(queue: .main)
```

Sending

Basic sends

```
let data = "Hello".data(using: .utf8)
connection.send(content: data, completion: .contentProcessed({ (error) in
   if let error = error {
      print("Send Error \(error)")
   }
}))
```

Batch sends

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

Receiving

Receive Message

```
connection.receive { (content, context, isComplete, error) in
   // Handle inbound message
}
```

Receive Partial Message

```
connection.receive(minimumIncompleteLength: 1, maximumLength: 1024)
{ (content, context, isComplete, error) in
   // Handle inbound data
}
```

Message Contexts

Custom send contexts

```
let sendContext = NWConnection.ContentContext(identifier: "hello",
                                              expiration: 1000,
                                              isFinal: false,
                                              antecedent: nil,
                                              metadata: [ datagramSettings ])
```

Convenience send contexts

```
let sendContext = NWConnection.ContentContext.defaultMessage

let sendContext = NWConnection.ContentContext.finalMessage

let sendContext = NWConnection.ContentContext.defaultStream
```

Candidate Gathering

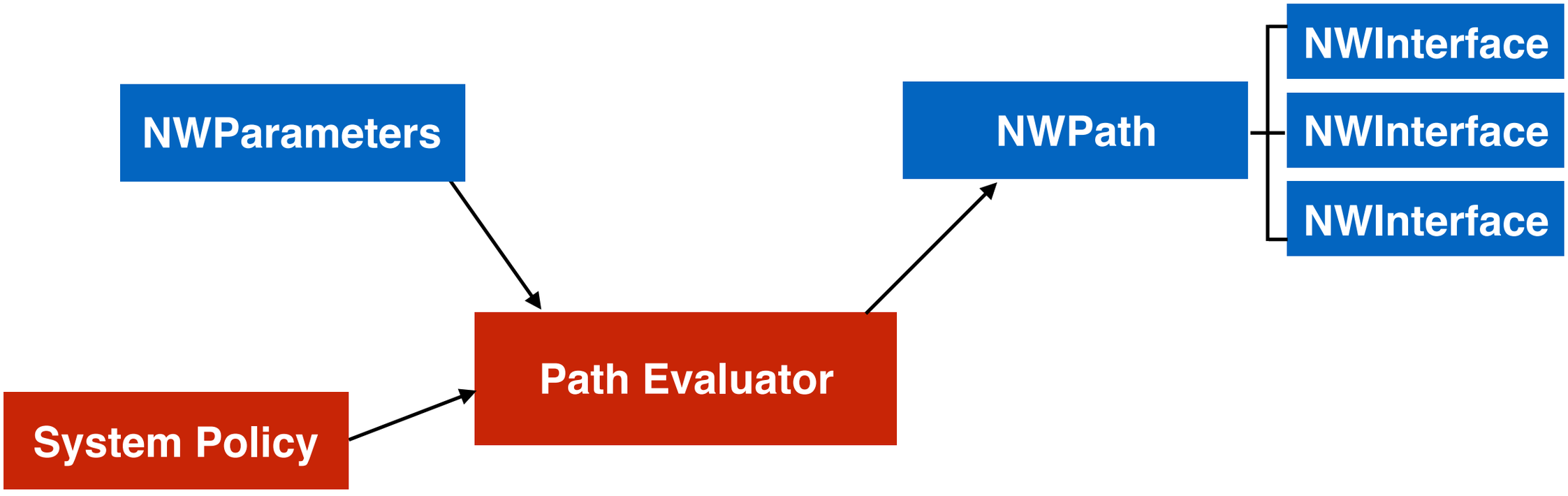
Path Selection

Applications can constrain the paths:

```
parameters.requiredInterfaceType = .wifi

parameters.prohibitedInterfaceTypes = [ .cellular ]
```

The system uses constraints to determine set of available paths



Candidate Gathering

System Policy

System configuration determines a ranked list of interfaces or paths for default application use

Extended policies are determined by per-application settings

- Interfaces or interface types may be prohibited or required by the system
- Certain application parameters can be matched to also influence connection path

Path includes protocol settings, such as ECN and TFO support and predicted RTT

Candidate Gathering

Protocol Selection

Currently determined fairly statically in a “default stack":

```
let transportOptions = NWProtocolTCP.Options()
transportOptions.enableKeepalive = true
parameters.defaultProtocolStack.transportProtocol = transportOptions
```

Adding an array of protocol stacks (composed of options) would allow explicit protocol racing

Conveniences create common stacks for TCP, UDP, TLS/TCP, DTLS/UDP

New conveniences could cover combinations of equivalent protocols (such as QUIC | HTTP/2 | TLS)

Connection Lifetime

State Diagram