Network Extensions

Based on the Network Extension Framework by Apple

Network Extension Framework

Network Filtering - “Content Filter API’s”

Filter at the system level

NEFilterManager: Used to create a content filter config that registers content filter w/system so system knows how to run your content filter.

You also create a system extenstion and this is where your code that actually filters network content will run.

Content can be filtered at two different layers: **flow layer** or **packet layer**.

Flow layer filtering:

NEFilterDataProvider class**;**

- *Gives **read only** access to flows*
- *By default system will pass any flow of tcp and udp data NEFilterDataProvider subclass*

NEFilterSettings;

- Can be used to tell the system which flows to pass to your filter.

Create a subclass of **NEFilterDataProvider**

Once content filter config is registered w/system & running;

New TCP & UDP flow of network data created on the system

they are passed to your **NEFilterDataProvider** subclass

they are represented as individual **NEFilterFlow** objects.

Your subclass allows or drops each individual flow.

This decision can be made during any lifecycle stage of the flow;

up front or after having seen some of flows data.

Packet Layer Filtering;

NEFilterPacketProvider;

Create subclass of NEFilterPacketProvider in your system extension.

System will pass packet objects to your NEFilterPacketProvider subclass

Subclass makes per packet allow/drop decisions.

Filter at the packet level

NEFilterPacketProvider

Transparent Proxy

Diverts traffic destined for a specific website to a cloud service.

Can apply encryption, cacheing, multiplex (multiple flows of network traffic over a single connection)

NETransparentProxyManager

Used to create transparent proxy app configs and register them with the system

Allows you to proxy flows of network data at the flow layer;

NEAppProxyProvider;

- create subclass to proxy flows of network data at flow layer
- By default system does not divert flows to your proxy
- Handles flow diversion

NENetworkRule;

- Specify what flow you want to proxy

DNS Proxy

API's that are part of the **NetworkExtension** Framework

- Applies additional security to the DNS protocol
  - App may
    - Apply some encryption to DNS traffic
    - Proxy DNS traffic over some sort of secure channel

NEDNSProxyManager;

- Used in main app to
  - Create DNS proxy configuration.
  - Register config w/system so system knows how to run your DNS proxy

NEDNSProxyProvider;

- Implement app proxy as subclass of this class.
- Once registered w/the system and your system extension is running
  - **System will start diverting all dns queries** to your **NEDNSProxyProvider** subclass.
  - After this is is up to you how you would like to handle each DNS query
    - **Encrypt it**
    - **Send over some sort of secure channel**
    - **etc.**

Implement as subclass of **NEDNSProxyProvider** class

VPN

Part of the **NetworkExtension** Famework

NETunnelProviderManager;

- Use to create VPN configs and register VPN client w/the system.
- Creates vpn extension and registers with system

NEPacketTunnelProvider;

- Implement vpn client as subclass of this class
- System creates utun if corresponding to your NEPacketTunnelProvider
- Responsible for telling the system about which networks you want to be routed through your vpn.

Once you've specified your routing rules for your vpn and those are installed in the system

- As ip packets are routed to your utun if per those routes
    - Those packets are diverted to your NEPacketTunnelProvider
        - You can send those packets thru your utun connection using your custom tunneling protocol

VPN API Enhancements (Catalina)

IncludeAllNetworks: Keeps all traffic in VPN. Drops traffic if network disconnects from VPN.

excludeLocalNetworks: Allows local traffic to still occur

Per-App VPN;

MailDomains, CalendarDomains, ContactsDomains

System Domains you can use to route traffic to your per app vpn.

If Host connects to domain > domain matches list > traffic goes through vpn

Specify work email domain in mail domain array to route work email through vpn when mail opens new connection to corporate email server that connection is routed through the per app vpn.

Virtual Machine app extensions

vmnet.framework

Allows you to connect a virtual machine to the network

Bridged Mode: Machines show up on the local network as if they were physically connected to the local network.

Apps that use Custom Protocols (Custom low-layer protocols)

comms w/ext if's using low level or highly optimized protocols

Custom IP Protocol: api

IN app create new kind of NWParameters specifying identifier number for your custom IP protocol;

Use that NWParameters object to creaet an NWConnection

Use that NWConnection just as you would a tcp or udp connection to communicate over the network using your custom protocol

```
let parameters = NWParameters(customIPProtocolNumber: 123)

let destination = NWEndpoint.hostPort(host: "fe80::70", port: 0)
let connection = NWConnection(to: destination, using: parameters)

connection.start(queue: .main)
connection.send(content: packet, completion: .contentProcessed({ error in
    // Check for errors
}))
```

Cutom Link Layer Protocol

NWEthernetChannel: Object created in app specifying custom ether type to use. Use to com over eth if.

(Must use custom ether type)

Set up **NWEthernetChannel** object;

```
import Foundation
import Network

let path = NWPathMonitor(requiredInterfaceType: .wiredEthernet).currentPath
guard let interface = path.availableInterfaces.first else {
    // Not connected to ethernet
    fatalError("Not connected to ethernet")
}

let channel = NWEthernetChannel(on: interface, etherType: 0xB26E)
```

Set up Callback blocks;

```
channel.stateUpdateHandler = { state in
    switch state {
        case .ready:
            let remoteAddress = NWEthernetChannel.EthernetAddress("CB:A9:87:65:43:21")
            let packet = Data("Hello".utf8)
            channel.send(content: packet, to: remoteAddress!, vlanTag: 0) { error in
                // Handle error
            }
        default:
            break
    }
}
channel.receiveHandler = { packetData, vlanTag, localAddress, remoteAddress in
    // Handle packet
}

channel.start(queue: .main)
```

Network Kernel Extensions were deprecated as of MacOS Catelina

Allegedly.