Network Extension Provider Memory Strategy

This thread has been locked. Questions are automatically locked after two months of inactivity, or sooner if deemed necessary by a moderator.

1.2k

A couple of folks have asked me about how to manage memory within a Network Extension provider. These discussions usually start with the observation that the Network Extension provider architecture doesn't support a memory warning callback. That's true, and that absence says a lot about the differences between the memory lifecycle of a Network Extension provider and a typical iOS app.

In an iOS app you typically want to use as much memory as possible in an effort to boost your performance. That 'greedy' approach means that you have to respond to memory warnings so that you can release memory when the system is under memory pressure.

A Network Extension provider is different. Rather than trying to use as much memory as possible, you should try to put a strict upper bound on your memory use. You can allocate a reasonable amount of buffer space for network data passing through your provider but you can't let your buffer space grow without bound. If you've buffered too much data, you should exert flow control. How you do this depends on the context:

- outbound — If you have too much outbound data buffered, you should stop reading it from the 'flow' object. For example, if a packet tunnel provider has too much outbound data buffered, it should stop calling

  ```
  NEPacketTunnelFlow.readPacketsWithCompletionHandler(_:)
  ```

  until it has sent some data down the tunnel and thus freed up some buffer space.
- inbound — When you receive data from the tunnel, you shouldn't buffer too much in your provider. The correct behaviour here depends on the type of provider you're implementing:
  - A packet tunnel provider works at the IP layer, and hence is allowed to drop packets. When you read data from the tunnel and pass it 'up' to the kernel, you call

    ```
    NEPacketTunnelFlow.writePackets(_:withProtocols:)
    ```

    . That routine doesn't support flow control (note that there's no completion handler). If the kernel doesn't have space to process the packets, it drops them. This is OK, because TCP/IP allows IP-layer packets to be dropped.
  - In contrast, an app proxy provider is not allowed to just drop packets. In this case the routines you call to pass data 'up' to the kernel (

    ```
    NEAppProxyTCPFlow.writeData(_:withCompletionHandler:)
    ```

    and

    ```
    NEAppProxyUDPFlow.writeDatagrams(_:sentByEndpoints:completionHandler:)
    ```

) have a completion handler. This completion handler is called when the data you've written is actually passed to the kernel. The flow object buffers the data internally until that happens.
    You should treat this case much like the outbound case. That is, keep track of the amount of data buffered by the flow object and, once it passes some reasonable limit, assert flow control on the tunnel itself (that is, stop reading data from the tunnel).

All-in-all these approaches allow you to put a strict bound on the amount of data your provider uses, which is important given the position of your provider within the iOS architecture as a whole.

Share and Enjoy — Quinn "The Eskimo!" Apple Developer Relations, Developer Technical Support, Core OS/Hardware

```
let myEmail = "eskimo" + "1" + "@apple.com"
```

(r. 25,680,245)

Network

Asked 4 years ago by eskimo

Reply to this question