Módulo 4
Desarrollo de escenarios de simulación

Gabriel Guerrero-Contreras
gabriel.guerero@uca.es
Sara Balderas-Díaz
sara.balderas@uca.es

Universidad de Cádiz
Escuela Superior de Ingeniería
Departamento de Ingeniería Informática

Índice

4. Desarrollo de escenarios de simulación

Este módulo está diseñado para introducir el proceso de creación, configuración y ejecución de escenarios de simulación utilizando ns-3. A través de ejemplos prácticos y explicaciones detalladas, se explorará cómo simular topologías de red simples, generar tráfico de red y modelar eventos dinámicos, tales como la desconexión de un enlace.

4.1. Hola, mundo

En esta sección veremos un ejemplo básico de script de ns-3. Este nos servirá para familiarizarnos con el proceso de compilación y ejecución de escenarios en ns.3.

Código 1: hello.cc

```
#include "ns3/core-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("HelloSimulator");

int main ()
{
  NS_LOG_UNCOND ("Hola, mundo");
  return 0;
}
```

Este código es el más básico posible, simplemente imprime "Hola, mundo" en la consola. Para compilar y ejecutar el script, seguimos los siguientes pasos:

1. Creamos un nuevo archivo nuestro directorio, llamado `hello.cc`.
2. Insertamos el Código 1.
3. Ejecutamos el contenedor de Docker, desde la terminal, una vez ubicados en el directorio en el que se encuentra nuestro script `hello.cc`, mediante la siguiente instrucción:

```
docker run -it -v .:/ns3/ns-allinone-3.35/ns-3.35/
  ↪ scratch ns3-container
```

4. Una vez dentro del contenedor, nos dirigimos al directorio de ns-3 y compilamos el script mediante la instrucción:

```
cd ns-allinone-3.35/
cd ns-3.35/
./waf
```

5. Una vez compilado el script, lo ejecutamos utilizando el siguiente comando:

```
./waf --run hello
```

Si todo está configurado correctamente, verás la salida "Hola, mundo.[en] su terminal.

4.2. Creación de topologías de red en ns-3

La creación de topologías de red en ns-3 es un paso fundamental en el desarrollo de escenarios de simulación, ya que define la estructura y las interconexiones entre los nodos de la red. ns-3 proporciona una variedad de herramientas y APIs que permiten a los usuarios crear topologías de red complejas y realistas para simular una amplia gama de entornos de red.

4.2.1. Topología punto a punto

En esta sección vamos a ver cómo crear una topología básica punto a punto de tres nodos. La topología constará de tres nodos (A, B, y C), donde A se conecta a B, y B se conecta a C, formando una línea.

El primer paso es incluir las cabeceras necesarias para nuestro script. Estas proporcionan las clases y funciones necesarias para configurar los nodos, dispositivos, canales y aplicaciones.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
```

Dentro de la función 'main', inicializamos los componentes y configuramos nuestra topología punto a punto.

Se crea un contenedor para nodos y se inicializan tres nodos dentro de él. Estos nodos serán usados para configurar la topología.

```
ns3::NodeContainer nodes;
nodes.Create (3);
```

Se crea un helper para configurar los dispositivos punto a punto, especificando atributos como el `DataRate` (tasa de datos) y el `Delay` (retraso) para los enlaces.

```
ns3::PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", ns3::StringValue ("
    ↪ 5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", ns3::StringValue ("2
    ↪ ms"));
```

Se crean contenedores para los dispositivos de red y se instalan enlaces punto a punto entre los nodos (A-B y B-C).

```
ns3::NetDeviceContainer deviceAB, deviceBC;
deviceAB = pointToPoint.Install (nodes.Get(0), nodes.Get(1));
deviceBC = pointToPoint.Install (nodes.Get(1), nodes.Get(2));
```

Se instala el stack de Internet en todos los nodos para permitirles comunicarse usando protocolos de Internet.

```
ns3::InternetStackHelper stack;
stack.Install (nodes);
```

Se asignan direcciones IP a los interfaces de los dispositivos en cada enlace punto a punto, creando dos subredes diferentes.

```
ns3::Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
ns3::Ipv4InterfaceContainer interfacesAB = address.Assign (
   ↪ deviceAB);
address.SetBase ("10.1.2.0", "255.255.255.0");
ns3::Ipv4InterfaceContainer interfacesBC = address.Assign (
   ↪ deviceBC);
```

Se instala una aplicación de servidor UDP en el nodo C (el último nodo), configurada para escuchar en el puerto 9.

```
ns3::UdpEchoServerHelper echoServer (9);
ns3::ApplicationContainer serverApps = echoServer.Install (nodes
   ↪ .Get(2));
serverApps.Start (ns3::Seconds (1.0));
serverApps.Stop (ns3::Seconds (10.0));
```

Se configura una aplicación de cliente UDP en el nodo A (el primer nodo) para enviar paquetes al servidor en el nodo C.

```
ns3::UdpEchoClientHelper echoClient (interfacesBC.GetAddress (1)
   ↪ , 9);
echoClient.SetAttribute ("MaxPackets", ns3::UintegerValue (1));
echoClient.SetAttribute ("Interval", ns3::TimeValue (ns3::
   ↪ Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", ns3::UintegerValue (1024)
   ↪ );
```

La aplicación cliente se instala en el nodo A y se configura para iniciar y detenerse en momentos específicos.

```
ns3::ApplicationContainer clientApps = echoClient.Install (nodes
   ↪ .Get(0));
```

```
clientApps.Start (ns3::Seconds (2.0));
clientApps.Stop (ns3::Seconds (10.0));
```

Finalmente, estas líneas lanzan la simulación y luego la destruyen una vez que ha completado su ejecución.

```
ns3::Simulator::Run ();
ns3::Simulator::Destroy ();
```

4.2.2. Topología en árbol

En esta sección, vamos a configurar una topología en árbol en ns-3. Los nodos estarán organizados en una estructura jerárquica, donde cada nodo, excepto el raíz, es hijo de exactamente un nodo padre, lo que es ideal para simular redes de sensores y otras aplicaciones de redes jerárquicas.

Inicializamos el contenedor de nodos. Este código crea un nodo raíz, dos nodos en el primer nivel, y dos nodos en el segundo nivel.

```
ns3::NodeContainer rootNode;
rootNode.Create (1);

ns3::NodeContainer levelOneNodes;
levelOneNodes.Create (2);

ns3::NodeContainer levelTwoNodes;
levelTwoNodes.Create (2);
```

La estructura en árbol se formará conectando estos nodos con enlaces punto a punto. `PointToPointHelper` configura los enlaces punto a punto con una tasa de datos de 5 Mbps y un retraso de 2 ms.

```
ns3::PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", ns3::StringValue ("
    ↪ 5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", ns3::StringValue ("2
    ↪ ms"));
```

Luego, conectamos los nodos para formar la topología en árbol. Conectamos el nodo raíz con los nodos del primer nivel y luego conectamos cada uno de estos nodos del primer nivel con un nodo en el segundo nivel.

```
ns3::NetDeviceContainer devicesRootToLevelOne = pointToPoint.
    ↪ Install (rootNode.Get(0), levelOneNodes.Get(0));
ns3::NetDeviceContainer devicesRootToLevelOneSecond =
    ↪ pointToPoint.Install (rootNode.Get(0), levelOneNodes.Get
    ↪ (1));
```

```
ns3::NetDeviceContainer devicesLevelOneToLevelTwo = pointToPoint
    ↪ .Install (levelOneNodes.Get(0), levelTwoNodes.Get(0));
ns3::NetDeviceContainer devicesLevelOneToLevelTwoSecond =
    ↪ pointToPoint.Install (levelOneNodes.Get(1), levelTwoNodes.
    ↪ Get(1));
```

Para permitir la comunicación utilizando protocolos de Internet, instalamos la pila de Internet en todos los nodos.

```
ns3::InternetStackHelper stack;
stack.Install (rootNode);
stack.Install (levelOneNodes);
stack.Install (levelTwoNodes);
```

Asignamos direcciones IP a los nodos para habilitar la comunicación de red entre ellos. Cada llamada a `SetBase` y `Assign` configura una subred distinta para un conjunto de enlaces entre los nodos.

```
ns3::Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
address.Assign (devicesRootToLevelOne);
address.Assign (devicesRootToLevelOneSecond);
```

`SetBase` configura la dirección base de la subred y la máscara de subred que se utilizará para la asignación de direcciones IP. En este caso, se establece la dirección base de la subred como `10.1.1.0` con una máscara de subred `255.255.255.0`. Esto significa que las direcciones IP que se pueden asignar en esta subred estarán en el rango de `10.1.1.1` a `10.1.1.254`.

`Assign` se utiliza para asignar direcciones IP a los dispositivos en los enlaces definidos por los `NetDeviceContainer` (en este caso, `devicesRootToLevelOne` y `devicesRootToLevelOneSecond`). Estos contenedores representan los dispositivos de red que conectan el nodo raíz con los nodos del primer nivel de la topología en árbol. Cada dispositivo en el contenedor recibe una dirección IP única de la subred configurada previamente.

Después de asignar direcciones IP a los primeros dispositivos, se procede a configurar una nueva subred para los siguientes enlaces.

```
address.SetBase ("10.1.2.0", "255.255.255.0");
address.Assign (devicesLevelOneToLevelTwo);
address.Assign (devicesLevelOneToLevelTwoSecond);
```

Se cambia la base de la subred a `10.1.2.0` con la misma máscara de subred, lo que permite crear una subred distinta para otro conjunto de enlaces. Esta nueva subred tendrá direcciones IP en el rango de `10.1.2.1` a `10.1.2.254`. Finalmente, las llamadas a `Assign` asignan direcciones IP de la nueva subred a los dispositivos en los enlaces entre el primer nivel de nodos y el segundo nivel, representados por `devicesLevelOne-`

ToLevelTwo y devicesLevelOneToLevelTwoSecond.

4.3. Generación de tráfico y eventos en ns-3

Esta sección explora cómo generar tráfico y eventos en ns-3, junto con ejemplos prácticos para ilustrar su aplicación.

4.3.1. Tráfico constante

Configuraremos una topología en árbol en ns-3 y generaremos tráfico constante entre los nodos. Esta configuración es ideal para simular entornos donde los nodos, como sensores, envían datos continuamente a un nodo central o servidor.

Para generar tráfico constante, utilizaremos las aplicaciones `OnOffApplication` para el envío de datos y `PacketSink` para la recepción.

Primero, definimos la topología en árbol como se explicó anteriormente. Luego, procedemos a instalar las aplicaciones.

```
ns3::OnOffHelper onOff("ns3::UdpSocketFactory",
                       Address());
onOff.SetAttribute("OnTime",
                   StringValue("ns3::ConstantRandomVariable[
                      ↪ Constant=1]"));
onOff.SetAttribute("OffTime",
                   StringValue("ns3::ConstantRandomVariable[
                      ↪ Constant=0]"));
onOff.SetAttribute("DataRate", DataRateValue(DataRate("500kbps")
   ↪ ));
onOff.SetAttribute("PacketSize", UintegerValue(1024));

ns3::PacketSinkHelper packetSink("ns3::UdpSocketFactory",
                                 InetSocketAddress(Ipv4Address::
                                    ↪ GetAny(), 9));
```

El `OnOffHelper` configura la aplicación para generar tráfico usando UDP. `OnTime` y `OffTime` configuran la aplicación para estar activa dentro de un periodo de tiempo concreto, generando un flujo constante de tráfico. La tasa de datos y el tamaño del paquete también se especifican en estas instrucciones.

A continuación, asignamos la aplicación de generación de tráfico a un nodo y la aplicación de recepción al nodo destino.

```
ApplicationContainer sinkApps = packetSink.Install(rootNode.Get
   ↪ (0));
sinkApps.Start(Seconds(0.0));
sinkApps.Stop(Seconds(10.0));
```

```
ApplicationContainer onOffApps = onOff.Install(levelTwoNodes.Get
    ↪ (1));
onOffApps.Start(Seconds(1.0));
onOffApps.Stop(Seconds(10.0));
```

Este código instala un `PacketSink` en el nodo raíz para actuar como receptor de los datos, y una `OnOffApplication` en uno de los nodos de nivel dos para generar y enviar los datos. El tráfico se genera constantemente desde el segundo después del inicio de la simulación hasta el final en el segundo 10.

Finalmente, se inicia la simulación y se configura el seguimiento del tráfico generado.

```
ns3::Ipv4GlobalRoutingHelper::PopulateRoutingTables();

Simulator::Run();
Simulator::Destroy();
```

4.3.2. Tráfico VoIP

Vamos a configurar una topología en árbol en ns-3 y simularemos tráfico VoIP a través de esta red. Este escenario es ideal para simular aplicaciones de comunicación en tiempo real en redes jerárquicas como redes de telefonía IP corporativas o sistemas de distribución de contenido multimedia.

Inicializamos el contenedor de nodos para formar la topología en árbol.

```
ns3::NodeContainer rootNode;
rootNode.Create (1);

ns3::NodeContainer levelOneNodes;
levelOneNodes.Create (2);

ns3::NodeContainer levelTwoNodes;
levelTwoNodes.Create (2);
```

Configuramos los enlaces punto a punto entre los nodos con características adecuadas para el tráfico VoIP, como una baja latencia.

```
ns3::PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", ns3::StringValue ("
    ↪ 10Mbps"));
pointToPoint.SetChannelAttribute ("Delay", ns3::StringValue ("1
    ↪ ms"));
```

`PointToPointHelper` configura los enlaces punto a punto con una tasa de datos de 10 Mbps y un retraso de 1 ms para soportar el tráfico VoIP.

Conectamos los nodos para formar la topología en árbol.

```
ns3::NetDeviceContainer devicesRootToLevelOne = pointToPoint.
   ↪ Install (rootNode.Get(0), levelOneNodes.Get(0));
ns3::NetDeviceContainer devicesRootToLevelOneSecond =
   ↪ pointToPoint.Install (rootNode.Get(0), levelOneNodes.Get
   ↪ (1));

ns3::NetDeviceContainer devicesLevelOneToLevelTwo = pointToPoint
   ↪ .Install (levelOneNodes.Get(0), levelTwoNodes.Get(0));
ns3::NetDeviceContainer devicesLevelOneToLevelTwoSecond =
   ↪ pointToPoint.Install (levelOneNodes.Get(1), levelTwoNodes.
   ↪ Get(1));
```

Instalamos la pila de Internet en todos los nodos.

```
ns3::InternetStackHelper stack;
stack.Install (rootNode);
stack.Install (levelOneNodes);
stack.Install (levelTwoNodes);
```

Asignamos direcciones IP a los nodos.

```
ns3::Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
address.Assign (devicesRootToLevelOne);
address.SetBase ("10.1.2.0", "255.255.255.0");
address.Assign (devicesRootToLevelOneSecond);
address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (devicesLevelOneToLevelTwo);
address.SetBase ("10.1.4.0", "255.255.255.0");
address.Assign (devicesLevelOneToLevelTwoSecond);
```

Para simular el tráfico VoIP, utilizamos el helper de aplicaciones `OnOff` y `PacketSink`. Configuramos la aplicación `OnOff` para generar tráfico similar al de una llamada VoIP, usando un códec típico de VoIP como `G.711`[1], que tiene un bitrate de aproximadamente 64 kbps.

```
ns3::OnOffHelper onoff ("ns3::UdpSocketFactory", Address ());
onoff.SetAttribute ("OnTime",  StringValue ("ns3::
   ↪ ConstantRandomVariable[Constant=1]"));
onoff.SetAttribute ("OffTime", StringValue ("ns3::
   ↪ ConstantRandomVariable[Constant=0]"));
// 160 bytes = 1280 bits, para 20 ms de audio a 64 kbps
onoff.SetAttribute ("PacketSize", UintegerValue (160));
onoff.SetAttribute ("DataRate", DataRateValue (DataRate ("64kbps
   ↪ ")));

ns3::ApplicationContainer app = onoff.Install (levelTwoNodes.Get
```

[1] https://www.itu.int/rec/T-REC-G.711

```
    ↪ (0));
app.Start (Seconds (1.0));
app.Stop (Seconds (10.0));

ns3::PacketSinkHelper sink ("ns3::UdpSocketFactory", Address ())
    ↪ ;
sink.Install (rootNode.Get(0));
```

4.3.3. Programación de eventos

La programación de eventos en ns-3 permite simular situaciones dinámicas en la red, como la conexión y desconexión de nodos, la pérdida de enlaces o la variación de parámetros de red en el tiempo. Veamos cómo programar eventos en ns-3 con un ejemplo.

Configuraremos un escenario básico en ns-3 donde dos nodos conectados por un enlace punto a punto experimentarán una desconexión programada después de un cierto tiempo. Este escenario es útil para simular la dinámica de la red y los efectos de la pérdida de conectividad.

Configuramos el enlace punto a punto entre los dos nodos.

```
ns3::NodeContainer nodes;
nodes.Create (2);

ns3::PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", ns3::StringValue ("
    ↪ 5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", ns3::StringValue ("2
    ↪ ms"));

ns3::NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);
```

Instalamos la pila de Internet y asignamos direcciones IP a los dispositivos.

```
ns3::InternetStackHelper stack;
stack.Install (nodes);

ns3::Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
address.Assign (devices);
```

Para simular la desconexión del enlace, programamos un evento que desactivará uno de los dispositivos de red después de 5 segundos.

```
ns3::Ptr<ns3::NetDevice> device = devices.Get (1);
ns3::Simulator::Schedule (ns3::Seconds (5.0), &TurnOffDevice,
```

```
    ↪ device);
```

`Simulator::Schedule` programa un evento que ocurrirá en 10 segundos. La función `DeviceDown` es una función auxiliar que desactivará el dispositivo, desconectando el enlace. Necesitamos definir esta función para cambiar el estado operativo del dispositivo.

```
void TurnOffDevice(ns3::Ptr<ns3::NetDevice> device)
{
    device->SetReceiveCallback(ns3::MakeNullCallback<bool, ns3::
       ↪ Ptr<ns3::NetDevice>, ns3::Ptr<const ns3::Packet>,
       ↪ uint16_t, const ns3::Address&>());
}
```

Finalmente, iniciamos la simulación.

```
ns3::Simulator::Run ();
ns3::Simulator::Destroy ();
```

4.4. Toma de medidas en ns-3

En esta sección, exploraremos cómo tomar medidas de rendimiento, específicamente latencia y throughput, en una topología punto a punto utilizando ns-3. La topología consistirá en dos nodos (A y B) conectados directamente. Implementaremos un escenario donde el nodo A envía paquetes al nodo B, y mediremos la latencia y el throughput de la transmisión.

Primero, incluimos las cabeceras necesarias que nos permitirán configurar los nodos, dispositivos, canales, y las herramientas para tomar medidas.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-module.h"
```

Dentro de la función main, inicializamos los componentes y configuramos nuestra topología punto a punto y el monitor de flujo para la recolección de medidas.

Creamos un contenedor para nodos y los inicializamos.

```
ns3::NodeContainer nodes;
nodes.Create (2);

ns3::PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", ns3::StringValue ("
   ↪ 10Mbps"));
pointToPoint.SetChannelAttribute ("Delay", ns3::StringValue ("1
```

```
    ↪ ms"));
ns3::NetDeviceContainer devices = pointToPoint.Install (nodes);

ns3::InternetStackHelper stack;
stack.Install (nodes);
ns3::Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
ns3::Ipv4InterfaceContainer interfaces = address.Assign (devices
    ↪ );
```

Configuramos el monitor de flujo para recoger estadísticas de la red.

```
ns3::FlowMonitorHelper flowmon;
ns3::Ptrns3::FlowMonitor monitor = flowmon.InstallAll();
```

Creamos y configuramos una aplicación de cliente y servidor UDP para simular el tráfico.

```
uint16_t port = 9;
ns3::UdpEchoServerHelper server(port);
ns3::ApplicationContainer serverApp = server.Install(nodes.Get
    ↪ (1));
serverApp.Start(ns3::Seconds(1.0));
serverApp.Stop(ns3::Seconds(10.0));

ns3::UdpEchoClientHelper client(interfaces.GetAddress(1), port);
client.SetAttribute("MaxPackets", ns3::UintegerValue(100));
client.SetAttribute("Interval", ns3::TimeValue(ns3::Seconds(0.1)
    ↪ ));
client.SetAttribute("PacketSize", ns3::UintegerValue(512));
ns3::ApplicationContainer clientApp = client.Install(nodes.Get
    ↪ (0));
clientApp.Start(ns3::Seconds(2.0));
clientApp.Stop(ns3::Seconds(10.0));
```

Se indica que se debe iniciar la simulación.

```
ns3::Simulator::Run ();
```

Se verifica la pérdida de paquetes utilizando el monitor de flujo. Esto permite contabilizar cualquier paquete que no haya sido recibido, lo cual es crucial para calcular la latencia media y el throughput correctamente.

```
monitor->CheckForLostPackets ();
```

Se obtiene un clasificador de flujo de la instancia de `FlowMonitor`. Este clasificador se usa para identificar los flujos de tráfico en la simulación, permitiendo la asociación de estadísticas de rendimiento con flujos específicos.

```
ns3::Ptrns3::Ipv4FlowClassifier classifier = DynamicCastns3::
   ↪ Ipv4FlowClassifier(flowmon.GetClassifier ());
```

Se recopilan las estadísticas de todos los flujos monitoreados. Estas estadísticas incluyen, entre otros, bytes transmitidos (`txBytes`), bytes recibidos (`rxBytes`), y sumas de latencia (`delaySum`).

```
std::map<ns3::FlowId, ns3::FlowMonitor::FlowStats> stats =
   ↪ monitor->GetFlowStats ();
```

Este bucle itera a través de cada flujo de tráfico, utilizando el clasificador para obtener la tupla de cinco elementos (direcciones IP de origen y destino, puertos, protocolo) que identifica el flujo. Luego, imprime las estadísticas de cada flujo: bytes transmitidos y recibidos, throughput (calculado como los bytes recibidos por segundo, convertidos a Mbps), y la latencia media (calculada como la suma total de las latencias de todos los paquetes recibidos dividida por el número de paquetes recibidos).

```
for (std::map<ns3::FlowId, ns3::FlowMonitor::FlowStats>::
   ↪ const_iterator i = stats.begin (); i != stats.end (); ++i)
{
    ns3::Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow
       ↪ (i->first);
    std::cout << "Flujo␣" << i->first << "␣(" << t.sourceAddress
       ↪  << "␣->␣" << t.destinationAddress << ")\n";
    std::cout << "␣Tx␣Bytes:␣" << i->second.txBytes << "\n";
    std::cout << "␣Rx␣Bytes:␣" << i->second.rxBytes << "\n";
    std::cout << "␣Throughput:␣" << i->second.rxBytes * 8.0 / (i
       ↪ ->second.timeLastRxPacket.GetSeconds()-i->second.
       ↪ timeFirstTxPacket.GetSeconds()) / 1024 / 1024 << "␣
       ↪ Mbps\n";
    std::cout << "␣Latencia␣media:␣" << i->second.delaySum.
       ↪ GetSeconds() / i->second.rxPackets << "␣s\n";
}
```