# Módulo 3
# Modelado de protocolos de red

© 2024

**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

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

To view a copy of this license, visit:
http://creativecommons.org/licenses/by-nc-sa/4.0/legalcode

# Índice

# 3. Modelado de protocolos de red

Este módulo, se adentrará en el modelado de los diferentes protocolos de red en ns-3. Esto incluye protocolos de la capa física, capa de enlace de datos, enrutamiento, transporte y aplicación.

## 3.1. Modelos de rred: OSI y TCP/IP

Los modelos de red son estructuras conceptuales que describen cómo se comunican los sistemas de una red y cómo se deben organizar las funciones de red. Dos de los modelos más importantes y ampliamente utilizados son el modelo OSI y el modelo TCP/IP.

### 3.1.1. Modelo OSI

El Modelo de Interconexión de Sistemas Abiertos (OSI) es un marco de referencia que define cómo se deben interconectar los sistemas informáticos para permitir la comunicación de red (Figura 1). Fue desarrollado por la Organización Internacional de Normalización (ISO) en la década de 1980 y se compone de siete capas, cada una con funciones específicas. Estas capas, en orden ascendente, son:

1. **Capa física (Physical Layer):** Se encarga de la transmisión de bits a través del medio de transmisión físico, definiendo las características eléctricas, mecánicas y funcionales del medio.
2. **Capa de enlace de datos (Data Link Layer):** Proporciona la transferencia confiable de datos a través de un enlace físico, corrigiendo errores que puedan ocurrir en la transmisión y organizando los datos en tramas.
3. **Capa de red (Network Layer):** Se encarga de la conmutación y enrutamiento de datos a través de la red, determinando la mejor ruta para el envío de paquetes y evitando congestiones.
4. **Capa de transporte (Transport Layer):** Proporciona la entrega de datos de extremo a extremo, controlando el flujo y la congestión de la información y asegurando que los datos lleguen correctamente al destino.
5. **Capa de sesión (Session Layer):** Establece, mantiene y cierra las conexiones entre las aplicaciones de los sistemas, permitiendo la comunicación entre ellas.
6. **Capa de presentación (Presentation Layer):** Se encarga de la representación de datos, la compresión y el cifrado, asegurando que los datos sean presentados de manera adecuada a las aplicaciones.
7. **Capa de aplicación (Application Layer):** Proporciona servicios de red a las aplicaciones y usuarios, permitiendo la interacción con los recursos de red.


Figura 1: Diagrama del Modelo OSI

### 3.1.2. Modelo TCP/IP

El Protocolo de Control de Transmisión/Protocolo de Internet (TCP/IP) es el modelo de red más utilizado en Internet y en muchas redes empresariales (Figura 2). Fue desarrollado por el Departamento de Defensa de los Estados Unidos en la década de 1970 para conectar sistemas informáticos en la red militar ARPANET. A diferencia del modelo OSI, el modelo TCP/IP consta de cuatro capas, que son:

1. **Capa de acceso a red (Network Access Layer):** Esta capa es responsable de las comunicaciones de bajo nivel hacia y desde la red física. Incluye todo lo necesario para que los datos se transmitan a través de diferentes tipos de medios, como Ethernet (cable) y Wi-Fi (inalámbrico). En el modelo OSI, esta capa abarca las funciones de las capas física y de enlace de datos.

2. **Capa de internet (Internet Layer):** Esta capa proporciona la capacidad de enviar paquetes de datos desde un nodo a otro dentro de la misma red o a través de redes interconectadas. Utiliza el Protocolo de Internet (IP) para direccionar y enrutamiento, asegurando que los datos lleguen a su destino correcto. Equivalente a la capa de red en el modelo OSI.

3. **Capa de transporte (Transport Layer):** Encargada de la transmisión fiable de datos entre puntos finales a través de la red. Los protocolos clave en esta capa son el Protocolo de Control de Transmisión (TCP), que ofrece conexiones orientadas a la conexión y garantiza la entrega de datos, y el Protocolo de Datagramas de Usuario (UDP), que es más simple y proporciona servicios sin conexión y sin garantía de entrega. Corresponde a la capa de transporte en el modelo OSI.

4. **Capa de aplicación (Application Layer):** La capa superior que permite a los usuarios interactuar con aplicaciones de red. Incluye protocolos de alto nivel como el Protocolo de Transferencia de Hipertexto (HTTP), el Protocolo de Transferencia de Archivos (FTP), y el Protocolo Simple de Transferencia de Correo (SMTP).

Esta capa se alinea con las capas de aplicación, presentación y sesión del modelo OSI.

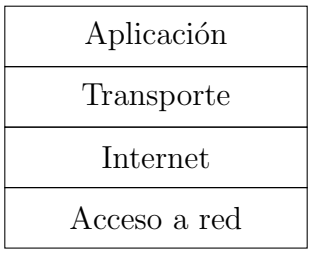


Figura 2: Diagrama del Modelo TCP/IP.

## 3.2. Protocolos de capa física

En ns-3, se pueden modelar una amplia variedad de protocolos de capa física, cada uno diseñado para satisfacer diferentes necesidades y requisitos de red. Estos protocolos abarcan desde estándares ampliamente utilizados hasta modelos de canal detallados para simular efectos de propagación realistas. Algunos de los protocolos de capa física más relevantes que se pueden modelar en ns-3 son:

- **IEEE 802.11:** Comúnmente conocido como Wi-Fi, define los estándares para redes inalámbricas de área local (WLAN). IEEE 802.11 es utilizado en una variedad de entornos, desde hogares y oficinas hasta entornos públicos como cafeterías y aeropuertos. Proporciona conectividad inalámbrica a dispositivos como computadoras portátiles, teléfonos inteligentes, tablet y dispositivos IoT. El protocolo IEEE 802.11 se ha vuelto fundamental en la conectividad moderna, con múltiples variantes como 802.11a, 802.11b, 802.11g, 802.11n, 802.11ac y 802.11ax, cada una con características y capacidades específicas.

  En el Código 1 se establece una red inalámbrica en ns-3 utilizando el estándar `802.11a`. Comienza configurando las propiedades del canal y la capa física de los dispositivos WiFi, utilizando los helpers `YansWifiChannelHelper` y `YansWifiPhyHelper` respectivamente. A continuación, se crea un helper de WiFi y se establece el estándar WiFi en `802.11a`. Se configura la capa MAC para los nodos de estación (STA) y el nodo de punto de acceso (AP) utilizando el helper `WifiMacHelper`. Se especifica el SSID de la red a la que se conectarán los nodos STA y se desactiva la sonda activa para ellos. Para el AP, se especifica el SSID de la red que va a anunciar. Luego, se instalan los dispositivos WiFi en los nodos STA y AP utilizando el método `Install` del helper de WiFi. Finalmente, los dispositivos WiFi se asignan a los nodos correspondientes.

Código 1: Simulación de IEEE 802.11 en ns-3

```
YansWifiChannelHelper channel = YansWifiChannelHelper::
    ↪ Default ();
```

```
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();

wifi.SetStandard (WIFI_PHY_STANDARD_80211a);

WifiMacHelper mac;

NetDeviceContainer staDevices;
NetDeviceContainer apDevice;

mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue ("network"),
             "ActiveProbing", BooleanValue (false));
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac",
             "Ssid", SsidValue ("network"));
apDevice = wifi.Install (phy, mac, wifiApNode);
```

- **IEEE 802.15.4:** Este protocolo se utiliza en redes de área personal inalámbrica (WPAN) y es común en aplicaciones de Internet de las cosas (IoT). IEEE 802.15.4 proporciona una conectividad inalámbrica de bajo consumo de energía y baja tasa de datos para dispositivos IoT, como sensores y actuadores. Es especialmente adecuado para aplicaciones donde la duración de la batería es crítica, como sistemas de monitoreo remoto y automatización del hogar. IEEE 802.15.4 define tanto la capa física como la capa de enlace de datos, lo que permite una comunicación eficiente y confiable en entornos con restricciones de recursos.

  El Código 2 configura una red inalámbrica en ns-3 con el estándar IEEE 802.15.4. Primero, se crea un helper de canal de WiFi llamado `channel`. Se establece un modelo de retardo de propagación constante y un modelo de pérdida de propagación de Friis en este canal. Luego, se configura la capa física de los dispositivos WiFi utilizando el helper `YansWifiPhyHelper`. Se asigna el canal creado anteriormente a esta capa física. Se inicializa un helper de WiFi, donde se configura el administrador de estación remota para utilizar tasas de datos constantes, tanto para datos como para control. El estándar de WiFi se establece en `802.11_15_4` para reflejar el estándar IEEE 802.15.4. Finalmente, se crea un helper de capa de control de acceso al medio (MAC) utilizando `NqosWifiMacHelper`, utilizando la configuración predeterminada.

Código 2: Configuración de IEEE 802.15.4 en ns-3

```
YansWifiChannelHelper channel;
channel.SetPropagationDelay ("ns3::
   ↪ ConstantSpeedPropagationDelayModel");
```

```
channel.AddPropagationLoss ("ns3::FriisPropagationLossModel"
   ↪ );

YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager"
   ↪ , "DataMode", StringValue ("DsssRate1Mbps"), "
   ↪ ControlMode", StringValue ("DsssRate1Mbps"));

wifi.SetStandard (WIFI_PHY_STANDARD_80211_15_4);

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
}
```

Adicionalmente, ns-3 proporciona modelos de canal que simulan la atenuación de la señal y otros efectos de propagación en el medio de transmisión. Estos modelos son esenciales para evaluar el rendimiento de los protocolos de capa física en diferentes entornos y condiciones, como la presencia de obstáculos físicos, interferencia y atenuación de la señal. Se pueden utilizar estos modelos para realizar análisis de la calidad de la señal, la tasa de error de bit (BER) y otros parámetros de rendimiento en diversas configuraciones de red. ns-3 ofrece una variedad de modelos de atenuación, que incluyen modelos de atenuación libre espacio, modelos de atenuación de propagación multi-camino (MIMO), modelos de atenuación de líneas de transmisión, entre otros.

En el Código 3 se crea un modelo de pérdida de propagación de tipo Friis[1] utilizando la función `CreateObject<FriisPropagationLossModel>()`. Este modelo se asigna a un puntero llamado `lossModel`. Luego, se agrega este modelo de pérdida al canal WiFi utilizando el método `AddPropagationLoss` del helper de canal. Después, se configura la capa física de los dispositivos WiFi utilizando el helper `YansWifiPhyHelper`. Por úlitmo, se asigna el canal creado anteriormente a esta capa física utilizando el método `SetChannel`.

Código 3: Configuración de la atenuación en ns-3

```
YansWifiChannelHelper channel;
channel.SetPropagationDelay ("ns3::
   ↪ ConstantSpeedPropagationDelayModel");

Ptr<PropagationLossModel> lossModel = CreateObject<
   ↪ FriisPropagationLossModel> ();
channel.AddPropagationLoss (lossModel);
```

[1]El modelo de pérdida de Friis describe cómo la potencia de la señal se reduce con el aumento de la distancia entre el transmisor y el receptor en un espacio libre sin obstáculos, basándose en la frecuencia de la señal y la distancia entre las antenas.

```
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());
}
```

## 3.3. Protocolos de capa de enlace de datos

Algunos de los protocolos de capa de enlace de datos más comúnmente utilizados en ns-3 incluyen:

- **CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance):** Este protocolo se utiliza en redes LAN inalámbricas para evitar colisiones entre los nodos que compiten por el acceso al medio de transmisión. CSMA/CA utiliza un mecanismo de escucha antes de transmitir, donde los nodos verifican si el canal está libre antes de iniciar la transmisión. Si detectan actividad en el canal, esperan un tiempo aleatorio antes de intentar transmitir nuevamente, lo que ayuda a reducir las colisiones y mejorar la eficiencia del canal.

  En el Código 4 se crea una red de dos nodos utilizando el helper `NodeContainer`, donde se llama al método `Create(2)` para crear dos nodos. Luego, se configura un canal CSMA (Carrier Sense Multiple Access) utilizando el helper `CsmaHelper`. Se establecen las características del canal, especificando la tasa de datos como `"100Mbps"` y el retardo como 6560 nanosegundos (correspondiente a 6560 ns o 6.56 µs). Estas configuraciones se aplican al canal mediante los métodos `SetChannelAttribute`. Finalmente, se instalan los dispositivos de red en los nodos recién creados utilizando el método `Install` del helper `CsmaHelper`, y los dispositivos resultantes se almacenan en el contenedor `devices`.

  Código 4: Configuración de CSMA/CA en ns-3

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

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"
   ↪ ));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds
   ↪ (6560)));

NetDeviceContainer devices = csma.Install (nodes);
}
```

- **TDMA (Time Division Multiple Access):** es un protocolo de acceso múltiple que divide el tiempo en intervalos y asigna cada intervalo a un nodo para la transmisión de datos. Cada nodo puede transmitir durante su intervalo asignado sin interferir con otros nodos en la red. TDMA se utiliza en redes de comunicaciones

móviles y satelitales, donde se necesita un acceso determinista y sincronizado al medio de transmisión.

En el Código 5 se configura un esquema TDMA en ns-3. Se crea un helper de TDMA llamado `tdma`. A través de este helper, se establecen las características del dispositivo TDMA, incluyendo la tasa de datos y el retardo del canal. La tasa de datos se configura como 1Mbps utilizando el método `SetDeviceAttribute`, mientras que el retardo del canal se establece en 2 milisegundos usando el método `SetChannelAttribute`. Posteriormente, se instala TDMA en los nodos de la red utilizando el método `Install` del helper de TDMA. Los dispositivos resultantes se almacenan en el contenedor `devices`.

Código 5: Configuración de TDMA en ns-3

```
TdmaHelper tdma;
tdma.SetDeviceAttribute ("DataRate", DataRateValue (DataRate
   ↪  ("1Mbps")));
tdma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds
   ↪ (2)));

NetDeviceContainer devices = tdma.Install (nodes);
}
```

- **Ethernet:** uno de los protocolos de capa de enlace de datos más ampliamente utilizados en redes cableadas. Define el formato de trama y las reglas para la transmisión de datos entre dispositivos conectados a través de un medio compartido, como un cable de cobre o fibra óptica. Ethernet utiliza el protocolo MAC para gestionar el acceso al medio de transmisión y garantizar una comunicación ordenada y sin colisiones entre los dispositivos de la red.

  En el Código 6 se configura una conexión punto a punto en ns-3 usando Ethernet. Primero se crea un helper de punto a punto llamado `PointToPointHelper`. Se establecen las características del dispositivo utilizando el método `SetDeviceAttribute`, donde se especifica la tasa de datos como `100Mbps`. Luego, se configuran las características del canal utilizando el método `SetChannelAttribute`, donde se establece el retardo del canal en `2ms`. Después, se crea un contenedor de dispositivos de red llamado `NetDeviceContainer`. Se instalan los dispositivos punto a punto en los nodos de la red utilizando el método `Install` del helper de punto a punto.

  Código 6: Configuración de Ethernet en ns-3

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("
   ↪ 100Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms
   ↪ "));

NetDeviceContainer devices;
```

```
devices = pointToPoint.Install (nodes);
}
```

## 3.4. Protocolos de enrutamiento

Algunos de los protocolos de enrutamiento más comúnmente utilizados en ns-3 son:

- **AODV (Ad hoc On-Demand Distance Vector):** protocolo de enrutamiento ampliamente utilizado en redes móviles ad hoc, donde los nodos pueden moverse libremente sin una infraestructura de red fija. AODV utiliza un enfoque bajo demanda para determinar las rutas hacia los destinos, lo que significa que solo se establecen rutas cuando es necesario enviar datos. Esto lo hace eficiente en términos de uso de ancho de banda y adaptabilidad a cambios en la topología de la red.

  En el Código 7 se configura el protocolo de enrutamiento AODV utilizando el helper `AodvHelper`. Se crea una lista de enrutamiento IPv4 utilizando el helper `Ipv4ListRoutingHelper`, y se agrega el protocolo AODV a la lista con un valor de prioridad de 10. Esto significa que este protocolo de enrutamiento se usará con prioridad sobre otros protocolos de enrutamiento IPv4. Luego, se establece esta lista como la lista de enrutamiento para la pila de protocolos utilizando el método `SetRoutingHelper`, y finalmente se instala la lista de enrutamiento en los nodos utilizando el método `Install` del helper `stack`.

  Código 7: Configuración de AODV en ns-3

```
AodvHelper aodv;
Ipv4ListRoutingHelper list;
list.Add (aodv, 10);
stack.SetRoutingHelper (list);
stack.Install (nodes);
```

- **DSR (Dynamic Source Routing):** otro protocolo de enrutamiento popular para redes ad hoc, que también opera bajo demanda. En lugar de mantener tablas de enrutamiento en cada nodo, DSR utiliza el enfoque de "fuente"donde los paquetes llevan información de enrutamiento sobre la ruta completa hacia el destino. Esto elimina la necesidad de mantener información de enrutamiento en los nodos intermedios, lo que lo hace adecuado para redes con alta movilidad y cambios frecuentes en la topología.

  En el Código 8 se configura DSR utilizando el helper `DsrHelper`. Se agrega el helper de DSR a la lista de helpers de enrutamiento IPv4 con un valor de prioridad de 10 utilizando el método `Add`. A continuación, se configura el helper de enrutamiento en la pila de protocolos de Internet utilizando el método `SetRoutingHelper`.

  Código 8: Configuración de DSR en ns-3

```
DsrHelper dsr;
Ipv4ListRoutingHelper list;
list.Add (dsr, 10);
stack.SetRoutingHelper (list);
stack.Install (nodes);
}
```

- **OSPF (Open Shortest Path First):** comúnmente utilizado en redes IP para calcular las rutas más cortas entre los nodos. Utiliza el algoritmo de Dijkstra para determinar las rutas óptimas en función de los costos de enlace, como la distancia o el ancho de banda disponible. Ampliamente utilizado en redes empresariales y de proveedores de servicios de Internet debido a su capacidad para adaptarse a cambios en la topología de la red y su escalabilidad.

  En el Código 9, primero, se crea un helper de OSPF llamado `ospf`. Luego, se crea un helper de enrutamiento estático de IPv4 denominado `staticRouting`. Se utiliza un helper de enrutamiento de lista de IPv4 llamado `list` para agregar los protocolos de enrutamiento estático y OSPF, asignándoles prioridades de 0 y 10 respectivamente.

Código 9: Configuración de OSPF en ns-3

```
OspfHelper ospf;
Ipv4StaticRoutingHelper staticRouting;
Ipv4ListRoutingHelper list;
list.Add (staticRouting, 0);
list.Add (ospf, 10);
stack.SetRoutingHelper (list);
stack.Install (nodes);
```

- **BGP (Border Gateway Protocol):** utilizado en Internet para intercambiar información de enrutamiento entre sistemas autónomos (AS). Es un protocolo de enrutamiento externo que permite a los routers intercambiar información de enrutamiento y tomar decisiones sobre cómo enrutar el tráfico entre diferentes AS. Es crucial para la interconexión de diferentes redes y proveedores de servicios.

  En el Código 10, se muestra cómo configurar este protocolo de enrutamiento.

Código 10: Configuración de BGP en ns-3

```
BgpHelper bgp;
Ipv4StaticRoutingHelper staticRouting;
Ipv4ListRoutingHelper list;
list.Add (staticRouting, 0);
list.Add (bgp, 10);
stack.SetRoutingHelper (list);
stack.Install (nodes);
```

## 3.5. Protocolos de transporte y aplicación

- **TCP (Transmission Control Protocol):** Uno de los protocolos de transporte más ampliamente utilizados en Internet, proporciona una transmisión de datos fiable y orientada a conexión. TCP utiliza técnicas como el control de congestión y la retransmisión de paquetes para garantizar la entrega de datos de manera ordenada y sin errores. En ns-3, se puede modelar el comportamiento de TCP en una variedad de escenarios de red para evaluar su rendimiento y comportamiento en condiciones realistas.

  En el Código 11, primero, se define un puerto de destino, almacenado en la variable `port`, con el valor `50000`. Luego, se crea una dirección de destino (`sinkAddress`) utilizando la dirección IP del segundo nodo de la red (`interfaces.GetAddress(1)`) y el puerto definido anteriormente. A continuación, se configura un helper (`PacketSinkHelper`) para crear una aplicación de recepción de paquetes TCP. Se especifica el tipo de socket a utilizar, en este caso, `ns3::TcpSocketFactory`, y se asigna la dirección de destino creada anteriormente. Finalmente, se instala la aplicación de recepción en el segundo nodo de la red (`nodes.Get(1)`) utilizando el método `Install` del helper `sinkHelper`, y se almacena en un contenedor de aplicaciones (`sinkApps`).

Código 11: Configuración de TCP en ns-3

```
uint16_t port = 50000;
Address sinkAddress (InetSocketAddress (interfaces.
   ↪ GetAddress (1), port));
PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory",
   ↪ InetSocketAddress (Ipv4Address::GetAny (), port));
ApplicationContainer sinkApps = sinkHelper.Install (nodes.
   ↪ Get (1));
```

- **UDP (User Datagram Protocol):** otro protocolo de transporte comúnmente utilizado en Internet para proporcionar una transmisión de datos no fiable y sin conexión. A diferencia de TCP, UDP no garantiza la entrega de datos ni el orden en que se reciben. Es adecuado para aplicaciones donde la velocidad y la eficiencia son más importantes que la integridad de los datos, como la transmisión de audio y video en tiempo real.

  En el Código 12, primero, se define un número de puerto UDP utilizando la variable `port`, que se establece en 9. Luego, se crea un servidor UDP de eco utilizando el helper `UdpEchoServerHelper`, especificando el puerto previamente definido. Este servidor se instala en el nodo 1 de la red utilizando el método `Install` del contenedor de aplicaciones `serverApps`. A continuación, se inicia la aplicación del servidor utilizando el método `Start` con un tiempo de inicio de 0 segundos y se detiene después de 10 segundos usando el método `Stop`. Después, se configura un cliente UDP de eco utilizando el helper `UdpEchoClientHelper`, especificando la dirección IP del nodo 1 y el mismo puerto que el servidor. Se establecen los atri-

butos del cliente, como el número máximo de paquetes, el intervalo entre paquetes y el tamaño del paquete.

Código 12: Configuración de UDP en ns-3

```
uint16_t port = 9;
UdpEchoServerHelper echoServer (port);
ApplicationContainer serverApps = echoServer.Install (nodes.
   ↪ Get (1));
serverApps.Start (Seconds (0.0));
serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (interfaces.GetAddress (1),
   ↪ port);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds
   ↪ (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024))
   ↪ ;
```

- **HTTP (Hypertext Transfer Protocol):** HTTP es el protocolo utilizado para la transferencia de datos en la web. Define cómo se solicitan y entregan los recursos, como páginas web, imágenes y archivos, entre un servidor web y un cliente.

  En el Código 13, se define un puerto `port` con el valor 80, que es el puerto estándar para HTTP. Luego, se crea un servidor HTTP utilizando el helper `HttpServerHelper` con el puerto especificado. Este servidor se instala en el segundo nodo de la red (`nodes.Get(1)`) y se almacena en un contenedor de aplicaciones llamado `serverApps`. Posteriormente, se inicia el servidor en el tiempo 0 segundos y se detiene en el tiempo 10 segundos. A continuación, se crea un cliente HTTP utilizando el helper `HttpClientHelper`. Se especifica la dirección IP del servidor (obtenida a través de `interfaces.GetAddress(1)`) y el puerto utilizado. Se establecen los atributos del cliente, como el número máximo de paquetes a enviar, el intervalo entre envíos y el tamaño de los paquetes. Este cliente se instala en el primer nodo de la red (`nodes.Get(0)`) y se almacena en un contenedor de aplicaciones llamado `clientApps`. Finalmente, se inicia el cliente en el tiempo 1 segundo y se detiene en el tiempo 10 segundos.

Código 13: Configuración de HTTP en ns-3

```
uint16_t port = 80;
HttpServerHelper httpServer (port);
ApplicationContainer serverApps = httpServer.Install (nodes.
   ↪ Get (1));
serverApps.Start (Seconds (0.0));
serverApps.Stop (Seconds (10.0));
```

```
HttpClientHelper httpClient (interfaces.GetAddress (1), port
   ↪ );
httpClient.SetAttribute ("MaxPackets", UintegerValue (1));
httpClient.SetAttribute ("Interval", TimeValue (Seconds
   ↪ (1.0)));
httpClient.SetAttribute ("PacketSize", UintegerValue (1024))
   ↪ ;
ApplicationContainer clientApps = httpClient.Install (nodes.
   ↪ Get (0));
clientApps.Start (Seconds (1.0));
clientApps.Stop (Seconds (10.0));
```

- **FTP (File Transfer Protocol):** utilizado para la transferencia de archivos entre sistemas en una red. Permite a los usuarios cargar y descargar archivos de servidores remotos de manera eficiente y segura.

  En el Código 14, se define un puerto de conexión FTP como un entero sin signo de 16 bits, utilizando la variable `port` y se le asigna el valor 21. Luego, se crea un objeto `FtpHelper` para configurar la conexión FTP. Se inicializa con la dirección IP del servidor FTP y se establecen atributos para la conexión FTP. Se especifica el directorio remoto al que se accederá mediante el método `SetAttribute` con el nombre de atributo `RemoteDir` y el valor `/` para indicar el directorio raíz. Además, se establece el límite máximo de bytes a transferir como 1024, utilizando el método `SetAttribute` con el nombre de atributo `MaxBytes` y el valor `UintegerValue(1024)`. Después de configurar la conexión FTP, se instala la aplicación FTP en el nodo 0 mediante el método `Install` del objeto `FtpHelper`, y el resultado se almacena en un contenedor de aplicaciones `ApplicationContainer` llamado `ftpApps`.

Código 14: Configuración de FTP en ns-3

```
uint16_t port = 21;
FtpHelper ftp (interfaces.GetAddress (1), port);
ftp.SetAttribute ("RemoteDir", StringValue ("/"));
ftp.SetAttribute ("MaxBytes", UintegerValue (1024));
ApplicationContainer ftpApps = ftp.Install (nodes.Get (0));
ftpApps.Start (Seconds (1.0));
ftpApps.Stop (Seconds (10.0));
}
```