Universidad de Castilla-La Mancha
Escuela Superior de Ingeniería Informática

Trabajo Fin de Grado
Grado en Ingeniería Informática
Tecnología específica de Ingeniería de Computadores

Modelado del protocolo RDMA en un simulador de redes de interconexión basado en OMNeT++

Víctor Manuel Romero Ramírez

11 de julio de 2022

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Tecnología específica de Ingeniería de Computadores

Modelado del protocolo RDMA en un simulador de redes de interconexión basado en OMNeT++

Autor: Víctor Manuel Romero Ramírez

Tutor: Jesús Escudero Sahuquillo

Co-Tutor: Francisco José Alfaro Cortés

11 de julio de 2022

A todas las personas que me han ayudado a llegar hasta aquí, sobre todo a mi familia y a mis amigos

Declaración de autoría

Yo, Víctor Manuel Romero Ramírez con DNI 49.215.454 , declaro que soy el único autor del trabajo fin de grado titulado "Modelado del protocolo RDMA en un simulador de redes de interconexión basado en OMNeT++", que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual, y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 11 de julio de 2022

Fdo.: Víctor Manuel Romero Ramírez

Resumen

Remote Direct Memory Access (RDMA) es un protocolo de comunicación de redes de interconexión de altas prestaciones que usa Acceso Directo a Memoria (DMA, Direct Memory Access), es decir, permite a las aplicaciones distribuidas en varios nodos intercambiar datos directamente desde un host accediendo directamente a la memoria de otro host.

En este trabajo usaremos la herramienta de simulación OMNeT++ que es extensible, modular y basado en componentes, principalmente usado para la construcción de simuladores de red. OMNeT++ ha ganado una gran popularidad como plataforma de simulación de red, pues proporciona una arquitectura de componentes reutilizables en diferentes modelos de simulación. Los componentes (módulos) se programan en C++ y luego se ensamblan en componentes y modelos más complejos utilizando un lenguaje de descripción de de red de alto nivel (NED). OMNeT++ tiene un amplio soporte de interfaz gráfica de usuario y, debido a su arquitectura modular, el kernel de simulación (y los modelos) se pueden integrar fácilmente en sus aplicaciones. [Omn, 2019b]

El objetivo de este TFG es el modelado del protocolo RDMA en un simulador de redes de interconexión de altas prestaciones basado en OMNeT++, pues la herramienta no tiene esta funcionalidad. Por último, compararemos este protocolo con protocolos TCP/IP como TCP y UDP mediante experimentos con el objetivo de mostrar que tiene un mejor funcionamiento.

Agradecimientos

Me gustaría agradecer a mis padres, por haberme pagado la carrera y haberme apoyado en todo momento con todo lo relacionado a los estudios, a todos las personas que he conocido a lo largo de mi vida, tanto a aquellas que te ayudan a crecer estando a tu lado, como a aquellas que suponen un reto para ti y te ayudan a superar momentos y superarte a tí mismo. Por supuesto, también agradecer a todos los profesores que me han ayudado a llegar hasta aquí y, sobre todo, a Jesús y a Paco, por confiar en mí y proponerme la realización de este TFG.

Índice general

Índice de figuras

1. Introducción

En este capítulo se describe la motivación de este trabajo, así como los objetivos del mismo. También se detalla la estructura y organización de este documento.

1.1. Motivación

La demanda de alto rendimiento y baja latencia es cada vez mayor en las redes de interconexión de altas prestaciones que forman parte de grades supercomputadores y centros de proceso de datos. En estas situaciones, las aplicaciones de cómputo intensivo es cada vez más frecuente en la computación de altas prestaciones (High Performance Computing, HPC) y las aplicaciones de almacenamiento masivo de datos (High Performance Data Analytics, HPDA), ejecutando un elevado número de operaciones de comunicación que involucran a la red de interconexión. Por ello, el papel de estas redes es cada vez más importante.

En el diseño de las redes de interconexión juegan un papel fundamental algunos factores como la topología, la estructura del switch, el algoritmo de encaminamiento, o las técnicas de control de flujo. El diseño de la topología puede variar dependiendo de cómo los nodos estén organizados y se comuniquen entre ellos. Tenemos varios tipos de topologías como la redes directas, cuyos nodos se conectan entre vecinos; las redes indirectas, donde los nodos se comunican a través de switches; las redes híbridas que combinan en concepto de red directa e indirecta; y las redes jerárquicas, cuyos nodos se organizan en grupos [Liu, 2018].

El protocolo TCP/IP tradicional supone un problema para las operaciones de comunicación en las redes de altas prestaciones debido a que los requisitos de las aplicaciones. TCP/IP utiliza los sockets para la comunicación entre nodos; realiza una copia de los datos que las aplicaciones quieren enviar desde su espacio de usuario en los buffers internos de los nodos, y después en la memoria reservada en el kernel del nodo origen, antes de enviar los datos por la red. En el nodo destino, se produce el proceso inverso: al recibirse los datos en la interfaz de red, se copia en la memoria del kernel del sistema operativo del nodo destino y luego en la memoria de usuario de ese mismo

nodo. Por ello, el protocolo RDMA (Remote Direct Memory Access) ha comenzado a tener un mayor protagonismo en las redes de altas prestaciones. Las redes con RDMA no realizan copias intermedias entre la memoria de usuario y kernel. Además, son capaces de proporcionar un gran ancho de banda, bajo uso de la CPU y una latencia significativamente menor con respecto a las redes que usan el protocolo TCP/IP.

Por ello, en este trabajo se persigue el objetivo de modelar e implementar en el simulador OMNeT++ el protocolo RDMA y comparar sus prestaciones con otros protocolos de transporte incluidos en TCP/IP, como son TCP y UDP.

Para realizar esta tarea nos basaremos en la implementación de UDP, que está disponible en el simulador, para desarrollar la implementación de RDMA. Además, también será necesario realizar unas pequeñas modificaciones a esta implementación de UDP; como incluirle el código para controlar el estado del enlace, ya que únicamente estaba implementado para la utilización de enlaces ideales, es decir, enlaces que no generan latencia; para poder realizar correctamente nuestras simulaciones. Asimismo, también será necesario tocar partes del código de TCP que al igual que UDP, no tiene el código necesario para la utilización de enlaces que generen latencia. Una vez tengamos implementado RDMA montaremos varios escenarios de prueba donde se ejecutará tráfico TCP, tráfico UDP y tráfico RDMA. Los experimentos se han ejecutado en el clúster CELLIA, donde se ha instalado el simulador OMNeT++, con su correspondiente biblioteca INET.

1.2. Objetivos

El objetivo de este Trabajo de Fin de Grado es el de modelar el protocolo de transporte RDMA en el modelo INET del simulador OMNeT++ y realizar las comparaciones necesarias para mostrar los beneficios que nos aporta el uso de una red con protocolo RDMA respecto a una red con protocolo TCP/IP. Para ello, se han llevado a cabo las siguientes tareas:

1. Conocer el estado del arte de las redes de interconexión en general y del protocolo RDMA en particular.
2. Estudiar las herramientas de simulación existentes basadas en OMNeT++, como el modelo INET que ya dispone del protocolo de transporte TCP/IP.
3. Modelar el protocolo RDMA en el modelo INET de OMNeT++, y verificar que se comporta conforme a los esperado.
4. Realizar experimentos de simulación comparando varias configuraciones de red de interconexión que usen TCP/IP y RDMA, y hacer un estudio comparativo de las prestaciones.

1.3. Estructura de la memoria

La memoria de este Trabajo de Fin de Grado está organizada en los capítulos que se desarrollan a continuación:

- **Introducción:** Se exponen la motivación del proyecto y los objetivos del mismo.
- **Antecedentes y estado de la cuestión:** Se enfoca en ofrecer una introducción sobre la redes de interconexión. En ella se describen las diferentes topologías de redes de interconexión y sus características específicas. También se indican los componentes de una red de interconexión, la arquitectura interna del switch, diferentes aspectos sobre el encaminamiento de los mensajes por la red y el control de flujo. También se describe el protocolo de transporte RDMA, y se dan algunos detalles sobre las herramientas de simulación.
- **Metodología y desarrollo:** Se explica la metodología utilizada, los procedimientos que se han llevado a cabo para alcanzar los objetivos de este trabajo. Se describen los pasos seguidos para la instalación del entorno de simulación OMNeT++ en el clúster CELLIA y el modelado del protocolo RDMA y las modificaciones necesarias en el código ya existente para adecuarlo a nuestras necesidades.
- **Experimentos y resultados:** En este capítulo se analizan los resultados de simulación obtenidos en los diferentes escenarios de red simulados. Se han configurado redes de diferente tamaño, tamaño de mensaje y tasa de inyección de mensajes. Todos los resultados obtenidos estarán reflejados en sus respectivas gráficas.
- **Conclusiones:** Finalmente, en este apartado se plasman las conclusiones del proyecto. Por último, también se proponen posibles líneas de trabajo futuro este proyecto.

2. Antecedentes y estado de la cuestión

En este capítulo vamos a conocer el estado del arte desde un punto de vista general de las redes de interconexión y, dentro de todos los protocolos de transporte que existen y se usan en estas, vamos a centrarnos en el protocolo de capa de transporte RDMA (Remote Direct Memory Access) que nos permite transferir datos de forma más rápida que los protocolos de transporte tradicionales incluidos en TCP/IP, como TCP y UDP.

2.1. Redes de interconexión

Los sistemas de computación de alto rendimiento utilizan redes de interconexión como base de comunicación entre los elementos que componen el procesador. En su forma más general, las redes de interconexión son el componente central de todos los sistemas informáticos y de comunicación, desde las interconexiones internas de las arquitecturas integradas a nivel de chip hasta los sistemas a escala geográfica, como las redes WAN e Internet.

El objetivo de todas las redes de interconexión es transferir información desde cualquier nodo origen a cualquier nodo destino que se desee, con la menor latencia posible y permitiendo un gran número de transferencias en ejecución simultánea.

Estas redes están compuestas por enlaces y conmutadores, que ayudan a enviar la información entre ambos nodos. Una red está especificada por su topología, algoritmo de enrutamiento, estrategia de conmutación y mecanismo de control de flujo [Int, 2021] [Jose Duato, 2003]. Como hemos dicho anteriormente, las redes de interconexión desempeñan un papel importante en el rendimiento de los computadores paralelos modernos. Hay muchos factores que pueden afectar a la elección de una red de interconexión adecuada, entre los que se encuentran:

- **Requisitos de rendimiento:** La latencia de los mensajes (tiempo transcurrido entre el momento en que se genera un mensaje en su nodo de origen y el momento en que se entrega el mensaje en su nodo de destino) afecta directamente al tiempo de inactividad del procesador y al tiempo de acceso a la memoria de

las ubicaciones remotas. Además, la red puede saturarse, es decir, puede ser incapaz de entregar el flujo de mensajes inyectados por los nodos, lo que limita la potencia de cálculo efectiva de un ordenador en paralelo. La cantidad máxima de información entregada por la red por unidad de tiempo define el rendimiento de dicha red.

- **Escalabilidad:** Una arquitectura escalable implica que a medida que se añaden más procesadores su ancho de banda (cantidad de datos que podemos enviar y recibir por unidad de tiempo, expresado en bits por segundo) aumenta proporcionalmente. De lo contrario, si el ancho de banda no escala puede convertirse en un cuello de botella para el resto del sistema, disminuyendo la eficiencia global en consecuencia.
- **Expansión incremental:** Las redes de interconexión deben ofrecer una capacidad de añadir un pequeño número de nodos y minimizar el desperdicio de los recursos ya existentes.
- **Particionado:** Los ordenadores en paralelo suelen ser compartidos por varios usuarios a la vez. Es deseable que el tráfico de red producido por cada usuario no afecte al rendimiento de otras aplicaciones. Esto puede garantizarse si la red puede dividirse en subsistemas funcionales más pequeños. La divisibilidad también puede ser necesaria por razones de seguridad, por ejemplo, para evitar posibles interferencias indeseadas entre distintos usuarios.
- **Simplicidad:** Los diseños sencillos suelen dar lugar a frecuencias de reloj más altas y pueden lograr un mayor rendimiento. Además, permiten al cliente explotar al máximo su rendimiento al ser fáciles de entender.
- **Distancia de expansión:** En los multicomputadores y en la memoria compartida distribuida, la red se monta dentro de unos pocos armarios. La distancia máxima entre nodos es pequeña. En consecuencia, las señales se suelen transmitir mediante cables de cobre que pueden disponerse de forma regular, lo que reduce el tamaño del ordenador y la longitud de los cables. Sin embargo, en los sistemas tipo clúster (compuesto por estaciones de trabajo independientes interconectadas entre sí), los enlaces tienen longitudes muy diferentes y algunos enlaces pueden ser muy largos, lo que produce problemas como el acoplamiento, el ruido electromagnético. El uso de enlaces ópticos resuelve estos problemas, igualando el ancho de banda de los enlaces cortos y largos hasta una distancia mucho mayor que cuando se utiliza cable de cobre.
- **Restricciones físicas:** A medida que aumenta el número de componentes, también aumenta el número de cables necesarios para interconectarlos. El empaquetado de estos componentes suele requerir el cumplimiento de ciertas restricciones físicas, como el control de la temperatura de funcionamiento, la limitación de la longitud del cableado y la limitación del espacio.

  La complejidad de la conexión está limitada por la máxima densidad de cables posible y por el número máximo de pines. La velocidad a la que puede funcionar

una máquina está limitada por la longitud de los cables, y la mayor parte de la energía consumida por el sistema se utiliza para conducir los cables.

- **Fiabilidad y reparabilidad:** Estas redes son capaces de enviar mensajes por caminos alternativos cuando se detectan algunos fallos. Además, los nodos también pueden fallar o ser retirados de la red. Por ello, las redes de interconexión suelen requerir algún algoritmo de reconfiguración para la reconfiguración automática de la red cuando un nodo se enciende o se apaga.
- **Cargas de trabajo previstas:** Si se conoce de antemano el tipo de aplicaciones que se ejecutarán en el ordenador, esa información puede utilizarse para la optimización de algunos parámetros de diseño. Cuando no es posible obtener información sobre las cargas de trabajo previstas, el diseño de la red debe ser robusto; es decir, los parámetros de diseño deben seleccionarse de forma que el rendimiento sea bueno en un amplio rango de condiciones de tráfico.
- **Restricciones de coste:** Es evidente que la ”mejor” red puede ser demasiado cara. Las decisiones de diseño suelen ser un compromiso entre el coste y otros factores de diseño.

En la figura 2.1 podemos ver seis terminales conectados a una red. Si uno de esos terminales quiere comunicarse con otro de los terminales, envía el mensaje del terminal origen con los datos a través de la red y ésta entrega el mensaje al terminal destino.

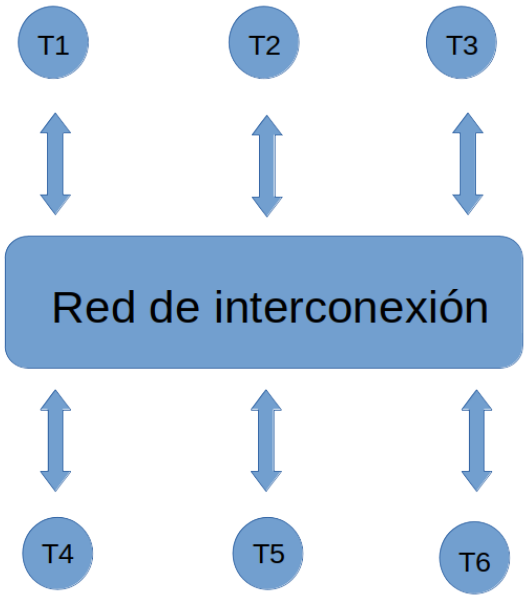


Figura 2.1: Red de Interconexión.

2.1.1. Componentes de una red de interconexión

El diseño de una red depende, entre otras cosas, del diseño del switch y del modo en que estos están conectados entre sí mediante los enlaces, que son los encargados de retransmitir los paquetes. Dependiendo del grado del switch, sus mecanismos de enrutamiento interno y su almacenamiento interno, pueden admitir o no un determinado tipo de topología y algoritmos de enrutamiento. Las redes de interconexión están compuestas por los siguientes tres componentes:

- **Enlaces:** Cable de una o varias fibras ópticas o hilos eléctricos con un conector en cada extremo conectado a un conmutador o puerto de interfaz de red. A través de él se transmite una señal analógica desde un extremo y se recibe en el otro para obtener el flujo de información digital original.
- **Switches:** Los switches (Figura 2.2) se componen de un conjunto de puertos de entrada y salida (normalmente el número de ambos puertos es el mismo), un crossbar interno que conecta toda la entrada con toda la salida, un buffer interno y una lógica de control. La misión de esta última es efectuar la conexión de entrada-salida en todo momento. Su función es dirigir los paquetes basados en la dirección de destino, localizada en la cabecera de los mismos, transmitiéndolos por el puerto de salida que corresponda.

  - **Buffers:** Son buffers FIFO para el almacenamiento de lo mensajes que transitan por la red. Para cada canal, tanto de entrada como de salida, hay un buffer asociado.
  - **Switch interno:** Es el responsable de la conexión de los buffers de entrada con los buffers de salida. Se utilizan redes crossbar.
  - **Unidad de encaminamiento y arbitraje:** Este componente selecciona el enlace de salida para un mensaje entrante. Si varios mensajes solicitan simultáneamente el mismo enlace de salida, uno de ellos lo utilizará mientras que para los demás aparecerá como ocupado, por lo que permanecerán en el buffer de entrada. Los mensajes que están a la espera de que se libere el enlace, se encaminarán de nuevo después de que éste se libere.
  - **Controlador de enlace:** El flujo de mensajes a través del canal físico entre switches adyacentes se realiza mediante el controlador de enlace. Los controladores de enlace a ambos lados de un canal se coordinan para transferir unidades de control de flujo.
  - **Puertos:** Interfaz entre el canal físico externo con el canal del switch interno.

- **Interfaces de red:** Se comportan de forma muy diferente a los nodos de conmutación y pueden estar conectadas a través de enlaces especiales. La interfaz de red formatea los paquetes y construye la información de enrutamiento y control. Puede tener un buffer de entrada y de salida, en comparación con un conmutador. Puede realizar una comprobación de errores de extremo a extremo y un control

de flujo. Por lo tanto, su coste depende de la complejidad del procesamiento, la capacidad de almacenamiento y el número de puertos.

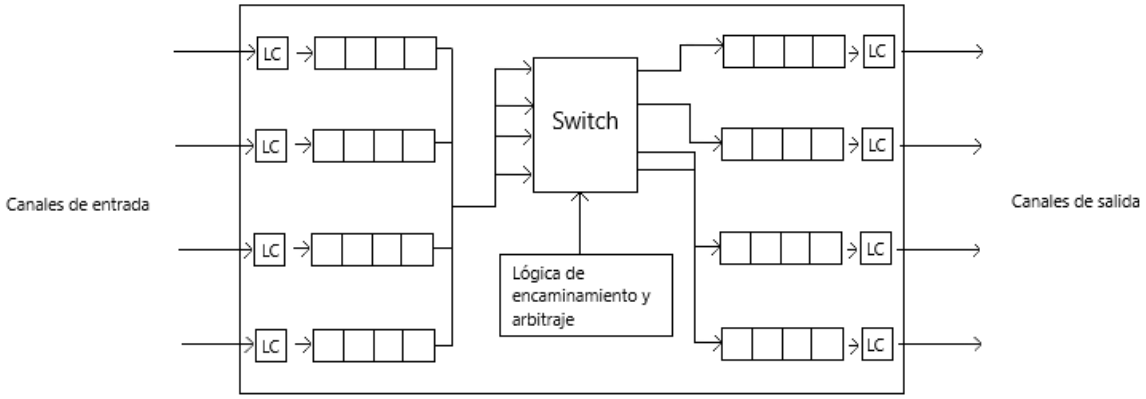


Figura 2.2: Arquitectura interna del switch.

2.1.2. Topología

Las redes de interconexión están compuestas por elementos de comunicación. La topología de la red se refiere a la disposición estática de los enlaces y los nodos de una red de interconexión: los caminos por los que viajan los paquetes. Ésta puede representarse mediante un grafo G(N,C), donde N es el conjunto de los nodos de la red, y C:NxN es el conjuntos de enlaces que conecta un nodo o switch con sus vecinos.

Para comprobar que una topología es eficiente, deben evaluarse diferentes parámetros [Top,]:

- **Ancho de bisección:** Se refiere al mínimo número de enlaces que hay que eliminar para dividir la red en dos partes iguales. Un ancho de bisección alto puede reducir el tiempo de comunicación cuando el movimiento de los datos es crítico, ya que la información puede viajar por caminos alternativos y evitar así o reducir la congestión en ciertos nodos. Además, esto también hace que el sistema sea más tolerante ante fallos, ya que un nodo defectuoso no hace inoperable al sistema.
- **Grado:** Se llama grado de un nodo al número de enlaces que tiene con otros nodos. Es conveniente que en una red el grado sea igual para todos los nodos, siendo en ese caso *regular*. Además, el grado está relacionado directamente con el coste, ya que indica el número de puertos que necesita.
- **Distancia:** Número mínimo de nodos que hay que atravesar para viajar de un nodo origen a un nodo destino. La distancia crea latencia y reducir la distancia entre los distintos nodos es la única forma de poder minimizarla.

- **Distancia media:** Representa la media de las distancias de cada par de nodos de la red.
- **Diámetro:** El diámetro es, para cualquier par de nodos, el camino más largo de todos los caminos posibles de la red, es decir, el mayor número de enlaces que hay que recorrer en la red para comunicar dos de los nodos que están en ella. Un menor diámetro indica mayor habilidad de comunicación en la red.
- **Longitud de los enlaces:** Determina la velocidad a la que la red puede operar y la potencia disipada en ella.
- **Regularidad:** Una red es regular si todos los nodos tienen el mismo grado. Es conveniente de cara a la modularidad y escalabilidad.
- **Simetría:** Se refiere a que desde cualquier nodo la vista de la red presenta el mismo aspecto. Esto implica que una red simétrica tiene que ser regular.
- **Conectividad:** Mínimo número de enlaces o nodos que hay que eliminar para obtener dos gráficos independientes. Cuanto mayor sea el número de enlaces que hay que eliminar para poder dividir la red en dos, mayor es la tolerancia a fallos.
- **Escalabilidad:** Facilidad con la que la red puede expandirse manteniendo sus prestaciones sin aumentar su coste en gran medida.

Una vez hemos diferenciado los diferentes parámetros que hay que tener en cuenta para comprobar la eficiencia de una red vamos a diferenciar también los diferentes tipos de redes que hay teniendo en cuenta la forma en que se comunican los diferentes nodos.

2.1.2.1. Redes de conexión de medio compartido

La estructura de este tipo de red de interconexión es la menos compleja ya que el medio de transmisión es compartido por todos los dispositivos que se comunican. Solo un dispositivo puede utilizar la red a la vez. Cada dispositivo conectado a la red dispone de circuitos solicitantes y controladores y receptores para gestionar el paso de direcciones y datos.

Una característica única de un medio compartido es su capacidad para soportar la difusión atómica, en la que todos los dispositivos del medio pueden supervisar las actividades de la red y recibir la información transmitida en el medio compartido. Debido al limitado ancho de banda de la red, un único medio compartido puede soportar un número limitado de dispositivos antes de que el propio medio se convierta en un cuello de botella, esto también restringe su uso en los multiprocesadores [Jose Duato, 2003].

2.1.2.1.1. Redes de área local

Las redes LAN de alta velocidad (Figura 2.3a) pueden utilizarse como la columna vertebral de la red para interconectar ordenadores y proporcionar un entorno informático integrado, paralelo y distribuido. Este tipo de redes son utilizadas principalmente para construir redes de ordenadores que abarcan distancias físicas no superiores a unos

pocos kilómetros. La topología de la red es un bus o un anillo. Tenemos tres posibles clases de LAN:

- **Bus de contención:** En la Figura 2.3b podemos ver un bus de contención donde todos los dispositivos compiten por el acceso exclusivo al bus. Debido a la naturaleza de difusión del bus, todos los dispositivos pueden monitorizar el estado del bus (inactivo, ocupado, colisión). El estado de colisión indica que dos o más dispositivos están tratando de utilizar el bus al mismo tiempo, lo que provoca que sus datos colisionen.
- **Token Bus:** El inconveniente del bus de contención es que al tener naturaleza no determinista, no es adecuado para soportar aplicaciones en tiempo real. Esto se soluciona mediante un token que pasa entre los dispositivos de la red, dando lugar a una topología conocida como token bus (Figura 2.3c). El propietario del token en un momento dado, es el único que tiene derecho de acceso al bus y, una vez completada la transmisión, el token pasa al siguiente dispositivo seleccionado dependiendo del algoritmo de selección utilizado.
- **Token Ring:** La idea del token ring (Figura 2.3d) es una extensión natural del token bus, ya que el paso del token forma una estructura en anillo.

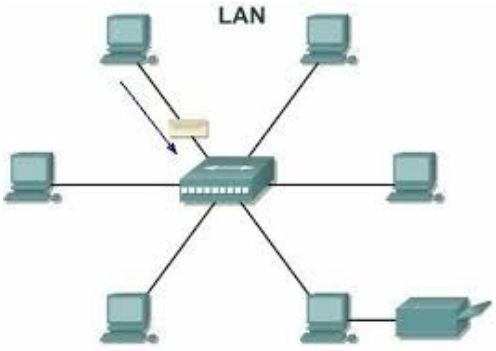
2.1.2.2. Redes de conexión directas o estáticas

Una red directa está formada por un conjunto de nodos que están conectados mediante enlaces punto a punto a un pequeño subconjunto de nodos. Un componente común de estos nodos es un encaminador o router, que se encarga de la comunicación de mensajes entre los nodos y tiene conexiones directas al router de sus vecinos.

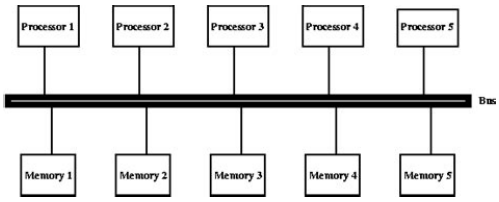
Entre las propiedades de las redes directas tenemos las especificadas en la sección 2.1.2 y, además, la propiedad ortogonal, es decir que todos los nodos y enlaces que componen la red pueden disponerse en n dimensiones, de manera que cada enlace está situado exactamente en una dimensión. Las topologías ortogonales pueden clasificarse además en estrictamente ortogonales y débilmente ortogonales. En una topología estrictamente ortogonal, cada nodo tiene al menos un enlace cruzando cada dimensión, por lo que la distancia entre dos nodos es la suma de las dimensiones atravesadas. En una topología débilmente ortogonal, algunos nodos podrían no tener ningún enlace en alguna dimensión, por lo que no es posible atravesar todas las dimensiones desde cada uno de los nodos [Jose Duato, 2003].

Dentro de las redes de conexión directas podemos diferenciar entre distintos tipos dependiendo de la forma en la que se los nodos se distribuyan y se conecten entre sí:

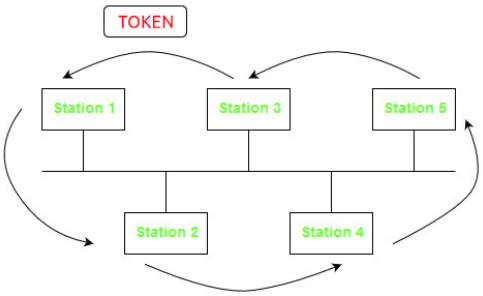
- **Estrella:** En las redes Estrella (Figura 2.4a) existe un nodo central que se conecta con todos los demás nodos de la red.
- **Anillo:** Al igual que la topología estrella, las redes en anillo, cómo la de la figura 2.4b, son una de las formas de conexión más simples, pues se conectan los nodos situándose todos en una fila, por lo que cada nodo tiene dos nodos vecinos a los que se conecta formando una circunferencia de nodos conectados.



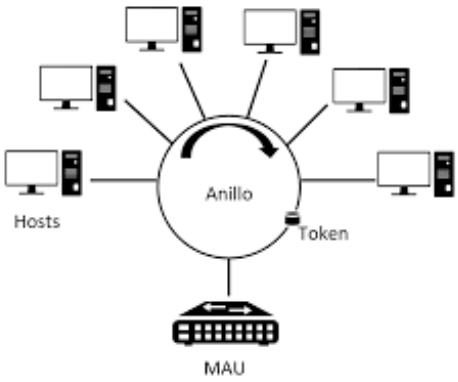
(a) Red de área local.



(b) Topología bus de contención.



(c) Topología token bus.

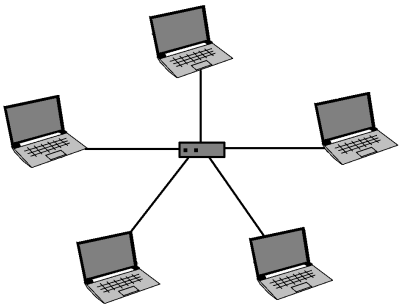


(d) Topología token ring.

Figura 2.3: Redes de área local.

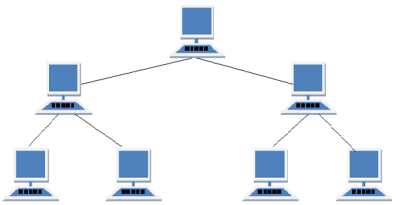
- **Árbol:** Las topologías de árbol cómo la de la figura 2.4c, tienen un nodo raíz conectado a un cierto número de nodos descendentes. Cada uno de estos nodos está a su vez conectado a un conjunto disjunto de nodos descendientes en los cuales un nodo sin descendientes se denomina nodo hoja.
- **Malla:** Una malla, cómo la de la figura 2.4d, es una topología de red directa estrictamente ortogonal que posee n dimensiones en la que cada una de ellas hay k nodos, por ello el número de nodos de la red es $k * n$ y cada uno de estos vértices está conectado a todos aquellos que difieren en una coordenada de él en cualquiera de las dimensiones.
- **Toro:** Las redes en toro, como la que podemos ver en la figura 2.4e, son mallas, es decir, son también topologías estrictamente ortogonales, en las cuales sus filas y columnas tienen conexiones en anillo, lo que contribuye a disminuir su diámetros.
- **Hipercubo:** Las redes con topología hipercubo, cómo la de la figura 2.4f, son redes estrictamente ortogonales que están formadas por 2^n nodos, que forman los vértices de los cuadrados para crear una conexión entre redes. Esta conexión se realiza conectando los nodos que solo difieren en un bit en su representación

binaria (como vemos en la figura 2.4f).

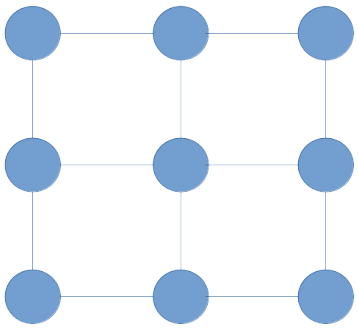


(a) Topología estrella.

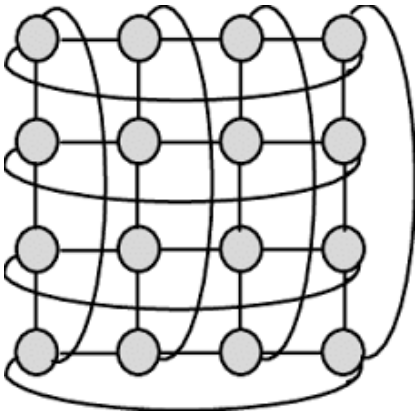
(b) Topología anillo.



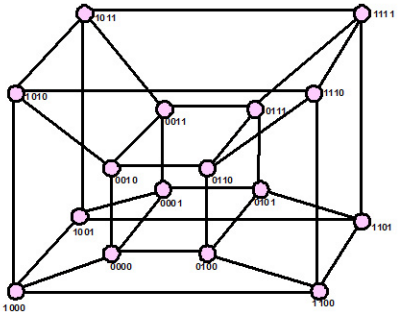
(c) Topología árbol.



(d) Topología malla 2D.



(e) Topología toro.



(f) Topología hipercubo de 4 dimensiones.

Figura 2.4: Redes de conexión directas.

2.1.2.3. Redes de conexión indirectas o dinámicas

En este tipo de redes, en lugar de proporcionar una conexión directa entre nodos, la comunicación entre cualquier par de nodos tiene que pasar por algunos conmutadores intermedios o switches. Cada nodo tiene un adaptador de red que se conecta a un

conmutador de red. Además, cada switch tiene un conjunto de puertos de los cuales cada uno consiste en un enlace de entrada y otro de salida [Jose Duato, 2003].

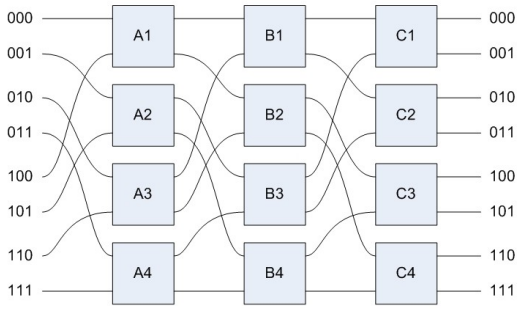
Al igual que para la redes de interconexión directas, tenemos distintos tipos de topologías pero en este caso, dependiendo de cómo los switches están interconectados por canales:

- **Redes multietapa:** Las redes multietapa conectan dispositivos de entrada a dispositivos de salida a través de un número de estaciones de switches, donde cada switch es una red crossbar. Las redes multietapa son buenas para construir ordenadores paralelos con cientos de procesadores. Entre las redes multietapa tenemos, entre otros, los siguientes tipos:

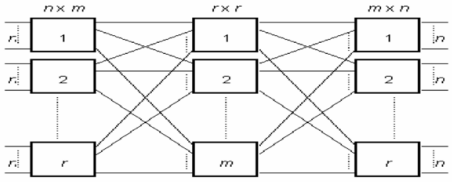
  - **Redes Omega:** Las redes Omega (Figura 2.5a) están formadas por $log_2 N$ etapas siendo N el número de entradas de la red y $N/2$ el número de switches de cada una de ellas con 2 entradas y 2 salidas cada uno. Cada etapa se interconecta con la siguiente siguiendo el patrón perfect shuffle[1].
  - **Redes CLOS:** Las redes CLOS (Figura 2.5b) son el ejemplo más conocido de redes no bloqueantes cuyo coste es prohibitivo para grandes tamaños de red. Estas redes constan de tres etapas construidas con conmutadores crossbar pero de diferentes tamaños en cada etapa. Estas redes están formadas por tres etapas: la primera de ellas tiene r conmutadores de n entradas y m salidas, la segunda etapa tiene m conmutadores de r entradas y salidas y la tercera etapa tiene r conmutadores de m entradas y n salidas.
  - **Redes de línea base:** Las redes de línea base (Figura 2.5c) están formadas por conmutadores 2x2 que se pueden generar recursivamente.
  - **Red de Benes:**Las redes de Benes (Figura 2.5d) son redes reorganizables que requieren menos etapas o switches más simples que las no bloqueantes.

- **Redes Crossbar:** Estas redes permiten a cualquier elemento del sistema conectarse a cualquier otro y así, muchos elementos pueden comunicarse simultáneamente entre sí sin disputas (Figura 2.6).

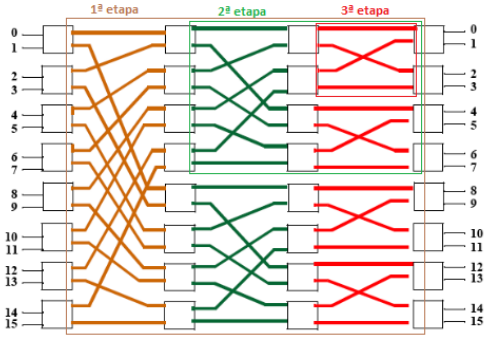
[1]Perfect Shuffle: consiste en, si el número de entradas es potencia de dos, desplazar el número en binario en una posición a la izquierda y con ello obtendremos a donde va conectado con la siguiente etapa.



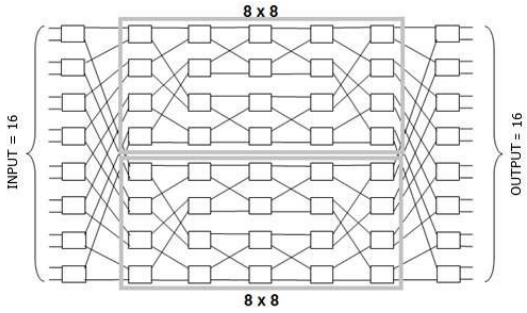
(a) Topología Omega.



(b) Topología CLOS.



(c) Topología de línea base.



(d) Topología de Benes.

Figura 2.5: Redes de conexión indirectas.

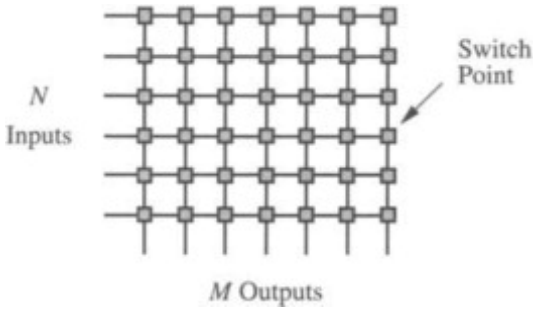


Figura 2.6: Redes de conexión Crossbar.

2.1.2.4. Redes de conexión híbridas

Estas redes combinan mecanismos de las redes de medio compartido y de las redes directas e indirectas. Con respecto a las redes de medio compartido, la redes de conexión híbridas incrementan el ancho de banda y, con respecto a las redes directas e indirectas, reducen la distancia entre nodos [Jose Duato, 2003].

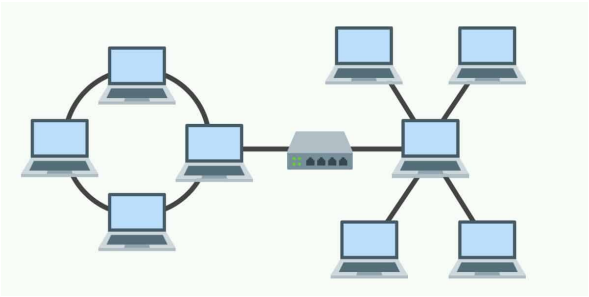
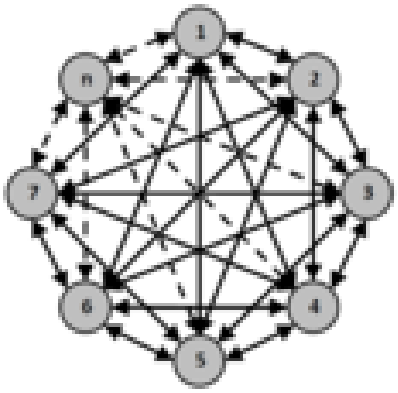


Figura 2.7: Redes de conexión híbridas.

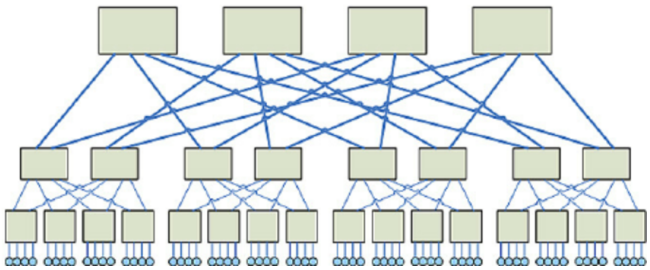
2.1.2.5. Redes de conexión jerárquicas

Con este tipo de redes se busca otro enfoque para aumentar el ancho de banda de la red. Las redes jerárquicas se caracterizan por agrupar los nodos en grupos. Cada uno de esos grupos se comunica directamente entre sí a través de un switch. Y cada uno de estos switches está directamente conectado al resto de switches (Figura 2.8a) o están interconectados mediante otro switch (Figura 2.8b).

La red jerárquica puede ampliar el área de la red y manejar más dispositivos, pero ya no es una simple red de medio compartido. Normalmente, se dispone de un mayor ancho de banda en el bus global. De lo contrario, puede convertirse en un cuello de botella. Esto puede conseguirse utilizando una tecnología más rápida [Jose Duato, 2003].



(a) Red jerárquica con todos los switches conectados.



(b) Red jerárquica con switches interconectados.

Figura 2.8: Redes de conexión jerárquicas

2.1.3. Encaminamiento

El algoritmo de encaminamiento determina cuál de los posibles caminos desde el origen hasta el destino se utiliza como ruta y cómo se calcula la ruta que sigue cada paquete en particular. El proceso para seleccionar la mejor ruta para nuestros paquetes lo realizará un protocolo de encaminamiento. El protocolo de encaminamiento que se utilizará en la red no está previamente estipulado sino que, dependiendo de diversos factores como la topología, la aplicación, etc, se elegirá el que más se adecúe a nuestro escenario. El principal objetivo del algoritmo de encaminamiento es seleccionar el camino que suponga un menor retardo para cada paquete, intentando elegir rutas que eviten en la medida de lo posible una mayor sobrecarga.

Los algoritmos de encaminamiento deben cumplir las propiedades de **simplicidad** y **rapidez** (tomar decisiones correctas en un tiempo limitado), **robustez** (capacidad de reconocer cambios de topología debido a fallos), **conectividad** (debe existir al menos un camino válido para cada par de nodos de la red), **optimalidad** (manejar la relación entre minimizar el retardo medio de los paquetes y maximizar la utilización total de la red) y **maximización de los enlaces** (garantizando la igualdad en el acceso a estos) [Jose Duato, 2003].

Como vemos en la figura 2.9 podemos clasificar los algoritmos de encaminamiento en función de diferentes criterios [Jose Duato, 2003]:

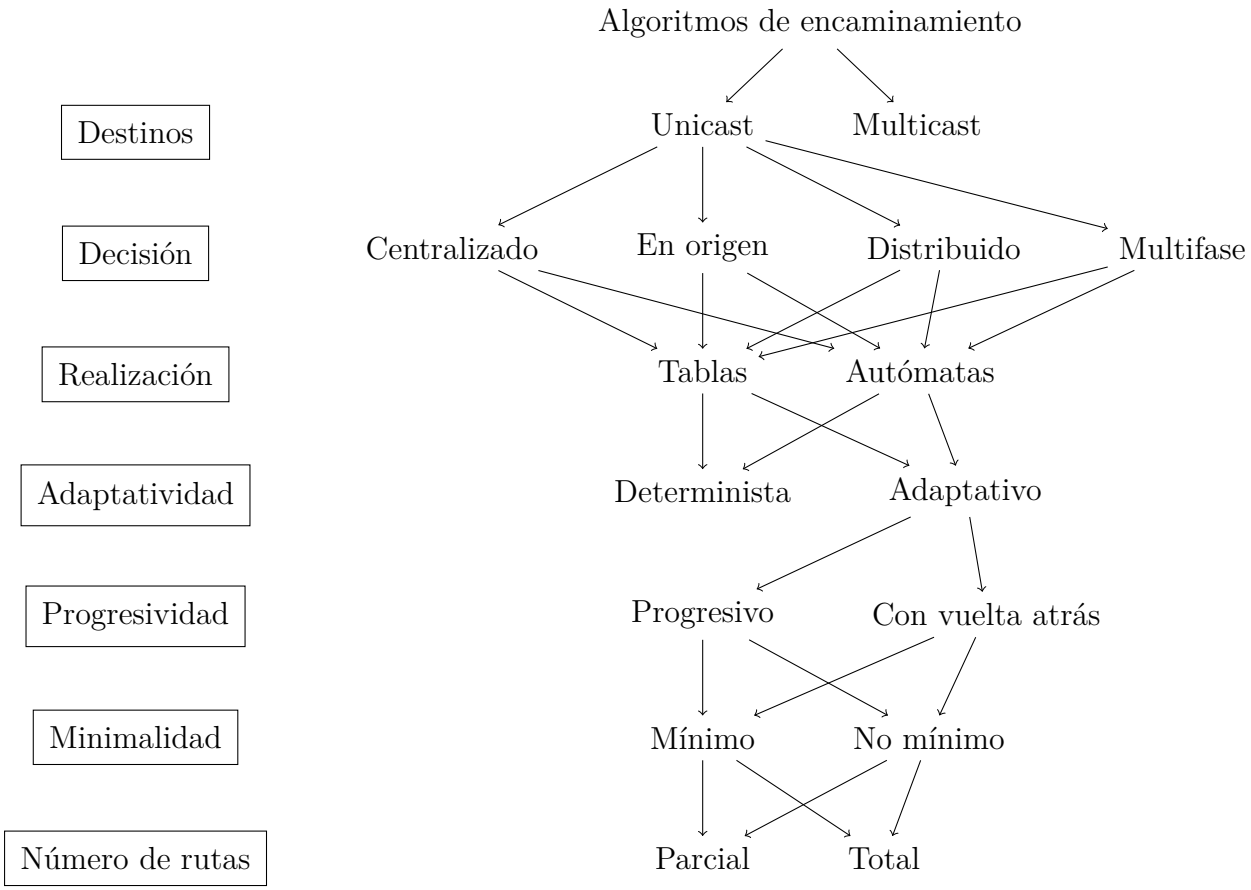


Figura 2.9: Clasificación de algoritmos de encaminamiento [Jose Duato, 2003].

2.1.3.1. Problemas relacionados con el encaminamiento

Existen fenómenos indeseables que pueden aparecer en la red y que deben ser evitados ya que provocan un degradación significativa en las prestaciones. Estos problemas aparecen debido a que los recursos disponibles son finitos y por la utilización de una técnica de encaminamiento ineficiente e inadecuada.

- **Deadlock:** Un deadlock o interbloqueo se produce cuando algunos paquetes no pueden avanzar hacia su destino porque los buffers solicitados por ellos están llenos.
- **Livelock:** Un livelock provoca la presencia indefinida de un paquete en la red que no es capaz de alcanzar su destino en ningún momento.
- **Starvation:** Un estado de starvation o inanición se produce cuando, para un mensaje, los recursos que solicita son siempre asignados a otros mensajes que también los solicitan, quedando ese mensaje a la espera de dichos recursos.

2.1.4. Técnicas de control de flujo

Los dispositivos contienen buffers que permiten almacenar datos evitando así, en cierta medida, la pérdida de paquetes. Debido a la presencia de estos, es necesario un control de flujo de los datos capaz de garantizar la capacidad de almacenamiento. Mediante el uso de un control de flujo se notifica al emisor la disponibilidad de espacio dentro del receptor de los datos. Las políticas de control de flujo más utilizadas son el control de flujo basado en créditos y el control de flujo basado en marcas o *Stop & Go* [Jose Duato, 2003].

2.1.4.1. Pérdida de paquetes

Los buffers de los dispositivos tienen una capacidad limitada para el almacenamiento de los paquetes que le llegan, los cuáles deberán ser procesados y esto conlleva un tiempo. Mientras los mensajes que se encuentran en el buffer son procesados, pueden ir llegando más mensajes a ese buffer hasta alcanzar la capacidad mínima del buffer. Cuando esto ocurre, la solución más fácil es que el dispositivo receptor descarte los paquetes que no es capaz de procesar.

2.1.4.2. Basado en créditos

En esta técnica cada conmutador tiene varios créditos para el envío de paquetes, que dependerá de la cantidad de espacio en el buffer del nodo vecino. En cada transmisión, el emisor consume un crédito hasta que haya consumido todos, en cuyo caso el emisor dejará de enviar paquetes. Una vez se haya liberado espacio en el receptor, es decir, se haya procesado algún paquete, se envían nuevos créditos al emisor, es decir, se incrementa el número de créditos de los que el emisor dispone y puede continuar con el envío. El problema es que se genera un mensaje de control para cada paquete, lo que puede ayudar a saturar la red.

2.1.4.3. Basado en marcas (*Stop & Go*)

Antes de empezar con la transmisión, se establecen dos límites, uno inferior y otro superior, para controlar la ocupación del buffer con respecto a su capacidad. El receptor detecta que su buffer de entrada está en el límite de su almacenamiento (límite superior), el cuál ha sido establecido previamente, y envía una señal (*Stop*) al nodo emisor para que detenga el envío de paquetes. Una vez que la capacidad del buffer de entrada del receptor es suficiente para recibir nuevos paquetes (límite inferior), se envía una nueva señal (*Go*) al emisor, indicando que puede continuar con el envío de los paquetes.

2.2. Protocolo de transporte Remote Direct Memory Access

Con el rápido crecimiento del procesamiento de datos, la computación y la comunicación en la arquitectura de la computación de alto rendimiento (HPC), el protocolo TCP/IP tradicional puede suponer un problema para la computación de alto rendimiento debido a que muchas aplicaciones necesitan una alta productividad y una baja latencia. Este protocolo realiza una copia de los datos que las aplicaciones quieren enviar desde su espacio de usuario en los buffers internos de los nodos, lo que puede conllevar a provocar cuellos de botella en la red, lo que añade latencia y consume un uso significativo de la CPU. Esto se debe a que la tecnología TCP/IP tradicional debe pasar a través del sistema operativo y otras capas de software en el proceso de procesamiento de paquetes de datos, lo que requiere una gran cantidad de recursos del servidor y ancho de banda del bus de memoria. Por lo tanto, es esencial una red de alto rendimiento que pueda cumplir los requisitos de las aplicaciones HPC.

Direct Memory Access (DMA) es un protocolo que permite a un dispositivo acceder directamente a la memoria del host sin la intervención de la CPU, y RDMA (Remote DMA) es un protocolo que utiliza DMA y que representa la capacidad de acceder a la memoria de una máquina remota sin interrumpir el procesamiento de la CPU de ese sistema. Es decir, RDMA permite eliminar las operaciones de copia de datos y reducir las latencias al permitir que un ordenador coloque directamente la información en la memoria de otro con una demanda mínima de ancho de banda del bus de memoria y de sobrecarga de procesamiento de la CPU, al tiempo que se conserva la protección de la memoria. Esto permite la realización de sistemas de alto rendimiento así como comunicaciones de baja latencia lo cual es muy importante en sistemas MPP.

En una transferencia IP típica, cuando una aplicación de un host envía datos a otro, sucede lo que se puede ver en la figura 2.10. Por la parte del host receptor ocurre lo que se indica a continuación [Redhat, 2022]:

- El núcleo debe recibir los datos.
- El núcleo debe determinar que los datos pertenecen a la aplicación.
- El núcleo despierta la aplicación.
- El kernel espera a que la aplicación realice una llamada al sistema en el kernel.
- La aplicación copia los datos del espacio de memoria interna del propio kernel en el buffer proporcionado por la aplicación.

Este proceso implica que la mayor parte del tráfico de red se copie a través de la memoria principal del sistema, si el adaptador de host utiliza el acceso directo a la memoria (DMA). Además, el ordenador ejecuta una serie de cambios de contexto para cambiar entre el contexto del núcleo y el de la aplicación lo que puede ralentizar otras tareas.

Con la técnica de RDMA se evita la intervención del kernel en la comunicación, lo que reduce la sobrecarga de la CPU. RDMA permite al adaptador del host saber

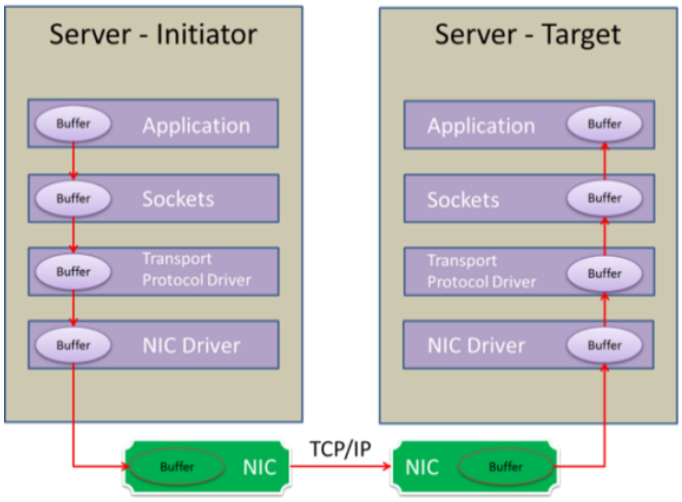


Figura 2.10: Esquema de funcionamiento del protocolo TCP/IP.

cuándo llega un paquete desde la red, qué aplicación debe recibirlo y en qué lugar del espacio de memoria de la aplicación debe almacenarse el paquete.

El protocolo RDMA promete una computación y un transporte de datos más eficiente y escalable dentro del centro de datos al reducir la carga de los procesadores y la memoria, con respecto al protocolo TCP/IP.

Por ello, podemos encontrar RDMA en industrias que necesiten baja latencia, como Computadores de altas prestaciones (HPC), servicios financieros o web 2.0; gran ancho de banda, como en HPC, aplicaciones médicas, sistemas de almacenamiento y de copia de seguridad o cloud computing, etc.

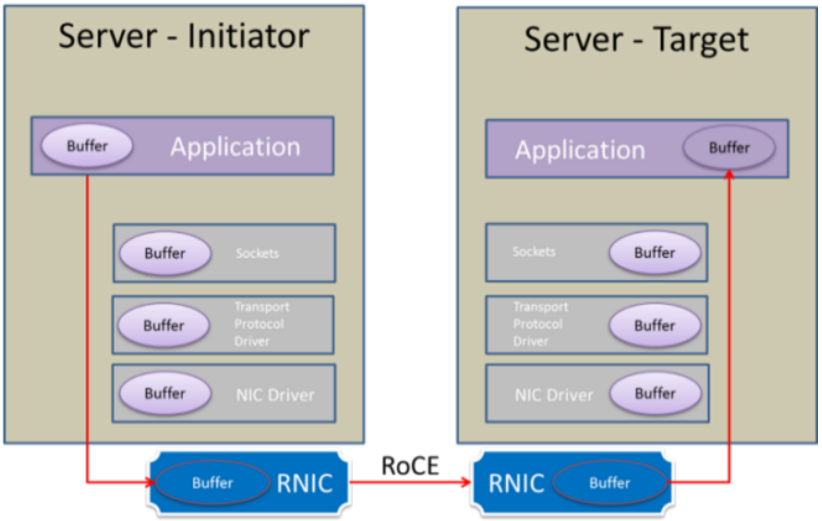


Figura 2.11: Esquema de funcionamiento del protocolo RDMA.

2.2.1. Ventajas de RDMA

RDMA presenta muchas ventajas con respecto a los protocolos de transporte correspondientes:

- **Zero-copy:** Las aplicaciones pueden realizar la transferencia de datos sin que intervenga la pila de software de la red y los datos se envían directamente a los buffers sin que se copien entre las capas de red.
- **Kernel bypass:** Las aplicaciones pueden realizar la transferencia de datos directamente desde el espacio de usuario, sin necesidad de realizar conmutaciones de contexto.
- **Sin intervención de la CPU:** Las aplicaciones pueden acceder a la memoria remota sin consumir CPU en la máquina remota. La memoria de la máquina remota se leerá sin ninguna intervención de elementos remotos (proceso/procesador). De esta forma, las cachés de la CPU remota no se llenarán con el contenido de la memoria accedida.
- **Transacciones basadas en mensajes:** Los datos se manejan como mensajes discretos y no como un flujo, lo que elimina la necesidad de que la aplicación separe el flujo en diferentes mensajes.
- **Soporte de entrada de dispersión/recopilación:** RDMA soporta de forma nativa el trabajo con múltiples entradas de dispersión/recopilación, es decir, la lectura de múltiples buffers de memoria y su envío como un flujo, o la obtención de un flujo y su escritura en múltiples buffers de memoria.

2.2.2. Tecnologías de red compatibles con RDMA

En este apartado vamos a describir las tecnologías más punteras de la actualidad que incorporan este protocolo como son Infiniband y Ethernet. Dentro de Ethernet nos vamos a fijar en dos protocolos como son RoCE e iWARP.

- **Ethernet:** Ethernet es la tecnología tradicional para conectar dispositivos en una red de área local o una red de área amplia (WAN) por cable, lo que les permite comunicarse entre sí a través de un protocolo. Dentro de esta tecnología, tenemos dos protocolos basados en RDMA:
  - **RoCE:** RoCE es el protocolo de red que permite el uso de RDMA sobre una red Ethernet, definiendo su funcionamiento en dicho entorno. Accede a la memoria del conmutador o del servidor remoto sin consumir ciclos de CPU en el servidor remoto, lo que permite un uso completo del ancho de banda disponible y una mayor escalabilidad. Sus cabeceras de red inferiores son cabeceras Ethernet y sus cabeceras de red superiores (incluyendo los datos) son cabeceras InfiniBand. Esto permite utilizar RDMA sobre una infraestructura Ethernet estándar (switches). Sólo las NICs deben ser especiales y soportar RoCE. Dentro de RoCE tenemos dos posibles versiones de éste:

  - **RoCEv1:** Es un protocolo de capa de enlace Ethernet que permite la comunicación entre dos host del mismo dominio de difusión Ethernet. Se aplican límites de longitud de trama del protocolo Ethernet (MTU = 1500 bytes).
  - **RoCEv2:** Supera la limitación de la versión 1, que se limita a un único dominio de difusión. Al cambiar la encapsulación de los paquetes para incluir cabeceras IP y UDP, RoCEv2 puede utilizarse ahora en redes L2 y L3. Esto permite el enrutamiento de Capa 3, que lleva RDMA a la red con múltiples subredes para una gran escalabilidad.

- **iWARP:** Aprovecha el protocolo TCP o el SCTP para la transmisión de los datos. Fue desarrollado para permitir que las aplicaciones de un servidor lean o escriban directamente en las aplicaciones que se ejecutan en otro servidor sin que el sistema operativo de ninguno de los dos lo soporte. Al igual que ocurre con RoCE, las NICs deben ser especiales y soportar iWARP de lo contrario, todas las pilas iWARP pueden ser implementadas en SW y perder la mayoría de las ventajas de rendimiento RDMA.

- **Infiniband:** Infiniband es una tecnología de red interconexión punto a punto. Sus características como el zero-copy y que RDMA es el protocolo estándar para la conexiones de red Infiniband de alta velocidad ayudan a reducir la sobrecarga del procesador al transferir directamente los datos de la memoria del emisor a la del receptor sin involucrar a los procesadores del host. Se necesitan tarjetas de red (NICs) y switches que admitan esta tecnología. En esta tecnología se utilizan dispositivos como los HCAs como dispositivo host. La arquitectura de una red Infiniband tiene los siguientes elementos clave:

  - **HCA:** Estructura de red de interconexión basada en la tecnología Infiniband. Proporciona a un ordenador un puerto de conexión con otros dispositivos Infiniband.
  - **Switches:** Un switch reenvía los paquetes entre los puertos del adaptador basándose en la dirección de destino en la cabecera de enrutamiento local (LRH) de cada paquete.
  - **Routers:** Al igual que los switches, los routers reenvían los paquetes desde el origen hasta el destino. Sin embargo, los routers reenvían los paquetes basándose en la cabecera de enrutamiento global (GRH).
  - **Subnet manager:** El SM configura los componentes locales de la subred. Proporciona LIDs a todos los nodos de la subred, y también proporciona información de reenvío a los conmutadores de la subred.

2.2.3. Comparaciones de RoCE con el resto de tecnologías y protocolos

Ya que el objetivo de este proyecto es implementar RDMA sobre una red Ethernet, vamos a comparar que nos ofrece RoCE frente a la tecnología InfiniBand y el protocolo iWARP y viceversa.

2.2.3.1. RoCE vs Infiniband

Como hemos dicho en la sección 2.2.2 RoCE es una tecnología que ofrece muchas de las ventajas de RDMA, como una menor latencia o una mejor utilización de la CPU, pero utilizando una red Ethernet en lugar de adaptadores y switches InfiniBand cómo los que se usan en una red Infiniband [DeCusatis, 2022]. Entonces, a parte de estas, ¿cuáles son las diferencias entre RoCE e InfiniBand? [roc, 2012]

- **Velocidad del cable y roadmap[2] del ancho de banda:** El roadmap de Ethernet está diseñado para satisfacer las necesidades de aplicaciones muy distintas cómo las redes domésticas y las LAN corporativas, pasando por la interconexión de centros de datos. Por otra parte, el roadmap de InfiniBand tiene el único objetivo de ser la interconexión de centros de datos de mayor rendimiento posible. Por lo tanto, sería mejor desplegar InfiniBand si nos preocupa el ancho de banda.
- **Curva de adopción:** Históricamente, la nueva generación Ethernet se ha desplegado primero como tecnología de red troncal (de conmutador a conmutador), y, finalmente, se ha ido extendiendo a los nodos finales. Por otro lado, los adaptadores de servidor InfiniBand suelen estar disponibles coincidiendo con el siguiente aumento de velocidad anunciado, lo que le permite a los servidores conectarse a una red InfiniBand con los últimos grados de velocidad de forma inmediata. Lo que ocurre aquí es que RoCE permite conservar la estructura conmutada[3] de Ethernet familiar, pero al precio de una curva de adopción más lenta en comparación con InfiniBand.
- **Gestión de la red:** La especificación de InfiniBand describe una arquitectura de gestión completa basado en un esquema de gestión centralizada de la red que contrasta mucho las estructuras conmutados de Ethernet, que generalmente se gestionan de manera autónoma. La arquitectura de gestión centralizada de InfiniBand le da al gestor de la red conmutada una amplia visión de toda la red de capa 2 lo que le permite ofrecer características avanzadas de red como el soporte de topología de capa 2 arbitrarias, partición, QoS, etc. Estás características permiten evitar las limitaciones del protocolo spanning tree tradicional.
- **Control de flujo a nivel de enlace frente a DCB:** RDMA funciona mejor cuando los cables subyacentes implementan la llamada red sin pérdidas que es aquel en el que los paquetes en el cable no se pierden de forma rutinaria. La Ethernet tradicional se considera una red con pérdidas ya que pierde paquetes con frecuencia y es preciso de la capa de transporte TCP para detectar estos paquetes perdidos y ajustarlos. Sin embargo, InfiniBand utiliza el control de flujo a nivel de enlace, que es una técnica que garantiza que los paquetes no se pierdan en la red en en caso de errores graves, lo que permite obtener todo el ancho de banda por el que se ha pagado. Pero, cuando se utiliza RoCE, se puede conseguir

[2]Roadmap: planificación del desarrollo de un software con los objetivos a corto y largo plazo, y posiblemente incluyendo unos plazos aproximados de consecución de cada uno de estos objetivos.

[3]Estructura conmutada: La estructura conmutada o estructura de conmutación es una topología de red en la que los nodos de red se interconectan a través de uno o más conmutadores de red

lo mismo es necesario implementar la última versión de Ethernet, conocida cómo Data Center Bridging (DCB), el cuál implementa el Priority Flow Control (PFC) el cuál permite el control de flujo sobre una red. PFC es el encargado de eliminar la pérdida de paquetes debido a la congestión en un vínculo de red [PFC, 2022]. Sin embargo, configurar una red Ethernet DCB puede ser más complejo que configurar una red InfiniBand.

2.2.3.2. RoCE vs iWARP

RoCe proporciona una latencia en el adaptador del orden de 1-5 microsegundos, pero requiere una red Ethernet sin pérdidas para lograr un funcionamiento de baja latencia. Esto significa que los switches Ethernet integrados en la red deben ser compatibles con los mecanismos de los DCB y el PFC para que se mantenga el tráfico sin pérdidas. Por tanto, es posible que tenga que reconfigurarse para utilizar RoCE, siendo esta un proceso complejo y la escalabilidad puedes ser muy limitada en un contexto empresarial moderno.

Por otra parte, iWARP tiene la ventaja de funcionar en TCP/IP, aprovechando el protocolo TCP. Proporciona una latencia mayor que RoCE pero iWARP proporciona una mayor facilidad de uso y una mayor escalabilidad, ya que se configura en la infraestructura existente y puede escalar fácilmente tanto entre bastidores como en largas distancias entre centros de datos [roc, 2019].

2.3. Entorno de simulación OMNeT++

OMNeT++ es una biblioteca y un marco de simulación C++ extensible, modular y basado en componentes, desarrollado principalmente para construir simuladores de red. El concepto *red* incluye tanto redes de comunicación cableadas e inalámbricas, redes en chip, redes de colas, etc. Este entorno de simulación ofrece un IDE basado en Eclipse, un entorno gráfico de ejecución con las opciones necesarias para la manipulación de éstas redes [Omn, 2019b].

Esta herramienta ofrece una arquitectura de componentes para los modelos que se desarrollen. Estos componentes que, desde ahora, llamaremos módulos, nos permiten construir módulos compuestos a partir de módulos simples y a partir de esos módulos compuestos, construir nuestra red, como vemos en la figura 2.12. Es decir, a partir de módulos simples podemos llegar a construir una red a partir de la cuál podemos simular tráfico. Todo ello debe realizarse mediante la programación de cada uno de esos módulos simples y cómo interaccionan o se conectan entre ellos. Todo esto se realiza a partir de diferentes tipos de archivos que OMNeT++ nos proporciona para realizar dicha labor.

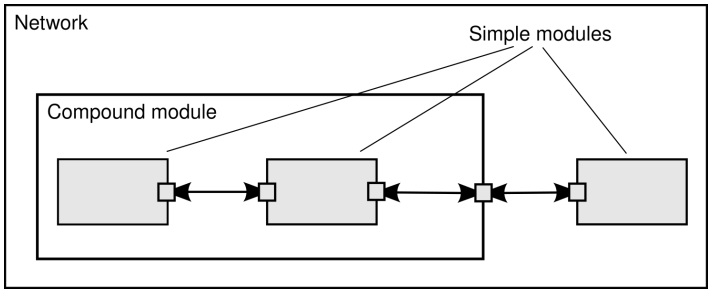


Figura 2.12: Modulos OMNeT++.

2.3.1. Tipos de archivos

Como hemos dicho, para construir una red es necesario el buen uso de los diferentes tipos de archivos, ya que cada uno de ellos tiene una función específica dentro del entorno y si uno de esos archivos no está bien configurado la simulación no será correcta. En las siguientes secciones se detallarán las características de cada uno de los tres tipos de archivos que posee OMNeT. Estos tipos de archivos son los siguientes:

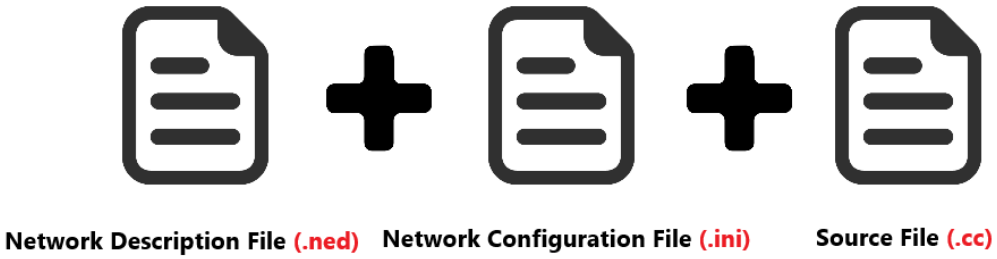


Figura 2.13: Estructura de archivos de OMNeT++.

2.3.1.1. Archivos de Descripción de Red (.ned)

Los Archivos de Descripción de Red (Network Description File) son los archivos donde el usuario describe la estructura del modelo de simulación. Permite al usuario declarar módulos simples y conectarlos para formar módulos compuestos. Y, mediante mediante la conexión de estos módulos compuestos es posible formar las redes que, al estar formado por la conexión de módulos, puede decirse que también son módulos compuestos. La unión de estos componentes se hace mediante el uso de canales, que son otro tipo de componente. [Omn, 2019a] Entre las características de este lenguaje tenemos algunas funcionalidades como las siguientes:

- **Jerárquico:** OMNeT++ permite la construcción de módulos compuestos mediante el uso de jerarquías, simplificando el uso de cada uno de los módulos simples que forman ese módulo compuesto que por sí solo sería demasiado complejo. Un ejemplo de esto es el módulo StandardHost que representa un host TCP/IP y que veremos más adelante.
- **Basado en Componentes:** Todos los módulos, tanto simples como compuestos, desarrollados son reutilizables, por lo que se pueden usar en diferentes tipos de redes y, sobre todo, permite la existencia de librerías como INET que veremos más adelante.
- **Interfaces:** Las interfaces permiten que sea en el momento de la configuración cuando se indique el tipo de módulo a utilizar dentro de todos aquellos que implementan dicha interfaz mediante un parámetro. Es decir, que un módulo compuesto puede tener un submódulo que sea una interfaz y a dicha interfaz se le pueden asignar diferentes comportamientos dependiendo del módulo que se indique por un parámetro, siendo ese módulo uno de los que implementen esa interfaz.
- **Herencia:** Esto permite que a un módulo se le puedan añadir nuevos parámetros, gates y, en los módulos compuestos, nuevos submódulos y conexiones.
- **Tipos internos:** Los tipos de canales y módulos utilizados localmente por un módulo compuesto pueden definirse dentro de ese módulo compuesto, con el fin de reducir la contaminación del espacio de direcciones.

A continuación podemos ver un ejemplo de este tipo de archivo. En el código de la figura 2.14 podemos ver que estamos tratando de crear una red que va a estar formada por módulos. Vemos que estamos generando 2 hosts, y un switch ethernet y otro módulo que es necesario para calcular correctamente el camino que seguirán los paquetes. También vemos que ambos hosts estarán directamente conectados al switch generado mediante un cable que será gigabitEthernet que se conectará por un extremo al interfaz del host y por el otro extremo al interfaz que le corresponda del switch. Como consecuencia de este código obtenemos la red de la figura 2.15, modelada como hemos explicado anteriormente. Por último, en la figura 2.16 se muestra como el host es un módulo compuesto que está formado por otros submódulos para representar cada una de las capas y el funcionamiento del protocolo TCP/IP. Cada uno de los módulos de esas capas tiene su funcionamiento necesario para la correcta transmisión de los paquetes a su destino y se conectan a la siguiente capa utilizando un bus.

2.3.1.2. Archivos Fuente (.cc)

Los Archivos Fuente (Source File) son archivos comunes de C++ cuyo cometido es programar el funcionamiento de cada uno de los módulos que se van a utilizar. Estos archivos programan las funciones declaradas en su respectivo archivo .h que es donde se declararán las cabeceras de las funciones, variables globales y constantes.

Además, también tenemos otros tipos de archivos especiales (.msg) que serán los archivos que contienen las definiciones de los mensajes. A partir de estos archivos,

```
network Modelo2Hosts
{
  submodules:
  H_0: StandardHost {}
  SW_1_1: EthernetSwitch {
    gates:
      ethg[2];
  }
  H_1: StandardHost {}

  configurator: Ipv4NetworkConfigurator {}

  connections:
    SW_1_1.ethg[0] <--> Eth40G <--> H_0.ethg++;
    SW_1_1.ethg[1] <--> Eth40G <--> H_1.ethg++;
}
```

Figura 2.14: Código en lenguaje ned.

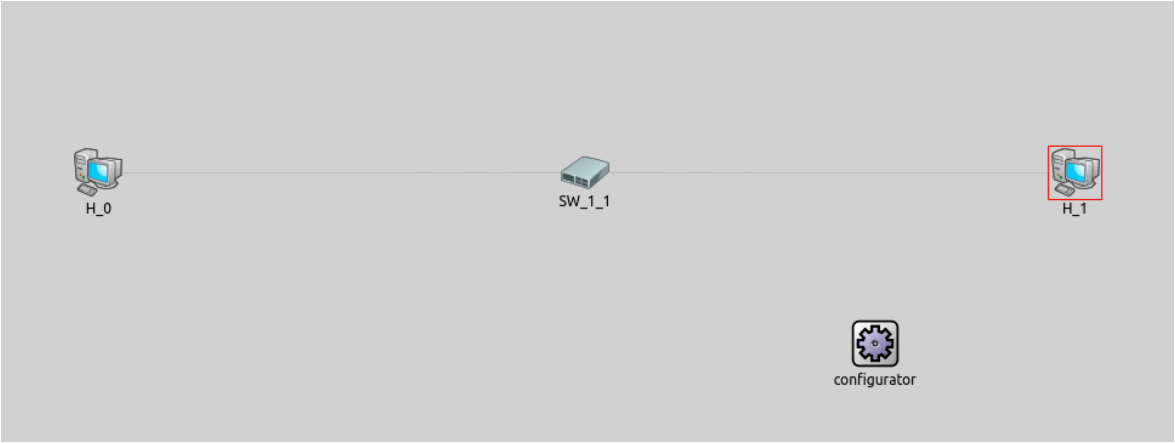


Figura 2.15: Ejemplo red generada en archivo .ned.

deberemos tener tambíen dos nuevos archivos con el mismo nombre y acabados en _m.h y _m.cc, por ejemplo, Mensaje.msg, Mensaje_m.h y Mensaje_m.cc. Para cada una de las variables introducidas en el archivo .msg tendremos esa misma variable en el archivo _m.h y sus correspondiente cabeceras de los métodos getter y setter cuyo funcionamiento se implementará en el archivo _m.cc.

Para especificar el módulo al que se le va a asignar dicho comportamiento es necesario incluir al principio del archivo .cc lo mostrado en el código de la figura 2.17. Lo incluido entre paréntesis hace referencia al archivo .h del archivo .cc en el que donde se encuentra dicho código, que debe tener el mismo nombre que el módulo simple al que se le asigna el comportamiento programado en el archivo .cc.

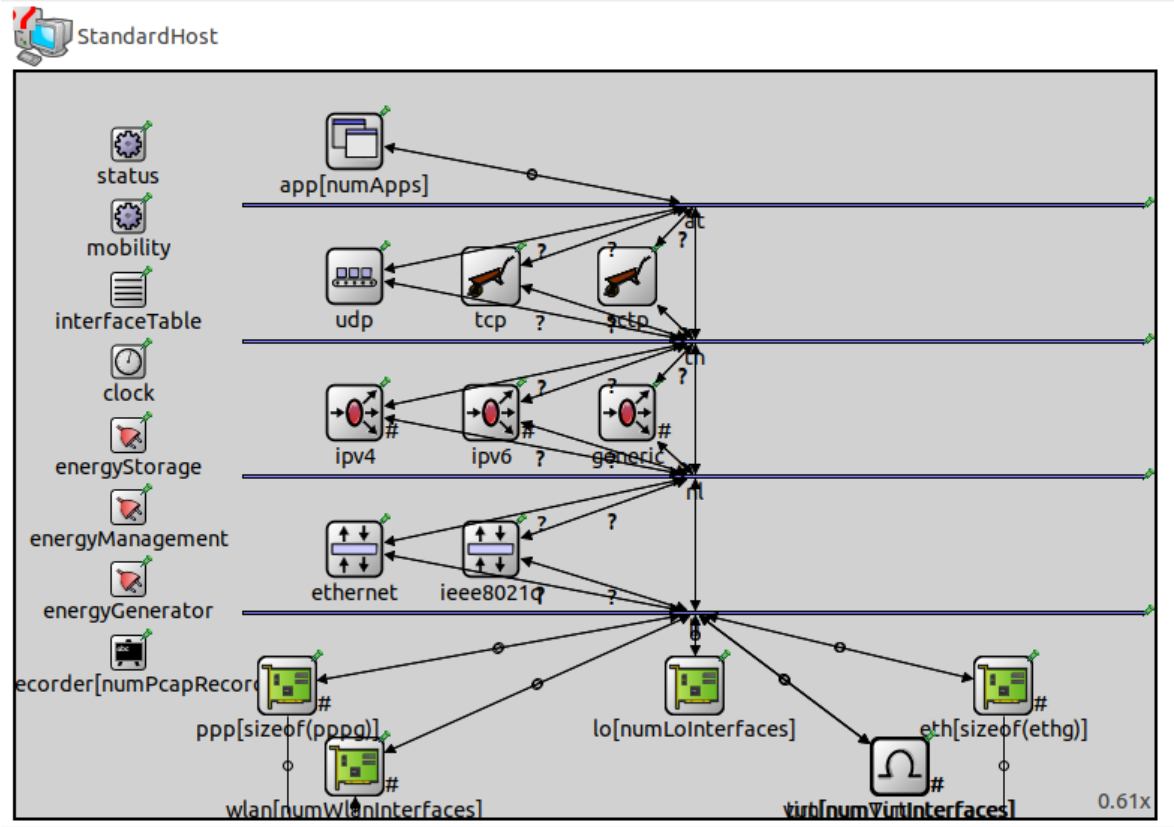


Figura 2.16: Host simulado de la red.

```
1    Define_Module("Modulo");
```

Figura 2.17: Elemento necesario para especificar a que módulo se le asigna el comportamiento programado en el archivo .cc actual.

2.3.1.3. Archivos de Configuración de Red (.ini)

Los Archivos de Configuración de Red (Network Configuration File) son los archivos que tienen como finalidad iniciar la simulación dependiendo de los parámetros que introduzcamos en él. En este tipo de archivos hay varios elementos clave, como por ejemplo, la red en la que se va a realizar la simulación, que hace referencia a un archivo .ned. Otro tipo de elemento importante es el tiempo que va a durar la simulación, es decir, cuando el tiempo de simulación alcance el valor indicado en este parámetro, la simulación finalizará automáticamente. Si no se establece ningún valor, la simulación no acabará por si sola y deberá ser detenida manualmente. También se permitirá tener diferentes configuraciones dentro de la misma simulación mediante el uso de corchetes ([]). Para establecer parámetros que serán generales para todas las configuraciones que se introduzcan en este archivo, deberemos ponerlos debajo de [General]. Dentro de cada

uno de los archivos de configuración podremos establecer distintos parámetros para la simulación, que dependerá del tipo de elementos que compongan la red. Algunos de estos parámetros son el tipo de mensaje que se enviará, el tamaño del mensaje y con qué intervalo de tiempo lo hará, el emisor y el receptor de dicho mensaje, etc.

En la figura 2.18 podemos ver un archivo de configuración en el que se especifica una única simulación, ya que solo tenemos la configuración [General]. Está simulación se realizará sobre la red que se muestra en la figura 2.15. Se ha configurado que la simulación se detendrá cuando el tiempo de simulación alcance los 2 segundos, que emplearemos tráfico UDP, que se enviarán continuamente mensajes desde el host H_0 al host H_1 y que se enviarán paquetes de tamaño 1000 bytes cada 0.01 segundos del tiempo de simulación.

```
1       network = Modelo2Hosts
2       sim-time-limit = 2s
3       **.H*.numApps = 1
4       **.H*.app[0].typename = "UdpBasicApp"
5       **.H*.app[0].localPort = 1000
6       **.H*.app[0].destPort = 1000
7       **.H_0.app[0].destAddresses = "H_1"
8       **.H*.app[*].sendInterval = 0.01s
9       **.H*.app[*].messageLength = 1000B
```

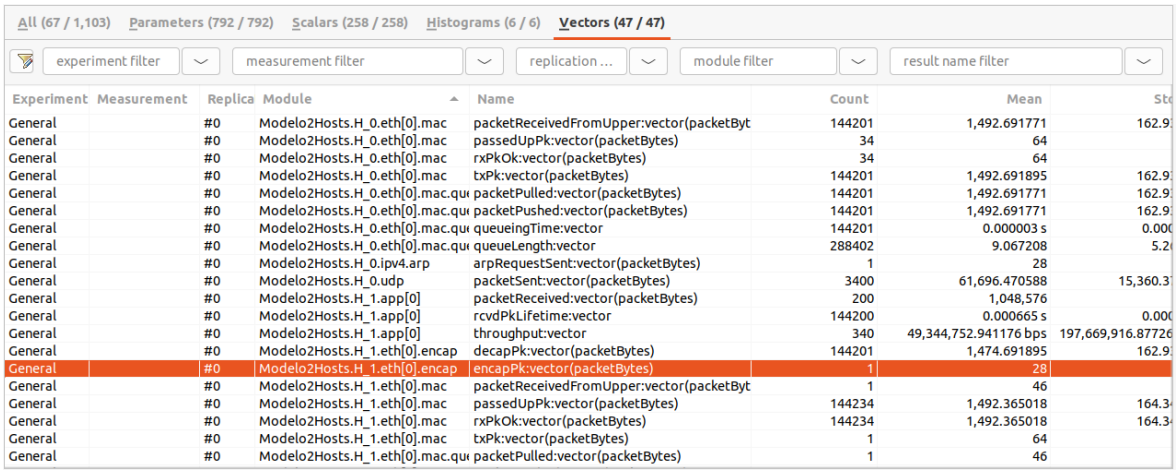
Figura 2.18: Archivo de configuración .ini.

2.3.1.4. Archivos de resultados

Cuando la simulación alcanza el valor establecido de tiempo límite de simulación, OMNeT++ por defecto genera dos tipos de archivos destinados a registrar tipos de datos distintos. Por un lado están los Archivos de Resultados Escalares (Scalar Result File (.sca)) que, como su propio nombre indica, son los encargados de almacenar los resultados cuyo valor es de tipo escalar y los Archivos de Vectores de Salida (Output Vector File (.vec)) que se encargan del tratamiento de los datos de tipo vectorial. Además, cuando usamos la interfaz gráfica de la aplicación, al pulsar dos veces para abrir uno de estos archivos, se nos permite generar un nuevo archivo con extensión .anf que agrupa los datos de estos dos tipos de archivos facilitándonos su tratamiento y permitiéndonos generar diferentes tipos de gráficos, tal y cómo se ve en la figura 2.19.

2.4. Framework de simulación INET

Dentro de OMNeT++, utilizaremos INET que es una biblioteca de modelos. Esta biblioteca proporciona protocolos, agentes y otros modelos para trabajar con redes de comunicación. Contiene modelos para la pila de Internet (TCP, UDP, IPv4, etc.),



All (67 / 1,103) Parameters (792 / 792) Scalars (258 / 258) Histograms (6 / 6) Vectors (47 / 47)

experiment filter | measurement filter | replication ... | module filter | result name filter

Experiment	Measurement	Replica	Module	Name	Count	Mean	St
General		#0	Modelo2Hosts.H_0.eth[0].mac	packetReceivedFromUpper:vector(packetByt	144201	1,492.691771	162.9
General		#0	Modelo2Hosts.H_0.eth[0].mac	passedUpPk:vector(packetBytes)	34	64	
General		#0	Modelo2Hosts.H_0.eth[0].mac	rxPkOk:vector(packetBytes)	34	64	
General		#0	Modelo2Hosts.H_0.eth[0].mac	txPk:vector(packetBytes)	144201	1,492.691895	162.9
General		#0	Modelo2Hosts.H_0.eth[0].mac.qu	packetPulled:vector(packetBytes)	144201	1,492.691771	162.9
General		#0	Modelo2Hosts.H_0.eth[0].mac.qu	packetPushed:vector(packetBytes)	144201	1,492.691771	162.9
General		#0	Modelo2Hosts.H_0.eth[0].mac.qu	queueingTime:vector	144201	0.000003 s	0.00
General		#0	Modelo2Hosts.H_0.eth[0].mac.qu	queueLength:vector	288402	9.067208	5.2
General		#0	Modelo2Hosts.H_0.ipv4.arp	arpRequestSent:vector(packetBytes)	1	28	
General		#0	Modelo2Hosts.H_0.udp	packetSent:vector(packetBytes)	3400	61,696.470588	15,360.3
General		#0	Modelo2Hosts.H_1.app[0]	packetReceived:vector(packetBytes)	200	1,048,576	
General		#0	Modelo2Hosts.H_1.app[0]	rcvdPkLifetime:vector	144200	0.000665 s	0.00
General		#0	Modelo2Hosts.H_1.app[0]	throughput:vector	340	49,344,752.941176 bps	197,669,916.87726
General		#0	Modelo2Hosts.H_1.eth[0].encap	decapPk:vector(packetBytes)	144201	1,474.691895	162.9
General		#0	Modelo2Hosts.H_1.eth[0].encap	encapPk:vector(packetBytes)	1	28	
General		#0	Modelo2Hosts.H_1.eth[0].mac	packetReceivedFromUpper:vector(packetByt	1	46	
General		#0	Modelo2Hosts.H_1.eth[0].mac	passedUpPk:vector(packetBytes)	144234	1,492.365018	164.3
General		#0	Modelo2Hosts.H_1.eth[0].mac	rxPkOk:vector(packetBytes)	144234	1,492.365018	164.3
General		#0	Modelo2Hosts.H_1.eth[0].mac	txPk:vector(packetBytes)	1	64	
General		#0	Modelo2Hosts.H_1.eth[0].mac.qu	packetPulled:vector(packetBytes)	1	46	

Figura 2.19: Archivo .anf.

protocolos de capa de enlace cableados e inalámbricos (Ethernet, IEEE 802.11, etc.), soporte para movilidad, etc [ine, 2019].

INET se basa en el concepto de módulos que se comunican entre sí mediante paso de mensajes. Los protocolos de red están representados por componentes que pueden combinarse entre sí para formar otros dispositivos como hosts, switches, routers, etc. Además, posibilita que el usuario cree nuevos componentes.

A la hora de relacionar INET con OMNeT++, podemos decir que el primero se beneficia de la infraestructura que proporciona el segundo es decir, hace uso de los servicios del núcleo y de la biblioteca de simulación y, además, permiten que los modelos puedan ser desarrollados, parametrizados, ejecutados y evaluados sus resultados desde la comodidad de OMNeT, tanto desde el IDE como de la línea de comandos.

Entre las características que proporciona INET vamos a destacar algunas de las más importantes a la hora de desarrollar nuestro trabajo:

- Capas OSI implementadas (física, capa de enlace, red, transporte, aplicación).
- Implementación de protocolos enchufables para varias capas
- Pila de red IPv4/IPv6.
- Protocolos de la capa de transporte: TCP, UDP.
- Protocolos de enrutamiento.
- Interfaces cableadas/inalámbricas (Ethernet, PPP, IEEE 802.11, etc.).

Los módulos de INET se organizan mediante una estructura de directorios similar a las capas OSI [use, 2022]:

- src/inet/applications/ - generadores de tráfico y modelos de aplicación
- src/inet/transportlayer/ - protocolos de la capa de transporte
- src/inet/networklayer/ - protocolos y accesorios de la capa de red

- src/inet/linklayer/ - protocolos y accesorios de la capa de enlace
- src/inet/physicallayer/ - modelos de la capa física
- src/inet/routing/ - protocolos de enrutamiento (Internet y ad hoc)
- src/inet/mobility/ - modelos de movilidad
- src/inet/power/ - modelos de consumo de energía
- src/inet/environment/ - modelo del entorno físico
- src/inet/node/ - modelos de nodos de red premontados
- src/inet/visualizer/ - componentes de visualización (2D y 3D)
- src/inet/common/ - componentes de utilidad diversos

INET proporciona dispositivos de red premontados a través de otros módulos a través de los cuáles podemos montar nuestro propia red de simulación mediante el lenguaje NED. De estos dispositivos vamos a destacar los que nosotros vamos a utilizar para la creación de nuestros escenarios como son [use, 2022]:

- **StandardHosts:** Contiene los protocolos de Internet más comunes como UDP, TCP, IPv4, Ethernet, etc. Admite cualquier número de aplicaciones totalmente configurables a través de los archivo INI.
- **EthernetSwitch:** Modela un switch Ethernet que contiene un relay unit y una MAC por puerto.

Como hemos dicho anteriormente, los modelos de nodos se crean a partir de módulos que se conectan a través de enlaces para formarlos. Estos módulos se dividen, entre otros, en las siguientes categorías:

- **Aplicaciones:** Modelan el protocolo de la capa de aplicación o el comportamiento del programa de aplicación o de usuario. Suelen utilizar protocolos de capa de transporte o, también de capa inferior a través de sockets.
- **Protocolos de encaminamiento:** Utilizan TCP, UDP e IPv4, y manipulan las rutas en el módulo en el módulo que representa las tablas de encaminamiento.
- **Protocolos de capa de transporte:** Están conectados a las aplicaciones y a los protocolos de capa de red. Actualmente se soportan TCP, UDP y SCTP.
- **Protocolos de capa de red:** Están conectados a los protocolos de capa de transporte y a las interfaces de red. Suelen modelarse como módulos compuestos, como por ejemplo el módulo Ipv4NetworkLayer para Ipv4 que contiene varios módulos de protocolo como Ipv4, Arp, Icmp e Icmpv6.
- **Interfaces de red:** Se representan mediante módulos compuestos que se conectan a los protocolos de capa de red y a otras interfaces de red en el caso de los cables.

- **Protocolos de capa de enlace:** Suelen modelarse como módulos sencillos aunque algunos se representan como módulos compuestos debido a la complejidad del protocolo.
- **Protocolos de capa física:** Son módulos compuestas que también forman parte de los módulos de interfaz de red.
- **Tabla de interfaz:** Mantiene el conjunto de interfaces de red en el nodo de red. Las interfaces se registran dinámicamente durante la inicialización de las interfaces de red.
- **Tablas de encaminamiento:** Mantienen la lista de rutas para el protocolo de red correspondiente. Las rutas son añadidas por los configuradores automáticos de red o por los protocolos de encaminamiento. Los protocolos de red utilizan las tablas de enrutamiento para encontrar la mejor ruta para los datagramas.

3. Metodología y desarrollo

En este capitulo vamos a explicar el camino que hemos seguido para desarrollar los objetivos que se han marcado anteriormente (Sección 1.2), es decir, para desarrollar el modelado del protocolo RDMA en un simulador de redes de interconexión como es OMNeT++. Lo primero que se ha hecho es realizar una labor de investigación para conocer el funcionamiento del protocolo RDMA. Una vez hecho esto, lo siguiente ha sido realizar la instalación del entorno de simulación OMNeT++ y, una vez instalado, incluirle el framework INET. Una vez se ha realizado todo lo anterior se ha procedido a la implementación del código necesario para poder simular el protocolo de transporte RDMA. Una vez implementado RDMA, procedemos a realizar las modificaciones necesarias, que no se haya realizado todavía, en el código existente para poder realizar las comparaciones. Una vez hecho todo lo anterior procederemos a realizar las simulaciones y obtener los resultados pertinentes y documentar los análisis realizados en el proyecto.

En los siguientes apartados vamos a desarrollar más en profundidad aquellos conceptos necesarios para entender la forma en la que se trabaja en OMNeT++.

3.1. Instalación y ejecución del entorno de simulación OMNeT++ en el clúster CELLIA

Una vez que tenemos el protocolo RDMA implementado es el momento de realizar las simulaciones necesarias para estudiar los experimentos. Para ello, lo primero que haremos es indicar en los archivos de configuración (.ini) los datos necesarios para llevar a cabo esas simulaciones. Una vez que hemos realizado esta parte, lo que haremos será llevar el entorno de simulación al clúster CELLIA para realizar en él las pruebas.

Lo primero que debemos hacer es irnos a la página de oficial de OMNeT++ y descargarnos la versión que queramos. En nuestro caso la versión utilizada es la 6.0pre11. Una vez lo tenemos descargado, extraemos los ficheros en el directorio que queramos, normalmente en nuestra carpeta $/home/ < user >$[1]. Esto lo haremos mediante el

[1]En nuestro caso <user> será vromero, como se verá posteriormente en los ejemplos.

siguiente comando:

```
$ tar -xzvf omnetpp-6.0pre11-src-linux.tgz
```

Una vez terminada la extraccion de los archivos, tendremos en la en nuestro directorio un nuevo subdirectorio llamado omnetpp-6.0pre11.

Una vez tenemos el subdirectorio en nuestro sistema, lo siguiente que haremos será añadir el directorio *bin* de la carpeta omnetpp-6.0pre11 en el path. Para ello, debemos abrir el archivo .bashrc:

```
$ vim ~/.bashrc
```

Una vez dentro, añadiremos al final del fichero la siguiente línea:

```
export PATH=$PATH:/home/vromero/omnetpp-6.0pre11/bin
```

Una vez hecho esto, para que los cambios tengan efecto, será necesario abrir y cerrar la sesión del clúster o por el contrario introducir el siguiente comando:

```
source .bashrc
```

Para realizar el siguiente paso es necesario tener el software necesario para que pueda ser instalado correctamente. Entre este software necesario se encuentran algunos paquetes de python. Para poder instalarlos es necesario instalar la herramienta pyenv que nos permite tener diferentes versiones de python coexistiendo en nuestro sistema. Para que no haya problemas a la hora de que OMNeT++ detecte los paquetes, vamos a utilizar la versión 2.7.18 de python. Para instalar los paquetes requeridos introduciremos lo siguiente en el terminal:

```
$ sudo apt-get install build-essential clang lld gdb bison flex perl
\ python3 python3-pip qt5-default libqt5opengl5-dev \ libxml2-dev
zlib1g-dev doxygen graphviz libwebkit2gtk-4.0-37
$ pip install numpy pandas matplotlib scipy seaborn posix_ipc
```

Una vez que tenemos esto hecho, lo siguiente será entrar en el directorio omnetpp-6.0pre11 y ejecutar dos comandos. El primer comando se utiliza para configurar todas las variables de entorno que OMNeT++ requiere que estén configuradas y establece esta versión de OMNeT++ como la versión activa que se iniciará cuando ejecutemos el programa y el segundo comando se encarga de detectar que el software necesario está instalado y escribe los resultados en el archivo Makefile.inc, que será leído por los Makefiles:

```
$ cd omnetpp-6.0pre11
. setenv
./configure WITH_TKENV=no WITH_QTENV=no WITH_OSG=no
```

Por último, si el comando ./configure WITH_TKENV=no WITH_QTENV=no WITH_OSG=no ha tenido éxito, solo nos queda compilar el OMNeT++ con el comando:

```
$ make
```

Una vez tenemos compilado OMNeT++ verificamos uno de los escenarios que vienen por defecto para comprobar que funciona.

Nuestro siguiente paso, será descargarnos la versión de INET que vayamos a utilizar desde la página oficial. En nuestro caso usaremos la versión 4.3.2, ya que es compatible con omnetpp-6.0pre11. Una vez descargado lo extraemos y lo introducimos en el subdirectorio samples/ de OMNeT++:

```
1    tar -xvfz inet-4.3.2-src.tgz
2    cp -r inet4.3 omnet6.0-pre11/samples/
```

En nuestro caso, como ya teníamos nuestro escenario INET implementado a través de nuestro ordenador local en el entorno OMNeT++, lo que haremos será subir el directorio al clúster e introducirlo en el directorio samples/.

Una vez hemos introducido el escenario de INET en el directorio samples/, entramos en él y dentro debemos compilar todos los archivos Makefile, para ello usaremos el siguiente comando:

```
1    make makefiles
```

Si no ha saltado ningún error, podemos comprobar que se ha compilado correctamente mediante la ejecución de uno de los escenarios de ejemplo de INET situados en el directorio examples/.

A partir de aquí, si todo ha salido correctamente ya podemos disponernos a lanzar las simulaciones para realizar los experimentos.

Para lanzar a ejecutar los experimentos cuyos escenarios, en nuestro caso, estarán situados en el directorio $HOME/omnetpp-6.0pre11/samples, ejecutaremos el siguiente comando:

```
1    opp_run -m -u Cmdenv -n $HOME:$HOME/omnetpp-6.0pre11/samples/inet/src
    :$HOME/omnetpp-6.0pre11/samples/inet/tutorials:$HOME/omnetpp-6.0pre11/
    samples/inet/showcases -l $HOME/omnetpp-6.0pre11/samples/inet/src/INET
     omnetpp.ini
```

Una vez lanzado este comando comenzará la ejecución de la simulación por línea de comandos, como podemos ver en la figura 3.1. Entre otras cosas en durante la ejecución de la simulación se nos muestra, entre otras cosas, el tiempo de la simulación, el tiempo que lleva ejecutándose la simulación, el porcentaje de ejecución de la simulación actual y el porcentaje de ejecución de todas las simulaciones lanzadas.

Figura 3.1: Ejecución mediante línea de comandos.

3.2. Implementación de RDMA y modificaciones en el código ya existente

Una vez tenemos instalado el entorno de simulación OMNeT++ y le hayamos introducido el framework INET, podemos proceder, una vez nos hayamos familiarizado con el entorno, a la implementación del protocolo RDMA.

En la figura 3.2 podemos ver cómo está estructurado el directorio inet dentro del simulador (mostrando únicamente los subdirectorios y archivos que nosotros hemos necesitado modificar o que hemos creado). De los directorios que vemos en la figura, aquellos que no contienen *RDMA* en su nombre son los que inicialmente venían ya implementados y que solo ha sido necesario modificar para que puedan trabajar de acuerdo con RDMA o bien simplemente para el cálculo de estadísticas que vamos a utilizar en nuestros experimentos como, por ejemplo, la latencia. Aquellos archivos que van dentro de un directorio que contiene *RDMA* no venían implementados y, por lo tanto, ha sido necesario crearlos y configurarlos para implementar el funcionamiento por RDMA.

Lo primero que se ha hecho para la implementación de RDMA sobre INET es crear un nuevo host que llamaremos RdmaHost que estará basado en el host con el modelo TCP/IP. Una vez tenemos el host sobre el que estará el protocolo RDMA podemos proceder a la implementación del código. Para ello iremos mostrando lo que se ha realizado en cada una de las capas.

Para la implementación de RDMA, nos hemos basado en la implementación ya existente de UDP, tanto para la realización del módulo que representa la aplicación cómo para el módulo de la capa de transporte.

3.2.1. Capa de aplicación

En la capa de aplicación, para una aplicación RDMA, se ha omitido el uso de sockets, los cuáles se utilizan en tráfico Ethernet como TCP o UDP. Una vez hemos suprimido los sockets deberemos hacer modificaciones en el código para controlar el estado del enlace y poder enviar únicamente a la capa de transporte cuando el enlace esté libre. Esto es algo que no estaba previamente programado, ya que venía únicamente implementado para que su funcionamiento se realice a través de canales ideales, los cuáles no generaban latencia y podían enviar mensajes ilimitados simultáneamente, por lo que deberemos hacerlo tanto para RDMA cómo para UDP y TCP.

Además, el tamaño máximo de un fragmento UDP, incluyendo su cabecera, es 65536 bytes. Conforme está implementado, cuándo un mensaje llega a la capa de transporte, se analiza si este supera el tamaño máximo y, en caso de que esto ocurra, simplemente manda un mensaje de error. Por ello se realizan modificaciones para fragmentar ese mensaje en fragmentos de tamaño máximo 65536 bytes.

Estás modificaciones que se han realizado corresponden a la parte del envío pero también será necesario realizar modificaciones para la recepción. Estas modificaciones corresponden a que debido a que los mensajes son fragmentados será necesario ”re-

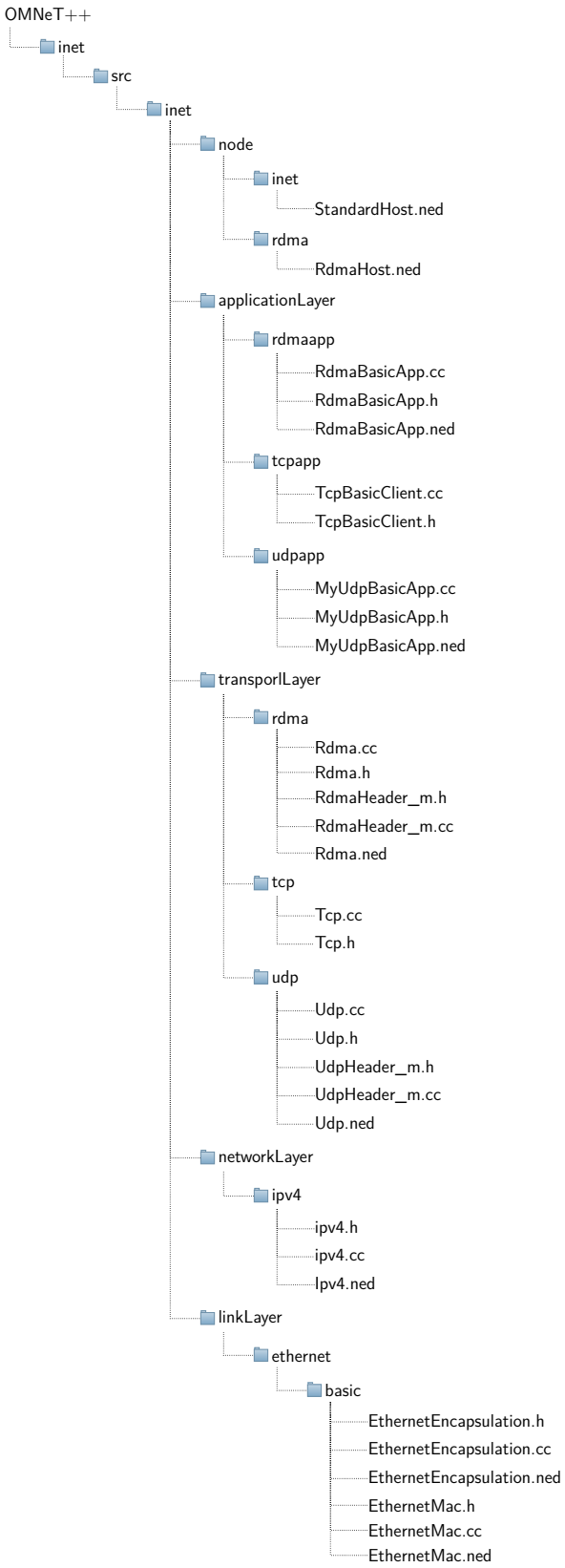


Figura 3.2: Directorios OMNeT++.

ensamblar" los fragmentos en un solo mensaje para que, una vez lleguen todos los fragmentos de un mismo mensaje, se pueda procesar dicho mensaje.

La última modificación realizada en esta capa será para el cálculo de la latencia, ya que únicamente recopilaremos la información de aquellos mensajes recibidos que no se hayan obtenido durante el tiempo de llenado de la red con intervalo de confianza del 95%.

3.2.2. Capa de transporte

En esta capa RDMA deberá procesar el mensaje que le llega de la capa de aplicación y comprobar si el mensaje que le llega es mayor que el tamaño máximo de mensaje establecido para RDMA que es de 4096 bytes. En caso de que lo supere, RDMA deberá fragmentar ese mensaje en otros de tamaño máximo 4096 bytes.

También, en esta capa será necesario realizar modificaciones para controlar el estado del enlace y poder enviar a la capa de red (cuándo estamos realizando un envío cuyo código tiene la estructura de la figura A.1) o a la capa de aplicación (cuándo estamos en una recepción cuyo código tiene la estructura de la figura A.2) únicamente cuando el canal correspondiente esté libre.

3.2.3. Capa de red

Una vez el paquete llega a la capa de red en un envío, este módulo, inicialmente, debía envíar un mensaje arp[2] y, una vez llegara el mensaje de respuesta y se supiera la dirección a la que enviar dicho mensaje, este se envía a la capa física. Posteriormente los mensajes arp no serían necesarios ya que en la configuración de la simulación se establecería el tipo de arp cómo "GlobalArp" lo que hace que la tabla de direcciones ser rellene al comienzo de la simulación.

Además, tendremos que realizar las modificaciones necesarias para controlar el estado del enlace y poder enviar a la capa física (cuándo estamos realizando un envío) o a la capa de transporte (cuándo estamos en una recepción) únicamente cuando el canal correspondiente esté libre.

3.2.4. Capa física

Una vez el mensaje llega a la capa física procedente de la capa de red tendremos que realizar modificaciones para, en el mensaje, indicar el tipo de tráfico que tenemos, ya que TCP y UDP enviarán el mensaje en la recepción a la capa de red mientras que RDMA lo enviará directamente a la capa de aplicación.

[2]ARP: es un protocolo clave en las comunicaciones TCP/IP que permite, dada un dirección IP, obtener su correspondiente dirección física. Esto se hace utilizando una tabla ARP donde se almacenan las correspondencias y, si la dirección IP requerida no se encuentra en esta tabla, este protocolo se encarga de enviar una solicitud a la red para obtener respuesta del nodo que tenga esa dirección IP requerida

Por último, al igual que en las capas anteriores, será necesario realizar modificaciones en el código para controlar el estado del enlace para enviar el mensaje sobre la red (cuándo estamos realizando un envío) o a la capa de red (cuándo estamos en una recepción) únicamente cuando el canal correspondiente esté libre.

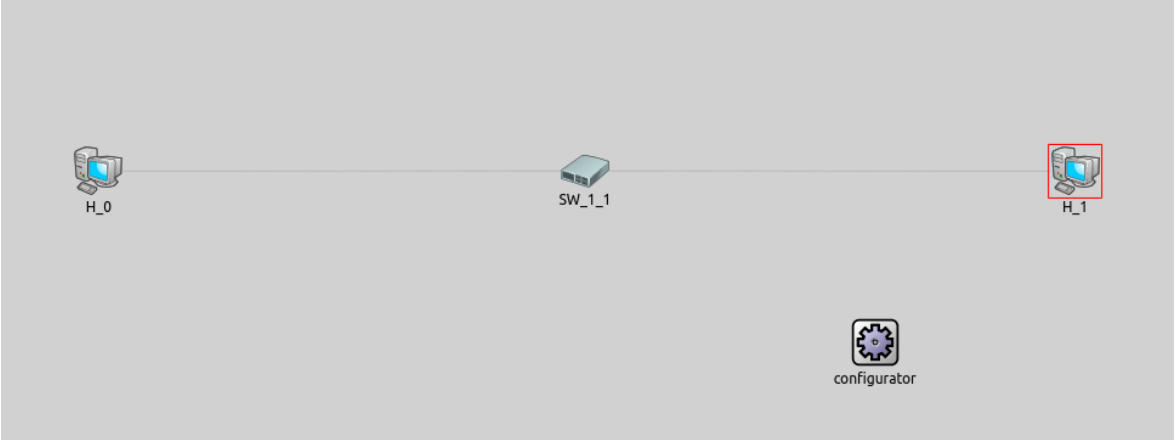
4. Experimentos y resultados

Para realizar los experimentos vamos a inyectar tres tipos de tráfico: tráfico TCP, tráfico UDP y tráfico RDMA. En primer lugar, utilizando el mismo escenario y para diferentes tamaños de mensaje, vamos a comprobar cómo afecta el tipo de tráfico que utilizamos a la latencia que vamos a obtener. También, vamos a comprobar la carga que es capaz de soportar cada uno de los tráficos usados hasta su saturación. Por último, vamos a estudiar el rendimiento que ofrecen el tráfico TCP/IP (vamos a utilizar tráfico UDP) y RDMA para comprobar cuál nos ofrece mayores beneficios en este aspecto.

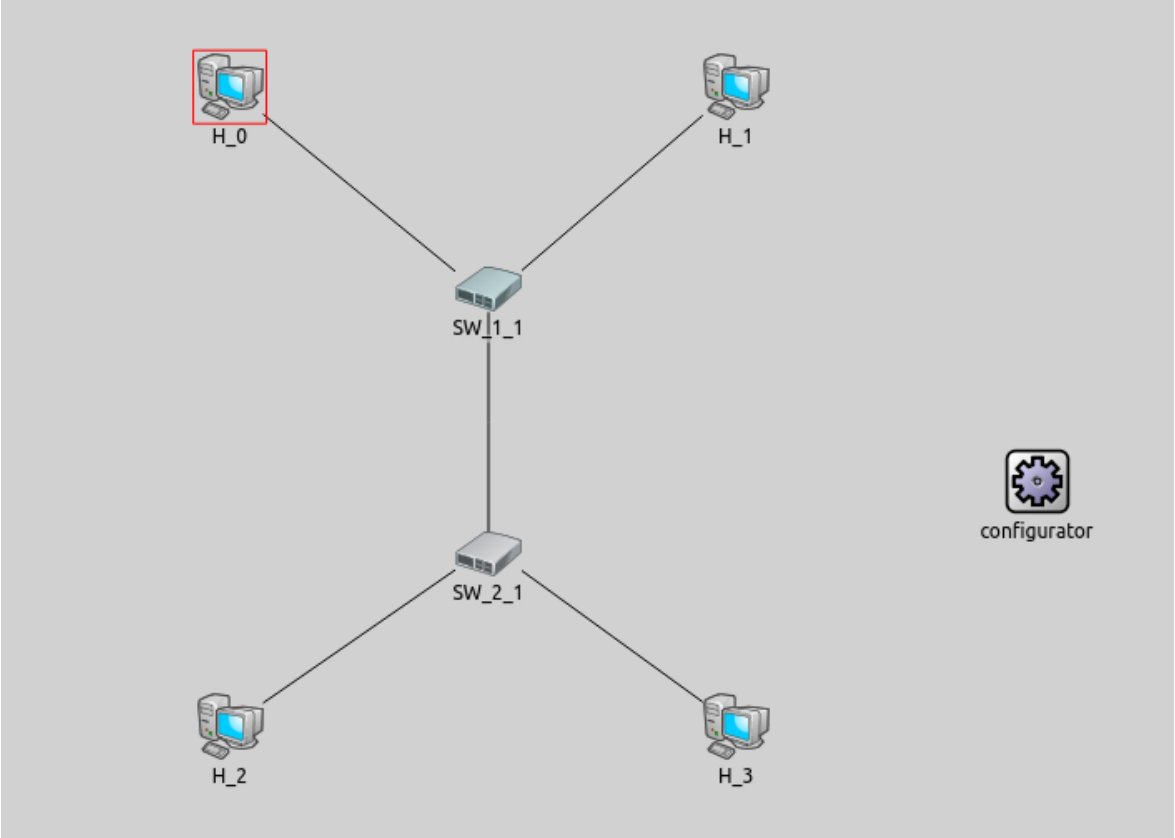
4.1. Configuración de los experimentos

Para la realización de los experimentos, como ya se ha dicho a lo largo de este trabajo, vamos a utilizar el simulador OMNeT++ para montar los escenarios de simulación con sus conexiones entre los nodos utilizando, para ello, los Archivos de Descripción de Red descritos en la sección 2.3.1.1 y vamos a ir alternando los parámetros necesarios en sus respectivos Archivos de Configuración de Red descritos en la sección 2.3.1.3. Estos escenarios constarán de 2 nodos (Figura 4.1a), 4 nodos (Figura 4.1b), 16 nodos (Figura 4.1c) y 128 nodos (Figura 4.1d). En todos los experimentos se inyectará tráfico RDMA, tráfico UDP y tráfico TCP y se variará el tamaño de los mensajes para ver como esto influye en la latencia de los mismos, salvo cuando calculemos el tráfico soportado. Además, supondremos que no utilizamos control de flujo por lo que tendremos una red con pérdida de paquetes.

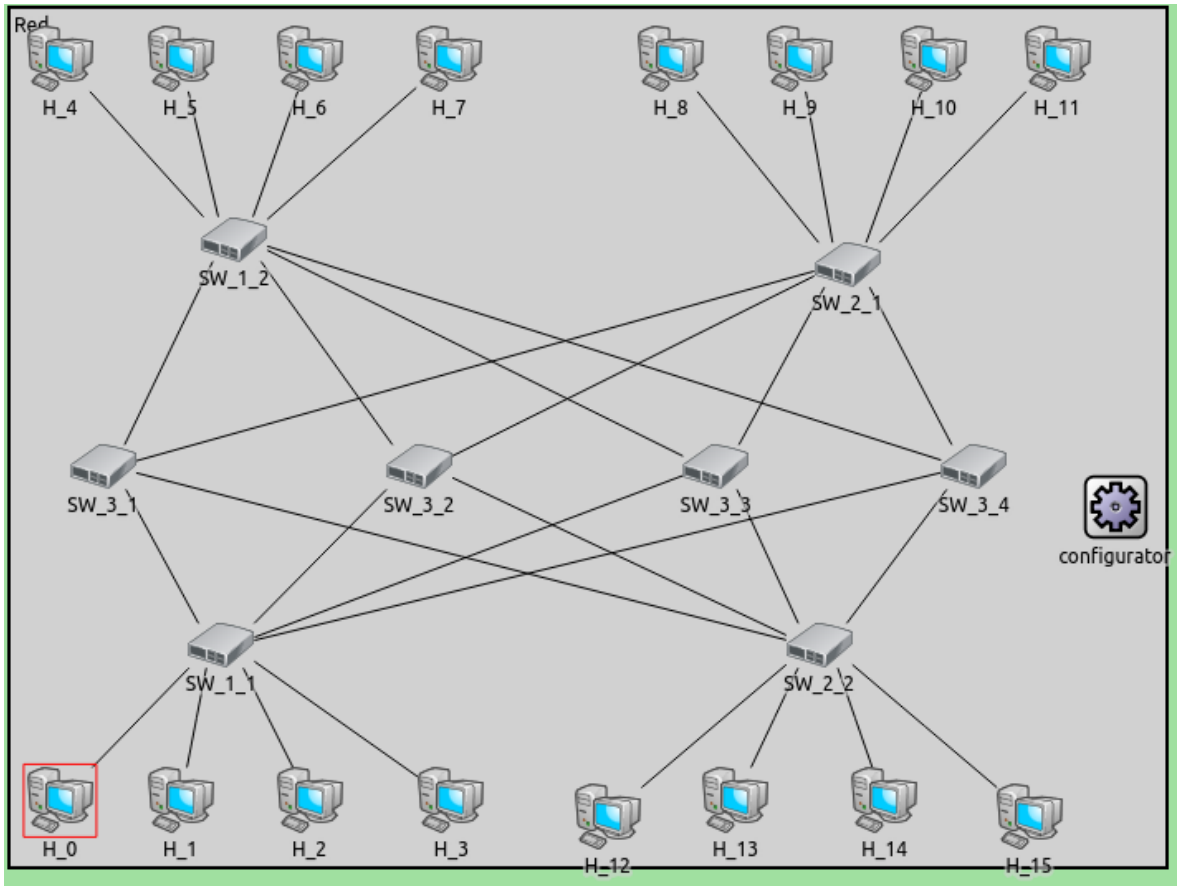
Una vez tengamos los escenarios de simulación montados, los llevaremos al clúster CELLIA, que pertenece al grupo de investigación RAAP, al cuál, previamente, le habremos instalado el simulador OMNeT++ de la manera que se ha explicado en la sección 3.1 para realizar en él las simulaciones.



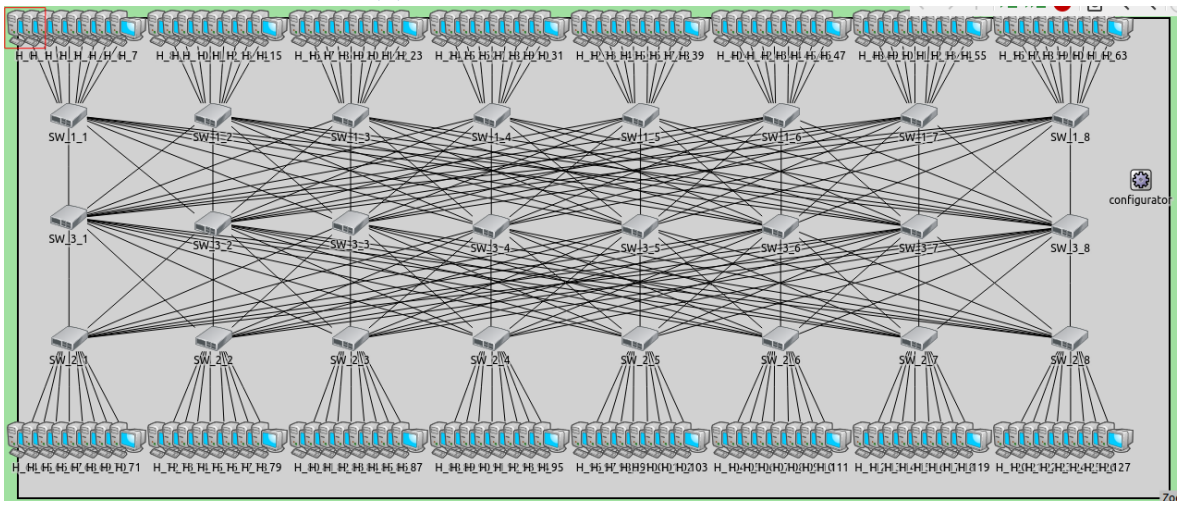
(a) Red de simulación de 2 hosts.



(b) Red de simulación de 4 hosts.



(c) Red de simulación de 16 hosts.



(d) Red de simulación de 128 hosts.

Figura 4.1: Escenarios de simulación

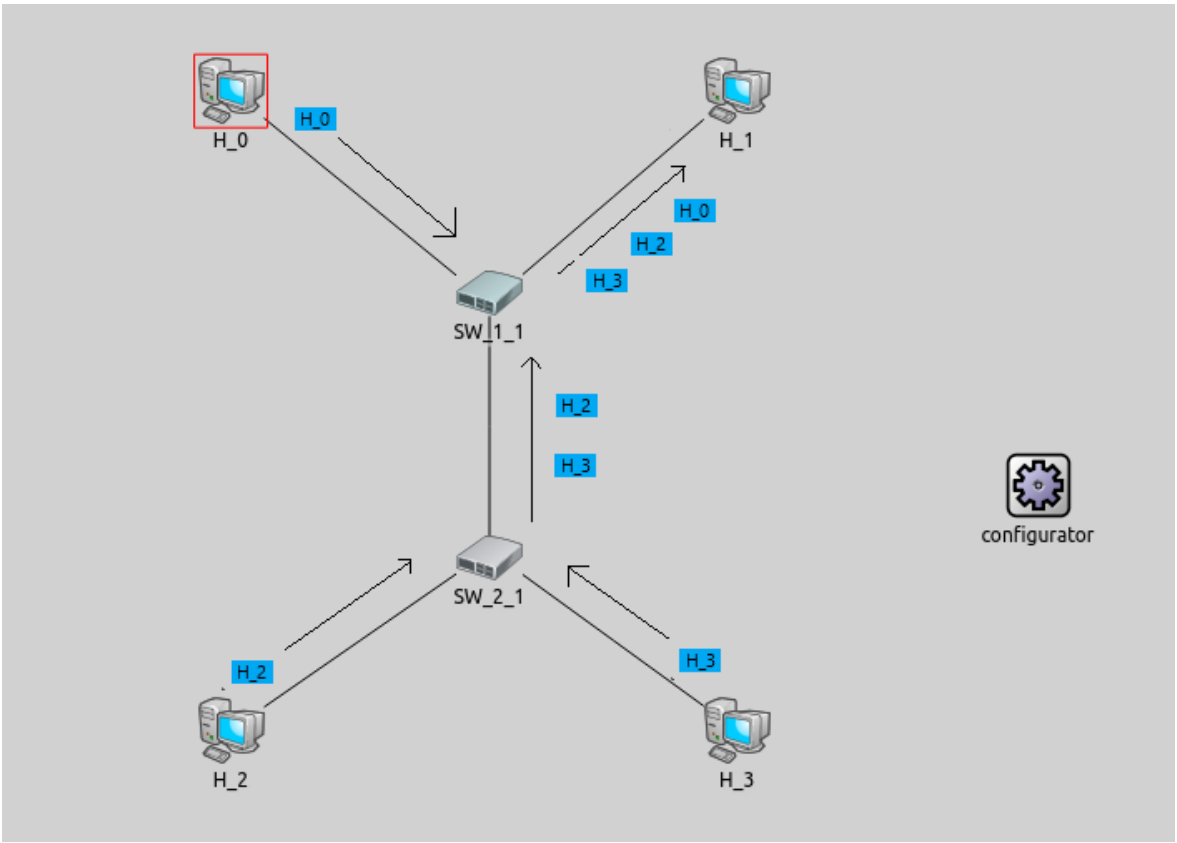
Por ejemplo, el código implementado para desarrollar el escenario 4.1b es el que podemos ver en la figura A.3 y la configuración que se ejecutará es la de la figura A.4

4.2. Análisis de los resultados

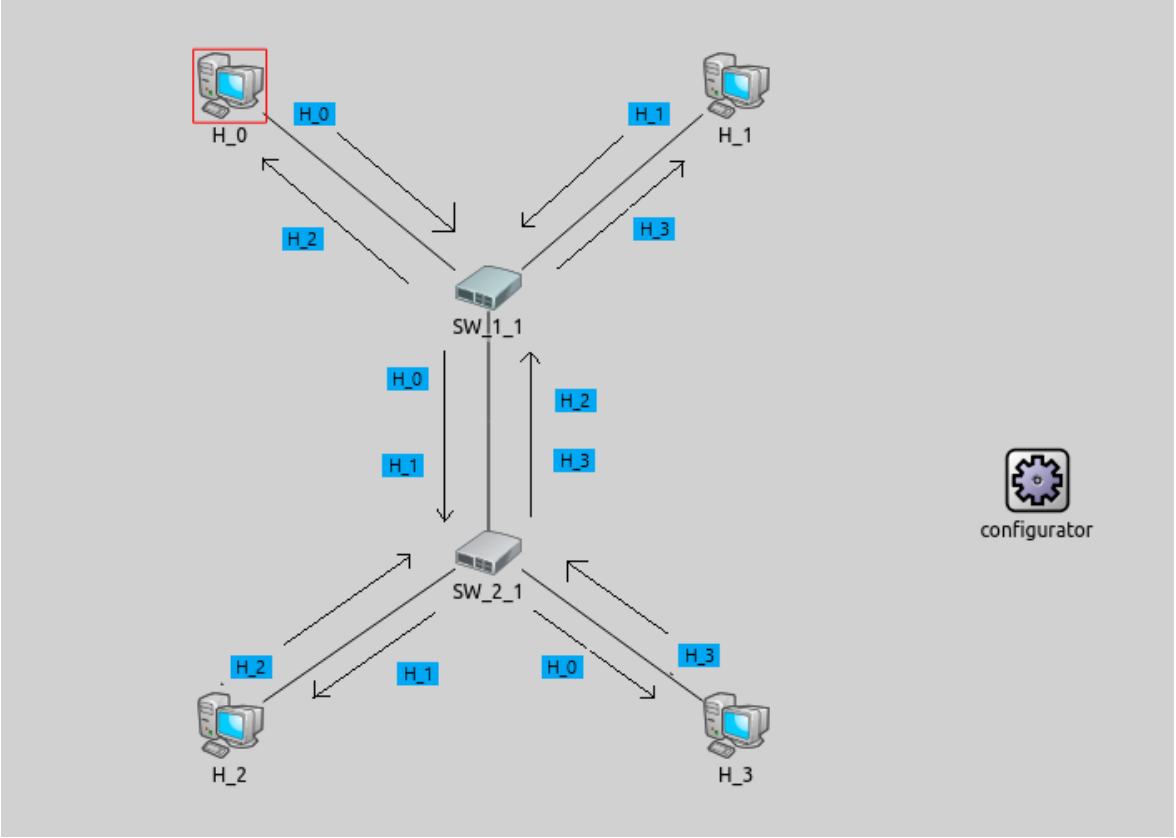
Como hemos dicho en el apartado anterior, vamos a realizar diferentes experimentos para comprobar los beneficios que un sistema obtiene utilizando tráfico RDMA en lugar de tráfico TCP/IP. Estos experimentos serán: comparar la diferencia de latencia que obtenemos al emplear un tipo de tráfico u otro utilizando, para ello, diferentes tamaños de paquete y distintos tamaños de red; la carga que soportan las redes dependiendo del tipo de tráfico utilizado y el rendimiento que nos ofrecen los distintos tipos de tráfico. En las siguientes secciones se detallan estas comparaciones y se comentan los resultados obtenidos.

4.2.1. Comparación de latencia variando el tamaño de paquete

Para la realización de este experimento disponemos de los cuatro escenarios de diferente tamaño mencionados en el apartado anterior a los que les vamos a inyectar los tres tipos de tráfico que vamos a comparar. Para realizar esta comparación vamos a estudiar también como afecta el tipo de envío que realizamos, diferenciando entre dos: envío all-to-one (Figura 4.2a) o envío all-to-all (Figura 4.2b).



(a) Envío all-to-one.



(b) Envío all-to-all.

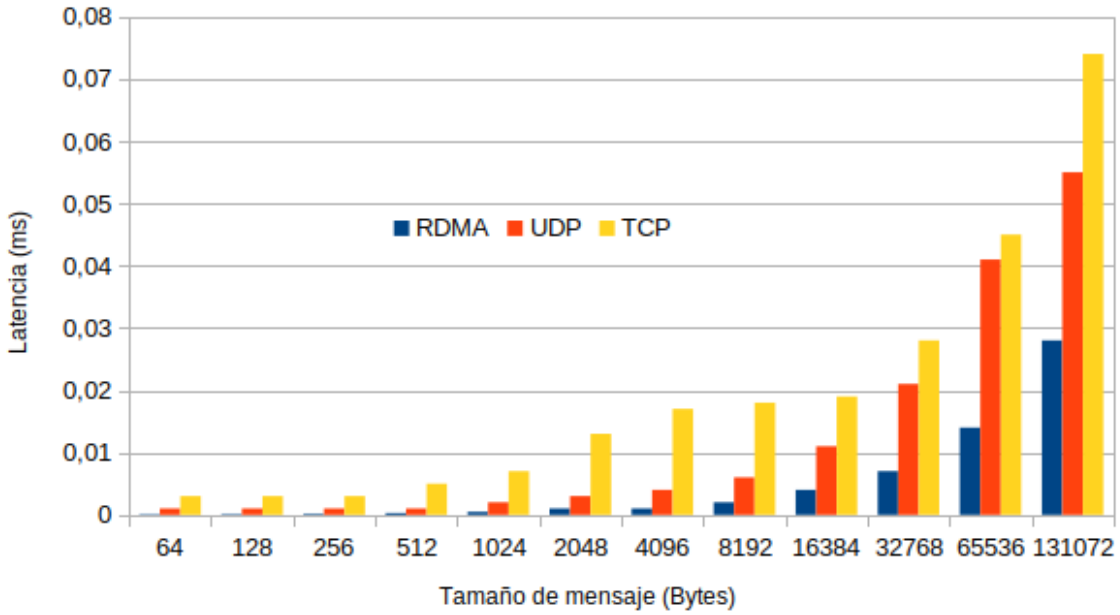
Figura 4.2: Tipos de envío

4.2.1.1. Envío All-To-One

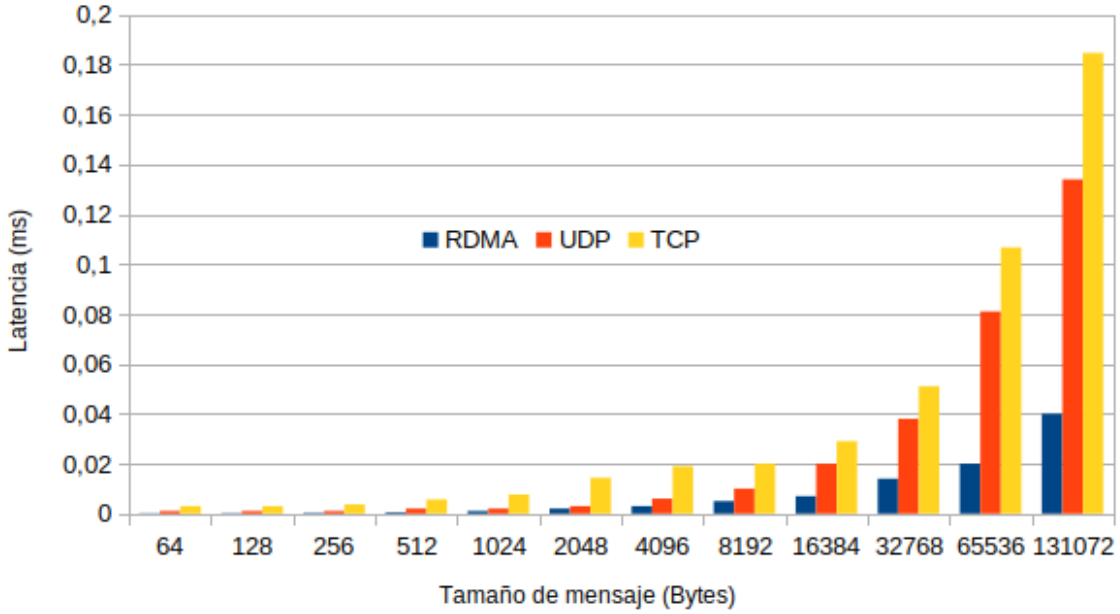
Para la realización de este estudio, vamos a utilizar los casos que se han indicado y, además, vamos a establecer a un host la función de ”sumidero” que recibirá todos los mensajes del resto de hosts, es decir, todos los hosts, menos el host ”sumidero” serán nodos fuente que enviarán hacia él, como se puede ver en la figura . Una vez realizada la ejecución, obtenemos los resultados que podemos ver en las figuras 4.3a, 4.3b, 4.3c, 4.3d, para los cuatro tamaños de red estudiados.

Si comenzamos el análisis desde la red de menor tamaño (Figura 4.3a), podemos ver como para los paquetes de menor tamaño la diferencia de latencia que obtenemos utilizando un tipo de tráfico u otro no supone un gran cambio. Si ahora nos fijamos secuencialmente en los tamaños de mensaje que se representan en la gráfica y miramos su respectiva latencia, vemos que el crecimiento de la latencia para tráfico UDP es mayor que para tráfico RDMA en cada una de ellas, aumentando la diferencia de latencia entre un tráfico y otro a medida que aumenta el tamaño del mensaje. Lo mismo ocurre si comparamos tráfico RDMA con tráfico TCP, pero haciéndose más notable todavía la diferencia de latencia que se obtiene entre el uso de un tráfico u otro

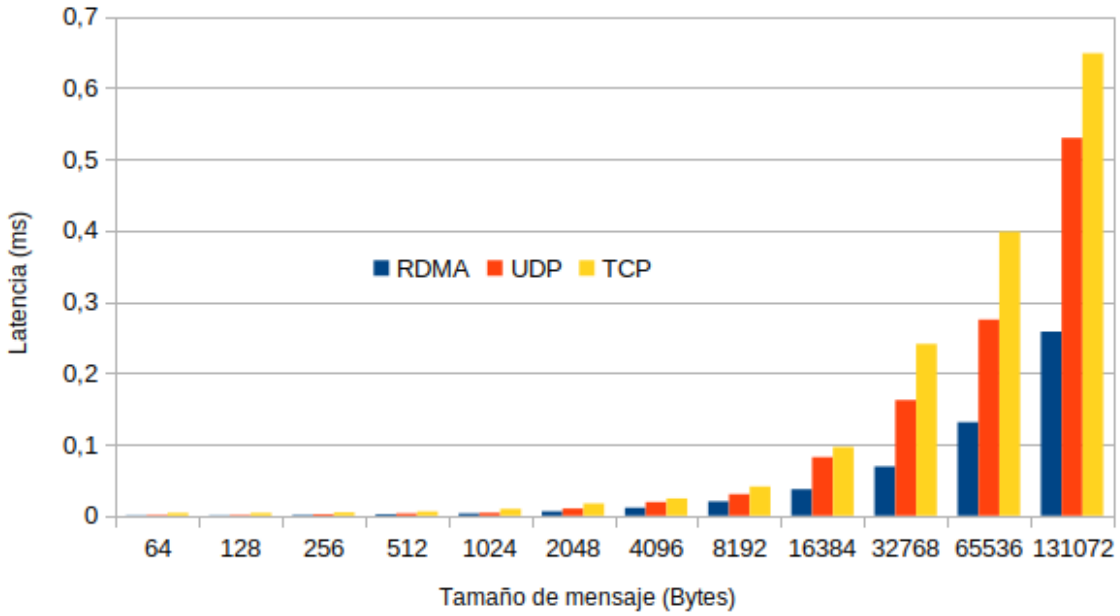
que comparando RDMA con UDP. Esto se debe a la sobrecarga de la red debido a que, además del envío del mensaje, también se envían los ACKs correspondiente para cada paquete. Además, si observamos las gráficas 4.3b, 4.3c, 4.3d vemos que ocurre lo mismo pero con latencias mucho más grandes.



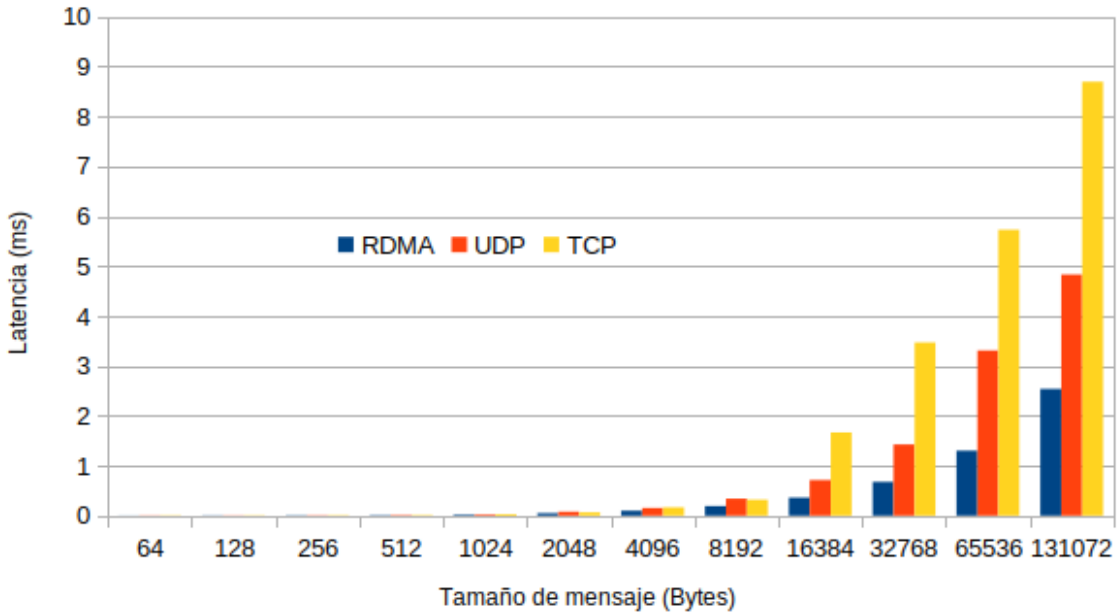
(a) Red de 2 hosts y envío all-to-one.



(b) Red de 4 hosts y envío all-to-one.



(c) Red de 16 hosts y envío all-to-one.

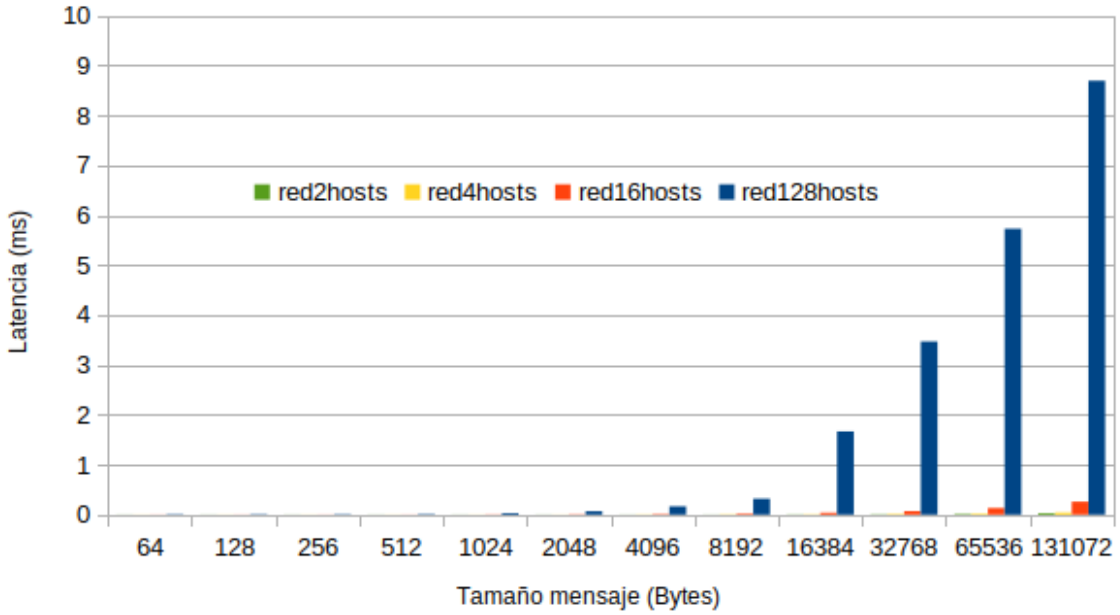


(d) Red de 128 hosts y envío all-to-one.

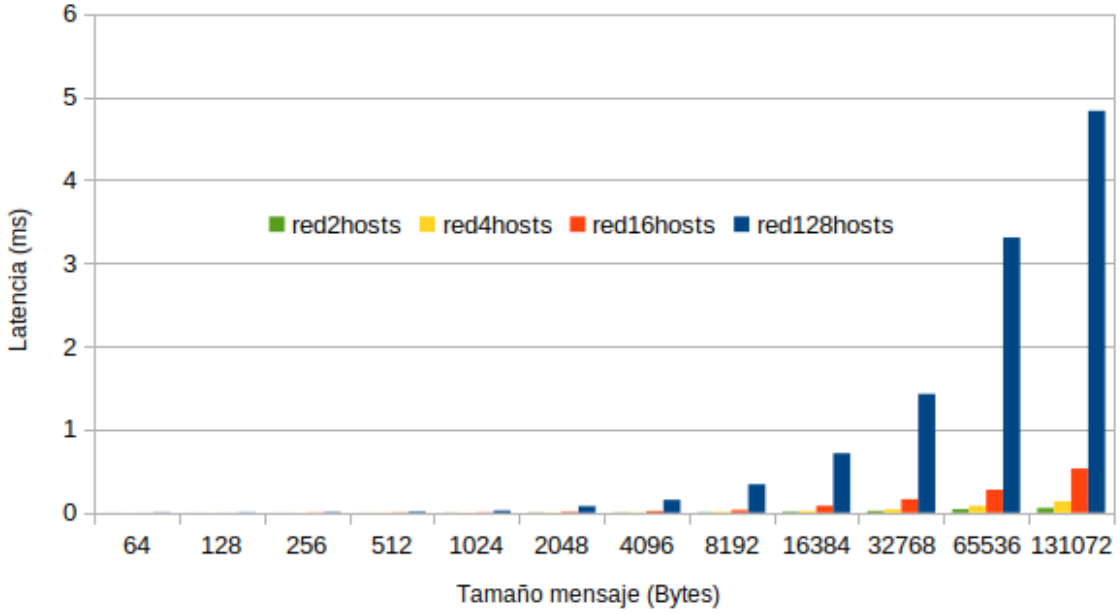
Figura 4.3: Comparación de latencia con envío all-to-one

También podemos ver como el tamaño de la red afecta directamente en la latencia que se obtiene en el envío de un mensaje, aumentando esta considerablemente para cada tamaño de red independientemente del tráfico usado, cómo se ve en las figuras

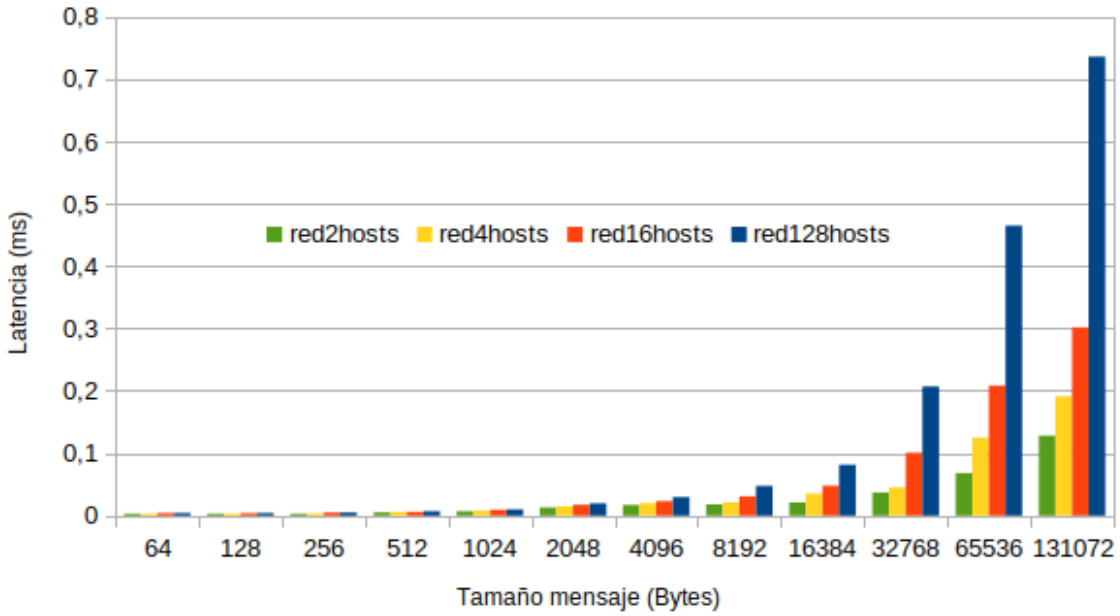
4.4a, 4.4b y 4.4c. Esto se debe a que al haber un mayor número de nodos que envían un mensaje, la sobrecarga de los enlaces y de los buffers es cada vez mayor y el envío de los mensajes se vuelve cada vez más lento y, como consecuencia, su llegada es más tardía.



(a) Tráfico RDMA y envío all-to-one.



(b) Tráfico UDP y envío all-to-one.



(c) Tráfico TCP y envío all-to-one.

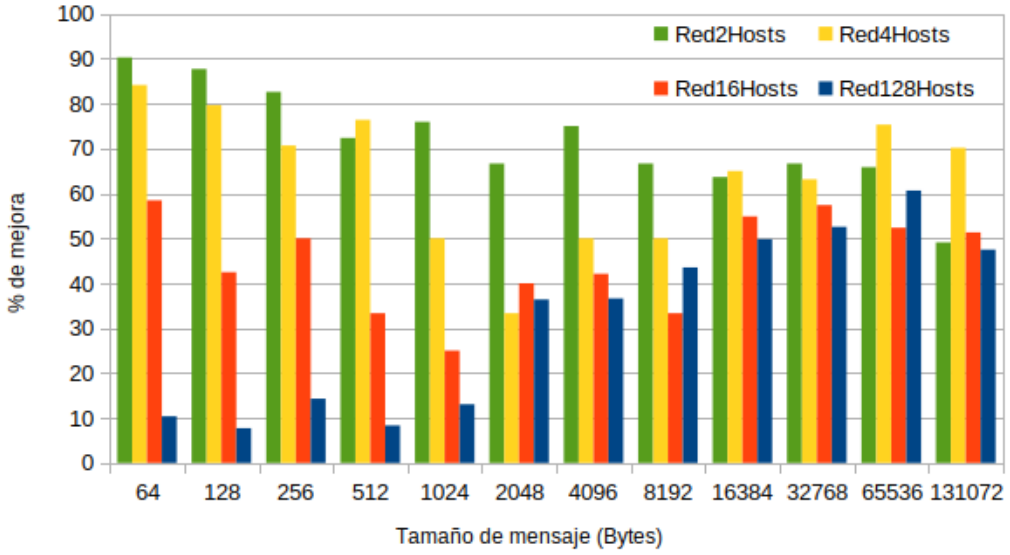
Figura 4.4: Comparación de latencia con envío all-to-one según tamaño de red

En resumen, podemos decir que el aumento de latencia está condicionado tanto por el aumento del tamaño del mensaje cómo por el aumento del tamaño de la red y el tipo de tráfico utilizado, ya que, de acuerdo a los experimentos ya analizados, la latencia de un mensaje irá aumentando a medida que el tamaño del mensaje se va incrementando, a medida que el tamaño de la red crece y cuando utilizamos tráfico TCP/IP en lugar de tráfico RDMA. También observamos que la mejora que se obtiene al utilizar tráfico RDMA es mayor si lo comparamos con tráfico TCP a si lo comparamos con tráfico UDP. Esto se puede ver si observamos las gráficas 4.5a y 4.5b, donde la mayoría de los casos en la gráfica que indica el porcentaje de mejora de RDMA frente a TCP el porcentaje está por encima del 60% y la mayoría de los casos en la gráfica que indica el porcentaje de mejora de RDMA frente a UDP el porcentaje está por debajo del 60%.

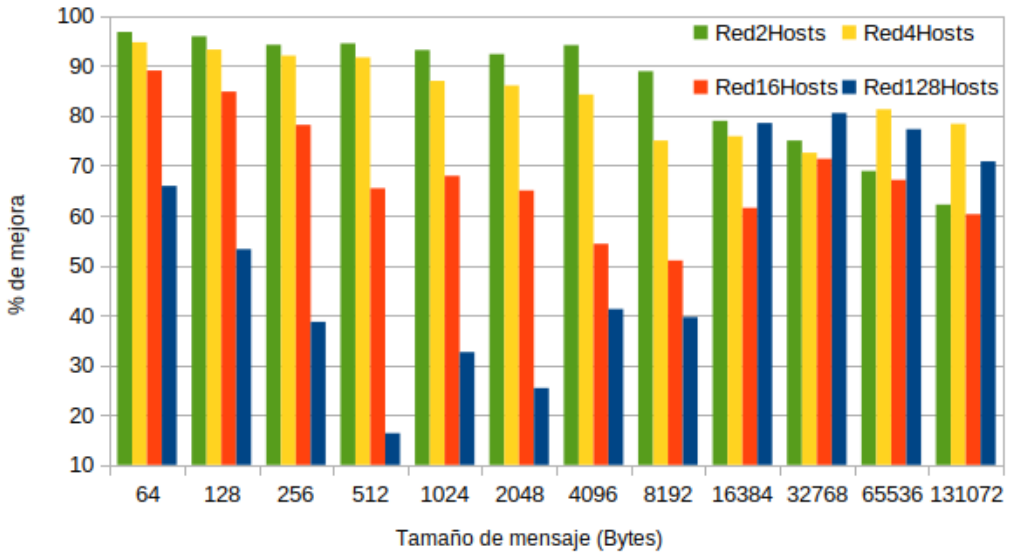
4.2.1.2. Envío All-To-All

Para la realización de este estudio, vamos a utilizar los casos que se han indicado y, además, a diferencia de en el caso anterior, vamos a establecer un envío aleatorio para todos los mensajes, es decir, todos los hosts enviarán mensajes de manera aleatoria a uno del conjunto de hosts de la red y recibirán mensajes del resto de hosts de la red. Una vez realizada la ejecución, obtenemos los resultados que podemos ver en las figuras 4.6a, 4.6b, 4.6c, 4.6d, para los cuatro tamaños de red estudiados:

Para este caso, el análisis que podemos hacer es el mismo que para el caso en el



(a) Porcentaje de mejora en la latencia de RDMA respeto a UDP con envío all-to-one.

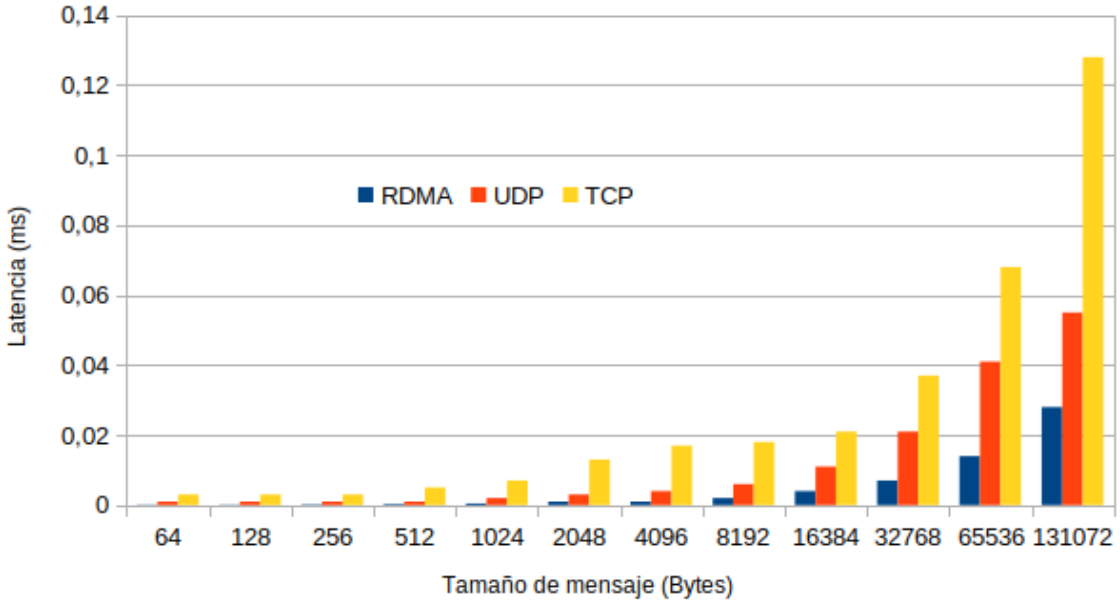


(b) Porcentaje de mejora en la latencia de RDMA respeto a TCP con envío all-to-one.

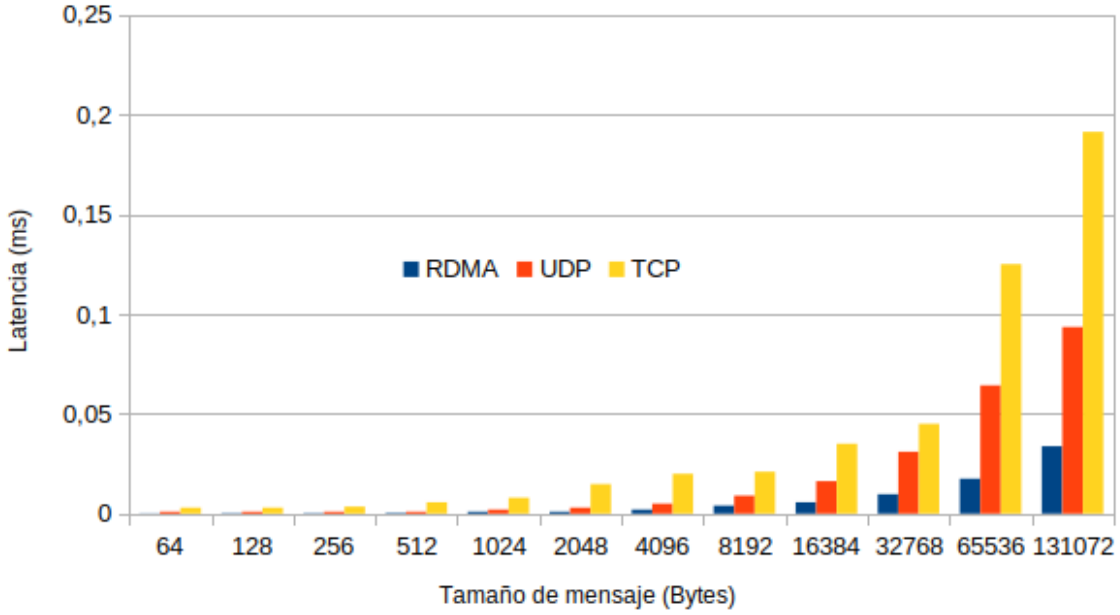
Figura 4.5: Porcentaje de mejora de RDMA frente a UDP y TCP con envío all-to-one

que tenemos envío all-to-one. Vemos, al igual que con envío all-to-one que cuándo el tamaño de mensaje es pequeño, la latencia que se obtiene para todos los tipos de tráfico es muy pequeña, siendo RDMA la menor de ellas, y que a medida que el tamaño del

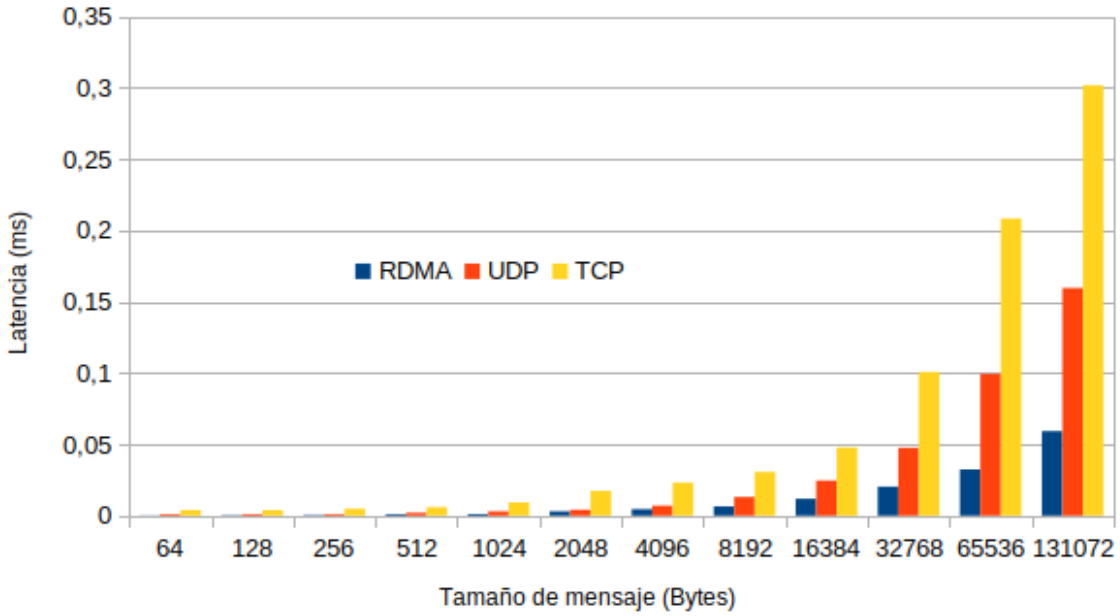
mensaje aumenta mayor es la latencia que se obtiene. Vemos que conforme aumenta el tamaño de paquete también aumenta considerablemente la latencia, pero en todo momento la latencia del tráfico RDMA se mantiene muy por debajo de la latencia del resto de tráficos.



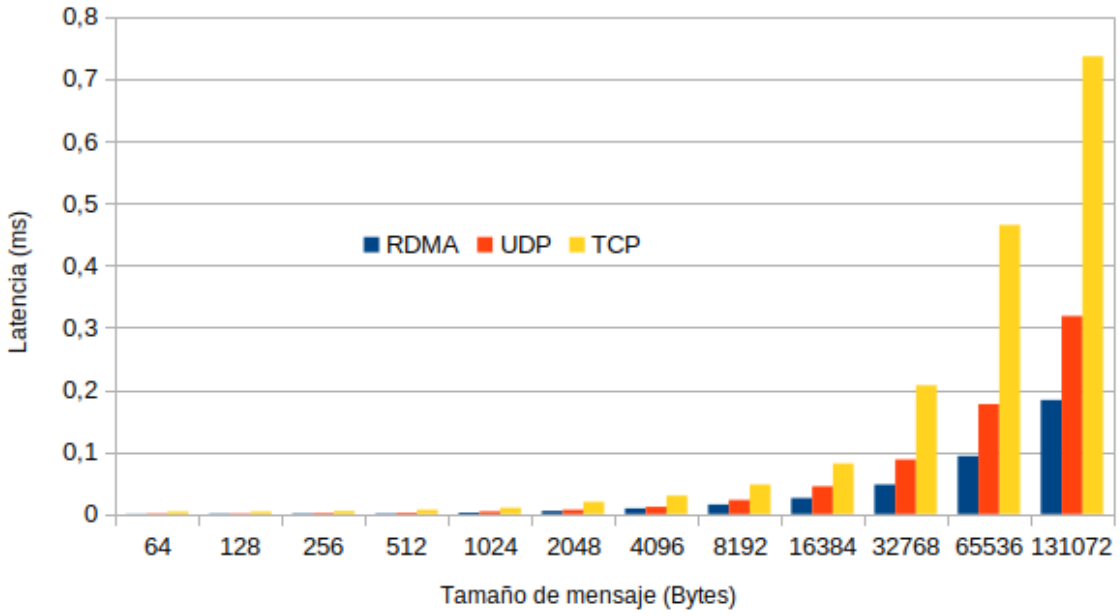
(a) Red de 2 hosts y envío all-to-all.



(b) Red de 4 hosts y envío all-to-all.



(c) Red de 16 hosts y envío all-to-all.

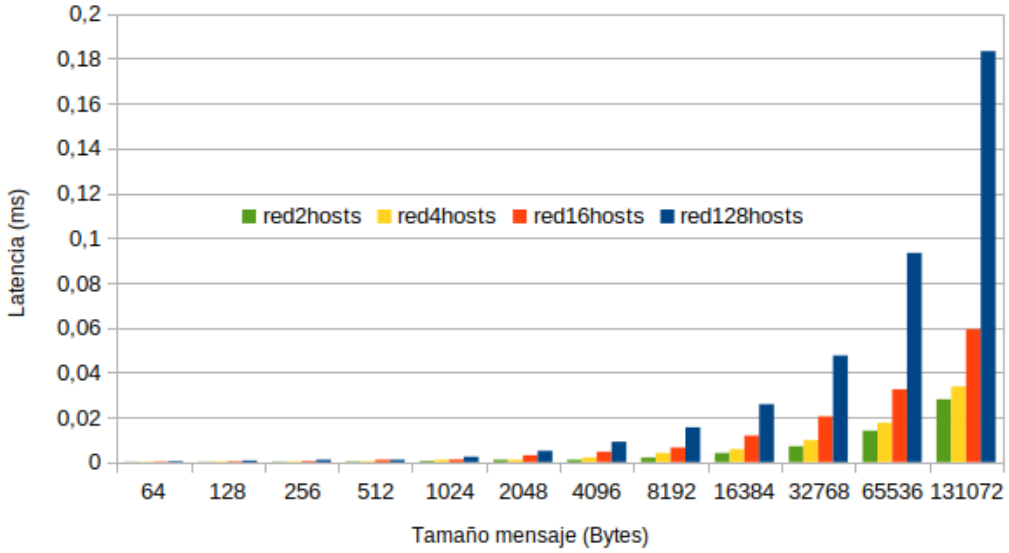


(d) Red de 128 hosts y envío all-to-all.

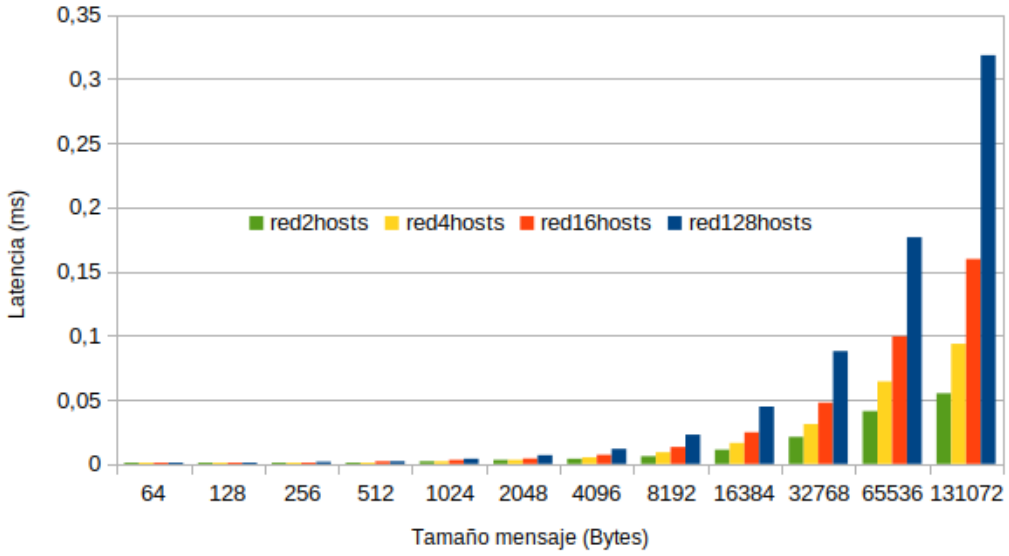
Figura 4.6: Comparación de latencia con envío all-to-all

Lo mismo ocurre con el tamaño de la red que, a medida que aumenta el tamaño de la red mayor es la latencia que se obtiene para un mismo tamaño de mensaje, como se puede ver en las figuras 4.7a, 4.7b y 4.7c. Sin embargo, para este tipo de envío vemos

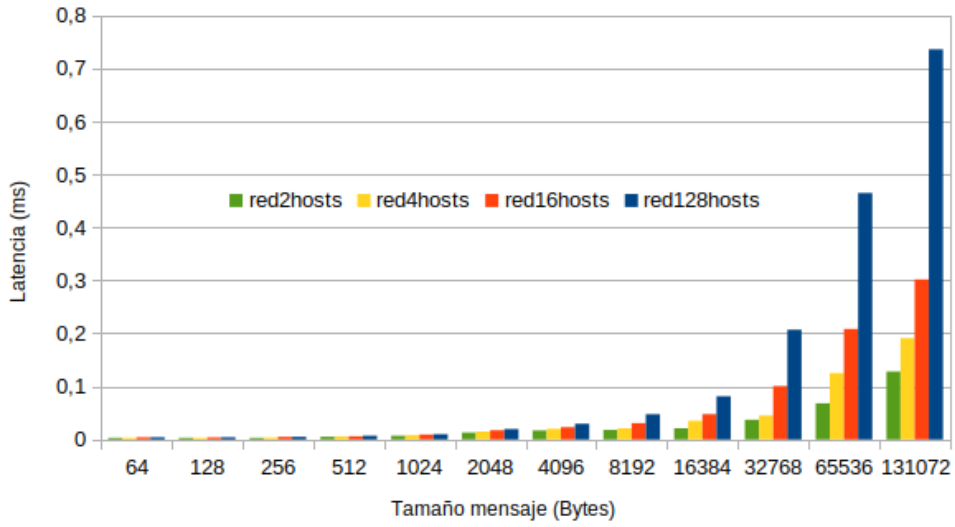
que la diferencia no es tan abismal como cuando se usa envío all-to-one.



(a) Tráfico RDMA y envío all-to-all.



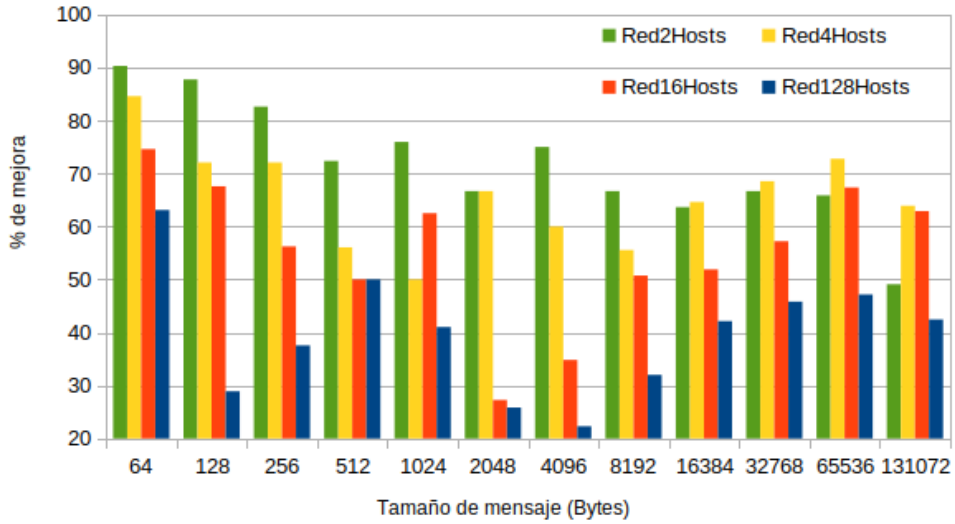
(b) Tráfico UDP y envío all-to-all.



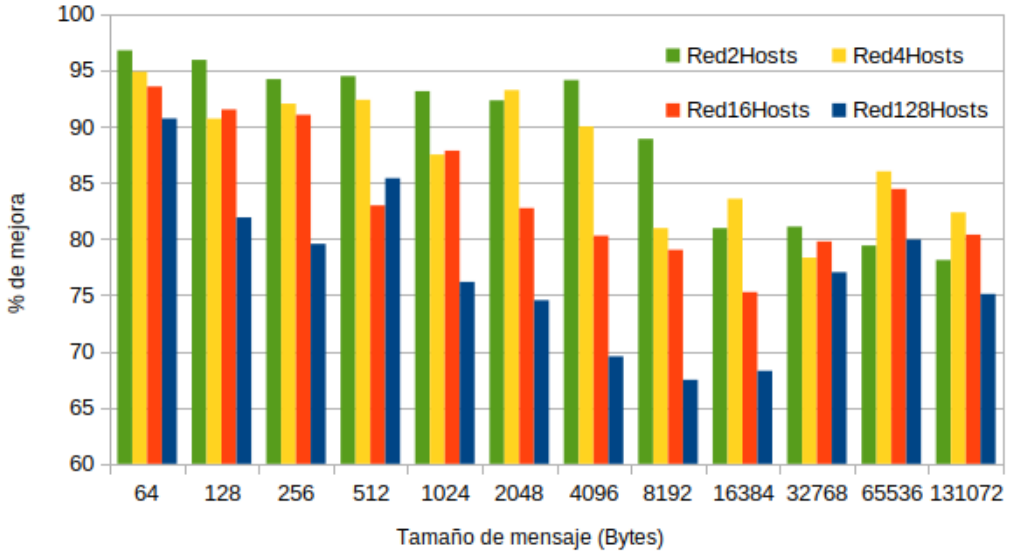
(c) Tráfico TCP y envío all-to-all.

Figura 4.7: Comparación de latencia con envío all-to-all según tamaño de red

Por último, también podemos ver como en estas pruebas el tráfico TCP genera una mayor latencia que los otros dos tráficos. Por lo tanto, la mejora que se observara al usar TCP frente a RDMA será mayor que la ofrecida por UDP, como se ve en las figuras 4.8b y 4.8a.



(a) Porcentaje de mejora en latencia de RDMA respeto a UDP con envío all-to-all.



(b) Porcentaje de mejora en latencia de RDMA respeto a TCP con envío all-to-all.

Figura 4.8: Porcentaje de mejora de RDMA frente a UDP y TCP con envío all-to-all

El último comentario que podemos hacer respecto a las pruebas realizadas para los dos tipos de envío de mensaje es que el tipo de tráfico que utilicemos también condicionará la latencia que se obtendrá, siendo el tráfico all-to-one el que mejor latencia obtendrá para redes de menor tamaño, ya que, al ser redes pequeñas los buffers no se colapsarán tanto como si se muchos nodos envían hacia el resto. Sin embargo, para redes de mayor tamaño, el tráfico all-to-all obtendrá mejores resultados de latencia con respecto al tráfico all-to-one ya que al enviar todos hacia el mismo host todos los mensajes tendrán que pasar por dispositivos que sean similares para todos lo que llegará a suponer un cuello de botella.

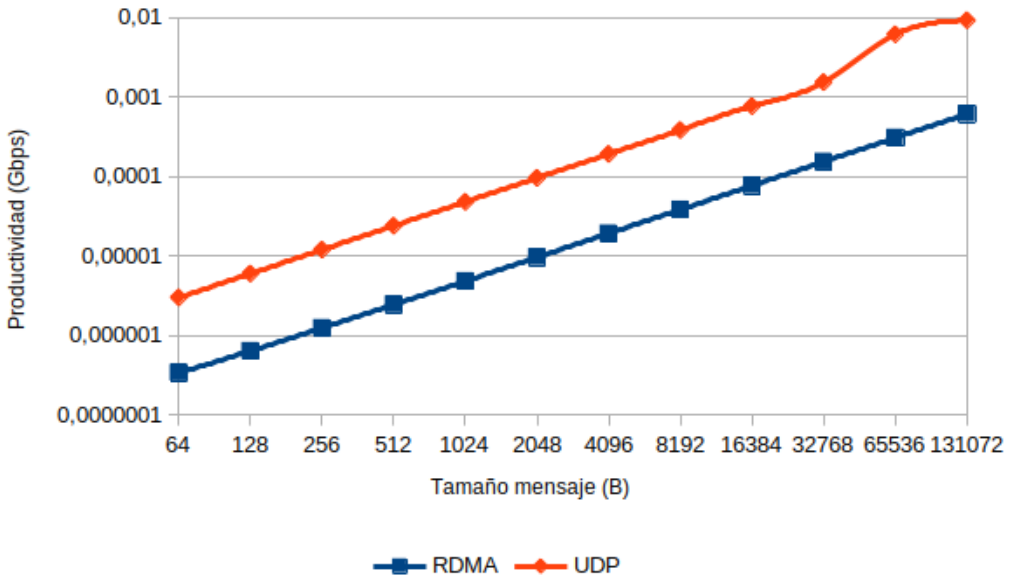
4.2.2. Comparación de rendimiento obtenido

En esta sección vamos a comprobar los efectos que se producen al utilizar un tipo de tráfico u otro en cuánto al rendimiento que obtenemos, es decir, la cantidad de datos que se pueden transferir del origen al destino en un plazo determinado. El rendimiento mide cuántos paquetes llegan a su destino con éxito. En general, el rendimiento se mide en bits por segundo. Para este estudio también vamos a diferenciar como afecta el utilizar un envío all-to-one o un envío all-to-all.

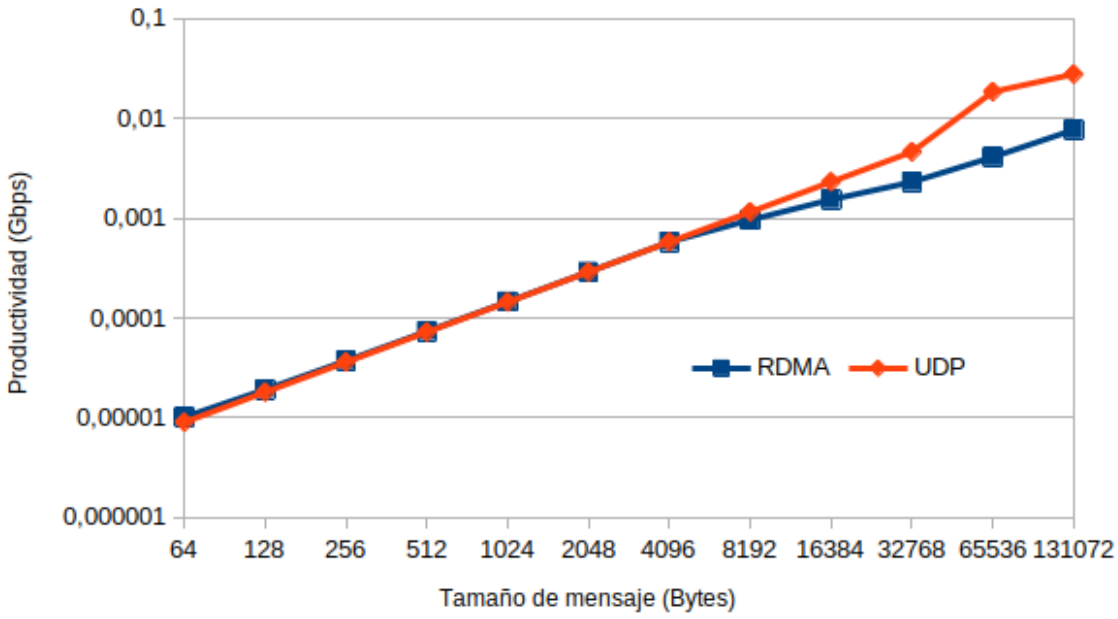
4.2.2.1. Rendimiento con envío All-To-One

Para el envío de tráfico en el que todos los hosts envían a uno solo y, por lo tanto, este recibe los mensajes de todos los demás nodos, en la figura 4.9a podemos ver como

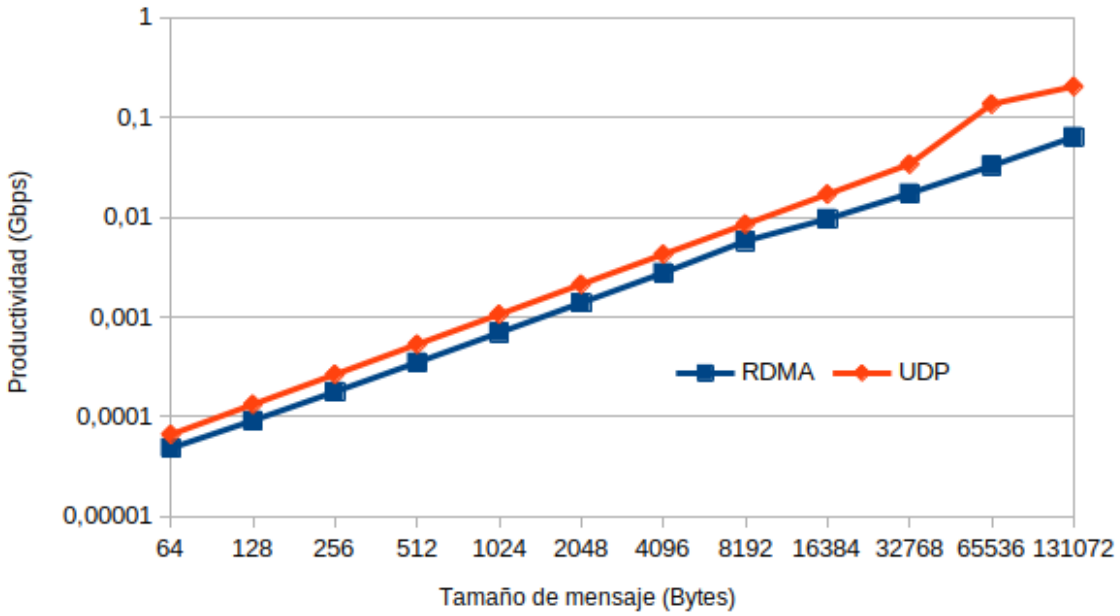
UDP hay una mayor productividad que si utilizamos tráfico RDMA pero, si nos vamos a las figuras 4.9b y 4.9c hay vemos como la diferencia en cuanto a utilizar UDP o RDMA es insignificante. Es decir, cuándo tenemos redes de mayor tamaño la diferencia entre usar RDMA o UDP es menor.



(a) Productividad red 2 hosts y envío all-to-one.



(b) Productividad red 4 hosts y envío all-to-one.



(c) Productividad red 16 hosts y envío all-to-one.

Figura 4.9: Comparación de la productividad con envío all-to-one

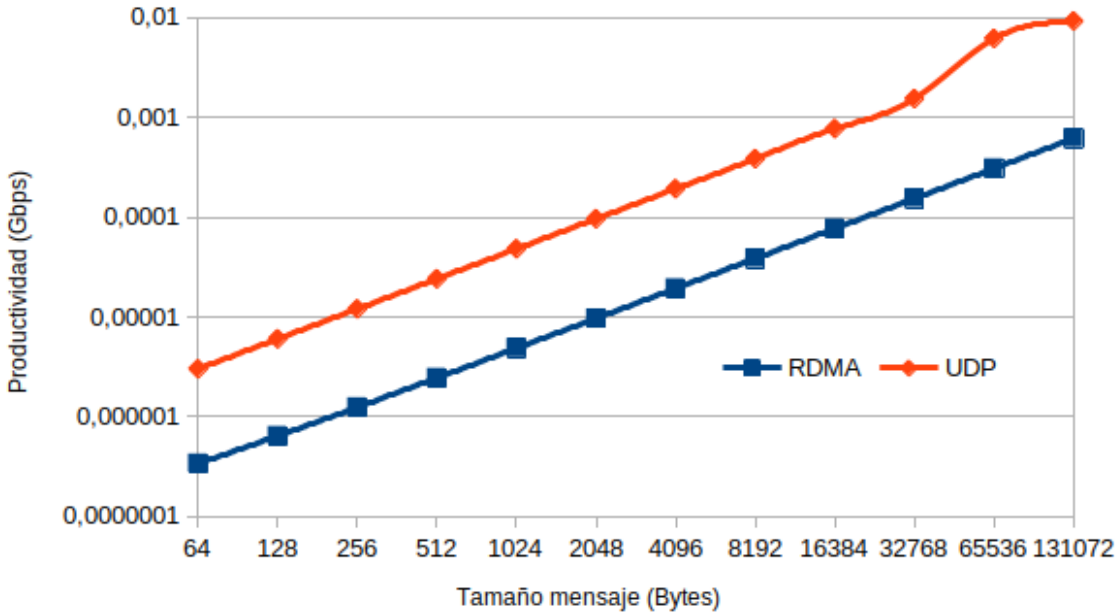
4.2.2.2. Rendimiento con envío All-To-All

Para el envío aleatorio en el que todos los hosts envían y reciben, en la figura 4.10a vemos que ocurre los mismo que en el caso de envío all-to-one, el rendimiento que obtenemos con UDP es mayor. Sin embargo, también ocurre lo mismo con las redes de mayor tamaño a la de 2 hosts. En las figuras 4.10b y 4.10c vemos como aquí también la diferencia en cuánto a productividad que existe entre usar RDMA y UDP es mínima. Por tanto, en este caso podemos llegar a la misma conclusión. Para redes de mayor tamaño, la diferencia entre usar tráfico RDMA o UDP es mínima.

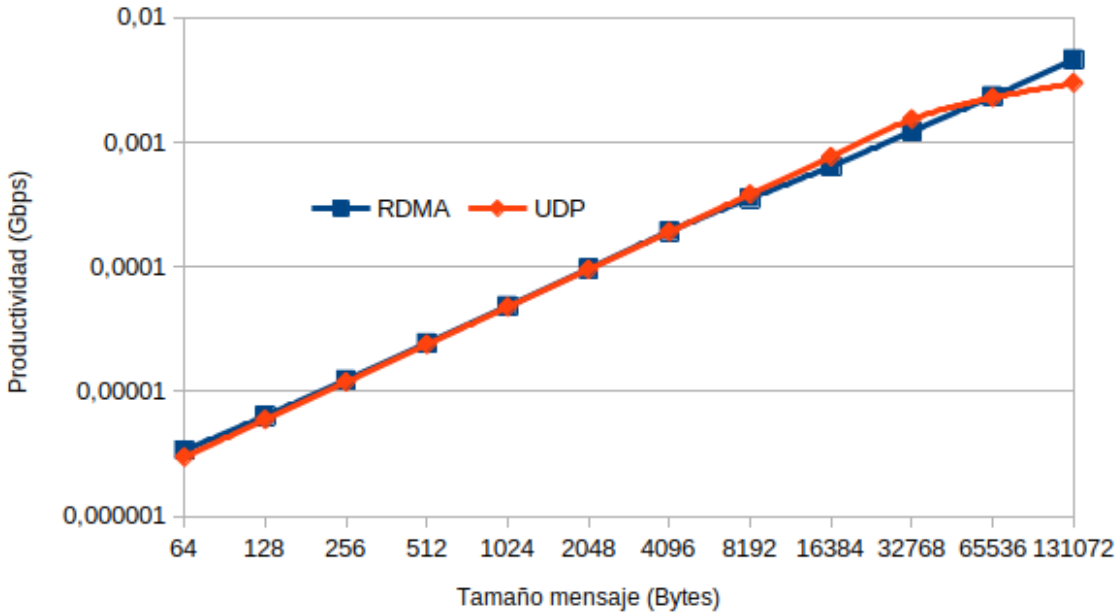
Como conclusión a esta sección, podemos decir que para de tamaño pequeño obtenemos una mayor productividad si utilizamos tráfico TCP/IP como UDP, sin embargo, cuándo la red crece, la diferencia de rendimiento entre un tráfico u otro decrece hasta llegar a hacerse mínima. Si unimos esta comparación con la de la sección 4.2.1 obtenemos mayores beneficios si utilizamos tráfico RDMA frente al uso de tráfico TCP/IP.

4.2.3. Comparación de carga soportada

En este apartado vamos a comprobar como el intervalo de envío de mensajes afecta a la latencia que obtenemos para cada uno de los tráficos utilizados. En otras palabras, vamos a comprobar la cantidad de carga que puede soportar la red. Para ello vamos a utilizar la red de 16 hosts que podemos ver en la figura 4.1c y un tamaño de mensaje de 64 KBytes o lo que es lo mismo 65536 bytes e iremos variando el intervalo de tiempo



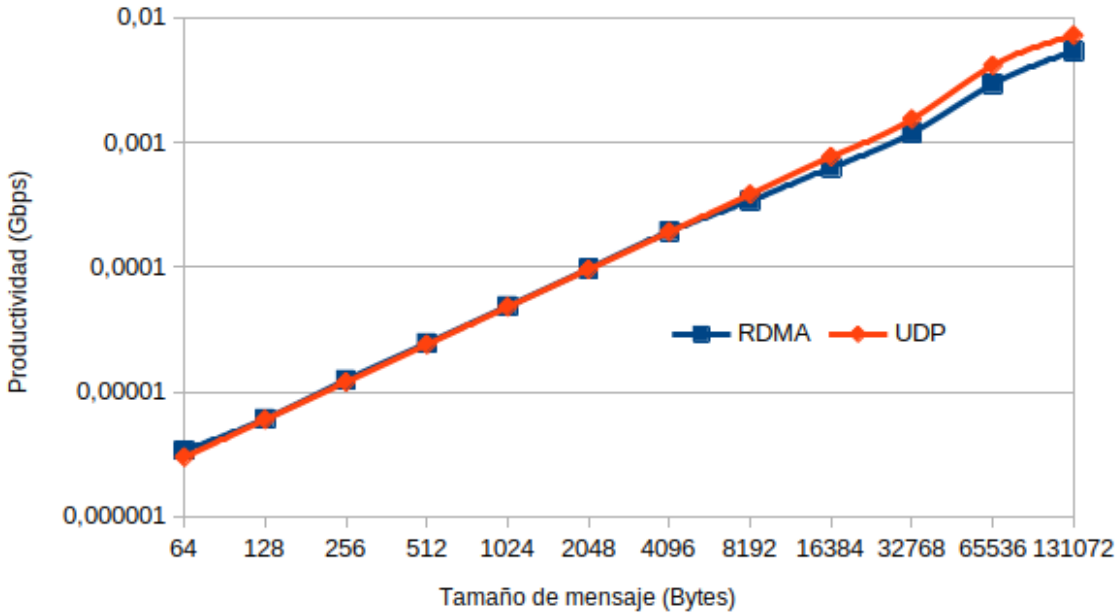
(a) Productividad red 2 hosts y envío all-to-all.



(b) Productividad red 4 hosts y envío all-to-all.

en el que se envían los mensajes.

El resultado que obtenemos en este experimento es que la diferencia entre usar un tipo de tráfico u otro no repercute en la carga que puede soportar la red, es decir, si observamos la figura 4.11 vemos que el momento en el que la latencia que se obtiene sufre un gran incremento respecto al intervalo de tiempo anterior para los diferentes



(c) Productividad red 16 hosts y envío all-to-all.

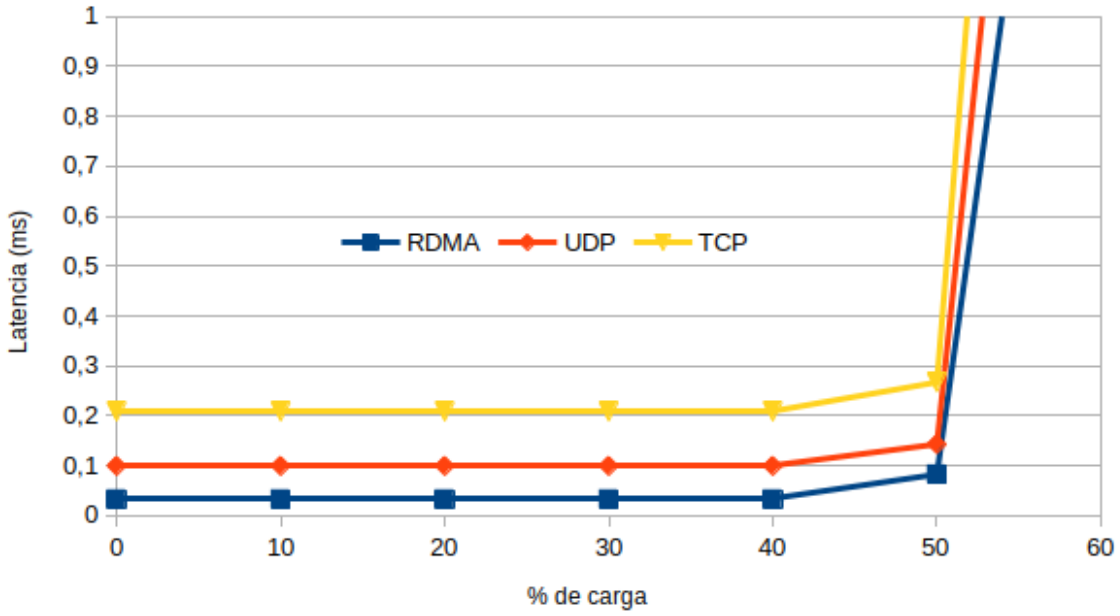


Figura 4.11: Carga soportada.

tipos de tráfico. Eso sí, podemos observar también, que al igual que en el apartado 4.2.1 en el que se varía el tamaño del mensaje, en este caso en el que variamos el intervalo de envío, la latencia con UDP y TCP también sufrirá un impacto mayor al que sufre la latencia obtenida con tráfico RDMA.

4.2.4. Conclusiones de los experimentos

Una vez hemos concluido con la realización y el análisis de los experimentos podemos obtener algunas conclusiones:

En primer lugar hemos comprobado que el uso de tráfico RDMA repercute en una latencia más baja para los diferentes tamaños de mensaje que podamos emplear independientemente del tamaño de la red que empleemos debido a que evitamos el uso de sockets y la necesidad de pasar por el kernel. Esto permite que podamos evitar atravesar cada una de las capas del protocolo TCP/IP en la recepción pudiendo enviar directamente el mensaje desde la NIC a la aplicación ya que estamos empleando tráfico RDMA basado en Ethernet (RoCE) y en el envío sí que es necesario atravesar cada una de las capas para conocer la dirección IP del host receptor.

En cuánto a la productividad, hemos observado que para redes muy pequeñas es mejor el uso de tráfico TCP/IP. Sin embargo, cuándo la red crece, el rendimiento que se obtiene para ambos tiene una diferencia mínima por lo que sería indiferente el uso de un tipo de tráfico u otro.

En cuanto a la carga soportada podemos concluir también que es indiferente el uso de tráfico TCP/IP o RDMA ya que el momento en el que se produce la saturación del sistema llega aproximadamente en el mismo momento eso sí, en ese momento, al igual que en todas las pruebas que hemos realizado a lo largo del trabajo, la latencia que se alcanzará se significativamente más elevada utilizando tráfico UDP que tráfico RDMA.

Para concluir, si tenemos en cuenta las tres comparaciones que hemos realizado, podemos decir que RDMA nos ofrece mayores prestaciones ya que, a pesar de que en términos de productividad los datos nos dicen que es indiferente el uso de un tráfico u otro (salvo en redes muy pequeñas) y que en la prueba de tráfico soportado vemos que la latencia se dispara en el mismo instante de tiempo para todos los tipos de tráfico, la latencia que obtenemos para los diferentes tamaños de paquete y de red es muy inferior a la que nos ofrece tanto el tráfico UDP con el tráfico RDMA. Es decir, la latencia obtenida por RDMA marca la diferencia con respecto al tráfico TCP/IP.

5. Conclusiones y Trabajo futuro

En este último capítulo se resume todo lo expuesto a lo largo del trabajo como las conclusiones que hemos alcanzado. Se indicarán posibles ideas para avanzar sobre este tema en el futuro. Por último, se indicarán las tareas que se han realizado a lo largo del tiempo en el que se ha desarrollado el proyecto y las competencias que nos han permitido adquirir el desarrollo de esas tareas.

5.1. Conclusiones

El protocolo TCP/IP tradicional comienza a ser un gran problema en la computación de altas prestaciones debido a la, cada vez mayor, demanda de un alto rendimiento y una baja latencia llegando a ser la causa de que se produzcan cuellos de botella en la red. Ahí es donde entra en escena RDMA.

RDMA nos ofrece ciertas ventajas que posteriormente repercuten en medidas como es la latencia ofreciéndonos, sobre todo, una latencia inferior a la que nos ofrecen protocolos de TCP/IP como UDP o TCP. Esto provoca que RDMA se esté implementando en otras tecnologías distintas a la tecnología donde surgió cómo es InfiniBand, como por ejemplo en Ethernet dando lugar a nuevos protocolos como son RoCE e iWARP basados en los tradicionales.

Además, realizar su implementación en simuladores nos permite realizar diferentes pruebas para verificar lo que se dice, algo que se ha hecho en este proyecto, y también seguir avanzando en su investigación para encontrar nuevas mejores y técnicas de uso

Por todo esto, RDMA seguirá avanzando en su incremento de uso, sobre todo en supercomputadores.

5.2. Trabajo futuro

Como trabajo futuro planeamos:

- Mejorar el código desarrollado para la implementación de RDMA basado en Ethernet (RoCEv1) en el programa OMNeT++ para hacerlo más eficiente y, cómo consecuencia, conseguir una simulación más eficiente.
- Incorporar el protocolo RDMA al host ya existente en INET.
- Investigar para conseguir implementar un nuevo código que ponga en funcionamiento el protocolo RoCEv2.
- Integrar la implementación desarrollada en este proyecto con el trabajo implementado en el correspondiente simulador OMNeT++ del RAAP.
- Intentar la realización de pruebas con mayor tamaño de mensaje y tamaño de red.

5.3. Tareas realizadas y competencias adquiridas

Las tareas que hemos realizado durante el desarrollo de este proyecto consisten en:

1. Revisar artículos, publicaciones y páginas web donde se detalle información acerca de la redes de interconexión.
2. Revisar artículos, publicaciones y páginas web donde se detalle información sobre RDMA.
3. Navegar por los diferentes archivos ya desarrollados y protocolos ya implementados en OMNeT e INET para familiarizarnos con el entorno.
4. Comenzar con la implementación del protocolo RDMA en el framework INET.
5. Modelar los escenarios sobre los que se van a realizar los experimentos.
6. Ejecución de los experimentos y obtención de los resultados y creación de las gráficas.
7. Escritura de la memoria.

Tarea	Descripción	Objetivo
1	Estudiar el estado del arte de las redes de interconexión	1
2	Estudiar el estado del arte de RDMA	1
3	Estudiar el funcionamiento del entorno de simulación OMNeT++ y del framework INET	2
4	Modelar el protocolo RDMA en OMNET++	3
5	Desarrollo de los entornos de prueba	4
6	Obtención de resultados y realización de gráficas	4
7	Escritura de la memoria	4

Tabla 5.1: Tareas realizadas

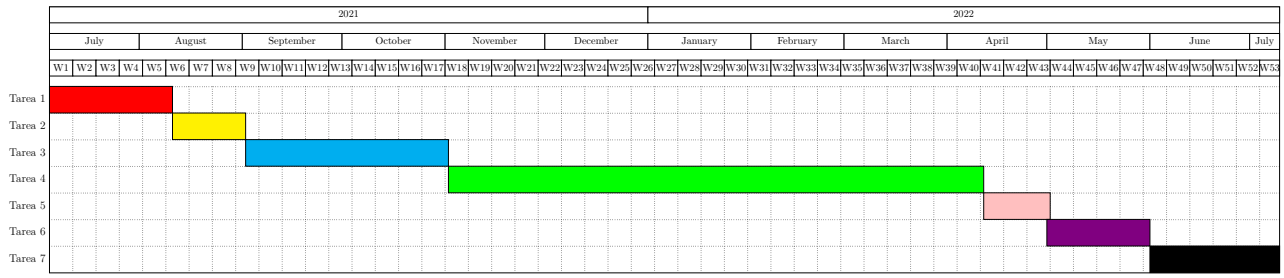


Figura 5.1: Diagrama de Gant

A partir del desarrollo de las tareas mencionadas anteriormente, se han conseguido alcanzar las competencias reflejadas en la tabla 5.2 con un reparto de tiempo correspondiente al reflejado en la figura 5.1

Competencias	Descripción	Tareas
1	Capacidad para analizar arquitecturas de computadores, incluyendo su organización, despliegue y comportamiento	1, 2
2	Capacidad para trabajar a partir de código ya existente, siendo capaz de entenderlo, modificarlo si fuera necesario y desarrollar código propia basado en él	3, 4, 5
3	Capacidad para proponer experimentos para la evaluación de las prestaciones que ofrece una red y su posterior documentación	6 y 7

Tabla 5.2: Competencias adquiridas

A. Anexo 1

```
1   void Rdma::handleUpperPacket(Packet *p)
2   {
3       EV_INFO << "Receive packet " << p << " from upper layer\n";
4       packet = p;
5       emit(packetReceivedFromUpperSignal, packet); //emit a signal
    indicating the arrival of a msg from app-layer
6
7       //Declaración de variables para el envío
8
9       if (totalLength.get() <= RDMA_MAX_MESSAGE_SIZE) {
10          //Envío si el tamaño del mensaje recibido es menor o igual
    que el tamaño máximo de mensaje (4096 bytes)
11
12          send(packet, "lowerLayerOut");
13          //Control de enlaces
14          numSent++;
15      }else{
16          //Envío si el tamaño del mensaje recibido es menor o igual
    que el tamaño máximo de mensaje (4096 bytes)
17
18          send(fragment, "lowerLayerOut");
19          offset += thisFragmentLength;
20          //Control de enlaces
21      }
22      numSent++;
23  }
24
```

Figura A.1: Ejemplo de código de envío de un mensaje

```
1   void Rdma::sendUp(Ptr<const RdmaHeader>& header, Packet *payload,
    ushort srcPort, ushort destPort)//, clocktime_t generationTime)
2   {
3       EV_INFO << "Sending payload up to app layer\n";
4
5       //Configuración del paquete a envíar a la capa superior
6       payload->addTagIfAbsent<L4PortInd>()->setSrcPort(srcPort);
7       payload->addTagIfAbsent<L4PortInd>()->setDestPort(destPort);
8
9       send(payload, "appOut");
10      //Control de enlaces
11      numPassedUp++;
12  }
13
```

Figura A.2: Ejemplo de código de recepción

```
network Red
{
    submodules:
        H_0: RdmaHost {
            @display("p=108,38");
        }
        H_1: RdmaHost {
            @display("p=288,38");
        }
        H_2: RdmaHost {
            @display("p=108,265");
        }
        H_3: RdmaHost {
            @display("p=288,265");
        }
        SW_1_1: EthernetSwitch {
            @display("p=200,110");
            gates:
                ethg[3];
        }
        SW_2_1: EthernetSwitch {
            @display("p=200,204");
            gates:
                ethg[3];
        }

        configurator: Ipv4NetworkConfigurator {
            @display("p=395,178");
        }
    connections:
        H_0.ethg++ <--> Eth40G <--> SW_1_1.ethg[0];
        H_1.ethg++ <--> Eth40G <--> SW_1_1.ethg[1];
        H_2.ethg++ <--> Eth40G <--> SW_2_1.ethg[0];
        H_3.ethg++ <--> Eth40G <--> SW_2_1.ethg[1];
        SW_1_1.ethg[2] <--> Eth40G <--> SW_2_1.ethg[2];

}
```

Figura A.3: Creación del escenario de 4 hosts

```
[General]
network = Red16Hosts_Tcp
sim-time-limit = 34s
**.H*.app[*].startTime = 32s

#Configuración de las variables de los escenarios

#Activación del protocolo STP

[Config All-To-All]
#Configuración Arp

#Configuración de la simulación
**.H*.numApps = 1
**.H*.app[0].typename = "RdmaBasicApp"
**.H*.app[0].localPort = 1000
**.H*.app[0].destPort = 1000
**.H*.app[0].destAddresses = moduleListByNedType("inet.node.inet.
    StandardHost")
**.H*.app[*].sendInterval = 0.01s
**.H*.app[0].messageLength = $
    {64,128,256,512,1024,2048,4096,8192,16384,32768, 65536, 131072}B

#########################################################################

[Config All-To-One]
#Configuración Arp

#Configuración de la simulación
**.H*.numApps = 1
**.H*.app[0].typename = "RdmaBasicApp"
**.H*.app[0].localPort = 1000
**.H*.app[0].destPort = 1000
**.H_1.app[0].destAddresses = ""
**.H*.app[0].destAddresses = "H_1"
**.H*.app[*].sendInterval = 0.01s
**.H*.app[0].messageLength = $
    {64,128,256,512,1024,2048,4096,8192,16384,32768, 65536, 131072}B
```

Figura A.4: Configuración de la simulación de la red de 4 hosts

Referencia bibliográfica

[Top,] Redes de interconexiÓn. https://www.infor.uva.es/~bastida/Arquitecturas%20Avanzadas/Redes.pdf. 9

[roc, 2012] (2012). Roce and infiniband: Which should i choose?). https://www.infinibandta.org/roce-and-infiniband-which-should-i-choose/. 24

[roc, 2019] (2019). Low latency networks: Roce or iwarp? https://semiengineering.com/low-latency-networks-roce-or-iwarp/. 25

[Omn, 2019a] (2019a). Omnet++ simulation manual. https://doc.omnetpp.org/omnetpp/manual/. 26

[ine, 2019] (2019). What is inet framework? https://inet.omnetpp.org/Introduction.html. 31

[Omn, 2019b] (2019b). What is omnet++? https://omnetpp.org/intro/. vii, 25

[Int, 2021] (2021). Interconnection network design - tutorialspoint. https://www.tutorialspoint.com/parallel_computer_architecture/parallel_computer_architecture_interconnection_network_design.htm. 5

[use, 2022] (2022). =https://inet.omnetpp.org/docs/users-guide/. 31, 32

[PFC, 2022] (2022). Control de flow basado en prioridad (pfc). https://docs.microsoft.com/es-es/windows-hardware/drivers/network/priority-based-flow-control--pfc. 25

[DeCusatis, 2022] DeCusatis, C. (2022). *Handbook of Fiber Optic Data Communication. A Practical Guide to Optical Networking*. 24

[Jose Duato, 2003] Jose Duato, Sudhakar Yalamanchili, L. N. (2003). *Interconnection Networks: An engineering approach*. Morgan Kaufmann. xiii, 5, 10, 11, 14, 15, 16, 17, 18, 19

[Liu, 2018] Liu, Q. (2018). Introducing rdma into computer networks. 1

[Redhat, 2022] Redhat (2022). Configuración de redes infiniband y rdma. https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/8/html/configuring_infiniband_and_rdma_networks/understanding-infiniband-and-rdma_configuring-and-managing-networking. 20