Preguntas generales sobre C++ y Omnet++

- ¿Por qué decimos que C++ es "compilado"? ¿Cuál es la diferencia con un lenguaje "interpretado"?

C++ es compilado, esto quiere decir que uno escribe el código en un archivo de texto, y este texto es compilado (traducido a un lenguaje máquina), y eso es lo que finalmente se ejecuta.

Los lenguajes compilados son convertidos directamente a código máquina que el procesador puede ejecutar. Como resultado, suelen ser más rápidos y más eficientes al ejecutarse en comparación con los lenguajes interpretados. También le dan al desarrollador más control sobre aspectos del hardware, como la gestión de memoria y el uso del CPU.
La principal diferencia entre un lenguaje compilado y uno interpretado es que el lenguaje compilado requiere un paso adicional antes de ser ejecutado, la compilación, que convierte el código que escribes a lenguaje de máquina. Un lenguaje interpretado, por otro lado, es convertido a lenguaje de máquina a medida que es ejecutado.

- ¿Qué queremos decir cuando decimos que C++ tiene "tipado fijo"?

C++ tenemos tipado fijo, las variables se declaran con un tipo

Cuando decimos que C++ tiene "tipado fijo", nos referimos al hecho de que el tipo de una variable en C++ se determina en tiempo de compilación y no puede cambiar durante la ejecución del programa. En C++, debes declarar explícitamente el tipo de cada variable antes de utilizarla, y una vez que se ha declarado, el tipo está fijo y no se puede cambiar.

- ¿Cual es la diferencia de declarar una clase con "MyClass* myObject" y con "MyClass myObject"?

El * es un puntero (algo que apunta a un espacio de memoria) que va a estar definido por la Estructura de MyClass. Un puntero es una dirección de memoria donde tengo información que a mi me interesa. Acá hay que inicializar el objeto después de declararlo (myObject = new myClass()) y al último borrarlo (delete myObject). Los métodos se llaman con ->.
MyClass myObject es más simple pero limita mucho la utilización del objeto en el momento donde estamos. Esto no es un puntero, es una variable, es un objeto que está ya declarado e inicializado. Los métodos se llaman con el .
La ventaja/desventaja de declararlo como una variable (dentro del main) es que cuando salga del bloque se va a borrar automáticamente.
Si uso un puntero lo tengo que borrar explícitamente con el delete.
Pero muchas veces quiero que este objeto sobreviva más allá de este bloque lógico (en el que lo declare), quiero tener el control de esa memoria hasta que ordene borrarla, para acceder desde otro método.
Omnet tiende a trabajar con punteros.

La diferencia entre declarar una clase con "MyClass* myObject" y "MyClass myObject" radica en cómo se almacena y se accede al objeto en la memoria.
1)"MyClass* myObject": En esta declaración, "myObject" es un puntero a un objeto de la clase "MyClass". El puntero es una variable que almacena la dirección de memoria donde se encuentra el objeto real. En este caso, se crea una variable que puede apuntar a un objeto existente o ser asignada a un objeto creado dinámicamente en tiempo de ejecución utilizando el operador "new". Para acceder a los miembros de la clase a través del puntero, se utiliza el operador de desreferencia "->". Es decir, si se tiene "myObject->doSomething()", se está llamando al método "doSomething()" del objeto al que apunta el puntero "myObject".
2)"MyClass myObject": En esta declaración, "myObject" es una instancia directa de la clase "MyClass". Esto significa que la variable "myObject" almacena el objeto completo en sí mismo. No

es un puntero, sino una representación completa del objeto "MyClass". Para acceder a los miembros de la clase, se utiliza el operador de punto ".". Es decir, si se tiene "myObject.doSomething()", se está llamando al método "doSomething()" del objeto "myObject".
En resumen, cuando se declara una clase con "MyClass* myObject", se está creando un puntero que puede apuntar a un objeto existente o ser asignado a un objeto creado dinámicamente. Mientras que cuando se declara una clase con "MyClass myObject", se está creando una instancia directa de la clase en sí misma.

- ¿Qué quiere decir que un método o una variable sea "private", "public" o "protected"?

Cuando declaramos algo private adentro de una clase estamos diciendo que solamente lo que este definido dentro de esta clase va a poder acceder a esta variable. Lo que es public se puede llamar desde afuera de la clase. Protected es muy parecida a private con la salvedad de que además protected se puede llamar desde clases que luego hereden de esta clase.

En la programación orientada a objetos, los conceptos de "private", "public" y "protected" son modificadores de acceso que se utilizan para controlar la visibilidad y el alcance de los miembros (variables y métodos) de una clase. Aquí te explico qué significa cada uno de ellos:
1)"private": Cuando se declara un miembro de una clase como "private", significa que ese miembro sólo es accesible desde dentro de la misma clase. No puede ser accedido ni modificado desde otras clases o funciones externas. Es la forma más restrictiva de acceso y se utiliza para encapsular y ocultar la implementación interna de la clase. Los métodos y variables "private" se utilizan para proporcionar funcionalidades internas y asegurar la integridad y coherencia de los objetos.
2)"public": Un miembro declarado como "public" es accesible desde cualquier parte del programa. Puede ser accedido, modificado y llamado desde cualquier otra clase o función. Los métodos y variables "public" se utilizan para definir la interfaz pública de la clase, es decir, las operaciones y propiedades que están destinadas a ser utilizadas por otros objetos o partes del programa.
3)"protected": Un miembro "protected" es similar a "private" en el sentido de que sólo es accesible desde dentro de la clase que lo define. Sin embargo, a diferencia de "private", un miembro "protected" también puede ser accedido desde las clases derivadas (subclases) de la clase que lo declara. Esto permite que las subclases heredan y accedan a los miembros protegidos, mientras que las clases externas no pueden. Los miembros "protected" se utilizan para implementar la herencia y proporcionar una forma controlada de acceso a las subclases.
En resumen, los modificadores de acceso "private", "public" y "protected" definen el nivel de visibilidad y acceso a los miembros de una clase. "private" restringe el acceso solo a la clase que lo define, "public" permite el acceso desde cualquier parte del programa y "protected" permite el acceso desde la clase que lo define y sus subclases. Estos modificadores de acceso son fundamentales para mantener la encapsulación, la seguridad y la estructura de las clases en la programación orientada a objetos.

- ¿Por qué utilizamos simuladores para las capas inferiores? ¿Qué dificultades hay para implementar protocolos de bajo nivel?

- ¿Qué es un simulador de eventos discretos? ¿Qué ventajas/propiedades tiene?

En una simulación de eventos discretos vamos a tener un tiempo de simulación que va a avanzar no necesariamente igual que el tiempo real, nuestra simulación va a tener un tiempo que es discreto (va a pegar saltos). El tiempo de simulación avanza en los eventos en una cola de eventos. Un evento sucede en momentos discretos de tiempo. El estado del sistema puede cambiar durante estos eventos. Una de las ventajas es que nos permite acelerar análisis que de otra manera serían muy complejos de hacer. Lo mismo pasa a la inversa. Podemos ir ejecutando paso a paso los procesos, viendo que está haciendo o eligiendo un algoritmo y poder estudiarlo en mayor

profundidad. El tiempo de simulación avanza por medio de una cola de eventos, que establece una cronología que indica en qué momento van a pasar los eventos.
Ventajas:
- Estudiar tiempos reales prolongados en tiempos de simulación reducidos
- Observar en detalle tiempos reales pequeños
- Desarrollar modelos de protocolos y algoritmos definidos por eventos
- Correr cientos de casos y hacer comparaciones paramétricas
- Corregir y actualizar los modelos basado en los resultados

Un simulador de eventos discretos es una herramienta computacional utilizada para simular y estudiar el comportamiento de sistemas dinámicos que evolucionan a través de eventos discretos en el tiempo. En este tipo de simulador, el tiempo avanza en forma de saltos discretos, y los eventos ocurren en momentos específicos.

- ¿Qué función cumplen los archivos .ini, .ned y .cc en Omnet++?

Vamos a usar clases de C++ para definir los comportamientos de los eventos/modulos, y vamos a ver un par de archivos más para definir el escenario. Un sistema/red lo vamos a definir con un archivo .ned y además, los parámetros que va a tener este sistema lo vamos a escribir en un archivo .ini. Tanto .ned como .ini son archivos de texto que no se compilan pero que los usa Omnet++ para simular una red. .ned definir los módulos que componen nuestra red y C++ para definir cómo se comportan estos módulos.

En OMNeT++, un framework de simulación de eventos discretos, los archivos .ini, .ned y .cc cumplen funciones clave en la definición y configuración de los modelos de simulación. Aquí te explico brevemente qué función cumple cada uno de ellos:
-Archivos .ini (Initialization): Estos archivos contienen los parámetros de configuración de la simulación. Se utilizan para especificar los valores iniciales de los parámetros de simulación, como la duración de la simulación, las tasas de llegada de eventos, los tamaños de los buffers, etc. Los archivos .ini permiten definir diferentes configuraciones para ejecutar la simulación y cambiar los parámetros sin necesidad de recompilar el código fuente. Proporcionan una forma conveniente de ajustar y personalizar la configuración de la simulación sin modificar el código.
-Archivos .ned (Network Description): Estos archivos se utilizan para describir la estructura y conectividad de los componentes del modelo de simulación. Los archivos .ned definen los módulos y sus puertos de entrada/salida, así como las conexiones entre ellos. En esencia, los archivos .ned describen la topología de la red o sistema que se está modelando. OMNeT++ utiliza una sintaxis específica para definir los módulos y sus relaciones en un formato legible por el framework.
-Archivos .cc (C++ source code): Estos archivos contienen el código fuente en C++ que implementa la lógica y el comportamiento de los módulos de simulación. Los archivos .cc se utilizan para escribir el código que define cómo se procesan los eventos, cómo se realizan las interacciones entre los módulos y cómo se recopilan y analizan los resultados de la simulación. Aquí es donde se implementa la funcionalidad específica de cada módulo y se escriben las operaciones y algoritmos que determinan el comportamiento del sistema modelado.
En resumen, los archivos .ini se utilizan para configurar los parámetros de la simulación, los archivos .ned describen la estructura y las conexiones de los componentes del modelo, y los archivos .cc contienen el código fuente en C++ que define la lógica y el comportamiento de los módulos de simulación en OMNeT++. Estos archivos son esenciales para definir y configurar modelos de simulación en OMNeT++ y permiten personalizar la simulación, describir la topología de la red y programar la funcionalidad específica de cada módulo.

- ¿Por qué usamos punteros para mensajes o eventos (cMessage * msg) y no declaración directa (cMessage msg)?

Porque utilizando punteros nos aseguramos la existencia del msg a pesar de que se haya terminado de ejecutar el bloque lógico que lo contiene. (CREO)

En OMNeT++, se utiliza punteros para mensajes o eventos (como cMessage *msg) en lugar de la declaración directa (cMessage msg) por varias razones:

1. Dinamismo y flexibilidad: Utilizar punteros permite crear y manejar dinámicamente los mensajes durante la simulación. En OMNeT++, los mensajes se crean y destruyen en tiempo de ejecución según las necesidades del sistema. Al utilizar punteros, se puede asignar memoria en el momento necesario y liberarla cuando el mensaje ya no es necesario. Esto proporciona flexibilidad para el manejo de la memoria y evita la necesidad de asignar memoria estáticamente antes de la ejecución.
2. Eficiencia de la memoria: El uso de punteros permite una gestión más eficiente de la memoria. En OMNeT++, los mensajes pueden ser enviados y procesados por múltiples módulos en diferentes momentos. Utilizar punteros para mensajes evita la copia de grandes bloques de memoria cada vez que se envía o procesa un mensaje, lo cual puede ser costoso en términos de rendimiento y uso de memoria. En su lugar, se pasa solo la dirección del mensaje, reduciendo la sobrecarga de memoria y acelerando el proceso de envío y procesamiento.
3. Polimorfismo y herencia: OMNeT++ admite la herencia y el polimorfismo en la jerarquía de mensajes. Al utilizar punteros, se puede trabajar con mensajes de diferentes tipos derivados de la clase base cMessage. Esto permite una mayor flexibilidad en el diseño del modelo y la implementación de la simulación, ya que se pueden definir y utilizar diferentes tipos de mensajes según las necesidades específicas de cada componente o escenario.

En resumen, el uso de punteros para mensajes o eventos en OMNeT++ ofrece dinamismo, flexibilidad y eficiencia en el manejo de la memoria. Además, permite el polimorfismo y la herencia, lo que facilita el diseño y la implementación de modelos de simulación más complejos y adaptables.

- ¿Por qué dimos un taller de matplotlib (notebooks)?¿Cuáles son las limitaciones de graficar en Omnet++?

El taller de matplotlib sirvió bastante para graficar las estadísticas que fuimos obteniendo al probar los casos de estudio.
Las limitaciones de graficar en Omnet++ son que solo podes ver los gráficos sobre ejes cartesianos y no de otra forma, también que solo podes ver los datos que fuiste recolectando y no alguna operación sobre ellos que resulte de interés.

Preguntas sobre Lab 3 (Capa Transporte)

- Una red veloz que alimenta a un receptor de baja capacidad ¿Qué tipo de problema es? ¿Cómo se soluciona?

Este es un problema de flujo. Se soluciona avisando de alguna manera al emisor de que reduzca su velocidad y sea acorde a la del receptor.

- Una red lenta que alimenta a un receptor de alta capacidad ¿Qué tipo de problema es? ¿Cómo se soluciona?

Si el host de origen es de baja capacidad, entonces no se trata de un problema, solo que estamos desaprovechando la alta capacidad del receptor..
Si el host de origen es de alta capacidad, entonces se trata de un problema de congestión.

Y se pueden solucionar avisando al emisor que deje de generar por un tiempo o baje su velocidad un poco.
La capa de transporte proporciona un mecanismo de control de congestión haciendo que el emisor regule su velocidad cuando la red está congestionada.
También podríamos solucionar dentro de la red, pero eso se hace en su propia capa.

- ¿Cuál es la diferencia entre los problemas de control de flujo y los problemas de control de congestión?

Flujo: de host origen a destino sin tener en cuenta a los routers intermedios.
El objetivo principal del control de flujo es evitar que el receptor se vea abrumado por un flujo de datos demasiado rápido y sufra desbordamiento o pérdida de datos.

Congestión: tiene en cuenta todo el camino entre host origen y destino.
El objetivo es evitar perdidas, retrasos y disminución en el rendimiento general de la red.

- Si los búferes son infinitos ¿Cual es la consecuencia de un problema de congestión?

Al poder crecer infinitamente nunca se eliminará ningún paquete nuevo ni viejo, por lo que tal vez nunca lleguen a destino.
Si trabajaramos con ACK, estos nunca llegarían al emisor por lo que generaran paquetes duplicados constantemente y solo empeora la situación.

Tal vez agregar memoria ayude hasta cierto punto, pero Nagle (1987) descubrió que si los enrutadores tienen una cantidad infinita de memoria, la congestión empeora en vez de mejorar. Esto se debe a que, para cuando los paquetes llegan a la parte frontal de la cola, ya expiraron (repetidas veces) y se enviaron duplicados. Esto empeora las cosas, no las mejora: conduce al colapso por congestión.

- Si los búferes son finitos ¿Cual es la consecuencia de un problema de congestión?

Cuando se llenen comenzarán a eliminar cada paquete nuevo dando lugar a que los primeros paquetes tengan la posibilidad de llegar.
Esto genera que la carga ofrecida sea mayor que la carga útil.

Preguntas sobre Lab 4 (Capa Red)

- ¿Qué ventajas tienen las topologías anillo desde el punto de vista de la tolerancia a fallos?

Un sistema tolerante a fallas es aquel que puede experimentar una falla (o múltiples fallas) en sus componentes, pero que continúa funcionando correctamente. Ahora bien, esta topología nos da dos rutas posibles para que un paquete pueda seguir y llegar al mismo destino. La disponibilidad de múltiples rutas aumenta la redundancia de una red. En una topología de anillo bidireccional, dos anillos permiten que los datos se envíen en ambas direcciones. Esta configuración crea redundancia (tolerancia a fallos), lo que significa que si uno de los anillos falla, los datos pueden transmitirse por el otro.

- ¿Qué ventajas tienen las topologías anillo desde el punto de vista del enrutamiento?

Desde el punto de vista de enrutamiento, el nodo de host de origen será el único que tendrá que tomar una decisión con respecto a cuál enlace elegir para transmitir un paquete, pero solo tiene 2 posibles opciones, y eso dependerá del algoritmo de enrutamiento que esté siguiendo. Luego de eso, los nodos posteriores tendrán que ir pasando el paquete por el enlace distinto al que recibió el paquete, y seguir ese proceso hasta que se llegue al nodo destino. (Depende igual, esto no puede ser siempre cierto)
Ventaja resumida: Un nodo sólo tiene 2 opciones para enrutar un paquete, y la decisión que tome el router se verá influenciada por el algoritmo de enrutamiento presente. (esto si siempre es cierto)

- ¿Para qué sirve utilizar inundación en los algoritmos de enrutamiento? ¿Se usa para datos o control?

Cuando se implementa un algoritmo de enrutamiento, cada enrutador debe tomar decisiones con base en el conocimiento local, no en la imagen completa de la red. Una técnica local simple es la inundación, en la que cada paquete entrante se envía en todas las líneas de salida, excepto en la línea por la que llegó.
La inundación no es práctica para enviar la mayoría de los paquetes, pero tiene algunos usos importantes. En primer lugar, asegura que un paquete se entregue en todos los nodos de la red. Esto podría ser un desperdicio si sólo hay un destino que necesite el paquete, pero es efectivo para difundir información. En segundo lugar, la inundación es en extremo robusta. Incluso si grandes cantidades de enrutadores vuelan en pedazos (por ejemplo, en una red militar ubicada en una zona de guerra), la inundación encontrará una ruta, si es que existe, para transmitir un paquete a su destino. Además, la inundación requiere muy poca configuración. Los enrutadores sólo necesitan conocer a sus vecinos. Esto significa que la inundación se puede usar como bloque de construcción para otros algoritmos de enrutamiento que son más eficientes pero requieren una configuración un poco más complicada.
Más que nada se utiliza para control, ya que son otros los algoritmos que utilizan inundación como un método para difundir su información de router a todos lados, luego de eso se aplica un algoritmo de enrutamiento.

- ¿Qué rol juega la capa de red en los problemas de congestión de la capa de transporte?

resumido: controla buffers de los routers y avisa al emisor q se calme. Si no logra nada, se encarga de eliminar paquetes.

Cuando hay demasiados paquetes presentes en una red (o en una parte de ella), hay retardo o pérdida en los paquetes y se degrada el desempeño. Esta situación se llama congestión.
La congestión ocurre en los enrutadores, por lo que se detecta en la capa de red, esta capa es quien la experimenta en forma directa y en última instancia debe determinar qué hacer con los paquetes sobrantes. Sin embargo, se produce debido al tráfico que la capa de transporte envía a la red. La única manera efectiva de controlar la congestión es que los protocolos de transporte envíen paquetes a la red con más lentitud. Por lo que el proceso de controlar la congestión para evitar este problema es la responsabilidad combinada de las capas de red y de transporte.
En Internet y en muchas otras redes de computadoras, los emisores ajustan sus transmisiones para enviar tanto tráfico como la red pueda distribuir. En este escenario, la red aspira a operar justo antes de que comience la congestión. Cuando la congestión es inminente, debe pedir a los emisores que reduzcan sus transmisiones y su velocidad. Ahora veamos algunos métodos para regular el tráfico que se pueden usar en las redes de datagramas y en las de circuitos virtuales. Cada método debe resolver dos problemas.
En primer lugar, los enrutadores deben determinar cuando se acerca la congestión, siendo lo ideal antes de que haya llegado. Para ello, cada enrutador puede monitorear en forma continua los recursos que utiliza. En el laboratorio nosotros monitoreamos el uso del buffer.
El segundo problema es que los enrutadores deben entregar una retroalimentación oportuna a los emisores que provocan la congestión. Ésta se experimenta en la red, pero para aliviarla se requiere una acción de parte de los emisores que están usando la red. Para entregar la retroalimentación, el enrutador debe identificar a los emisores apropiados. Después, debe advertirles con cuidado, sin enviar más paquetes a la red que ya está congestionada.
Cuando ninguno de los métodos anteriores elimina la congestión, los enrutadores pueden sacar la artillería pesada: el desprendimiento de carga, que es una manera rebuscada de decir que cuando se inunda a los enrutadores con paquetes que no pueden manejar, simplemente se tiran.

- ¿Cuales son las posibles opciones cuando un paquete llega a un router/nodo que no tiene una ruta adecuada para el destino?

Si eso ocurre supongo que o no existe el destino, o salio mal el algoritmo de enrutamiento para determinar una ruta para ese destino,

Descarte del paquete: El router o nodo puede descartar el paquete y no hacer nada más con él. Esto podría ocurrir si el router no tiene información de enrutamiento para la dirección de destino y no puede determinar una ruta alternativa.

Generación de un mensaje de error ICMP: En lugar de descartar el paquete, el router puede generar un mensaje de error ICMP (Protocolo de mensajes de control de Internet) para informar al remitente que no se pudo encontrar una ruta adecuada para el destino. El mensaje de error ICMP puede contener información específica sobre el problema, como la falta de una ruta válida.
Pensar esta solucion si el paquete ya paso varios routers y luego no supo que hacer, al parecer los router anteriores supieron q hacer.

http://librosnetworking.blogspot.com/2012/04/la-ruta-por-defecto.html
https://ccnadesdecero.es/decisiones-de-routing-mejor-ruta/

La predeterminada (“gateway de último recurso”) es la que usa el router cuando no existe ninguna otra ruta conocida para una red de destino.

Una ruta predeterminada identifica un nodo específico como el siguiente salto al que viajan los paquetes y, a continuación, éstos prosiguen su viaje hasta el destino final en una red diferente. Los paquetes toman la ruta predeterminada cuando no hay ninguna otra ruta (más específica) que coincida con la dirección IP destino.

La puerta de enlace predeterminada (default gateway) es la ruta predeterminada o ruta por defecto que se le asigna a un equipo y tiene como función enviar cualquier paquete del que no conozca por cuál interfaz enviarlo y no esté definido en las rutas del equipo, enviando el paquete por la ruta predeterminada.