Capítulo 3

Medioambiente de Simulación

El medioambiente utilizado para implementar la simulación fue Ns[13] – Network Simulator – versión 2.0, un simulador orientado a objetos basado en eventos discretos cuyo objetivo es la simulación de redes de comunicación. Ns está actualmente en desarrollo en el marco del proyecto VINT (Virtual InterNetwork Testbed, USC/ISI/, UC Berkeley y Xerox Parc).

Ns se encuentra implementado en C++ y en Otcl [14][15], extensión de Tcl[16] orientada a objetos. Soporta de esta manera dos jerarquías de clases relacionadas, una de ellas en C++ (jerarquía compilada) y otra en Otcl (jerarquía interpretada). Desde el punto de vista del usuario, existe una relación uno a uno entre una clase de la jerarquía interpretada y una de la compilada. Una característica importante es que puede ampliarse la funcionalidad de las clases compiladas creando dinámicamente nuevos métodos en Otcl.

La clase *Simulator* es la que provee al usuario con elementos para configurar una determinada simulación y seleccionar el tipo de planificador de eventos (scheduler) a utilizar. En la actualidad existen cuatro tipos de planificadores: “linked-list”, utilizado en el presente desarrollo, “heap”, “calendar queue” y “real-time”. Cada uno de los tres primeros tipos de scheduler utiliza una estructura de datos diferente para el manejo de los eventos, cada uno de ellos presentando mejor performance en determinados casos. El último, aún en estado experimental, provee la sincronización entre los eventos manejados por el simulador y el tiempo real. Su objetivo es posibilitar la emulación, es decir, la capacidad para introducir el simulador en una red real.

A continuación se describen brevemente las clases que son utilizadas para la configuración de las simulaciones. Esta descripción sólo abarca los aspectos más importantes desde el punto de vista de un usuario[10], no pretendiendo profundizar en otros detalles.

Las clases que permiten configurar una topología sobre la cual se producirá la simulación, son principalmente *node*, cuyas instancias son nodos con la funcionalidad de routers y *link*, cuyas instancias son vínculos de comunicación..

3.1 Nodos

Un objeto de la clase *node*, representa la funcionalidad de un router (capacidad de reenvío de paquetes). Un nodo está conectado por objetos de la clase *link* a otros nodos, pudiendo a su vez soportar procesos (protocolos tales como tcp, udp, procesos de usuarios, etc.). Estos procesos, son derivados de la clase *Agent*, que se describe más adelante.

La dinámica de un nodo es simple: recibe un paquete por uno de sus vínculos (*link*) o un paquete generado por alguno de los agentes que en él residen, y lo procesa de acuerdo a su dirección de destino[11], entregándolo al vínculo o al agente local que corresponda.[12]

[10] Por usuario debe entendrse el programador de un script de simulación en Otcl.

Si bien existen métodos de la clase *node* relativos a su configuración, no se describen aquí por razones de brevedad.

En la figura 3.1 se muestra la estructura de un nodo sin capacidad multicast[13]. Un paquete recibido por el nodo es procesado en orden por un conjunto determinado de objetos[14]. El primero de ellos está indicado por la variable de instancia *_entry* y en el caso unicast, corresponde a un objeto de la clase *AddressClassifier*, derivada de la clase *Classifier*[15]. El objeto *AddressClassifier* del nodo (indicado por el contenido de la variable *_classifier*) analiza su "dirección", es decir, la identificación del nodo de destino que lleva dicho paquete y en base a ella lo envía a un vínculo (si la dirección de destino no coincide con la del nodo) o al objeto encargado de determinar el agente local a quien va destinado el paquete, en función del campo port. Este objeto, el *Port Classifier* del nodo, derivado de la clase *Classiffier,* está indicado por la variable de instancia *_dmux*.

Un vínculo será el encargado de entregar el paquete al próximo nodo, mientras que el agente representa un proceso que tendrá a su cargo el tratamiento del contenido del paquete (por ejemplo, una aplicación).

Se ha descripto en los párrafos anteriores el recorrido de un paquete dentro del nodo. Debe notarse que las transferencias del paquete no implican la generación de eventos para el

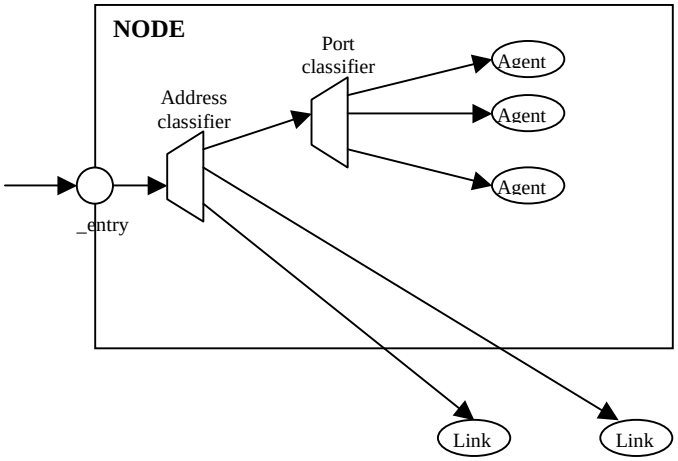

Figura 3.1. Estructura de un nodo sin capacidad multicast

planificador de eventos; Ns asume que los tiempos de proceso son nulos. Más adelante se verá la generación de eventos a cargo de un componente de los objetos *link*, que implica demoras.

---

[11] En Ns las direcciones se componen, al igual que en la jerarquía TCP/IP, de una dirección de nodo (equivalente a una dirección IP) y una identificación del agente (similar al port).

[12] En el caso que nos ocupa, multicast, el nodo podrá enviar el paquete por más de un link de salida, de acuerdo al árbol de distribución.

[13] Un nodo no posee por defecto capacidad multicast en Ns. Es posible configurar la totalidad de los nodos (no algunos) de la simulación con dicha capacidad, indicándolo explícitamente al Simulador.

[14] Este proceso es instantáneo desde el punto de vista de la simulación, mientras no se generen eventos.

[15] Un classifier, en general, es un objeto que analiza una cierta parte del paquete recibido, y en base a su contenido, envía el paquete a un objeto determinado, el siguiente en la secuencia que procesará dicho paquete.

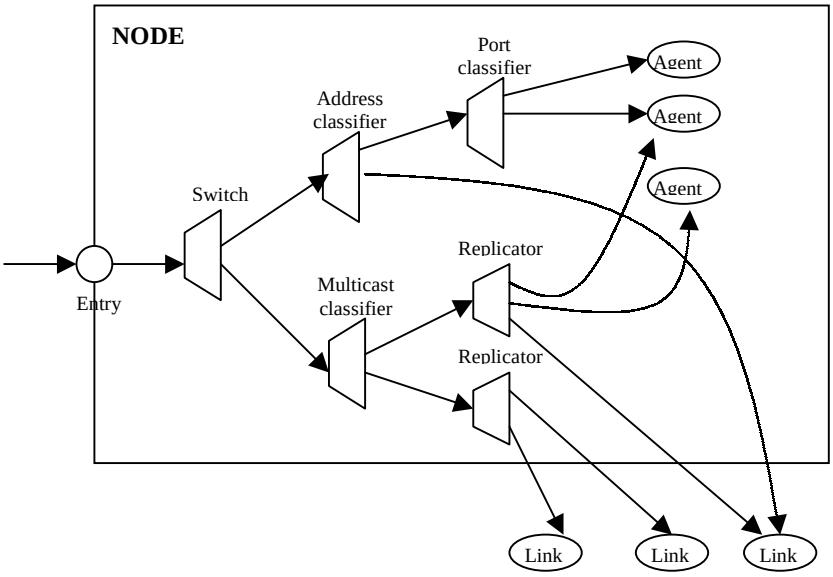

Figura 3.2. Estructura de un nodo con capacidad multicast

En el caso de nodos con capacidad multicast, cuya estructura se presenta en la figura 3.2, se agregan elementos adicionales.

El objeto que constituye la entrada al nodo, es en este caso un C*lassifier* que sólo distingue los paquetes en función del primer bit de su dirección de destino (*Switch*). En el caso de un paquete con dirección unicast (primer bit en 0) es pasado al objeto *AddressClassifier*, produciendo en el nodo un comportamiento como el descripto para el caso unicast. Si en cambio la dirección de destino del paquete es multicast (primer bit en 1), el *Switch* entrega el paquete a un C*lassiffier* (*Multiclassifier*) que clasifica los paquetes según su dirección de origen y de destino[16]. De acuerdo a estas direcciones, el paquete es entregado a un objeto *replicator*[17], cuya función es crear copias del paquete para distribuirlas entre los posibles receptores, ya sea vínculos para su reenvío o agentes locales miembros del grupo multicast.

3.2 Vínculos

Así como los nodos están compuestos de objetos *classifier*, que realizan una selección del paquete en base al contenido de algunos de sus campos, los vínculos están compuestos de objetos derivados de la clase *connector*. Básicamente, un objeto de esta clase recibe un paquete, realiza una alguna función sobre el mismo, y luego lo entrega al próximo objeto en la secuencia (*target_*) o lo descarta (*drop_target_*).

[16] La clasificación del paquete por dirección origen y destino corresponde a un paradigma source based trees. Como se explica en el capítulo correspondiente a la implementación, esta característica obligó a tener en cuenta consideraciones adicionales para el caso que nos ocupa, shared trees. Además de la clasificación por dirección y destino, el soporte provisto por el simulador provee un chequeo de interfaz de entrada (upstream) que se detalla en el capítulo correspondiente a la implementación.

[17] Un *replicator*, si bien se deriva de la clase *classifier*, no realiza una clasificación del paquete, sino que sólo lo replica. Se encuentra implementado de esta manera sólo por la conveniencia de utilizar las tablas de slots de la clase *classifier*.

De acuerdo a las características de un vínculo, éste estará compuesto de varios conectores. Las variables de instancia del objeto, contienen referencias a su entrada (*head_*), al elemento principal de su cola de paquetes (*queue_*), al elemento que realmente modela las características de la transmisión: demora de propagación y velocidad de transmisión (*link_*), al elemento que maneja el time to live (ttl) en cada paquete, y al primer elemento de la cola de elementos que procesan el descarte de paquetes (*drophead_*).

Los conectores que pueden conformar un link, son los siguientes:

- *Networkinterface*: su función es rotular los paquetes con una identificación de interfaz de arribo. Esta característica es utilizada por los protocolos de ruteo multicast. Debe habilitarse dando valor uno a la variable de clase *NumberInterfaces_* del objeto Simulator.
- *DelayLink*: Tiene por objeto producir las demoras propias del vínculo de transmisión, dependiendo de la demora de propagación de la señal, la velocidad de transmisión y la longitud de los paquetes. Normalmente, recibe los paquetes y genera un evento que el planificador de eventos entregará, en el tiempo adecuado, al siguiente conector. En casos de vínculos dinámicos, forma una cola interna con los paquetes previendo casos de caída del vínculo, en los cuales debe descartarse los paquetes en la cola.
- *DynamicLink*: Es un conector cuyo objeto es permitir o no el paso de los paquetes según el estado del vínculo (funcionando o caído). Para configurar el estado de un vínculo, debe utilizarse la capacidad de caída dinámica de vínculos provista por el

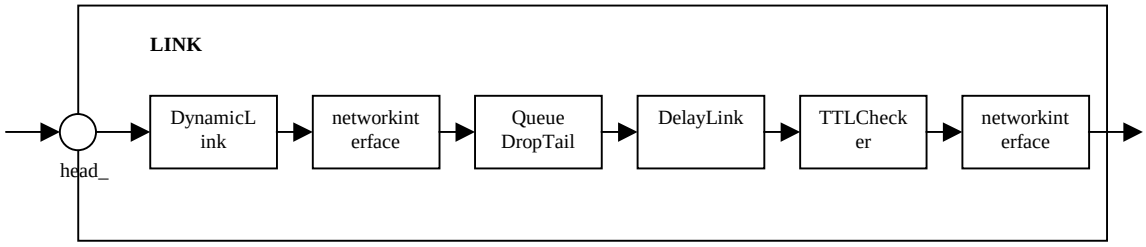

Fig. 3.3. Componentes de un link dinámico sin capacidad de tracing.

simulador.

- *Queues*: Son conectores utilizados para modelar las colas asociadas a los vínculos en los routers. Existen varios tipos, cada uno de ellos con una política de descarte de paquetes diferente.
- *TTLChecker*: Es un conector que decrementa el campo ttl de cada paquete, descartándolo en caso de que llegue a cero.
- *Trace*: Se proveen conectores destinados a realizar el seguimiento (trace) de los paquetes. Estos objetos se insertan antes y después de la cola (*Queue*) que representa los buffers del router para poder realizar el seguimiento de los paquetes que llegan y son enviados o descartados.

En la figura 3.3 se muestra la estructura de un vínculo dinámico que no incluye los objetos para realizar tracing. El primer conector, *Dynamiclink*, determina si el vínculo está o no operativo, descartando paquetes en este último caso. Los objetos *networkinterface* se ocupan de colocar en el encabezamiento del paquete, la identificación de la interfaz. El objeto *Queue*

DropTail, representa los buffers del router, implementándose aquí la política de descarte de paquetes. *DelayLink* produce las demoras de transmisión e interactúa con el planificador de eventos, registrando el evento correspondiente a la llegada del paquete a destino y a su vez el que corresponde a la liberación del buffer que ocupaba en el router de origen (una demora de propagación antes). Por último, *TTLChecker* controla el campo ttl del paquete para descartarlo en caso de que llegue a cero.

Las características vistas corresponden a la clase *Link*; de ella se deriva la clase *simplex-link*, que une dos nodos a través de un vínculo unidireccional, y la clase *duplex-link*, que determina la unión de dos nodos a través de un vínculo bidireccional.

3.3 Agentes

Finalizada la descripción de los elementos básicos que permiten definir la topología, nodos y vínculos, resta describir la clase base (*Agent*) de la que se derivan los elementos activos en la simulación: los procesos residentes en los nodos. Estos pueden ser procesos de soporte, tales como protocolos, o simplemente aplicaciones.

Un agente incluye estado acerca de su dirección (identificación del nodo y del port dentro del nodo) y de la dirección del agente al cual enviará los paquetes que genere. La dirección propia la adquiere al asociarlo a un nodo a través del método *attach-agent* provisto por la clase *Simulator*, mientras que la dirección de destino es configurada a través del método *connect* [18]o asignando un valor específico a la variable *dst_* del agente.

Se define además el valor ttl que el agente asignará a cada paquete generado, la longitud en bytes del paquete (utilizada sólo para el cálculo del tiempo de transmisión por los objetos DelayLink), la prioridad IP, un identificador de flujo también a nivel IP, y el tipo de paquete.

Los métodos provistos por la clase permiten generar y enviar un paquete y recibir paquetes destinados al agente.

Entre los agentes provistos se encuentran varias clases de agentes TCP, agentes CBR, UDP, un agente para envío de mensajes, y el agente nulo cuya función es descartar paquetes.

3.4 Ejemplo

En el siguiente ejemplo se muestra una simulación de una topología simple, compuesta de tres nodos unidos por dos vínculos bidireccionales de 1Mbps con una demora de propagación de 5 milisegundos y una política de descarte de paquetes específica (DropTail). En los nodos n1 y n2 hay dos procesos que envían información a una tasa constante a los agentes correspondientes en n0.

```
#Instanciacion del simulador
set ns [new Simulator]

#Facilidad de trace
set f  [open salida.tr w]
$ns trace-all $f

#Nodos
set n0 [$ns node]
```

[18] No debe asociarse la invocación del método *connect* entre dos agentes con el establecimiento de una conexión entre ellos. Sólo es una asociación de direcciones. Es así por ejemplo que dos agentes UDP deben ser relacionados por el método connect para que puedan comunicarse.

```
set n1 [$ns node]
set n2 [$ns node]

#Vinculos
$ns duplex-link $n0 $n1 1Mb 5ms DropTail
$ns duplex-link $n0 $n2 1Mb 5ms DropTail

#Agentes
set emisor1 [new Agent/CBR]
set emisor2 [new Agent/CBR]
set receptor1 [new Agent/Null]
set receptor2 [new Agent/Null]

#Asociación de agentes y nodos
$ns attach-agent $n0 $receptor1
$ns attach-agent $n0 $receptor2
$ns attach-agent $n1 $emisor1
$ns attach-agent $n2 $emisor2

#Conexiones entre los agentes
$ns connect $emisor1 $receptor1
$ns connect $emisor2 $receptor2

#Eventos al simulador (arranque de agentes)
$ns at 1.0 “$emisor1 start”
$ns at 1.1 “$emisor2 start”
$ns at 5.0 “$ns halt”

$ns run
```

Los agentes CBR generan paquetes a una tasa constante desde que son activados (*start*). Ambos toman valores por defecto (longitud de paquetes y tasa de envío). A través del método *attach-agent* se asocian los agentes (procesos) a los nodos.

Como fue mencionado, las “conexiones” entre los pares de agentes, no tienen el significado de una comunicación orientada a conexión, como es por ejemplo TCP. El método *connect* provisto por el simulador, sólo tiene como consecuencia dar valor a las variables de instancia *dst_* en ambos agentes, para que envíen sus paquetes a la dirección indicada.

El comienzo de la actividad de los agentes (*start*) y la indicación al simulador para que detenga la simulación, se especifican a través de la generación de eventos específicos para el planificador de eventos utilizando el método “*at*” provisto por el simulador. Notar que en el transcurso de la simulación, serán generados otros eventos, por medios diferentes a “*at*” por los diferentes componentes (por ejemplo, objetos *DelayLink*).

Por último, se indica al simulador que comience la corrida, a través del método “*run*”. La salida de resultados se especifica a través de sentencias trace al simulador (en este caso *trace-all*) indicando que la salida se dirija al archivo “salida.tr”.

3.5 Otras facilidades provistas por el simulador

En las secciones anteriores fueron presentados los componentes básicos del simulador, de manera de dar una idea general respecto a su estructura y funcionamiento. Por razones de

extensión, se describe brevemente en los siguientes párrafos otras características de importancia de Ns.

- **Paquetes**: Los paquetes constituyen eventos. Contienen un encabezamiento común (longitud, tiempo, uid, rótulo de interfaz), un encabezamiento definido según cada protocolo y opcionalmente datos.
- **Modelos de error**: Simula errores a nivel data-link, marcando el paquete como erróneo a través de flags del encabezamiento o descartando el paquete (envío al drop-target). La generación de errores puede hacerse a través de modelos simples (tasa de errores en paquetes) o modelos más complejos. La granularidad puede ir desde paquetes hasta bits.
- **Generación de tráfico**:Se provee la facilidad de generación de tráfico, determinando tasas de envío y longitud de paquetes. La generación y envío de paquetes está a cargo de agentes UDP. Como clases derivadas de *TrafficGenerator*, actualmente se provee *EXPOO_Source* que genera tráfico de acuerdo períodos on/off de acuerdo a una distribución exponencial y *POO_Source*, igual que la anterior pero utilizando una distribución Pareto.
- **Administración de buffers**: Se refiere a la política que se utiliza para regular qué paquetes ocuparán las colas que representan buffers en los routers. Puede ser FIFO (DropTail) o RED (Random Early Detection).
- **Planificación (scheduling) de paquetes**: Se refiere al mecanismo para seleccionar la política de atención de los paquetes en los buffers (en las colas que representan los buffers). Puede ser FIFO, variantes de Fair Queueing, incluyendo Wiighted Fair Queueing, etc.
- **Facilidades de trace**: Las facilidades de tracing provistas por el simulador, permiten almacenar datos en archivos seleccionados por el usuario, para realizar posteriormente a la simulación, el análisis de los mismos. Ns incluye objetos de la clase *Trace*, que permiten registrar cada paquete individualmente, cuando arriba a un nodo, cuando deja el nodo o es descartado, y objetos de la clase *Monitor*, que permiten registrar contadores respecto a la totalidad o según el tipo de paquetes (por ejemplo contador de paquetes que arriban a un nodo). Tanto los objetos *Trace* como los *Monitor* utilizan el encabezamiento común que tienen los paquetes.
- **Redes locales**: Permite especificar características de redes locales, capturando el comportamiento de los tres niveles inferiores: nivel físico, subnivel de acceso al medio y subnivel LLC. Se provee el soporte para la definición de diferentes protocolos a nivel MAC y LLC.
- **Ruteo unicast**: Las facilidades provistas para ruteo unicast permiten calcular las tablas de ruteo en los nodos definidos.
  El ruteo centralizado consiste en un cálculo de rutas realizado externamente a la simulación. Se calcula los caminos mínimos (algoritmo SPF). Se proveen dos alternativas: ruteo estático, calculado sólo al comienzo de la simulación y Ruteo por Sesión, en la cual se calculan las rutas al comienzo de la simulación y cada vez que se produce un cambio en la topología.
  El ruteo detallado consiste de un cálculo de rutas realizado como parte de la simulación. En la actualidad, está implementado el ruteo Distance Vector, pudiendo configurarse el tiempo de intercambio de tablas, el uso de split horizon with poison reverse y triggered updates. El uso de ruteo detallado en topologías con cantidad considerable de nodos puede dar lugar a tiempos más elevados en las corridas.
- **Ruteo multicast**: La estructura de un nodo con capacidad multicast fue descripta antes. Las clases que soportan el desarrollo de protocolos multicast serán descriptas en el capítulo correspondiente a la implementación. Las facilidades multicast que actualmente ofrece Ns consisten de protocolos centralizados (CtrMcast: Sparse mode y Dense mode) y protocolos detallados, (Dynamic Dense Mode y Pim modo denso). En la actualidad se encuentra en desarrollo PIM modo denso y sparse. En el presente trabajo se implementó CBTv2 detallado.

3.6 Resumen

En este capítulo fueron presentadas las características más relevantes del medioambiente de simulación utilizado. Este medioambiente permite la rápida y sencilla generación de topologías y agentes utilizando las facilidades provistas como así también la incorporación de nuevos elementos a través de la creación o extensión de clases compiladas o interpretadas. En el presente desarrollo se extendieron algunas clases ya provistas y se incorporaron nuevas clases íntegramente en Otcl a efectos de obtener un código fácilmente portable al no ser necesaria la recompilación del simulador.