Model Checking: Un Modelo del Protocolo Gnutella. Simulación y Verificación usando SPIN *

María Cecilia De Vito[1], Germán L. Osella Massa[2]
{cdevito, gosella}@lidi.info.unlp.edu.ar

Instituto de Investigación en Informática (III-LIDI
Facultad de Informática. Universidad Nacional de La Plata.
La Plata, 1900, Argentina

Abstract

This paper presents a specification of the Gnutella communication protocol for Peer-to-Peer networks. A model of this protocol has been made using SPIN, an automatic verification system which includes model-checking algorithms. Two models are presented: a complete one, suitable for simulating the behavior of a Gnutella node; and a limited one, which allows checking whether certain properties expressed in LTL (Linear Temporal Logic) are true. As example, two formulas have been checked in this model to verify them.

Keywords: Distributed systems, Model Checking, Simulation, Verification, P2P protocols.

Resumen

En este trabajo se presenta una especificación del protocolo de comunicación Gnutella para redes Peer-to-Peer. Se realizó un modelo de este protocolo utilizando SPIN, un sistema de verificación automática que incluye algoritmos de comprobación de modelos. Se presentan dos modelos: uno completo, apropiado para la simulación del comportamiento de un nodo Gnutella, y otro acotado, que permite comprobar si determinadas propiedades expresadas en LTL (Lógica Temporal Lineal) son ciertas. A manera de ejemplo, dos fórmulas fueron comprobadas en el modelo con el fin de verificarlas.

Palabras Clave: Sistemas Distribuidos, Comprobación de Modelos, Simulación, Verificación, Protocolos P2P.

I WORKSHOP DE ARQUITECTURA, REDES Y SISTEMAS OPERATIVOS (WARSO)

[1] Becaria de Doctorado del Proyecto PAV 076 - Jefe de Trabajos Prácticos. Facultad de Informática UNLP. cdevito@lidi.info.unlp.edu.ar
[2] Becario de Doctorado del CONICET. Ayudante Diplomado. Facultad de Informática UNLP. gosella@lidi.info.unlp.edu.ar
**** Trabajo dirigido por la Lic. Patricia Pesado y el Ing. Armando De Giusti.***

1. INTRODUCCIÓN

Los campos de principal aplicación de los métodos formales son la especificación y la verificación [11].

La **especificación** es el proceso de describir un sistema y las propiedades que se desea que tenga. De la correcta especificación de un sistema depende en gran medida el éxito o el fracaso en su desarrollo.

La especificación formal se basa en un lenguaje con una sintaxis y una semántica matemáticamente definidas. En estos métodos es común la utilización de los conceptos matemáticos de abstracción y composición.

La **verificación** es el proceso que permite comprobar, en mayor o menor medida, si un sistema cumple las propiedades deseadas en él. En la verificación formal nos encontramos con dos aproximaciones: Comprobación de Modelos y Prueba de Teoremas.

La comprobación de modelos [10] es una técnica que consiste en construir un modelo finito de un sistema y comprobar si determinadas propiedades deseadas del sistema se cumplen en el modelo (Figura 1). Dicha comprobación es totalmente automática y rápida, pudiendo llegar a dar una respuesta en corto tiempo, si se toman los suficientes recaudos para evitar una explosión combinatoria de estados que haga inmanejable el problema.

Cabe destacar que la comprobación de modelos produce contraejemplos a partir de los cuales se podría analizar posibles errores subyacentes en el diseño. Esto resulta interesante ya que detectar errores de diseño en etapas tempranas del desarrollo de un sistema se traduce en un importante ahorro de esfuerzo y recursos. La verificación automática no está centrada en demostrar la corrección de un modelo, sino que busca encontrar errores lo más pronto posible durante la construcción del Sistema.

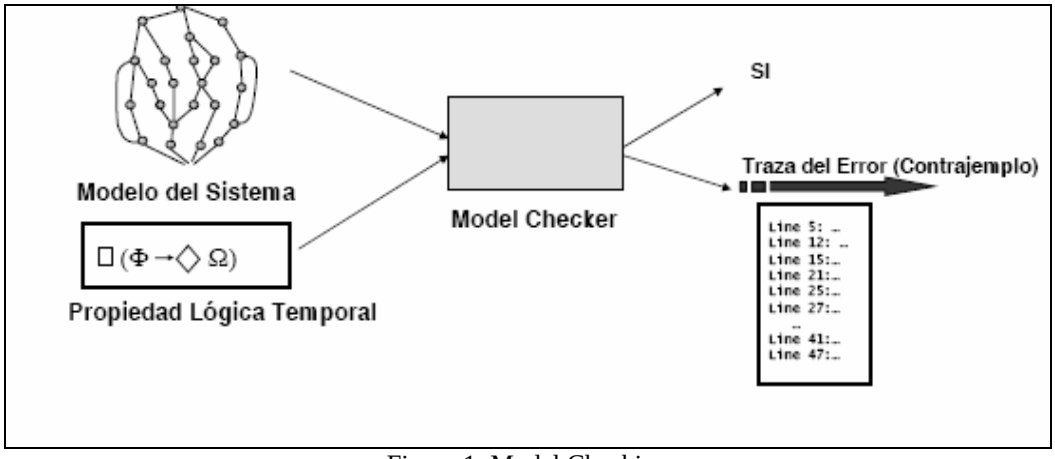

Figura 1: Model Checking

En comparación con la prueba de teoremas, la comprobación de modelos es totalmente automática y rápida, pudiendo llegar a dar una respuesta en cuestión de minutos. Presenta la desventaja de que se puede enfrentar a una explosión combinatoria que haga inmanejable el problema.

La comprobación de modelos puede ser utilizada para verificar especificaciones parciales, en cuyo caso nos proporciona una valiosa información sobre la corrección de un sistema, aun cuando éste no esté completamente especificado [13].

Dos comprobadores de modelos comúnmente usados son SPIN [1], [2] y SMV [3]. Se han estudiado estas herramientas explorando sus posibilidades en la verificación de propiedades en modelos de aplicaciones distribuidas.

SPIN usa la Lógica Temporal Lineal (LTL) [6] para generar un comprobador de modelos a partir de la especificación del sistema. El SMV usa la Lógica Temporal CTL (Computacional Tree Logic) con el mismo fin. SMV es una herramienta para comprobar sistemas desde sus modelos representados mediante autómatas finitos, contra especificaciones en Lógica Temporal CTL [3].

SPIN posee un lenguaje propio (PROMELA) para la especificación de Sistemas Concurrentes y Distribuidos, similar al lenguaje C con algunas propiedades de CSP (Communicating Sequential Processes) [12].

Las redes Peer-to-Peer (P2P) se basan en un concepto que ha existido desde hace muchos años, pero fue a finales de la década de los 90 cuando cobró popularidad por la creación de Napster. A través de esta herramienta, los usuarios compartían espacio de sus discos duros para distribuir archivos en la red; se conectaban a un servidor central que realizaba las búsquedas y posteriormente se establecía una conexión punto a punto para descargar la información. Poco tiempo después se entabló un juicio contra Napster por promover la distribución de archivos protegidos por derecho de autor y su desaparición prosperó debido a la arquitectura centralizada de la red.

Napster sentó el precedente y propició la evolución y creación de un gran número de herramientas P2P, como AudioGalaxy y Fastrack a cargo de programas como Morpheus, KaZaA y Grokster, Edonkey y su clon Emule, Gnutella y Imesh, entre otras.

Este tipo de sistemas ha ganado gran popularidad por la distribución descentralizada de archivos en Internet. En una red P2P no existen clientes ni servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red. Este modelo de red contrasta con el modelo cliente-servidor. Cualquier nodo puede iniciar, detener o completar una transacción.

Gnutella es un proyecto de software distribuido para crear un protocolo de red de distribución de archivos entre pares, sin un servidor central. El protocolo Gnutella define un conjunto de tipos de mensajes empleados para comunicar datos entre los nodos y un conjunto de reglas que gobiernan el intercambio de mensajes entre ellos.

Utilizar una herramienta como SPIN para modelar un protocolo de comunicación permite ganar confianza en él pudiendo asegurar su correcto funcionamiento. En [7] se puede encontrar un modelo del protocolo Gnutella en un lenguaje de especificación de protocolos de comunicación para sistemas multiagentes. SPIN se ha empleado para especificar otros tipos de protocolos [4] [5] y comprobar determinadas propiedades de ellos.

En este trabajo se presentan dos modelos del protocolo Gnutella utilizando SPIN: uno exhaustivo, que permite simular el comportamiento completo del protocolo y otro reducido, acotado a partir de una serie de suposiciones, con el objetivo de poder verificar propiedades en dicho modelo. Estas restricciones fueron impuestas para evitar una explosión de estados.

En la sección 2 se describe el protocolo Gnutella. En la sección 3 se presenta la herramienta SPIN utilizada para modelar, simular y demostrar propiedades. En la sección 4 se detalla el modelo construido para la simulación. En la sección 5 se describen las suposiciones hechas para poder obtener un modelo sobre el cual es posible demostrar propiedades descriptas en LTL. Como ejemplo, dos propiedades son demostradas sobre este modelo. Por ultimo, en la sección 6 se esbozan algunas conclusiones.

2. PROTOCOLO GNUTELLA

El modelo desarrollado en esta propuesta especifica el Protocolo Gnutella [8] [9]. El protocolo se define respecto a un Nodo de la Red al que se denomina “Nodo General”.

Gnutella ofrece como mínimo la funcionalidad necesaria para que un nodo pueda ser unido a la red P2P y abandonarla fácilmente. También, un nodo debe poder publicar la información que dispone para compartir en la red P2P. Por ultimo, un nodo debe ser capaz de buscar y obtener recursos de la red.

Son tres las funciones que pueden ser realizadas por un Nodo: **Conectar, Buscar y Transmitir.** Se ha utilizado la versión 0.4 del protocolo, la cual cuenta con 4 mensajes básicos: Ping, Pong, Query y QueryHit.

En esta versión, un mensaje comienza con una cabecera y le sigue una sección de los datos. Entre otros, la cabecera de un mensaje posee un campo *ID*, un campo que determina el tipo de datos contenidos en el mensaje y dos campos más, llamados *TTL* y *HOPS*.

El campo *ID*, que se utiliza como identificador de una transacción, es un número único en toda la red y debe ser preservado cuando se reenvían mensajes entre nodos. Es utilizado para detectar mensajes duplicados con el fin de reducir el tráfico innecesario en la red.

Los campos TTL y HOPS representan el tiempo de vida del mensaje y el número de saltos, respectivamente. El campo TTL se descuenta en 1 por cada salto o “hop”, es decir, cada vez que es retransmitido por un nodo. Por el contrario, el campo HOPS se incrementa en cada salto. Un mensaje se descarta cuando es recibido con TTL igual a 0. El campo TTL es establecido generalmente es 10.

En la función de **Conexión,** un Nodo utiliza un protocolo simple Ping/Pong con cada Nodo adyacente para localizar nuevos Nodos. El mecanismo para conocer a los nodos iniciales no es especificado en el protocolo. En general, los nodos iniciales son provistos por nodos Host Server Cache, conocidos de antemano. Este protocolo es mostrado en la figura 2.

Un Nodo emplea los mensajes Ping para buscar activamente otros nodos en la red. Existen dos tipos de mensajes Ping. Uno Directo (con *TTL*=1 y *HOPS*=0), que es empleado para conocer el estado de otro Nodo, el cual debe ser respondido con un Pong afirmando la existencia del receptor. Un mensaje Ping puede ser también de Navegación (con *TTL*=2 y *HOPS*=0), el cual debe ser respondido con mensajes Pong que contengan información de los nodos accesibles desde el receptor del mensaje.

En la función de **Búsqueda,** un Nodo puede iniciar la búsqueda de un recurso particular usando un mensaje Query. Este mensaje inunda la Red con el objetivo de encontrar dicho recurso, ya que el Query es enviado a cada Nodo adyacente activo al nodo que solicita el recurso. Si alguno de ellos posee la copia del requerimiento solicitado, éste enviará un mensaje de contestación QueryHit al Nodo solicitante. En caso contrario, este Nodo enviará un Query solicitando el recurso a todos sus Nodos accesibles activos (exceptuando al Nodo que envió el Query original) y así sucesivamente. Este protocolo Query/QueryHit es mostrado en la figura 3.

Debe notarse que el número de mensajes requeridos en la búsqueda crece de forma exponencial. De esta manera el protocolo es ineficiente, ya que una búsqueda puede demorar un tiempo considerable en ser completada y genera una importante cantidad de tráfico en la red.

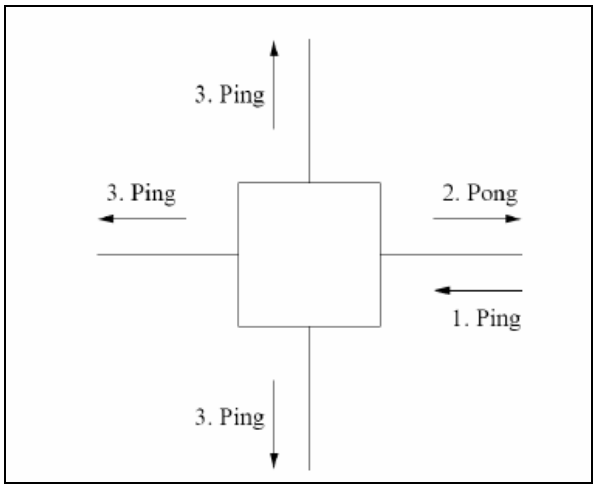

Figura 2: **Protocolo Ping/Pong**

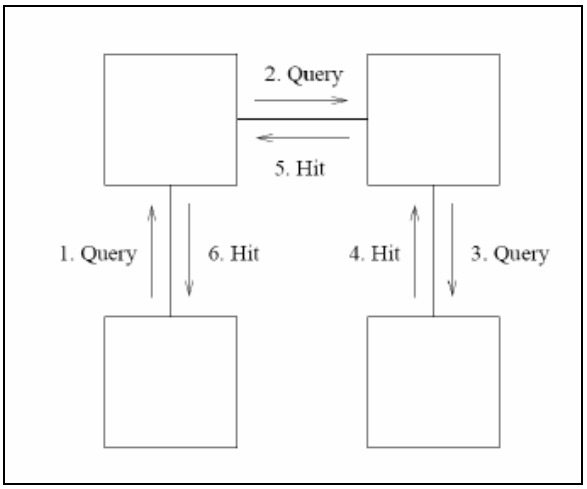

Figura 3: **Protocolo Query/QueryHits**

Cada nodo mantiene una lista de mensajes recibidos recientemente. Con el objetivo de evitar transmisiones duplicadas, cada mensaje Query identificado por su campo *ID* sólo se reenvía una única vez. Los mensajes QueryHit se encaminan sólo hacia el nodo del que se recibió el mensaje Query. Los mensajes QueryHit con un campo *ID* para el que no se haya visto un mensaje Query correspondiente se descartan, al igual que los mensajes duplicados.

Por último, en la función de **Transmisión**, que es realizada en caso de que el recurso haya sido localizado, el Nodo solicitante simplemente contacta al Nodo que posee el recurso buscado e inicia su descarga o transmisión.

3. LA HERRAMIENTA SPIN

Utilizada en la verificación de Software, SPIN resulta útil para encontrar errores en el diseño lógico de sistemas distribuidos. Esta herramienta comprueba la consistencia lógica de una especificación y puede ser utilizada como comprobador de modelos LTL.

SPIN trabaja *on-the-fly*, pues evita tener que construir un grafo de estados global o una estructura Kripke, como prerrequisito para la verificación de algunas de las propiedades del sistema.

Usa un lenguaje propio no deterministico para especificar descripciones de sistemas PROMELA (PROccess MEta LAnguage) Este lenguaje acepta especificaciones de corrección en la sintaxis del estándar LTL.

Además de permitir probar propiedades en LTL, también aporta información acerca de la aparición de bloqueos (deadlocks: “invalid end-states”), el cumplimiento de aserciones (por medio de la sentencia Promela assert), los estados locales de procesos, código muerto (“unreachable code”) y ciclos de no progreso (livelocks).

SPIN requiere el uso de LTL para expresar las propiedades a verificar. Las fórmulas son escriras a partir de proposiciones atómicas unidas por medio de conectores lógicos y los siguientes operadores temporales:

[] p, **siempre** p

<> p, **eventualmente** (en un futuro) p

X p, p es cierto en el **siguiente** estado

p U q, p es cierto **hasta que** q es cierto

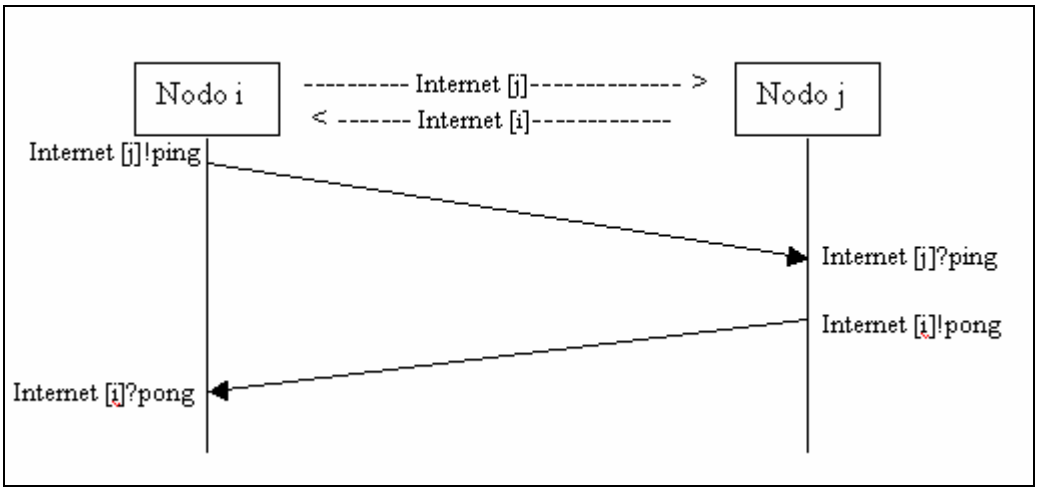

Figura 4: Ejemplo de comunicación entre nodos

Debe tenerse en cuenta que muchas de las propiedades de un sistema pueden ser especificadas y verificadas sin utilizar LTL.

El número de procesos puede aumentar o menguar dinámicamente, para lo que utiliza una técnica *rubber state vector*. Con respecto a la comunicación, soporta tanto *redezvous*, como el pase de mensajes con buffers, admitiendo tanto comunicaciones sincrónicas como a asincrónicas y a través de memoria compartida.

Proporciona simulación aleatoria, interactiva y guiada, así como técnicas de pruebas exhaustivas y parciales. Está pensada para enfrentarse a problemas de gran tamaño.

Para optimizar la verificación, utiliza técnicas de reducción de orden parcial y (opcionalmente) técnicas de almacenamiento BDD.

SPIN puede ser utilizado en 3 modos básicos:

- Como simulador, permitiendo crear rápidamente un prototipo del modelo mediante una simulación aleatoria, guiada o interactiva.
- Como un analizador exhaustivo del espacio de estados, capaz de demostrar de forma rigurosa la validez de los requerimientos de corrección. Aquí utiliza la teoría de las reducciones de orden parcial para optimizar la búsqueda.
- Como un analizador de un espacio de estados-bit puede validar, incluso sistemas de protocolos de gran tamaño, con la máxima cobertura del espacio de estados.

En lo referente a la descripción de un Sistema Concurrente en PROMELA, ésta está formada por uno o más procesos definidos por el usuario o definiciones de tipos de procesos y, al menos, una instanciación de proceso. Los procesos pueden ser creados usando la sentencia run o anteponiendo “active” en la declaración del proceso.

Los procesos definidos por proctype se ejecutan concurrentemente y en forma simultanea con otros procesos. Existe un proceso especial denominado **init** con el cual la ejecución de un programa es iniciada. El estado de un proceso se define por su identificador (p_id) y el contenido de sus variables locales.

Las sentencias definen el cuerpo de un proceso. Una sentencia es ejecutable si puede ser ejecutada inmediatamente o bien puede estar bloqueada en caso de que la sentencia no se pueda ejecutar. Existen otras sentencias que dependen del estado global del modelo PROMELA.

4. DESCRIPCIÓN DEL MODELO

Para modelar la red P2P, se utilizó un vector de canales denominado Internet. Cada canal posee una capacidad de *NumMensajes*. Un nodo recibe mensajes a través de un único canal de ese vector y se comunica con los otros Nodos vía los canales restantes. Se asocia la dirección IP de un Nodo con el índice de este vector que contiene el canal desde cual se reciben los mensajes dirigidos a este nodo (Figura 4). La cantidad de nodos en la red es configurable. Existe un Nodo adicional denominado Host Cache que tiene un comportamiento diferente al resto de los nodos. Dicho nodo tiene asignada por defecto la dirección de IP 0 (y por lo tanto su canal reside en la posición 0 del vector Internet). La función del Host Cache en recolectar y comunicar al resto de los nodos las direcciones IP de nodos activos. El Host Cache mantiene una lista de direcciones activas, que se actualiza al recibir mensajes del resto de los Nodos. Este nodo es creado ni bien comienza el proceso **init**, antes que cualquier otro nodo, ya que todos los demás lo buscarán al comenzar su ejecución. Este Nodo no participa en la búsqueda de recursos.

Cada mensaje del protocolo es enviado a través de un canal incluyendo los siguientes campos: Tipo de Mensaje, ID de Transacción, TTL, Hops, la dirección IP de nodo que envió el mensaje y (si es necesario) el recurso buscado. La Red P2P es modelada en PROMELA como se muestra a continuación.

```
mtype = {ping, pong, query, queryHit};
chan Internet[CantNodos] = [NumMensajes] of {mtype, int, byte, byte, byte, byte};
```

Como se mencionó anteriormente, el protocolo Gnutella se define respecto a un Nodo de la red al que en este modelo se denomina **Nodo General**. Dicho nodo es representado por un proceso para que sea capaz de ejecutarse concurrentemente junto a los otros nodos del modelo. Cuando un nodo es creado, recibe como parámetro la dirección IP que le ha sido asignada.

```
proctype NodoGeneral(byte miIP) { ... }
```

Un nodo almacena localmente las direcciones IP de sus nodos adyacentes, junto con el estado de cada nodo (activo/inactivo). El número de nodos adyacentes está acotado. La lista de Nodos Adyacentes a un Nodo es representada por:

```
typedef DatosNodoAdy {
  byte IP;
  bool Activo;
  byte TiempoUltMsjRecibido;
};
DatosNodoAdy NodosAdyacentes[MaxNumeroNodosAdy];
```

El contador TiempoUltMsjRecibido es usado para determinar si un nodo está activo o si es necesario mandarle un ping para determinar su estado.

Cada Nodo también cuenta con un vector de recursos (representados por valores booleanos), el cual es inicializado al comenzar la ejecución del nodo general asignando valores a sus elementos de manera aleatoria. Además, cada nodo mantiene una lista circular actualizada con los ID de las últimas transacciones procesadas por él junto con la dirección IP del nodo emisor de dicha transacción. Esta última lista es usada para evitar el trafico de mensajes repetidos y poder encaminar (en caso de ser necesario) un mensaje QueryHit al nodo que solicitó el recurso.

Se utiliza una variable global para calcular en forma atómica el campo ID contenido en los mensajes, de la forma mostrada a continuación:

```
inline get_trans(x) {
  atomic {
    x = idTrans;
    idTrans++;
  }
}
```

Al comenzar su ejecución, un nodo General envía un ping de Navegación al Host Cache e inicializa sus recursos. Luego, pasa a un estado de escucha donde recibe mensajes desde su canal y actúa de acuerdo al tipo de mensaje recibido. Antes de procesar cada mensaje, el nodo actualiza su lista de nodos adyacentes, determinando si alguno de ellos debe ser eliminado en función del tiempo transcurrido desde el último mensaje recibido desde cada nodo. En caso de ser eliminado, el espacio liberado por ese nodo podrá ser usado para almacenar la información de otro nuevo nodo adyacente.

Si en un momento dado, los nodos adyacentes no superan un número determinado, el nodo intentará buscar nuevos nodos activos utilizando el protocolo Ping/Pong descrito anteriormente. La estructura de un nodo general es presentada en la Figura 5.

El nodo se comporta de acuerdo al tipo de mensaje recibido, descartando el mensaje si no cumple con la condición TTL > 0.

En caso de recibir un Ping, los campos *TTL* y *Hops* son evaluados para determinar si es un Ping Directo o de Navegación. En caso de ser Directo, simplemente envía un mensaje Pong con el mismo valor de *estaTrans*, la dirección de IP *miIP* y los campos *TTL* y *Hops* actualizados. Si el mensaje entrante fuera de Navegación, se le enviará un mensaje Pong con la dirección IP de cada Nodo Activo Adyacente, con el identificador de transacción *estaTrans* y los campos *TTL* y *Hops* actualizados.

En caso de recibir un mensaje Pong, se debe actualizar la lista de Nodos Adyacentes. Primero, se determina si la dirección IP recibida es conocida, dejando al emisor nodo en estado activo si así lo fuera. Si no es conocida, se intenta agregar a este nodo como un nuevo Nodo Adyacente, ocupando el lugar del primer nodo inactivo. En caso de no existir nodos inactivos el mensaje Pong es ignorado.

En caso de que el mensaje entrante sea un Query, se debe buscar en la lista las últimas transacciones procesadas. En caso de no haber visto la transacción anteriormente, el Nodo en primer lugar verifica si posee el recurso requerido, en cuyo caso responderá de manera inmediata con un mensaje QueryHit. En caso de no disponer del recurso, el nodo propaga el mensaje Query a todos sus nodos adyacentes activos. En ambos casos, se almacena la dirección IP del nodo emisor y el identificador de la transacción del mensaje en la lista de transacciones procesadas.

Al recibir un QueryHit, se debe recorrer la lista de últimas transacciones para descubrir la dirección IP del nodo que envió el Query solicitando originalmente el recurso. Esta búsqueda es llevada a cabo con el número de transacción que contiene el mensaje recibido. Una vez encontrada la dirección IP, se le envía un mensaje QueryHit con la dirección IP del nodo que envió el QueryHit entrante, es decir, el nodo que posee el recurso.

Por último, el proceso **init** como ya se mencionó anteriormente, crea en primer lugar al Proceso Host Cache y luego crea a los Procesos Generales. Una vez creados los procesos, se envía un Query a un nodo arbitrario, con un recurso elegido al azar para iniciar la simulación.

```
/* Tipo de proceso que modela el comportamiento de un nodo */
/* en una red Gnutella. El proceso recibe como parámetro   */
/* la dirección IP con la que va a acceder a la red para   */
/* recibir los mensajes que se le envíen.                  */

proctype NodoGeneral(byte miIP)
            :
/* Inicializa los recursos */
            :
/* Envía un ping de Navegación al Host Cache para obtener nodos adyacentes */
Internet[0]!ping(trans, 2, 0, miIP)

do

  /* Actualiza sus nodos adyacentes en función del tiempo que transcurrió */
  /* desde el último mensaje recibido de cada nodo                        */
            :
  if
  :: nempty(Internet[miIP]) ->
    /* El nodo posee mensajes entrantes. Lo recibe uno y actúa según el tipo */
    if
    /*Recibe y procesa un mensaje Ping*/
    :: Internet[miIP]??ping(estaTrans, TTL, Hops, ipNodo, NumRecurso) ->
            :
    /* Recibe y procesa un mensaje pong */
    :: Internet[miIP]??pong(estaTrans, TTL, Hops, ipNodo) ->
            :
    /* Recibe y procesa un mensaje Query */
    :: Internet[miIP]??query(estaTrans, TTL, Hops, ipNodo, NumRecurso) ->
            :
    /* Recibe y procesa un mensaje QueryHit */
    :: Internet[miIP]??queryHit(estaTrans, TTL, Hops, ipNodo, NumRecurso) ->
            :
  :: empty(Internet[miIP]) ->
    /* No hay mensajes para procesar. Si quedan pocos nodos adyacentes */
    /* utiliza el protocolo Ping/Pong para conseguir más               */
  fi
od
```

Figura 5: Estructura de un nodo general

5. SIMULACIÓN Y VERIFICACIÓN

El modelo presentado en la sección anterior funciona según las expectativas esperadas para simulaciones. Para probar la veracidad de propiedades con este modelo usando SPIN, se obtuvo como resultado una explosión de estados que no permitió finalizar la verificación.

La complejidad de este modelo depende de muchos factores: El número de procesos generales, el número de Nodos Adyacentes que puede registrar cada nodo, el número de recursos existentes en la red, la capacidad de los canales de comunicación y la cantidad de transacciones almacenadas en las listas locales de cada nodo. Como primera medida, con el objetivo de poder realizar verificaciones de propiedades, se asignaron valores mínimos a los parámetros antes mencionados, de acuerdo a sus características. A pesar de los cambios realizados, el número de estados alcanzables siguió siendo inmanejable.

En segundo lugar, se decidió comenzar a restringir el comportamiento del nodo general, empezando por acotar el protocolo Ping/Pong permitiendo únicamente el uso de Pings directos para determinar el estado de los nodos adyacentes. Al comenzar su ejecución, un nodo general envía un ping de navegación a Host Cache para obtener sus nodos Adyacentes iniciales. Luego, solo mensajes Ping directos podrán ser enviados. Los nodos adyacentes no serán actualizados cada vez que se termina de procesar un mensaje ya que el tiempo transcurrido desde la última comunicación no será registrado. En caso de recibir un mensaje Pong, en este modelo se actualiza la lista de Nodos Adyacentes de la siguiente forma: Si el nodo ya estaba registrado como adyacente, su estado debe quedar como activo. En caso no haber estado registrado, si existe algún nodo inactivo en la lista, se utiliza su lugar para guardar la dirección IP del nuevo nodo. En caso contrario, el mensaje es descartado.

A diferencia del modelo anterior, un nodo no enviará mensajes Ping para descubrir nuevos nodos en la red cuando tenga pocos nodos adyacentes.

Con respecto a la búsqueda de recursos, un nodo general en este modelo se comporta según el protocolo Query/QueryHit antes descrito, comportándose igual que en el modelo anterior.

En la verificación, se analizan todos los posibles caminos de ejecución. Con tal fin, se modificó el proceso **init** para que envíe un mensaje Query a cada uno de los nodos existentes preguntando por cada uno de los posibles recursos. El proceso quedará a la espera de mensajes QueryHit, los cuales arribaran al completar cada búsqueda. En el momento en que todos los nodos hayan procesado el mensaje y se encuentren bloqueados, el proceso detectará esta situación y enviará un nuevo mensaje Query hasta agotar todas las posibilidades.

A manera de ejemplo serán presentadas pruebas de dos propiedades que deberían ser cumplidas en este modelo. En primer lugar, es deseable que:

"*Siempre que un nodo recibe un mensaje ping directo, responda con un mensaje pong*"

En SPIN es necesario usar macros para definir las propiedades, de la forma mostrada a continuación.

```
#define recibiUnPingDirecto Internet[ipNodoPing]?[ping, idTransPing, ttlPing,
     hopsPing, ipNodoPong] && ttlPing == 1 && hopsPing == 0
#define envioUnPong Internet[ipNodoPong]?[pong, idTransPong, ttlPong,
     hopsPong, ipNodoPing] && ttlPong == 1 && hopsPong == 0 &&
     idTransPing == idTransPong

/*La fórmula a probar: "[] (recibiUnPingDirecto -> <> envioUnPong)" */
```

La fórmula anterior significa: "Siempre que un nodo reciba un Mensaje Ping entonces eventualmente responderá enviando un mensaje Pong". La propiedad ejecutando SPIN en modo verificación:

```
$ spin -f "[] (recibiUnPingDirecto -> <> envioUnPong)" > prop1.ltl
$ spin –a prop1.ltl NodoGeneral.pml
$ gcc –o pan pan.c
$ pan
```

Al finalizar la ejecución, la comprobación terminó en forma exitosa. A partir de esto, se puede concluir que la fórmula siempre se cumple en el modelo. En el caso de existir algún contraejemplo (secuencia de estados para la cual la fórmula no se satisface), SPIN genera un archivo NodoGeneral.ltl conteniendo la secuencia de estados que lleva a que la fórmula no sea válida en el modelo.

Otra propiedad que es posible verificar es:

"*Siempre que un Nodo General reciba un Query solicitando un recurso que el nodo posee, eventualmente le será enviado un mensaje QueryHit al nodo solicitante*"

La definición de esta propiedad en SPIN es:

```
#define recibiUnQueryYTengoElRecurso Internet[ipNodoQuery]?[query, idTransQuery,
     ttlQuery, hopsQuery, ipNodoQueryHit, recursoQuery] && Recursos[recursoQuery]
#define envioUnQueryHit Internet[ipNodoQueryHit]?[queryHit, idTransQueryHit, \
     ttlQueryHit, hopsQueryHit, ipNodoQuery, recursoQueryHit] &&
     ttlQueryHit == hopsQuery  && idTransQuery == idTransQueryHit &&
     recursoQuery == recursoQueryHit

/* La fórmula a probar:                                                  */
/*        "[] (recibiUnQueryYTengoElRecurso -> <> envioUnQueryHit)"  */
```

Además de propiedades expresadas en LTL, es posible probar aserciones por medio de la sentencia de PROMELA **assert** (*exp*). En caso que *exp* sea evaluada y no se cumpla, se termina la ejecución de la comprobación. Al finalizar la comprobación, se informará el número de aserciones violadas.

6. CONCLUSIONES

El modelo de simulación presentado puede ser utilizado evaluar el comportamiento de diferentes criterios de propagación para la búsqueda de recursos en Redes P2P usando el protocolo de comunicación Gnutella. Mejorar este criterio es importante pues reducir el número de mensajes requeridos en la búsqueda haría más eficiente al protocolo.

El modelo presentado cuando debe realizar la búsqueda de un recurso propaga la consulta entre todos los nodos adyacentes al "Nodo Actual". Una aplicación importante de este modelo es poder realizar simulaciones seleccionando los nodos a consultar con un criterio determinado y obtener resultados que puedan ser útiles para analizar la efectividad y refinar el método usado en la selección de los nodos para la búsqueda.

Este trabajo fue realizado con el objetivo de poner en práctica conceptos de comprobación de modelos y ganar experiencia en este tema. Resultó de gran provecho estudiar la herramienta SPIN para poder aplicarla en futuros trabajos.

Referencias:

[1] Holzmann, G.J., "The SPIN Model Checker: Primer and Reference Manual". Addison-Wesley, 2003

[2] Holzmann, G.J. "The Model Checker SPIN", IEEE Trans. on Software Engineering, Vol. 23, No. 5, pp. 279-295, 1997

[3] McMillan, K.L. "Symbolic Model Checking: An Approach to the State Explosion Problem", Kluwer Academic, 1993.

[4] Pieter Hartel, Pascal van Eck, Sandro Etalle, Roel Wieringa. "Modelling mobility aspects of security policies" January 26, 2004.

[5] Bernald Boigelot and Patrice Godefroid. “Model Checking in Practice: An Analysis of the ACCESS.bus Protocol using SPIN” Proceedings of Formal Methods Europe'96, Oxford, March 1996. Lecture Notes in Computer Science, vol. 1051, pages 465-478, Springer-Verlag.

[6] Z. Manna and A.Pnueli. “The temporal Logic of Reactive and Concurrent System: Specification”Springer-Verlag, 1992.

[7]Christopher D. Walton. “Typed Protocols for Peer-to-Peer Service Composition”. Proceedings of the 2nd International Workshop on Peer to Peer Knowledge Management (P2PKM 2005), San-Diego, USA, July 2005

[8] “The annotated Gnutella Protocol Specification v0.4”.
http://rfcgnutella.sourceforge.net/developer/stable/index.html

[9] “CS328: Computer Networks and Distributed Systems”. Department of Computer Science, University of Illinois, USA. http://www-courses.cs.uiuc.edu/~cs328/

[10] Clarke, E.M., Grumberg, O., Peled, D.A. “Model Checking”, MIT Press, 1999

[11] Tuya, J. “Especificación y Verificación de Sistemas Reactivos Utilizando Métodos Estructurados y Lógica Temporal”. Tesis Doctoral, de Ingeniería Eléctrica, Electrónica y Automática, Universidad de Oviedo, 1994

[12] Hoare, C.A.R “Communicating Sequential Processes”. Prentice-Hall, 1985

[13] de la Riva Álvarez, C. "Generación Automática de Restricciones del Entorno en la Verificación Modular de Sistemas Reactivos" Tesis Doctoral, Universidad de Oviedo, 2003.