NUEVAS FUNCIONES EN KivaNS 1.1

AUROVA
Grupo de Automática, Robótica y Visión Artificial
www.aurova.ua.es

Universitat d'Alacant
Universidad de Alicante

Proyecto financiado por el Vicerrectorado de Tecnología e Innovación Educativa de la Universidad de Alicante en la convocatoria de 2006.

Francisco A. Candelas Herías
Oscar Ferrer Ballester

Mayo 2007

INTRODUCCIÓN

Kiva es una aplicación gratuita de código abierto, basada en java, para la simulación de tipologías de redes de datos. Está orientada a simular el comportamiento del protocolo IP, su tratamiento de los datagramas y el recorrido por los distintos equipos que conforman su red. Su uso puede ser tanto a nivel didáctico como profesional, ya que permite una amplia gama de posibilidades de configuración que permiten la simulación de un amplio registro de topologías.

Durante los meses que ha durado el proyecto de ampliación, se ha realizado un gran esfuerzo por mantener un código estructurado y limpio que permita localizar incidencias fácilmente, así como la realización de futuras ampliaciones por parte de cualquier persona con conocimientos suficientes sobre java para abordar esta tarea. A su vez, se ha mantenido un importante control de eficiencia en cuanto a ejecución y simulación.

Kiva está en constante evolución, por lo que en un futuro las posibilidades de creación y simulación pueden ser enormes.

CORRECCIONES A LA VERSION 1.0

Al añadir mejoras al software original, nos hemos encontrado con aspectos que debían ser modificados para permitir desarrollar las nuevas funcionalidades debido a que, como es normal, en principio no se habían tenido en cuenta. Estas correcciones han sido mínimas y no se ha variado el comportamiento del proyecto inicial.

Las correcciones se resumen en las siguientes:

En caso de que el equipo destino del datagrama sea un Switch, es necesario filtrarlo y tratarlo aparte. Anteriormente pasaban directamente a ser procesados por los niveles superiores de la pila TCP/IP (cabe recordar que un Switch trabaja sobre el nivel Fisico/MAC por lo que no es posible dicho tratamiento).

Por otra parte, actualmente se controla el número de interfaces que posee un equipo y se filtra la entrada de datos por estos, actualizando el indicador de interfaz del paquete en proceso únicamente para equipos distintos a un Switch, ya que éste realiza un tratamiento especial a través de una estructura del datagrama definida para tal fin.

Hasta ahora, en la aplicación, cada vez que un dato era recibido por cualquier equipo, se modificaba el parámetro interfaz del dato para así conocer posteriormente si lo había emitido otro equipo o ese en concreto, en cuyo caso lo desecharía para evitar que el emisor reciba su propio paquete. El caso del Switch es distinto, ya que su función es diferente a la de cualquier equipo de la aplicación: reenvía la información recibida hacia otras redes, por lo que no debe modificar ese valor. El Switch posee dentro de una trama su propia estructura de datos que le

permitirá trabajar de manera correcta, al estilo de tramas BPDU de equipos reales. Se ha ampliado la clase *Dato*, añadiéndole una estructura específica para uso y consulta de los Switchs (interfaces de entrada, id, etc...). Mediante esta estructura, un Switch es capaz de saber si un paquete en concreto fue emitido por él, así como implementar en el futuro el algoritmo Spanning Tree a través de identificadores para evitar bucles en las redes basadas en ellos.

A la hora de simular la conexión directa entre un switch y otros equipos, no se podía usar la clásica configuración de la interfaz, ya que requería de un bus externo para poder conectar un equipo a otro, por lo que ha sido necesario simular este mismo bus de manera interna, desde la interfaz, haciéndola invisible para el usuario de modo que la conexión parezca directa.

Al panel que traza el funcionamiento de la red y el recorrido del paquete, se le ha añadido más información. En concreto, ahora muestra el interfaz por el que entra o sale el datagrama. Para poder ampliar la información mostrada por el panel *eventos*, se han modificado las clases que modelan la tabla que deberá estructurar la información, teniendo en cuenta este nuevo valor.

Como en el caso anterior, también se ha ampliado la información mostrada por el panel *conexiones* de la pantalla *propiedades* de la aplicación. Además de mostrar el equipo destino de una conexión, muestra la interfaz por la que está conectada a éste.

Desde la interfaz gráfica se mostraban para cualquier equipo las IPs que pertenecían a cada interfaz. Esto es correcto para cualquier equipo menos para el Switch, ya que, como hemos comentado en el primer párrafo de correcciones, el Switch trabaja en el nivel Físico/MAC de la pila TCP/IP, por lo que no tiene sentido identificar sus puertos con direcciones IP puesto que no poseen ninguna utilidad. Por ello, se ha modificado el redibujado para que evite su impresión para los Switch.

NUEVAS CARACTERÍSTICAS

MODIFICACIONES DEL API DE SIMULACIÓN

- **EQUIPO SWITCH**

  Se ha implementado el Switch como nuevo equipo a poder usar en la simulación. Permite interconectar dos o más redes del mismo tipo. Su función es la de hacer de puente entre ellas y reenviar la información recibida por los diferentes equipos que componen la red. Para su desarrollo se ha partido de la clase *Equipo*, que nos proporciona el interfaz común necesario para el desarrollo de un nuevo equipo. Contiene nuevos elementos necesarios para el funcionamiento del Switch como son una memoria interna (*MemoriaInterna*), una tabla de direcciones (*tabla*) y un temporizador.

- La memoria interna es utilizada para almacenar temporalmente las tramas recibidas con el fin de procesarlas posteriormente y prepararlas para su reenvío.
- La tabla de direcciones se utiliza por el Switch para almacenar un conjunto de pares Dirección(MAC)/Puerto(Switch) que utilizará para conocer a qué puerto debe reenviar la trama recibida.
- El temporizador únicamente indica cuánto tiempo de vida posee cada relación de la tabla de direcciones antes de expirar; se podría decir que es la fecha de caducidad.

◘ BROADCAST IP

Hasta ahora, únicamente cabía la posibilidad de realizar una petición Broadcast a nivel de enlace, por ejemplo para una petición ARP. Lo que se pretendía era dar un paso más y proporcionar la capacidad de poder hacer la petición a nivel IP, con lo que la pudieran recibir todos los ordenadores pertenecientes a la red y subredes asociadas a éstas. Para ello, se ha tenido en cuenta la configuración de la red conectada a cada uno de los interfaces de los equipos. No solo basta con ver si la dirección de broadcast pertenece a esa red directamente, sino también a cualquiera de sus subredes.

Una vez localizada la o las interfaces a las que pertenecen, en caso de ser un Router o un PC, se realiza un proceso u otro: el PC recibe la petición y la procesa y un Router, además de recibirla, la reenvía al resto de sus redes en caso de que pertenezcan a la dirección de destino de broadcast del datagrama. A diferencia de los PCs, los Routers tienen activada la opción de forwarding que permite, que además de recibir una trama, la reenvíe por el canal más adecuado para que pueda proseguir su camino.

Para implementarlo ha sido necesario modificar la manera en la que se realizaban algunas comprobaciones, como por ejemplo verificar si un paquete pertenece o no a una máquina, y además conocer el interfaz concreto que debe retransmitirlo en caso de ser necesario.

MODIFICACIONES DE LA PARTE GENÉRICA

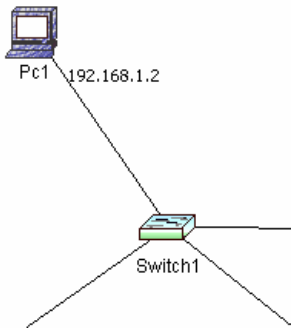
◘ MEJORA EN LA VISUALIZACIÓN DEL LISTADO DE EVENTOS

La manera en la que hasta ahora se visualizaban los eventos ocurridos durante la simulación de la red, no era lo cómoda que debería ser para el usuario.

Una vez que el número de eventos superaba la capacidad visual del panel en el que se mostraban, aparecía una barra de desplazamiento que debía desplazarse manualmente para visualizar las últimas acciones. Ahora ya no es necesario, porque por defecto la barra se sitúa siempre en el último evento mostrado. De esta manera, al ir ejecutando la simulación, ya sea paso a paso o simulación completa, siempre se tendrá a la vista el último evento ejecutado.

◘ DIBUJO CORRECTO DEL SWITCH

Las interfaces de un Switch, al trabajar sobre el nivel Físico/MAC, no poseen direcciones IP como en el resto de equipos. La aplicación hasta su modificación, pintaba las IPs correspondientes a las interfaces de todos los equipos, por lo que se han debido de modificar las clases encargadas de tal tarea para que ignore las del Switch a nivel visual.

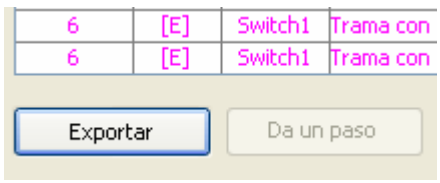


◘ GENERACIÓN AUTOMÁTICA DE IPS

Como ya se ha comentado anteriormente, las interfaces del Switch no requieren de dirección IP para ser reconocidas en la red, pero nos encontramos que para la aplicación si es necesario registrarlas para cualquier equipo que se defina en ella. Actualmente, la aplicación autogenera las IPs del Switch sin que el usuario se tenga que preocupar por ello. Hay que destacar que la función de las direcciones IPs generadas para un switch es la de mantener la lógica y coherencia de la aplicación y que a nivel de simulación IP no tienen utilidad, lo que es acorde con los switches reales que no tienen direcciones de red.

◘ NUEVA FUNCIONALIDAD DE EXPORTAR

Una vez creada y simulada la red, tenemos la posibilidad de extraer el listado de los eventos generados por la aplicación a un fichero con extensión .csv que nos va a permitir manejarlo cómodamente desde el Excel, pudiendo imprimirlo o guardarlo para un futuro estudio.



◘ AMPLIACIÓN DE LA INFORMACIÓN DE LA PESTAÑA CONEXIONES Y EVENTOS

Dentro del panel que muestra las propiedades de un equipo de la red, para la pestaña *Conexiones* se ha añadido, además del equipo al que va

conectado, por la interfaz que lo hace. Y en la pestaña *Eventos,* ahora muestra por que interfaz entra o sale el paquete en circulación.

◘ CAPACIDAD DE AÑADIR DIRECCIONES DESTINO NO EXISTENTES Y BROADCAST

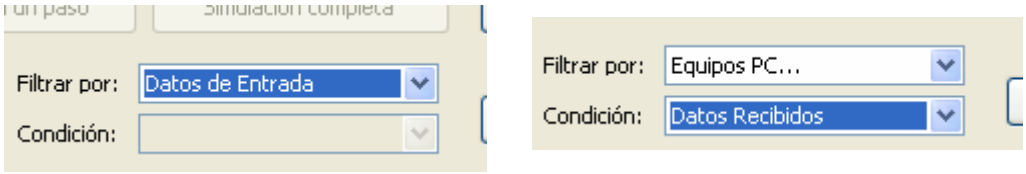
Anteriormente cada vez que se quería generar un dato para asignárselo a un equipo de la red, se debían indicar las direcciones de emisor y de destino al cuál iban dirigidos, desde unos listados (combos) que te permitían seleccionar cualquier equipo perteneciente a la red. Esto tenía el problema de no contar con la capacidad de simular direcciones inexistentes o de broadcast, por lo que, para la dirección de destino, se ha prescindido del listado y ahora es posible asignar directamente una dirección cualquiera, existente o no y de broadcast.

◘ FILTRADO DE EVENTOS

En muchas de las ocasiones en las que hemos realizado la simulación de una red, nos hemos encontrado con cierta dificultad al querer realizar un seguimiento sobre uno o varios eventos en concreto, debido a que el listado era especialmente extenso, y al aumentar se complicaba cada vez más la visualización de determinadas acciones producidas durante la simulación.

Por todo esto, se ha visto necesario el crear una herramienta que pudiera extraer la información que nos interesara del listado: los filtros. A través de ellos, podemos controlar qué tipo de datos queremos visualizar, mediante la selección de diversos criterios de filtrado. Para ello, se ha provisto de dos *combos* relacionados entre sí.

Hay dos clases de filtros: los simples y los condicionados. Los simples dependen de ellos mismos para filtrar la información, y en cambio los condicionados, dependen de algún criterio más para poder mostrar la información requerida. Para saber si estás seleccionando uno simple o uno condicionado, basta observar el listado inferior al de *Filtrar por: Condición*. En caso de encontrarte sobre uno simple, el listado inferior estará deshabilitado; en cambio para uno condicionado, se habilitará el inferior ofreciendo la posibilidad de realizar una segunda selección para afinar aun más la selección.



Una vez realizada la elección del filtro apropiado, verás como la lista de eventos de la simulación se ha regenerado únicamente con los elementos que se han considerado relevantes a la selección. Sea simple o no, el funcionamiento siempre es igual: seleccionas el filtro y la lista se regenera. Para volver a la situación inicial sobrará con seleccionar la primera opción de la lista *Filtrar por* (la que esta en blanco).

PASOS PARA UNA FUTURA AMPLIACIÓN DE FILTROS

Desde un primer momento, el desarrollo de los filtros se ideó de manera que posibilitaran su ampliación dependiendo del uso de cada usuario o centro de estudios. Por este motivo, se ha seguido una estructura que facilita en gran medida la incorporación de nuevas reglas. A continuación, paso a explicar la forma en la que se pueden añadir:

La aplicación visual se comunica con las reglas a través de la clase *Filtro*, que hace de intermediaria entre ambas. Para incorporar una o varias reglas, únicamente nos fijaremos en la clase *Reglas* y *RegistroReglas*. Estas dos clases son las que nos van a permitir añadir un número indeterminado de ellas.

Toda regla debe heredar de una clase raíz denominada *Regla*. Ésta, proporciona los métodos y variables adecuados para su comunicación con la clase *Filtro*. Hay que asegurarse de asignar en el constructor un *String* que describa el tipo de filtrado que va a realizar. Este texto será el que posteriormente se visualizará desde la aplicación en el combo *Filtrar por*. En caso de que sea una regla condicionada, se deberá rellenar el *ArrayList condiciones* desde el método *cargaCondiciones;* estas serán las descripciones que se visualicen desde el combo *condición*. Rellenando este *ArrayList* se indica a la aplicación que es una regla condicionada; en caso de no ser así, simplemente se deja ese método vacío.

Finalmente, el siguiente paso es desarrollar el filtrado que se realizará sobre el listado de eventos. Se debe implementar sobre la función *pasaRegla*. Ésta es la que se llamará desde la clase *Filtro* para obtener el listado de los eventos que la regla considere apropiados para mostrar, dependiendo de su criterio. Para conocer cuál es la condición seleccionada ha de preguntarse mediante *dameCondSeleccion*, que devolverá un valor de 0 a n, siendo n el número de condiciones que anteriormente se ha especificado (desde *cargaCondiciones*) más uno. Debe tener en cuenta que el sistema por defecto inserta una condición vacía, que puede interpretar como desee. Esta condición es la número 0, continuando a partir de ahí con las que ha especificado.

Una vez se tiene la regla generada correctamente, si queremos que la clase *Filtro* la capture y la añada a su lista, es imprescindible registrarla. Para ello se debe añadir al vector de *String listado* de la clase *RegistroReglas.java*. Se debe insertar el nombre exacto de la clase creada. Por ejemplo, si se llama *MiRegla* se deberá añadir de la siguiente manera:

*protected static String[] listado = {"RglEntrada", "RglSalida",..., "**MiRegla**"};*

Y así sucesivamente con todas las que vayamos creando.

Siguiendo estos simples pasos se podrá generar cualquier tipo de regla que se desee. En la aplicación existen un conjunto de ellas que se pueden usar como referencia para crear otras nuevas. Se debe recordar que la selección del filtrado se realiza sobre un vector de String, el cual contiene cadenas con una estructura similar a esta: "*19 [E] Pcx Envío datagrama.... eth0*", donde cada elemento se corresponde con "Instante Tipo Equipo Descripción Interfaz". Estos van a ser los datos que nos van a dar juego para discriminar un evento u otro.