ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

DESARROLLO DE CASOS DE ESTUDIO SOBRE LA HERRAMIENTA DE PLANIFICACIÓN DE REDES *NET2PLAN*

DEVELOPMENT OF CASE STUDIES ON *NET2PLAN* NETWORK PLANNING TOOL

AUTOR: ÁNGEL FERNÁNDEZ GAMBÍN

DIRECTOR: PABLO PAVÓN MARIÑO

CODIRECTOR: JOSÉ LUIS IZQUIERDO ZARAGOZA

Septiembre / 2014

Autor	Ángel Fernández Gambín
E-mail del Autor	af.gambin@gmail.com
Director(es)	Pablo Pavón Mariño
E-mail del Director	pablo.pavon@upct.es
Codirector(es)	José Luis Izquierdo Zaragoza
Título del TFG	Desarrollo de casos de estudio sobre la herramienta de planificación de redes Net2Plan
Título en Inglés	Development of Case Studies on Net2Plan network planning tool
Descriptores	Planificación de redes, Herramientas de Planificación

Resumen
El presente proyecto incluye un conjunto de casos de estudio en relación a la herramienta de planificación de redes de comunicaciones Net2Plan con el objetivo de mejorar la documentación existente de la aplicación, de cara a facilitar el uso y comprensión por parte del usuario. Cada uno de ellos consta de un vídeo-tutorial explicativo y documentación relativa a la información contenida en él.

Titulación	Grado en Ingeniería Telemática
Intensificación	
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Septiembre 2014

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. NECESIDAD DE HERRAMIENTAS DE PLANIFICACIÓN

Como consecuencia de la proliferación de dispositivos conectados de forma permanente a Internet o la operación de nuevos servicios como vídeo bajo demanda o televisión IP, el tráfico en las redes troncales crece a un ritmo superior al 20% anual [1]. Evidentemente, esta explosión de tráfico se traduce en grandes exigencias para las infraestructuras de las operadoras.

En este sentido, las fases de diseño y optimización comprenden en sí mismas un proceso crítico en el despliegue y operación de redes de comunicación, ya que los ingenieros de red deben mantener el equilibro entre el coste de la red y las capacidades ofrecidas a los usuarios. Por tanto, se hacen necesarias herramientas capaces de planificar correctamente las redes de comunicaciones, permitiendo optimizar al máximo los recursos de las mismas.

Existen multitud de herramientas de planificación y optimización, abarcando una amplia gama de plataformas, sistemas, lenguajes, funcionalidades y aplicaciones [2]. Generalmente, las aplicaciones comerciales están orientadas a la industria, mientras que otras se han desarrollado en el mundo académico para investigación, y excepcionalmente con fines docentes. El principal problema es que las herramientas existentes no proporcionan la flexibilidad suficiente para probar distintos algoritmos de planificación (es decir, los usuarios solo pueden utilizar algoritmos ya incorporados dentro de las aplicaciones) o no han sido diseñadas con fines docentes (es decir, los detalles internos de la aplicación tales como el código de los algoritmos no están disponibles públicamente), lo que limita la aplicación de los trabajos de investigación en las redes de las operadoras así como en cursos de planificación de red.

1.2. HERRAMIENTA DE PLANIFICACIÓN NET2PLAN

En este contexto, investigadores de la UPCT, concretamente los directores de este proyecto, desarrollaron la herramienta *open-source* [3] denominada Net2Plan [4]. Net2Plan es una herramienta de optimización de redes distribuida bajo licencia LGPL, de gran flexibilidad ya que no está restringida a ninguna tecnología específica (y adecuada para la mayoría de ellas) ni fabricante, permitiendo a los usuarios probar sus propios algoritmos, además de los que incorpora la herramienta. Toda la información sobre la herramienta, así como los enlaces de descarga, se puede encontrar en la página web [5].

Net2Plan proporciona una interfaz gráfica que permite la definición de diseños de red propios, así como su análisis antes y después de aplicar algoritmos de optimización. Asimismo, incorpora una serie de funcionalidades que permiten la generación automática de informes de red, así como la evaluación de los diseños en diferentes escenarios de simulación: (i) un simulador de fallos, que permite evaluar el rendimiento de disponibilidad de un diseño de red al que se le aplican algoritmos de protección y restauración; (ii) un simulador de control de admisión, que tiene como objetivo analizar el rendimiento de los sistemas de aprovisionamiento dinámicos que asignan recursos a las conexiones entrantes; y (iii) un simulador de tráfico variable, que tiene como objetivo evaluar las prestaciones de sistemas que reaccionan a variaciones de tráfico. En el capítulo 2 se describen con más detalle cada una de las funcionalidades disponibles.

Por otro lado, en algunas ocasiones los algoritmos de diseño de red requieren resolver uno o más problemas de optimización (ILPs, formulaciones convexas…), e interactuar con un *solver* especializado capaz de obtener la solución a dichos problemas. La librería JOM [6] (Java Optimization Modeler), también desarrollada dentro del grupo de investigación, está dirigida a ese uso. Permite modelar problemas de optimización en una sintaxis de alto nivel, basándose en interfaces de un conjunto de *solvers* libres (GLPK, IPOPT) y comerciales (CPLEX) para obtener soluciones numéricas. Esta librería está integrada en Net2Plan, ya que varios ejemplos incluidos en la aplicación hacen uso de ella para obtener la solución óptima a ciertos problemas de diseño de red.

1.3. OBJETIVOS

El objetivo de partida de este proyecto fue mejorar la documentación existente de la herramienta, y para ello, se han desarrollado un conjunto de casos de estudio relacionados con las distintas funcionalidades de Net2Plan descritas anteriormente, de forma que resulten una manera práctica de aprender a interaccionar con la interfaz gráfica, así como aprender a desarrollar algoritmos propios ejecutables en la aplicación.

1.4. METODOLOGÍA

Los pasos seguidos para cada uno de los casos de estudio son los siguientes:

1. Determinar el contenido del caso de estudio, seleccionando los conceptos a desarrollar y los ejemplos incluidos.
2. Estructurar el vídeo-tutorial que contendrá el caso de estudio.
3. Grabar el vídeo-tutorial (en inglés, para lograr una mayor difusión) usando el programa Camtasia Studio 5 [7].
4. Integrar el vídeo-tutorial en un canal de YouTube [8] creado a este efecto.

2. NET2PLAN

2.1. INTRODUCCIÓN

Net2Plan [4] es una herramienta *open-source* implementada en Java, dedicada a la planificación, optimización y evaluación de redes de comunicación. Se diseñó en un principio como una herramienta de soporte a la docencia en cursos de planificación de redes en la Universidad Politécnica de Cartagena. Con el tiempo se ha convertido en una poderosa herramienta de planificación de redes, tanto para la docencia como la industria [4], junto con un repositorio creciente de recursos, principalmente algoritmos, de planificación de redes.

Net2Plan está construido sobre una representación abstracta de la red, llamada "*Network plan"*, basada en siete componentes: nodos, enlaces, rutas, demandas de tráfico, segmentos de protección, SRGs (Shared Risk Groups – Grupos de Riesgo Compartido) y capas de red. Esta representación de red es independiente de la tecnología, por lo tanto Net2Plan puede adaptarse a multitud de tecnologías. Para ello, el usuario debe introducir la información específica de una determinada tecnología a través de una serie de atributos, unidos a cualquiera de los componentes mencionados anteriormente. Algunos nombres de atributos se han fijado para facilitar la adaptación de las tecnologías conocidas (por ejemplo, redes IP).

Net2Plan proporciona una librería de Java, así como dos interfaces de usuario: una de línea de comandos y otra gráfica (CLI y GUI, respectivamente). La interfaz gráfica es especialmente útil para las sesiones de laboratorio como un recurso docente, o para una inspección visual de la red. Por otro lado, la interfaz de línea de comandos es comúnmente utilizada en investigación en tareas de procesamiento por lotes o simulación a gran escala de múltiples escenarios. Por lo tanto, Net2Plan es una herramienta destinada a un amplio espectro de usuarios: la industria, la investigación y la docencia [4]. Independientemente de la interfaz (CLI o GUI) seleccionada por el usuario, Net2Plan dispone actualmente (versión 0.2.3, marzo 2014) de seis herramientas diferentes:

- **Interfaz para diseños de red offline (*Offline network design)*:** Se utiliza para realizar el diseño de redes utilizando algoritmos incorporados en la aplicación o definidos por el usuario, decidiendo sobre aspectos tales como la topología de red, el enrutamiento de tráfico, la capacidad por enlace, segmentos de protección y así sucesivamente. Si es necesario, los algoritmos basados en formulaciones de optimización con restricciones pueden ser resueltos utilizando alguno de los *solvers* incluidos en la librería JOM mencionada anteriormente.
- **Generador de matrices de tráfico (*Traffic matrix generation)*:** Se utiliza en el proceso de generación y normalización de matrices de tráfico, por ejemplo, usando modelos de generación aleatoria que se pueden encontrar en la literatura, tales como el modelo población–distancia, el modelo de gravedad, o diferentes modelos basados en distintas distribuciones, así como modelos de generación de matrices a partir de una dada. Además, proporciona diversos modelos de normalización de matrices de tráfico (por ejemplo que el tráfico total ofrecido a la red sea igual a un cierto valor).
- **Simulador de fallos (*Resilience simulation)*:** Este simulador se utiliza para evaluar

el funcionamiento de algoritmos de protección/restauración ante fallos en nodos y enlaces.

- **Simulador de tráfico variable (*Time-varying traffic simulation)*:** Este simulador se utiliza para evaluar algoritmos de reacción a variaciones de tráfico que, por ejemplo, reencaminen el tráfico o modifiquen la capacidad de los enlaces.
- **Simulador de control de admisión (*Connection-admission-control simulation)*:** Este simulador se utiliza para evaluar el funcionamiento de algoritmos de control admisión de conexiones, que asignan dinámicamente recursos para las peticiones de conexión.
- **Generador de informes de red (*Reports)*:** Net2Plan permite generar informes, definidos por el usuario o incluidos en el software, de cualquier diseño de red. La herramienta de generación de informes está integrada dentro de todas las funcionalidades anteriores, excepto obviamente en el generador de matrices de tráfico, por lo que es posible crear informes con medidas de rendimiento en cualquiera de los aspectos comentados en las otras funcionalidades.

Los algoritmos en cada una de las funcionalidades mencionadas son clases Java, o archivos *.jar* que contienen un conjunto de clases Java, que implementan una interfaz pública concreta (toda la información se encuentra recogida en el Javadoc, disponible en la web de Net2Plan [5]). Como ya se ha mencionado, Net2Plan permite la ejecución de algoritmos desarrollados por el usuario, y para ello, únicamente requiere la programación de una clase Java que implemente la interfaz apropiada.

Siguiendo con la introducción inicial, las herramientas de planificación actuales están orientadas a la industria o a la investigación/docencia, sin proporcionar un nexo de unión entre ambos mundos. Por el lado de la industria, generalmente se utilizan herramientas comerciales que ocultan las decisiones de planificación y los detalles algorítmicos internos bajo interfaces gráficas sencillas e intuitivas al usuario que se asemejan a herramientas asistidas por ordenador (*point & click*). Por otra parte, desde el lado académico, los investigadores que se dedican a estudiar nuevos métodos de planificación suelen necesitar un control total sobre las decisiones de planificación, y con frecuencia desarrollan sus algoritmos y herramientas casi desde cero. Los algoritmos de planificación pueden estar basados en resolver problemas de optimización usando *solvers* especializados o aplicando heurísticos, ya que la mayor parte de los problemas de diseño de redes son *NP-hard*. En cualquier caso, las herramientas comerciales ofrecen un conjunto más o menos completo de características para el diseño y análisis de redes, sin depender de un fabricante específico. Algunas de estas características son: simulación de varios escenarios de configuración (*what if…?*, ¿qué ocurriría si…?), sistemas de encaminamiento, pruebas de sistemas de restauración de red, o análisis bajo distintas cargas de tráfico.

El problema del software comercial es que solo sirve para tecnologías y protocolos ya consolidados en el mercado, pero no proporciona soporte para estudiar tecnologías emergentes, que es en lo que suelen trabajar los investigadores. Por su parte, Net2Plan proporciona ese puente de unión entre academia e industria, sin ser específico para ninguna tecnología, lo que permite probar nuevas bajo estudio, ni tampoco para ningún fabricante, siendo además una herramienta gratuita. La filosofía de Net2Plan hace hincapié en la reutilización del software. Una vez que un algoritmo se desarrolla en el marco de Net2Plan, puede ser modificado o reutilizado como partes de otros, y se puede aplicar a

cualquier instancia de red, de manera que la red puede ser diseñada progresivamente. Es decir, los usuarios pueden encadenar algoritmos de forma sucesiva, de manera que cada uno de ellos complete una parte del diseño de red. Por ejemplo, se puede comenzar con una red en la que se definen solo los nodos. Después, un algoritmo se utiliza para definir los enlaces en la red de acuerdo a algún patrón de rendimiento. Tras esto, otro algoritmo se puede ejecutar para decidir conjuntamente sobre la capacidad de los enlaces y las rutas de los flujos de tráfico, para una matriz de tráfico dada. Como resultado, Net2Plan puede ser una poderosa herramienta para asignaturas relacionadas con la planificación de redes de comunicaciones, ya que permite a los estudiantes ver paso a paso cómo crecen sus diseños.

Asimismo, una vez que se haya completado el diseño de red, los usuarios pueden analizar y evaluar su diseño haciendo uso de todas las funcionalidades que proporciona Net2Plan, generando informes y ejecutando simulaciones en muy diversos escenarios.

Además, la herramienta es en sí misma un repositorio de algoritmos de planificación en multitud de tecnologías. Por lo tanto, los algoritmos están disponibles al público, de forma que pueden ser verificados y validados, mejorando la fiabilidad de los resultados.

La siguiente figura resume la filosofía descrita anteriormente:

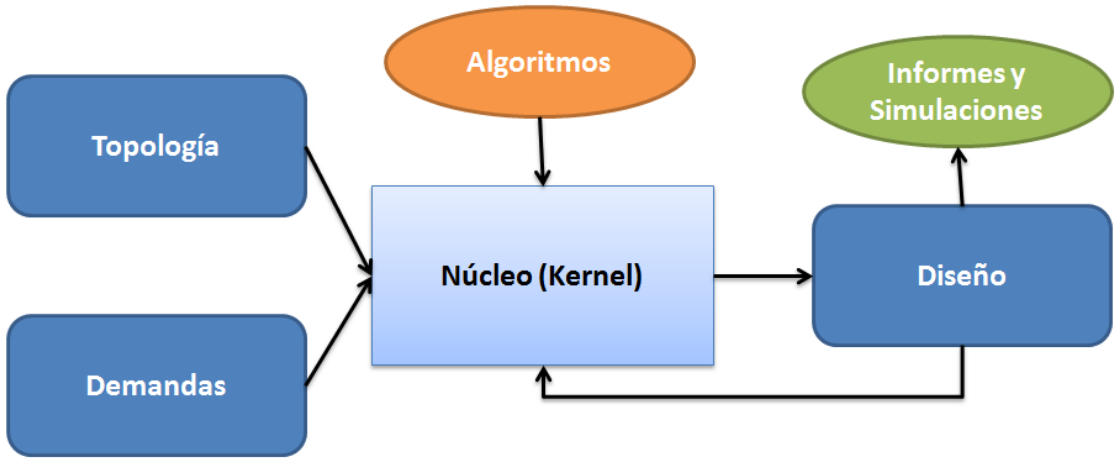


Figura 1: Flujo de trabajo

2.2. MODELO DE RED

Como se indica en la sección anterior, Net2Plan está diseñado con el objetivo de superar las barreras impuestas por las herramientas de planificación de red existentes, por dos razones principales: (i) los usuarios no están limitados a ejecutar algoritmos integrados, ya que pueden integrar los suyos propios, aplicables a cualquier instancia de red, ya que las clases Java implementan interfaces particulares; (ii) Net2Plan define una representación de red, llamada "*Network plan*", basada en siete componentes: nodos, enlaces, rutas, demandas de tráfico, segmentos de protección (SRGs, Shared Risk Groups) y capas de red, sin depender de ninguna tecnología de red.

No obstante, a pesar de que Net2Plan es una herramienta independiente de la tecnología, los usuarios pueden introducir información específica de la tecnología a través de atributos

definidos por el usuario para cada elemento en el diseño de red. No obstante, Net2Plan ofrece algunas librerías para tecnologías concretas como IP o redes ópticas.

2.2.1. REPRESENTACIÓN DE LA RED

El diseño de red se almacena en una estructura de datos llamada "*Network plan"* (clase `com.net2plan.interfaces.networkDesign.NetPlan` de la librería de Net2Plan). En el diseño de red offline (herramienta *Offline network design*), los algoritmos reciben un objeto *NetPlan* y devuelven ese objeto modificado. El objeto *NetPlan* solo contiene un conjunto de atributos básicos (por ejemplo posición de nodo, capacidad de enlace, longitud de enlace, tráfico ofrecido por demanda, etc.), que pueden ser accedidos por métodos *getX()* o modificados por métodos *setX()*. Los atributos para tecnologías o problemas específicos, así como atributos propios definidos por el usuario, que a menudo se asocian a un elemento de la red (nodo, enlace) en una herramienta específica, aquí se almacenan en mapas clave–valor dentro del objeto *NetPlan*. De esta manera, los usuarios pueden añadir o quitar atributos a cada elemento de la red en tiempo de ejecución sin tener que cambiar la representación del objeto *NetPlan*. Para obtener más información, se puede consultar la clase antes mencionada dentro del Javadoc de Net2Plan.

Asimismo, el objeto NetPlan representa en su conjunto un elemento "red". Por lo tanto, hay uno y solo un elemento de red en cada representación de red. El elemento de red es solo un soporte de información general que se puede asociar a la misma. Tiene tres variables asociadas: nombre, descripción y atributos. El nombre de la red es una cadena de texto (*String)*, de forma que el usuario puede definir un título para la red. La descripción de la red es también una cadena de texto, que se puede utilizar para asociar una breve descripción a la red. Ambos parámetros se muestran en la interfaz gráfica de usuario. Por último, la red puede tener un conjunto de atributos de pares clave-valor que se puede utilizar para adjuntar cualquier información arbitraria a la red.

2.2.1.1. Nodos

Los nodos son la entidad básica de un diseño de red, y son, o bien un punto de conexión, un punto de redistribución o un punto final de la comunicación, siendo capaces de enviar, recibir o reenviar tráfico a través de un canal de comunicación (o enlace). Estos se caracterizan por cuatro variables básicas: identificador, posición, nombre y atributos. El identificador se define internamente por el núcleo y determina un número de serie único para el nodo. La posición del nodo establece la localización del nodo en un plano cartesiano bidimensional. El nombre del nodo es una cadena de texto (*String)* que se asigna al nodo, por ejemplo, para mostrarse en la interfaz gráfica. Por último, los atributos del nodo son un conjunto arbitrario de pares clave-valor que se pueden utilizar para asociar cualquier información arbitraria al nodo.

2.2.1.2. Enlaces

Junto con los nodos, los enlaces comprenden la topología de red. Estos son los canales de comunicación que permiten la conectividad entre dos nodos. En Net2Plan, los enlaces son unidireccionales, desde un nodo a otro. Dos nodos pueden estar conectados por cero, uno o más enlaces. Sin embargo, están prohibidos auto-enlaces (enlaces donde el nodo origen y el nodo destino son el mismo). Los enlaces se caracterizan por seis variables fundamentales: identificador, nodo de origen, nodo de destino, capacidad, longitud y

atributos. Una vez más, el identificador es un número de serie único para el enlace. Los nodos de origen y destino son los identificadores de los nodos correspondientes. La capacidad (medida en *Erlangs*) es la cantidad de tráfico que el enlace es capaz de cursar. La longitud del enlace (que se supone en kilómetros) representa la distancia física del enlace, para ser utilizada por ejemplo, para los cálculos del retardo de propagación. Por último, los atributos de enlace son un conjunto arbitrario de pares clave-valor que se pueden utilizar para asociar cualquier información arbitraria al enlace.

2.2.1.3. Demandas de tráfico

El tráfico se modela a través de un conjunto de demandas. Cada demanda representa un flujo de tráfico ofrecido extremo a extremo en la red. En Net2Plan, las demandas se consideran *unicast*, es decir, que solo tienen un nodo de entrada y un nodo de salida. Además, las auto-demandas no están permitidas.

Las demandas tienen cinco variables base: identificador, nodo de entrada, nodo de salida, volumen de tráfico ofrecido y atributos de la demanda. El identificador de la demanda es un número de serie asignado por el núcleo que identifica de forma única la demanda. El nodo de entrada es el identificador del nodo que inyecta el tráfico a la red, mientras que el nodo de salida es el identificador del nodo que recibe ese tráfico. El volumen de tráfico ofrecido es la cantidad de tráfico (en *Erlangs*) para la demanda que se inyecta en la red. Todo, una parte, o nada de este tráfico será cursado (esta información viene dada por las rutas de la red). Por último, los atributos de la demanda son un conjunto arbitrario de pares clave-valor que se puede utilizar para asociar cualquier información arbitraria a la demanda.

Un enfoque simplificado para modelar el tráfico ofrecido entre nodos es la llamada matriz de tráfico (*Traffic matrix)*. Una matriz de tráfico es una matriz $|N|\times|N|$ (donde $|N|$ es el número de nodos en la red) en la que cada par (i,j) representa el (promedio) tráfico desde el nodo i al nodo j. El principal inconveniente que representa el tráfico ofrecido por medio de matrices de tráfico, es que en la representación de la matriz de tráfico se supone que existe como máximo una demanda entre cada par de nodos. Entonces, si calculamos la representación de la matriz de tráfico de un conjunto de demandas D donde algunos pares de nodos tienen más de una demanda entre ellos, se puede producir una ambigüedad. Esta situación se plantea en la figura de abajo, donde las demandas 1 y 2 del conjunto de demandas de la izquierda se agrupan en una sola entrada en la matriz de tráfico. El mismo resultado se obtiene con el conjunto de demandas de la derecha. Por tanto, para proporcionar una manera completa de modelar cualquier forma de tráfico, la representación del tráfico ofrecido en Net2Plan se basa en un conjunto arbitrario de demandas.

d	a_d	b_d	h_d
1	1	2	5
2	1	2	3
3	2	1	4

$\rightarrow \begin{pmatrix} 0 & 8 \\ 4 & 0 \end{pmatrix} \leftarrow$

d	a_d	b_d	h_d
1	1	2	8
2	2	1	4

Figura 2: Ambigüedad en la matriz de tráfico en función de las demandas asociadas

2.2.1.4. Rutas

Las rutas son los caminos en una red a lo largo de los que se envía el tráfico de las demandas. Las rutas se definen como una secuencia ordenada y contigua de enlaces que van desde el nodo de entrada de la demanda hasta el nodo de salida, cursando una cantidad particular de tráfico de la demanda, y que tienen cero, uno o más segmentos de protección asociados a ser utilizados en caso de fallo. Cada demanda puede tener cero, una o más rutas que transportan tráfico al mismo tiempo, con la condición de que el tráfico cursado no puede superar el tráfico ofrecido. Comúnmente, cuando cada demanda en la red tiene una única ruta asociada, el encaminamiento se denomina *unsplittable* o no bifurcado. A la inversa, si al menos una demanda se realiza por dos o más rutas, el enrutamiento de la red se conoce como *splittable* o bifurcado. El tráfico cursado por cada ruta puede ser cualquiera. Habitualmente, cuando todas las rutas en la red tienen una cantidad de tráfico cuantizada (múltiplo de un número entero), el encaminamiento se denomina *integral*.

Las rutas tienen seis atributos principales: identificador, demanda, secuencia de enlaces, tráfico cursado, lista de segmentos de protección, y los atributos de ruta. El identificador es un número de serie único determinado por el núcleo. La demanda es el identificador de la demanda asociada a la ruta. La secuencia de enlaces es la sucesión ordenada de identificadores de enlace seguidos por la ruta, pudiendo contener bucles. La única restricción, natural, es que un enlace debe terminar en el nodo en el que inicia el siguiente enlace, y la ruta debe empezar y acabar en el nodo de entrada y de salida de la demanda, respectivamente. El tráfico cursado es el volumen de tráfico que lleva esa ruta. La lista de segmentos de protección es la sucesión ordenada de los segmentos de protección que respaldan el tráfico de esta ruta, de los cuales se hablará en el siguiente apartado. Por último, los atributos de ruta son un conjunto arbitrario de pares clave-valor que se pueden utilizar para asociar cualquier información arbitraria a la ruta.

2.2.1.5. Segmentos de protección

Un segmento de protección se define como una secuencia ordenada de enlaces con un volumen específico de capacidad reservada en cada enlace. Los segmentos de protección se utilizan para respaldar de manera parcial o total a una o más rutas de tráfico, tomando una ruta alternativa a la original cuando se produce un fallo. El ancho de banda reservado se toma de la capacidad de los enlaces. Así, la capacidad de un enlace utilizable para cursar tráfico de las rutas, es la capacidad del enlace nominal, menos la capacidad reservada de los segmentos de protección que atraviesan el enlace. Mientras que los bucles están permitidos para las rutas, se prohíben para los segmentos de protección. Los segmentos de protección se utilizan comúnmente junto con el algoritmo `GenericProtectionSegmentAlgorithm`, incorporado en el repositorio de algoritmos para el simulador de fallos, que especifica una forma general de reaccionar a los fallos en nodos y enlaces, haciendo uso de los segmentos de protección definidos en el diseño de red. Cuando se produce un fallo, este algoritmo procesa las rutas afectadas, e intenta desviar el tráfico en la medida de lo posible haciendo uso de los segmentos de protección asociados a esas rutas.

Los segmentos de protección tienen cuatro variables base: identificador, secuencia de enlaces, ancho de banda reservado y atributos para los segmentos de protección. El identificador es un número de serie único determinado por el núcleo. La secuencia de

enlaces es la sucesión ordenada de enlaces seguida por el segmento de protección. Un segmento de protección solo se puede asignar a una ruta en particular, si los nodos extremos del segmento de protección son atravesados por la ruta. El ancho de banda reservado es la cantidad de capacidad reservada en cada uno de los enlaces atravesados. Por último, los atributos de los segmentos de protección son un conjunto arbitrario de pares clave-valor que se puede utilizar para conectar cualquier información arbitraria al segmento de protección.

Junto con `GenericProtectionSegmentAlgorithm`, los segmentos de protección se convierten en un medio para modelar fácilmente muchos sistemas de protección de red:

- Si un segmento de protección está asociado a una sola ruta, se modela la denominada *protección dedicada*: el ancho de banda reservado en ese segmento solo se puede utilizar si la ruta primaria falla. Si un segmento de protección está asociado a más de un ruta, estamos modelando un esquema de *protección compartida.*
- Si el ancho de banda reservado de un segmento de protección está por debajo del tráfico cursado por la ruta primaria, se está modelando un esquema *de protección parcial.*
- Si un segmento de protección comparte los nodos de origen y de destino de la ruta, se está modelando un esquema de *protección de ruta*. Si solo comparte los nodos de origen y destino de un enlace de la red, se está modelando un esquema de *protección de enlace*. De lo contrario, es el sistema de *protección subruta*.

2.2.1.6. Grupos de riesgo compartido (SRGs)

Un grupo de riesgo compartido (SRG) es un concepto que representa un riesgo particular de fallo en la red que, si sucede, provoca un fallo simultáneo en un determinado conjunto de enlaces y/o nodos. Por ejemplo, un SRG puede estar asociado con el riesgo de cortar accidentalmente un determinado conducto que contiene los enlaces entre dos nodos (por ejemplo, uno en cada dirección). Si se produce este corte, los dos enlaces fallarían simultáneamente. Entonces, estarían indisponibles hasta que se completara la reparación del daño.

Los SRGs tienen cinco variables base: identificador, conjunto de nodos afectados, conjunto de enlaces afectados, tiempo medio hasta fallo (*Mean Time To Fail, MTTF)*, tiempo medio de reparación (*Mean Time To Repair, MTTR*) y atributos asociados como en el resto de elementos mencionados antes. El identificador es un número de serie único asignado por el núcleo. El conjunto de nodos afectados contiene los identificadores de los nodos que fallan simultáneamente cuando el riesgo asociado al SRG suceda. De forma idéntica, el conjunto de enlaces afectados contiene los identificadores de los enlaces. Tenga en cuenta que ambos, nodos y enlaces, fallan al mismo tiempo. El tiempo medio hasta fallo es la expectativa de tiempo (en horas) desde el momento en que el fallo sea reparado hasta el momento en que vuelva a suceder. El tiempo medio de reparación es el tiempo esperado (en horas) desde el momento en que el fallo se produce hasta que sea reparado. Por último, los atributos para los SRGs son un conjunto arbitrario de pares clave-valor que se puede utilizar para asociar cualquier información arbitraria al SRG.

Los SRGs se utilizan para modelar los riesgos de fallo que amenazan la red, y facilitan el

diseño y la evaluación de la misma para recuperarse adecuadamente ante estos fallos. Como ejemplo, la información de un SRG se lee en el simulador de fallos para generar eventos de fallo/reparación a los que los algoritmos de aprovisionamiento deben reaccionar. Además, el cálculo de la disponibilidad y los informes incluidos para calcular la vulnerabilidad frente al desastre, que estima numéricamente medidas sobre la disponibilidad de la red (es decir, sin simulación), también hacen uso de la información de los SRGs.

2.2.1.7. Redes multicapa

La clase *NetPlan*, en un esfuerzo por hacer que la herramienta sea lo más genérica y flexible posible, proporciona una forma sencilla de modelar redes de múltiples capas genéricas. De manera conceptual, dada una red multicapa, los enlaces de la capa superior se pueden interpretar como las demandas de tráfico en la capa inferior, de manera que si tienen una ruta asociada implica que dicha demanda es cursada; en caso contrario, la demanda es bloqueada. Así, los usuarios que tratan de desarrollar algoritmos multicapa solo tienen que descomponer la red de múltiples capas en una lista de redes de una sola capa, es decir, como una lista de objetos *NetPlan*.

2.2.1.8. Modelado de tecnologías específicas

Como se ha indicado en apartados anteriores, el objeto *NetPlan* solo contiene un conjunto de atributos básicos, como por ejemplo: posición de nodos, capacidad de enlaces, longitud de enlaces, tráfico ofrecido por demanda, etc. Los atributos de problemas o tecnologías específicos que un algoritmo relacionado debe definir y manejar (como los pesos por enlace en un algoritmo para el diseño de encaminamiento en redes IP / OSPF) se pueden almacenar como atributos clave-valor dentro del objeto *NetPlan*. De esta forma, los algoritmos pueden manipular los atributos de cada elemento de la red en tiempo de ejecución sin tener que cambiar la representación del objeto *NetPlan*.

A modo de ejemplo, se supone que se está usando un algoritmo para determinar las capacidades de los enlaces, dado un conjunto de demandas de tráfico, y se tiene que elegir entre varios valores de capacidad con el fin de minimizar el coste total de la red, cursando todo el tráfico. Pues bien, para este caso se pueden definir las opciones para los valores de capacidad y los costes asociados usando dos atributos de red que serían utilizados por el algoritmo.

Es importante destacar que todos los pares clave-valor se almacenan como cadenas de texto. Los usuarios son responsables de hacer las conversiones entre tipos de dato adecuadas. Por ejemplo, se puede almacenar un vector de números enteros como una sucesión de números separados por espacios.

En la sección de *Convenciones tecnológicas (Technology conventions)* en la página web de Net2Plan, los usuarios pueden encontrar algunos ejemplos de convenios tecnológicos utilizados dentro de Net2Plan. Aun así, los usuarios son libres de utilizar sus propias convenciones, sin hacer uso de las ya incluidas.

2.2.2. INTEGRACIÓN DE CÓDIGO DE USUARIOS

Una de las características más importantes de Net2Plan es que permite a los usuarios ejecutar su propio código (algoritmos, informes, en general se hace referencia a código

ejecutable). A continuación, se describe brevemente cómo integrar código propio en Net2Plan. El código ejecutable se implementa como clases Java, usando archivos únicos *.class* o archivos con conjuntos de clases integradas *.jar*, siguiendo esta estructura:

- Algoritmos para el diseño de redes offline deben implementar la interfaz:

  `com.net2plan.interfaces.networkDesign.IAlgorithm`

- Los informes o *reports* deben implementar la interfaz:

  `com.net2plan.interfaces.networkDesign.IReport`

- Los algoritmos de aprovisionamiento que reaccionan ante fallos y reparaciones en la red (usados en el simulador de fallos e informes), deben implementar la interfaz:

  `com.net2plan.interfaces.resilienceSimulation.IProvisioningAlgorithm`

- Los módulos que generan eventos de fallo y reparación para ser consumidos por los algoritmos de aprovisionamiento deben implementar la interfaz:

  `com.net2plan.interfaces.resilienceSimulation.IResilienceEventGenerator`

- Los algoritmos *Connection- Admisión -Control* que reaccionan a las peticiones de conexión (usados en el simulador de Control de Admisión) , deben implementar la interfaz:

  `com.net2plan.interfaces.cacSimulation.ICACAlgorithm`

- Los módulos que generan las peticiones de conexión para ser consumidas por los algoritmos CAC deben implementar la interfaz:

  `com.net2plan.interfaces.cacSimulation.IIConnectionEventGenerator`

- Los algoritmos de aprovisionamiento que reaccionan a las variaciones de tráfico (usados en el simulador de Tráfico Variable) deben implementar la interfaz:

  `com.net2plan.interfaces.timeVaryingTrafficSimulation.ITrafficAllocationAlgorithm`

- Los módulos que generan eventos de variación del tráfico a lo largo del tiempo para ser consumidos por los algoritmos de aprovisionamiento deben implementar la interfaz:

  `com.net2plan.interfaces.timeVaryingTrafficSimulation.ITrafficGenerator`

La información completa de cada interfaz se puede encontrar en el Javadoc disponible en la web. La integración del código ejecutable requiere simplemente guardarlo en cualquier directorio del ordenador, aunque es buena práctica guardarlo en el directorio de trabajo de Net2Plan. Además, con el fin de mejorar la experiencia del usuario, el núcleo de Net2Plan es capaz de capturar cualquier excepción lanzada por el código ejecutable, así como los mensajes de salida por consola (`System.out`, `System.err`) y mostrarlos en la consola Java incluida en la aplicación. Este es un recurso valioso para la depuración de los algoritmos ejecutados en Net2Plan. Concretamente, los mensajes de excepciones incluyen una traza completa del error (archivos, el número de línea de la excepción...).

Cuando el código ejecutable quiere detener su ejecución lanzando una excepción que debe ser informada al usuario de una forma más clara (no a través de la consola de Java), se

recomienda usar la clase `Net2PlanException`. El mensaje asociado a esta excepción se imprime en una ventana emergente en lugar de la consola de Java, y por lo tanto es más visible para el usuario.

Por ejemplo, dado un algoritmo que recibe un parámetro de entrada del usuario, que debe ser positivo, una buena práctica de programación es comprobar si los parámetros recibidos están dentro de sus rangos válidos. Si se recibe un número negativo como parámetro de entrada (o algo que no es un número), es mejor generar una `Net2PlanException` que muestra el mensaje de información en un pop-up en Net2Plan, que una excepción en tiempo de ejecución cuyo mensaje puede ser leído solo si el usuario comprueba la consola Java.

Cuando el código ejecutable se implementa como un archivo de clase Java, la ruta completa a la clase debe seguir el nombre del paquete de la clase, con el fin de cargar correctamente el código. Por ejemplo, si se crea un algoritmo llamado `testAlgorithm` en el paquete `test.myAlgorithms`, la ruta completa de la clase debe ser algo como *... / test / myAlgorithms / testAlgorithm.class*. Para los archivos tipo *.jar* no hay ningún tipo de restricciones en la ruta.

Además, Net2Plan permite hacer cambios dinámicos en el código ejecutable, es decir, los usuarios pueden modificar su código, recompilar y volver a ejecutarlo (solo haciendo clic en el botón “Execute” en la interfaz gráfica) sin la necesidad de reiniciar Net2Plan.

2.3. DISEÑO DE REDES OFFLINE

La herramienta *Offline network design* ayuda a los usuarios en el proceso de diseño de red *offline* (es decir de manera estática, sin introducir cambios dinámicos en cuanto a fallos, reparaciones, fluctuación de tráfico, peticiones de conexión, etc.) y la planificación. En la práctica, el diseño de red implica diferentes variables: nodos de red, enlaces de red, capacidades de enlace, el encaminamiento del tráfico, demandas de tráfico… Por lo general, los problemas de diseño de red *offline* reciben parte de esta información como parámetros de entrada (por ejemplo, las demandas de tráfico, y la topología de la red) y tratan de optimizar los demás (por ejemplo, las capacidades de los enlaces y el encaminamiento del tráfico) según una determinada función objetivo. Naturalmente, el número de posibles variantes y subtipos de problemas de diseño de red es infinito. Por otra parte, diferentes tecnologías pueden añadir sus propios aspectos particulares de diseño de red. Por esta razón, el diseño de red se ha convertido en una mezcla de arte e ingeniería.

Por este motivo, en un intento de proporcionar un criterio sistemático para categorizar problemas de diseño de red, Net2Plan adopta el siguiente esquema de nomenclatura, que es solo una extensión de la taxonomía de los problemas de diseño de red descritos por Kleinrock [9]:

- Asignación de topología (*Topology assignment* - *TA*): Decide sobre los nodos y/o enlaces en la red.
- Asignación de capacidades (*Capacity assignment* - *CA*): Decide sobre las capacidades de los enlaces.
- Asignación de rutas/encaminamiento (*Flow assignment* - *FA*): Decide sobre el enrutamiento de las demandas de tráfico a través de los enlaces de red.

- Asignación de ancho de banda (*Bandwidth assignment - BA*): Decide sobre el volumen de tráfico que cursa cada demanda.

De acuerdo con este esquema de nombres, las combinaciones de estos problemas se llaman fusión de siglas. Por ejemplo, un problema de *asignación de capacidad y encaminamiento* (CFA) está relacionado con el cómputo conjunto de capacidades de enlace y del encaminamiento. Es importante destacar que esta taxonomía debe ser considerada como un intento de organizar lo más didácticamente posible la gran diversidad de problemas de planificación que pueden surgir en las redes de comunicación.

Seleccionando la pestaña *Offline network design* en la interfaz gráfica se abre la herramienta de diseño de red *offline*. Un diseño de red en Net2Plan es un objeto de esta clase `com.net2plan.interfaces.networkDesign.NetPlan`, que contiene una representación de red con los elementos descritos en puntos anteriores. La palabra *offline* implica que todos los valores de las variables definidas en el objeto *NetPlan* son deterministas y no varían a lo largo del tiempo. Por ejemplo, los valores de tráfico ofrecido se asumen constantes, pues representan valores promedio de tráfico, aunque en un entorno real el tráfico puede fluctuar alrededor de esta media.

2.4. DISEÑO DE REDES ONLINE

En un entorno real, las condiciones de red varían durante su funcionamiento, de acuerdo a diferentes fenómenos tales como: fallos en nodos y enlaces, creación de nuevos circuitos virtuales o variación en los volúmenes de tráfico. En este caso, los usuarios podrían estar interesados en analizar, mediante una simulación por eventos, cómo sus redes reaccionan a estos cambios y cómo sus diseños se adaptan en consecuencia para ello.

Net2Plan ofrece tres herramientas de simulación post-análisis:

- El simulador de fallos permite evaluar el rendimiento de disponibilidad de algoritmos de protección y restauración de red.
- El simulador de control de admisión (CAC) está dirigido a evaluar el rendimiento de sistemas de aprovisionamiento dinámicos que asignan recursos a las conexiones entrantes. Dependiendo de la tecnología de red subyacente que se esté modelando, nuevas conexiones pueden representar por ejemplo, peticiones de circuitos virtuales, llamadas telefónicas, sesiones multimedia, etc.
- El simulador de tráfico variable en el tiempo se dedica a analizar el rendimiento de algoritmos dinámicos de asignación que reaccionan a las variaciones en los volúmenes de demandas de tráfico. La asignación no solo se refiere a modificar el encaminamiento del tráfico, sino que también significa que la topología de red puede cambiar a lo largo del tiempo (por ejemplo, la adición de nuevos enlaces, la actualización de las capacidades de enlace...) para adaptarse a la nuevas condiciones de tráfico.

La arquitectura de los tres simuladores es similar, basada en el paradigma de simulación por eventos discretos. El funcionamiento de la red se modela como una secuencia de eventos discretos en el tiempo [10]. Cada evento tiene lugar en un momento concreto en el tiempo y marca un cambio de estado en el sistema. Entre eventos consecutivos, no se produce ningún cambio en la red; por tanto, la simulación salta directamente en el tiempo de un evento a otro.

El modelo básico de simulación orientado a eventos se ilustra en la siguiente figura:

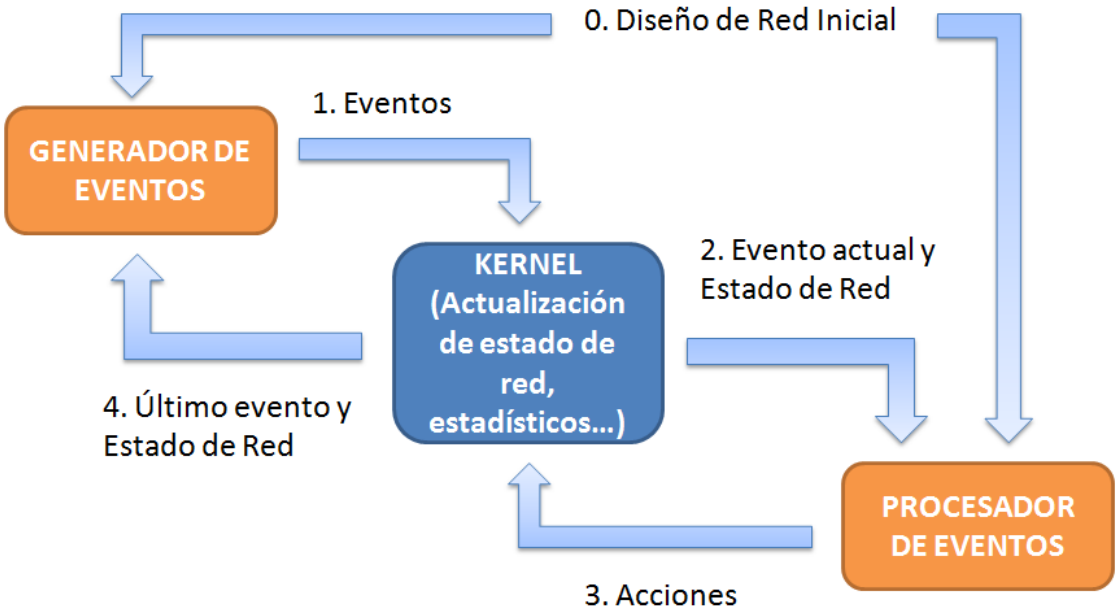


Figura 3: Modelo de simulación por eventos

El generador de eventos crea eventos de acuerdo a un algoritmo determinado (incorporado o definido por el usuario), y deposita los eventos ordenados en la lista de eventos futuros (FEL) en el kernel. El planificador incluido en el kernel procesa uno a uno y de manera cronológica según el reloj global de simulación, cada uno de los eventos almacenados dentro de la lista de eventos, llamando al procesador de eventos (que también puede ser implementado por el usuario) para cada evento. A continuación, el procesador de eventos reacciona al evento tomando algunas acciones, que son procesadas por el kernel para actualizar el estado de la red. Asimismo, las estadísticas se recogen en el kernel durante la simulación.

En cuanto a la recogida de estadísticas, la interfaz del procesador de eventos proporciona un método denominado *finish()*, que se invoca cuando termina la simulación, y se puede utilizar para imprimir estadísticas específicas de la tecnología que el propio algoritmo de procesado de eventos (no el kernel) está recogiendo.

3. CASOS DE ESTUDIO

A continuación, se detallan cada uno de los casos de estudio realizados.

3.1. DISEÑO DE MATRICES DE TRÁFICO

En Net2Plan, el tráfico ofrecido en una red viene representado como un conjunto de demandas de tráfico asociadas al diseño de red, en el que cada demanda representa el volumen de tráfico en media (medido en *Erlangs*) que inyecta la demanda a la red desde un nodo origen a un nodo destino. En algunos casos, el conjunto de demandas de tráfico está compuesto de una demanda por cada par de nodos de la red, además las demandas son *unicast*. Cuando esto ocurre, se puede representar el conjunto de demandas usando una representación matricial, denominada **matriz de tráfico** (***traffic matrix***).

Para un diseño de red concreto que consta de *N* nodos, su matriz de tráfico asociada será una matriz de dimensiones *NxN* con ceros en la diagonal, ya que obviamente no está permitido que un nodo genere tráfico hacia sí mismo. Por tanto, la coordenada en la fila *i-ésima* y la columna *j-ésima* de la matriz, representa la cantidad de tráfico que inyecta el nodo *i* a la red con objetivo el nodo *j*, es decir, el volumen de tráfico de la demanda asociada al par de nodos *(i,j)*.

Dada una topología como la de la figura siguiente:

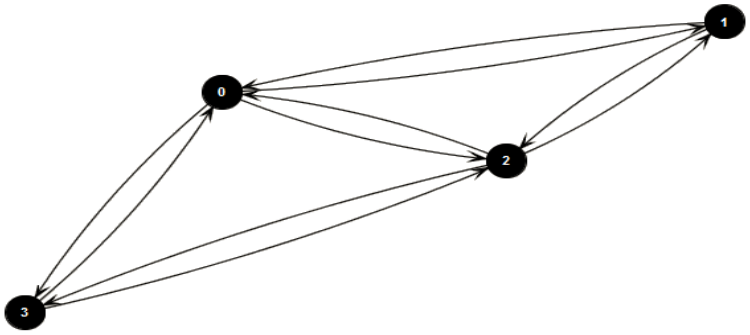


Figura 44: Diseño de red *example4nodes.n2p* incluido en el repositorio de Net2Plan

Se puede definir una matriz de tráfico para dicha red como la aquí mostrada:

$$T = \begin{pmatrix} 0 & 9 & 2 & 3 \\ 4 & 0 & 6 & 2 \\ 7 & 1 & 0 & 1 \\ 8 & 3 & 4 & 0 \end{pmatrix}$$

Figura 5: Matriz de tráfico para una red de 4 nodos

Se puede observar como el tráfico que va del nodo 0 al nodo 0 le corresponde un valor de 0 *Erlangs*, el tráfico del nodo 0 al nodo 1 un valor de 9 *Erlangs*, y así sucesivamente.

La herramienta de **diseño de matrices de tráfico (*Traffic matrix design)*** integrada en Net2Plan ayuda a los usuarios en el proceso de generación de sus propias matrices de tráfico. Permite la generación de nuevas matrices de forma manual, o usando varios modelos que se encuentran en la literatura (por ejemplo, uniformes-aleatorias, modelos de

población distancia, modelo de gravedad...). Por otra parte, las matrices creadas se guardan en formato *.n2p* para después aplicarse sobre diseños de red creados en Net2Plan. Además, los modelos de generación de matrices de tráfico están disponibles en la librería de Net2Plan y por lo tanto se pueden integrar directamente en los algoritmos de diseño de red. Para obtener más información, se debe consultar la clase `com.net2plan.libraries.TrafficMatrixGenerationModels` dentro del Javadoc incluido en la aplicación.

Las siguientes imágenes muestran la ventana del área de trabajo para esta herramienta dentro de Net2Plan. La parte superior izquierda de la ventana da acceso a un conjunto de modelos generales de generación de tráfico. Por otro lado, el usuario puede generar matrices usando un método particular: el modelo de tráfico población-distancia.

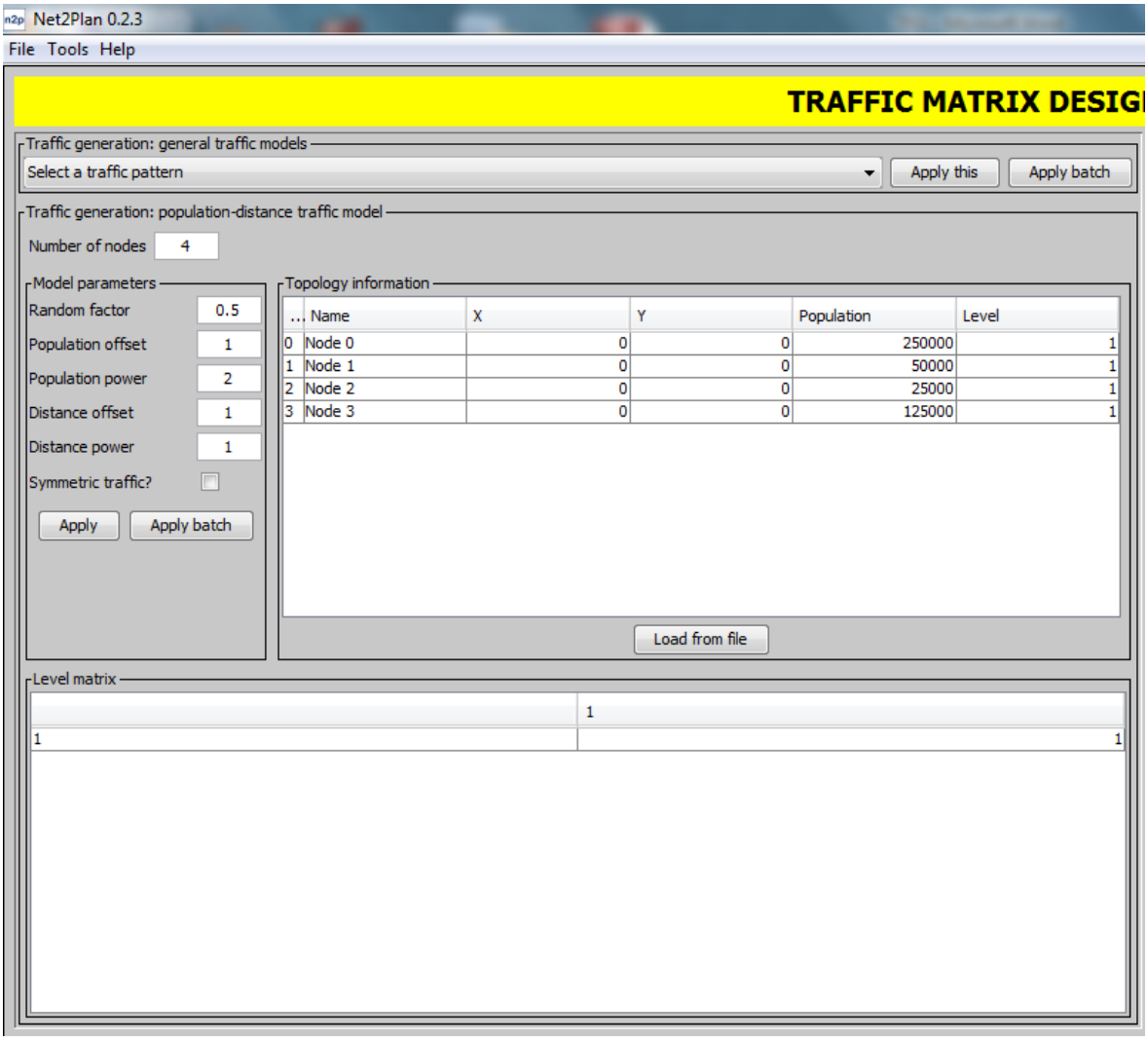


Figura 6: Ventana de diseño de matrices de tráfico (sección izquierda)

Como se puede observar, el panel derecho muestra las matrices de tráfico generadas, y permite guardar y cargar matrices en formato *.n2p*, cambiar el tamaño de estas y algunas otras modificaciones sencillas integradas en el resto de botones de la interfaz. En la parte inferior derecha, se encuentra el panel de normalización del tráfico que permite la aplicación de un método de normalización a una o todas las matrices de tráfico mostradas

en el panel superior. El panel inferior permite la selección de un método para producir un conjunto de matrices de tráfico a partir de una dada, utilizando para ello diferentes patrones.

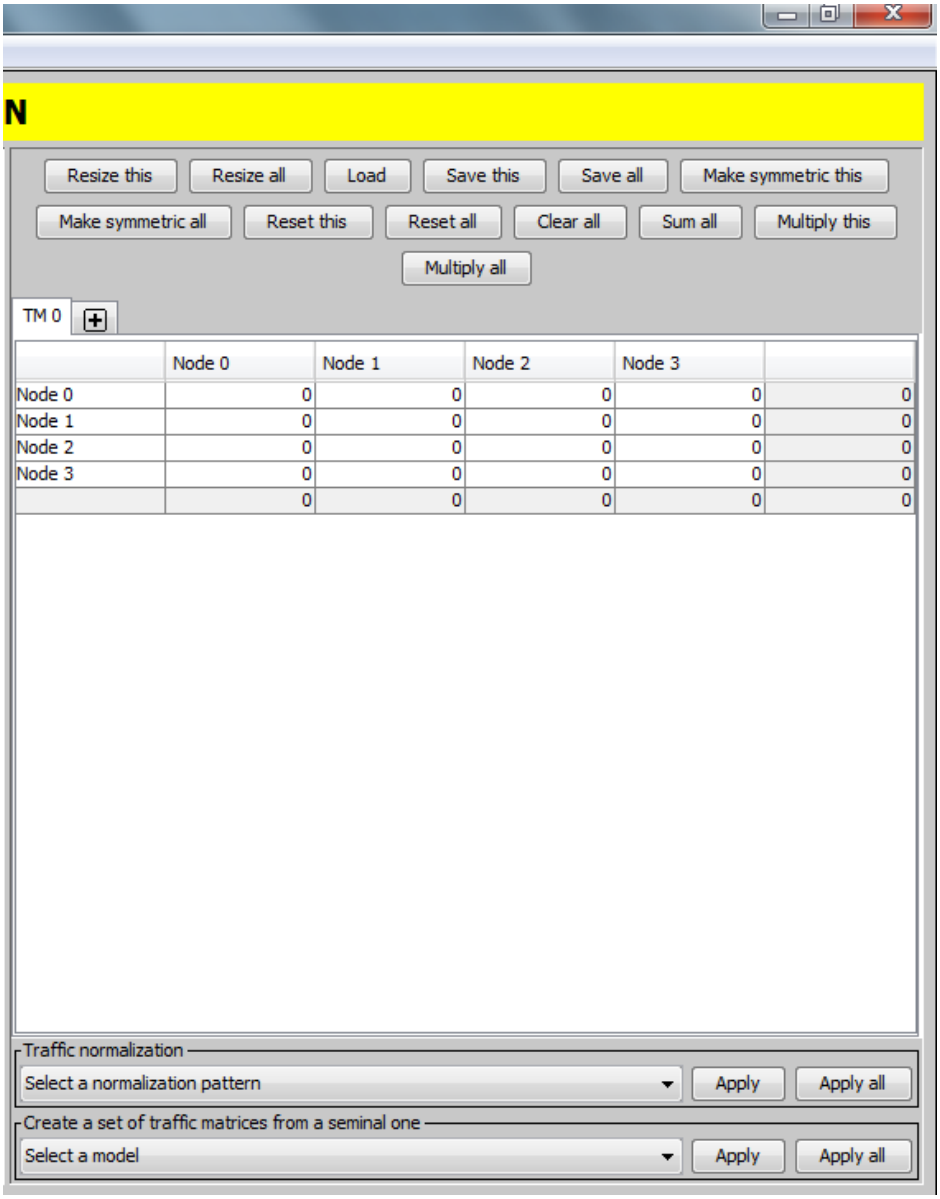


Figura 7: Ventana de diseño de matrices de tráfico (sección derecha)

Siguiendo el ejemplo anterior, es posible crear de manera manual una matriz introduciendo los valores en las respectivas casillas en el panel derecho. En la siguiente imagen se muestra la matriz representada en la Figura 5 introducida en la aplicación:

TM 0

	Node 0	Node 1	Node 2	Node 3	
Node 0	0	9	2	3	14
Node 1	4	0	6	2	12
Node 2	7	1	0	1	9
Node 3	8	3	4	0	15
	19	13	12	6	50

Figura 8: Matriz de tráfico generada en Net2Plan

Además, se incluyen una fila y una columna adicionales que muestran los valores totales de tráfico por fila y por columna, y el tráfico total de la matriz.

3.1.1. MODELOS DE TRÁFICO GENERALES

En el panel *Traffic generation: general traffic models* mostrado anteriormente, el usuario puede generar una o varias matrices de tráfico, al seleccionar uno de los siguientes patrones de generación de tráfico:

- Constante: Genera una matriz de tráfico *NxN* (siendo *N* igual al número de nodos de la red) con un valor constante en todas sus coordenadas.
- Uniforme (0,10): Genera una matriz de tráfico *NxN* (siendo *N* igual al número de nodos de la red) con valores aleatorios en el rango (0,10).
- Uniforme (0,100): Genera una matriz de tráfico *NxN* (siendo *N* igual al número de nodos de la red) con valores aleatorios en el rango (0,100).
- 50% Uniforme (0,10) y 50% Uniforme (0,100): Genera una matriz de tráfico *NxN* (siendo *N* igual al número de nodos de la red) con un 50% de sus entradas con valores aleatorios en el rango (0,100), y el resto de entradas con valores aleatorios en el rango (0,10).
- 25% Uniforme (0,10) y 75% Uniforme (0,100): Genera una matriz de tráfico *NxN* (siendo *N* igual al número de nodos de la red) con el 25% de sus entradas con valores aleatorios en el rango (0,100), y el resto de entradas con valores aleatorios en el rango (0,10).
- Modelo de *gravedad* [11]: Genera una matriz de tráfico *NxN* (siendo *N* igual al número de nodos de la red) de acuerdo con el modelo de *gravedad*. El usuario debe proporcionar para cada nodo *n*, el tráfico total generado (representado como *OUT(n)*) y el tráfico total recibido (representado como *IN(n)*). Naturalmente, la suma de todo el tráfico generado por todos los nodos, debe ser igual a la suma de todo el tráfico recibido por todos los nodos (representado ese valor como *H*). A partir de esta información, la coordenada *(i, j)* de la matriz de tráfico viene dada por:

$$\frac{OUT(i) * IN(j)}{H}$$

Entonces, el tráfico desde el nodo *i* al nodo *j* es proporcional al tráfico total producido en *i* y proporcional al tráfico total recibido en *j*.

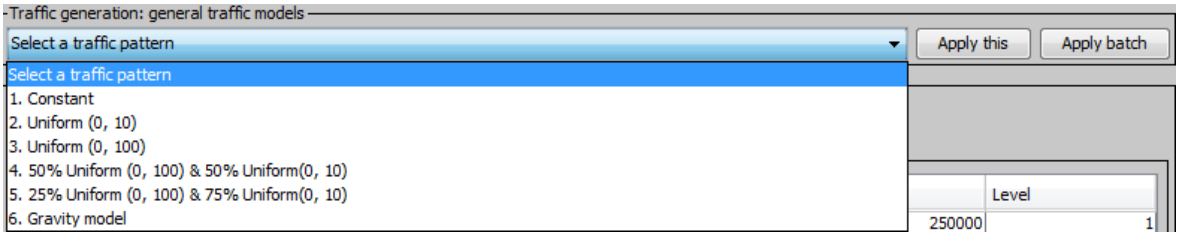


Figura 9: Modelos de tráfico generales

En cualquier caso, los valores de la diagonal de la matriz de tráfico son siempre cero, ya que las auto-demandas (demandas de un nodo hacia sí mismo) no están permitidas. Al pulsar el botón "*Apply this*", se crea una matriz de tráfico, y se muestra en el panel derecho. A su vez, el botón "*Apply batch*" crea un número arbitrario de matrices, que también se muestran en el panel derecho.

3.1.2. MODELO POBLACIÓN–DISTANCIA

En el panel *Traffic generation: population-distance traffic model* el usuario puede generar una o varias matrices de tráfico, utilizando el modelo población-distancia de matriz de tráfico descrito en [11]. Este modelo recibe como entrada el número de nodos en la red, y para cada nodo, su posición *(x, y)*, su población y un factor llamado *nivel de nodo*. Esta información se puede introducir manualmente en el panel de información de topología, o se carga desde un archivo *.n2p*, donde los nodos tienen la población y factor nivel de nodo definido. En el modelo de población - distancia, el tráfico γ_{ij} del nodo *i* al nodo *j* se calcula con la siguiente expresión:

$$\gamma_{ij} = (1 - rf + 2 * rf * rand()) * Level(L_i, L_j) \frac{\left(\frac{Pop_i * Pop_j}{Pop_{max}^2} + Pop_{off}\right)^{Pop_{power}}}{\left(\frac{dist(i,j)}{dist_{max}} + dist_{off}\right)^{dist_{power}}}$$

- **Factor aleatorio *(1−rf+2·rf·rand()) ∈ [1−rf,1+rf]:*** Este factor aleatorio es controlado por el parámetro de entrada *rf ∈ [0,1]*, que controla la cantidad de aleatoriedad en el modelo. Si *rf = 0* el factor *(1−rf+2·rf·rand())* es siempre igual a 1, y las coordenadas en la matriz de tráfico no son al azar. *Si rf = 1*, el factor de azar es igual a *1-rand()*, y por lo tanto es una variable aleatoria uniforme entre *[0,2]*. Los valores intermedios de *rf* son variables aleatorias uniformes en el intervalo *[1-rf, 1+rf]*.
- **Factor de nivel de nodo *Level(L_i,L_j):*** Este factor permite multiplicar el tráfico entre dos nodos por una constante dependiendo del nivel de cada nodo. Los valores de la matriz *Level(L_i,L_j)* se definen en una matriz de dimensiones *LxL* siendo *L* el número de niveles o tipos de nodos definidos por el usuario.

  Por ejemplo, dada una red con dos tipos de nodos, "clientes" y "servidores". Se pretende crear una matriz de tráfico para esta red donde los clientes no envían tráfico a clientes y los servidores no envían tráfico a servidores. Esto puede hacerse definiendo dos niveles de nodos en la red separando "nodos cliente" y "nodos servidor". Para ello, se puede utilizar una matriz de nivel con ceros en las

diagonales, por lo que el tráfico cliente-cliente y el tráfico servidor-servidor se multiplica por 0 en el modelo.

- **Factor de población** $\left(\frac{Pop_i * Pop_j}{Pop_{max}^2} + Pop_{off}\right)^{Pop_{power}}$**:** El factor de población hace que el tráfico entre dos nodos sea proporcional al producto de la población normalizada de ambos nodos, dividiendo el producto por la población máxima de un nodo. El factor Pop_{off} se utiliza para suavizar los efectos del producto (por ejemplo, si una población es 0, el tráfico todavía no es cero). El factor Pop_{power} controla el efecto de la población en las matrices. Los valores típicos son:
  - Pop_{power} = 1 en modelos basados en la llamada "atracción gravitatoria".
  - Pop_{power} = 0 si el tráfico es independiente de la población.

- **Factor de distancia** $\left(\frac{dist(i,j)}{dist_{max}} + dist_{off}\right)^{dist_{power}}$**:** El factor de distancia hace que el tráfico entre dos nodos sea inversamente proporcional a la distancia entre ellos. Los valores $dist_{off}$, $dist_{max}$ y $dist_{power}$ tienen una función similar como en el factor de población: evitar cualquier posible división por 0. Los valores típicos son:
  - $dist_{power}$ = 2 en los modelos basados en la llamada "atracción gravitatoria".
  - $dist_{power}$ = 3 en los modelos basados en el llamado "magnetismo".

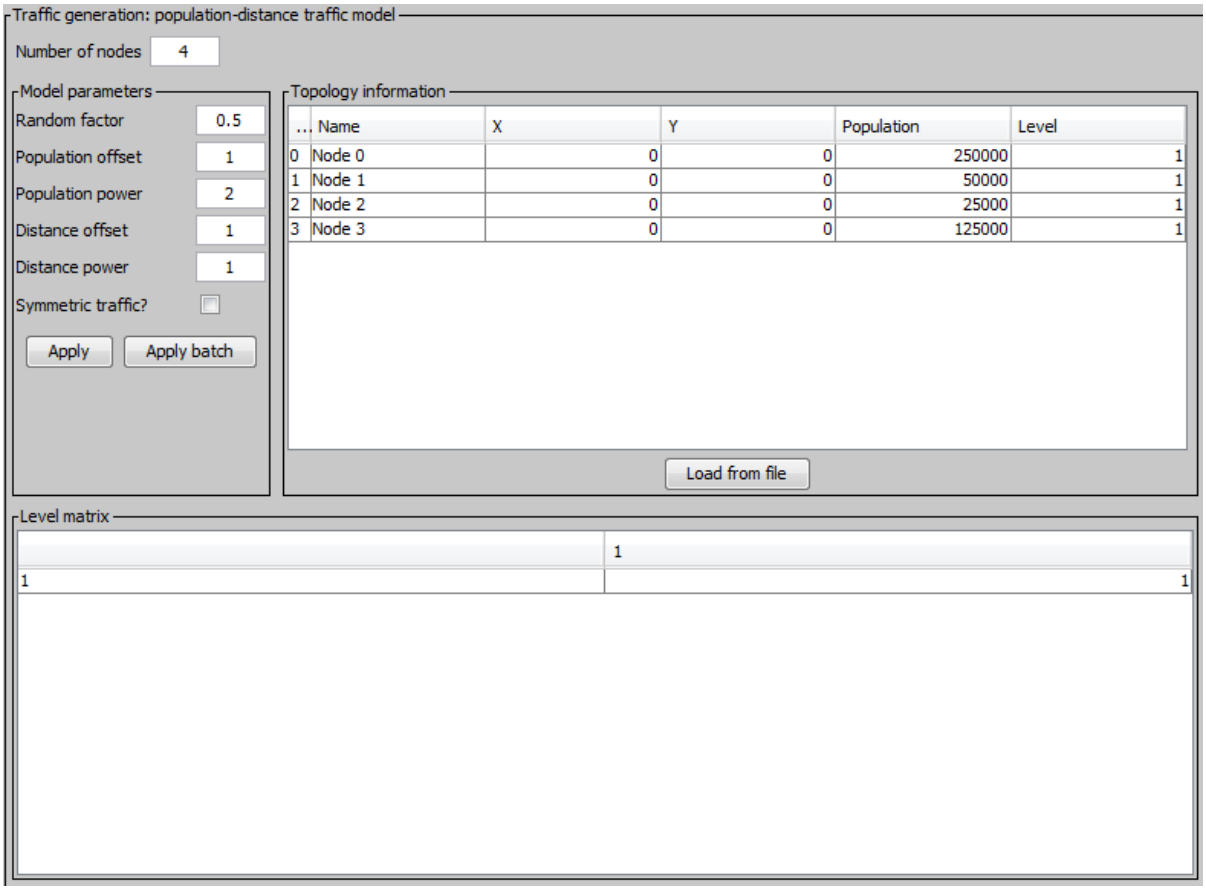


Figura 10: Generador de tráfico según el modelo población-distancia

Al igual que ocurría con los modelos generales de tráfico, al pulsar el botón "*Apply this*", se crea una matriz de tráfico, y se muestra en el panel derecho. Mientras que, el botón "*Apply batch*" crea un número arbitrario de matrices, que también se muestran en el panel derecho.

3.1.3. NORMALIZACIÓN DE TRÁFICO

El menú desplegable de normalización del tráfico que se encuentra en la parte inferior derecha de la ventana, permite seleccionar un método de normalización para aplicar a la matriz de tráfico visible en la pestaña activa de la interfaz gráfica (usando el botón "*Apply*") o a todas las matrices de tráfico (usando el botón "*Apply all*"). Se implementan cuatro métodos de normalización: total, por fila, por columna, máximo tráfico que puede ser cursado [12]. Se detallan a continuación:

- **Normalización total**: El objetivo de la normalización total, es la modificación de una matriz de tráfico M_T, de manera que en la M'_T normalizada, la cantidad total de tráfico generado es igual a un valor M definido por el usuario. Esto se hace multiplicando todos los elementos de la matriz de tráfico original por un factor constante de acuerdo con la expresión:

$$M'_T(i,j) = M_T(i,j)\frac{M}{\sum_{(i,j)} M_T(i,j)}$$

- **Normalización por fila:** El objetivo de la normalización por fila, es la modificación de una matriz de tráfico M_T, de manera que en la M'_T normalizada, la cantidad total de tráfico generado por cada nodo *i*, es igual a un valor M_i definido por el usuario. Esto se hace multiplicando todos los elementos de la fila *i-ésima* de la matriz de tráfico original por el mismo factor constante de acuerdo con la expresión:

$$M'_T(i,j) = M_T(i,j)\frac{M_i}{\sum_j M_T(i,j)^j}$$

- **Normalización por columna:** El objetivo de la normalización por columna, es la modificación de una matriz de tráfico M_T, de manera que en la M'_T normalizada, la cantidad total de tráfico recibido para cada nodo *j*, es igual a un valor M_j definido por el usuario. Esto se hace multiplicando todos los elementos de la columna *j-ésima* de la matriz de tráfico original por el mismo factor constante de acuerdo con la expresión:

$$M'_T(i,j) = M_T(i,j)\frac{M_j}{\sum_i M_T(i,j)^j}$$

- **Normalización al máximo tráfico que puede ser cursado:** El objetivo de esta normalización es multiplicar la matriz de tráfico M_T por un factor α constante de manera que la matriz normalizada $\alpha \cdot M_T$, es la matriz de tráfico más grande que puede ser cursada por una red concreta usando el encaminamiento óptimo (sin sobrecargar los enlaces). El usuario puede elegir entre dos formas de calcular esta normalización:
    - Método estimativo: Este método produce una matriz que representa el tráfico máximo que un encaminamiento óptimo podría cursar. Para cada par de nodos en la red *(i, j)*, se calcula el número de saltos siguiendo el camino más corto entre esos dos nodos, llamado SP_{ij}. Entonces, la cantidad $\alpha \cdot M_T(i,j) \cdot SP_{ij}$ es la cantidad mínima posible de ancho de banda para ese enlace que el tráfico entre los nodos podría consumir en cualquier encaminamiento. Por tanto, la matriz normalizada $\alpha \cdot M_T$ es tal que la suma de esta cantidad a lo largo de todas las coordenadas, es igual a la cantidad total de ancho de banda sumado para todos los enlaces de la red $\sum_e u_e$. En otras palabras $\alpha = \frac{\sum_e M_T(i,j) \cdot SP_{ij}}{\sum_e u_e}$. Cabe destacar que esto puede producir que una matriz de tráfico sea más grande que lo que cualquier encaminamiento práctico (en que por ejemplo ningún enlace esté saturado) pueda cursar.
    - Método exacto: Este método resuelve el problema lineal que calcula exactamente el factor α máximo para el cual la matriz $\alpha \cdot M_T$ todavía tiene un encaminamiento factible en la red. Esta formulación se resuelve utilizando la biblioteca JOM. El *solver* utilizado y la ubicación de su .DLL / .so se obtiene a partir de los valores por defecto en el menú de opciones de Net2Plan.

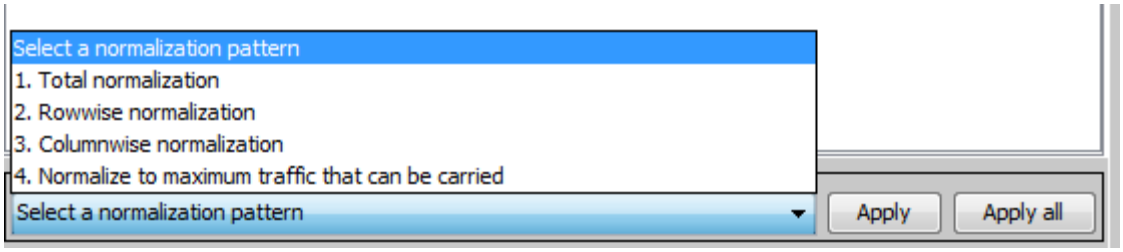


Figura 11: Patrones de normalización de matrices de tráfico

3.1.4. GENERACIÓN DE MATRICES A PARTIR DE UNA MATRIZ DE REFERENCIA

En algunas ocasiones, los estudios de diseño de red requieren un conjunto de matrices de tráfico, que por ejemplo reflejen las previsiones de tráfico en los años siguientes, de acuerdo con un crecimiento previsto. Además, puede ser necesario para producir variaciones aleatorias a partir de una sola matriz de tráfico, así como para comprobar cómo varía el rendimiento de la red si el tráfico fluctúa. Para este u otros fines, Net2Plan ofrece los siguientes métodos:

- **Nuevas matrices con una tasa de crecimiento anual compuesto**: Se genera una secuencia de matrices de tráfico, donde cada una representa el tráfico en uno de los años venideros, utilizando como año cero la matriz de referencia. Cada matriz es igual a la matriz del año anterior, multiplicada por un factor *(1 + CAGR)*, donde *CAGR* (*Compound Annual Growth Rate*) es la tasa de crecimiento anual compuesto.
- **Variaciones aleatorias uniformes**: Se genera un conjunto de matrices a partir de una de referencia, donde cada coordenada de cada una de ellas viene dada por una muestra de una variable aleatoria uniforme con promedio *x* (el valor de la coordenada en la matriz de referencia), y una variación relativa máxima definida por el usuario (en el intervalo [0, 1]). Por ejemplo, un valor 0,4 de variación relativa máxima significa que la nueva coordenada será tomada de manera uniforme entre *(1 - 0,4) x* y *(1 + 0,4) x*.
- **Variaciones aleatorias gaussianas:** Se crea un conjunto de matrices a partir de una de referencia de manera que para cada nueva matriz, cada coordenada viene dada por la misma coordenada en la matriz de referencia, además de una muestra de una variable aleatoria gaussiana, con valor medio *x* (el valor de la coordenada en la matriz de referencia), y un coeficiente de variación definido por el usuario (cociente entre la desviación estándar y la media), así como un máximo de variación relativa también definido por el usuario. Este último valor se utiliza para truncar la muestra. Por ejemplo, un valor 0,4 de variación relativa máxima significa que si la muestra está por debajo de *(1-0,4) x* entonces el valor que se produce es *(1-0,4) x*, y si la muestra es mayor que *(1+0.4) x*, el valor tomado es *(1+0.4) x*. Por último, si la muestra es inferior a 0, la coordenada se establece en 0 (ya que no puede haber tráfico negativo).

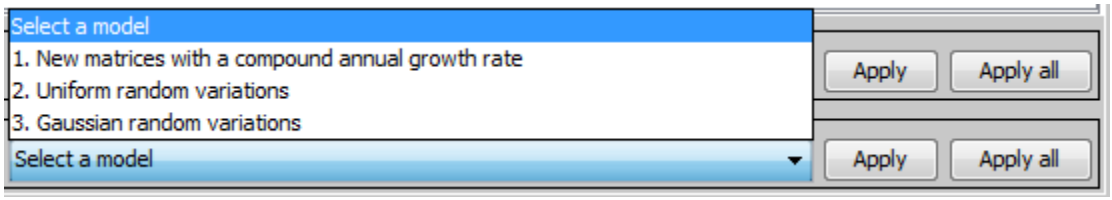


Figura 12: Modelos de generación a partir de una matriz de referencia

3.1.5. OBJETIVOS DEL CASO DE ESTUDIO

La finalidad de este caso de estudio es mostrar al usuario de Net2Plan cómo usar esta herramienta de generación de matrices de tráfico de una forma sencilla, introduciendo todos los conceptos teóricos citados anteriormente en este apartado, de manera que el vídeo-tutorial describa paso a paso cada una de las funcionalidades que proporciona la herramienta, poniéndolas en práctica desarrollando ejercicios reales.

3.1.6. ESTRUCTURA DEL VÍDEO-TUTORIAL

Se inicia el vídeo-tutorial con una introducción sobre el concepto de matriz de tráfico en Net2Plan, mostrando el ejemplo aquí plasmado al inicio del caso de estudio. Se apoya la explicación con la siguiente diapositiva:

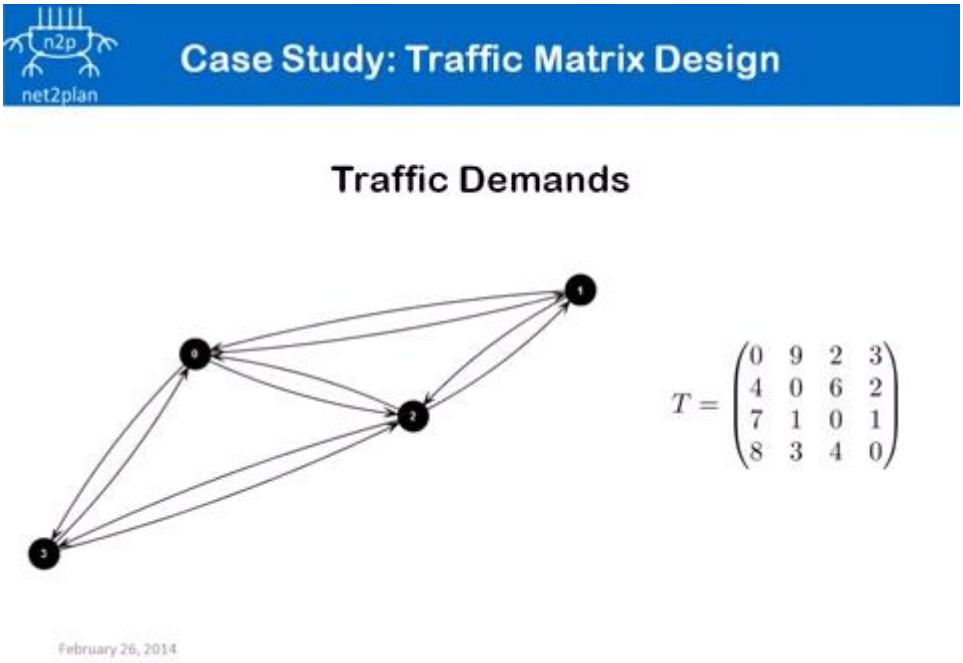


Figura 13: Introduciendo el concepto de matriz de tráfico

Después se inicia la aplicación, abriendo la herramienta *Traffic matrix design*, y posteriormente, descripción a grandes rasgos de la interfaz de usuario explicando las distintas secciones que componen la ventana.

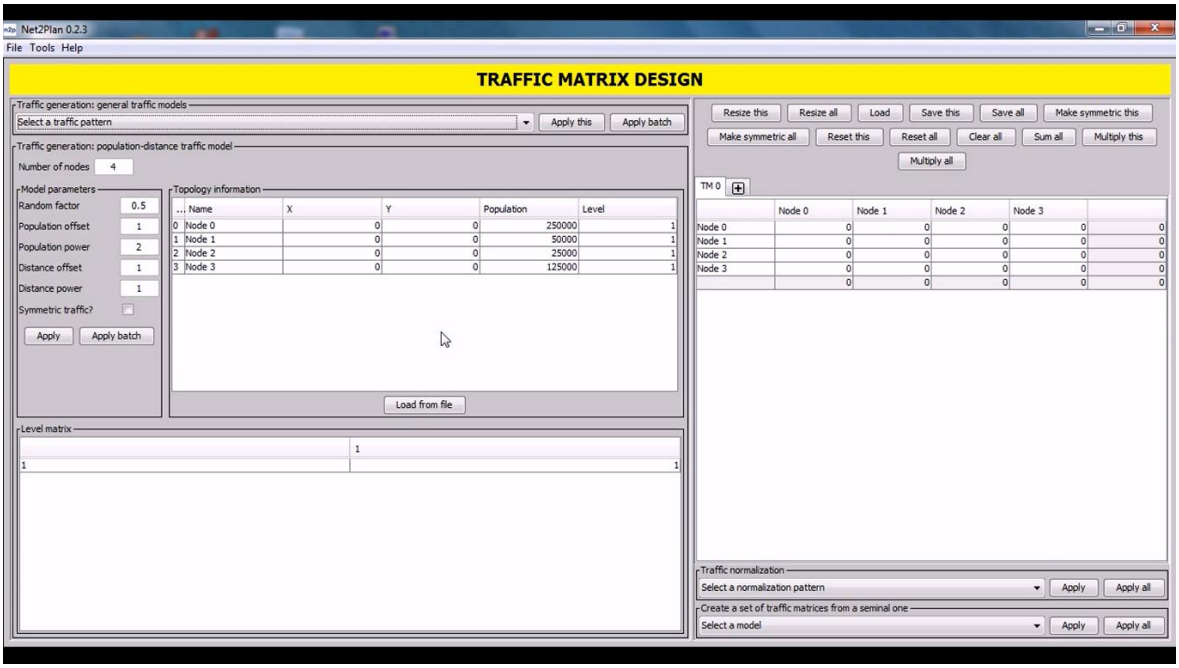


Figura 14: Explicación de la interfaz de *Traffic matrix design*

El siguiente paso consiste en la inserción de una matriz de tráfico real en la herramienta, siguiendo el ejemplo citado al inicio del caso de estudio (Figura 5), y en la explicación de cada uno de los botones superiores de la parte derecha de la interfaz, que permiten realizar modificaciones en la matriz creada manualmente.

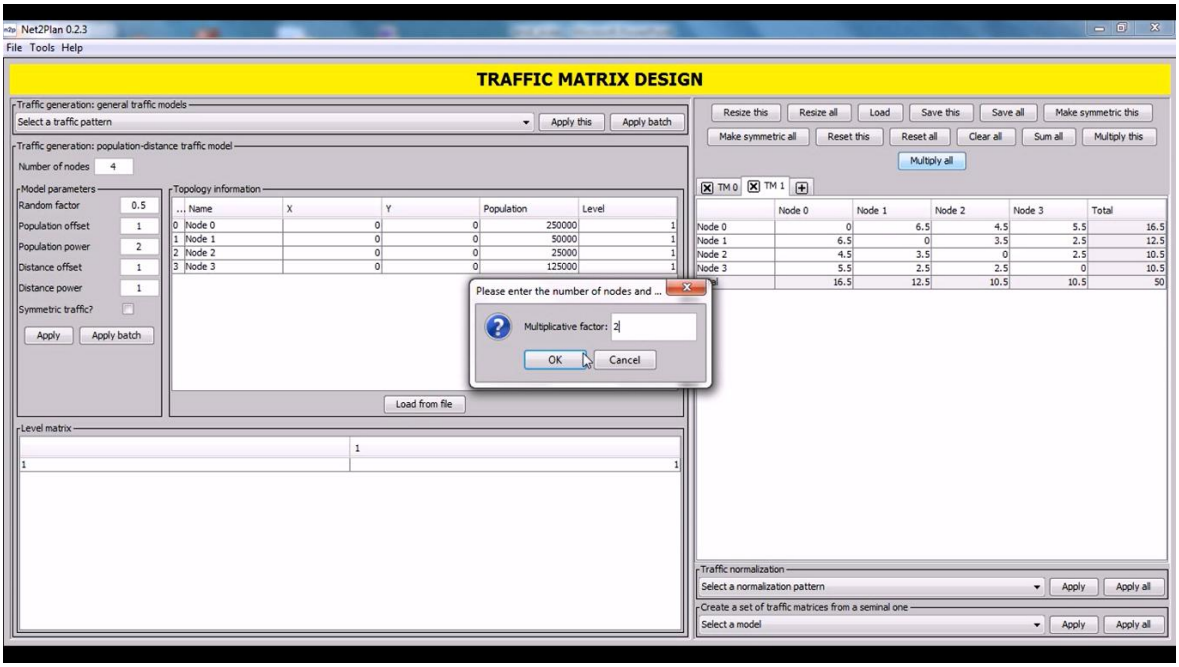


Figura 15: Multiplicación de la matriz insertada manualmente

Posteriormente, se genera de manera automática un conjunto de matrices de tráfico usando los modelos generales y el modelo población-distancia. Para mostrar al usuario las distintas posibilidades, se generan en algunas ocasiones varias matrices usando un modelo y en otras, una única matriz para el modelo empleado.

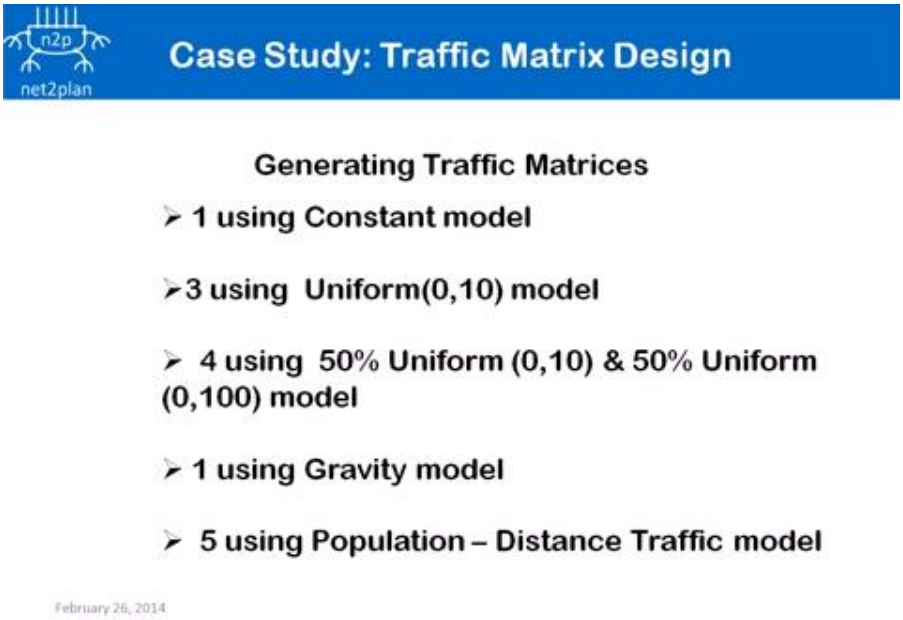


Figura 16: Conjunto de matrices generadas en el vídeo-tutorial

De entre todos los modelos empleados, se hace mayor hincapié en el uso del modelo población-distancia por ser el más complejo, mostrando la utilidad de la matriz de Nivel de Nodo. En la siguiente imagen se representa el ejemplo de tráfico entre "nodo cliente" y "nodo servidor", de manera que la diagonal queda rellena de ceros, y el tráfico de nodos servidores a nodos clientes es mucho más intenso que al contrario.

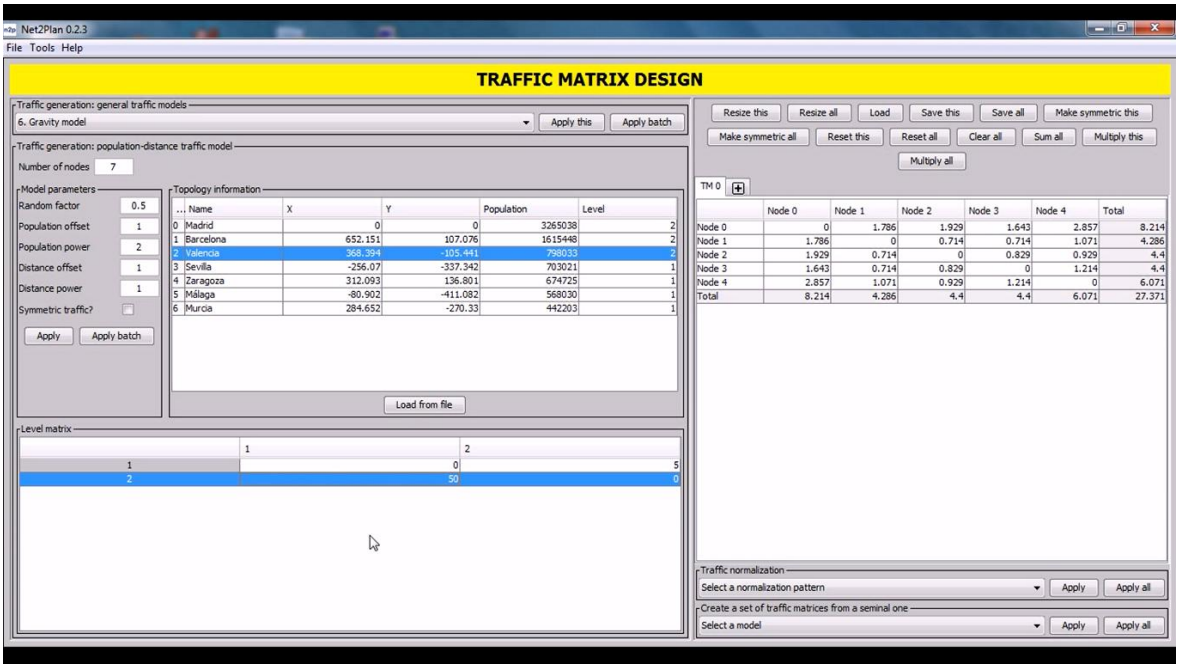


Figura 17: Aplicación de la matriz "nivel de nodo"

A lo largo del vídeo, también se muestra al usuario cómo guardar las matrices generadas para su posterior uso en otras herramientas de Net2Plan, como *Offline network design*.

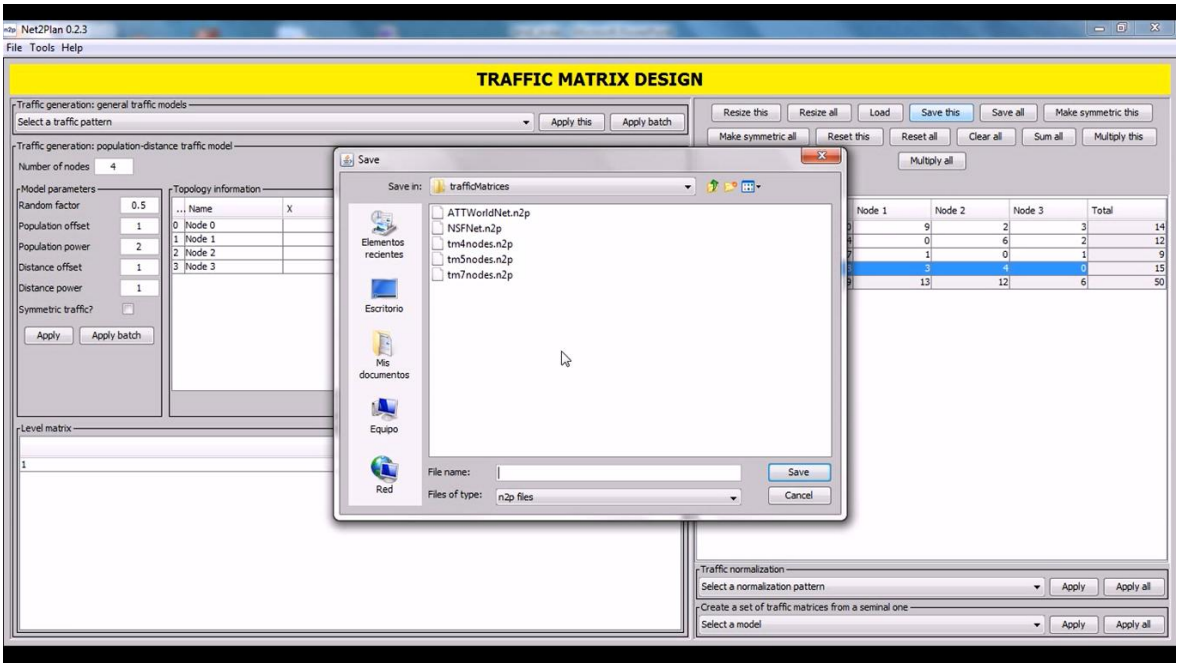


Figura 18: Guardando una de las matrices generadas

El siguiente paso habitual tras generar una matriz de tráfico es la normalización. En este caso, se normalizan varias matrices de tráfico, aplicando los distintos modelos explicados anteriormente.

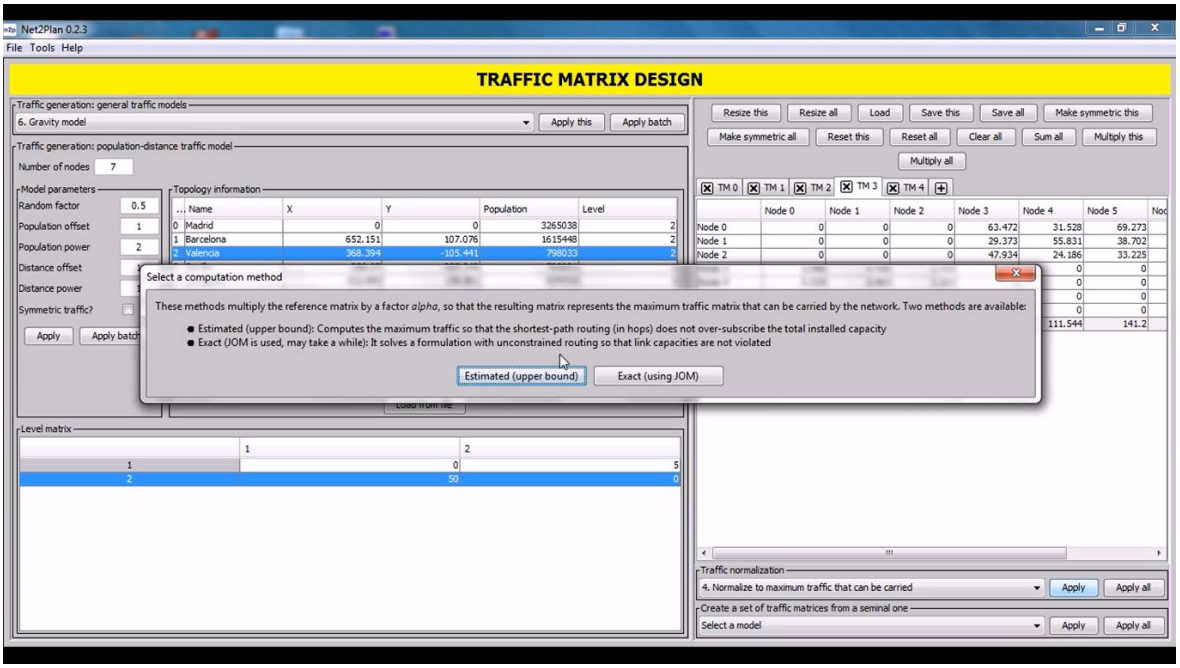


Figura 19: Normalización al máximo tráfico que puede ser cursado por la red

Como último paso que completa la descripción de todas las funcionalidades de esta herramienta, se generan un conjunto de matrices a partir de una dada, usando los distintos patrones desarrollados en la herramienta.

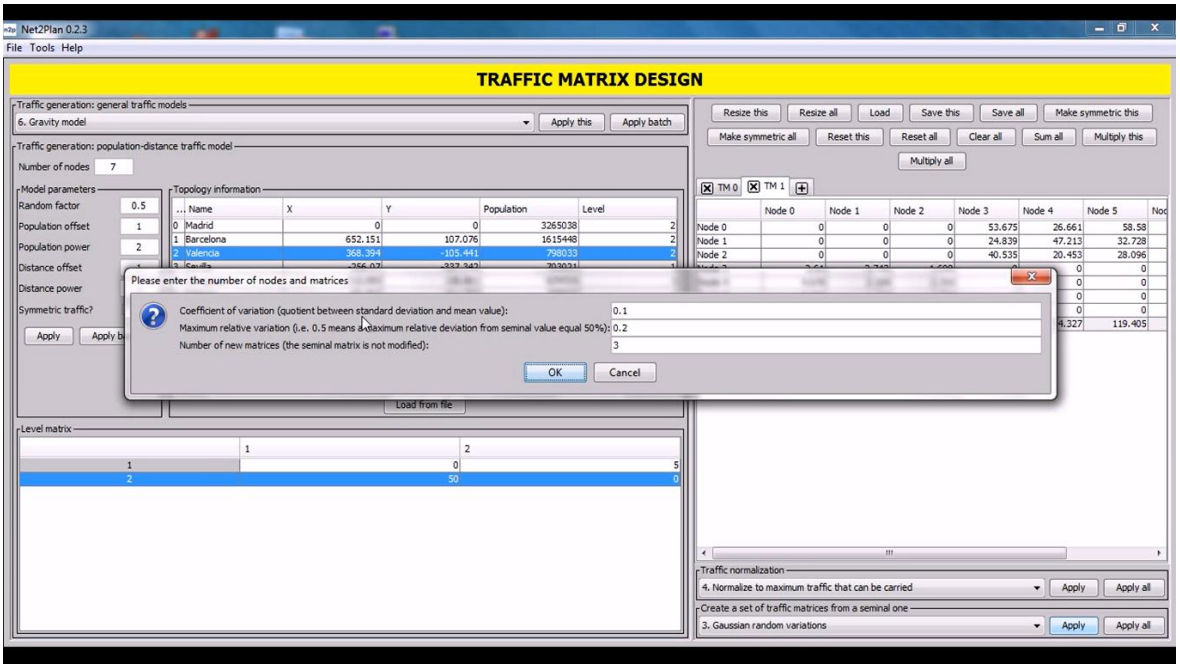


Figura 20: Generación de matrices a partir de una dada utilizando un modelo aleatorio gaussiano

Tras la grabación del vídeo-tutorial, el siguiente paso es la integración del mismo en un canal de YouTube [8], para su difusión al conjunto de usuarios de la herramienta. En este

caso, el título del vídeo es "*Getting started with Net2Plan: Traffic Matrix Design tool*". Puede ser visualizado en [8] o accediendo a la URL https://www.youtube.com/watch?v=EZOSugFmGuY.

Figura 21: Captura del vídeo-tutorial *Traffic matrix design*

3.2. GENERACIÓN DE INFORMES

Como ya se comentó al inicio, Net2Plan permite la generación de informes, ya sean definidos por el usuario o los integrados en la aplicación, de cualquier diseño de red. La herramienta de generación de informes está integrada dentro de todas las funcionalidades de Net2Plan, excepto en *Traffic matrix design*, por lo que es posible crear informes que recogen medidas de rendimiento de muy diversos tipos. Una vez que el diseño de red se ha completado, los usuarios pueden analizar, evaluar y recoger estadísticas del diseño por medio de estos informes.

Tal como ocurre con los algoritmos, los informes son clases Java que implementan la interfaz `com.net2plan.interfaces.networkDesign.IReport`, cuya información puede ser consultada en el Javadoc de Net2Plan.

3.2.1. OBJETIVOS DEL CASO DE ESTUDIO

La finalidad de este caso de estudio es mostrar al usuario de Net2Plan como usar esta funcionalidad de generación de informes asociados a diseños de red, usando concretamente la herramienta *Offline network design*, y por otra parte, introducir los conceptos necesarios para desarrollar algoritmos que generen informes personalizados a las necesidades del usuario.

3.2.2. ESTRUCTURA DEL VÍDEO-TUTORIAL

Al comienzo del vídeo-tutorial se introduce el concepto de *Report* en Net2Plan. Se muestra al usuario la manera de generar un informe asociado a un diseño de red concreto (empleado el diseño de red incluido en Net2Plan `NSFNet_N14_E42_complete.n2p`), usando la herramienta *Offline network design*, pero como se dijo en la introducción, el

resto de funcionalidades salvo *Traffic matrix design*, permite la generación de informes con una interfaz gráfica similar a la de herramienta empleada en este caso de estudio. Para este ejemplo, se aplica el informe `Report_networkDesign.java`. Este informe genera información relativa al diseño de red en cuanto a medidas de tráfico, capacidad y topología se refiere.

Hay un conjunto de informes incluidos en la aplicación, destinados a distintos objetivos, en función de las características del diseño de red, la herramienta donde se aplique, y las estadísticas que se quieran recoger en cada momento. Se puede ver el conjunto de informes predefinidos en la web de Net2Plan, entre ellos se encuentra el aplicado en este ejemplo.

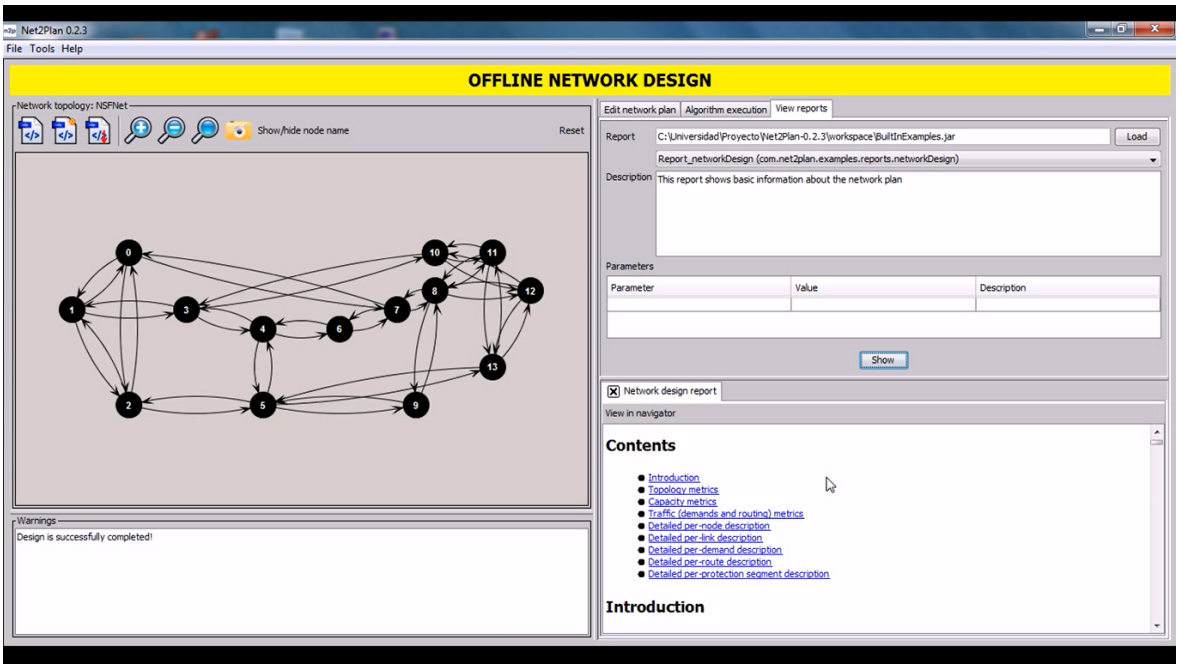


Figura 22: Generando un informe sobre el diseño de red

Una vez generado el informe, este aparece en la interfaz de usuario como se puede observar en la figura de arriba. Se explica al usuario la información contenida en el mismo, así como las diferentes estadísticas que se recogen. Para una mejor visualización se permite abrir el informe en un navegador.

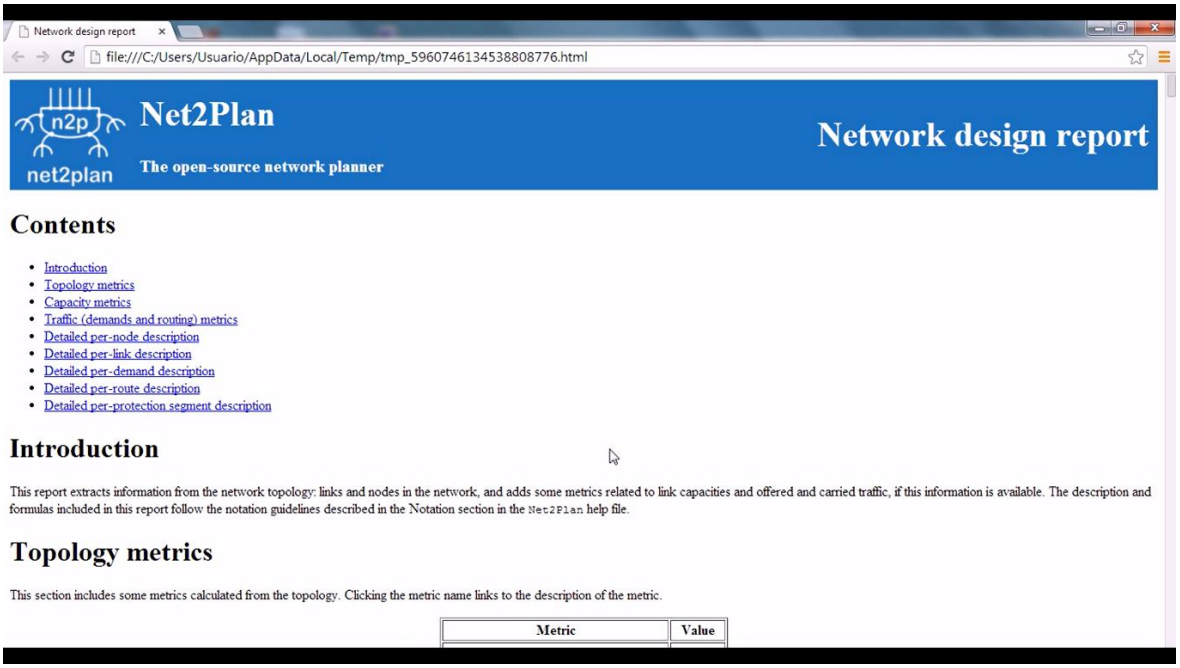


Figura 23: Visionado del informe en el navegador web

A continuación, se pretende modificar el informe mostrado con el objetivo de que presente algún estadístico más, para mostrar al usuario el proceso de generación de informes. Para ello, se hace uso de un entorno de programación Java (*IDE*, Integrated Development Environment). El usuario puede utilizar cualquier IDE para desarrollar algoritmos o informes, pero en este proyecto se ha escogido la suite de *Eclipse* [13] como entorno. Obviamente, *Eclipse*, *Java Runtime* y *Java JDK* (versión 7 o superior) deben estar instalados y configurados en el sistema con anterioridad.

Para facilitar el proceso de puesta a punto del IDE, se muestra al usuario cómo configurar *Eclipse*, incluyendo las librerías externas de Net2Plan, necesarias para el desarrollo de código, y asimismo, como introducir el código del informe `Report_networkDesign.java` de dos maneras distintas: desde la web de Net2Plan descargando directamente el archivo concreto, o desde el archivo *.jar* que contiene la aplicación tras la descarga de la misma. Para este caso, se ejecuta la segunda opción al tratarse de un informe que lleva asociado una plantilla *HTML* junto con un conjunto de figuras e imágenes para su representación en la web, de forma que el archivo *.jar* ya lleva incluida toda esta información.

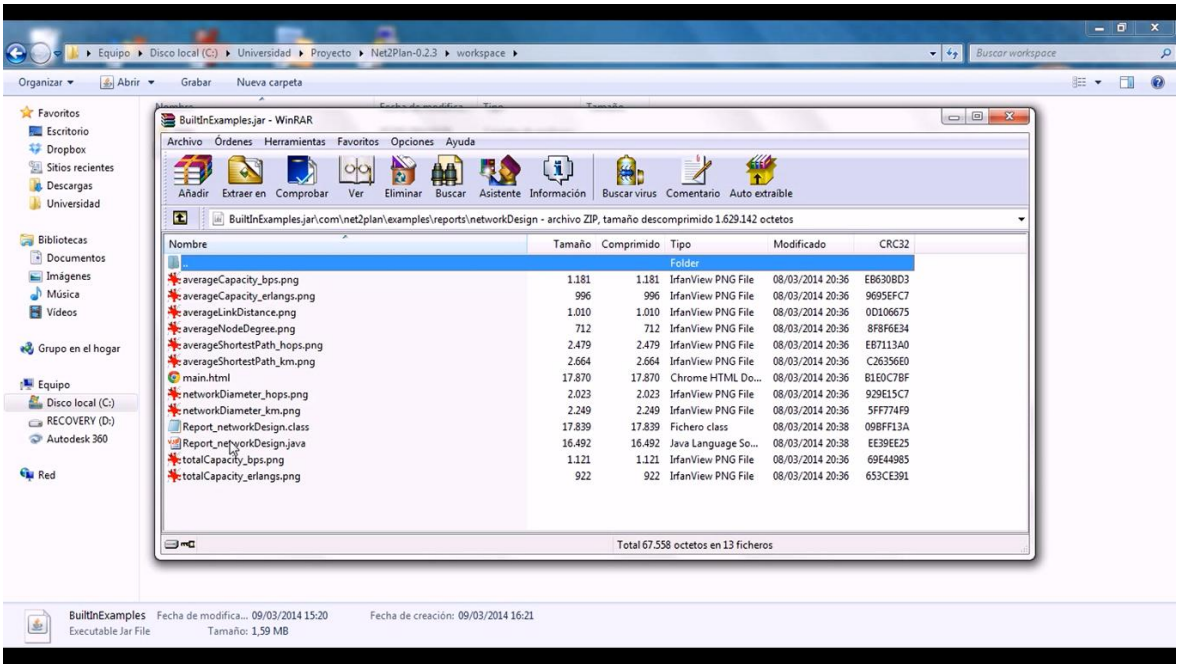


Figura 24: Descomprimiendo los archivos necesarios para la modificación del código

Posteriormente, se explica el código del informe, detallando el conjunto de métodos que describe la interfaz `IReport`, y que deben implementarse en los informes. Tras la explicación, se pasa a modificar el actual código con el fin de introducir una nueva métrica. Esa nueva métrica indica si el tráfico es simétrico o no. Es decir, comprueba si la topología física tiene el mismo número de enlaces entre cada par de nodos en ambas direcciones (suponiendo múltiples grafos) y mismos pesos por cada dirección. A continuación se muestran las partes del código modificado con el nombre `Report_networkDesign_modified.java`:

```
public String executeReport(NetPlan netPlan, Map<String, String>
reportParameters, Map<String, String> net2planParameters)
	{
		int N = netPlan.getNumberOfNodes();
		int E = netPlan.getNumberOfLinks();
		int D = netPlan.getNumberOfDemands();
		int R = netPlan.getNumberOfRoutes();
		int S = netPlan.getNumberOfProtectionSegments();

		double averageNodeDegree = N == 0 ? 0 : (double) E / (double) N;
		int[][] linkTable = netPlan.getLinkTable();
		DoubleMatrix1D d_e =
DoubleFactory1D.dense.make(netPlan.getLinkLengthInKmVector());
		DoubleMatrix1D u_e =
DoubleFactory1D.dense.make(netPlan.getLinkCapacityInErlangsVector());
		double averageLinkLength = 0;
		double maxCapacity = 0;
		double minCapacity = 0;
		double averageCapacity = 0;
		double totalCapacity = 0;
		int networkDiameter_hops = 0;
```

```
double networkDiameter_km = 0;
double averageShortestPath_hops = 0;
double averageShortestPath_km = 0;
double capacityModule = 0;

boolean isConnected = false;
boolean isBidirectional = false;
boolean isWeightedBidirectional = false;
boolean isSimple = false;
// introduce the new variable
boolean isTrafficSymmetric = false;
```

En el método *executeReport()* se genera una nueva variable booleana con el nombre de la métrica que queremos introducir.

```
if (E > 0)
{
	isConnected = GraphUtils.isConnected(netPlan);
	isBidirectional = GraphUtils.isBidirectional(netPlan);
	isWeightedBidirectional = GraphUtils.isWeightedBidirectional(netPlan,
u_e.toArray());
	isSimple = GraphUtils.isSimple(netPlan);

	// generate a value with the auxiliary function
	int[][] demandTable = netPlan.getDemandTable();
	double[] h_d = netPlan.getDemandOfferedTrafficInErlangsVector();
	isTrafficSymmetric = GraphUtils.isWeightedBidirectional(demandTable,
h_d, N);
```

A continuación, se genera el valor de la nueva métrica haciendo uso de la función *isWeightedBidirectional()*, recogida dentro del Javadoc de Net2Plan, que calcula exactamente lo definido para el nuevo estadístico al inicio del caso de estudio. Tras esto, se inserta el valor de la nueva métrica en la plantilla HTML que lleva asociada este informe de la siguiente manera:

```
// Put capacity metrics
html = html.replaceFirst("#capacityMetrics_maxCapacity#", String.format("%.3f
(%.3f)", maxCapacity, maxCapacity * binaryRate));

html = html.replaceFirst("#capacityMetrics_minCapacity#", String.format("%.3f
(%.3f)", minCapacity, minCapacity * binaryRate));

html = html.replaceFirst("#capacityMetrics_averageCapacity#",
String.format("%.3f (%.3f)", averageCapacity, averageCapacity * binaryRate));

html = html.replaceFirst("#capacityMetrics_totalCapacity#", String.format("%.3f
(%.3f)", totalCapacity, totalCapacity * binaryRate));

html = html.replaceFirst("#capacityMetrics_capacityModule#",
String.format("%.3f (%.3f)", capacityModule, capacityModule * binaryRate));
```

```
DoubleMatrix1D h_d =
DoubleFactory1D.dense.make(netPlan.getDemandOfferedTrafficInErlangsVector());

DoubleMatrix1D r_d =
DoubleFactory1D.dense.make(netPlan.getDemandCarriedTrafficInErlangsVector());

double H_d = h_d.zSum();
double R_d = r_d.zSum();

html = html.replaceFirst("#trafficMetrics_totalOfferedTraffic#",
String.format("%.3f (%.3f)", H_d, H_d * binaryRate));

html = html.replaceFirst("#trafficMetrics_totalCarriedTraffic#",
String.format("%.3f (%.3f)", R_d, R_d * binaryRate));

html = html.replaceFirst("#trafficMetrics_averageOfferedTraffic#",
String.format("%.3f (%.3f)", D == 0 ? 0 : H_d / D, D == 0 ? 0 : H_d / D *
binaryRate));

html = html.replaceFirst("#trafficMetrics_averageCarriedTraffic#",
String.format("%.3f (%.3f)", D == 0 ? 0 : R_d / D, D == 0 ? 0 : R_d / D *
binaryRate));

html = html.replaceFirst("#trafficMetrics_blockedTraffic#",
String.format("%.3f", H_d == 0 ? 100 : 100 * (1 - R_d / H_d)));

html = html.replaceFirst("#trafficMetrics_numHops#", String.format("%.3f", R_d
== 0 ? 0 : Y_e / R_d));

// introduce in the html code
html = html.replaceFirst("#trafficMetrics_isSymmetric#", String.format("%s",
isTrafficSymmetric ? "Yes" : "No"));
```

El código superior representa el conjunto de métricas definidas usando la función *replaceFirst()*, que busca el primer elemento en la plantilla HTML asociada al informe (en este caso) que coincida con el elemento pasado como argumento, asociando a cada elemento unas características de formato.

De la misma forma, para que al invocar a la función *replaceFirst()*, esta encuentre el elemento que se le ha pasado como argumento se debe modificar la plantilla HTML, incluyendo la etiqueta asociada a la nueva métrica. El código incluido en la plantilla es el siguiente:

```
<a name='sectionTrafficMetrics'><h1>Traffic (demands and routing)
metrics</h1></a>
<p>This section includes some metrics calculated from traffic demands
and routing (if available). Clicking the metric name links to the
description of the metric.</p>
<center>
<table border="1">
      <tr><th><b>Metric</b></th><th><b>Value</b></th></tr>
      <tr><td><a href='#metrics_totalOfferedTraffic'>Total offered
```

```
traffic (Erlangs,
bps)</a></td><td>#trafficMetrics_totalOfferedTraffic#</td></tr>
      <tr><td><a href='#metrics_totalCarriedTraffic'>Total carried
traffic (Erlangs,
bps)</a></td><td>#trafficMetrics_totalCarriedTraffic#</td></tr>
      <tr><td><a href='#metrics_averageOfferedTraffic'>Average offered
traffic per demand (Erlangs,
bps)</a></td><td>#trafficMetrics_averageOfferedTraffic#</td></tr>
      <tr><td><a href='#metrics_averageCarriedTraffic'>Average carried
traffic per demand (Erlangs,
bps)</a></td><td>#trafficMetrics_averageCarriedTraffic#</td></tr>
      <tr><td><a href='#metrics_blockedTraffic'>Blocked traffic
(%)</a></td><td>#trafficMetrics_blockedTraffic#</td></tr>
      <tr><td>Avg. number of
hops</td><td>#trafficMetrics_numHops#</td></tr>


<tr><td><a href='#trafficMetrics_isSymmetric'>is traffic Symmetric?</a>
      </td><td>#trafficMetrics_isSymmetric#</td></tr>
```

Tras la modificación del algoritmo, se vuelve a Net2Plan para mostrar las modificaciones repitiendo los pasos citados 1 y 2. El siguiente paso en el caso de estudio consiste en generar un informe sencillo desde cero, que no tenga plantilla HTML ni figuras asociadas para reforzar el aprendizaje. Este nuevo informe indica el coste total de la red, asignando un coste por enlace y por nodo, que son parámetros de entrada del algoritmo y modificables desde la interfaz de usuario en cada ejecución. El código es el siguiente:

```
package com.net2plan.examples.reports.networkDesign;


import com.net2plan.interfaces.networkDesign.IReport;
import com.net2plan.interfaces.networkDesign.NetPlan;
import com.net2plan.utils.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;


public class Report_Cost implements IReport
{
      @Override
      public String executeReport(NetPlan netPlan, Map<String, String>
reportParameters, Map<String, String> net2planParameters)
      {
            int N = netPlan.getNumberOfNodes();
            int E = netPlan.getNumberOfLinks();

            double costoneNode =
Double.parseDouble(reportParameters.get("NodeCost"));
```

```
            double costoneLink =
Double.parseDouble(reportParameters.get("LinkCost"));
            double linkcostperKm =
Double.parseDouble(reportParameters.get("LinkCostperKm"));

            double costNodes = N * costoneNode;
            double costLinks = E * costoneLink;

            double[] linkLengthInKmVector =
netPlan.getLinkLengthInKmVector();
            double totalCostperkm = 0;

            StringBuilder networkInfo = new StringBuilder();

            networkInfo.append("The total cost of nodes is: " +
costNodes);
            networkInfo.append("<br>The total cost of set links is: " +
costLinks);

            for (int linkId = 0; linkId < E; linkId++)
            {
                  totalCostperkm = totalCostperkm +
linkLengthInKmVector[linkId] * linkcostperKm;
                  networkInfo.append("<br>The cost of link " + linkId +
" per km installed is: " + linkLengthInKmVector[linkId] *
linkcostperKm);
            }
            networkInfo.append("<br>The total cost of links is: " +
totalCostperkm);

            return networkInfo.toString();
      }

      @Override
      public String getDescription()
      {
            return "This report shows basic information about the cost
of the network";
      }

      @Override
      public String getTitle()
      {
            return "Network Cost report";
      }

      @Override
      public List<Triple<String, String, String>> getParameters()
      {
```

```
            List<Triple<String, String, String>> reportParameters = new
ArrayList<Triple<String, String, String>>();
            reportParameters.add(Triple.of("LinkCost", "1", "Cost of set
a link"));
            reportParameters.add(Triple.of("NodeCost", "1", "Cost of set
a node"));
            reportParameters.add(Triple.of("LinkCostperKm", "1", "Cost
per km of a link"));

            return reportParameters;

      }

}
```

Se pueden observar los distintos métodos que implementa la interfaz `IReport`. El método *executeReport()* que es el encargado de ejecutar y obtener toda la información relativa al informe, es el método principal. El método *getDescription()* devuelve una definición del algoritmo en la interfaz gráfica. El método *getTitle()* devuelve un título para el informe en la interfaz gráfica. El método *getParameters()* define los distintos parámetros de entrada del algoritmo, tal y como se mostrarán al usuario en la interfaz gráfica, con un nombre, un valor por defecto y una descripción asociadas.

Se explica el contenido del código y, finalmente, se vuelve a Net2Plan para mostrar los resultados. Podemos observar en la siguiente imagen, los diferentes parámetros de entrada definidos, así como la información resultante tras la ejecución del informe:

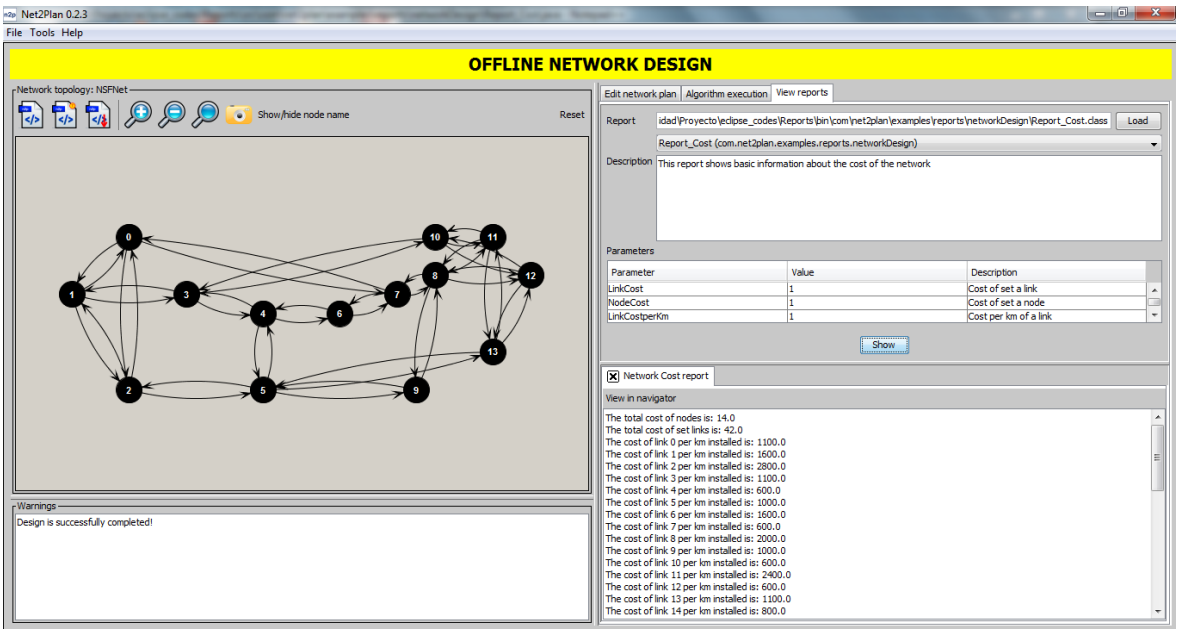


Figura 25: Generación del informe sobre costes del diseño de red

Tras la grabación del vídeo-tutorial, el siguiente paso es la integración del mismo en el canal de YouTube [8], para su difusión al conjunto de usuarios de la herramienta. En este caso, el título del vídeo es "*Getting started with Net2Plan: Report Generation tool"*. Puede ser visualizado en [8] o accediendo a la URL https://www.youtube.com/watch?v=ND8kOiiK1sY.

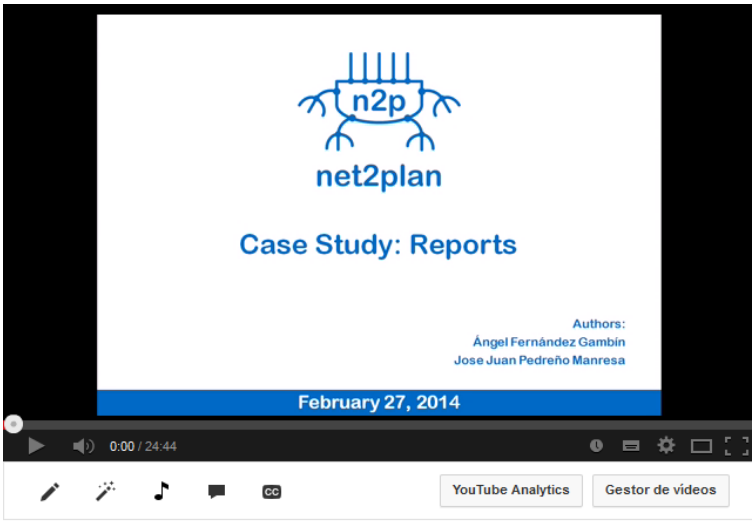


Figura 26: Captura del vídeo-tutorial *Reports*

3.3. SIMULADOR DE FALLOS

Como se comentó en los apartados introductorios, Net2Plan posee tres herramientas de simulación. Particularmente, el simulador de fallos ("*Resilience Simulation*") simula el funcionamiento de la red, donde los fallos en los enlaces y nodos aparecen aleatoriamente según la información sobre la disponibilidad de nodos y enlaces definida por el usuario. Además, permite probar algoritmos de aprovisionamiento que reaccionan a los eventos de fallo/reparación de un diseño inicial de red. Asimismo, el simulador recoge los resultados sobre la disponibilidad de la red y el servicio, y los muestra al usuario.

La simulación se basa en el concepto de grupo de riesgo compartido (SRG). Un SRG es un conjunto de enlaces y/o nodos que se agrupan de acuerdo a una causa potencial de fallo en la red. Por lo tanto, cuando se produce este fallo, todos los elementos del SRG fallan al mismo tiempo. Los usuarios pueden definir los SRG que precisen en la red, de acuerdo con la identificación de los recursos que están sujetos al fallo (nodos, enlaces, enlaces bidireccionales...). Para cada SRG definido, se proporciona el tiempo medio hasta fallo (MTTF) y el tiempo medio de reparación (MTTR). El generador de eventos es el encargado de generar eventos de fallo/reparación del SRG, mientras que el procesador de eventos recibe los eventos generados, uno por cada elemento de red que falla (por ejemplo, dado un SRG compuesto por un enlace y un nodo, el generador de eventos genera un evento fallo/reparación, mientras que el procesador de eventos divide y procesa ese evento, en dos eventos fallo/reparación distintos: uno asociado al nodo y otro al enlace) y toma las acciones necesarias para solventar el fallo: desviar el tráfico, restaurar la ruta primaria, etc.

Mientras Net2Plan ofrece algunos modos para definir fácilmente SRGs en un diseño de red para algunos casos típicos (un SRG por nodo, por enlace unidireccional, por conjunto de enlaces unidireccionales, conjunto de enlaces bidireccionales, etc.), los usuarios pueden definir sus propios modelos SRG arbitrariamente. Para ello, los usuarios deben utilizar la pestaña de grupo de riesgo compartido en la herramienta *Offline Network Design.*

En la figura 3 se hizo una representación general del ciclo de vida de los simuladores en Net2Plan. Una representación particular para este simulador sería la siguiente:

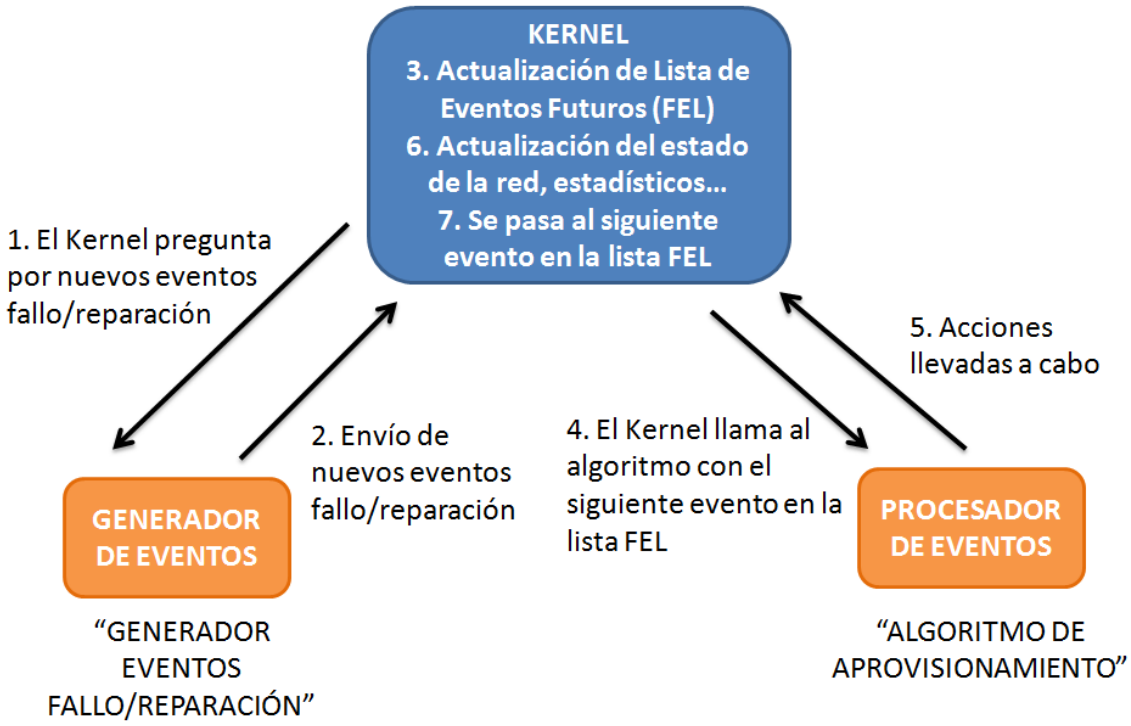


Figura 27: Ciclo de vida del simulador de fallos

3.3.1. OBJETIVOS DEL CASO DE ESTUDIO

La finalidad de este caso de estudio es introducir al usuario de Net2Plan los conceptos necesarios para desarrollar y evaluar algoritmos de aprovisionamiento que reaccionen ante eventos fallo/reparación en la red, así como algoritmos generadores de dichos eventos. Los conceptos relacionados con el uso de la herramienta *Resilience simulation* quedan fuera de este caso de estudio, y se considera que el usuario previamente ha leído la guía de usuario de la aplicación o visto otro caso de estudio relacionado con el uso de la herramienta.

3.3.2. ESTRUCTURA DEL VÍDEO-TUTORIAL

Al inicio del vídeo-tutorial se introduce el concepto de *simulador de fallos* en Net2Plan. Se explica el funcionamiento del simulador y se introduce el concepto de SRG, Tiempo medio hasta fallo, Tiempo medio de reparación, la función del generador de eventos fallo/reparación y la función del algoritmo de aprovisionamiento encargado de procesar los eventos generados.

Figura 28: Explicación del concepto de grupo de riesgo compartido (SRG)

Tras esta explicación, se muestra brevemente el proceso de cargar ambos algoritmos (generador de eventos y algoritmo de aprovisionamiento) en la interfaz gráfica de esta herramienta, y cómo arrancar la simulación sin entrar en más detalles, tales como parámetros de la simulación y visionado de estadísticas. Para este caso de estudio, se hace uso de los siguientes algoritmos integrados: (i) el generador de eventos fallo/reparación será `NRSim_EG_exponentialSRGFailureGenerator.java`, que genera fallos aleatorios en SRGs de acuerdo a un proceso aleatorio con una distribución exponencial de valores medios MTTF y MTTR; y (ii) el algoritmo de aprovisionamiento será `NRSim_AA_MPLS_fastReroute.java`, que implementa un mecanismo de restauración local.

Como se ha mencionado en otras ocasiones, en la web de Net2Plan se encuentra el repositorio completo de algoritmos incluidos en la aplicación.

A continuación, se pretende modificar los dos algoritmos cargados anteriormente en Net2Plan, con el fin de analizar su código y mostrar la manera de desarrollar nuevos algoritmos.

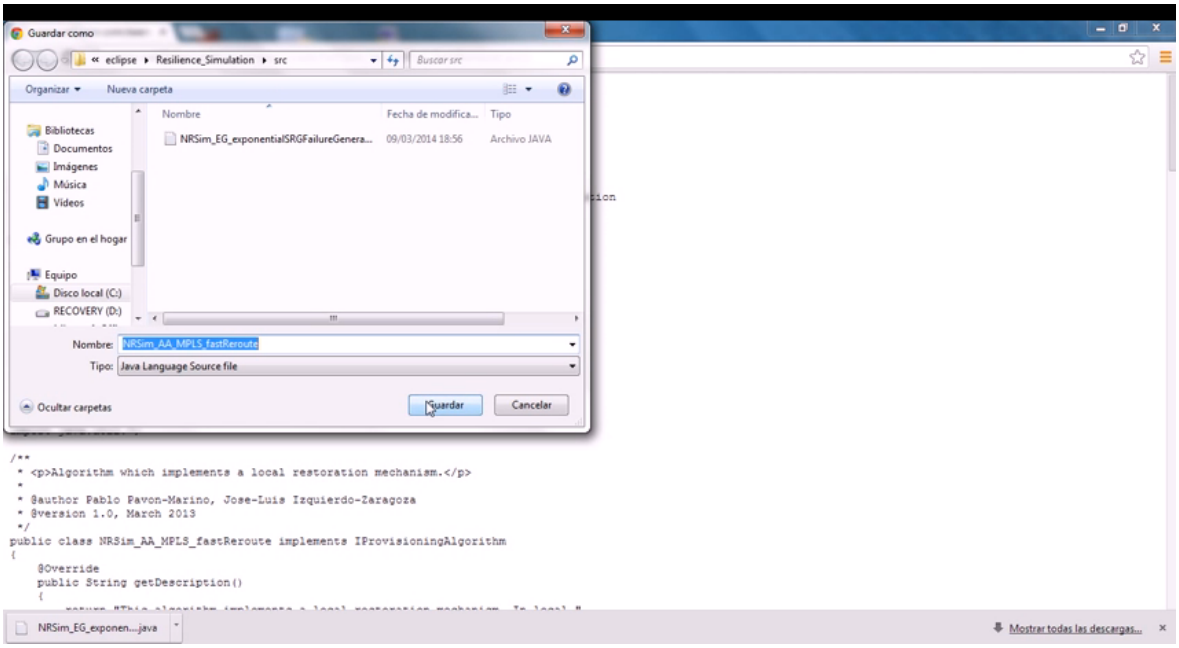


Figura 29: Descargando el código del algoritmo de la web de Net2Plan

Después de obtener el código de ambos algoritmos desde la web, se pasa a explicar el código del algoritmo de aprovisionamiento, detallando la interfaz que debe implementar este tipo de algoritmos en Net2Plan. Se trata de la interfaz `IProvisioningAlgorithm`, cuya información se detalla en el Javadoc. El algoritmo *Fast Reroute* implementa un mecanismo de restauración local. En la restauración local cada ruta se trata de restaurar cuando ocurre un fallo, en el nodo inmediatamente anterior al fallo. Este nodo, llamado Punto de Reparación Local (PLR) reenvía el tráfico usando la ruta de respaldo, y el nodo final donde la ruta de respaldo vuelve a unirse a la ruta principal se llama Punto de Unión (MP). Este mecanismo proporciona una recuperación más rápida debido a que la decisión de recuperación es estrictamente local. La siguiente figura representa este mecanismo:

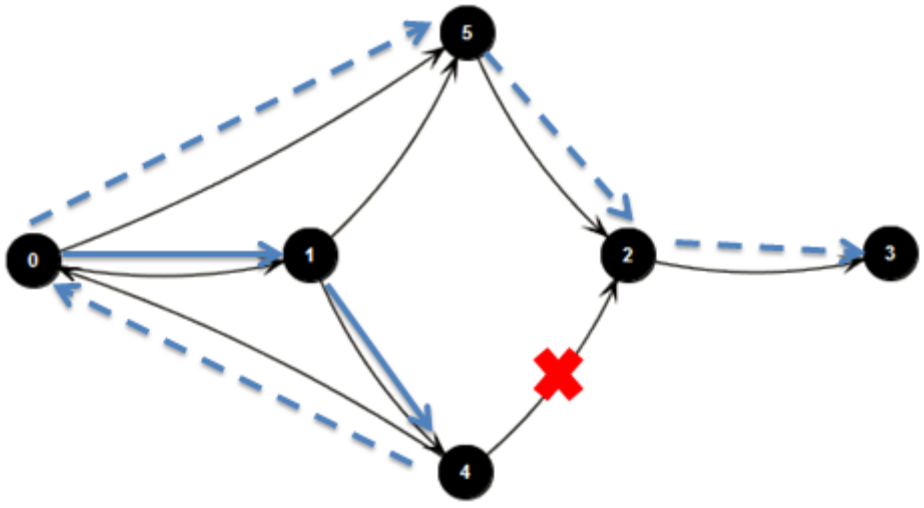


Figura 30: Mecanismo de *Fast Reroute*

En el vídeo se explica al usuario este mecanismo haciendo uso de un conjunto de transparencias con dibujos explicativos como el anterior. Tras la explicación teórica, se

detallan las líneas de código que implementan este mecanismo en el algoritmo:

```
int firstAvailableNodeUpstream =
netState.getFirstAvailableNodeUpstream(routeId, nodesDown, linksDown);

int firstAvailableNodeDownstream =
netState.getFirstAvailableNodeDownstream(routeId, nodesDown, linksDown);

if (firstAvailableNodeUpstream == -1 || firstAvailableNodeDownstream == -1)
continue;

int[] csp = GraphUtils.getCapacitatedShortestPath(linkTable,
firstAvailableNodeUpstream, firstAvailableNodeDownstream, costPerLink,
currentLinkAvailableCapacity, trafficVolume);
```

Una vez explicado el código de todo el algoritmo, el siguiente paso es modificarlo. Se ha hecho especial énfasis en el mecanismo de restauración local, ya que la modificación a este algoritmo consiste en cambiar ese mecanismo por el algoritmo *Path-Recovery*. Este algoritmo realiza la restauración de la ruta desde el nodo inicial de la misma hasta el nodo final, de manera que cuando se detecta un fallo, el tráfico se reenvía desde el nodo origen hasta el nodo destino, en contraposición al anterior mecanismo, donde el tráfico se reencamina desde el nodo local que detecta el fallo. La siguiente imagen muestra este otro tipo de restauración:

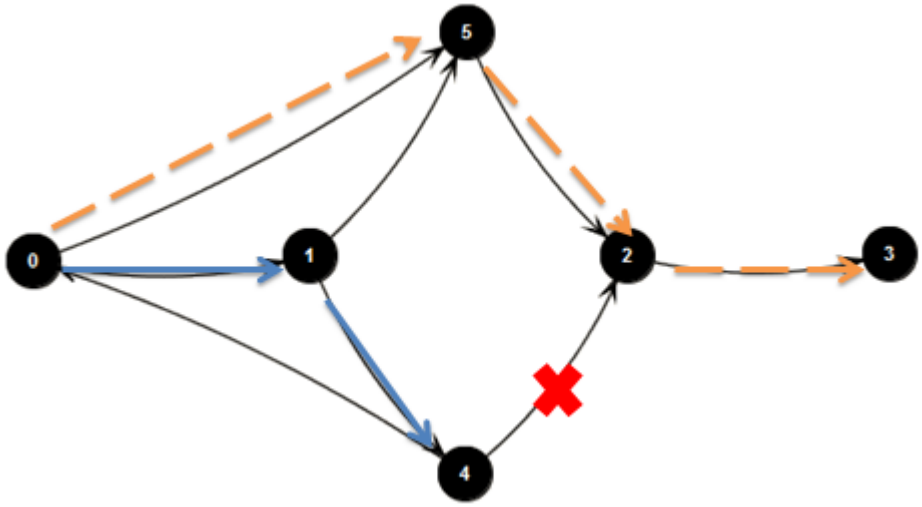


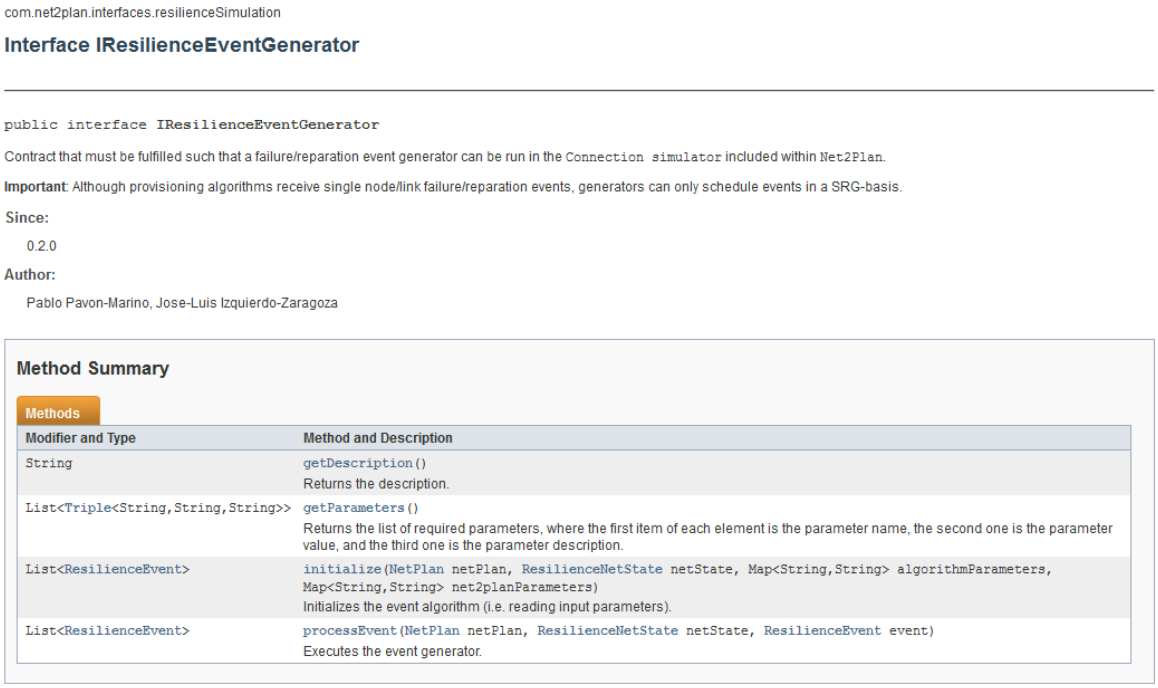
Figura 31: Mecanismo de *Path-Recovery*

Como se hizo anteriormente, tras la explicación teórica del nuevo mecanismo de restauración, se muestra en el vídeo el código del algoritmo modificado (`NRSim_AA_MPLS_fastReroute_modified.java`), y se pasa a su explicación. Las líneas de código modificadas en este caso son las siguientes:

```
int demandId = netState.getRouteDemand(routeId);
int ingressNode = netPlan.getDemandIngressNode(demandId);
int egressNode = netPlan.getDemandEgressNode(demandId);

int[] csp = GraphUtils.getCapacitatedShortestPath(linkTable, ingressNode,
egressNode, costPerLink, currentLinkAvailableCapacity, trafficVolume);
```

Una vez explicado y modificado el algoritmo de aprovisionamiento, se pasa al algoritmo generador de eventos fallo/reparación `NRSim_EG_exponentialSRGFailureGenerator`. De la misma forma anterior, se muestra la interfaz requerida para este algoritmo, y el conjunto de métodos que debe implementar. En este caso es la interfaz `IResilienceEventGenerator`. En la siguiente imagen se puede observar parte de la información que proporciona el Javadoc para esta interfaz:



com.net2plan.interfaces.resilienceSimulation

Interface IResilienceEventGenerator

`public interface IResilienceEventGenerator`

Contract that must be fulfilled such that a failure/reparation event generator can be run in the `Connection simulator` included within `Net2Plan`.

Important: Although provisioning algorithms receive single node/link failure/reparation events, generators can only schedule events in a SRG-basis.

Since:

0.2.0

Author:

Pablo Pavon-Marino, Jose-Luis Izquierdo-Zaragoza

Method Summary

Methods

Modifier and Type	Method and Description
`String`	`getDescription()` Returns the description.
`List<Triple<String,String,String>>`	`getParameters()` Returns the list of required parameters, where the first item of each element is the parameter name, the second one is the parameter value, and the third one is the parameter description.
`List<ResilienceEvent>`	`initialize(NetPlan netPlan, ResilienceNetState netState, Map<String,String> algorithmParameters, Map<String,String> net2planParameters)` Initializes the event algorithm (i.e. reading input parameters).
`List<ResilienceEvent>`	`processEvent(NetPlan netPlan, ResilienceNetState netState, ResilienceEvent event)` Executes the event generator.

Figura 32: Explorando el Javadoc de Net2Plan

Siguiendo con el generador de eventos fallo/reparación, este algoritmo podría ser modificado de manera que la generación de fallos aleatorios en la red siguiera una distribución diferente a la exponencial, tal como podría ser el modelo "bañera" pensado inicialmente como modificación para este caso. Sin embargo, dada la complejidad del modelo, lo cual supondría un esfuerzo de comprensión elevado para el usuario, se opta por no modificarlo. Por tanto, tras la explicación detallada de cada uno de los métodos del algoritmo, se vuelve a Net2Plan para probar las modificaciones del algoritmo de aprovisionamiento.

Una vez en Net2Plan, tras cargar los algoritmos modificados, se arranca la simulación. Tras unos segundos, se detiene la misma y en la pestaña *View current state* observamos el estado actual de la red, en función del estado de la simulación. Se observa para una ruta seleccionada en la que ha ocurrido un fallo, la aplicación del algoritmo *Path-Recovery* que

nos indica el correcto funcionamiento de la modificación.

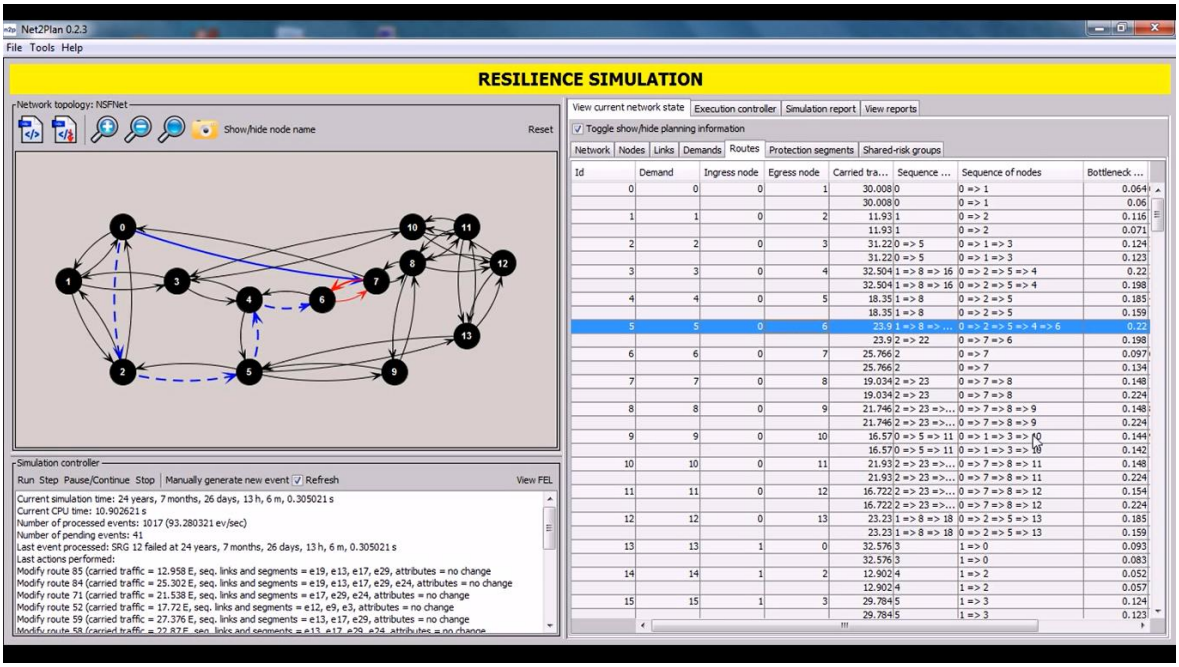


Figura 33: Observando el estado actual del diseño de red con la simulación pausada

Tras la grabación del vídeo-tutorial, el siguiente paso es la integración del mismo en el canal de YouTube [8], para su difusión al conjunto de usuarios de la herramienta. En este caso, el título del vídeo es "*Developing algorithms for Net2Plan: Network Recovery algorithms for Resilience simulation"*. Puede ser visualizado en [8] o accediendo a la URL https://www.youtube.com/watch?v=107S4twJxZA.

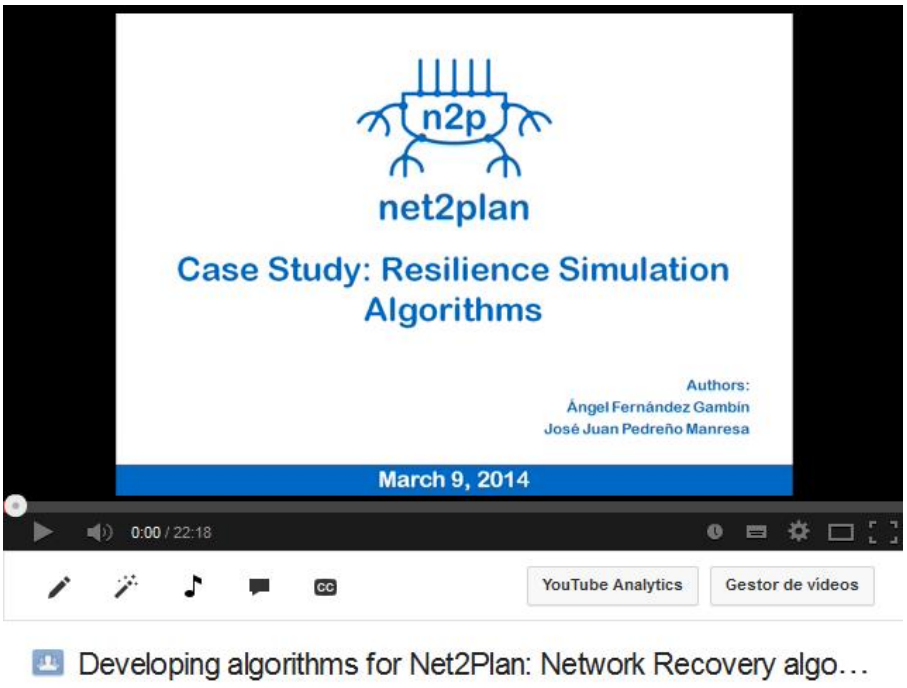


Figura 34: Captura del vídeo-tutorial sobre desarrollo de algoritmos para el simulador de fallos

3.4. SIMULADOR DE CONTROL DE ADMISIÓN

Esta herramienta de post-análisis simula el funcionamiento de la red, donde las demandas de tráfico son la fuente de llegadas aleatorias de peticiones de conexión. Cada petición de conexión tiene una duración y un volumen de tráfico asociados, que pueden ser escogidos al azar o de manera determinista en función de un patrón incluido o definido por el usuario. Generalmente, el tráfico ofrecido por cada una de las peticiones de conexión de una demanda sigue, en cierta medida, el valor de tráfico ofrecido para dicha demanda.

Además, el simulador permite comprobar el rendimiento de algoritmos de control de admisión (CAC, *Connection Admission Control*), ya sean los que incluye la herramienta o los desarrollados por el usuario, aceptando las peticiones (y asignándoles una o más ruta(s)), o rechazándolas. Asimismo, el simulador recoge algunos estadísticos como por ejemplo medidas de bloqueo.

El generador de eventos (llamado generador de eventos de conexión) es el encargado de generar las peticiones de conexión (instante de llegada, volumen de tráfico, duración de la conexión), mientras que el procesador de eventos (llamado algoritmo CAC) decide si se acepta o no (y si acepta todo el volumen de tráfico solicitado o una parte de él) y puede tomar algunas medidas adicionales: liberar las conexiones existentes, variar el volumen de tráfico ... Además, puede reaccionar a eventos de liberación de conexión entre otras acciones. El ciclo de vida específico de este simulador se muestra en el siguiente diagrama:

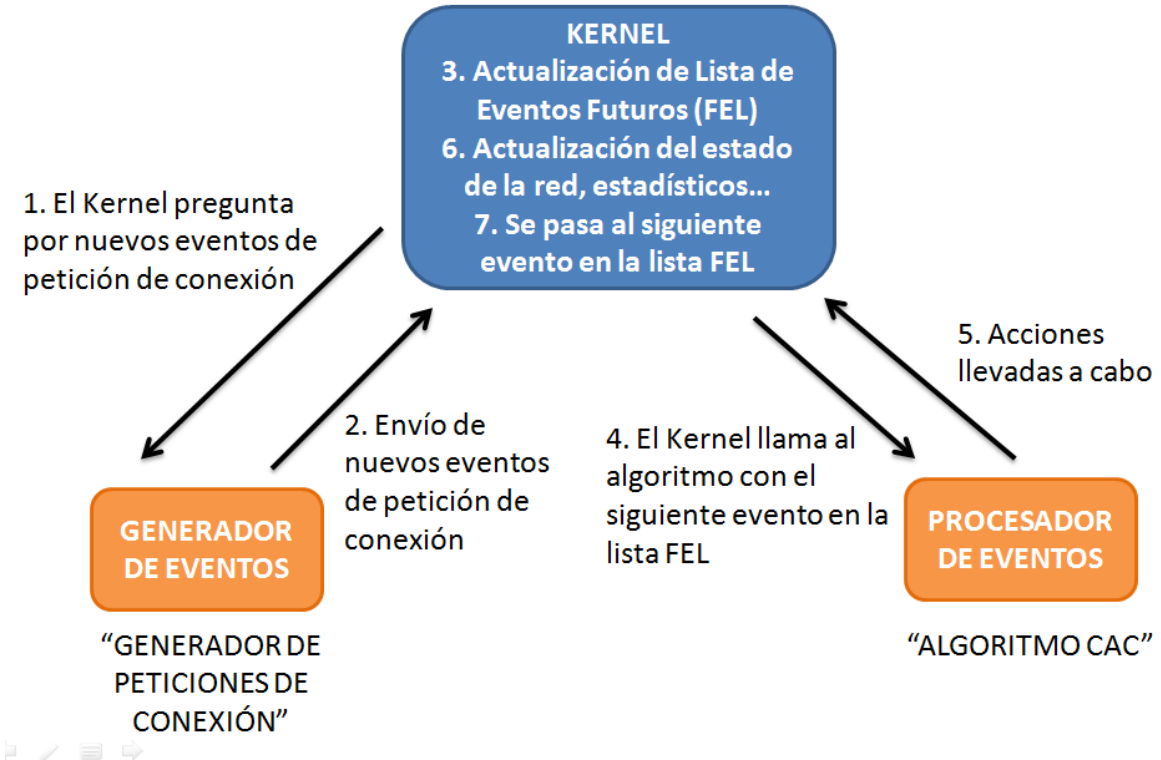


Figura 35: Ciclo de vida del simulador CAC

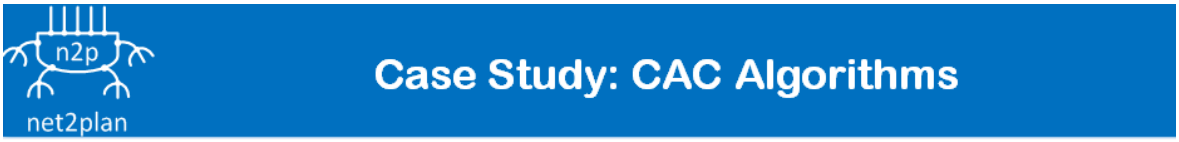
3.4.1. OBJETIVOS DEL CASO DE ESTUDIO

La finalidad de este caso de estudio es introducir al usuario de Net2Plan los conceptos

necesarios para desarrollar y evaluar algoritmos de control de admisión (CAC) que reaccionen ante eventos de petición de conexión en la red, así como algoritmos generadores de dichos eventos. Los conceptos relacionados con el uso de la herramienta *Connection-admission-control simulation* quedan fuera de este caso de estudio, y se considera que el usuario previamente ha leído la guía de usuario de la aplicación o visto otro caso de estudio relacionado con el uso de la herramienta.

3.4.2. ESTRUCTURA DEL VÍDEO-TUTORIAL

Para comenzar el vídeo se introduce el concepto de *simulador CAC* en Net2Plan. Se explica el funcionamiento del simulador, que tareas realiza y se abre la herramienta para este simulador. Tras esta explicación, se presenta la funcionalidad de cada uno de los algoritmos de este simulador (generador de eventos de peticiones de conexión y algoritmo de control de admisión).



➢The Connection event generator is in charge of generating connection requests (arrival time, requested traffic volume, holding time).

➢The event processor (CAC Algorithm) decides whether or not is accepted and can take some extra actions: release existing connections, vary traffic volume... Also, it can react to connection release events and can take other actions.

Figura 36: Explicación de los algoritmos empleados en el simulador CAC

Visualizando la interfaz gráfica de la herramienta, se muestra como cargar ambos algoritmos, se presentan sus descripciones y parámetros de entrada, y cómo arrancar la simulación sin entrar en más detalles. En este caso de estudio, se hace uso de los siguientes algoritmos integrados: (i) el generador de eventos de petición de conexión será `CACSim_EG_exponentialConnectionGenerator.java`, en el que las peticiones de conexión de las demandas de tráfico llegan según un proceso de Poisson y son independientes la una de la otra; y (ii) el algoritmo de control de admisión será `CACSim_AA_basicAlgorithm.java`, que trata de encaminar cada petición de conexión por el camino más corto. Dado que es un algoritmo muy simple, los enlaces pueden sufrir de congestión y convertirse en los cuellos de botella de la red.

De nuevo, se propone la modificación de los algoritmos que se usan en este simulador para

realizar una tarea concreta. Para ello, en primer lugar se puede utilizar como plantilla el código original de los algoritmos, que se puede descargar de la web de Net2Plan.

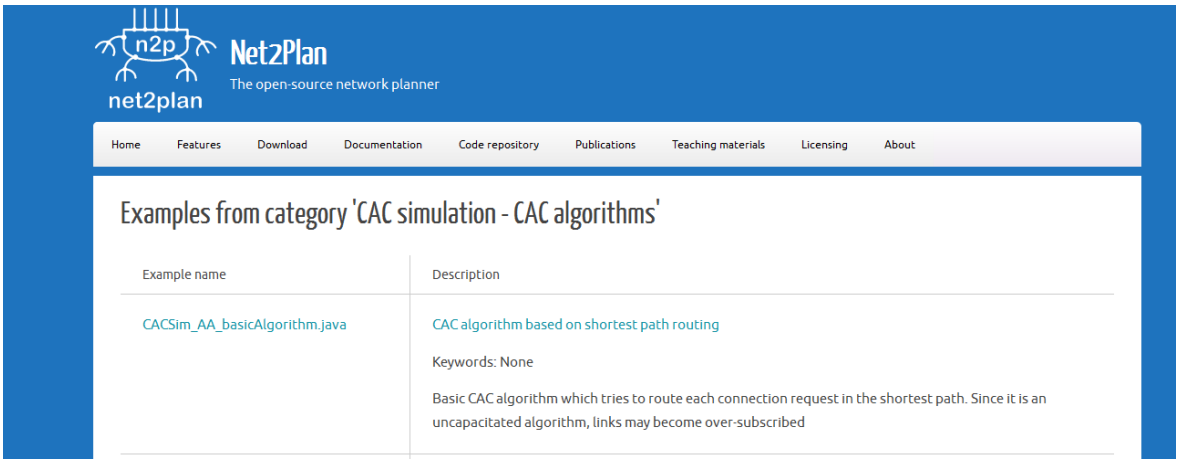


Figura 37: Repositorio de algoritmos en la web de Net2Plan

Tras la descarga desde el repositorio, se pasa a explicar el código del algoritmo de control de admisión citado antes, detallando la interfaz que debe implementar este tipo de algoritmos en Net2Plan. Se trata de la interfaz `ICACAlgorithm`, cuya información se detalla en el Javadoc de Net2Plan. Como se ha mencionado antes, este es un algoritmo básico de control de admisión que trata de encaminar cada conexión entrante por el camino más corto, en función del número de saltos ("hops") o de la distancia real ("km"), siendo esto elegido por el usuario en el momento de ejecución del algoritmo, es decir, es un parámetro de entrada del algoritmo.

Por tanto, cuando llega una petición de conexión, el algoritmo posee un método encargado de realizar esta tarea que invoca a la función *getShortestPath()*. Esta función, que puede ser consultada en el Javadoc, devuelve la secuencia de enlaces entre un nodo origen y un nodo destino, pasados como argumentos, determinando el camino más corto entre ellos teniendo en cuenta el coste por enlace. El código concreto que realiza esta función viene dado dentro del método *processEvent()* del algoritmo:

```
public List<CACAction> processEvent(NetPlan netPlan, ConnectionNetState
connectionNetState, CACEvent event)
    {
      final List<CACAction> actions = new LinkedList<CACAction>();

        switch(event.getEventType())
        {
            case CONNECTION_REQUEST:
                final int demandId = event.getRequestDemandId();
                final double trafficVolume =
event.getRequestTrafficVolumeInErlangs();

                final int ingressNodeId =
netPlan.getDemandIngressNode(demandId);
                final int egressNodeId = netPlan.getDemandEgressNode(demandId);
```

```
                final int[] spLinks = GraphUtils.getShortestPath(linkTable,
ingressNodeId, egressNodeId, costVector);
                if (spLinks.length == 0) actions.add(CACAction.blockRequest("No
path from ingress node to egress node"));
                else actions.add(CACAction.acceptRequest(spLinks,
trafficVolume, null));

                break;

            case CONNECTION_RELEASE:
                break;
        }

        return actions;
    }
```

Una vez explicado el código original, se introduce la modificación, se explica al usuario y se muestra dicha código. En este caso, la modificación consiste en variar el método mostrado antes de manera que ahora el algoritmo encamina cada conexión entrante por el camino más corto pero teniendo en cuenta la capacidad de los enlaces, esto quiere decir, que un enlace no puede cursar más tráfico del de su capacidad, y esto puede provocar que determinadas rutas que antes se establecían ahora no se puedan establecer por el mismo camino. Incluye un aspecto más restrictivo que se acerca más a la realidad de una red.

Por esto, en el nuevo código se emplea la función *getCapacitatedShortestPath()*. Dado un nodo de origen y un nodo de destino, un vector de costes de los enlaces, la capacidad de los enlaces para el estado actual de la red y un volumen de tráfico, esta función obtiene la ruta más corta entre estos dos nodos cumpliendo con los requisitos de capacidad para cada enlace de la ruta. Por tanto, la modificación requiere simplemente la variación de unas pocas líneas dentro del método *processEvent()* del algoritmo:

```
public List<CACAction> processEvent(NetPlan netPlan, ConnectionNetState
connectionNetState, CACEvent event)
    {
      List<CACAction> actions = new LinkedList<CACAction>();

        switch(event.getEventType())
        {
            case CONNECTION_REQUEST:
                int demandId = event.getRequestDemandId();
                double trafficVolume =
event.getRequestTrafficVolumeInErlangs();

                int ingressNodeId = netPlan.getDemandIngressNode(demandId);
                int egressNodeId = netPlan.getDemandEgressNode(demandId);
```

```
                double[] capacityVector =
connectionNetState.getLinkCurrentSpareCapacityInErlangsVector();

                int[] spLinks =
GraphUtils.getCapacitatedShortestPath(linkTable, ingressNodeId, egressNodeId,
costVector, capacityVector, trafficVolume);
                if (spLinks.length == 0) actions.add(CACAction.blockRequest("No
path from ingress node to egress node"));
                else actions.add(CACAction.acceptRequest(spLinks,
trafficVolume, null));

                break;

            case CONNECTION_RELEASE:
                break;
        }

        return actions;
    }
```

Una vez más, el siguiente paso consiste en la explicación del algoritmo generador de eventos, en este caso, de peticiones de conexión. Para comenzar, se muestra la interfaz que deben implementar estos algoritmos, y el conjunto de métodos asociados. En este caso es la interfaz `IConnectionEventGenerator`. Este algoritmo genera eventos de petición de conexión según un proceso de Poisson que depende de una semilla, siendo esta un parámetro de entrada del algoritmo que puede ser definida por el usuario en cada ejecución, pudiendo ser también de carácter aleatorio. Lo que realmente sigue un proceso de Poisson es la duración de una conexión, y el tiempo entre llegadas, que también dependen de unos parámetros de entrada modificables por el usuario. Se explica el código haciendo hincapié en estos conceptos de aleatoriedad.

El código principal que ejecuta estas funciones viene recogido en el método *initialize()* del algoritmo:

```
public List<CACEvent> initialize(NetPlan netPlan, ConnectionNetState netState,
Map<String, String> algorithmParameters, Map<String, String>
net2planParameters)
    {
        int D = netPlan.getNumberOfDemands();
        if (D == 0) throw new Net2PlanException("No demands are defined");

      long seed = Long.parseLong(algorithmParameters.get("randomSeed"));
      if (seed == -1) seed = RandomUtils.random(0, Long.MAX_VALUE - 1);
        Random seedGenerator = new Random(seed);

        int defaultConnectionSize =
Integer.parseInt(algorithmParameters.get("defaultConnectionSize"));
```

```
        if (defaultConnectionSize <= 0) throw new
Net2PlanException("'defaultConnectionSize' must be greater than zero");

        double defaultHoldingTime =
Double.parseDouble(algorithmParameters.get("defaultHoldingTime"));
        if (defaultHoldingTime <= 0) throw new
Net2PlanException("'defaultHoldingTime' must be greater than zero");

        iat_d = new AbstractDoubleDistribution[D];
        ht_d = new AbstractDoubleDistribution[D];
        s_d = new double[D];

        List<CACEvent> events = new LinkedList<CACEvent>();

        for(int demandId = 0; demandId < D; demandId++)
        {
          double h_d = netPlan.getDemandOfferedTrafficInErlangs(demandId);

            String aux_str_s_d = netPlan.getDemandAttribute(demandId,
"connectionSize");
            double aux_s_d;
            try { aux_s_d = Double.parseDouble(aux_str_s_d); }
            catch(Exception ex) { aux_s_d = defaultConnectionSize; }
            if (aux_s_d <= 0) throw new Net2PlanException("'connectionSize' for
demand " + demandId + " is lower or equal than zero");

            String aux_str_holdTime = netPlan.getDemandAttribute(demandId,
"holdingTime");
            double aux_holdTime;
            try { aux_holdTime = Double.parseDouble(aux_str_holdTime); }
            catch(Exception ex) { aux_holdTime = defaultHoldingTime; }
            if (aux_holdTime <= 0) throw new Net2PlanException("'holdingTime'
for demand " + demandId + " is lower or equal than zero");


            s_d[demandId] = aux_s_d;
            double holdingTime = aux_holdTime;
            double interArrivalTime = s_d[demandId] * holdingTime / h_d;

            iat_d[demandId] = new Exponential(1 / interArrivalTime, new
MersenneTwister64(seedGenerator.nextInt()));
            ht_d[demandId] = new Exponential(1 / holdingTime, new
MersenneTwister64(seedGenerator.nextInt()));

            events.add(scheduleNewConnectionArrival(demandId, 0));
        }

        return events;
```

```
    }
```

El siguiente paso consiste en modificar este algoritmo. Como se ha dicho, el algoritmo genera eventos de petición de conexión en función de una distribución poissoniana; la modificación consiste en variar esta distribución. Ahora, el tiempo entre llegadas vendrá determinado por una distribución Gaussiana, y la duración para cada conexión será un valor determinista fijado por el usuario en el momento de ejecutar el algoritmo, en el parámetro de entrada *defaultHoldingTime*.

A continuación se muestra el nuevo código, las modificaciones en este caso están presentes principalmente en dos métodos, el método *initialize()* y *scheduleNewConnectionArrival()*. Además para hacer efectivos los cambios se han introducido dos nuevas variables (*cv*, es decir, el coeficiente de variación, y *truncationParameter)*. Estas dos variables son parámetros de entrada del algoritmo y por tanto, son modificables por el usuario. El coeficiente de variación (*cv*) es la relación entre la media y la variabilidad de una variable, es decir, su desviación estándar. Y el parámetro de truncado es la desviación máxima del valor medio, medida en unidades de número de desviaciones estándar desde la media.

```
public List<CACEvent> initialize(NetPlan netPlan, ConnectionNetState netState,
Map<String, String> algorithmParameters, Map<String, String>
net2planParameters)
    {
        int D = netPlan.getNumberOfDemands();
        if (D == 0) throw new Net2PlanException("No demands are defined");

      long seed = Long.parseLong(algorithmParameters.get("randomSeed"));
      if (seed == -1) seed = RandomUtils.random(0, Long.MAX_VALUE - 1);
        Random seedGenerator = new Random(seed);

        int defaultConnectionSize =
Integer.parseInt(algorithmParameters.get("defaultConnectionSize"));
        if (defaultConnectionSize <= 0) throw new
Net2PlanException("'defaultConnectionSize' must be greater than zero");

        double defaultHoldingTime =
Double.parseDouble(algorithmParameters.get("defaultHoldingTime"));
        if (defaultHoldingTime <= 0) throw new
Net2PlanException("'defaultHoldingTime' must be greater than zero");

      cv = Double.parseDouble(algorithmParameters.get("cv"));
      if (cv < 0) throw new Net2PlanException("'cv' must be greater or equal
than zero");

      truncationParameter =
Double.parseDouble(algorithmParameters.get("truncationParameter"));
      if (truncationParameter < 0) throw new
Net2PlanException("'truncationParameter' must be greater or equal than zero");
```

```
        avg_iat_d = new double[D];
        sigma_d = new double[D];
        iat_d = new AbstractDoubleDistribution[D];
        ht_d = new double[D];
        s_d = new double[D];

        List<CACEvent> events = new LinkedList<CACEvent>();

        for(int demandId = 0; demandId < D; demandId++)
        {
          double h_d = netPlan.getDemandOfferedTrafficInErlangs(demandId);

            String aux_str_s_d = netPlan.getDemandAttribute(demandId,
"connectionSize");
            double aux_s_d;
            try { aux_s_d = Double.parseDouble(aux_str_s_d); }
            catch(Exception ex) { aux_s_d = defaultConnectionSize; }
            if (aux_s_d <= 0) throw new Net2PlanException("'connectionSize' for
demand " + demandId + " is lower or equal than zero");

            String aux_str_holdTime = netPlan.getDemandAttribute(demandId,
"holdingTime");
            double aux_holdTime;
            try { aux_holdTime = Double.parseDouble(aux_str_holdTime); }
            catch(Exception ex) { aux_holdTime = defaultHoldingTime; }
            if (aux_holdTime <= 0) throw new Net2PlanException("'holdingTime'
for demand " + demandId + " is lower or equal than zero");

            s_d[demandId] = aux_s_d;
            ht_d[demandId] = aux_holdTime;

            avg_iat_d[demandId] = s_d[demandId] * ht_d[demandId] / h_d;

            sigma_d[demandId] = cv * avg_iat_d[demandId];

            iat_d[demandId] = new Normal(0, sigma_d[demandId], new
MersenneTwister64(seedGenerator.nextInt()));

            events.add(scheduleNewConnectionArrival(demandId, 0));
        }
        return events;
    }
```

Como se puede observar, se aprecia el cambio de función, en vez de usar una distribución exponencial, empleamos una normal de media cero y desviación estándar determinada en función de la demanda (`sigma_d[demandId]`). Además, la duración de cada conexión viene fijada de manera determinista, en función del parámetro de entrada comentado

anteriormente (`ht_d[demandId] = aux_holdTime`).

```
private CACEvent scheduleNewConnectionArrival(int demandId, double simTime)
    {
        double variationFromMeanValue = iat_d[demandId].nextDouble();
        if (variationFromMeanValue < -sigma_d[demandId] * truncationParameter)
variationFromMeanValue = -sigma_d[demandId] * truncationParameter;
        if (variationFromMeanValue > sigma_d[demandId] * truncationParameter)
variationFromMeanValue = sigma_d[demandId] * truncationParameter;

        double timeToNextArrival = avg_iat_d[demandId] +
variationFromMeanValue;
        if (timeToNextArrival < 0) timeToNextArrival = 0;

        double nextArrivalTime = simTime + timeToNextArrival;
        double duration = ht_d[demandId];

        double trafficVolumeInErlangs = s_d[demandId];
        Map<String, String> attributes = new HashMap<String, String>();

        return new CACEvent(nextArrivalTime, demandId, duration,
trafficVolumeInErlangs, attributes);
    }
```

En el código superior se puede apreciar en las primeras líneas como se ejecuta el truncado desde la media, y posteriormente, se fija el tiempo hasta la próxima llegada, la duración y el volumen de la siguiente petición de conexión.

Tras la explicación, se vuelve a Net2Plan para probar los algoritmos y sus actualizaciones.

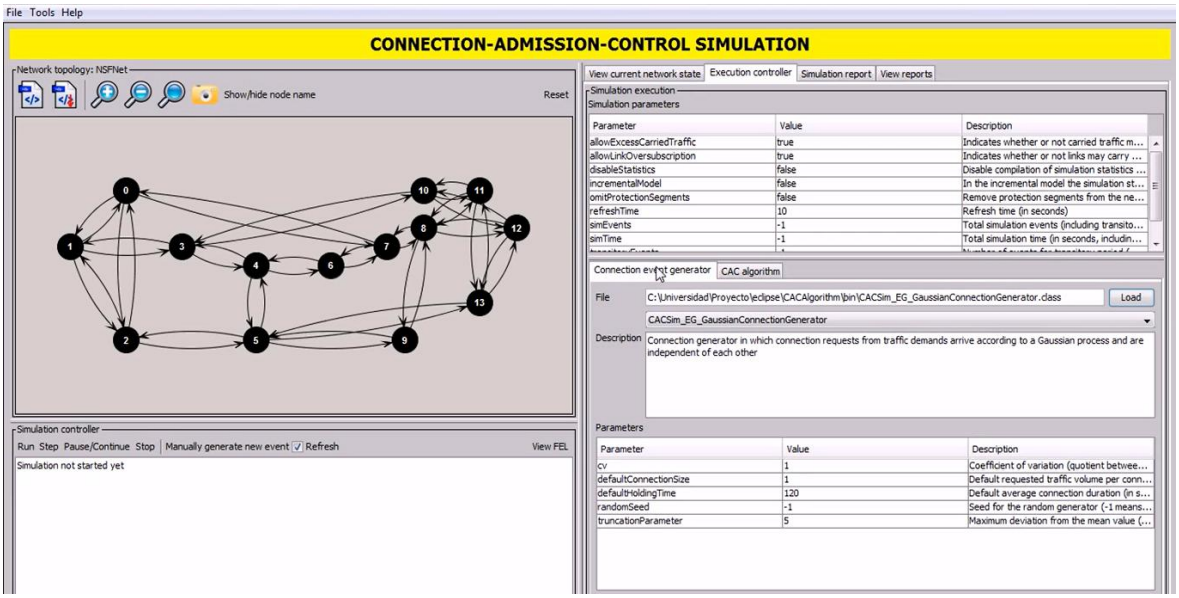


Figura 38: Probando los algoritmos modificados en la interfaz CAC

Tras la grabación del vídeo-tutorial, el siguiente paso es la integración del mismo en el canal de YouTube [8], para su difusión al conjunto de usuarios de la herramienta. En este caso, el título del vídeo es "*Developing algorithms for Net2Plan: CAC (Connection-Admission and Control) algorithms"*. Puede ser visualizado en [8] o accediendo a la URL https://www.youtube.com/watch?v=wwG0hmUSyNg.

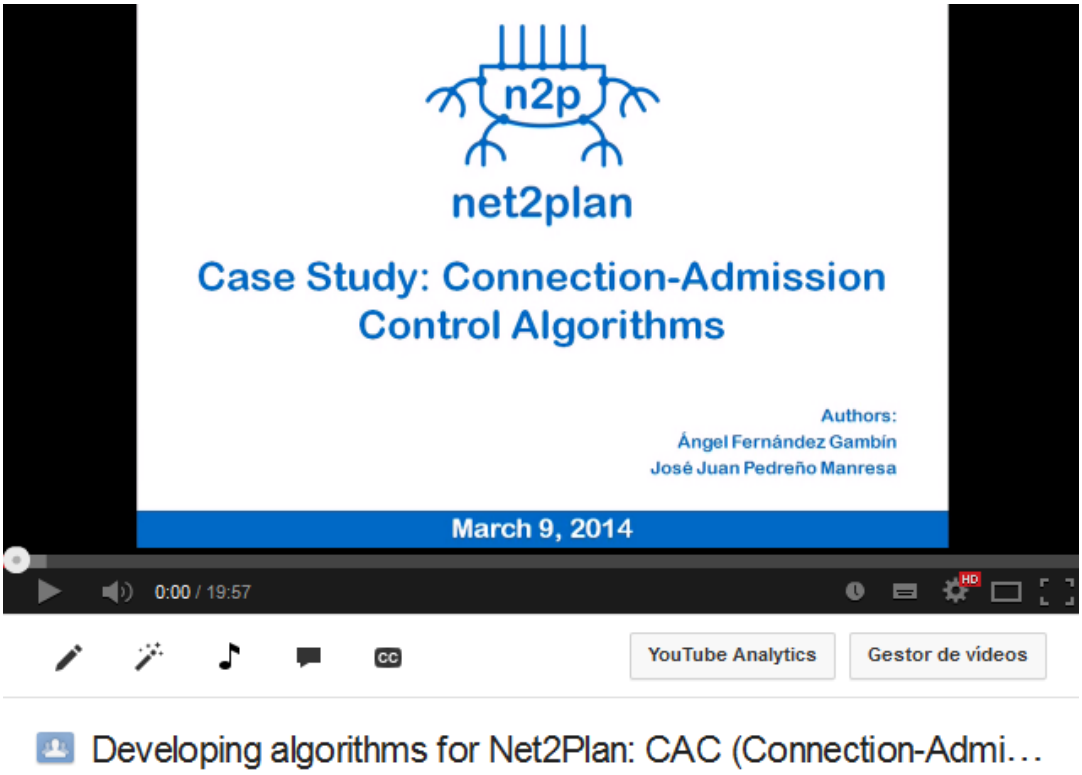


Figura 39: Captura del vídeo-tutorial sobre desarrollo de algoritmos *CAC*

3.5. SIMULADOR DE TRÁFICO VARIABLE

Esta herramienta de simulación post-análisis emula el funcionamiento de la red, donde los volúmenes de las demandas de tráfico varían con el tiempo de acuerdo a un patrón definido por el usuario. Está dirigida a evaluar las actuaciones y el rendimiento de sistemas que reaccionan a variaciones de tráfico (por ejemplo, sistemas de re-encaminamiento de tráfico, sistemas de aprovisionamiento de capacidad bajo demanda, etc.) Aquí, las estadísticas recopiladas están referidas al estado medio de la red durante la simulación (capacidad de enlace, utilización, grados de protección...).

El generador de eventos (llamado generador de cambios de tráfico) se encarga de modificar los volúmenes de tráfico para las demandas a intervalos periódicos, mientras que el procesador de eventos (llamado algoritmo de asignación) puede tomar acciones que reaccionan a esos cambios: añadir/eliminar enlaces, desviar demandas de tráfico , y así sucesivamente. Los generadores de eventos, tal y como ocurre en el resto de simuladores de Net2Plan, son clases Java incorporadas o definidas por el usuario que para este simulador en concreto implementan la interfaz `ITrafficGenerator`. Los algoritmos de asignación, por su parte, implementan la interfaz `ITrafficAllocationAlgorithm`.

El ciclo de vida de este simulador se diferencia con el resto de simuladores. En este simulador, no hay lista de eventos futuros (FEL), por lo que en cada período de tiempo (definido en el parámetro de simulación *timeGranularityInSeconds*) el simulador genera un evento y este evento es procesado. Este diagrama muestra el proceso completo:

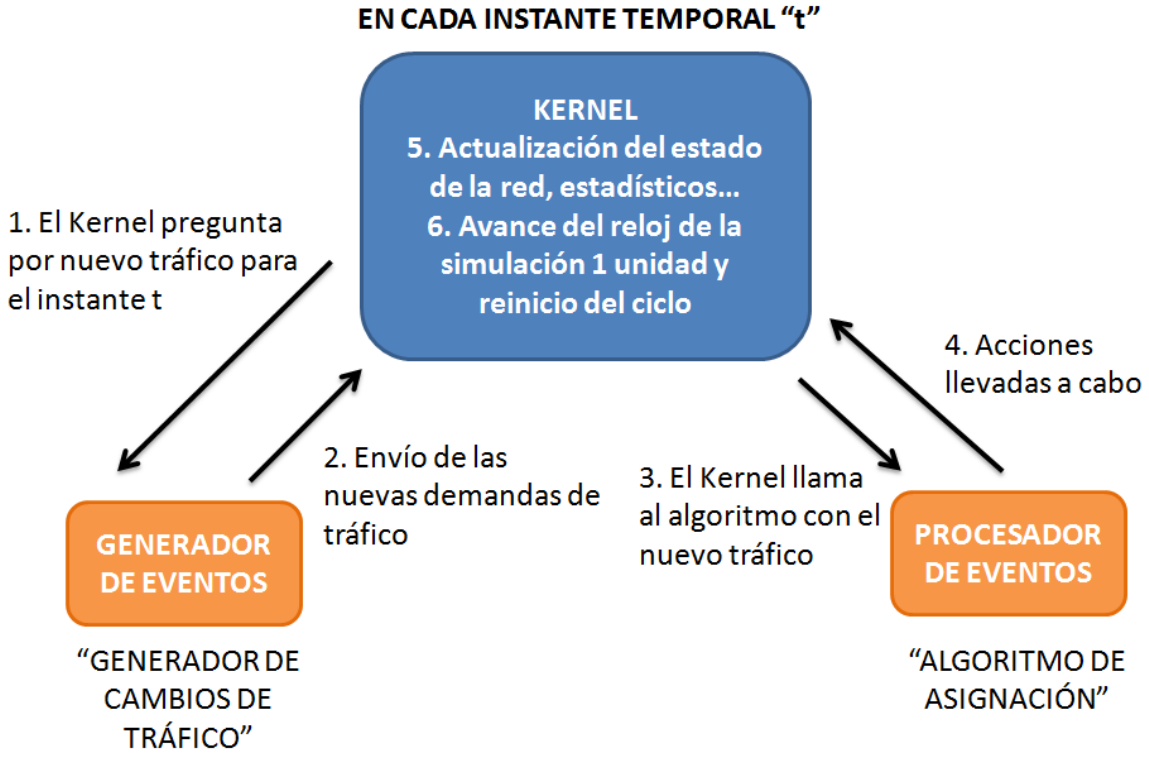


Figura 40: Ciclo de vida del simulador de tráfico variable

3.5.1. INTERFAZ GRÁFICA

Como el resto de herramientas en Net2Plan, exceptuando la herramienta de diseño de matrices de tráfico, este simulador tiene la interfaz gráfica dividida en las siguientes partes: Por un lado, los datos de entrada tales como parámetros de simulación o algoritmos concretos, el panel de ejecución de algoritmos, así como el panel de presentación de datos relacionados con el diseño de red, informes y estadísticas se encuentran en la zona derecha.

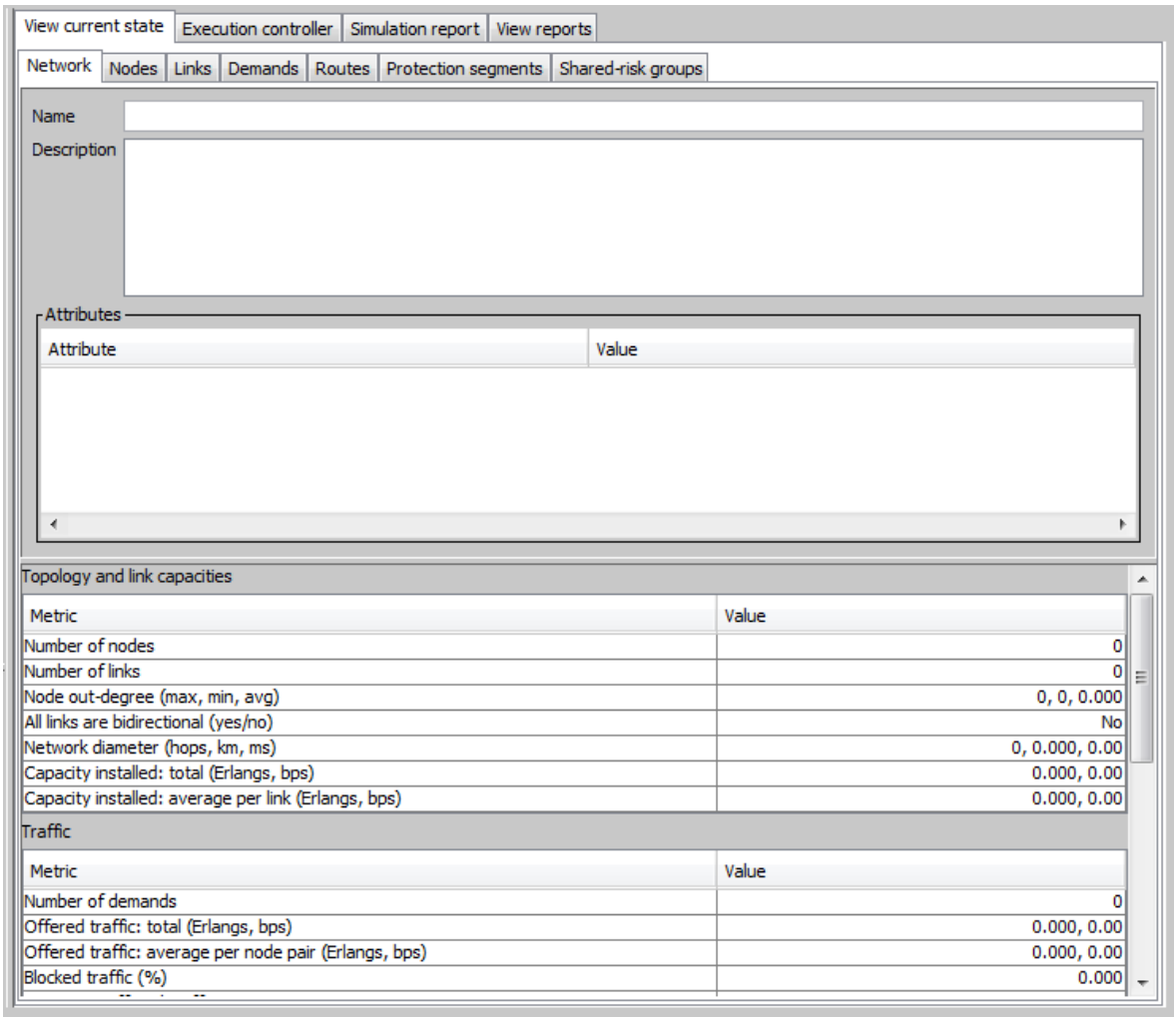


Figura 41: Ventana del simulador de tráfico variable (sección derecha)

Por otra parte, el área de dibujado donde se plasma la topología con la que se está trabajando se encuentra en el área superior izquierda, y el área de control y visionado de información de la simulación se encuentra en la zona inferior izquierda.

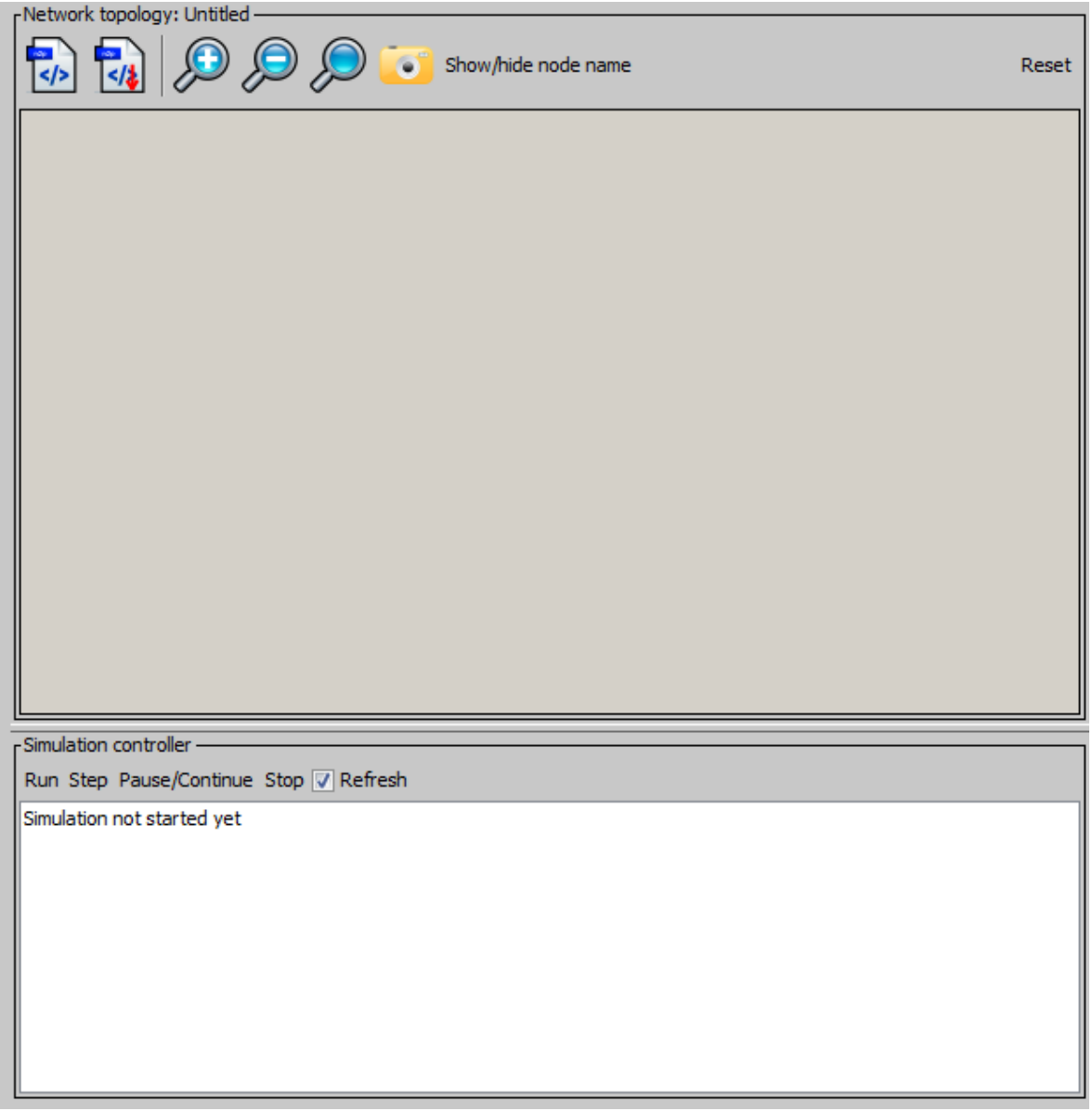


Figura 42: Ventana del simulador de tráfico variable (sección izquierda)

3.5.1.1. Panel "Network Topology"

El panel de topología de red se utiliza para observar gráficamente (pero no editar, ya que la edición solo es posible en la herramienta *Offline Network Design*) los diseños de redes en el inicio de la simulación. A medida que la simulación avanza este diseño puede verse modificado por la aparición o eliminación de algún enlace debido a las acciones de los algoritmos de asignación. Además, los usuarios pueden realizar las mismas acciones que en otros simuladores, y los botones del panel asociados son los mismos (cargar y guardar la topología actual, zoom-in, zoom-out, mostrar/ocultar los nombres de los nodos, y reset de la simulación manteniendo el diseño de red previamente cargado).

3.5.1.2. Panel "Simulation Controller"

Este es el panel encargado de controlar la ejecución de la simulación por parte del usuario. Los botones asociados a este panel son los siguientes:

- ***Run***: Arranca la simulación.
- ***Pause-Continue***: Pausa y reanuda la simulación, permitiendo al usuario observar el estado actual del diseño de red así como las estadísticas generadas.
- ***Stop***: Detiene la simulación por completo, no puede ser reanudada.
- ***Step*****:** Ejecuta la simulación paso a paso, realmente evento a evento.
- ***Refresh***: Este botón permite aplicar o no el parámetro de entrada de la simulación *refreshTime*, que indica cada cuantos segundos el panel de control de la simulación se actualiza.

Como se comentó al principio, este simulador no tiene FEL (Future Event List – Lista de Eventos Futuros) ni el botón que permite generar nuevos eventos de manera manual, ya que este simulador procesa eventos uno a uno.

A modo resumen este panel muestra la siguiente información: el estado de la simulación actual y la hora actual de la CPU, el número de eventos procesados y pendientes, el último evento procesado y, por último, las últimas acciones realizadas (volumen de tráfico para cada ruta modificada y nuevo valor del tráfico).

3.5.1.3. Pestaña "Execution Controller"

Esta pestaña permite introducir los parámetros de entrada necesarios para ejecutar una simulación, como son los parámetros de la simulación y los algoritmos necesarios para este simulador. Entre los parámetros de la simulación, destacan un conjunto que son comunes a todos los simuladores:

- "***allowExcessCarriedTraffic***": Este parámetro indica si el tráfico cursado puede ser mayor que el ofrecido por alguna demanda. Los valores permitidos son "true" o "false". Si es "false", el núcleo controla en todo momento si una demanda cursa más tráfico que su volumen de tráfico ofrecido, como se indica en el diseño original, y lanza una excepción (deteniendo la simulación) si eso sucede.
- "***allowLinkOverSubscription***": Este parámetro indica si un enlace puede cursar más tráfico que su capacidad o no. Los valores permitidos son "true" o "false". Si "false", el núcleo controla en cada momento si un enlace cursa más tráfico que su capacidad, como se indica en el diseño original, y lanza una excepción (deteniendo la simulación) si eso sucede.
- "***disableStatistics***": En algunas ocasiones, los usuarios pueden estar interesados en la recolección solo de sus propias estadísticas, y preferirían evitar la sobrecarga que requiere la recopilación de estadísticas por el núcleo de la aplicación. Esto se puede hacer con este parámetro. En este caso, las estadísticas del núcleo se presentan con valores de cero, mientras que las estadísticas específicas del algoritmo se muestran con los valores recogidos. Los valores permitidos son "true" o "false".

- "***omitProtectionSegments***": Los segmentos de protección reservan una cierta cantidad de ancho de banda para proporcionar caminos (parciales en algunos casos) de respaldo para rutas primarias. Sin embargo, en algunas situaciones, los usuarios pueden estar interesados en la ejecución de sus simulaciones asumiendo que no hay segmentos de protección definidos. Si este parámetro se establece a "true", entonces los segmentos de protección se eliminan del diseño de red justo antes de iniciar la simulación, y su ancho de banda en los enlaces está disponibles para cursar el tráfico común. Los valores permitidos son "true" o "false".
- "***refreshTime***": Si la opción "*Refresh*" se ha activado en el panel de control de simulación, la información sobre la simulación actual, (número de eventos procesados, simulación y tiempo de CPU...) se mostrará allí. Esta información se actualiza cada un cierto número de segundos dado por este parámetro. Los valores permitidos son números enteros mayores que 0.
- "***simEvents***": Número total de eventos (incluyendo eventos transitorios) a simular. Cada fallo y cada reparación en un SRG (independientemente del número de enlaces o nodos asociados) cuenta como un solo evento. Si también se especifica el parámetro "*simTime*", la simulación finalizará automáticamente cuando se cumpla la primera condición de parada. Los valores permitidos son números enteros mayores que cero, o -1 que indica sin límite (es decir, la simulación debe ser detenida manualmente).
- "***simTime***": Tiempo de simulación total (incluyendo los eventos transitorios) expresado en segundos. Si también se especifica el parámetro "*simEvents*", la simulación finalizará automáticamente cuando se cumpla la primera condición de parada. Los valores permitidos son números enteros mayores que cero, o -1 que indica sin límite (es decir, la simulación debe ser detenida manualmente).
- "***transitoryEvents***": Número de eventos para el período transitorio. Cada fallo y cada reparación en un SRG (independientemente del número de enlaces o nodos asociados) cuenta como un solo evento. Si también se especifica el parámetro "*transitoryTime*", el período transitorio finalizará automáticamente cuando se cumpla la primera condición de parada. Los valores permitidos son números enteros mayores que cero, o -1 que indica sin límite (es decir, la simulación debe ser detenida manualmente).
- "***transitoryTime***": Tiempo transitorio expresado en segundos. Si también se especifica "*transitoryEvents*", el período transitorio finalizará cuando se cumpla la condición. Los valores permitidos son números enteros mayores que 0, o -1 para ningún período transitorio.

Y otros parámetros específicos para este simulador:

- "***startTime***": Este parámetro establece el tiempo de inicio de la simulación, en formato horas y minutos.
- "***startWeekDay***": Este parámetro establece el día de inicio de la simulación, estableciendo un día de la semana.
- "***timeGranularityInSeconds***": Este parámetro define el tiempo fijo entre cambios de tráfico, es decir, entre los eventos generados.

Además de los parámetros de la simulación, los paneles de selección de algoritmos, tanto para el generador de cambios de tráfico como el de asignación, permiten ver la descripción y modificar los parámetros de entrada de estos.

3.5.1.4. Pestaña "View Current Network State"

En esta pestaña, los usuarios pueden ver la misma información relacionada con el estado de la red (en el momento actual de la simulación) que se muestra en la herramienta *Offline network design*. Podemos observar información acerca de la red, nodos, enlaces, demandas, rutas, segmentos de protección o grupos de riesgo compartido.

3.5.1.5. Pestaña "Simulation Report"

En este panel, los usuarios pueden obtener un resumen de la evolución del diseño de red, así como las métricas recogidas por el algoritmo de aprovisionamiento (si las hay, utilizando el método de *finish()*).

Los usuarios pueden ver el informe en cualquier momento (al hacer clic en el botón "*Update"*), mientras que la simulación está en pausa o se detiene. También es posible su visualización en un navegador haciendo clic en el botón "*View in navigator"*.

Como se comentó en el apartado de parámetros de simulación, en algunas ocasiones, los usuarios pueden estar interesados en la recolección únicamente de sus propias estadísticas, queriendo eliminar la (gran) sobrecarga, en tiempo y memoria, que requiere la recopilación de estadísticas por parte del núcleo. Esto se puede hacer mediante el establecimiento del parámetro "*disableStatistics*" a "true". En este caso, no se muestran las estadísticas obtenidas por el kernel, mientras que las estadísticas específicas del algoritmo se muestran con sus valores correspondientes.

La información que se muestra es muy diversa. En primer lugar se muestra información general acerca de la simulación: número de eventos simulados, eventos pendientes, tiempo simulación, tiempo real, etc. Después de esto, se imprimen los resultados de la evolución de la red, con una descripción de su significado. Entre estos resultados están las métricas, siendo típicas las relacionadas con las pérdidas de tráfico, y los períodos en los cuales hay saturación en los enlaces. Las definiciones sobre estas también se incluyen en el informe. Además, se puede incluir información específica sobre el algoritmo a través del método *finish()* de la clase `ITrafficAllocationAlgorithm`.

3.5.1.6. Pestaña "View reports"

De manera similar a la herramienta de generación de informes insertada en otras herramientas de Net2Plan, los usuarios pueden seleccionar un informe de entre los incluidos en la aplicación, o los propios definidos por el usuario, para aplicarlo al estado actual de la red. Tras la selección y haciendo clic en "*Show*" el informe se muestra en la parte inferior derecha de la interfaz gráfica. Una vez más, los informes se pueden abrir con un navegador web usando la opción "*View in navigator*".

3.5.2. OBJETIVOS DEL CASO DE ESTUDIO

La finalidad de este caso de estudio es introducir al usuario de Net2Plan los conceptos necesarios para entender y poder usar la herramienta *Time-varying traffic simulation*. Se hace un recorrido haciendo uso de cada posibilidad que proporciona el simulador, detallando las características que lo constituyen. Los conceptos relacionados con el desarrollo de algoritmos generadores de eventos que producen cambios en los volúmenes de tráfico, así como de algoritmos de asignación que reaccionan ante esos cambios quedan fuera de este caso de estudio.

3.5.3. ESTRUCTURA DEL VÍDEO-TUTORIAL

El vídeo-tutorial comienza con una breve introducción del concepto de *simulador de tráfico variable* en Net2Plan. Se describe el funcionamiento del mismo y se muestra la interfaz de la herramienta.

Network conditions vary during its operation:

- Failures in nodes and links
- Establishment of virtual circuits
- Variation traffic volumes

This tool simulates the network operation, where traffic demands vary with time according to a user-defined pattern.

Figura 43: Explicación sobre las tareas del simulador

Una vez abierta la herramienta, se comenta sin profundizar la interfaz de usuario, las distintas partes que componen la ventana, y la necesidad de cargar una topología con demandas asociadas antes de realizar cualquier simulación. Se hace un ejemplo demostrando lo que ocurre cuando se carga una topología sin demandas de tráfico.

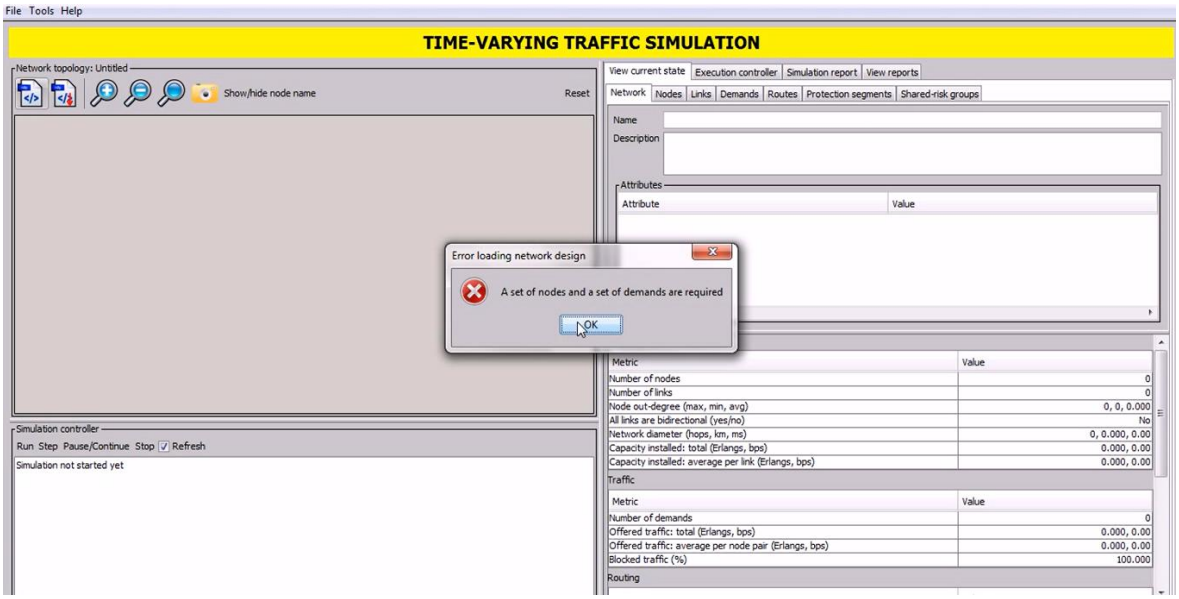


Figura 44: Error provocado por falta de demandas de tráfico en el diseño de red

Posteriormente, se explica el resto de la interfaz del simulador. Primero, se explica la pestaña *View current state* y se hace referencia a la herramienta *Offline network design*, ya que es realmente en esa herramienta donde se define la topología y el diseño de red. Debido a esto, en esta interfaz los elementos se imprimen en color gris ya que no son modificables.

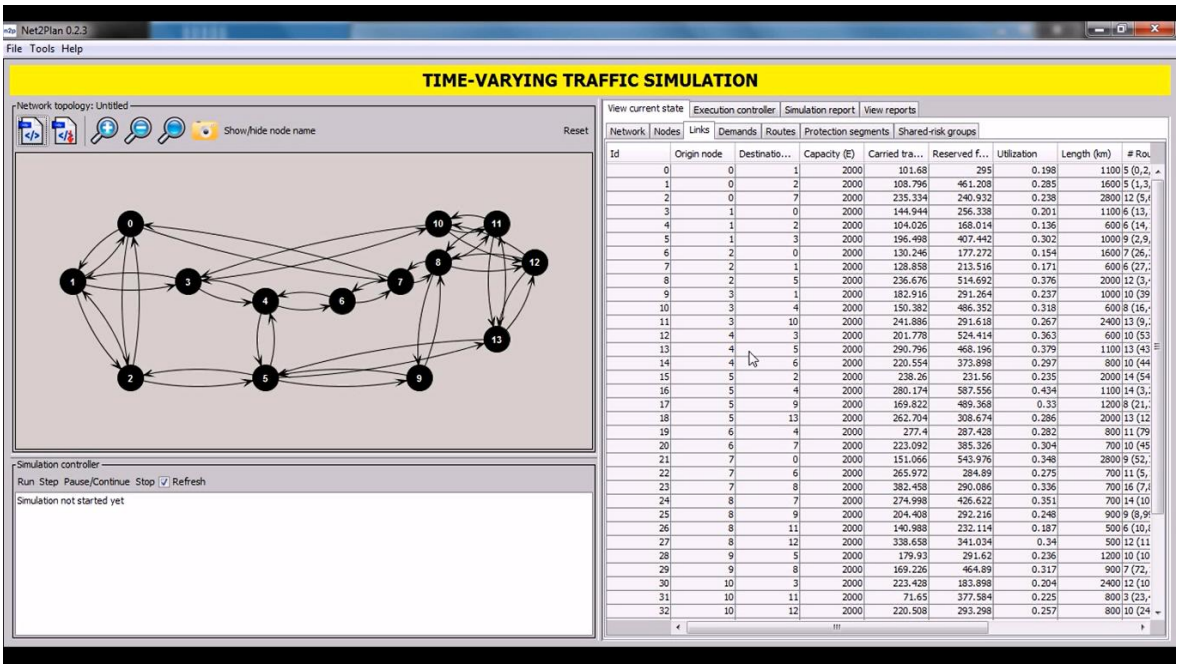


Figura 45: Explorando las pestañas de la interfaz del simulador de tráfico variable

El siguiente paso consiste en abrir la pestaña *Execution controller*. Se comienza explicando cada uno de los parámetros de simulación, tal y como se definen en el apartado inicial de este caso de estudio, haciendo hincapié en algunos parámetros más destacados, y ejemplificando el uso de otros, tales como el parámetro de actualización de la consola de simulación "*refreshTime*". Además, se hace una explicación más profunda del parámetro específico para este simulador "*timeGranularityInSeconds*", de forma que el vídeo se apoya en la siguiente transparencia para su mejor entendimiento:

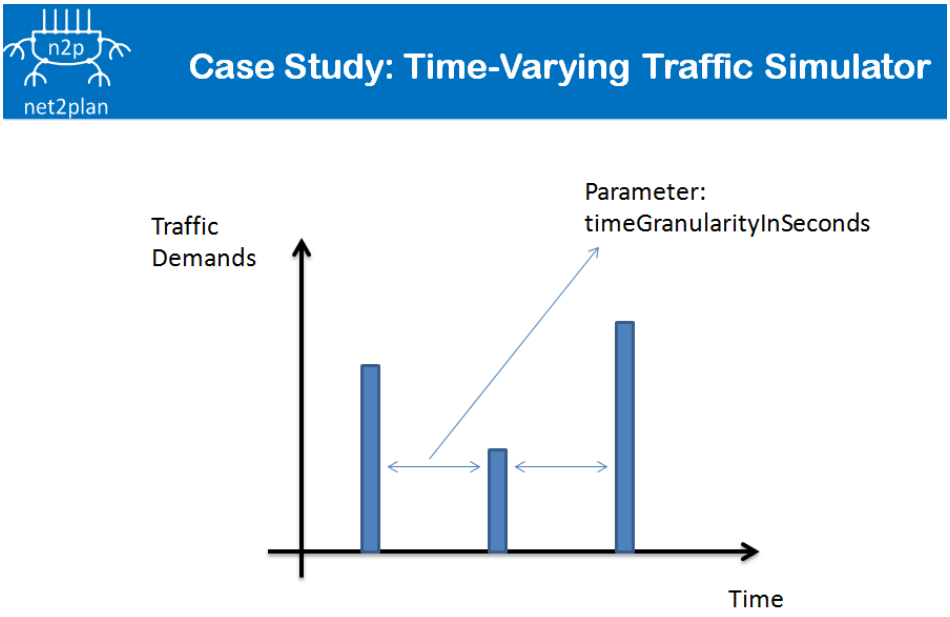


Figura 46: Explicación del parámetro de simulación *timeGranularityInSeconds*

La siguiente captura muestra el conjunto de parámetros de simulación:

Simulation execution

Simulation parameters

Parameter	Value	Description
allowExcessCarriedTraffic	true	Indicates whether or not carried traffic may b...
allowLinkOversubscription	true	Indicates whether or not links may carry more...
disableStatistics	false	Disable compilation of simulation statistics (onl...
omitProtectionSegments	false	Remove protection segments from the networ...
refreshTime	10	Refresh time (in seconds)
simEvents	-1	Total simulation events (including transitory p...
simTime	-1	Total simulation time (in seconds, including tra...
startTime	12:00	Time (in 24-hour hh:mm format) at the start o...
startWeekDay	1	Week day at the start of the simulation (1 = ...
timeGranularityInSeconds	300	Minimum time between traffic changes
transitoryEvents	-1	Number of events for transitory period (-1 me...
transitoryTime	-1	Transitory time (in seconds) (-1 means no tra...

Figura 47: Parámetros de simulación del simulador de tráfico variable

Finalmente, se acaba explicando el resto de componentes de esta pestaña, mostrando al usuario como cargar los dos algoritmos necesarios para este simulador: el generador de variaciones de tráfico y el algoritmo de asignación.

El siguiente paso consiste en explicar el ciclo de vida del simulador apoyándonos en la

transparencia introducida al inicio de este caso de estudio. La arquitectura del simulador se basa en el paradigma de simulación por eventos discretos. El funcionamiento de la red se modela como una secuencia discreta de eventos en el tiempo. Cada evento tiene lugar en un momento particular en el tiempo y marca un cambio de estado en el sistema. Entre eventos consecutivos, no se produce ningún cambio en la red; por tanto, la simulación puede saltar directamente en el tiempo de un evento a otro.

En este caso concreto, los cambios de tráfico son generados por el generador de eventos que envía estos al núcleo. El kernel a su vez envía los eventos al algoritmo de asignación, que reacciona ante la llegada de estos eventos, tomando o no acciones al respecto. Tras esas acciones se producen cambios que son recogidos por el núcleo mediante estadísticas, o por métricas propias definidas en el algoritmo de asignación. Como se comentó, este simulador genera y procesa eventos uno a uno, de manera que no tiene una lista de eventos futuros porque no los hay, se genera un evento, se procesa ese evento y se vuelve a iniciar el ciclo.

A continuación, se pasa a la simulación. Para ello, el vídeo se centra en la pestaña de control de simulación, donde se explican el conjunto de botones asociados al panel, y se comienza la simulación.

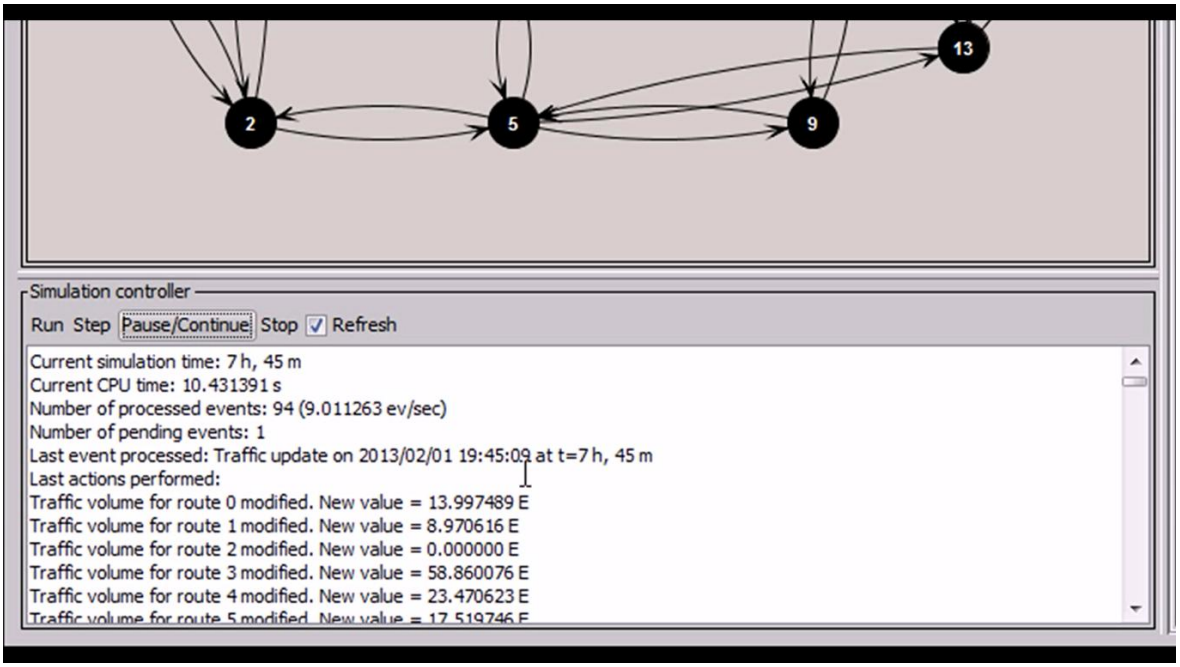


Figura 48: Panel de control de la simulación

Tras unos segundos se pausa la simulación, se observa el estado actual de la red en la pestaña *View current state*, y de nuevo se vuelve a reanudar y pausar la simulación con el objetivo de mostrar al usuario como se producen los cambios en el diseño de red. Después de esto, con la simulación en pausa, se accede a la pestaña *Simulation report* para visualizar las estadísticas generadas, y se explica al usuario como la información puede ser actualizada en cada momento de la simulación.

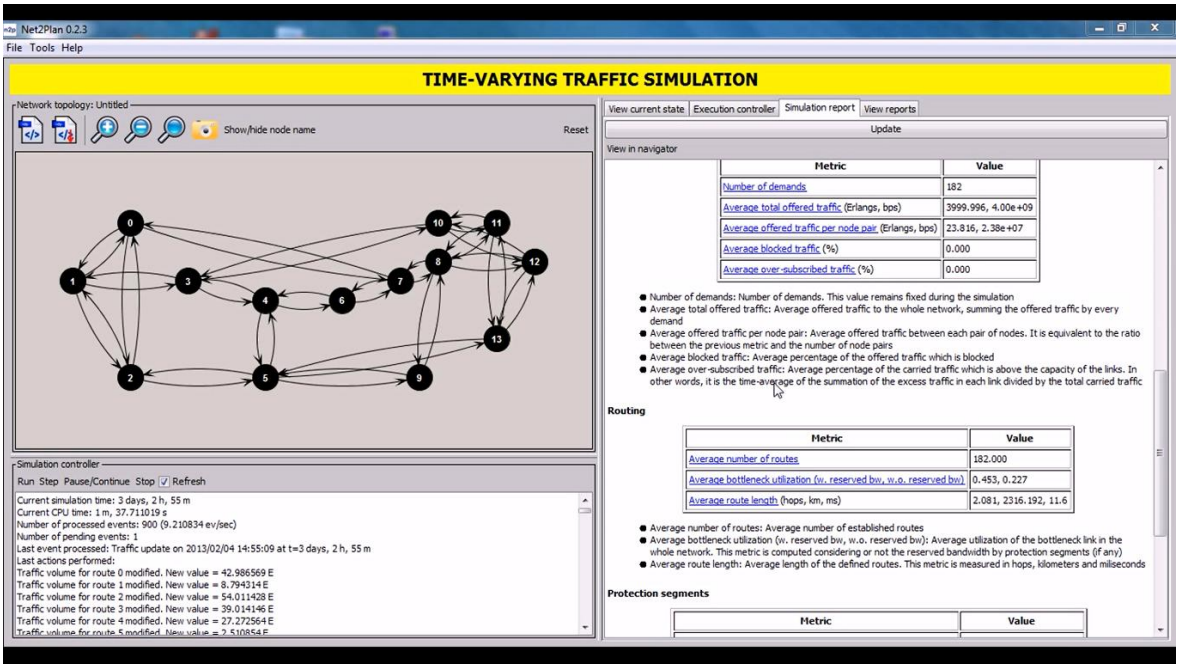


Figura 49: Visualización de estadísticas en la pestaña *Simulation report*

Tras esto, se explica cómo es posible guardar el estado actual de la red como un objeto *NetPlan* que puede ser modificado en la herramienta *Offline network design*.

Finalmente, en la pestaña *View reports*, tal y como se puede aplicar en las otras herramientas, se genera un informe aplicado al estado actual de la red.

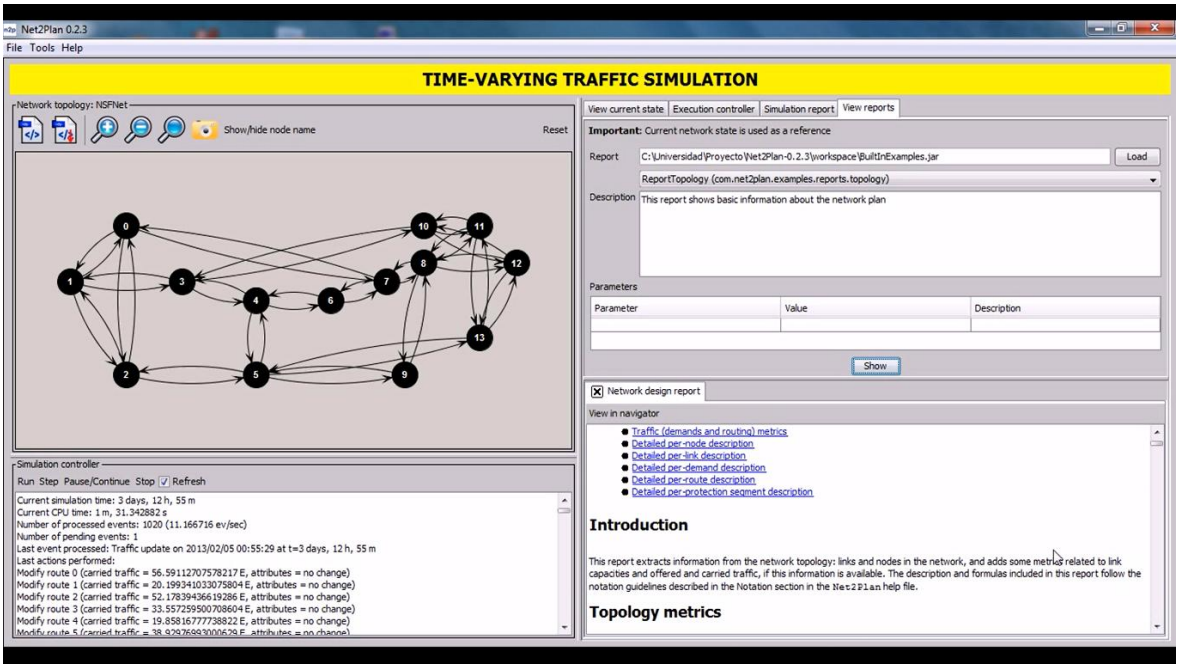


Figura 50: Generación de un informe desde la interfaz *Time-varying traffic simulation*

Para acabar, se muestra el funcionamiento del botón *Reset*, que permite reiniciar la simulación cargando de nuevo el diseño de red original empleado.

Tras la grabación del vídeo-tutorial, el siguiente paso es la integración del mismo en el canal de YouTube [8], para su difusión al conjunto de usuarios de la herramienta. En este caso, el título del vídeo es "*Getting started with Net2Plan: Time-Varying Traffic Simulation tool"*. Puede ser visualizado en [8] o accediendo a la URL https://www.youtube.com/watch?v=K982ji1fH2Y.

Figura 51: Captura del vídeo-tutorial *Time-varying traffic simulation*

4. CONCLUSIONES Y LÍNEAS FUTURAS

Net2Plan es una herramienta de planificación de redes potente ya que permite unir en una sola aplicación los requerimientos de la industria y la academia / investigación, eliminando las grandes diferencias existentes en las herramientas de estos dos campos: es flexible, ya que es aplicable a cualquier tecnología de redes actual y futura, y permite al usuario probar sus propios algoritmos de planificación; es didáctica, porque incluye un repositorio de algoritmos de diferentes tipos y para muy diversos escenarios; proporciona una interfaz gráfica sencilla y muy práctica para el usuario, con múltiples funcionalidades incluidas como diseño de topologías y matrices de tráfico, junto con simuladores y generación de informes y estadísticas. Además, es una aplicación de código abierto, de licencia gratuita y fácilmente ejecutable.

Todas estas características dotan a la aplicación de unas condiciones excelentes para su difusión e implantación tanto en empresas del sector como en asignaturas relacionadas en carreras universitarias, o para el uso de proyectos de investigación. Sin embargo, para que esto ocurra es necesario dar a conocer la aplicación en diferentes ámbitos así como dar una buena impresión al usuario final. Para conseguir esto, es de gran utilidad definir una interfaz gráfica manejable y sencilla al usuario, una web atractiva y una gran documentación que permita aprender a usar la herramienta de la manera más fácil posible, donde el usuario pueda acudir en caso de necesidad. Y es en esto precisamente donde este proyecto se ha centrado.

La idea fundamental ha sido la de mejorar la documentación escrita existente en la aplicación, la llamada guía de usuario (User's guide), pero sobre todo crear una ayuda para la herramienta más allá de un simple documento escrito que en muchas ocasiones no resuelve las dudas, principalmente prácticas, que puedan surgir al usuario. Para ello, se han desarrollado los distintos casos de estudio presentados en este trabajo en forma de vídeo-tutoriales, que recogen de una manera práctica toda la funcionalidad de Net2Plan. Estos vídeo-tutoriales no solo muestran las distintas herramientas disponibles, sino que también se describe cómo realizar e introducir algoritmos propios. Para mayor difusión, todos los vídeos están recopilados en un canal de YouTube [8], con el fin de ser lo más accesibles posible.

Como muestra de su utilidad, los vídeos se han utilizado como soporte a la docencia en las asignaturas relacionadas con planificación de redes de comunicaciones en las titulaciones de la escuela, que ya utilizaban Net2Plan con anterioridad, sirviendo de material de apoyo a los alumnos, y como material adicional en las prácticas.

En cuanto a las líneas de trabajo futuro, ya se está trabajando en la integración de Net2Plan como herramienta de soporte en redes definidas por software (SDN, *Software-Defined Networking*) [14]. Las redes SDN representan un cambio de paradigma en el concepto de control y gestión de redes. Se basan en la separación de los planos de datos y control en dos entidades distintas. Por un lado, el plano de control se localiza en un dispositivo centralizado denominado controlador, que contiene la inteligencia de red. Aunque también existen aplicaciones con múltiples controladores. Por otro lado, los dispositivos de conmutación se convierten en dispositivos sencillos que siguen las indicaciones del controlador. Este nuevo enfoque sobre la arquitectura de red permite a los administradores de la misma gestionar los servicios que ofrecen a través de la abstracción de la propia red,

separando el plano de control, gestionado de manera remota, del plano de *forwarding*, que se encarga de encaminar el tráfico. Las redes definidas por software están revolucionando el sector, atrayendo la atención tanto de la industria como de los investigadores. En este sentido, se ha creado la *Open Networking Foundation* [15] con el objetivo de desarrollar el paradigma SDN mediante el uso de estándares abiertos. Uno de los estándares que promueve es el protocolo *OpenFlow* [16], encargado de la comunicación entre los planos de control y datos, y de proporcionar un esquema de encaminamiento basado en tablas y reglas sencillas.

Otro ejemplo sobre la difusión de esta corriente es *OpenDaylight* [17], un controlador con soporte por parte de los principales *partners* de la industria (por ejemplo Cisco, Juniper, HP…). Es un proyecto *open-source*, bajo el paraguas de la Linux Foundation, con una plataforma de control modular y flexible. Está desarrollado en Java, por lo que se puede instalar en cualquier plataforma que soporte Java, al igual que Net2Plan. El controlador expone una serie de APIs, que proporcionan una interfaz para interactuar con aplicaciones externas. *OpenDaylight* soporta el desarrollo de extensiones como módulos Java, utilizando la plataforma OSGi, y la API REST para aplicaciones basadas en la API *northbound.* La plataforma OSGi se utiliza para aplicaciones que se ejecutan dentro del controlador, mientras que la API REST se utiliza para aplicaciones que se ejecutan fuera del controlador. La lógica de funcionamiento de la red y los algoritmos residen en las aplicaciones, como pudiera ser Net2Plan. Estas aplicaciones utilizan el controlador para obtener información de la red (como la topología o estadísticos), usan sus algoritmos de gestión para llevar a cabo una determinada tarea de análisis u optimización, y luego usan el controlador para insertar nuevas reglas en la red, si son necesarias.

Uno de los últimos desarrollos, no incluido en este trabajo, ha sido la conexión de Net2Plan con el controlador *OpenDaylight* para controlar una red SDN basada en *OpenFlow*. Net2Plan es capaz de obtener la información relativa al diseño de red que el controlador gestiona. Una vez que obtiene esa información sobre el diseño de red (almacenada como un objeto *NetPlan)*, se estima la matriz de tráfico a partir de las medidas de utilización de los enlaces que proporciona *OpenDaylight*, de forma que se pueda ejecutar un algoritmo de encaminamiento óptimo, y volcar las acciones correspondientes a los switches de la red. Este trabajo ha dado como resultado una ponencia invitada en el congreso internacional SaCoNeT 2014 [18]. Esto sirve como punto de partida para el desarrollo de futuros proyectos de investigación y de aplicación real dentro del ámbito de SDN.

5. REFERENCIAS

[1] Cisco Systems Inc., "Cisco Visual Networking Index: Forecast and Methodology, 2013-2018," White Paper, June 2014.

[2] J.L. Izquierdo-Zaragoza, P. Pavon-Marino, "Educational and research tools for network optimization," in *Proceedings of the 15th International Conference on Transparent Optical Networks* (ICTON 2013), Cartagena (Spain), June 2013.

[3] P. Pavon-Marino, J.L. Izquierdo-Zaragoza, "On the Role of Open-Source Optical Network Planning," in *Proceedings of the 2014 Optical Fiber Communication Conference and Exposition* (OFC 2014), San Francisco (USA), March 2014.

[4] P. Pavon-Marino, J.L. Izquierdo-Zaragoza, "Net2Plan: An open-source network planning tool for bridging the gap between academia and industry," *IEEE Network Magazine*, in press.

[5] Net2Plan – The open-source network planner [Online]. Available: http://www.net2plan.com/ [Last accessed: July 2014]

[6] JOM – Java Optimization Modeler [Online]. Available: http://ait.upct.es/~ppavon/jom/. [Last accessed: July 2014]

[7] Camtasia Studio [Online]. Available: http://www.techsmith.com/camtasia.html [Last accessed: July 2014]

[8] YouTube Net2Plan Channel [Online]. Available: https://www.youtube.com/channel/UCCgkr1wlMlO221yhFGmWZUg [Last accessed: July 2014]

[9] L. Kleinrock, *Queueing Systems, Volume 2: Computer Applications*, Wiley, 1976.

[10] S. Robinson, *Simulation: The Practice of Model Development and Use*, Wiley, 2004.

[11] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast accurate computation of large-scale IP traffic matrices from link loads," in *Proceedings of the International Conference on Measurements and Modeling of Computer Systems* (SIGMETRICS 2003), June 2003.

[12] R.S. Cahn, *Wide area network design: concepts and tools for optimization*, Morgan Kaufmann Publishers Inc., 1998.

[13] Eclipse Website [Online]. Available: http://www.eclipse.org/ [Last accessed: July 2014]

[14] B.A.A. Nunes, M. Mendonça, X.N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, in press.

[15] Open Networking Foundation [Online]. Available: https://www.opennetworking.org/ [Last accessed: July 2014]

[16] OpenFlow Protocol [Online]. Available: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow [Last accessed: July 2014]

[17] OpenDaylight [Online]. Available: http://www.opendaylight.org/ [Last accessed: July 2014]

[18] J.L. Izquierdo-Zaragoza, A. Fernandez-Gambin, J.J. Pedreno-Manresa, P. Pavon-Marino, "Leveraging Net2Plan planning tool for network orchestration in OpenDaylight," in *Proceedings of the 5th International Conference on Smart Communications in Network Technologies* (SaCoNeT 2014), Vilanova i la Geltru (Spain), June 2014.