Redes de Interconexión

Manual de Sauron

Antonio Morán Muñoz

Escuela superior de ingeniería informática

Redes de Interconexión

Manual de Sauron

Antonio Morán Muñoz
Instituto de Investigación en Informática (I3A)

Esta publicación está basada en la plantilla NIST Handbook de Overleaf

28 de octubre de 2024

Departamento de Sistemas Informáticos

Universidad de Castilla-La Mancha

National Institute of Standards and Technology Handbook XXXX
Natl. Inst. Stand. Technol. Handbook XXXX, 19 pages (Month Year)

This publication is available free of charge from:
https://doi.org/10.6028/NIST.HB.XXXX

Foreword

Delete if not applicable

Preface

Delete if not applicable

Abstract

Required

Key words

Required, alphabetized, separated by semicolon, and end in a period.

Índice general

Índice de cuadros

Índice de figuras

Glosario

- Sentencia de demora *(delay statement)*.

Capítulo 1

Introducción

SAURON es un proyecto académico para investigar mejoras en redes de interconexión de altas prestaciones utilizando el entorno de simulación que proporciona OMNeT++. Se trata de una red de computación de altas prestaciones basada en el framework OMNeT++. SAURON es un acrónimo de *Simulador de ArqUitecturas de Red en OmNet++*.

Esta herramienta ha sido desarrollada en el departamento de sistemas informáticos de la Universidad de Castilla-La Mancha, en Albacete. Su desarrollo arrancó en el verano de 2011 como el Trabajo Fin de Grado de Pedro Yébenes Segura (programador principal), que fue supervisado por Jesús Escudero Sahuquillo, Pedro Javier García García y Francisco José Quiles Flor.

1.1. Características

SAURON modela las siguientes características:

- **Topologías.** Meshes, tori, fat trees, KNS, DragonFly, SlimFly.
- **Algoritmos de encaminamiento.** DOR, destro, Hybrid-DOR, MIN routing for dragonfly topology, etc.
- **Esquemas de colas.** VOQnet, VOQsw, DBBM, OBQA, BBQ, Flow2SL, H2LQ, IO-DET, etc.
- **Calidad de servicio (QoS).** InfiniBand-based VLTables.
- **Tráfico basado en tráfico real.** Por medio de la biblioteca TraceLib.

Capítulo 2

Modo usuario

2.1. Clonación

El primer paso a la hora de clonar el repositorio de SAURON es crear un directorio donde albergarlo. Llamaremos a este directorio `src`. La clonación del repositorio en una máquina local se puede llevar a cabo a través de SSH o de HTTPS.

- SSH. Este método sólo funciona desde la red de la universidad por lo que, si se desea clonar fuera de dicha red, será necesario utilizar una VPN. Tampoco funciona en CELLIA. Otra cosa a tener en cuenta es la creación de las correspondientes claves privada/pública[1].
- HTTPS. Este método pedirá introducir el usuario y la contraseña, a no ser que usemos tokens[2]. El Listing 2.3 muestra la instrucción necesaria para incluir un token.

Listing 2.1. Instrucción para incluir un token.

```
git config --global url."https://git:token@gitraap.i3a.info".
    insteadOf "https://gitraap.i3a.info"
```

El Listing 2.2 muestra las instrucciones necesarias para clonar el repositorio de SAURON y moverse a su directorio raíz.

Listing 2.2. Instrucción para incluir un token.

```
mkdir -p ~/src; cd ~/src
git clone https://gitraap.i3a.info/jesus.escudero/SAURON.git
cd SAURON
```

[1] https://docs.gitlab.com/ee/user/ssh.html
[2] https://gitraap.i3a.info/-/user_settings/personal_access_tokens

2.2. Ramas

El desarrollo de SAURON se divide en una serie de ramas de desarrollo de características (e.g. `dev-scripts`, `dev-tracelib`, `dev-qos`...) que, una vez implementadas y testeadas, se integrarán en la rama principal de desarrollo, llamada `Development`.

2.2.1. Merge Requests

Cuando se implementa alguna funcionalidad dentro de una de las ramas de desarrollo de características, es recomendable hacer un *merge* de `origin/Development`, resolver los conflictos que puedan surgir y revisar el pipeline CI/CD. Asimismo, es recomendable programar tests (tanto escenario como script de ejecución) que comprueben los resultados de simulaciones para añadirlos al CI/CD.

2.3. Scripts

En SAURON se incluyen una serie de scripts que configuran diferentes aspectos del simulador. Los scripts más importantes son los siguientes:

- `./scripts/server/compilar`. Este script hay que ejecutarlo para compilar todo el proyecto de SAURON. Para su correcta ejecución hay que cambiar las variables de entorno `SAURON_ROOT` y `OMNET_ROOT`.
- `./scripts/server/run`. Este script hay que ejecutarlo para lanzar una simulación.
- `cellia_remote_build`. Este script compila SAURON en CELLIA directamente.
- `run_cellia`. Este script ejecuta el modelo en CELLIA.
- `./scripts/interactive/gen-vscode-cfgs.py`. Este script genera las configuraciones necesarias para trabajar con VSCode.
- `./scripts/server/callgrind`. Este script lleva a cabo un profile de llamadas a métodos.
- `./scripts/server/valgrind`. Este script lleva a cabo un profile de memoria.

Para que funcionen ciertos scripts es necesario instalar una serie de dependencias en Ubuntu/Debian. Para instalarlas, se utiliza el siguiente comando:

Listing 2.3. Instrucción para instalar las dependencias de SAURON.

```
sudo apt install fzf libnotify-bin valgrind kcachegrind
```

2.4. INIs

En el directorio `simulations/` hay archivos de configuración listos para lanzar simulaciones. Cada directorio, si pertenece a una topología específica, contendrá también el fichero `NED` que define dicha topología.[3]
Las estadísticas que se quieran recoger se definirán dentro de los módulos en los ficheros `NED` correspondientes. Para escoger cómo se guardarán los resultados hay que indicarlo en los ficheros `INI`, de configuración. **Es importante elegir correctamente los modos de guardado de las estadísticas** [4].

2.4.1. Estadísticas

Cuando termina una simulación, los resultados se guardan por defecto en formatos `.vec`, que contiene datos vectoriales, y `.sca`, que contienen datos escalares. Para generar gráficos a partir de los resultados obtenidos, se puede usar el IDE o generar un CSV a partir de los ficheros antes mencionados[5]. El comando a ejecutar para la generación de los ficheros `.csv` se muestra en el Listing 2.4

Listing 2.4. Instrucción para generar un CSV a partir de los resultados.

```
opp_scavetool x test/ci-tests/qos/results/*sca -o synth.csv
```

Algunas opciones extra para formatear las gráficas (tamaño de la letra, ejes, grosor de las líneas...) se pueden encontrar en Internet [6].

2.5. Wiki (WIP)

La wiki del repositorio incluye información sobre esquemas de colas, patrones de tráfico y diferentes guías (guía de inicio, guía del directorio `simulations/`).

2.6. Patrones de tráfico

En SAURON podemos generar diferentes tipos de tráfico. A continuación se describen algunos de los siguientes:

2.6.1. Random

2.6.2. Hotspot

El tráfico hotspot consiste en la generación de dos tráficos en realidad:

[3] https://gitraap.i3a.info/jesus.escudero/SAURON/-/wikis/simulations-dir
[4] https://doc.omnetpp.org/omnetpp/manual/#sec:ana-sim:configuring-recording-modes
[5] https://doc.omnetpp.org/omnetpp/manual/#sec:ana-sim:scavetool
[6] https://matplotlib.org/stable/users/explain/customizing.html#the-default-matplotlibrc-file

- Tráfico hotspot propiamente dicho. Dicho tráfico se encarga de generar un punto caliente en la red (*hotspot*). Este tráfico se combinará con un tráfico random sintético, de tal manera que se muestre cómo afecta dicho punto caliente en éste último.
- Tráfico random sintético. Este tráfico envía mensajes de cualquier nodo a cualquier nodo de manera aleatoria.

La definición de estos tráficos se puede llevar a cabo de diferentes maneras:

1. Creación de dos aplicaciones. Este método es útil para crear un conjunto de estadísticas para cada aplicación. Por ejemplo, si sólo interesa ver las estadísticas para el tráfico random sin tener en cuenta las del hotspot, este método puede resultar más conveniente.
2. Creación de ficheros `.IN`. Este método es útil si se quiere obtener unas estadísticas globales para toda la red. Para ello, hay que especificar un fichero con extensión `.IN` en el que se especifica dónde y qué tipo de tráfico va a enviar cada nodo de la red. Como una red puede constar de miles de nodos, existe un script en Python (`/scripts/interactive/TrafficGen.py`) que se encarga de crear dichos ficheros en función de una serie de parámetros.

Capítulo 3

Modo desarrollador

Esta sección está dedicada a explicar más en detalle los aspectos más relevantes a tener en cuenta a la hora de desarrollar dentro de SAURON.

3.1. Estructura de SAURON

Para entender la estructura de SAURON, primero se van a explicar unos breves conceptos generales aplicados al mismo SAURON. Tras esto, se pasará a explicar más en detalle la estructura del proyecto y cada uno de sus subdirectorios.

3.1.1. Conceptos generales

Este proyecto, como cualquier proyecto de OMNeT++, incluye una serie de componentes que se enumeran a continuación:

- **Módulos compuestos.** Estos módulos son resultado de la composición de diferentes módulos simples que encapsulan funcionalidades concretas y, generalmente, no tienen una clase asociada. Todas las redes que se modelen incluirán los siguientes módulos compuestos:
  - *Sys.* Genera los mensajes, los empaqueta y manda a las HCAs para que se inyecten en la red. AÑADIR IMAGEN
  - *HCA.* Inyectan y reciben los paquetes de la red. AÑADIR IMAGEN
  - *Switches.* Dirigen los paquetes hacia su destino, conmutando estos de la entrada a la salida que más convenga a su destino. AÑADIR IMAGEN

NetConf

Un fichero importante es `NetConf` que, a pesar de no ser un módulo compuesto, incluye parámetros de configuración de la red.
AÑADIR IMAGEN

- **Módulos simples.** Estos módulos contienen la definición de elementos que se encargan de una función particular y cuya composición da lugar a módulos más complejos que modelan dispositivos reales, como pueden ser switches, NICs o routers. Los módulos simples tienen una clase de C++ asociada en la que se implementa su funcionalidad. La clase que se asocia a cada módulo simple hereda de `cSimpleModule` y, tiene que sobreescribir una serie de funciones que hereda de la clase padre, y que se encargan de definir qué hace el módulo cuando se inicia, recibe un paquete o finaliza su ejecución. La declaración de estas funciones se muestra en el Listing 3.1.

Listing 3.1. Declaración de las funciones que se heredan de cSimpleModule, y que hay que sobreescribir.

```
void initialize();
virtual void handleMessage(cMessage *p_msg);
void finish();
```

initialize ()

Esta función se ejecuta al inicio de la simulación, cuando se crea el módulo correspondiente. Aquí se pueden definir etapas, si algún módulo necesita inicializarse después que otro. La función puede leer los parámetros declarados en los ficheros `.NED` e inicializados en el fichero `.INI`. También se puede llevar a cabo la inicialización de instancias de otras clases.

Listing 3.2. Sobreescritura de la función initialize() dentro de uno de los módulos.

```
1  void Arbiter_2P_DTable::initialize(int stage) {
2      Arbiter_TwoPhased::initialize(stage);
3      if (stage == 1) {
4          NumPorts = getAncestorPar("numPorts");
5          int numQueues = getAncestorPar("numQueues");
6          const char* fileName = par("VLTableFileName").
               stringValue();
7          uint16_t mtuFlits = getSimulation()->
               getModuleByPath("sys.sysMng")->par("mtuFlits").
               intValue();
8          DTable = new ArbitrationDeficitTable*[NumPorts];
9          for (int i = 0; i < NumPorts; i++) {
10             DTable[i] = new ArbitrationDeficitTable(
                   fileName, numQueues, mtuFlits);
11         }
12         #if DEBUG_ARBITER_QUEUE_SELECTION
13         bytesGranted.resize(NumQueues);
14         #endif
15     }
16 }
```

handleMessage (cMessage *p_msg)

Esta función se ejecuta cuando el módulo recibe un mensaje. Dependiendo del tipo de mensaje, se pueden ejecutar funciones específicas.

Listing 3.3. Sobreescritura de la función handleMessage() dentro de uno de los módulos.

```
1  void Arbiter::handleMessage(cMessage *p_msg) {
2  #if DEBUG
3      EV << getFullPath() << " - FUNC: handleMessage " <<
           endl;
4  #endif
5
6      switch (p_msg->getKind()) {
7      case REQUEST_MSG:
8          handleRequestMsg((SimpleRequestMsg*) p_msg);
9          break;
10     case CREDIT_MSG:
11         handleCreditMsg((CreditMsg*) p_msg);
12         break;
13     case CONTROL_MSG:
14         handleControlMsg((ControlMsg*) p_msg);
15         break;
16     case FREE_MSG:
17         handleFreeMsg((FreeMsg*) p_msg);
18         break;
19     default:
20         delete p_msg;
21         break;
22     }
23 }
```

finish()

Esta función se ejecuta al final de la simulación y se puede utilizar para emitir y recolectar estadísticas del módulo, así como para limpiar la memoria que se haya reservado al inicio y durante la simulación.

Listing 3.4. Sobreescritura de la función finish() dentro de uno de los módulos.

```
void Arbiter::finish() {
    delete[] OBusy;
    for (int i = 0; i < NumPorts; i++) {
        delete[] IBusy[i];
    }
    if (BlockingOQueue) {
        for (int i = 0; i < NumPorts; i++) {
            delete[] numCreditsToUnblock[i];
        }
        delete[] numCreditsToUnblock;
    }
    delete[] IBusy;

    delete Grants;
    delete InstanceVOQLogic;
    delete InstanceVNAllocator;
    delete InstanceCreditChecker;
    delete InstanceRoutingUnit;
}
```

Algunos de los módulos simples definidos en SAURON son los siguientes:

- Port
- Crossbar
- Arbiter
- InjectionQueues
- Sink
- SystemManager
- Application
  - SystheticApplication
  - TraceApplication

- **Mensajes.** Incluyen la definición de formatos de mensaje específicos. Los mensajes se utilizan para que los módulos se comuniquen entre sí, o para retroalimentar la máquina de estados de un módulo. Los mensajes se definen dentro del fichero `src/Message.msg` y, a partir de este, OMNeT++ generará las diferentes clases correspondientes a cada tipo de paquete.

Listing 3.5. Definición de mensajes.

```
void initialize();
virtual void handleMessage(cMessage *p_msg);
void finish();
```

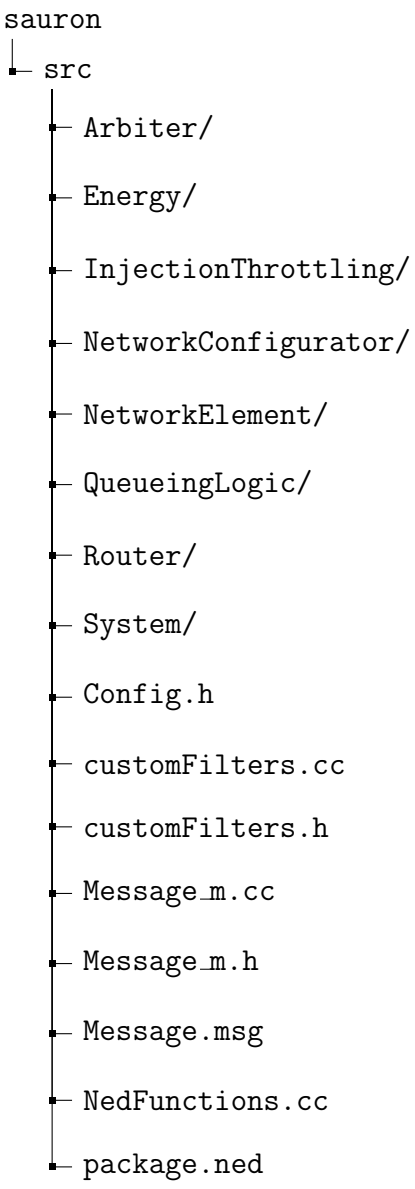
3.1.2. Árbol de directorios

El árbol de directorios de SAURON es el siguiente:

```
sauron
├── Binaries/
├── Archives/
├── Includes/
├── docker/
├── src/
├── test/
├── TraceLib/
├── out/
├── scripts/
├── simulations/
├── CHANGELOG
├── Makefile
└── README.md
```

src

El directorio `src/` contiene los ficheros fuente de los diferentes elementos que compondrán la red de interconexión. Dicho directorio contiene la siguiente estructura:



- `Arbiter/`. Este directorio contiene los ficheros fuente que definen y describen los módulos dedicados al arbitraje dentro de la arquitectura de los switches y las NIC.
- `Energy/`.
- `InjectionThrottling/`.
- `NetworkConfigurator/`. Este directorio contiene los ficheros fuente que definen diferentes aspectos de la red, como la topología a usar. En él se definen parámetros genéricos para todo tipo de topología, como el número de nodos, el número de apli-

caciones, y otros parámetros más específicos del tipo de topología, como el número de fases en las topologías indirectas.

- `NetworkElement/`. Este directorio contiene los ficheros fuente que definen y describen módulos relativos a los dispositivos de red. Incluye tanto módulos simples, como son los puertos (*Port*) o el crossbar (*CrossbarRegister*); como módulos compuestos, como son los switches y HCAs, que se componen de módulos simples situados en este directorio o fuera de este (e.g. los localizados en el directorio `Arbiter/`).
- `QueueingLogic/`.
- `Router/`.
- `System/`. Este directorio contiene los ficheros fuente que definen los módulos que se encargan de generar los mensajes, empaquetarlos e inyectarlos a la red a través de las HCA.

simulations

El directorio `simulations/` contiene los ficheros fuente de las diferentes topologías a simular, así como los patrones de tráfico que se lanzarán durante la simulación. Aquí se incluyen los diferentes ficheros de configuración `omnetpp.ini`, donde se inicializan los diferentes parámetros declarados en los diferentes ficheros `.NED` que componen la simulación. A continuación se muestra el árbol de directorios de `simulations/`. Con el fin de que no salga un árbol demasiado extenso, sólo se mostrarán los directorios que se desarrollarán más adelante, mientras que los puntos suspensivos representan una serie de directorios de topologías cuya estructura es similar a la de la topología que se explicará:

```
sauron
└─ simulations
   ├─ ...
   ├─ RLFT/
   ├─ ...
   ├─ TrafficConfigurations/
   └─ package.net
```

- `RLFT/`. Este directorio contiene los ficheros fuente que definen una red con topología *Recursive Low-Diameter Fat Tree*. Dichos ficheros incluyen el fichero de configuración `omnetpp.ini` y el fichero de descripción de la red `RLFT.ned`. Además, aquí también se incluye un directorio que contiene los resultados de las simulaciones lanzadas con esta topología.

- `TrafficConfigurations/`. Este directorio contiene los ficheros fuente que definen los diferentes patrones de tráfico que se pueden lanzar en la red que se va a simular. Debido a que el patrón de tráfico depende de la clase de topología, existen diferentes directorios para cada clase (e.g. redes directas, redes indirectas, *dragonfly*...). Cada uno de estos subdirectorios contiene los ficheros de configuración para cada patrón de tráfico.

- `package.ned`.

3.1.3. Ramas

Al igual que se comentó en el Capítulo 2 cuando se implementa alguna funcionalidad dentro de una de las ramas de desarrollo de características, es recomendable hacer un *merge* de `origin/Development`, resolver los conflictos que puedan surgir y revisar el pipeline CI/CD. Así como programar tests (tanto escenario como script de ejecución) que comprueben los resultados de simulaciones para añadirlos al CI/CD.

AÑADIR IMAGEN

3.1.4. Issues

Cuando se detecte algún problema en el simulador o se eche en falta alguna funcionalidad, se crea un *Issue* que refleje dicha incidencia. Con el fin de mantener los issues organizados, se asignará una etiqueta cuando se vayan creando. Además, se crea una rama asociada al issue para resolverlo. Una vez resuelto, se hace un *merge* en la rama `Development`.

AÑADIR IMAGEN

3.1.5. Integración continua (Test CI)

Para asegurar que las funcionalidades básicas del simulador funcionan cada vez que se suben *commits* al repo, se ejecutan una serie de test que comprueban que el simulador funcione como antes de incorporar los nuevos cambios.

AÑADIR IMAGEN

¿Qué hacer si falla algún test?

El primer paso es inspeccionar la salida de los *CI tests*. Tras esto, se descarga el artefacto para, a continuación, analizarlo en local repitiendo las pruebas en busca del fallo.

Tras la implementación de los tests de integración continua nuevos, hay que declararlos dentro del fichero `.gitlab-ci.yml` que será utilizado por los runners de GitLab CI/CD para ejecutar dichos tests dentro de un pipeline.

Dentro del directorio `tests/` se encuentran los scripts de los diferentes tests, los archivos `INI` utilizados y las trazas que usan los tests.

3.2. Ejemplo: Generación de simulaciones y recogida de resultados

A continuación, se verá un ejemplo en el que se lanzará una simulación de una red *Fat Tree* con 8 nodos utilizando un patrón de tráfico sintético que lance mensajes a la red de manera aleatoria. Dicho tráfico se lanzará en un conjunto de 10 simulaciones en las que se variará el porcentaje de carga en la red.

3.2.1. Configuración de la red

La configuración de la red a simular se llevará a cabo en el fichero `omnetpp.ini` localizado en el directorio de la topología correspondiente que se quiera simular (véase Sección 3.1.2). Dentro de dicho fichero habrá que especificar parámetros como la topología, el algoritmo de encaminamiento, el patrón de tráfico... de la red.

El fichero `omnetpp.ini` presenta diferentes secciones. La principal, llamada `[General]` define los parámetros generales de la red. Hay otra sección llamada `portConfig` que define la configuración de los puertos de los dispositivos que participan en la red. Esta sección es una extensión de una de las extensiones definidas en los ficheros de configuración `PortPFC.ini` y `OriginalPort.ini`, donde se encuentran la definición de los parámetros que definen la arquitectura de los switches. Por último, se pueden definir secciones propias en el caso de que se quieran lanzar diferentes simulaciones con pequeñas variaciones en algún aspecto de la red.

En la Figura 3.6 se muestra un ejemplo de un fichero de configuración en el que se muestran las diferentes secciones comentadas anteriormente.

Listing 3.6. Configuración de la red.

```
[General]
network = RLFT

cmdenv-status-frequency = 5s
sim-time-limit = 0.002000001s
cmdenv-interactive=true
#Net
**.topology = "rlft"
**.channelDistance = 5m

#Routing and Arbitration
**.routingAlgorithm = "destro"
**.H[*].arbiter.typename = "Arbiter_TwoPhased"
**.SW[*].arbiter.typename = "Arbiter_ThreePhased"
**.requestProcessingTime = 6ns

#Logging
**.logInterval = 10us

**.sysMng.**.result-recording-modes = default, +vector,+histogram
```

```
**.sys.app*-**.result-recording-modes = default, +vector,+histogram
**.H[*].**.scalar-recording = false
**.SW[*].**.scalar-recording = false
**.param-recording = false

#Port Configurations
include ../TrafficConfigurations/PortPFC.ini
include ../TrafficConfigurations/originalPort.ini


# Port configuration for IQ switch
[Config portConfig] #Note that the config included extends from
   General and portConfig, change the next extend to test other
   technologies
#extends=IB-QDR  # IQ
extends=bxi3  # CIOQ

#Queuing schemes configurations
include ../TrafficConfigurations/indirect/schemes.ini

#Traffic generation modes
include ../TrafficConfigurations/indirect/random.ini
include ../TrafficConfigurations/indirect/hotspot.ini
include ../TrafficConfigurations/indirect/victim.ini
include ../TrafficConfigurations/indirect/local.ini
include ../TrafficConfigurations/indirect/storage.ini
include ../TrafficConfigurations/indirect/ebb.ini
include ../TrafficConfigurations/indirect/ebb_hs.ini
include ../TrafficConfigurations/indirect/zipf1.ini
include ../TrafficConfigurations/indirect/sla3.ini
include ../TrafficConfigurations/indirect/ptrans.ini
include ../TrafficConfigurations/indirect/natRing.ini
include ../TrafficConfigurations/indirect/graph500.ini


# 8 nodes - Random traffic - Aging arbiter - IQ switch
[Config SynthAgingIQ8Nodes]
extends=portConfig

# Network
**.arity = ${arity=2}
**.numStages = ${numStages=2}
**.numNodes = int(2*pow(${arity},${numStages}))

# Switch
#**.SW[*].virtualOutputQueues = false

# Applications
# These are defined in the configuration file of the specific
   traffic configuration
**.firstLog = 0.00000s
```

```
**.numApps = 1
**.app[0].typename = "SyntheticApplication"
**.app[0].msgSize = ${header}B+${data}B
**.app[0].load = ${load=10..100 step 10}
**.app[0].fileName = "R"
#**.app[0].msgForStats = 10000

# Queueing scheme
**.congestionControlTechnique = "1q"
**.numQueues = 1
```

Topología

La topología se indica en el parámetro `topology`, que viene declarado en el fichero `NetworkConfigurator.ned`. En este caso, se debe indicar una topología `"rlft"`, correspondiente a la *Fat Tree*.

Encaminamiento y arbitraje

El algoritmo de arbitraje

References

[1] Wilkinson JP (1990) Nonlinear resonant circuit devices. United States Patent 3 624 125.

[2] Joint Task Force Transformation Initiative Interagency Working Group (2013) Security and privacy controls for federal information systems and organizations (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-53, Rev. 4, Includes updates as of January 22, 2015. https://doi.org/10.6028/NIST.SP.800-53r4

[3] National Institute of Standards and Technology (2001) Security requirements for cryptographic modules (U.S. Department of Commerce, Washington, D.C.), Federal Information Processing Standards Publications (FIPS PUBS) 140-2, Change Notice 2 December 03, 2002. https://doi.org/10.6028/NIST.FIPS.140-2

[4] Xiong H (2015) Multi-level bell-type inequality from information causality and noisy computations. *Chinese Journal of Electronics* 24(2):408–413. https://doi.org/10.1049/cje.2015.04.031

[5] Prives L (2016) For whom the bell tolls: Inventing success through creativity and analytical skills [wie from around the world]. *IEEE Women in Engineering Magazine* 10(1):37–39. https://doi.org/10.1109/MWIE.2016.2535841

[6] Roberts LJ (1982) Cameras and systems: A history of contributions from the bell; howell co. (part i). *SMPTE Journal* 91(10):934–946. https://doi.org/10.5594/J00229

[7] Maloney TJ (2016) Unified model of 1-d pulsed heating, combining wunsch-bell with the dwyer curve: This paper is co-copyrighted by intel corporation and the esd association. *38th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD)* (Publisher name, location), Vol. 22, pp 1–8. https://doi.org/10.1109/EOSESD.2016.7592562

[8] Giancoli D (2008) *Physics for Scientists and Engineers with Modern Physics* (Pearson Education), 4th Ed.

[9] Eston P (1993) *Book section title* (The name of the publisher, The address of the publisher), Vol. 4, Chapter 8, 3rd Ed., pp 201–213.

[10] Behrends R, Dillon LK, Fleming SD, Stirewalt REK (2006) White paper: Programming according to the fences and gates model for developing assured, secure software systems (Department of Computer Science, Michigan State University, East Lansing, Michigan), MSU-CSE-06-2.

[11] Farindon P (1993) The title of the collection section. *The title of the book*, ed Lastname F (The name of the publisher, The address of the publisher), Vol. 4, pp 201–213.

[12] Marcheford P (1993) The title of the unpublished work.

[13] Joslin P (1993) *The title of the PhD Thesis*. Ph.D. thesis. The school of the thesis, The address of the publisher. An optional note.

[14] Caxton P (1993) The title of the booklet. How it was published, The address of the publisher. An optional note.

[15] Isley P (1993) The title of the webpage. Available at https://nist.gov.

Acknowledgments

Delete if not applicable

Appendix A: Supplemental Materials

Brief description of supplemental files

Appendix B: Change Log

If updating document with errata, detail changes made to document – delete if not applicable.