

# DRMSIM: a network simulator for the investigation of routing schemes

## *User manual*

Luc Hogie<sup>1</sup>, Issam Tahiri<sup>1</sup>, Dimiti Papadimitriou<sup>2</sup>, Frédéric Majorczyk<sup>3</sup>

<sup>1</sup>Mascotte project, INRIA Sophia-Antipolis (FR)

<sup>2</sup>LaBRi, University of Bordeaux (FR)

<sup>3</sup>Alcatel Lucent-Bell, Antwerpen (BE)

February 1, 2010

## Contents

<b>1</b>	<b>Package overview</b>	<b>4</b>
<b>2</b>	<b>Installation</b>	<b>4</b>
2.1	Prerequisite . . . . .	4
2.2	Procedure . . . . .	5
2.3	Files installed . . . . .	5
<b>3</b>	<b>DRMSim's commands</b>	<b>6</b>
<b>4</b>	<b>Software architecture</b>	<b>8</b>
4.1	Module dependencies . . . . .	8
4.1.1	Mascsim . . . . .	8
4.1.2	Dipergrafs . . . . .	9
4.1.3	Java4unix . . . . .	9
4.1.4	Graphstream . . . . .	9
4.2	Object-oriented model . . . . .	9
4.3	Measurement model . . . . .	9
4.4	Graphical monitoring of the routing process . . . . .	12
<b>5</b>	<b>Introduction to routing</b>	<b>12</b>
5.1	Routing in the current Internet . . . . .	12
5.2	Classification of routing protocols . . . . .	13
5.2.1	Distance-vector and link-state protocols . . . . .	13
5.2.2	AS-level or router-level routing . . . . .	13

<b>6</b>	<b>Simulation models</b>	<b>13</b>
6.1	Network model . . . . .	14
6.1.1	On memory consumption and computational performance	14
6.2	Routing protocols . . . . .	14
6.2.1	Standard set . . . . .	14
6.2.2	BGP . . . . .	15
6.2.3	RIP . . . . .	19
6.3	Metrics . . . . .	19
6.4	Topology generators . . . . .	20
6.4.1	Standard set . . . . .	21
6.4.2	Randomized set . . . . .	21
6.4.3	Tier 1/Tier 2/Tier T3 . . . . .	21
6.4.4	Inet . . . . .	22
6.4.5	Brite . . . . .	22
6.4.6	GT-ITM . . . . .	22
6.4.7	CAIDA . . . . .	23
6.5	Dynamics . . . . .	23
6.6	Traffic model . . . . .	23
6.7	Time model . . . . .	23
6.8	Granularity . . . . .	23
<b>7</b>	<b>Frequently Asked Questions (FAQs)</b>	<b>23</b>
7.1	What is the input of the simulator? . . . . .	23
7.2	Is it possible to define my own topology? . . . . .	24
7.3	...deal with the output . . . . .	24
7.4	How can I add a new metric . . . . .	24
7.5	How can I generate a GLP graph . . . . .	24
7.6	How to implement a new routing protocol . . . . .	25
7.7	Is there a way to import a new topology (CAIDA, Inet)) . . . . .	25
7.8	How to run DRMSIM? . . . . .	25
7.8.1	One to one . . . . .	25
7.8.2	All to all . . . . .	25
7.8.3	DoNothing . . . . .	25
7.9	...monitor a simulation process . . . . .	25
<b>8</b>	<b>Implementation notes</b>	<b>26</b>
8.1	Code reuse . . . . .	26
8.1.1	Madhoc . . . . .	26
8.1.2	Dipergrafs . . . . .	26
8.1.3	GraphStream . . . . .	26
8.1.4	CoLibA . . . . .	26
8.1.5	Ploticus . . . . .	27
8.2	Invocation . . . . .	27
8.2.1	Drawing a network: <code>alu-network-printer</code> . . . . .	27
8.2.2	Running a simulation: <code>mascsim.compute.simulation</code> . . . . .	27
8.2.3	Running a simulation: <code>alu-scan-available-topology-generators</code> . . . . .	27

8.3	Results . . . . .	27
8.4	Simulation campains . . . . .	27

This working paper describes DRMSIM, a software simulator that aims at the evaluation of routing protocols at a large scale. Indeed most simulator do not allow the simulations of more than a few hundred nodes. DRMSIM is an attempt to simulate up to ten thousand nodes on typical workstations. Its development happens in the context of a study focusing on dynamic compact routing. A key target is to provide a software that is able to handle simulated networks consisting of a minimum of ten thousands nodes. This requirement imposes a careful analysis of the data structures that will be used in the network model as well as on the granularity and time management of the simulation model. This joint research project is conducted by Alcatel-Lucent Bell, Universit de Bordeaux (LaBri) and INRIA labs at Sophia Antipolis (Mascotte project). It is funded by Alcatel-Lucent Bell. The design/development philosophy of DRMSIM is make to compromise on the quality of the code written so that extensibility and reusability are maximized.

## 1 Package overview

The official webpage for the DRMSIM project is:

<http://www-sop.inria.fr/mascotte/projets/DCR/>

The software is the union of several projects, including MASCSIM, a discrete-event simulation platform:

<http://www-sop.inria.fr/members/Luc.Hogier/mascsim/>

A graph library:

<http://www-sop.inria.fr/members/Luc.Hogier/dipergrafs/>

A bridge between Java and UNIX:

<http://www-sop.inria.fr/members/Luc.Hogier/java4unix/>

And a graph render library:

<http://graphstream.sourceforge.net/>

## 2 Installation

### 2.1 Prerequisite

In order to use DRMSIM, you must have an UNIX operating system. Of course, Linux is fine. The choice for the Java Virtual Machine is more tricky. More

UNIX computers around actually are Linux boxes. Linux comes with a Java Virtual Machine: GCJ<sup>1</sup>.

According to its official website, “GCJ is a portable, optimizing, ahead-of-time compiler for the Java Programming Language. It can compile Java source code to Java bytecode (class files) or directly to native machine code, and Java bytecode to native machine code”. According to most users, it does not work. DRMSIM proved unoperational when running atop GCJ. If DRMSIM (or its installation program) does not work on your machine, checking the JVM is the first thing to do.

DRMSIM turns out to work fine with both Sun’s official JVM and OpenJDK.<sup>2</sup>

DRMSIM was successfully tested on the following platforms, using Sun’s JDK version 1.6 or OpenJDK:

- Ubuntu 9.04;
- Fedora 10;
- CentOS 5.4;

We still have not tested it under Mac OS X.

## 2.2 Procedure

In order to install DRMSIM on your computer, you need to run the following command:

```
wget http://www-sop.inria.fr/mascotte/projets/DCR/releases/install-drmsim.sh
bash install-drmsim.sh
```

This will query the official website for the last version of the code and download it. Next it will copy the corresponding files into your home directory, as follows.

## 2.3 Files installed

The installer creates 3 directories in your \$HOME, if they do not yet exist:

**\$HOME/.drmsim** contains the jar files of the last version;

```
.drmsim
.drmsim/jars
.drmsim/jars/java4unix-2010.01.20.11.58.52.jar
.drmsim/jars/dipergrafs-2010.01.20.11.58.52.jar
.drmsim/jars/tools-2010.01.04.16.38.41.jar
.drmsim/jars/graphstream.jar
```

---

<sup>1</sup><http://gcc.gnu.org/java/>

<sup>2</sup>OpenJDK is the effort by Sun Microsystems to release a fully buildable Java Development Kit based completely on free and open source code.

```
.drmsim/jars/alusim-brite-2010.01.04.16.38.41.jar
.drmsim/jars/mascsim-2010.01.20.11.58.52.jar
.drmsim/jars/up-2010.01.04.16.38.41.jar
.drmsim/jars/inria.drmsim-2010.01.20.11.58.52.jar
.drmsim/current_version.txt
```

**\$HOME/lib** contains a set of symbolic links to the jar in **\$HOME/.drmsim**;

```
lib/java4unix-2010.01.20.11.58.52.jar
lib/dipergrafs-2010.01.20.11.58.52.jar
lib/toools-2010.01.04.16.38.41.jar
lib/graphstream.jar
lib/alusim-brite-2010.01.04.16.38.41.jar
lib/mascsim-2010.01.20.11.58.52.jar
lib/up-2010.01.04.16.38.41.jar
lib/inria.drmsim-2010.01.20.11.58.52.jar
```

**\$HOME/bin** contains the executable programs coming with DRMSIM.

```
bin/mascsim_cache_clear
bin/mascsim_result_plotter
bin/drmsim_bench_protocol
bin/mascsim_campaign
bin/mascsim_develop_config
bin/mascsim_cache_publish
bin/drmsim_scan_available_topology_generators
bin/mascsim_result_config_extractor
bin/drmsim_cleaner
bin/mascsim_result_info
bin/mascsim_compute_simulation
bin/mascsim_cache_info
bin/mascsim_rmi_lan_scanner
bin/mascsim_determine_input_filename
bin/mascsim_result_values_extractor
bin/mascsim_cache_query
bin/mascsim_result_merger
```

Those files are simple bash scripts that invoke the java virtual machine.

### 3 DRMSim's commands

Just like any UNIX application, DRMSIM appears to the user as a set of commands.

The following commands are available:

**drmsim\_bench\_protocol bench**

**drmsim\_cleaner** ALU file cleaner

**drmsim\_network\_printer** ALU compact routing simulator: performs simulation of routing protocols on internet-like dynamic networks

**drmsim\_scan\_available\_metrics** scans for available class `inria.mascsim.metrics.Metric`

**drmsim\_scan\_available\_topology\_generators** scans for available class `inria.dipergrafs.topology.TopologyGenerator`

**mascsim\_cache\_clear** Clears the result cache. This operation is applied to the local cache (stored on the local disk)

**mascsim\_cache\_info** Prints info on the local cache (e.g. number of results stored)

**mascsim\_cache\_publish** Publish the given result file to a cache. This can be applied either to the local cache or to the web cache

**mascsim\_cache\_query** Query a cache and indicate whether the given result number identifies a result that is stored in the cache. This can be applied to both local and web cache.

**mascsim\_campaign** Performs a simulation campaign. In practise, it generates a set of independant simulation jobs and execute them according to a given execution strategy.

**mascsim\_compute\_simulation** Compute the simulation job described in the given file.

**mascsim\_determine\_input\_filename** Determines the file name that the given input should have. This name should match the name of the file

**mascsim\_develop\_config** Develops the input configuration. This configuration must defined a set of parameters than will be deployed.

**mascsim\_result\_config\_extractor** Restore the `.mascsim` file(s) taht were used to compute the given result file(s).

**mascsim\_result\_info** Prints information on the content of given result files.

**mascsim\_result\_merger** Compute the average of a set of given result files. The result is stored in a special result file.

**mascsim\_result\_plotter** Plots the content of result files. The output is a PDF file.

**mascsim\_result\_values\_extractor** Restore the `.mascsim` file(s) that were used to compute the given result file(s)

**mascsim\_rmi\_lan\_scanner** Scans the local area network for RMI-Mascsim cooperative hosts.

## 4 Software architecture

In this section, we describe the software architecture of DRMSIM and explain the choices that were made so as to be able to simulate large-scale networks and maintain modularity of the code.

### 4.1 Module dependencies

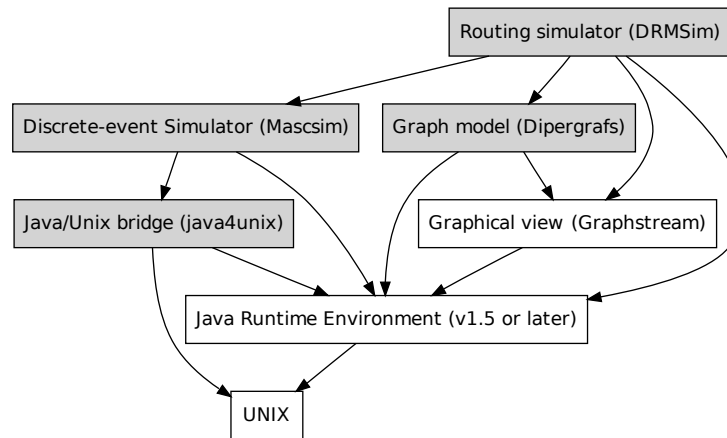


Figure 1: The dependencies of modules

Figure 4.1 represents the DRMSIM modules and their dependencies. DRMSIM relies, through the Java4unix framework, on UNIX facilities. DRMSIM relies also on DIPERGRAFS [?] for the network and topology models and on MASCSIM [?] for the discrete-event simulation engine. The Graphstream module can be used to obtain a view of the routing scheme. In practice, each of these modules comes as a JAR file that must be included in the JVM classpath. The installation procedure of DRMSIM automates the process of downloading the modules and installing them in the UNIX user account. More details concerning these modules are given in the following.

#### 4.1.1 Mascsim

MASCSIM operates at the level of the simulation, it models and implements a distributed discrete-event simulation infrastructure. It defines the following entities: the simulation, the system, the events, the metrics, the measures, etc. At this level, the network is not considered. MASCSIM also provides a set of tools



for dealing with user simulations, with simulation campaigns, result processing, and graph plotting.

#### 4.1.2 Dipergafs

DIPERGRAFS defines the concept and efficient implementations of "network" that is used in DRMSIM. It also provide a set of algorithms and input/output mechanisms for dealing with the corresponding data structure (a directed hypergraph).

#### 4.1.3 Java4unix

Java4unix provides the "glue" between the Java implementation of DRMSIM and the UNIX operating system. Specific issues related to the execution of Java applications are solved by Java4unix by bridging specific "executable" classes by *bash* scripts copied in the \$HOME/bin directory of the local user. Java4unix also provides the capabilities required for the installation and updates of DRMSIM.

#### 4.1.4 Graphstream

Graphstream is a graph rendering toolkit developed by Dutot and al. at the Le Havre University. Graphstream is used in DRMSIM as a "routing monitor". It allows the user to observe the behaviour of the simulated routing scheme at each step of its execution.

### 4.2 Object-oriented model

DRMSIM is an object-oriented application designed as a set of independent software components. In particular, it uses the DRMSIM discrete-event simulation engine (itself derived from Madhoc[?], a mobile wireless network simulator), and the DIPERGRAFS library. Many components take part in the workflow of the MASCSIM simulation engine: a simulation campaign is a set of execution for individual simulation computations. Once all individual simulation have been completed, the simulation campaign computes statistically confident results out of all results collected. The way individual simulation are executed is called an "execution strategy". We have developed three different strategies. First the "sequential execution" takes the set of individual simulation jobs in a sequence and process them all in a row. Second, the "multi-thread strategy" instantiates a predefined number of threads and uses them to execute the jobs in parallel. This technique allows to take advantage of multi-core workstations. Last, the "distributed strategy", which is still under tests, use the RMI technology to distribute the individual simulation jobs across a set of cooperating workstations.

### 4.3 Measurement model

Taking measures along a discrete-event simulation (see Figure 4.3) can be performed in a number of ways. The most simple way consists in considering that

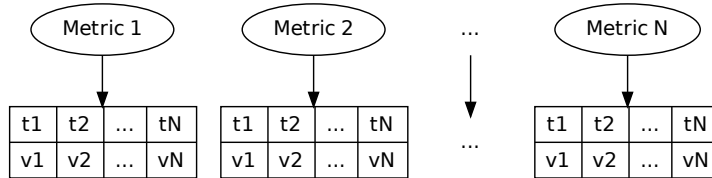


Figure 2: Measurement Approach

all events will change the state of the system so as it is worthwhile to take new measures for all the defined metrics. Nevertheless, this approach fails to consider that a measure on the state of a system is not always obtainable in constant time complexity. Taking measure often requires additional –possibly time consuming– computations. Another approach consists thus in taking only the metrics that might have been affected after an event executes. This approach reduces the computational overload presented herein before but does not solve it: the event defines a set of metrics which are potentially affected by its execution but there may still exist metrics which were actually not affected but for which new measures would still be computed.

For this purpose, DRMSIM uses a different approach which consists in taking a measure as soon as the corresponding metric has actually been affected. This approach effectively reduces additional computations to the minimum. The simulation code becomes targeted to a specific simulation objective: what has to be observed is hard-coded in the events themselves. In practice, when an event performs an action (of interest) that affects the state of the system, it can subsequently compute a new value for the corresponding metric and store this value in the "measure history". A drawback of this approach is that it introduces a dependency between the simulation and the measurement code: if one decides to carry on the same simulation but look at different metrics, the original code will have to be duplicated and modified. Also, specifying a new metric and its associate measure imposes a modification of the simulation code. Nevertheless, in the context of DRMSIM, where priority is on performance for the sake of simulation on large networks, this efficient but less flexible approach was opted for.

As explained before, the main goal of DRMSIM is to quantitatively evaluate some of the main performance metrics of the routing models and especially those related to scalability and stability. The following performance metrics have been in DRMSIM (other can easily complement this set):

- *Stretch*: the stretch (of a routing scheme) is defined as the ratio over all source-destination pairs between the routing scheme path length and the minimum path length (actual distance) for the same source-destination

pair. Intuitively, the stretch is a quality measure of the paths length increase as produced by a routing scheme compared to shortest path lengths. Shortest path routing either AS-path length based (path vector routing) or cost metric based (link-state routing) are stretch 1. This metric is interesting to measure since compact routing schemes usually, since producing reduced routing tables, are not always able to choose the minimum path for a given destination but on the other hand, the routing scheme should favour selection / computation of routes whose stretch remains closer to 1.

- *Routing table size*: it is calculated using the size of a single entry and the number of entries in the routing tables (RT). RT size is directly related to routing system scalability because the less memory a router needs to store its entries, the more scalable the routing system would be. Shortest-path routing schemes are incompressible, i.e, for all nodes in for all graphs, their lower bound equal their upper bound i.e.  $O(n \log n)$  bits are required to store their RT entries [?, ?]. Note, when designing a routing scheme, there is a fundamental trade-off between the stretch of a routing scheme and the size of the RT it produces.
- *Communication cost*: the dynamic nature of the routing protocol such as those currently deployed over the Internet allows each router to be kept up to date wrt to non-local topological changes (resulting from topological failures, addition/withdraw of routes and ASs). The latter information is exchanged between routers by means of routing information updates (each router timely distributes to its own peers following specific selection criteria the routing information received from other peers). Communication cost is defined as the number of routing updated messages that needs to be exchanged between routers to converge after a topology change. Recently, [?] showed that the communication cost lower bound for scale-free graphs is at best linear up to logarithmic factors. The number of routing updates may change according to the advertisement technique (time or event-driven).
- *Computational complexity*: routing updates processing results in recalculation of the RT entries and can lead to convergence delay, and instabilities but also processing overhead. The computational complexity is defined as the number of cycles needed to recompute a RT entry for a given destination and insert it as part of the RT (or replace/remove an existing entry in the RT).

To validate and test DRMSIM, we have implemented many different routing schemes, from simple ones such as source routing to more complex ones such as BGP. We describe these algorithms in the next Section.

## 4.4 Graphical monitoring of the routing process

DRMSIM comes with a set of command-line tools which allow the execution of simulation campaigns and the extraction of results. Also, for the purpose of monitoring, which is of paramount importance when prototyping distributed applications, an aircraft view of the network is also provided.

## 5 Introduction to routing

The Internet is organized in a number of AS (Autonomous System). An Autonomous System (AS) is an administrative entity which consists in a collection of connected IP routing prefixes under the control of one or more network operators that presents a common, clearly defined routing policy to the Internet. AS inner and outer routing obey to different rules. Two classes of protocols are then defined: IGP (Interior Gateway Protocol) and AGP (Exterior Gateway Protocol).

### 5.1 Routing in the current Internet

Routing over the current Internet is achieved by the coupling of hierarchical addressing and a set of routing protocols. Most commonly found include:

**BGP** (Border Gateway Protocol) is an EGP;

**OSPF** (Open Shortest Path First) is complex and resource demanding. OSPF is an IGP;

**IS-IS** (Interior System to Interior System) is the only OSI IGP;

**IGRP** (Interior Gateway Routing Protocol) is proprietary of Cisco Systems. The large deployment of Cisco routers make it a largely used protocol. IGRP is an IGP.

**RIP** (Routing Information Protocol). RIP is an IGP;

**RIP2** add VLSM and authentication functionalities to RIP.

**EIGRP** (Enhanced IGRP) is a Cisco proprietary routing protocol loosely based on their original IGRP. EIGRP is an advanced distance-vector routing protocol, with optimizations to minimize both the routing instability incurred after topology changes, as well as the use of bandwidth and processing power in the router. Routers that support EIGRP will automatically redistribute route information to IGRP neighbors by converting the 32 bit EIGRP metric to the 24 bit IGRP metric. Most of the routing optimizations are based on the Diffusing Update Algorithm (DUAL) work from SRI, which guarantees loop-free operation and provides a mechanism for fast convergence.



## 5.2 Classification of routing protocols


### 5.2.1 Distance-vector and link-state protocols

There are two major types of routing protocols, some with variants: link-state routing protocols and (path vector protocols) distance-vector routing protocols.

A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically and, in some cases, when a change is detected in the topology of a network. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

The link-state protocol is performed by every switching node in the network (i.e. nodes that are prepared to forward packets; in the Internet, these are called routers). The basic concept of link-state routing is that every node constructs a map of the connectivity of the network, in the form of a graph showing which nodes are connected to which other nodes. Each node then independently calculates the best next hop from it to every possible destination in the network. The collection of best next hops forms the node's routing table.

### 5.2.2 AS-level or router-level routing

In the specific context of the Internet, routing protocols can be classified according to their usage domain: whether they are used to route packets within AS (Autonomous System) or between AS. 

## 6 Simulation models

Before making a large scale simulation model there are many issues that should be addressed and the most important are:

- How to generate a realistic Internet topology?
- How to simulate policy configuration?
- What kind of metrics to gather to facilitate reasoning about both control plane and data plane behavior?
- How to simulate protocol procedures for a very large routing topology while making efficient use of resources such as memory?
- How to identify meaningful and realistic failure/update scenarios that reflect reality?

## 6.1 Network model

The network model of DRMSIM relies on the concept of directed hypergraph[4], which adequately matches the structure of actual interconnection networks. More precisely directed hyper graphs are able to represent the whole set of peculiarities of practical networks, including bus topologies, asymmetric links, redundant connections. . .

In simple terms, a directed hypergraph can be seen as a digraph on  $P(E)$ : it is a directed graph that connect sets of nodes. Since they define no constraints on the content of the set of nodes, all connection configurations are possible.

### 6.1.1 On memory consumption and computational performance

DRMSIM graph model makes use coupled incidence lists. Every graph object (resp. node and edge) is assigned an integer ID which is used to access the set of incident objects (resp. edges and nodes). Four incidence lists allow to get, for a given node, the sets of incoming and outgoing edges and, for a given edge, the sets of source and destination nodes.

This technic allows fast graph navigation, since jumping from node to node consists in two array indexing. Also, it is efficient in memory consumption if the sets of IDs are dense.

## 6.2 Routing protocols

DRMSIM routing model allows unicast and multicast routing. It defines a routing protocol as a function  $(L, R)route(r, l, m)$  where:

**m** is the incoming message;


**r** is the router the message  $m$  came from;

**l** is the link the router  $r$  used to forward the message  $m$ ;

**L** is a set of outgoing links where the message will be sent;

**R** is a set of relay nodes to which the message will be sent. In practise all nodes that are destination nodes of links in  $L$  will receive the message but only those in  $R$  are allowed to relay it.

### 6.2.1 Standard set

 DRMSIM features a set of routing protocol that were developed in the context of the modeling/testing of the simulation engine.

**source routing** the source locally computes the path the message will follow and embeds this path into the message header. Every forwarding node will hence know where to route the message. This is a centralized protocol.

**QoS-based routing** is a polymorph routing protocol that choose the actual routing strategy according to QoS information stored in the header of the message.

**blind broadcasting routing** upon receiving a message that is not target to itself, the router forwards the message on all outgoing links. It is a localized protocol.

**shortest path routing** upon receiving a message that is not target to itself, the router forwards the message on the links that belong to the shortest paths to the destinations. This routing protocol should be used as a reference for the evaluation of other protocols. It relies on global network information.

**closest address routing** forwards messages to nodes whose the ID is the closest to messages destination ID. This routing strategy requires a structured addressing of nodes. It is used in Distributed HashTables (DHTs).

**random routing** upon receiving a message that is not target to itself, the router forwards the message using a randomly selected outgoing link, or to a randomly selected relay. It is a localized protocol.


**round robin routing** links are used in a sequential order. Although this form routing cannot be used for multi-hop routing, it can be used for load balancing purposes.

These protocols can be used to simulate denial of service:

**no routing** do not forward any message. Previous hop router is not informed of the denial of service.

**back to previous hop router routing** messages are not altered and sent back to router at the previous hop (which will have to select another next-hop relay to achieve relay). This is the same as “no routing” except that the previous hop router gets informed of the denial of service by receiving the message it just forwarded.

### 6.2.2 BGP

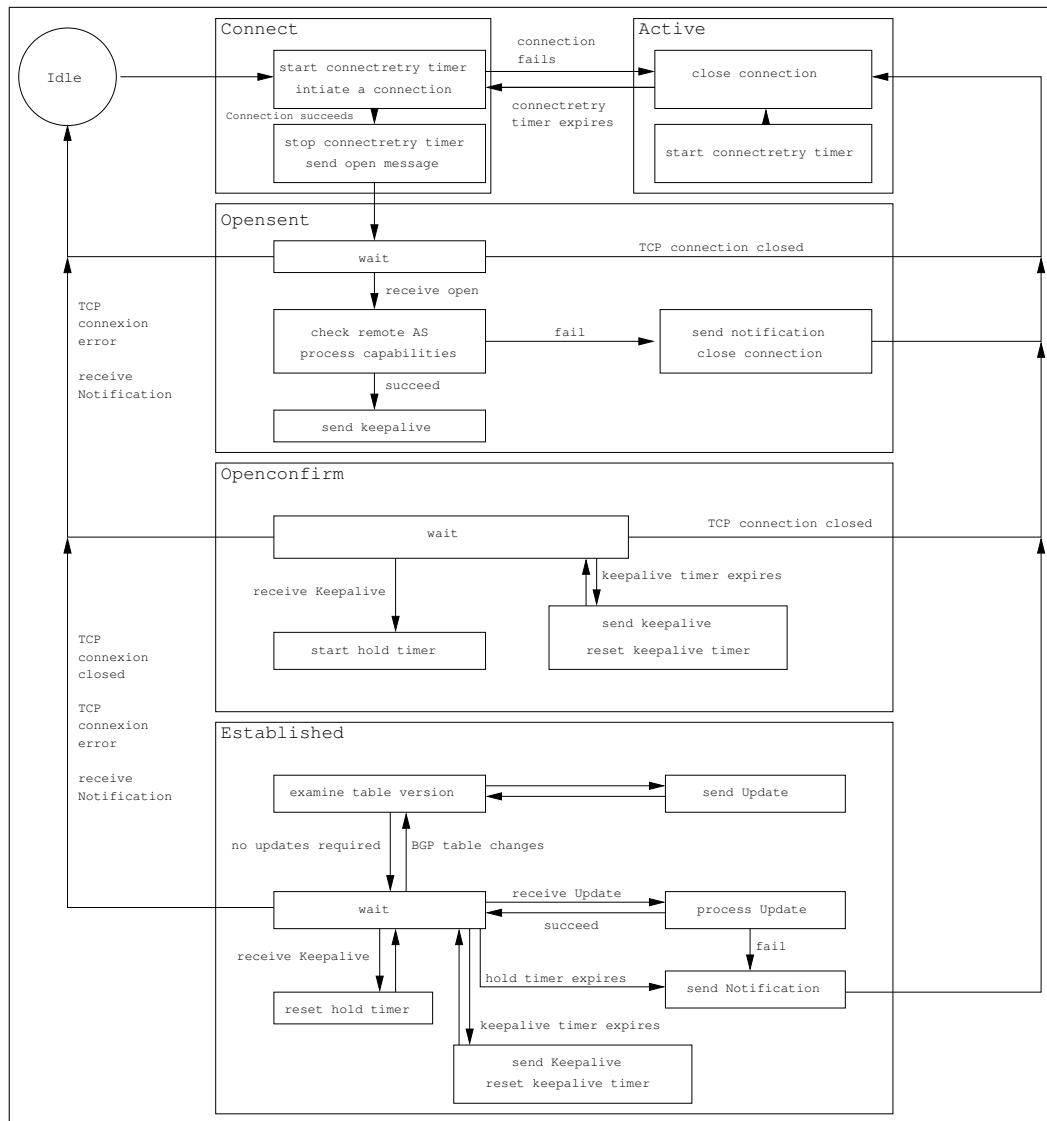
 The BGP protocol is used to communicate information about networks, defined by IP address blocks, between entities known as autonomous systems (AS), so that one part of the Internet knows how to reach another part. The exchange of network information is done by setting up a communication session between bordering autonomous systems. For reliable delivery of information, a TCP-based communication session is set up between bordering autonomous systems using TCP listening port number 179. This communication session is required to stay connected, which is used by both sides to periodically exchange and update information. When this TCP connection breaks for some reason, each side is required to stop using information it has learned from the other side.

In other words, the TCP session serves as a virtual link between two neighboring autonomous systems, and the lack of communication means that this virtual link is down. Now imagine that each autonomous system is a virtual supernode; then the entire Internet can be thought of as a graph connecting virtual supernodes by virtual links.

The first model (The closest to reality) implemented in the simulator corresponds to the latter, since every node is considered as an AS. This means that it can simulate the external BGP communication (EBGP) in a an optimal way, but not the internal one (IBGP). In a router where BGP is running, many types of Routing Information Bases are stored. First, there is the Loc-RIB which contains all the paths known by the router and that are actually used in the forwarding process; in addition to this, a BGP router has an Adj-RIB-IN and an ADJ-RIB-OUT for every adjacent node (which is linked to by a virtual TCP connection). The aim of these RIBs is to provide a neighbor-based filtering for both incoming and outgoing advertised routes. Since the security policies are not part of the simulation targets, only the loc-RIB was implemented. It is an aggregation of many routes taking into consideration the most important attributes (the path and its destination network) while leaving the flexibility for adding new ones according to simulation needs. However the implemented BGP routing model contains all the steps needed to establish, maintain, retrieve or to close a BGP session. Thus, the majority of the events that may occur in the latter are handled.

Then we tried to simplify this model by reducing the number of events and assuming that a BGP session has only two possible states. It is either IDLE or ESTABLISHED. this only has an impact on the establishment of the sessions. But after, there is exactly no difference between the two models. In term of performance, the initial phase in every simulation will end faster if we consider similar scenarios. Eventhough this simplification seemed to be a better solution, it was still not enough to perform a simulation on a Internet-like graph. The idea then was to find another way to simplify more the model without creating a lack in the information provided by the simulation. Noticing that there was a redondacy between the information stored in the RIB and the one stored in the forwarding table, we decided to forward the packets using the RIB and then remove the forwarding table. this made our model simpler and decreased the time and the amount of memory needed by a simulation.





The BGP events implemented in every package:

BGP Events	bgp.full	bgp.simple	bgp.simpler
Local administrator starts BGP connection	yes	yes	yes
Local administrator stops BGP connection	yes	yes	yes
Indication that ConnectRetryTimer has just expired	yes	no	no
Indication that HoldTimer has just expired	yes	yes	yes
Indication that KeepAliveTimer has just expired	yes	yes	yes
Indication that the connection is successfully set up	yes	no	no
Confirmation of the initiation of a connection	yes	no	no
Receiving a connection failure indication	yes	no	no
Receiving a valid OPEN message	yes	no	no
Receiving a BGP message with invalide header	yes	no	no
Detecting an error in the received OPEN message	yes	no	no
Receiving an invalide NOTIFICATION message	yes	no	no
Indication that a KEEPALIVE message is received	yes	no	no
Indication that a valide UPDATE message is received	yes	yes	yes
Indication that a invalide UPDATE message is received	yes	yes	yes

**Other BGP simulators and their limits:** BGP simulators are not rare. Several projects have developed some abstract model of the protocol to check its properties. For instance, SSF.OS.BGP4 (implementation of the BGP protocol on SSFNet simulator) was developped to study BGP stability behavior in different network topologies, when confronted to various events. The simulator contains a lot of features from the protocol specification, and even includes implementation-specific aspects. However, the model did not incorporate user

traffics effects on BGP. Moreover, the simulation model is too rich: the memory and CPU consumption required makes it hardly usable on large topologies. Unlike the former simulator, C-BGP [10] was designed to work on topologies of several thousands of ASes. This simulator, built as an efficient decision process simulator, allow experimentations on BGP protocol and attributes. However, it lacks temporal dimension, as the simulator does not implement BGP timers, nor messages delays (link crossing delay, router CPU waiting). Another large scale BGP simulator was proposed in [5]. Its aim was to run BGP simulations on several thousands nodes topologies by keeping the temporal delays introduced by the BGP timers. But, like all the previous simulators, this one uses Logical AS topologies, ignoring IBGP interactions and the effects of AS-internal event on the convergence. Same goes for the RouteSim BGP simulator [9]. Although the model was abstracted enough to allow large topologies simulation, it lacked realistic temporal delay model, not allowing the observation of user traffic variation consequences on BGP signalisation. Another way in BGP simulator conception was explored by the creators of BGP++. By integrating a real BGP implementation into a simulator, the simulations gained a realness never obtained before, but the low level of abstraction made memory and CPU consumption too large to allow large topologies simulation.

### 6.2.3 RIP

The Routing Information Protocol, or RIP, as it is more commonly called, is one of the most enduring of all routing protocols. RIP is also one of the more easily confused protocols because a variety of RIP-like routing protocols proliferated, some of which even used the same name! RIP and the myriad RIP-like protocols were based on the same set of algorithms that use distance vectors to mathematically compare routes to identify the best path to any given destination address. These algorithms emerged from academic research that dates back to 1957.

## 6.3 Metrics

Definition/implementation the metrics for the evaluation of routing protocols

- the stretch of a routing path is the ratio between the length of this routing path and the length of the shortest path. It takes its values in  $[1, +\infty]$ . The closer to 1, the better.
- size (in bit) of the routing tables. The lower the better;
- size (in bit) of the message headers. The lower the better;
- the amount of traffic that is generated for inter-changing topology informations;
- number of entries in the routing tables. The lower the better;

- observed duration to go back to a steady state, after a sudden change of the topology. The lower the better;

Also the algorithmic complexity of the routing function is an important metric. When the number of entries in the forwarding table is large, the routing routine may fail to provide routing information efficiently.

Also the algorithmic comprehensiveness is to be looked at.

In order to have more details about the behavior of a protocol, we are investigating other metrics that we would like to add without decreasing the simulator performance and those are mainly:

- the Number of *stable* and *transient* updates generated during convergence: In fact, during convergence, some candidate routes may become transient. Upon occurrence of an event  $E$  that necessitates reconvergence from the current state, at each router, we say that a candidate route  $r$  to a prefix  $p$  is transient if  $E$  causes the AS Path of  $r$  to be no longer valid. A route with AS Path  $P = A_0A_1A_2...A_n$  for a destination  $d$  at node  $A_{n+1}$  with  $A_0$  being the origin AS is said to be *transient* if there exists some  $A_i$  ( $0 \leq i \leq n - 1$ ) such that the AS Path of the route at  $A_i$  is not the same as  $A_0A_1A_2...A_i$ . Therefore, during convergence, candidate routes may be either stable or transient. Note that in standard BGP procedures, for instance, it is not possible for a node to figure out if any of its candidate paths is transient.
- data plane convergence time: A node or a set of nodes has reached data plane convergence when the node or set of nodes has a stable forwarding path to the destination implied by the unvarying next hop at the node and at each of the downstream nodes. Data plane convergence is equivalent to forwarding convergence. Note that an AS path at a node can change without causing a change of next hop at a node, but next hop will not change unless there is a route (AS path) change. This leads to the following observation. Protocol convergence implies data plane convergence, but the converse is not true. Hence, for a prefix  $m$ ,  $0 < DP_m \leq PC_m$ , where  $DP_m$  and  $PC_m$  are data plane and protocol where convergence times for the prefix  $m$ , respectively.
- average downtime is also proposed as a metric to capture the notion of potential disruption of reachability to a destination.

## 6.4 Topology generators

Most generators produce random networks [11].

The problem of creating a topology can be expressed as “creating or deleting a number of edges so as the resulting graphs will have the required properties”. This is how DRMSIM model topology generators: they are a function  $f(g, C)$  where  $g$  is a graph (that may already feature some edges) and  $C$  is a set of constraints. DRMSIM implements the topologies described in the following,

assuming that all of these generators are composeable (in the sense that they can be invoked in a sequence on the same graph).

#### 6.4.1 Standard set

DRMSIM implements a number of basic topology generator which allow to create networks whose the nodes are connected by:

**directed chain** iterate on the set of nodes and connect every node towards its predecessor.

**directed grid** create a sequence of directed chains and connect every node in a given directed chain with the node at the same index in the predecessor directed chain.

**symmetric** makes all links to be symmetric. This does not generate any new link.

**asymmetric** create a new link in the opposite direction (asymmetric link). This doubles the number of links in the network.

#### 6.4.2 Randomized set

DRMSIM implements the following randomized topology generators:

**tree** creates a random tree;

#### 6.4.3 Tier 1/Tier 2/Tier T3



On the Internet, nodes can be classified according to their structural role in the network. Three categories have been identified:

- T1 nodes which belong to the international operators of the internet (like AT&T, Sprint, etc). These nodes form Wide-Area networks.
- T2 nodes which usually belong to national organizations like university networks or public access provider (PCCW Global, France Tlcom, Tiscali, etc). These nodes form the Metropolitan-Area networks.
- T3 nodes corresponds to nodes in Local-Area networks (end users).

The topology of the network depends on the type of nodes it connect. The network of T1s is the heart of the internet. It is very dense. T2 nodes are connected to T1 ones through very fast links. Generally a T2 is connected to two T1 nodes for redundancy purposes. In the vast majority of the cases a T3 node is connected to one single T2 one.

Another generator that implements models trying to imitate the structure of the Internet is Tiers [2].

#### 6.4.4 Inet


Inet[7] currently at version 3.0, is an Autonomous System (AS) level Internet topology generator. Inet cannot generate network consisting of less than 3037 nodes. As a consequence, Inet output cannot be visualized. The `dipergrafs` project provides a front-end to Inet through the class `inria.dipergrafs.topology.InetTopologyGenerator`. In order for the topology generation to work, the Inet application must be installed on the computer and the `inet` command should be accessible through the `PATH` environment variable. The path to the `inet` command can also be specified in the configuration file by filling the `path-to-inet` entry.

#### 6.4.5 Brite

BRITE [8] is the precursor to the universal generation tool. It implements a single generation model that has several degrees of freedom with respect to how the nodes are placed in the plane and the properties of the interconnection method to be used. Under certain configuration of the parameters, BRITE 1.0 generation model is equivalent to Waxman. Under other configuration of parameters, BRITE 1.0 implements the Barabasi-Albert model proposed in [2] in which a network grows incrementally and the nodes interconnect with preference towards higher degree nodes. The `dipergrafs` project provides a bridge to Inet through the class `inria.dipergrafs.topology.BriteTopologyGenerator`.


To achieve this result, `Dipergrafs` exploits a modified copy of the source code of Brite. Indeed Brite was designed in such a way that it exploit the text-based I/O streams. Modification of the source code consists in bypassing this interface in order to directly interact with the Brite's framework.

#### 6.4.6 GT-ITM


 One of the most popular generators available is [1]. The following text is taken from [8]: The main characteristic of GT-ITM is that it provides the Transit-Stub (TS) model, which focuses on reproducing the hierarchical structure of the topology of the Internet. In the TS model, a connected random graph is first generated (e.g. using the Waxman method or a variant of it). Each node in that graph represents an entire Transit domain. Each transit domain node is expanded to form another connected random graph, representing the backbone topology of that transit domain. Next, for each node in each transit domain, a number of random graphs are generated representing Stub domains that are attached to that node. Finally, some extra connectivity is added, in the form of “back-door” links between pairs of nodes, where a pair of nodes consists of a node from a transit domain and another from a stub domain, or one node from each of two different stub domains. GT-ITM also includes about five flavors of flat random graphs.

We sent a mail to the authors to get a version that works on Linux. The source code was designed for SunOS and cannot get compiled on Linux Ubuntu.

#### 6.4.7 CAIDA

 The Cooperative Association for Internet Data Analysis (CAIDA) provides data about the structure of Internet. The router-level topology map is particularly relevant to us. It can be downloaded at the following URL: [http://www.caida.org/tools/measurement/skitter/router\\_topology/](http://www.caida.org/tools/measurement/skitter/router_topology/)


### 6.5 Dynamics

Dynamics consider maintenance operations on the network infrastructure as well as router failures. End-user mobility is not looked at. 

### 6.6 Traffic model



### 6.7 Time model

 DRMSIM is built atop a general purpose discrete-event simulation engine. In discrete-event simulation, the operation of a system is represented as a chronological sequence of events. Each event occurs at an instant in time and marks a change of state in the system (Stewart Robinson (2004). 'Simulation - The practice of model development and use'. Wiley.) The simulation must keep track of the current simulation time. Time “hops” because events are instantaneous the clock skips to the next event start time as the simulation proceeds.

An event is described by the time at which it occurs and the code that will be used to simulate that event. This code may schedule new event that will occur in the future or cancel existing events.

### 6.8 Granularity

DRMSIM performs packet-level simulations. The basic event that are modelled within DRMSIM are:

- emission of a packet from one node to another node
- reception of a packet from one node

Routing protocols may individually define other events, for their own needs.

## 7 Frequently Asked Questions (FAQs)

### 7.1 What is the input of the simulator?

The input of DRMSIM is an ASCII configuration file. The syntax of this This file is the following:

```
        an entry={one value}
another entry={this one, has two values}
```

The name of this file is given to the command `mascsim_compute_simulation`.

## 7.2 Is it possible to define my own topology?

### 7.3 ...deal with the output

The output of a simulation job is a binary file whose the extension if `.result`. The content of such file can be displayed using the command `mascsim_result_info`.

Since a result file contains a number of metric, each of them defining a list of measure that have been taken along the simulation, it is possible to get plots out of it. The command `mascsim_result_plotter` takes a result file as input and generate a plot (in the form of a PostScript file) for every metric found.

### 7.4 How can I add a new metric

The addition of a new metric can be done by deriving the class `inria.mascsim.metrics.Metric`. Basically doing this implies to define the domain of the metric (define if a given value is an acceptable value for the metric), and to implement the way a new value for the metric can be obtained on the simulated system.

In most of the cases, the user will want to define a metric that is numeric. In this case, there is the need for defing the unit for this metric. A list of of pre-defined units are available as static fields of the `inria.mascsim.metrics.Unit` class. If none of these units suit then new metric, then the user must subclass `inria.mascsim.metrics.Unit` and implement a new unit that meets its particular needs.

### 7.5 How can I generate a GLP graph

A specific topology generator takes care of generating GLP topologies: `inria.dipergrafs.topology.GLPTopol`. The name of this class must be given to the `topology_generator` parameter key for the simulator. In addition, the following configuration must be provided, as it is used for initializing the topology generator:

```
glp_numberOfInitialNodes ;
```

```
glp_numberOfEdgesPerStep ;
```

```
glp_beta ;
```

```
glp_stepProbability .
```



## 7.6 How to implement a new routing protocol

The definition of a new routing protocol can be achieved by deriving the class `inria.alu.routing.AbstractRoutingAlgorithm`. Basically, doing this consists in defining:

- the set of outgoing links that will be used to forward an incoming message;
- the type of header messages need to embed;
- the reactive behavior of the protocol when a network link breaks.

The new routing protocol class finally need to be specified for simulation, using the configuration key `routing_protocol`.

## 7.7 Is there a way to import a new topology (CAIDA, Inet))

DRMSIM defines a number of “special” topology generator whose the aim is to enable the import of topologies generated using external tools like Inet, or available as description files such as the CAIDA maps.

In the specific cas

## 7.8 How to run DRMSim?

The most obvious way to run DRMSIM is to use the `mascsim_compute_simulation` command. It takes as the only parameter it accepts the name of the configuration file which describes the simulation.

### 7.8.1 One to one

In this scenario, a source and a destination are defined at random. Then routing are scheduled in a sequence until one succeeds. Then the simulation stops.

### 7.8.2 All to all

### 7.8.3 DoNothing

Instantiate the system and immediatly stop.

## 7.9 ... monitor a simulation process

In order to enable the monitoring of the simulation all along the simulation, DRMSIM provides a bridge with the GraphStream graph rendering framework.

In order to enable the graphical rendering, the line `system_listeners=inria.alu.views.BasicRoutingView` must be uncommented.

## 8 Implementation notes

DRMSIM is developed and tested under the Java Development Kit version 1.6.0\_12-b04, on a 64 bits Linux box. It is not guaranteed that it will work on different platforms. Also Graphstream have shown to fail on v1.7 virtual machines which makes views based on it unusable with this virtual machine.

### 8.1 Code reuse

A development strategy followed in the DRMSIM project is to maximize the use of existing code/software, hereby to minimize the number of lines of code written from scratch.

#### 8.1.1 Madhoc

Madhoc [6] is a Mobile ad hoc NETWORK (MANET) simulator developed at the Universities of Luxembourg (CSC lab) and Le Havre (LITIS lab). Its primary objective is to enable researchers to investigate broadcasting protocols over MANETs. The code of Madhoc is modular and allowed us to extract a number of reusable components. In particular we will use its time-slicing simulation engine as well as its measurement system.

#### 8.1.2 Dipergafs

The simulation network implementation relies on the Dipergafs [4] framework. Dipergafs is a graph manipulation toolkit which support directed hypergraphs and a large set of algorithms dedicated to the navigations, query, etc.

Although it can be presented as a general-purpose graph library, Dipergafs was developed with the objective to be adequate to network simulation.

#### 8.1.3 GraphStream

GraphStream [3] is a java library developed at the University of Le Havre that handles and display dynamic directed graphs in a very simple way. We use it to see what happens in the simulated routing protocol all along the simulation. Because of computational cost inherent to graph drawing, Graphstream can be used only when the number of vertices is small (two hundred nodes constitute a practical limit).

#### 8.1.4 CoLibA

CoLibA is a framework for the integration of Java-based applications into the UNIX command line environment. In particular, it provides command-line parsing, input/output facilities, deployment, enhanced access to functionalities of the operating system, etc.

### 8.1.5 Ploticus



## 8.2 Invocation

DRMSIM is meant to be invoked from the command line, using the command `alu-simulator`. The simulator expects to find in the current directory a set of files whose the name extension is `.alu`. Each configuration file contains a set of *key* = {*value1,value2,...,valueN*} entries. Example will be provided. These files contain the description of the starting simulation.

### 8.2.1 Drawing a network: `alu-network-printer`

The command `alu-network-printer` reads the configuration files, build the model but do not start the simulation. Instead it produces a files containing the graphical rendering of the network before the simulation starts. The network is rendered by the GraphViz graph drawing application.

### 8.2.2 Running a simulation: `mascsim.compute_simulation`

The command `alu-simulator` runs the simulation whose the model is described in the configuration files. The simulation runs until its termination condition is satisfied.

### 8.2.3 Running a simulation: `alu-scan-available-topology-generators`

The command `alu-scan-available-topology-generators` performs a full scan of the available classes (according to the entries listed in the classpath) a prints a list of the classes that can be used as topology generators.

## 8.3 Results

Once the simulation finished, DRMSIM writes a number of `.alu.metric` files in the current directory. Each of these files correspond to a given metric whose the value changed along the simulation. A `.alu.metric` file contains a sequence of *datevalue* entries which describe the timestamped evolution of the related metric.

## 8.4 Simulation campaigns



## References

- [1] K. Calvert, M. Doar, A. Nexion, and E. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 1997.
- [2] M. Doar. A better model for generating test networks. In *Proceedings of Globecom '96*, Nov. 1996.
- [3] Antoine Dutot, Frédéric Guinand, Damien Olivier, and Yoann Pigné. Graphstream: A tool for bridging the gap between complex systems and dynamic graphs. In *EPNACS: Emergent Properties in Natural and Artificial Complex Systems*, 2007.
- [4] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201, 1993.
- [5] Fang Hao and Pramod Koppol. An internet scale simulation setup for bgp. *SIGCOMM Comput. Commun. Rev.*, 33(3):43–57, 2003.
- [6] Luc Hogie. The Madhoc Simulator. Technical report, Le Havre University, <http://agamemnon.uni.lu/~lhogie/madhoc/>, 2005.
- [7] Cheng Jin, Qian Chen, and Sugih Jamin. Inet: Internet topology generator. Technical Report CSE-TR-433-00, University of Michigan at Ann Arbor, 2000.
- [8] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. Technical report, Computer Science Department at Boston University, Boston, MA, USA, 2000.
- [9] J. Nykvist and L. Carr-Motykova. Simulating convergence properties of bgp. pages 124–129, Oct. 2002.
- [10] Bruno Quoitin. *C-BGP Users Guide*, 2005.
- [11] Bernard M. Waxman. Routing of multipoint connections. pages 347–352, 1991.