# Platform 101

## Getting Started with SimGrid Platforms*

Da SimGrid Team

June 13, 2012

* LaTeX Sources: <scm.gforge.inria.fr:/gitroot/simgrid/propaganda.git>/platform-101

# Outline

# Network Communication Models

**Packet-level simulation** Networking community has standards, many popular open-source projects (NS, GTneTS, OmNet++,. . . )

- full simulation of the whole protocol stack
- complex models $\leadsto$ hard to instantiate
- inherently **slow**
- beware of simplistic packet-level simulation

Along the same lines: Weaver and MsKee, *Are Cycle Accurate Simulations a Waste of Time?*, Proc. of the Workshop on Duplicating, Deconstruction and Debunking, 2008

# Network Communication Models

**Packet-level simulation** Networking community has standards, many popular open-source projects (NS, GTneTS, OmNet++,...)

- full simulation of the whole protocol stack
- complex models $\rightsquigarrow$ hard to instantiate
- inherently **slow**
- beware of simplistic packet-level simulation

> Along the same lines: Weaver and MsKee, *Are Cycle Accurate Simulations a Waste of Time?*, Proc. of the Workshop on Duplicating, Deconstruction and Debunking, 2008

**Delay-based models** The simplest ones...

- communication time = constant delay, statistical distribution, LogP
  $\rightsquigarrow$($\Theta(1)$ footprint and $O(1)$ computation)
- coordinate based systems to account for geographic proximity
  $\rightsquigarrow$($\Theta(N)$ footprint and $O(1)$ computation)

Although very scalable, these models ignore network congestion and typically assume large bissection bandwidth

# Network Communication Models (cont'd)

**Flow-level models** A communication (flow) is simulated as a single entity:

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

# Network Communication Models (cont'd)

**Flow-level models** A communication (flow) is simulated as a single entity:

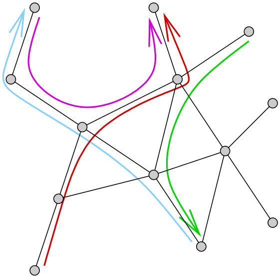
$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

Assume steady-state and **share bandwidth** every time a new flow appears or disappears

Setting a set of flows $\mathcal{F}$ and a set of links $\mathcal{L}$

Constraints For all link $j$: $\displaystyle\sum_{\textbf{if flow i uses link j}} \varrho_i \leqslant C_j$



# Network Communication Models (cont'd)

**Flow-level models** A communication (flow) is simulated as a single entity:

$$T_{i,j}(S) = L_{i,j} + S/B_{i,j}, \text{ where } \begin{cases} S & \text{message size} \\ L_{i,j} & \text{latency between } i \text{ and } j \\ B_{i,j} & \text{bandwidth between } i \text{ and } j \end{cases}$$

Estimating $B_{i,j}$ requires to account for interactions with other flows

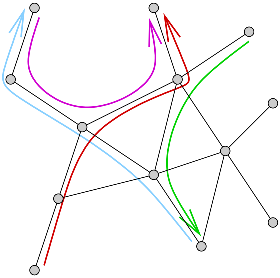
Assume steady-state and **share bandwidth** every time a new flow appears or disappears

Setting a set of flows $\mathcal{F}$ and a set of links $\mathcal{L}$

Constraints For all link $j$: $\displaystyle\sum_{\textbf{if flow i uses link j}} \varrho_i \leqslant C_j$

Objective function

- Max-Min $\max(\min(\varrho_i))$
- or other fancy objectives e.g., Reno $\sim \max(\sum \log(\varrho_i))$



# Wrap up on flow-level models

Such **fluid models can account** for TCP key characteristics

- slow-start
- flow-control limitation
- RTT-unfairness
- cross traffic interference

They are a very reasonable approximation for most LSDC systems

Yet, many people think they are too complex to scale.

Let's prove them wrong! ☺

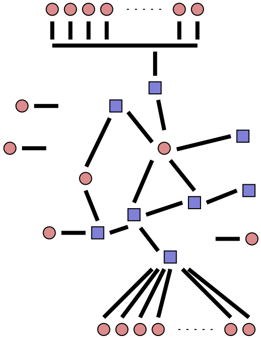
# How to achieve scalability

## Platform description

Main issues with topology

- description size, expressiveness
- memory footprint
- computation time

*N* nodes and *E* links



| Representation | Input | Footprint | Parsing | Lookup |
|---|---|---|---|---|
| | | | | |

# How to achieve scalability

## Platform description

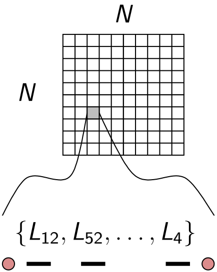
Main issues with topology

- description size, expressiveness
- memory footprint
- computation time

Classical network representation

1. Flat representation
   5000 hosts doesn't fit in 4Gb!

$N$ nodes and $E$ links




| Representation | Input | Footprint | Parsing | Lookup |
|---|---|---|---|---|
| Flat | $N^2$ | $N^2$ | $N^2$ | 1 |

# How to achieve scalability

## Platform description

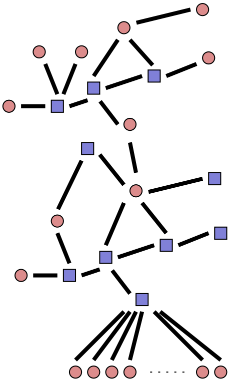
Main issues with topology

- description size, expressiveness
- memory footprint
- computation time

Classical network representation

1. Flat representation
   5000 hosts doesn't fit in 4Gb!
2. Graph representation assuming shortest path routing

*N* nodes and *E* links



| Representation | Input | Footprint | Parsing | Lookup |
|---|---|---|---|---|
| Dijsktra | $N + E$ | $E + N \log N$ | $N + E$ | $E + N \log N$ |
| Floyd | $N + E$ | $N^2$ | $N^3$ | 1 |

# How to achieve scalability

## Platform description

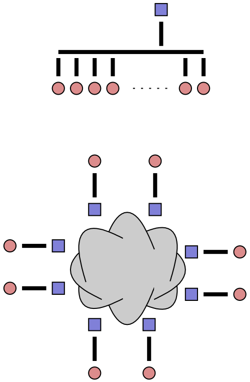
Main issues with topology

- description size, expressiveness
- memory footprint
- computation time

Classical network representation

1. Flat representation
   5000 hosts doesn't fit in 4Gb!
2. Graph representation assuming shortest path routing
3. Special class of structures (star, cloud, ...)

$N$ nodes and $E$ links

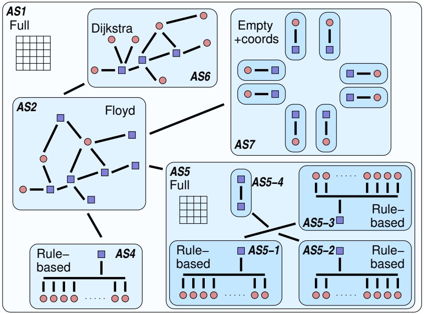


| Representation | Input | Footprint | Parsing | Lookup |
|---|---|---|---|---|
| Star | 1 | $N$ | $N$ | 1 |
| Cloud | $N$ | $N$ | $N$ | 1 |

# Our proposal

Every such representation has drawbacks and advantages
Let's build on the fact that most networks are mostly hierarchical

1. Hierarchical organization in AS
   ⇝ cuts down complexity
   ⇝ recursive routing
2. Efficient representation of classical structures
3. Allow bypass at any level



# Step by step routing

① SRC → $AS_{t_0}$ → ⋯ → $AS_{t_n}$ → $AS_{common}$

DST → $AS_{r_0}$ → ⋯ → $AS_{r_m}$ → $AS_{common}$

② get_route($AS_{t_n}$,$AS_{r_m}$) ⇒ from = $GW_{SRC}$ to = $GW_{DST}$
links = $L_0 \cdots L_k$

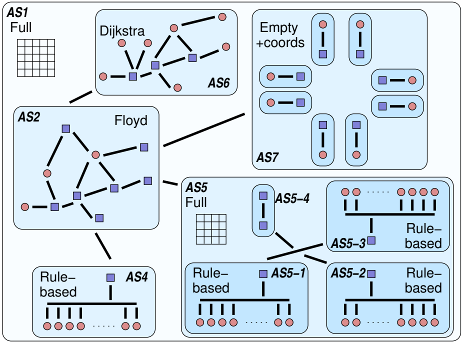
③ SRC ⋯ $GW_{SRC}$ — $L_0$ ⋯ $L_k$ — $GW_{DST}$ ⋯ DST

solve recursively (i) solve recursively (ii)

④ get_route(SRC,$GW_{SRC}$) get_route($GW_{DST}$,DST)

from = SRC to = $GW_{SRC}$
links = $L_{s_0} \cdots L_{s_x}$

from = $GW_{DST}$ to = DST
links = $L_{d_0} \cdots L_{d_z}$



# Outline

- The Network Representation Issue
- XML Based Formalism for Platform Description
  - The XML Approach
  - Specifying Host
  - Specifying inter-host network connections
  - Compacting the XML platform Description
  - Autonomous systems
  - Describe Availabilities in the XML File
  - Setting Properties
  - Examples of use
- Lua Based Formalism For Platform Description
  - Specifying Platform Element
  - Examples of use
  - Deploy Application

# Outline

# XML Platform Description

## XML Platforms

platform.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "surfxml.dtd">
<platform version="2">
<AS id="AS0" routing="Full">
<host name="host1" power="1E8"/>
   <host name="host2" power="1E8"/>
<link name="link1" bandwidth="1E6"
    latency="1E-2" />
<route src="host1" dst="host2">
<link:ctn id="link1"/>
</route>
...
</AS>
</platform>
```

- Introcuced since version 3 (realeased in 2005)
- Separate the Application Scenario
- FleXML based Mechanism
- SAX Approach (Callbacks)

# Outline

- The Network Representation Issue
- XML Based Formalism for Platform Description
  - The XML Approach
  - Specifying Host
  - Specifying inter-host network connections
  - Compacting the XML platform Description
  - Autonomous systems
  - Describe Availabilities in the XML File
  - Setting Properties
  - Examples of use
- Lua Based Formalism For Platform Description
  - Specifying Platform Element
  - Examples of use
  - Deploy Application

# Specifying Host

<Host> Tag

```
<host id="host_id"
   power="500000000"
   [availability_file="host.trace"]
   [state="ON"] />
```

- id : Host Identifier
- power : Host Power in Flops
- availabilitiy_file : Trace file associated
- state : Specify the initial state of Host ON(Up)/OFF(Down)

# Expressing dynamicity

**Adding a trace file**

```
<host id="BOB"
   power="500000000"
   availability_file="bob.trace" />
```

**Example of "bob.trace" file**

```
PERIODICITY 1.0
0.0 1.0
11.0 0.5
20.0 0.8
```

- At time 0 ⇒ the host will deliver 500 Mflop/s
- At time 11.0 ⇒ it will deliver half that is 250 Mflop/s until time 20.0
- At time 20.0 ⇒ it will start delivering 80% of its power, that is 400 Mflops/s
- Last, at time 21 (20.0 plus the periodicity) ⇒ we loop back to the beginning and the host will deliver again 500 Mflops/s

# Outline

- The Network Representation Issue
- XML Based Formalism for Platform Description
  - The XML Approach
  - Specifying Host
  - Specifying inter-host network connections
  - Compacting the XML platform Description
  - Autonomous systems
  - Describe Availabilities in the XML File
  - Setting Properties
  - Examples of use
- Lua Based Formalism For Platform Description
  - Specifying Platform Element
  - Examples of use
  - Deploy Application

# Declaring Network Links

**<link> Tag**

```
<link id="link_id"
  bandwidth="125000000"
  latency="5E-5" [sharing_policy="SHARED"] />
```

- `id` : Link Identifier
- `bandwidth` : Link bandwidth in bytes/s
- `latency` : Link latency in seconds
- `sharing_policy` :
  - `SHARED` (by default) ⇒ if more than one flow go through a link, each get an equal share of the available bandwidth
  - `FATPIPE` ⇒ each flow going through this link will get all available bandwidth, whatever the number of flows(this allows to describe switches or Intern backbones)

# Expressing dynamicity

**Adding a trace file**

```
<link id="LINK1"
   bandwidth="80000000" latency=".0001"
   bandwidth_file="link1.bw"
   latency_file="link1.lat" />
```

**Example of "link1.bw" file**

```
PERIODICITY 12.0
4.0 40000000
8.0 60000000
```

**Example of "link1.lat" file**

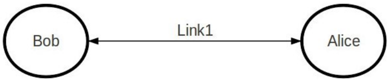
```
PERIODICITY 5.0
1.0 0.001
2.0 0.01
3.0 0.001
```

- It is possible to declare links whose `state`, `bandwidth` or `latency` change over time
- In this case, the `bandwidth` and `latency` are respectively replaced by the `bandwidth_file` and `latency_file` attributes in the corresponding text files

# Declaring routes

## <route> Tag

```
<host id="Bob" power="100000000"/> <host id="Alice" power="500000000"/> <link id="Link1"
bandwidth="125000000" latency="5E-5"/>
<route src="Bob" dest="Alice"> <link_ctn id="Link1"/> </route>
<route src="Alice" dest="Bob"> <link_ctn id="Link1"/> </route>
```



# Expressing multi-hop routes

## Multi-hop routes and asymmetry

```
<host id="BOB" power="100000000"/>
<host id="ALICE" power="50000000"/>

<link id="LINK_BOB" bandwidth="125000000" latency="5E-5"/>
<link id="LINK_ALICE" bandwidth="125000000" latency="5E-5"/>
<link id="SWITCH" bandwidth="125000000" latency="5E-5"
      sharing_policy="FATPIPE"/>
```

```
<route src="BOB" dest="ALICE">
  <link_ctn id="LINK_BOB"/>
  <link_ctn id="SWITCH"/>
<link_ctn id="LINK_ALICE"/> </route>
```

```
 <route src="ALICE" dest="BOB">
<link_ctn id="LINK_ALICE"/>
<link_ctn id="SWITCH"/>
<link_ctn id="LINK_BOB"/>
</route>
```

# Specifying routers

A router is like a host except it is invisible from the user level.

## <router> Tag

```
<router id="R1">
<router id="R2">
```

## Using it

```
<route src="A" dest="R1">
  <link_ctn id="Link1"/>
</route>

<route src="R1" dest="B">
  <link_ctn id="Link2"/>
</route>

<route src="R1" dest="C">
  <link_ctn id="Link3"/>
</route>
```

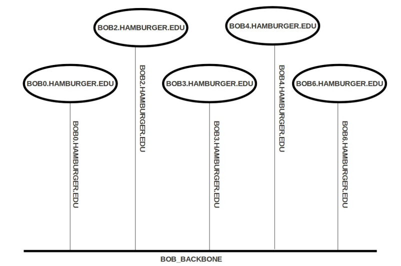
⇒/examples/msg/small_platform_with_routers.xml

# Outline

# Compacting the XML platform Description

## <cluster> Tag

```
<cluster id="MYCLUSTER"
   prefix="BOB" suffix=".HAMBURGER.EDU"
   radical="0,2-4,6" power="100000000"
   bw="125000000"
   lat="5E-5"
   bb_bw="250000000" bb_lat="5E-4" />
```




A cluster is actually expended as an AS with a special type of routing...

# Outline

# Full routing

```
<?xml version='1.0'?>
 <!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
 <platform version="3">
 <AS  id="AS0"  routing=" Full">
   <host id="Tremblay" power="98095000"/>                       Host declaration
   <host id="Jupiter" power="76296000"/>
...
   <link id="6" bandwidth="41279125" latency="5.9904e-05"/>   Link declaration
   <link id="11" bandwidth="252750" latency="0.00570455"/>
   <link id="3" bandwidth="34285625" latency="0.000514433"/>
   <link id="7" bandwidth="11618875" latency="0.00018998"/>
   <link id="9" bandwidth="7209750" latency="0.001461517"/>
...
   <route src="Tremblay" dst="Fafard">                          Route declaration
     <link_ctn id="4"/><link_ctn id="3"/><link_ctn id="2"/><link_ctn id="0"/><link_ctn id="1"/><l
   </route>
   <route src="Tremblay" dst="Ginette">
     <link_ctn id="4"/><link_ctn id="3"/><link_ctn id="5"/>
   </route>
   <route src="Tremblay" dst="Bourassa">
     <link_ctn id="4"/><link_ctn id="3"/><link_ctn id="2"/><link_ctn id="0"/><link_ctn id="1"/><l
   </route>
...                                                             All routes!!!! :(
 </AS>
 </platform>
```

# Hierarchy of AS

## Cluster with cabinets: platforms/griffon.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
<AS  id="AS_griffon"  routing="Full">
     <cluster id="griffon_cluster_cabinet1" prefix="griffon-" suffix=".nancy.grid5000.fr"
            radical="1-29,58,59,60" power="286087" bw="1.25e8" lat="2.4e-5"
            bb_bw="1.25e9" bb_lat="0" sharing_policy="FULLDUPLEX" bb_sharing_policy="SHARED"/>
     <cluster id="griffon_cluster_cabinet2" prefix="griffon-" suffix=".nancy.grid5000.fr"
            radical="30-57" power="286087" bw="1.25e8" lat="2.4e-5"
            bb_bw="1.25e9" bb_lat="0" sharing_policy="FULLDUPLEX" bb_sharing_policy="SHARED"/>
     <cluster id="griffon_cluster_cabinet3" prefix="griffon-" suffix=".nancy.grid5000.fr"
            radical="61-92" power="286087" bw="1.25e8" lat="2.4e-5"
            bb_bw="1.25e9" bb_lat="0" sharing_policy="FULLDUPLEX" bb_sharing_policy="SHARED"/>
     <link id="backbone" bandwidth="1.25e9" latency="2.4e-5" sharing_policy="SHARED"/>
     <ASroute src="griffon_cluster_cabinet1" dst="griffon_cluster_cabinet2"
        gw_src="griffon-griffon_cluster_cabinet1_router.nancy.grid5000.fr"
        gw_dst="griffon-griffon_cluster_cabinet2_router.nancy.grid5000.fr"
        symmetrical="YES">
               <link_ctn id="backbone"/>
     </ASroute>
     <ASroute src="griffon_cluster_cabinet2" dst="griffon_cluster_cabinet3"
       ...
     </ASroute>
     <ASroute src="griffon_cluster_cabinet1" dst="griffon_cluster_cabinet3"
       ...
     </ASroute>
</AS>
</platform>
```

# Hierarchy of AS 2

## A “Cloud” platform

```
< config id="General">
<prop id="network/coordinates" value="yes"></prop>
</config>

< AS  id="AS0"  routing="Vivaldi">
  < AS  id="AS1_dc1"  routing="RuleBased">
    <cluster id="AS1_cb1" prefix="cb1-" suffix=".dc1.acloud.com" radical="1-40" power="5.2297E9"
    <cluster id="AS1_cb2" prefix="cb2-" suffix=".dc1.acloud.com" radical="1-50" power="8.8925E9"
    <cluster id="AS1_cb3" prefix="cb3-" suffix=".dc1.acloud.com" radical="1-30" power="13.357E9"
    <AS id="gw_AS1_dc1" routing="Floyd">...
    <ASroute src="AS1_cb(.*)" dst="AS1_cb(.*)" gw_src="cb$1src-AS1_cb$1src_router.dc1.acloud.com
         <link_ctn id="link_dc1_cb$1src"/>
         <link_ctn id="link_dc1_cb$1dst"/>
    </ASroute>
      ...
   </AS>

   <AS  id="AS2_dc2"  routing="RuleBased">
...
   </AS>
...
   <!-- internal routes between clusters -->
   <ASroute src="AS3_cb(.*)" dst="AS3_cb(.*)" gw_src="cb$1src-AS3_cb$1src_router.dc3.acloud.com"
       <link_ctn id="link_dc3_cb$1src"/>
       <link_ctn id="link_dc3_cb$1dst"/>
   </ASroute>
...
   </AS>
```

# Routing types

- Full
- Floyd
- Dijkstra
- Dijkstra / cache
- Rulebased
- Cluster
- Vivaldi

We'll keep on adding new constructs.

**Peer**

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">

<config id="General">
  <prop id="network/coordinates" value="yes"></prop>
</config>
<AS  id="AS0"  routing="Vivaldi">
  < peer id="peer-0" coordinates="173.0 96.8 0.1" power="730000000.0"
        bw_in="13380000" bw_out="1024000" lat="5E-4"/>
....
```

# Outline

- The Network Representation Issue
- XML Based Formalism for Platform Description
  - The XML Approach
  - Specifying Host
  - Specifying inter-host network connections
  - Compacting the XML platform Description
  - Autonomous systems
  - Describe Availabilities in the XML File
  - Setting Properties
  - Examples of use
- Lua Based Formalism For Platform Description
  - Specifying Platform Element
  - Examples of use
  - Deploy Application

# Outline

- The Network Representation Issue
- XML Based Formalism for Platform Description
  - The XML Approach
  - Specifying Host
  - Specifying inter-host network connections
  - Compacting the XML platform Description
  - Autonomous systems
  - Describe Availabilities in the XML File
  - Setting Properties
  - Examples of use
- Lua Based Formalism For Platform Description
  - Specifying Platform Element
  - Examples of use
  - Deploy Application

# Attaching properties to elements

## Adding properties to Host

```
<host id="BOB" power="500000000">
  <prop id="memory" value="100000000"/>
  <prop id="desk" value="80E9" />
  <prop id="OS" value="Linux 2.6.22-14"/>
</host>
```

## Adding properties to Link

```
<link id="l1" bandwidth="125000000" latency="0.000100">
  <prop id="type" value="Ethernet"/>
</link>
```

⇒/examples/platforms/prop.xml

# Retrieving values

## SimDag interface

```
xbt_dict_t SD_link_get_properties(SD_link_t link);
const char* SD_link_get_property_value(SD_link_t link, const char* name);

xbt_dict_t SD_get_workstation_properties(SD_workstation_t workstation);
const char* SD_workstation_get_property_value(SD_workstation_t workstation, const char* name);
```

## MSG interface

```
xbt_dict_t MSG_host_get_properties(m_host_t host);
const char* MSG_host_get_property_value(m_host_t host, const char* name);
xbt_dict_t MSG_process_get_properties(m_process_t process);
const char* MSG_process_get_property_value(m_process_t process, const char* name);
```

## GRAS interface

```
xbt_dict_t gras_process_properties(void);
const char* gras_process_property_value(const char* name);
xbt_dict_t gras_os_host_properties(void);
const char* gras_os_host_property_value(const char* name);
```

# Outline

- The Network Representation Issue
- XML Based Formalism for Platform Description
  - The XML Approach
  - Specifying Host
  - Specifying inter-host network connections
  - Compacting the XML platform Description
  - Autonomous systems
  - Describe Availabilities in the XML File
  - Setting Properties
  - Examples of use
- Lua Based Formalism For Platform Description
  - Specifying Platform Element
  - Examples of use
  - Deploy Application

# Example of use

- Where to find XML platform examples ?
  ⇒ <simgrid_dir>/examples/platforms
  ⇒ <simgrid_dir>/examples/msg
- Where to find XML platform generators for SimGrid ?
  ⇒ <simgrid_dir>/contrib/trunk/platform_generation
  ⇒ <simgrid_dir>/contrib/trunk/VisualGrid

# Outline

# Outline

# Lua Based Formalism For Platform Description

## lua Platforms

platform.lua

```
require "simgrid"
simgrid.AS.new{id="AS0",mode="Full"};


simgrid.Host.new{id="Tremblay",power=98095000};

simgrid.Host.new{id="Jupiter",power=76296000};
...
for i=10,0,-1 do

simgrid.Link.new{id=i,bandwidth=252750 +
            i*768,latency=0.000270544+i*0.087};


simgrid.Route.new("Tremblay","Jupiter",{"1"});

simgrid.Route.new("Tremblay","Fafard",
          "0","1","2","3","4","8");
...
simgrid.msg_register_platform();
```

- lua console application ⇒ scripting language
- Using loops and conditional instructions
- Simple and lightweight edition
- Good performances when interconnecting with C Code.

# Specifying element in lua script

## Specifying host

```
simgrid.Host.new{id="Tremblay",power=98095000};
```

## Specifying link

```
simgrid.Link.new{id="3",bandwidth=98095000, latency=5E-5};
```

## Specifying route

```
simgrid.Route.new{"Tremblay","Ginette",{"3","4","5"}};
```

## Register platform

```
[MSG]     simgrid.msg_register_platform();
[SimDAG]  simgrid.sg_register_platform();
[GRAS]    simgrid.gras_register_platform();
```

# Outline

- The Network Representation Issue
- XML Based Formalism for Platform Description
  - The XML Approach
  - Specifying Host
  - Specifying inter-host network connections
  - Compacting the XML platform Description
  - Autonomous systems
  - Describe Availabilities in the XML File
  - Setting Properties
  - Examples of use
- Lua Based Formalism For Platform Description
  - Specifying Platform Element
  - Examples of use
  - Deploy Application

# Example of use

- Where to find lua console examples ? ⇒ `<simgrid_dir>/examples/msg/masterslave` ⇒ `<simgrid_dir>/examples/simdag` ⇒ `<simgrid_dir>/examples/gras/console`
- Where to find lua script examples ? ⇒ `<simgrid_dir>/examples/lua/masterslave_bypass.lua` ⇒ `<simgrid_dir>/examples/msg/masterslave/platform_script.lua` ⇒ `<simgrid_dir>/examples/simdag/platform_script.lua` ⇒ `<simgrid_dir>/examples/gras/console/gras_platform_script.lua`

# Outline

- The Network Representation Issue
- XML Based Formalism for Platform Description
  - The XML Approach
  - Specifying Host
  - Specifying inter-host network connections
  - Compacting the XML platform Description
  - Autonomous systems
  - Describe Availabilities in the XML File
  - Setting Properties
  - Examples of use
- Lua Based Formalism For Platform Description
  - Specifying Platform Element
  - Examples of use
  - Deploy Application

# Deploy applciation in lua script

## Set funtion to process

```
simgrid.Host.setFunction("Tremblay","master",{"20","550000","1000","4"});
simgrid.Host.setFunction("Jupiter","slave",{"1"});
```

## Register application

```
[MSG]     simgrid.msg_register_platform();
[SimDAG] simgrid.sg_register_platform();
[GRAS]   simgrid.gras_register_platform();
```