Rapport TP1

March 21, 2018

Thomas COUCHOUD
thomas.couchoud@etu.univ-tours.fr
Victor COLEAU
victor.coleau@etu.univ-tours.fr

Contents

Chapter 1

Ibsim

Après avoir recopié le fichier de configuration topologique et lancé le programme avec ce dernier, on peut observer que la sortie dans la console correspond exactement avec la figure 2 du sujet.

```
administrateur@vm-ubuntu-base:~/Bureau/ibsim/ibsim$ ibsim example.topo
parsing: example.topo
example.topo: parsed 10 lines
#########################
Network simulator ready.
MaxNetNodes    = 2048
MaxNetSwitches = 256
MaxNetPorts    = 13312
MaxLinearCap   = 30720
MaxMcastCap    = 1024
sim (inactive)> dump
# Net status - Wed Feb 14 14:45:28 2018

Ca 1 "workstation1"     nodeguid 100000 sysimgguid 100000
100001  [1]     "router1"[1]     lid 0 lmc 0 smlid 0  4x  2.5G Init/LinkUp

Ca 1 "workstation2"     nodeguid 100002 sysimgguid 100002
100003  [1]     "router2"[2]     lid 0 lmc 0 smlid 0  4x  2.5G Init/LinkUp

Switch 2 "router1"      nodeguid 200000 sysimgguid 200000
#       linearcap 30720 FDBtop 0 portchange 1
200000  [0]     "Sma Port"[0]    lid 0 lmc 0 smlid 0  4x  2.5G Active/LinkUp
200000  [1]     "workstation1"[1]         4x  2.5G Init/LinkUp
200000  [2]     "router2"[1]      4x  2.5G Init/LinkUp

Switch 2 "router2"      nodeguid 200001 sysimgguid 200001
#       linearcap 30720 FDBtop 0 portchange 1
200001  [0]     "Sma Port"[0]    lid 0 lmc 0 smlid 0  4x  2.5G Active/LinkUp
200001  [1]     "router1"[2]      4x  2.5G Init/LinkUp
200001  [2]     "workstation2"[1]         4x  2.5G Init/LinkUp
#  dumped 4 nodes
sim (inactive)>
```

Figure 1.1 – Sortie du programme avec dump

On peut par la suite rentrer de nouvelles commandes. La liste de ces dernières est disponible en tapant `help`.

- !: Lance des commandes depuis un fichier
- dump: Affiche les informations du réseau
- Route: Affiche la route entre deux noeuds
- Link: permet de faire une connexion réseau.
- Relink: Réétabli une connexion réseau.

- Unlink: Supprime un lien réseau.
- Clear: Supprime les liens réseau & réinitialise les ports.
- Guid: Définit le GUID d'un noeud.
- Error: Définit le statut d'erreur d'un port/noeud.
- PerformanceSet: Définit les conteurs de performance.
- Baselid: Change le lid d'un port.
- Verbose: Définit le niveau de sortie dans la console.
- Wait: Suspend la simulation.
- Attached: Liste tous les clients attachés.
- X: Déconnecte un client.
- Quit: Ferme le programme.

Nous avons tenté route mais on nous dit que les lid ne sont pas définis alors que nous les avons définis pour chaque nœud.

Chapter 2

Générateur de topologie

Tout d'abord le dossier de sources contient un Makefile. Il est donc possible de compiler facilement le programme grâce à la commande `make`.

Plusieurs classes sont présentes:

- Node: représente un nœud du réseau. Ce dernier est composé d'un nom, type, ID, et d'une liste de ports avec leurs connections.
- Machine: représente une machine du réseau. Cette classe étend Node avec comme type "Hca" et 1 port.
- Commutator: représente un switch du réseau. Cette classe étend Node avec comme type "Switch".

Le fichier main réalise la création des éléments ainsi que les liens entre ces derniers:

- Création de toutes les machines
- Création de tout les switchs
  - Aggregation
    - Création de tout les switch dans la zone d'aggregation
    - Création des liens entre le switch et les machines qui le concernent
  - Edge
    - Création de tout les switch dans la zone edge
    - Création des liens entre le switch et les switchs de la zone d'aggregation
  - Core
    - Création de tout les switch dans la zone core
    - Création des liens entre le switch et les switchs de la zone edge
- Ajustements des ports pour suivre la norme imposée par le PDF

Voici un exemple de fichier de sortie pour $k = 4$:

Listing 2.1 – topo.topo

```
#Fat tree topology file
#Value of k = 4
#Total number of hosts = 16
#Number of hosts under each switch = 2
#Number of commutators = 20
#########################################################

#Machines:
Hca   1  "Node(0)" lid 0
[1]"Edge(0 0 1)"[2]

Hca   1  "Node(1)" lid 1
[1]"Edge(0 0 1)"[4]

Hca   1  "Node(2)" lid 2
[1]"Edge(0 1 1)"[2]

```

```
Hca   1  "Node(3)" lid 3
[1]"Edge(0 1 1)"[4]

Hca   1  "Node(4)" lid 4
[1]"Edge(1 0 1)"[2]

Hca   1  "Node(5)" lid 5
[1]"Edge(1 0 1)"[4]

Hca   1  "Node(6)" lid 6
[1]"Edge(1 1 1)"[2]

Hca   1  "Node(7)" lid 7
[1]"Edge(1 1 1)"[4]

Hca   1  "Node(8)" lid 8
[1]"Edge(2 0 1)"[2]

Hca   1  "Node(9)" lid 9
[1]"Edge(2 0 1)"[4]

Hca   1  "Node(10)" lid 10
[1]"Edge(2 1 1)"[2]

Hca   1  "Node(11)" lid 11
[1]"Edge(2 1 1)"[4]

Hca   1  "Node(12)" lid 12
[1]"Edge(3 0 1)"[2]

Hca   1  "Node(13)" lid 13
[1]"Edge(3 0 1)"[4]

Hca   1  "Node(14)" lid 14
[1]"Edge(3 1 1)"[2]

Hca   1  "Node(15)" lid 15
[1]"Edge(3 1 1)"[4]

#Edge:
Switch  4  "Edge(0 0 1)" lid 16
[1]"Aggr(0 2 1)"[2]
[2]"Node(0)"[1]
[3]"Aggr(0 3 1)"[2]
[4]"Node(1)"[1]

Switch  4  "Edge(0 1 1)" lid 17
[1]"Aggr(0 2 1)"[4]
[2]"Node(2)"[1]
[3]"Aggr(0 3 1)"[4]
[4]"Node(3)"[1]

Switch  4  "Edge(1 0 1)" lid 18
[1]"Aggr(1 2 1)"[2]
[2]"Node(4)"[1]
[3]"Aggr(1 3 1)"[2]
[4]"Node(5)"[1]

Switch  4  "Edge(1 1 1)" lid 19
[1]"Aggr(1 2 1)"[4]
[2]"Node(6)"[1]
[3]"Aggr(1 3 1)"[4]
[4]"Node(7)"[1]

Switch  4  "Edge(2 0 1)" lid 20
[1]"Aggr(2 2 1)"[2]
[2]"Node(8)"[1]
[3]"Aggr(2 3 1)"[2]
[4]"Node(9)"[1]
```

```
87
88  Switch  4  "Edge(2 1 1)" lid 21
89  [1]"Aggr(2 2 1)"[4]
90  [2]"Node(10)"[1]
91  [3]"Aggr(2 3 1)"[4]
92  [4]"Node(11)"[1]
93
94  Switch  4  "Edge(3 0 1)" lid 22
95  [1]"Aggr(3 2 1)"[2]
96  [2]"Node(12)"[1]
97  [3]"Aggr(3 3 1)"[2]
98  [4]"Node(13)"[1]
99
100 Switch  4  "Edge(3 1 1)" lid 23
101 [1]"Aggr(3 2 1)"[4]
102 [2]"Node(14)"[1]
103 [3]"Aggr(3 3 1)"[4]
104 [4]"Node(15)"[1]
105
106 #Aggregation:
107 Switch  4  "Aggr(0 2 1)" lid 24
108 [1]"Core(4 1 1)"[1]
109 [2]"Edge(0 0 1)"[1]
110 [3]"Core(4 1 2)"[1]
111 [4]"Edge(0 1 1)"[1]
112
113 Switch  4  "Aggr(0 3 1)" lid 25
114 [1]"Core(4 2 1)"[1]
115 [2]"Edge(0 0 1)"[3]
116 [3]"Core(4 2 2)"[1]
117 [4]"Edge(0 1 1)"[3]
118
119 Switch  4  "Aggr(1 2 1)" lid 26
120 [1]"Core(4 1 1)"[2]
121 [2]"Edge(1 0 1)"[1]
122 [3]"Core(4 1 2)"[2]
123 [4]"Edge(1 1 1)"[1]
124
125 Switch  4  "Aggr(1 3 1)" lid 27
126 [1]"Core(4 2 1)"[2]
127 [2]"Edge(1 0 1)"[3]
128 [3]"Core(4 2 2)"[2]
129 [4]"Edge(1 1 1)"[3]
130
131 Switch  4  "Aggr(2 2 1)" lid 28
132 [1]"Core(4 1 1)"[3]
133 [2]"Edge(2 0 1)"[1]
134 [3]"Core(4 1 2)"[3]
135 [4]"Edge(2 1 1)"[1]
136
137 Switch  4  "Aggr(2 3 1)" lid 29
138 [1]"Core(4 2 1)"[3]
139 [2]"Edge(2 0 1)"[3]
140 [3]"Core(4 2 2)"[3]
141 [4]"Edge(2 1 1)"[3]
142
143 Switch  4  "Aggr(3 2 1)" lid 30
144 [1]"Core(4 1 1)"[4]
145 [2]"Edge(3 0 1)"[1]
146 [3]"Core(4 1 2)"[4]
147 [4]"Edge(3 1 1)"[1]
148
149 Switch  4  "Aggr(3 3 1)" lid 31
150 [1]"Core(4 2 1)"[4]
151 [2]"Edge(3 0 1)"[3]
152 [3]"Core(4 2 2)"[4]
153 [4]"Edge(3 1 1)"[3]
154
155 #Core:
```

```
156 Switch  4  "Core(4 1 1)" lid 32
157 [1]"Aggr(0 2 1)"[1]
158 [2]"Aggr(1 2 1)"[1]
159 [3]"Aggr(2 2 1)"[1]
160 [4]"Aggr(3 2 1)"[1]
161
162 Switch  4  "Core(4 1 2)" lid 33
163 [1]"Aggr(0 2 1)"[3]
164 [2]"Aggr(1 2 1)"[3]
165 [3]"Aggr(2 2 1)"[3]
166 [4]"Aggr(3 2 1)"[3]
167
168 Switch  4  "Core(4 2 1)" lid 34
169 [1]"Aggr(0 3 1)"[1]
170 [2]"Aggr(1 3 1)"[1]
171 [3]"Aggr(2 3 1)"[1]
172 [4]"Aggr(3 3 1)"[1]
173
174 Switch  4  "Core(4 2 2)" lid 35
175 [1]"Aggr(0 3 1)"[3]
176 [2]"Aggr(1 3 1)"[3]
177 [3]"Aggr(2 3 1)"[3]
178 [4]"Aggr(3 3 1)"[3]
```

Appendix A

Sources du générateur

A.1 Node

Listing A.1 – Node.h

```
//
// Created by Thomas Couchoud on 08/02/2018.
//

#ifndef GENERATOR_NODE_H
#define GENERATOR_NODE_H

#include <string>
#include <vector>

/**
 * A node of the network.
 */
class Node
{
private:
   //! Its unique ID.
   int ID;
   //! Next unique ID for the generation.
   static int NEXT_ID;
   //! The type of the node.
   std::string type;
   //! The name of the node.
   std::string name;
protected:
   //! The ports of the node.
   std::vector<Node *> connectedTo;
public:
   /**
    * Constructor.
    * @param type The type of the node.
    * @param name The name of the node.
    * @param connections The number of ports of the node.
    * @param i The x coordinate.
    * @param j The y coordinate.
    * @param k The z coordinate.
    */
   Node(const std::string &type, const std::string &name, int connections, int i, int j, int k);

   /**
    * Get the type.
    * @return The type.
    */
   std::string &getType();

   /**
    * Get the ID.
```

```
48      * @return The ID.
49      */
50     int getID();
51 
52     /**
53      * Get the name.
54      * @return The name.
55      */
56     std::string &getName();
57 
58     /**
59      * Get the number of ports.
60      * @return The number of ports.
61      */
62     int getPortCount();
63 
64     /**
65      * Set where a port is connected.
66      * @param index The port index.
67      * @param node Where this port is connected.
68      */
69     void setLink(int index, Node * node);
70 
71     /**
72      * Get where a port is connected.
73      * @param index The port concerned.
74      * @return The Node connected to, or null if disconnected.
75      */
76     Node * getLink(int index);
77 
78     /**
79      * Get the port where a Node is connected.
80      * @param node The Node to find.
81      * @return The index of the port, -1 if not found.
82      */
83     int getPortWhereIs(Node * node);
84 
85     bool operator==(const Node &rhs) const;
86 
87     bool operator!=(const Node &rhs) const;
88 };
89 
90 #endif //GENERATOR_NODE_H
```

Listing A.2 – Node.cpp

```
1  //
2  // Created by Thomas Couchoud on 08/02/2018.
3  //
4 
5  #include <algorithm>
6  #include <sstream>
7  #include "Node.h"
8 
9  using namespace std;
10 
11 int Node::NEXT_ID = 0;
12 
13 Node::Node(const std::string &type, const std::string &name, int connections, int i, int j, int k) :
       type(type)
14 {
15     this->ID = Node::NEXT_ID++;
16     connectedTo = vector<Node *>();
17     connectedTo.resize(connections, nullptr);
18 
19     stringstream s;
20     s << name << "(" << i;
21     if(j != -1 && k != -1)
```

```
22       s << " " << j << " " << k;
23    s << ")";
24    this->name = s.str();
25 }
26
27 string &Node::getType()
28 {
29    return type;
30 }
31
32 int Node::getID()
33 {
34    return ID;
35 }
36
37 string &Node::getName()
38 {
39    return name;
40 }
41
42 void Node::setLink(int index, Node * node)
43 {
44    if(index >= connectedTo.size())
45    {
46       printf("Trying to set a link on a non existing port (%d/%d) on %s", index, getPortCount(),
           getName().c_str());
47       exit(EXIT_FAILURE);
48    }
49    if(connectedTo[index] != nullptr)
50       printf("WARN: Overriding existing link of %s (%d)\n", getName().c_str(), index);
51    connectedTo[index] = node;
52 }
53
54 int Node::getPortCount()
55 {
56    return static_cast<int>(connectedTo.size());
57 }
58
59 Node * Node::getLink(int index)
60 {
61    if(index >= connectedTo.size())
62    {
63       perror("Trying to get a link of a non existing port");
64       exit(EXIT_FAILURE);
65    }
66    return connectedTo[index];
67 }
68
69 int Node::getPortWhereIs(Node * node)
70 {
71    auto it = std::find(connectedTo.begin(), connectedTo.end(), node);
72    return static_cast<int>(it == connectedTo.end() ? -1 : distance(connectedTo.begin(), it));
73 }
74
75 bool Node::operator==(const Node &rhs) const
76 {
77    return ID == rhs.ID;
78 }
79
80 bool Node::operator!=(const Node &rhs) const
81 {
82    return !(rhs == *this);
83 }
```

A.2 Machine

Listing A.3 – Machine.h

```
//
// Created by Thomas Couchoud on 08/02/2018.
//

#ifndef GENERATOR_MACHINE_H
#define GENERATOR_MACHINE_H

#include "Node.h"

/**
 * A machine.
 */
class Machine : public Node
{
public:
   /**
    * Constructor.
    *
    * @param i The x coordinate.
    * @param j The y coordinate.
    * @param k The z coordinate.
    */
   Machine(int i, int j, int k);

   /**
    * Set the node where the machine is connected to.
    * @param node The node to connect.
    */
   void setConnectedTo(Node * node);
};

#endif //GENERATOR_MACHINE_H
```

Listing A.4 – Machine.cpp

```
//
// Created by Thomas Couchoud on 08/02/2018.
//

#include "Machine.h"

Machine::Machine(int i, int j, int k) : Node("Hca", "Node", 1, i, j, k)
{
}

void Machine::setConnectedTo(Node * node)
{
   setLink(0, node);
}
```

A.3 Commutator

Listing A.5 – Commutator.h

```
//
// Created by Thomas Couchoud on 08/02/2018.
//

#ifndef GENERATOR_COMMUTATOR_H
#define GENERATOR_COMMUTATOR_H

#include "Node.h"
#include "Machine.h"

```

```
/**
 * A Switch.
 */
class Commutator : public Node
{
public:
   /**
    * Constructor.
    * @param name The name of the switch.
    * @param connections The number of ports of teh switch.
    * @param i The x coordinate.
    * @param j The y coordinate.
    * @param k The z coordinate.
    */
   Commutator(const std::string &name, int connections, int i, int j, int k);

   /**
    * Normalize ports to follow those in the TP subject.
    */
   void normalizePorts();
};

#endif //GENERATOR_COMMUTATOR_H
```

Listing A.6 – Commutator.cpp

```
//
// Created by Thomas Couchoud on 08/02/2018.
//

#include <vector>
#include <cmath>
#include "Commutator.h"
#include "Machine.h"

using namespace std;

Commutator::Commutator(const std::string &name, int connections, int i, int j, int k) : Node("Switch",
    name, connections, i, j, k)
{
}

void Commutator::normalizePorts()
{
   unsigned long s = connectedTo.size();
   vector<Node *> normalized = vector<Node *>(s, nullptr);
   for(int i = 0; i < s; i ++)
      normalized[(i * 2) % s + (i * 2 < s ? 1 : 0)] = connectedTo[i];
   swap(connectedTo, normalized);
}
```

A.4 Main

Listing A.7 – main.cpp

```
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>
#include "Commutator.h"

using namespace std;

/**
 * Write a node into a stream.
 *
```

```
 * @param s The stream to write into.
 * @param node The node to write.
 */
void writeNode(ostream &s, Node * node)
{
   //Write node
   s << node->getType() << '\t' << node->getPortCount() << '\t' << '"' << node->getName() << '"' << " lid
       " << node->getID() << endl;
   for(int i = 0; i < node->getPortCount(); i++)
   {
      //Write connection
      Node * connectedTo = nullptr;
      if((connectedTo = node->getLink(i)) != nullptr)
         s << '[' << (i + 1) << ']' << '"' << connectedTo->getName() << '"' << '[' <<
             (connectedTo->getPortWhereIs(node) + 1) << ']' << endl;
   }
   s << endl;
}

/**
 * Write the topology file.
 * @param s The stream to write into.
 * @param k The k used.
 * @param machines The machines.
 * @param edge The edge switches.
 * @param aggregation The aggregation switches.
 * @param core The core switches.
 */
void writeAll(ostream &s, int k, vector<Machine *> machines, vector<Commutator *> edge, vector<Commutator
    *> aggregation, vector<Commutator *> core)
{
   s << "#Fat tree topology file" << endl;
   s << "#Value of k = " << k << endl;
   s << "#Total number of hosts = " << machines.size() << endl;
   s << "#Number of hosts under each switch = " << (k / 2) << endl;
   s << "#Number of commutators = " << (edge.size() + aggregation.size() + core.size()) << endl;
   s << "############################################################" << endl << endl;

   s << "#Machines:" << endl;
   for(auto m : machines)
      writeNode(s, m);

   s << "#Edge:" << endl;
   for(auto c : edge)
      writeNode(s, c);

   s << "#Aggregation:" << endl;
   for(auto c : aggregation)
      writeNode(s, c);

   s << "#Core:" << endl;
   for(auto c : core)
      writeNode(s, c);

   s.flush();
}

int main(int argc, char ** argv)
{
   int k;
   if(argc == 2)
   {
      k = atoi(argv[1]);
   }
   else
   {
      cout << "Enter k: ";
      cin >> k;
   }
```

```

cout << "Selected k: " << k << endl;
 if(k%2 != 0)
 {
     cout << "K must be a multiple of 2" << endl;
     exit(EXIT_FAILURE);
 }

//Init constants (count of elements par category)
auto machineCountPerPod = static_cast<int>(pow(k / 2, 2));
auto machineCount = machineCountPerPod * k;
auto edgeCount = static_cast<int>(pow(k, 2) / 2);
auto aggregationCount = static_cast<int>(pow(k, 2) / 2);
auto coreCount = static_cast<int>(pow(k / 2, 2));

vector<Machine *> machines = vector<Machine *>();
vector<Commutator *> edge = vector<Commutator *>();
vector<Commutator *> aggregation = vector<Commutator *>();
vector<Commutator *> core = vector<Commutator *>();

//Create machines
for(int i = 0; i < machineCount; i++)
{
   cout << "Creating machine " << i << endl;
   machines.push_back(new Machine(i, -1, -1));
}

//Create edge
for(int i = 0; i < edgeCount; i++)
{
   //Create switch
   cout << "Creating edge " << i << endl;
   Commutator * commutator = new Commutator("Edge", k, i / (k / 2), i % (k / 2), 1);
   edge.push_back(commutator);

   //Link switch to the machines
   for(int j = 0; j < k / 2; j++)
   {
      Machine * m = machines[i * (k / 2) + j];
      m->setConnectedTo(commutator);
      commutator->setLink(j, m);
   }
}

//Create aggregation
for(int i = 0; i < aggregationCount; i++)
{
   //Create switch
   cout << "Creating aggregation " << i << endl;
   Commutator * commutator = new Commutator("Aggr", k, i / (k / 2), i % (k / 2) + (k / 2), 1);
   aggregation.push_back(commutator);

   //Link to edge switches
   int group = i / (k / 2);
   int offsetPort = i % (k / 2);
   for(int j = 0; j < k / 2; j++)
   {
      Commutator * c = edge[group * k / 2 + j];
      c->setLink(k / 2 + offsetPort, commutator);
      commutator->setLink(j, c);
   }
}

//Create core
for(int i = 0; i < coreCount; i++)
{
   //Create switch
   cout << "Creating core " << i << endl;
   Commutator * commutator = new Commutator("Core", k, k, 1 + i / (k / 2), 1 + i % (k / 2));
```

```
147         core.push_back(commutator);
148
149         //Link to aggregation
150         int offsetAggregation = 2 * i / k;
151         int offsetPortAggregation = i % (k / 2);
152         for(int j = 0; j < k; j++)
153         {
154            Commutator * c = aggregation[offsetAggregation + j * (k / 2)];
155            c->setLink(k / 2 + offsetPortAggregation, commutator);
156            commutator->setLink(j, c);
157         }
158      }
159
160      //Normalize ports to follow the naming in the TP subject
161      for(auto c : edge)
162         c->normalizePorts();
163      for(auto c : aggregation)
164         c->normalizePorts();
165
166      //Write file
167      ofstream outFile;
168      outFile.open("topo.topo");
169      writeAll(outFile, k, machines, edge, aggregation, core);
170      outFile.close();
171
172      //Bye bye
173      cout << flush;
174      return 0;
175   }
```