Rapport TP1

March 14, 2018

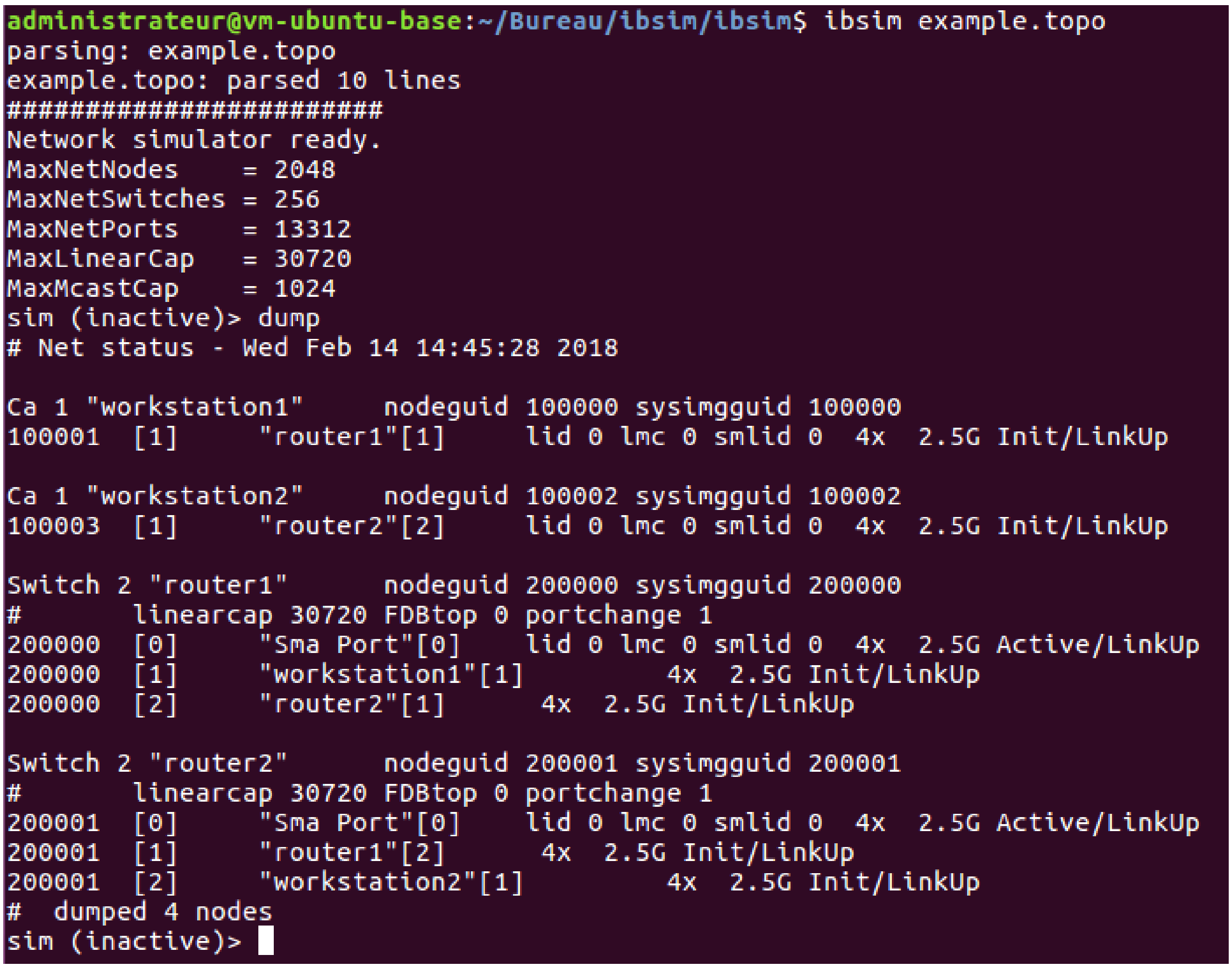
Thomas COUCHOUD
thomas.couchoud@etu.univ-tours.fr
Victor COLEAU
victor.coleau@etu.univ-tours.fr

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]
```

```

Switch  4  "Edge(2 1 1)" lid 21
[1]"Aggr(2 2 1)"[4]
[2]"Node(10)"[1]
[3]"Aggr(2 3 1)"[4]
[4]"Node(11)"[1]

Switch  4  "Edge(3 0 1)" lid 22
[1]"Aggr(3 2 1)"[2]
[2]"Node(12)"[1]
[3]"Aggr(3 3 1)"[2]
[4]"Node(13)"[1]

Switch  4  "Edge(3 1 1)" lid 23
[1]"Aggr(3 2 1)"[4]
[2]"Node(14)"[1]
[3]"Aggr(3 3 1)"[4]
[4]"Node(15)"[1]

#Aggregation:
Switch  4  "Aggr(0 2 1)" lid 24
[1]"Core(4 1 1)"[1]
[2]"Edge(0 0 1)"[1]
[3]"Core(4 1 2)"[1]
[4]"Edge(0 1 1)"[1]

Switch  4  "Aggr(0 3 1)" lid 25
[1]"Core(4 2 1)"[1]
[2]"Edge(0 0 1)"[3]
[3]"Core(4 2 2)"[1]
[4]"Edge(0 1 1)"[3]

Switch  4  "Aggr(1 2 1)" lid 26
[1]"Core(4 1 1)"[2]
[2]"Edge(1 0 1)"[1]
[3]"Core(4 1 2)"[2]
[4]"Edge(1 1 1)"[1]

Switch  4  "Aggr(1 3 1)" lid 27
[1]"Core(4 2 1)"[2]
[2]"Edge(1 0 1)"[3]
[3]"Core(4 2 2)"[2]
[4]"Edge(1 1 1)"[3]

Switch  4  "Aggr(2 2 1)" lid 28
[1]"Core(4 1 1)"[3]
[2]"Edge(2 0 1)"[1]
[3]"Core(4 1 2)"[3]
[4]"Edge(2 1 1)"[1]

Switch  4  "Aggr(2 3 1)" lid 29
[1]"Core(4 2 1)"[3]
[2]"Edge(2 0 1)"[3]
[3]"Core(4 2 2)"[3]
[4]"Edge(2 1 1)"[3]

Switch  4  "Aggr(3 2 1)" lid 30
[1]"Core(4 1 1)"[4]
[2]"Edge(3 0 1)"[1]
[3]"Core(4 1 2)"[4]
[4]"Edge(3 1 1)"[1]

Switch  4  "Aggr(3 3 1)" lid 31
[1]"Core(4 2 1)"[4]
[2]"Edge(3 0 1)"[3]
[3]"Core(4 2 2)"[4]
[4]"Edge(3 1 1)"[3]

#Core:
```

```
Switch  4  "Core(4 1 1)" lid 32
[1]"Aggr(0 2 1)"[1]
[2]"Aggr(1 2 1)"[1]
[3]"Aggr(2 2 1)"[1]
[4]"Aggr(3 2 1)"[1]

Switch  4  "Core(4 1 2)" lid 33
[1]"Aggr(0 2 1)"[3]
[2]"Aggr(1 2 1)"[3]
[3]"Aggr(2 2 1)"[3]
[4]"Aggr(3 2 1)"[3]

Switch  4  "Core(4 2 1)" lid 34
[1]"Aggr(0 3 1)"[1]
[2]"Aggr(1 3 1)"[1]
[3]"Aggr(2 3 1)"[1]
[4]"Aggr(3 3 1)"[1]

Switch  4  "Core(4 2 2)" lid 35
[1]"Aggr(0 3 1)"[3]
[2]"Aggr(1 3 1)"[3]
[3]"Aggr(2 3 1)"[3]
[4]"Aggr(3 3 1)"[3]
```

Appendix A

Sources du générateur

A.1 Node

Listing A.1 – Node.h

```cpp
//
// 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.
```

```
     * @return The ID.
     */
    int getID();

    /**
     * Get the name.
     * @return The name.
     */
    std::string &getName();

    /**
     * Get the number of ports.
     * @return The number of ports.
     */
    int getPortCount();

    /**
     * Set where a port is connected.
     * @param index The port index.
     * @param node Where this port is connected.
     */
    void setLink(int index, Node * node);

    /**
     * Get where a port is connected.
     * @param index The port concerned.
     * @return The Node connected to, or null if disconnected.
     */
    Node * getLink(int index);

    /**
     * Get the port where a Node is connected.
     * @param node The Node to find.
     * @return The index of the port, -1 if not found.
     */
    int getPortWhereIs(Node * node);

    bool operator==(const Node &rhs) const;

    bool operator!=(const Node &rhs) const;
};

#endif //GENERATOR_NODE_H
```

Listing A.2 – Node.cpp

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

#include <algorithm>
#include <sstream>
#include "Node.h"

using namespace std;

int Node::NEXT_ID = 0;

Node::Node(const std::string &type, const std::string &name, int connections, int i, int j, int k) :
    type(type)
{
    this->ID = Node::NEXT_ID++;
    connectedTo = vector<Node *>();
    connectedTo.resize(connections, nullptr);

    stringstream s;
    s << name << "(" << i;
    if(j != -1 && k != -1)
```

```
      s << " " << j << " " << k;
   s << ")";
   this->name = s.str();
}

string &Node::getType()
{
   return type;
}

int Node::getID()
{
   return ID;
}

string &Node::getName()
{
   return name;
}

void Node::setLink(int index, Node * node)
{
   if(index >= connectedTo.size())
   {
      printf("Trying to set a link on a non existing port (%d/%d) on %s", index, getPortCount(),
          getName().c_str());
      exit(EXIT_FAILURE);
   }
   if(connectedTo[index] != nullptr)
      printf("WARN: Overriding existing link of %s (%d)\n", getName().c_str(), index);
   connectedTo[index] = node;
}

int Node::getPortCount()
{
   return static_cast<int>(connectedTo.size());
}

Node * Node::getLink(int index)
{
   if(index >= connectedTo.size())
   {
      perror("Trying to get a link of a non existing port");
      exit(EXIT_FAILURE);
   }
   return connectedTo[index];
}

int Node::getPortWhereIs(Node * node)
{
   auto it = std::find(connectedTo.begin(), connectedTo.end(), node);
   return static_cast<int>(it == connectedTo.end() ? -1 : distance(connectedTo.begin(), it));
}

bool Node::operator==(const Node &rhs) const
{
   return ID == rhs.ID;
}

bool Node::operator!=(const Node &rhs) const
{
   return !(rhs == *this);
}
```

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"

```

```cpp
/**
 * 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

```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

```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";
         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));
```

```
      core.push_back(commutator);

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

   //Normalize ports to follow the naming in the TP subject
   for(auto c : edge)
      c->normalizePorts();
   for(auto c : aggregation)
      c->normalizePorts();

   //Write file
   ofstream outFile;
   outFile.open("topo.topo");
   writeAll(outFile, k, machines, edge, aggregation, core);
   outFile.close();

   //Bye bye
   cout << flush;
   return 0;
}
```