UNIVERSITY OF CALIFORNIA, SAN DIEGO

Visualization software for Data Center Switch Network

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Aram Shahinfard

Committee in charge:

Professor Amin Vahdat, Chair
Professor William Griswold
Professor Geoffrey Voelker

2011

The thesis of Aram Shahinfard is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____
Chair

University of California, San Diego

2011

EPIGRAPH

The true sign of intelligence is not knowledge but imagination.

—Albert Einstein

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Amin Vahdat for his support as my advisor and the chair of my committee. Without his guidance and vision, this work would not have been as successful.

Further, I would like to acknowledge Professors William Grisworld and Geoffrey Voelker for their guidance and for serving in my thesis committee.

ABSTRACT OF THE THESIS

Visualization software for Data Center Switch Network

by

Aram Shahinfard

Master of Science in Computer Science

University of California, San Diego, 2011

Professor Amin Vahdat, Chair

Today's data centers may contain tens of thousands computers. Manually monitoring these networks is time consuming, so there is an emerging need to automate these tasks by implementing network management systems. It is important for network engineers to have a tool enabling them to monitor the network to spot problems before users are affected. This need is also critical during the network design since it will make the network debugging easier. A well structured management software will also enables the designers to demonstrate the network behavior before development. One important component of most network management tools is visualization tool.

Data centers' network architecture typically consists of a tree of routing and switching elements with more specialized and expensive equipment moving up the network hierarchy. When deploying the highest-end IP switches/routers, resulting topologies may only support 50% of the aggregate bandwidth available at the edge of the network, while still incurring tremendous cost. Non uniform bandwidth among data center nodes complicates application design and limits overall system performance. UCSD data center switch research group has proposed a new

technique on how to leverage largely commodity Ethernet switches to support the full aggregate bandwidth of clusters consisting of tens of thousands of elements. The group argues that appropriately architected and interconnected commodity switches will deliver more performance at less cost than higher-end solutions.

In this project a visualization tool was developed for the proposed network. This tool uses data captured from the network and use graphic techniques to visualize the data.

Chapter 1

Introduction

Network monitoring and management are critical tasks of network engineers. Due to the scale of current networks, it is not possible to accomplish these tasks manually. Using automated software is a more cost and time effective alternative. Companies that deal with large-scale and complicated networks are seeing the emerging need to have smarter network management tools on a daily basis.

In addition to monitoring an existing network, a network management tool can be used during the design and development phases. Developing a new network architecture without adequate evidence of accuracy could be very time and cost consuming. Designers, as well as network engineers, need automated tools to study the network behavior in order to adjust their design decision, and to debug the network during the development phase.

One of the main component of management tools is visualization. The use of visualization to present information is not a new phenomenon. It has been used in maps, scientific drawings, and data plots for over a thousand years. Computer graphics has, from its beginning, been used to study scientific problems.

Visualization is an important aspect of data network because of their dynamic nature. And graphical displays of networks are particularly attractive, since they enable designers and engineers to display in a compact way the relevant factors in a network, how they work in different situations, how they tolerate the faults, and what the overall structure looks like. The ability to visualize the status of a network allows human users to rapidly assess the health of the system and

identifying problems that span across components. Visualization tools help deal with inconsistencies since autonomic management tools have difficulty when the structure of the application does not fit the assumptions of the management system. A visualization tool allows operators to see the information they need and then direct the system correctly, without requiring changes to the assumptions of the management system.

On the other hand, data center networks are getting more and more attention these days as they are growing very fast. Several architectures are being proposed to utilize these networks more effectively. Having a fully autonomic data center that is expressed declaratively, maintained and healed automatically is still quite impossible. For most data centers, human operators need to periodically act by writing maintenance scripts, controlling updates, and modifying systems. These characteristics makes data center networks a good candidate for visualization.

Today's data centers leverage a large number of commodity parts, which can deliver the required computational and storage capacities, while helping minimize the total cost. On the other hand, the communication networks that are used to inter-connect these clusters comprise of largely non-commodity parts, simply because one such solution today can neither provide the necessary communication band-width nor allow for large clusters to be built. Trying to solve a similar problem for the telephone network a solution was proposed to interconnect small, low-cost switches to build a practical multi-stage telephone switching system. Visualization tools will help operators to understand the current state of the system, even when the network is in an unstable situation due to an upgrade or failure.

The UCSD data center switch research group had proposed a data center network architecture that leverages commodity switches interconnected in a fat tree topology. This topology has the nice property of providing full bisection bandwidth, something that today's architectures can only achieve at a very high cost. Further, they propose a unique routing algorithm called two-level lookup that aims to balance trace across the network.

In the current thesis a visualization software was developed for the proposed network as part of a management tool. Data is captured periodically form the

network and is saved to a database. The visualization software reads this data and creates a graphical display to show the state of the network.

The rest of the thesis is organized as follows. Chapter 2 gives a more detailed overview of the network. Chapter 3 reviews some of the related works. Chapter 4 discusses the technologies being used to implement this software. Chapter 5 describes the software in more details. And Chapter 6 focuses on software evaluation.

Chapter 2

Background

This project focuses on visualizing the fat-tree topology for data center networks. In order to understand the developed software, we need to review the basic concepts of this topology and its advantages over traditional topologies. In this chapter will first review the traditional topologies, and then the proposed fat-tree topology and its advantages. Then I will review the OpenFlow standard that has been used to develop this network topology. We also need to review the routing and forwarding mechanism used in order to understand how the software works.

2.1 Current Data Center Network Topologies

Data center networks are mainly based on a hierarchical, layered topology[6]. They usually consist of two or three-level trees of switches and routers. Here we will look at the three-level trees. Three-tiered designs have a core tier in the root of the tree, an aggregation tier in the middle and an edge tier at the leaves of the tree (figure 2.1).

These data center networks have some shortcomings. The main issues are that they leverage mostly non-commodity parts and are heavily oversubscribed[1].

[1]oversubscription: Ratio of the aggregate bandwidth available among the end hosts to the total bisection bandwidth of a particular communication architecture.

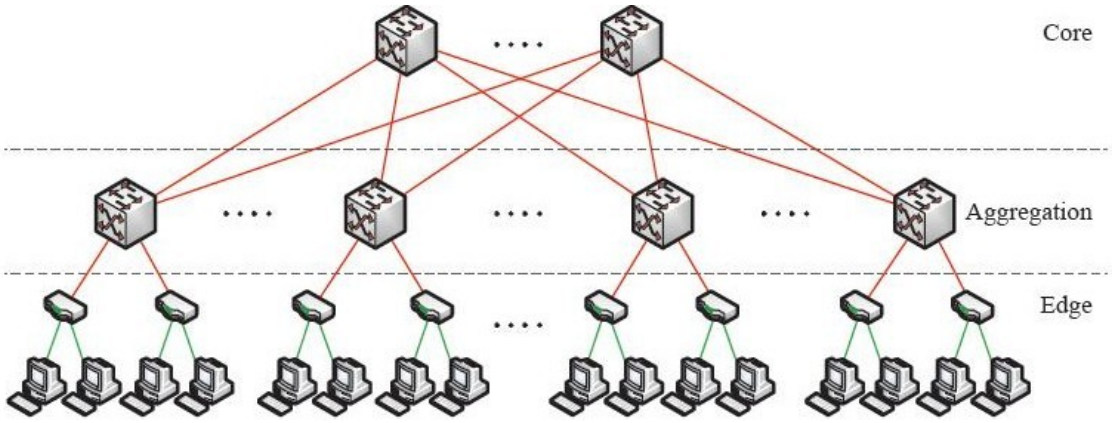

Figure 2.1: Typical three-level layered design[1].

2.2 Fat-Tree Topology

Fat-tree topology was invented to increase the efficiency of communications in networks, influenced by Clos network used by telephone industry.

The proposed data center network architecture[1] leverages strictly commodity switches organized in a fat-tree and provides full bisection bandwidth for tens of thousands of hosts.

Fat-tree consists of three layers: *Core, Aggregation and Edge*. Switches used in all layers identical k-port switches[2]. There are k pods in the tree, each containing k end hosts. Each end host is connected to one of the $k/2$ switches in edge layer. Each edge switch is connected to $k/2$ of the k ports in aggregation level switches. There are $(k/2)^2$ core switches, each one has one port connected to one of the pods. The ith port of any core switch is connected to pod i.

Figure 2.2 shows a 16-port switch built as a fat-tree topology consisting 4-port switches. This hardware has been built and used by the research group. And so it was used for the purpose of creating the visualization tool.

The main advantage of this architecture over the traditional multi-level architectures is that it uses commodity Ethernet switches to achieve scalable bandwidth for large-scale clusters at a significantly lower cost[1]. As claimed, the goal is to treat the entire data center as a single plug-and-play fabric. Using this architecture and related protocol such as PortLand data center networks can become more flexible, efficient, and fault tolerant.

2.2.1 Routing and Forwarding

In this section, I will review the routing and forwarding algorithm used in the proposed fat-tree architecture. General knowledge of these algorithms is necessary to understand the developed software.

Routing has been done using two-level look-up because to achieve full bisectional bandwidth. For each packet, the longest matching prefix algorithm is initially run on the primary table. If the hit is a terminating entry, then the

[2]Unlike the multi-level architecture described earlier that might consist of different switches and routers.

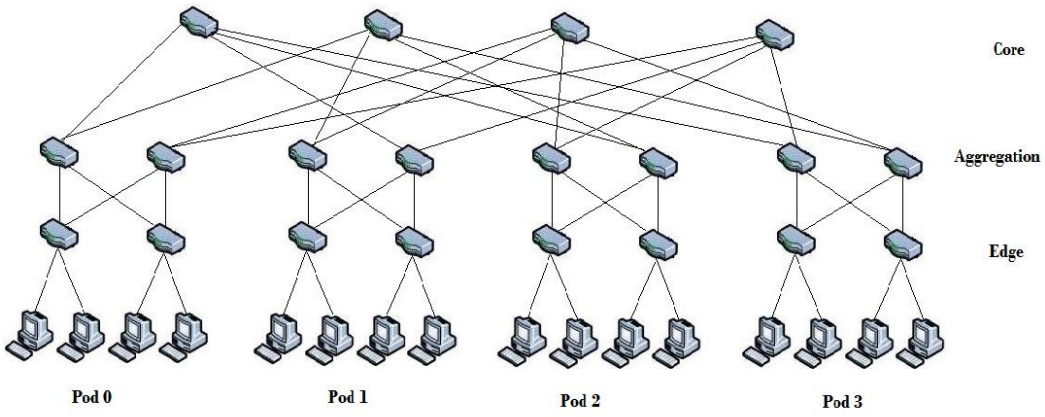

Figure 2.2: Sample fat-tree topology with 4 pods[7].

packet is forwarded to the port designated by the entry, just as with the regular IP routing mechanism. However, if the hit is a non-terminating entry, then the secondary table pointed by the entry is searched as well. The algorithm looks for the longest matching suffix in the secondary table pointed by the non-terminating entry. The packet is then forwarded to the port indicated by that entry.

2.2.2 Flow Scheduling

Flow scheduling is a mechanism to leverage global information at data center scale. It enables the fabric manager to direct flows to the least utilized paths. Connected switches are periodically polled for flow statistics. Information gathered from these switches will be send to required switches so that they can update their flow tables.

2.3 OpenFlow

OpenFlow is an open standard added to commercial Ethernet switches in order to enable researchers to run their experiments, without requiring vendors to expose the internal workings of their network devices[8].

OpenFlow has two separate functions for data path and control path. Data path is on the switch like the classical routers, but the high level routing decisions are made on a separate controller. Switch and controller communicate together by OpenFlow protocol, which defines messages, such as packet-received, send-packet-out, modify-forwarding-table, and get-stats.

The data path of an OpenFlow Switch presents a flow table abstraction[5]. Each flow table entry contains a set of packet fields to match, and an action. When an OpenFlow Switch receives a packet that it has no matching flow entries, it sends this packet to the controller. The controller then makes a decision on whether to the packet, or to add a new flow entry. The controller is responsible for determining how to handle packets without valid flow entries, and it manages the switch flow table by adding and removing flow entries. Each flow table entry contains three main parts that are as below:

Table 2.1: Fields from packets used to match against flow entries[5].

Ingress Port	Ether source	Ether dst	Ether type	VLAN id	IP src	IP dst	IP proto	TCP/ UDP src port	TCP/ UDP dst port

- Header fields to match against packets.
- Counters to update for matching packet.
- Actions to apply to matching packets.

Header fields are used in order to visualize the flows of the network. In order to understand the rest of this thesis better, we need to know about the header fields.

Table 1.1 shows the header fields that an incoming packet is compared against. These fields together create a 10-tuple that can be used to uniquely define each flow. Our visualization tool will read these fields from a database in order to distinguish all the available flows in the network. The fields in the OpenFlow 10-tuple are listed in Table 1.1 and details on the properties of each field are described in Table 1.2.

2.4 How Much Information research program

This thesis was originally part of *How Much Information* research program (http://hmi.ucsd.edu/howmuchinfo.php) in UCSD school of International Relations and Pacific Studies. This HMI research program was created to answer questions like: "what is the rate of new information growth each year?", "who produces the greatest amounts of information annually?" and "how does information growth in North America compare with growth in other geographies, markets, and people globally?" Data visualization was an important part of this research program. As mentioned in the first chapter data visualization is one of the most useful tools that can be used to help researchers understand large scale data. The

Table 2.2: Field lengths and the way they must be applied to flow entries[5].

Field	Bits	When applicable	Notes
Ingress Port		All packets	Numerical representation of incoming port.
Ethernet source address	48	All packets on enabled ports	
Ethernet destination address	48	All packets on enabled ports	
Ethernet type	16	All packets on enabled ports	A Type 0 switch is required to match the type in both standard Ethernet and 802.2 with a SNAP header and OUI of 0x000000. The special value of 0x05FF is used to match all 802.3 packets without SNAP headers.
VLAN id	12	All packets of Ethernet type 0x8100	
IP source address	32	All IP packets	Can be subnet masked
IP destination address	32	All IP packets	Can be subnet masked.
IP protocol	8	All IP packets	
Transport source port / ICMP Type	16	All TCP, UDP, and ICMP packets	Only lower 8 bits used for ICMP Type.
Transport destination port / ICMP Code	16	All TCP, UDP, and ICMP packets	Only lower 8 bits used for ICMP Code.

HMI project was defined to process and find meaningful information from massive amount of data available to the industry.

The current thesis was also defined as collaboration between the "How Much Information" program and the "Data Center Switch" program in fall 2008. The original plan was for this thesis to be part of the HMI data visualization program, but the partnership with the HMI was canceled in winter 2008.

Chapter 3

Related Works

There are many works that have attempted to reduce or eliminate human involvement in monitoring and maintaining networks. And many of them have used visualization for this purpose. The use of visualization to present information is not a new phenomenon. It has been used in maps, scientific drawings, and data plots for over a thousand years. Most of the concepts learned in devising these images carry over in a straight forward manner to computer visualization. Computer graphics has always been used to study scientific problems. With the latest technology in computer graphics, visualization has became more popular among scientists to study complex challenges. Communication network researchers are not an exception.

Network simulators have always been powerful tools for designing and studying networks behavior and adjusting design decisions. For researchers who design network protocols, simulation allows the evaluation of network protocols under varying network conditions, and especially under interaction with other protocols. For network traffic researcher, simulation is critical as it allows them to recreate problematic points on the network and study the cause of problems. Security researchers use visualization to generate a simulation of an attack in order to understand the source and cause.

Many tools have been developed that attempt to reduce or eliminate human involvement in network monitoring. Visualization tools have been mostly focusing on network security issues, while our work here focuses on visualization for the

specific data center network proposed by our research group.

3.1 VINT

VINT is attempting to facilitate the design and deployment of new wide area Internet protocols by providing network researchers with an improved set of simulation tools[2]. The VINT project, through the ns simulator and related software, provides several critical innovations that broaden the range of conditions under which protocols can be evaluated while making this experimentation tractable. Abstraction, emulation, scenario generation, visualization and extensibility are the main characteristic of VINT.

The VINT project, using ns as its simulator base and NAM as its visualization tool, has constructed a common simulator containing a large set of models for use in network research. By including algorithms still in the research phase of development, users of the simulator are able to explore how their particular work interacts with these future techniques. Furthermore, because of the many protocols and models included with the system, researchers are often able to modify and construct their own simulations based on the provided models with relative ease. In several cases, modules developed outside the VINT project have been incorporated as a standard component to the simulator.

3.2 NAM: Network Animator

Nam, the network animator that was developed at the VINT project, provides packet-level animation, protocol graphs, time-event plots of protocol actions, and scenario editing capabilities[3]. NAM can collect detailed protocol information from a trace file created by a simulator like ns or can use processing data from a live network to produce a trace. It usually runs offline by reading the trace file from the disk but can also play traces from a running program through a UNIX pipe.

The trace file contains static network layout and dynamic events such as

packet arrivals, departures, and drops and link failures. The input file for wireless networking simulations also includes information on node location and movement.

3.3 Microsoft Visual-I

Visual-I has been designed for the management of networks and applications inside large data centers by Microsoft Visualization and Interaction (VIBE) Research Group[4]. It was designed to ease the visualization of logical structures with information overlays that leverage the structure to help users correlate data. Visual-I provides a single view of the entire cloud service, as a network. It shows high-level objects as a single unit. The images are animated, with decorations and displays such as CPU load and activity information updating as the data changes. It also let past behavior to be replayed. This helps operators correlate the overlaid data across time to discover recurring and time-dependent behaviors. This feature of Visual-I is similar to our work, as we let the operator to run the system for a specific time period.

Chapter 4

Technology Review

Like any other software implementation, while starting to design this software I made some technology choices based on the project requirements, future directions and compatibility with the rest of the group. In this chapter I will explain each of these technologies, reasons that I decided to use them, and their benefits. Although my choices seemed to be the best possibilities, they were not completely convenient for the project, so I will also explain their shortcomings.

This software was developed using Java OpenGL (JOGL), which is a wrapper library that allows OpenGL to be used in the Java programming language. It allows use of object-oriented tools for Java with hardware-accelerated 2D and 3D graphics.

Data, captured from the network, is recorded in PostgreSQL database, which is an open source ORDBMS, and read by the program in order to visualize the network current state. In this chapter I will first review OpenGL and JOGL, theirs advantages over other graphic libraries, and then I will describe the PostgreSQL database and its advantages.

4.1 OpenGL

Open Graphics Library is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface has many function calls which can be used to draw com-

plex three-dimensional scenes from simple primitives. OpenGL is widely used in scientific visualization and information visualization. Prior to OpenGL programmers had to develop their graphical applications for each individual platform, and had to have knowledge of the graphic hardware. Now applications can create the same result in different operating systems because of OpenGLs graphic adaptor.

OpenGL has two main purposes; first it hides the complexities of interfacing with different 3D accelerators, by presenting a single, uniform interface. Second it hides the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set.

OpenGL will accepts primitives (points, lines, polygons) and convert them to pixels. There are two ways of creating commands. Programmer can create a list of commands or immediately executed functions. Each command directs a drawing action or causes special effects. A list of these commands can be created for repetitive effects. This is one of the more routine ways that the program interface is used. But it is also possible to create and execute one time commands within the perimeters of the computer graphics as well.

4.2 Java OpenGL - JOGL

Java OpenGL is a Java package providing bindings to the OpenGL libraries for the Java Virtual Machine. It allows programmers to use the object-oriented tools for Java with hardware accelerated two or three dimensional graphics while leveraging their existing knowledge of OpenGL. Hardware accelerated graphics helps to improve user perceived performance in Java programs with heavy graphics requirements.

Java OpenGL allows access to most features available to C programming language programmers, with a few exceptions like the window-system related calls in GLUT, and some extensions. The base OpenGL C API is accessed in JOGL via Java Native Interface calls. As such, the underlying system must support OpenGL for JOGL to work.

JOGL, which is used in this project, is different from some other Java

OpenGL wrappers as it exposes the procedural OpenGL API via methods on a few classes, rather than attempting to map OpenGL functionality onto the object oriented programming paradigm. The majority of the JOGL code is automatically generated from the OpenGL C header files. The procedural and state machine nature of OpenGL is inconsistent with the typical method of programming under Java. However, the straightforward mapping of the OpenGL C API to Java methods makes conversion of existing C applications and example code much simpler.

The layer of abstraction provided by JOGL makes runtime execution quite efficient, but is more difficult to code compared to higher-level abstraction libraries like Java3D. Because most of the code is automatically generated, changes to OpenGL can be rapidly added to JOGL.

Although the project could have been done without JOGL, by using more light-weighted libraries, this design decision was made to make future adjustment easier. One of which could be changing it to support three dimensional visualization. The current software only supports two dimensional, but creating the three dimensional version is also possible due to my technology choice. Choosing to program using JOGL was more time consuming as I had no prior experience with this wrapper or any other graphic libraries, and I had to spend some time at the beginning to learn the basic concepts. But the trade off was creating a platform that could support future advanced development.

Despite the advantages JOGL and in general OpenGL provide to this project, they had some shortcomings. OpenGL is a low-level, procedural API which closely matches many of the algorithms and methods graphics programmers have used historically. But it means that the programmer have to dictate the exact steps required to render a scene; as oppose to descriptive APIs, where a programmer only needs to describe a scene and can let the library manage the details of rendering it. This means I had to have a good knowledge of the graphics pipeline.

Although the procedural approach to graphics is an important strength of OpenGL, it simultaneously is a weakness for many Java programmers like me. I have always used Java using object-oriented methodologies, and OpenGL's procedural method does not mesh well with object-oriented approaches and good

engineering practice. For these reasons the learning process was quite time consuming.

As mentioned before, one of my concerns for choosing to program with JOGL was to make the software more flexible for future changes. Despite giving the project this advantage, my technology choice seemed to be an extra burden for some part of the project. In some phases of implementing the software, I found JOGL unnecessary and extremely heavy.

With such trade-off between development speed and flexibility in mind, I chose to stay with JOGL, and gain the required knowledge and make the software a better platform for future improvement.

4.3 PostgreSQL

PostgreSQL is an open source object-relational database management system (ORDBMS) that supports a large part of the SQL standard and offers many modern features such as complex queries, foreign keys, triggers, views, transactional integrity and multi-version concurrency control. It can also be extended by the user in many ways, especially by taking advantage of its support for user-defined data types.

PostgreSQL works as a client/server model. Each session consists of a server process and the users client application. The server process manages the database files, accepts connections to the database, and performs database actions on behalf of the clients. Users application performs database operations. Communication between the server and client (if on different hosts) is handled over TCP/IP network connection. The server can handle multiple concurrent connections from clients.

In general, PostgreSQL is a fast and mature database that provides reliability, stability, scalability and extensibility. Besides these general advantages we decide to use it in our project for several reasons: First, and probably most important, it is open source; and as we were using it for research purposes it provides us with flexibility of customization. Second, PostgreSQL is cross-platform and supports all major operating systems (Windows, Linux, Mac OS X, UNIX). The

visualization tool is designed on Windows, but the rest of our group use mainly Linux. Third, user-defined data types are supported which was a key feature in our database tables. Last, there are a number of PostgreSQL-specific ties which make migration to other databases a non-trivial process, so we have the flexibility to change our design decision later.

Chapter 5

Design and Implementation

In this chapter, I will explain the implementation of the software in more details. I will first discuss how the data is captured from the network and saved to the database, how the software recognize individual flows from the database and how the data is being visualized, then I will explain the software structure, classes and functions.

The complete code has been attached in Appendix A. The code has two basic functionalities: finding and processing the data, drawing and animating the data.

5.1 Capturing Data from the Network

This part of the project was done by other researchers from the DCSwitch research team. Nelson Huang added required functionalities to the core code of the project. This code captures every flow that is created in the network. This data is then recorded to the database, which will be discussed later. My software later reads this data from the database.

5.2 Recognizing Flows from Database

As mentioned in section 2.3, OpenFlow has a 10-tuples that can be used to uniquely identify individual flows in the network. In order to recognize active

flows the database is searched for entries with same OpenFlow 10-tuples (more details in *DBOperation* class), and then the path will be calculated based on the source and destination end hosts and the upward switches. Each flow entry in the database has the sender id and next switch id, based on this information the upward path will be recognized. There is always only one downward path from each core switch to each end host, so the downward path will be recognized based on the core switch and receiver. During my development process there were some issues with the way data was captured to the database; one of these issues was redundant entries. These redundant entries will be dropped while the path is being recognized.

5.3 How to Visualize?

One of the main challenges of this project was to find the best way to visualize the network. My main choices were between using graphs to show statistic of the network, using three dimensional presentations, or using the two dimensional image that has been used in the research papers to represent the network (figure 2.2). Since the project was done during the design phase of the network and the research group was already using this image everywhere as the network representation I decided to go with the same choice. My software now draws the network as shown in the figure 2.2 (screen shot shown in figure 5.1) and while on the animation mode, it shows the movement of data on the network by changing the color of active components (screen shots shown in figures 5.2 and 5.3). Red lines will be drawn over the cables while they are being used to transfer data from one point to another. The thickness of these red lines shows the portion of the bandwidth that is being used.

5.4 Software Structure

As mentioned before Java OpenGL was used to program this project. There are two main parts to the software, one part deals with accessing the database,

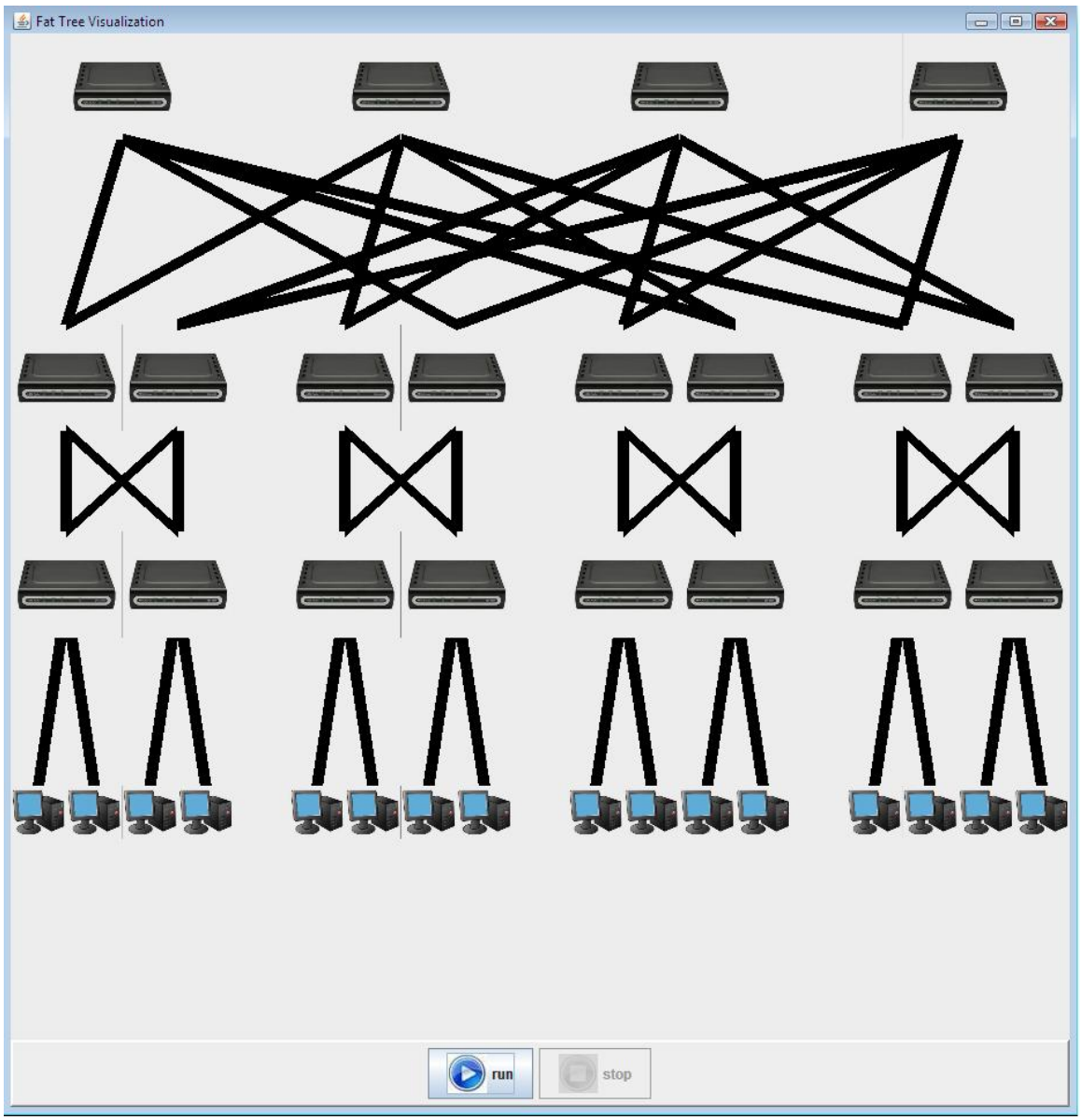

Figure 5.1: Screen shot of the software.

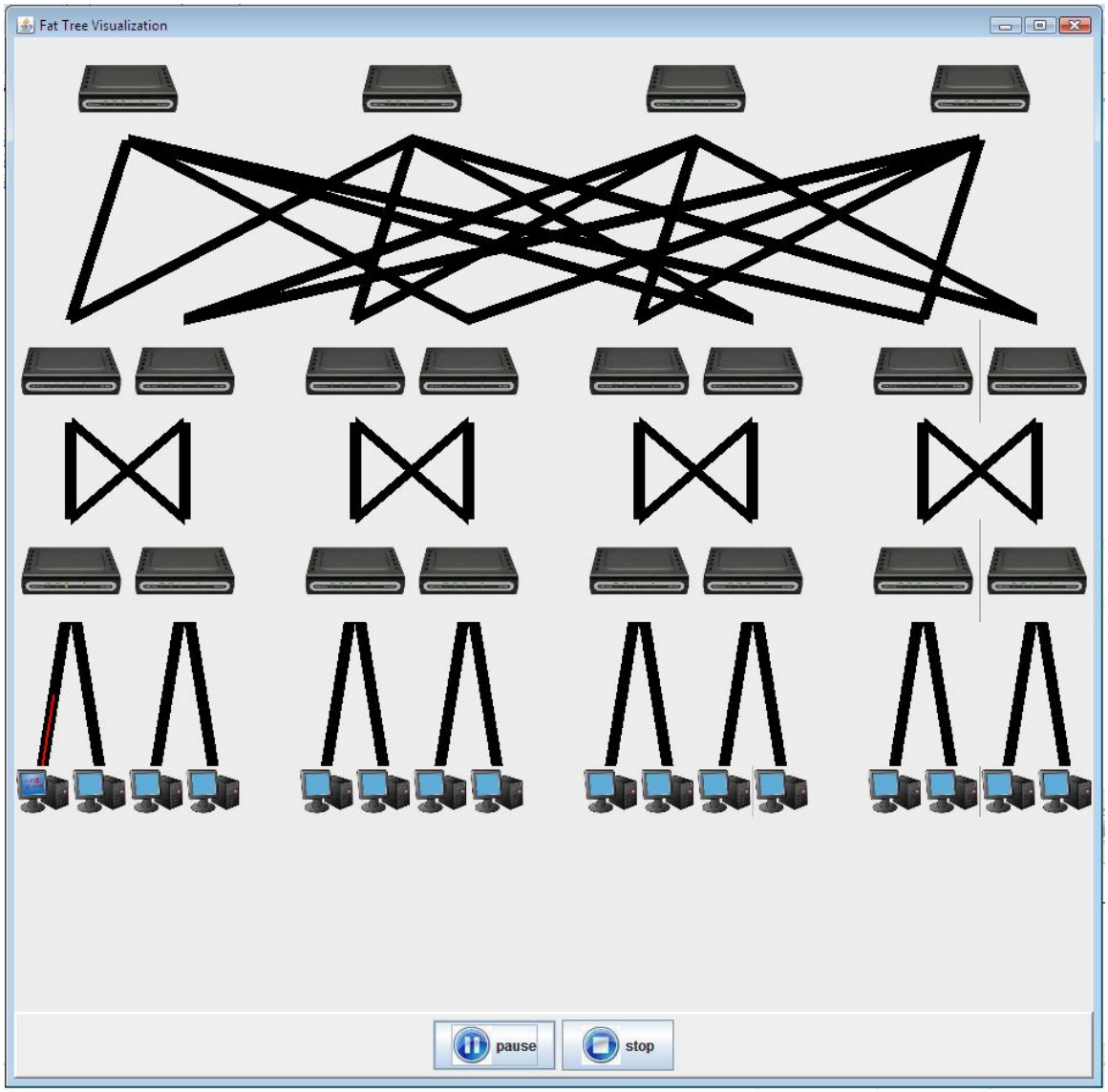

Figure 5.2: Screen shot of the software in the animation mode.

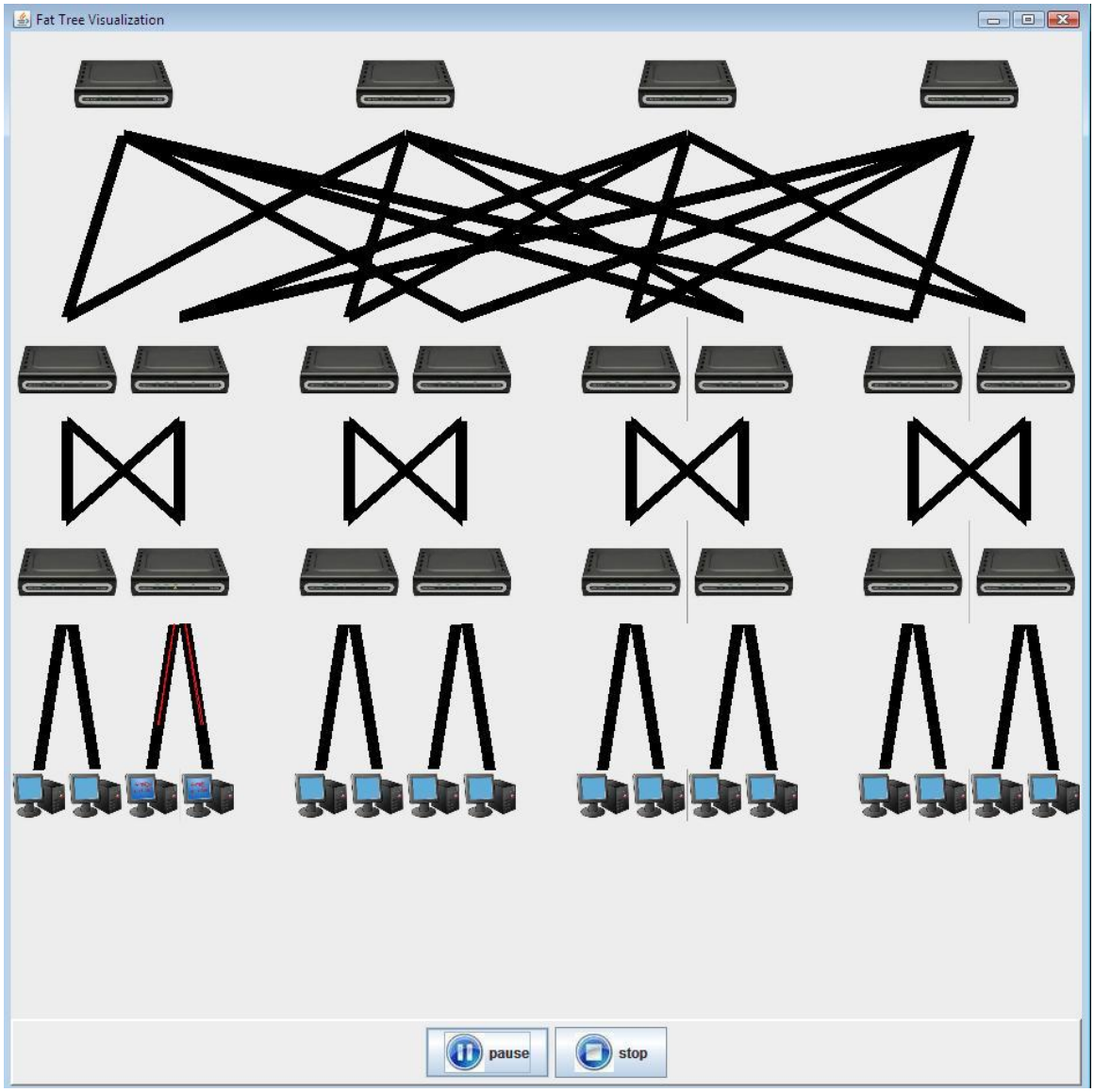

Figure 5.3: Screen shot of the software in the animation mode.

processing the data, storing the data in to appropriate data structures and finally updating the state of the software. The other part is in charge of visualizing the data captured from the database and running the animation.

This software contains four main classes: *MainWindow*, *Node*, *FlowData* and *DBOperation*. These classes contain the main functionality and logic of the software. I will describe theses classes in more details in the next section.

StreightLine, *Texture*, *TextureManager* and *Toolbar* classes contain the drawing aspect of the software. *StreightLine* draws the cables of the network. The others are common JOGL classes and discussing them in more details are beyond the scope of this thesis.

5.5 More Details

Node Class

A *Node* could be both a switch and an end host in the network. This class has been created to structure the distinction between different item of the network based on their role, their pod number and their placement in the pod.

The *Node* class has the following attributes:

- desc (description): it has one of the following values:
  - aggr: node is an aggregation switch.
  - edge: node is an edge switch.
  - core: node is a core switch.
  - host: node is an end host.
- podNum: Which pod does the node belong to?
- index: placement of the node in the pod. This will be between 0 and 3 for end hosts and core switches. And 0 or 1 for edge and aggregation switches.
- selected: this item is for further development.

- active: showing if the node is sending or receiving data.
- degree: showing how many active connection the node has.

FlowData Class

For every active flow in the network one instance of *FlowData* class will be created. This class keeps all the specific information about the flow including the OpenFlow 10-tuple, all the switches envolved in the path from source to destination, flow duration, packet count, byte count and all the critical timestamps.

Each FlowData contains one *FlowTuple* that includes the OpenFlow 10-tuple information. Each *FlowTuple* has the following attributes:

- inPort: Incoming port.
- dlVlan: VLAN id
- dlType: Ethernet type
- nwSrc: Ethernet source
- nwDst: Ethernet destination
- nwProto: IP protocol
- tpSrc: TCP/UDP source
- tpDst: TCP/UDP destination
- dlSrc:
- dlDst

The path that each flow travels is recognized by the following variables:

- srcHostIndex: Index number of the source host.
- dstHostIndex: Index number of the destination host.
- upEdgeSwitch: Index number of the edge switch of the upward path.

- upAggrSwitch: Index number of the aggregation switch of the upward path.
- coreSwitch: Index number of the core switch.
- downAggrSwitch: Index number of the aggregation switch of the downward path.
- downEdgeSwitch: Index number of the edge switch of the downward path.

One of the important functions of this class is *isPathBroken*. This function checks that the path that has been assigned to the flow is valid; it checks that it has all the switches, and that if the sender and receiver are not on the same pod then there is a core switch available to the path. This function was created because during the development there were some invalid entries in the database. So I added this function to make sure all the flows that I find on the network are valid and I can show the complete path.

DBOperation

This class takes care of all the database operations. The basic operations are making the connection to the database, receiving the flow data by different attributes, finding the oldest and newest flow and finding the switch index by using the switch id.

The *getFlowData* function comes with three different signatures. One will query all the flows from the database, second one only selects data between two given timestamps and the last one will select flows after a specific time.

The *getFlowDataFromResultSet* function is called by all of the *getFlowData* functions to process the result set returned by the database and it outputs an array of FlowData. As mentioned before, there are currently some redundant entries in the database because of some capturing errors. The getFlowDataFromResultSet function drops these redundant entries, and makes sure that only valid FlowData are sent back to the calling functions.

The *getMinDate* and *getMaxDate* functions return the newest and oldest flow entries of the database.

Besides the main database table, there is another table called *dcswitch graph nodes*. This table holds the id of all switches in the network and assigns them to their location. The *findSwitchIndex* functions query this table with the switch id and return a *Node* that contains the description, pod number and index number of the requested switch. The main purpose of this table is to make it easier for future changes, since the switch ids will change based on the used hardware.

MainWindow Class

This class contains the *main* method, it first read *config.properties* file in order to find out about the network architecture. This file includes all the essential information about the network like how many pods are on the network, how many layers the network has, how many end hosts are on each pod and how many aggregation and edge switches are on each pod. Although during the development process we only used a system with four pods and three layers, this file was created for future changes and to make the software more flexible for design changes. When this file is read, we can draw the appropriate network.

When the network is recognized one object will be created for every node on the network as well as every cable. This is because we want to be able to interact with each of these components individually. Also one feature that might be added in the future is making each component selectable and just visualizing the data related to that component.

The next step is creating the connection to the database. While the connection is established a call to *getFlowData* function will return all the valid flows from the database. These flows will be saved in an array of *FlowData*, ready to be displayed.

The following part of the code focuses on setting up the Java OpenGL requirements. The canvas is initiated here, and the drawing frame, panels, content pane and other required drawing components are all set up here.

init function initiates the objects for the network components. The followings are array of *Nodes*:

- endHosts: all the end hosts.

- edAgSwitches: all aggregation and edge switches.
- coreSwitches: all the core switches.

Six arrays of integers will be initiated for all the cables on the network. The integers represent how many flows are traveling over the cable.

- upLinkActive_1: all upward cables between end hosts and edge switches.
- upLinkActive_2: all upward cables between edge switches and aggregation switches.
- upLinkActive_3: all upward cables between aggregation switches and core switches.
- downLinkActive_1: all downward cables between end hosts and edge switches.
- downLinkActive_2: all downward cables between edge switches and aggregation switches.
- downLinkActive_3: all downward cables between aggregation switches and core switches.

drawHost, *drawEdgeAggregation* and *drawCore* functions are called by the *drawScene* function, they read the property file and draw the component on the desired location on the screen.

updateAnimation function checks the status of the animation. The animation speed is set by *frame_rate* variable. If one round of animation is done, this function will check if the current round of the animation is done, and reset the variables if it is. If it is not, then it will change the variables for the next step of the animation. This updates basically involve changing the colors of the active components.

setupFlowInformation reads the flow data received from the database and sets up the initial state of the required components to active, this includes the sender end host, the link from the end host to the edge switch, and finally the edge switch.

findAggCoreLink function gets one aggregation and one core switch and returns an integer that indicates the index of the link between them, if any. Similarly *findEdgeAggLink* function gets one edge and one aggregation switch and returns an integer that indicates the index of the link between them.

Chapter 6

Future Works

In this chapter I will explain some of the ideas that I had about the future works of the projects. The main ideas includes: adding the ability to work real time, making the network components selectable, running the animation for a selected period of time or selected part of the network and showing statistical graphs.

Real time animation

The available software queries the database, finds and processes the available flows and animates them regardless of their timestamps. Obviously one very useful addition to the software would be making the animation to run on real time. It was not possible to develop during the time of this project mainly because of the way data was captured from the hardware to the database. This feature could be particularly very useful for the network monitoring, the network operator could use this tool to monitor the network for accuracy or too look for any fault congestion or any other misbehavior in the network.

Although this could be a very valuable addition to the software, the software was not designed initially to support it. One main obstacle in developing to develop this feature is the current use of database. Saving the flows to the database and then reading them back by the software is not a practical solution for the real time animation; it would be more useful to connect the software directly to the

hardware.

Animation for specific time period

As I explained in the previous chapter, this software now only animates all the flows in the network at the same time. But there are two functions that can query the database for specific time period. These functions have been developed, but not used in the software.

This feature could be very useful if the network operator or designer wants to know what had happened in the network during a period of time. This feature does not need lots of work to be done since the database functions already exist. The only development necessary for this feature is adding the GUI required for selecting the timestamps, and calling the appropriate function.

Animation for specific components

The software as available now animates the entire flows available to the database, regardless of senders, receivers or any other parts of the paths. One extra feature that could be developed is to let the users select which component of the network they want to focus on and only show the flows traveling to/from that component.

The hardest part of adding this feature is dealing with the zoning of the screen and making the zones clickable. This part has already been done; every node on the network is now selectable. Next step is querying the database for those specific flows and displaying them. This feature also needs to be associated with the previous one since the user may only needs to see traffic from one component in a specific time period.

Showing statistical graphs

This project started as visualization software, so there was no focus on displaying any kind of statistical graphs. Having graphs could be useful for any network management, monitoring tool. The types of graphs that might be useful

for this project are: traffic rate for cables, connection rate between two end hosts and the path that has been taken for these connections.

The visualization software could provide a right click drop-down menu for each component of the network and let the users to choose the criteria they want to see on a graph and the types of graphs they want to see.

Chapter 7

Appendix: Source Code

The maincode:

```
public class MainWindow implements GLEventListener {
    protected static int pod_num;
    protected Texture host_tex;
    protected Texture host_s_tex;
    protected Texture switch_tex;
    protected Texture switch_s_tex;
    protected Texture cable_tex;

    protected GLU glu;
    protected static Point pickPoint;
    protected Hashtable<Integer, Node> namedNodes;
        // drawables indexed

    public static Properties props = new Properties();
    protected static Properties db_props = new Properties();

    protected static int hosts_per_pod;
    protected static int edge_per_pod;
    protected static int number_layer;
```

```
protected static Node endHosts[];
protected static Node edAgSwitches[];
protected static Node coreSwitches[];
protected static int upLinkActive_1[];
    // links between end hosts and edge switches going upward
protected static int upLinkActive_2[];
    // links between edge and aggregation switches going upward
protected static int upLinkActive_3[];
    // links between aggregation and edge switches going upward
protected static int downLinkActive_1[];
    // links between end hosts and edge switches going downward
protected static int downLinkActive_2[];
    // links between edge and aggregation switches going downward
protected static int downLinkActive_3[];
    // links between aggregation and edge switches going downward

protected static FlowData[] currFlowData;

protected static double hostStartY;
protected static double edgeStartY;
protected static double aggrStartY;
protected static double coreStartY;
protected static double hostEndY;
protected static double edgeEndY;
protected static double aggrEndY;
protected static double coreEndY;

protected static int frame_rate;
protected static int frame_count;
protected static final int STOPPED = -1;
protected static final int RUNNING = 0;
```

```
protected static final int PAUSED = 1;

protected static int activityStatus;

protected static Timestamp minTimestamp;
protected static Timestamp maxTimestamp;
protected static long current_time;

protected static JLabel minTimeLabel;
protected static JLabel maxTimeLabel;
protected static JFrame frame;
protected static GLCanvas canvas;
protected static Animator animator;
protected static JPanel runPanel;
protected static JButton runButton;
protected static JButton stopButton;

protected static String runText;
protected static String stopText;
protected static String pauseText;
protected static String minTimeText;
protected static String maxTimeText;

protected static ImageIcon runIcon;
protected static ImageIcon pauseIcon;
protected static ImageIcon stopIcon;

protected static boolean flowDisplayDone[];

public static void main(String[] args) throws Exception {
```

```
pod_num = 4;
props.load(new FileInputStream("test"));

hosts_per_pod = Integer.parseInt(
        props.getProperty("HOSTS_PER_POD"));
edge_per_pod = Integer.parseInt(
        props.getProperty("EDGE_AGG_PER_POD"));
number_layer = Integer.parseInt(
        props.getProperty("NUMBER_OF_LAYER"));

DBOperation.creatConnection();
currFlowData = DBOperation.getFlowData();
int len = currFlowData.length;
flowDisplayDone = new boolean[len];

init();

frame_rate = Integer.parseInt(
        props.getProperty("FRAME_RATE"));
activityStatus = STOPPED;

canvas = new GLCanvas();
canvas.setBackground(new Color(0.93f, 0.93f, 0.93f));
canvas.addGLEventListener(new MainWindow());
canvas.addMouseListener(new MouseListener() {
    public void mouseClicked(MouseEvent e) {
    }
    public void mouseEntered(MouseEvent e) {
    }
    public void mouseExited(MouseEvent e) {
    }
```

```
    public void mousePressed(MouseEvent e) {
        if(e.getButton() == MouseEvent.BUTTON1) {
        pickPoint = e.getPoint();
        canvas.display();
        }
    }
    public void mouseReleased(MouseEvent e) {
    }
});

frame = new JFrame("Fat Tree Visualization");
frame.add(canvas);
animator = new Animator(canvas);
frame.setSize(1000, 1000);
frame.setVisible(true);
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        animator.stop();
        System.exit(0);
    }
});

runText = props.getProperty("RUN_BUTTON_TEXT");
pauseText = props.getProperty("PAUSE_BUTTON_TEXT");

runPanel = new JPanel();
runButton = new JButton(runText);
runIcon = new ImageIcon(
        props.getProperty("RUN_ICON_FILE"));
pauseIcon = new ImageIcon(
        props.getProperty("PAUSE_ICON_FILE"));
```

```
runButton.setIcon(runIcon);
runButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) {
        JButton source = (JButton) e.getSource();
        switch(activityStatus) {
        case STOPPED:
        activityStatus = RUNNING;
        frame.getRootPane().setCursor(null);
        source.setText(pauseText);
        source.setIcon(pauseIcon);
        stopButton.setEnabled(true);
        try {
        setupFlowInformation();
        } catch (Exception e1) {
        e1.printStackTrace();
        }
        if(!animator.isAnimating()) animator.start();
        break;
        case RUNNING:
        activityStatus = PAUSED;
        frame.getRootPane().setCursor(null);
        source.setText(runText);
        source.setIcon(runIcon);
        animator.stop();
        break;
        case PAUSED:
        activityStatus = RUNNING;
        source.setText(pauseText);
        source.setIcon(pauseIcon);
        if(!animator.isAnimating())animator.start();
        break;
```

```
        }
    }
});

stopText = props.getProperty("STOP_BUTTON_TEXT");
stopIcon = new ImageIcon(props.getProperty("ST_ICON_FILE"));
stopButton = new JButton(stopText);
stopButton.setIcon(stopIcon);
stopButton.setEnabled(false);
stopButton.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) {
        JButton source = (JButton) e.getSource();
        if(activityStatus != STOPPED) {
        activityStatus = STOPPED;
        runButton.setText(runText);
        runButton.setIcon(runIcon);
        source.setEnabled(false);
        init();
        if(!animator.isAnimating()) animator.start();
        }
    }
});

runPanel.setAlignmentX(Component.CENTER_ALIGNMENT);
runPanel.setMaximumSize(
        new Dimension(Integer.MAX_VALUE, 20));
runPanel.setBorder(new BevelBorder(BevelBorder.RAISED));
runPanel.add(runButton);
runPanel.add(stopButton);

frame.getContentPane().add(runPanel, BorderLayout.SOUTH);
```

```
}

public void init(GLAutoDrawable drawable) {
    glu = new GLU();
    try {
        host_tex = TextureManager.instance().newTexture(
            props.getProperty("HOST_IMAGE_FILE"),
            drawable.getGL());
        host_s_tex = TextureManager.instance().newTexture(
            props.getProperty("ACTIVE_HOST_IMAGE_FILE"),
            drawable.getGL());
        switch_tex = TextureManager.instance().newTexture(
            props.getProperty("SWITCH_IMAGE_FILE"),
            drawable.getGL());
        switch_s_tex = TextureManager.instance().newTexture(
            props.getProperty("ACTIVE_SWITCH_IMAGE_FILE"),
            drawable.getGL());
    } catch (IOException e) {
        e.printStackTrace();
    }
    namedNodes = new Hashtable¡Integer, Node¿();
}

private static void init() {
    int len = currFlowData.length;
    endHosts = new Node[pod_num*hosts_per_pod];
    edAgSwitches = new Node[pod_num*edge_per_pod*2];
    coreSwitches = new Node[pod_num*1];
    upLinkActive_1 = new int[pod_num*hosts_per_pod];
    upLinkActive_2 = new int[pod_num*edge_per_pod*2];
    upLinkActive_3 = new int[pod_num*edge_per_pod*2];
```

```
        downLinkActive_1 = new int[pod_num*hosts_per_pod];
        downLinkActive_2 = new int[pod_num*edge_per_pod*2];
        downLinkActive_3 = new int[pod_num*edge_per_pod*2];

        for(int i = 0; i < pod_num*hosts_per_pod; i++) {
            upLinkActive_1[i] = 0;
            downLinkActive_1[i] = 0;
        }
        for(int i = 0; i < pod_num*edge_per_pod*2; i++) {
            upLinkActive_2[i] = 0;
            upLinkActive_3[i] = 0;
            downLinkActive_2[i] = 0;
            downLinkActive_3[i] = 0;
        }
    }

    public void reshape(GLAutoDrawable drawable, int x,
                int y, int width, int height) {
        GL gl = drawable.getGL();
        gl.glViewport(0, 0, width, height);
        gl.glMatrixMode(GL.GL_PROJECTION);
        gl.glLoadIdentity();
        glu.gluOrtho2D(0.0, 1.0, 0.0, 1.0);
        gl.glMatrixMode(GL.GL_MODELVIEW);
        gl.glLoadIdentity();
    }

    public void display(GLAutoDrawable drawable) {
        updateAnimation();
        GL gl = drawable.getGL();
        gl.glClear(GL.GL_COLOR_BUFFER_BIT
```

```
            | GL.GL_DEPTH_BUFFER_BIT);
        if(pickPoint != null) selectObj(gl);
        drawScene(gl, GL.GL_RENDER);
        gl.glFlush();
}

void drawScene(GL gl, int mode) {
        int count = 0;
        gl.glColor3f(0.93f, 0.93f, 0.93f);
        gl.glClearColor(0.93f, 0.93f, 0.93f, 0.93f);
        gl.glClear(GL.GL_COLOR_BUFFER_BIT
            | GL.GL_DEPTH_BUFFER_BIT);

        gl.glEnable(GL.GL_TEXTURE_2D);
        gl.glTexEnvf(GL.GL_TEXTURE_ENV,
            GL.GL_TEXTURE_ENV_MODE, GL.GL_MODULATE);

        double portion = (double)1/(pod_num*hosts_per_pod+pod_num-1);
        double space = (double)(1-(pod_num*hosts_per_pod*portion))
            /(pod_num-1);
        for(int i = 0; i < pod_num*hosts_per_pod; i++) {
            if(endHosts[i] == null) {
            int pod = i/hosts_per_pod;
            int ndx = i % hosts_per_pod;
            endHosts[i] = new Node(Node.END_HOST, pod, ndx);
            }
        boolean active = (endHosts[i].isActive())?true:false;
            drawHost(gl, endHosts[i], active, space, portion);
            if(mode == GL.GL_SELECT) {
                gl.glLoadName(count);
                namedNodes.put(count, endHosts[i]);
```

```
            count++;
        }
    }

    double portion_sw = (double)(1-(pod_num-1)*space)/
        (pod_num*edge_per_pod);
    for(int i = 0; i < pod_num*edge_per_pod*2; i++) {
        if(edAgSwitches[i] == null) {
            int pod = i/(edge_per_pod*2);
            int row = (i-(edge_per_pod*2*pod))/2;
            int ndx = i%(edge_per_pod*2);
            String str = (row==0)?Node.EDGE_SW:Node.AGGR_SW;
            edAgSwitches[i] = new Node(str, pod, ndx);
        }
        boolean active = (edAgSwitches[i].isActive())?true:false;
        drawEdgeAggregation(gl, edAgSwitches[i], active, space,
            portion, portion_sw);
        if(mode == GL.GL_SELECT) {
            gl.glLoadName(count);
            namedNodes.put(count, edAgSwitches[i]);
            count++;
        }
    }

    for(int i = 0; i < pod_num; i++) {
        if(coreSwitches[i] == null)
            coreSwitches[i] = new Node(Node.CORE_SW, i, 0);
        boolean active = (coreSwitches[i].isActive())?true:false;
        drawCore(gl, coreSwitches[i], active, space, portion, portion_sw);
        if(mode == GL.GL_SELECT) {
            gl.glLoadName(count);
```

```
            namedNodes.put(count, coreSwitches[i]);
            count++;
        }
    }

    double space_between = Double.parseDouble(
        props.getProperty("SPACE_BETWEEN_OVERLAP_LINKS"));
    double h_pos = Double.parseDouble(
        props.getProperty("HOST_Y_POS"));
    double e_pos = Double.parseDouble(
        props.getProperty("EDGE_Y_POS"));
    double a_pos = Double.parseDouble(
        props.getProperty("AGGR_Y_POS"));
    double c_pos = Double.parseDouble(
        props.getProperty("CORE_Y_POS"));

    for(int i = 0; i < pod_num*hosts_per_pod; i++) {
        int pod = i/hosts_per_pod;
        gl.glLineWidth(10.0f);
        gl.glBegin(GL.GL_LINES);
        gl.glColor3f(0f, 0f, 0f);
        double startX = pod*space+i*portion+portion/2;
        double endX = pod*space+((i%2==0)?i+1:i)
            *portion+((i%2==0)?-1:1)*space_between;
        double midX = 0.01;
        gl.glLoadName(count);
        count++;
        gl.glVertex2d(pod*space+i
            *portion+portion/2, portion+h_pos);
        gl.glVertex2d(pod*space+((i%2==0)?i+1:i)
            *portion+((i%2==0)?-1:1)*space_between, e_pos);
```

```
gl.glEnd();
if(upLinkActive_1[i] > 0) {
   gl.glColor3f(1f, 0f, 0f);
   gl.glLineWidth(upLinkActive_1[i]);
   gl.glBegin(GL.GL_LINES);
   StreightLine sl = new StreightLine(
   pod*space+i*portion+portion/2, portion+h_pos,
   pod*space+((i%2==0)?i+1:i)
   *portion+((i%2==0)?-1:1)*space_between, e_pos);
   double h = sl.height/frame_rate;
   double X = pod*space+i*portion+portion/2;
   double Y = portion+h_pos;
   for(int jj = 0; jj < frame_count; jj++) {
   X = sl.getXCordByHeight(X, Y, h);
   Y = sl.getYCordByHeight(X, Y, h);
   }
   gl.glVertex2d(pod*space+i*portion+portion/2,
   portion+h_pos);
   gl.glVertex2d(sl.getXCordByHeight(X, Y, h),
   sl.getYCordByHeight(X, Y, h));
   gl.glEnd();
}
if(downLinkActive_1[i] > 0) {
   gl.glColor3f(1f, 0f, 0f);
   gl.glLineWidth(downLinkActive_1[i]);
   gl.glBegin(GL.GL_LINES);
   StreightLine sl = new StreightLine(
   pod*space+i*portion+portion/2, portion+h_pos,
   pod*space+((i%2==0)?i+1:i)*portion+
   ((i%2==0)?-1:1)*space_between, e_pos);
   double h = -1*sl.height/frame_rate;
```

```
        double X = (pod*space+((i%2==0)?i+1:i)
        *portion+((i%2==0)?-1:1)*space_between);
        double Y = e_pos;
        for(int jj = 0; jj < frame_count; jj++) {
        X = sl.getXCordByHeight(X, Y, h);
        Y = sl.getYCordByHeight(X, Y, h);
        }
        gl.glVertex2d(pod*space+((i%2==0)?i+1:i)
        *portion+((i%2==0)?-1:1)*space_between, e_pos);
        gl.glVertex2d(sl.getXCordByHeight(X, Y, h),
        sl.getYCordByHeight(X, Y, h));
        gl.glEnd();
    }
}

for(int i = 0; i < pod_num*edge_per_pod*2; i++) {
    int pod = i/(edge_per_pod*2);
    int col = i/edge_per_pod;
    gl.glColor3f(0f, 0f, 0f);
    gl.glLineWidth(10.0f);
    gl.glBegin(GL.GL_LINES);
    gl.glLoadName(count);
    count++;
    gl.glVertex2d(pod*space+(edge_per_pod*col+1)
        *portion, e_pos+portion_sw);
    gl.glVertex2d(pod*space+(edge_per_pod*((i%2==0)
        ?col:((i%4==1)?col+1:col-1))+1)*
        portion, e_pos+portion_sw);
    gl.glEnd();
    if(upLinkActive_2[i] > 0) {
        gl.glColor3f(1f, 0f, 0f);
```

```
    gl.glLineWidth(upLinkActive_2[i]);
    gl.glBegin(GL.GL_LINES);
    StreightLine sl = new StreightLine(
    pod*space+(edge_per_pod*col+1)*portion, e_pos+portion_sw,
    pod*space+(edge_per_pod*((i%2==0)?col:((i%4==1)
    ?col+1:col-1))+1)*portion, e_pos+a_pos+portion_sw);
    double h = sl.height/frame_rate;
    double X = pod*space+(edge_per_pod*col+1)*portion;
    double Y = e_pos+portion_sw;
    for(int jj = 0; jj < frame_count; jj++) {
    X = sl.getXCordByHeight(X, Y, h);
    Y = sl.getYCordByHeight(X, Y, h);
    }
    gl.glVertex2d(pod*space+(edge_per_pod*col+1)
    *portion, e_pos+portion_sw);
    gl.glVertex2d(sl.getXCordByHeight(X, Y, h),
    sl.getYCordByHeight(X, Y, h));
    gl.glEnd();
}
if(downLinkActive_2[i] > 0) {
    gl.glColor3f(1f, 0f, 0f);
    gl.glLineWidth(downLinkActive_2[i]);
    gl.glBegin(GL.GL_LINES);
    StreightLine sl = new StreightLine(
    pod*space+(edge_per_pod*col+1)*portion, e_pos+portion_sw,
    pod*space+(edge_per_pod*((i%2==0)?col:((i%4==1)
    ?col+1:col-1))+1)*portion, e_pos+a_pos+portion_sw);
    double h = -1*sl.height/frame_rate;
    double X = (pod*space+(edge_per_pod*
    ((i%2==0)?col:((i%4==1)?col+1:col-1))+1)*portion);
    double Y = e_pos+a_pos+portion_sw;
```

```
        for(int jj = 0; jj < frame_count; jj++) {
        X = sl.getXCordByHeight(X, Y, h);
        Y = sl.getYCordByHeight(X, Y, h);
        }
        gl.glVertex2d(pod*space+(edge_per_pod*((i%2==0)
        ?col:((i%4==1)?col+1:col-1))+1)
        *portion, e_pos+a_pos+portion_sw);
        gl.glVertex2d(sl.getXCordByHeight(X, Y, h),
        sl.getYCordByHeight(X, Y, h));
        gl.glEnd();
    }
}

for(int i = 0; i < pod_num*edge_per_pod*pod_num/2; i++) {
    int pod = i/pod_num;
    int half = i/edge_per_pod;
    int which_agg = (half%2==0)?2:3;
        //aggregation index is 2 for first half and 3 for second
    int which_link = (i%2==0)?1:2;
    int core_ndx = Integer.parseInt(props.getProperty(
        "AGGR_"+pod+"_"+which_agg+"_"+which_link));

    gl.glColor3f(0f, 0f, 0f);
    gl.glLineWidth(10.0f);
    gl.glBegin(GL.GL_LINES);
    gl.glLoadName(count);
    count++;
    gl.glVertex2d(pod*space+(2*half+1)*portion,
        e_pos+a_pos+2*portion_sw);
    gl.glVertex2d(core_ndx*space +
        (hosts_per_pod*core_ndx+2)*portion, c_pos-portion_sw);
```

```
gl.glEnd();

if(upLinkActive_3[i] > 0) {
   gl.glColor3f(1f, 0f, 0f);
   gl.glLineWidth(upLinkActive_3[i]);
   gl.glBegin(GL.GL_LINES);
   StreightLine sl = new StreightLine(
   pod*space+(2*half+1)*portion, e_pos+a_pos+2*portion_sw,
   core_ndx*space+(hosts_per_pod*core_ndx+2)*portion,
   c_pos-portion_sw);
   double h = sl.height/frame_rate;
   double X = pod*space+(2*half+1)*portion;
   double Y = e_pos+a_pos+2*portion_sw;
   for(int jj = 0; jj < frame_count; jj++) {
   X = sl.getXCordByHeight(X, Y, h);
   Y = sl.getYCordByHeight(X, Y, h);
   }
   gl.glVertex2d(pod*space+(2*half+1)*
   portion, e_pos+a_pos+2*portion_sw);
   gl.glVertex2d(sl.getXCordByHeight(X, Y, h),
   sl.getYCordByHeight(X, Y, h));
   gl.glEnd();
}

if(downLinkActive_3[i] > 0) {
   gl.glColor3f(1f, 0f, 0f);
   gl.glLineWidth(downLinkActive_3[i]);
   gl.glBegin(GL.GL_LINES);
   StreightLine sl = new StreightLine(
   pod*space+(2*half+1)*portion, e_pos+a_pos+2*portion_sw,
   core_ndx*space+(hosts_per_pod*core_ndx+2)
```

```
                *portion, c_pos-portion_sw);
                double h = -1*sl.height/frame_rate;
                double X = core_ndx*space+(hosts_per_pod*core_ndx+2);
                double Y = c_pos-portion_sw;
                for(int jj = 0; jj < frame_count; jj++) {
                X = sl.getXCordByHeight(X, Y, h);
                Y = sl.getYCordByHeight(X, Y, h);
                }
                gl.glVertex2d(core_ndx*space+(hosts_per_pod*core_ndx+2)
                *portion, c_pos-portion_sw);
                gl.glVertex2d(sl.getXCordByHeight(X, Y, h),
                sl.getYCordByHeight(X, Y, h));
                gl.glEnd();
            }
        }
}

private void processHits(int hits, int buffer[]) {
        Node node;
        int offset = 0;

        if(hits != 1) return;

        // skip the number of names
        offset++;
        // skip the two z values
        offset++;
        offset++;
        // assume each selected object only has one name
        int name = buffer[offset++];
        node = namedNodes.get(name);
```

```
        node.setSelected(true);
        return;
}

void selectObj(GL gl) {
        int selectBufSize = 200;
        int selectBuf[] = new int[selectBufSize];
        IntBuffer selectBuffer = BufferUtil.newIntBuffer(selectBufSize);
        int viewport[] = new int[4];
        int hits;

        gl.glGetIntegerv(GL.GL_VIEWPORT, viewport, 0);
        gl.glSelectBuffer(selectBufSize, selectBuffer);
        gl.glRenderMode(GL.GL_SELECT);
        gl.glInitNames();
        gl.glPushName(0);
        gl.glMatrixMode(GL.GL_PROJECTION);
        gl.glPushMatrix();
        gl.glLoadIdentity();
        glu.gluPickMatrix((double) pickPoint.x,
            (double) (viewport[3] - pickPoint.y), 10, 10, viewport, 0);
        glu.gluOrtho2D(0.0, 1.0, 0.0, 1.0);

        drawScene(gl, GL.GL_SELECT);

        gl.glMatrixMode(GL.GL_PROJECTION);
        gl.glPopMatrix();
        gl.glFlush();

        hits = gl.glRenderMode(GL.GL_RENDER);
        selectBuffer.get(selectBuf);
```

```
        processHits(hits, selectBuf);
}

void updateAnimation() {
        frame_count++;
        if(frame_count == frame_rate) {
            for(int i = 0; i < currFlowData.length; i++) {
                if(!currFlowData[i].isPathBroken() && !flowDisplayDone[i]) {
                Node srcHost=currFlowData[i].getSrcHostIndex();
                Node dstHost=currFlowData[i].getDstHostIndex();
                Node upEdgeSwch=currFlowData[i].getUpEdgeSwitch();
                Node upAggrSwch=currFlowData[i].getUpAggrSwitch();
                Node coreSwch=currFlowData[i].getCoreSwitch();
                Node downAggrSwch=currFlowData[i].getDownAggrSwitch();
                Node downEdgeSwch=currFlowData[i].getDownEdgeSwitch();
                boolean samePod = false;
                if(srcHost.getPodNum() == dstHost.getPodNum())
                samePod = true;
                if((srcHost != null && dstHost != null) && (samePod ||
                (upEdgeSwitch != null && upAggrSwitch != null
                && coreSwitch != null
                && downAggrSwitch!=null && downEdgeSwitch != null)))
                int srcIndex = srcHost.getPodNum()*
                hosts_per_pod + srcHost.getIndex();
                int dstIndex = dstHost.getPodNum()*
                hosts_per_pod + dstHost.getIndex();
                int upEdgeSwIndex = upEdgeSwitch.getPodNum()*
                hosts_per_pod + upEdgeSwitch.getIndex();
                int upAggrSwIndex = upAggrSwitch.getPodNum()*
                hosts_per_pod + upAggrSwitch.getIndex();
                int coreSwIndex = -1;
```

```
int downAggrSwIndex = -1;
int downEdgeSwIndex = -1;
if(coreSwitch != null)
coreSwIndex = coreSwitch.getPodNum();
if(downAggrSwitch != null)
downAggrSwIndex = downAggrSwitch.getPodNum()*
hosts_per_pod + downAggrSwitch.getIndex();
if(downEdgeSwitch != null)
downEdgeSwIndex = downEdgeSwitch.getPodNum()*
hosts_per_pod + downEdgeSwitch.getIndex();
if(endHosts[srcIndex].isActive()) {
// From edge to aggregation (upward)
endHosts[srcIndex].setActive(false);
edAgSwitches[upAggrSwIndex].setActive(true);
upLinkActive_1[srcIndex] = updateLink(
upLinkActive_1[srcIndex], false);
int l = findEdgeAggLink(upEdgeSwitch, upAggrSwitch);
upLinkActive_2[l] = updateLink(upLinkActive_2[l], true);
} else if(!samePod) {
if(edAgSwitches[upEdgeSwIndex].isActive()) {
//From aggregation to core (upward)
edAgSwitches[upEdgeSwIndex].setActive(false);
coreSwitches[coreSwIndex].setActive(true);
int l = findEdgeAggLink(upEdgeSwitch, upAggrSwitch);
upLinkActive_2[l] = updateLink(upLinkActive_2[l], false);
l = findAggCoreLink(upAggrSwitch, coreSwitch);
upLinkActive_3[l] = updateLink(upLinkActive_3[l], true);
} else if(edAgSwitches[upAggrSwIndex].isActive()) {
// From core to aggregation (downward)
edAgSwitches[upAggrSwIndex].setActive(false);
edAgSwitches[downAggrSwIndex].setActive(true);
```

```
int l = findAggCoreLink(upAggrSwitch, coreSwitch);
upLinkActive_3[l] = updateLink(upLinkActive_3[l], false);
l = findAggCoreLink(downAggrSwitch, coreSwitch);
downLinkActive_3[l] = updateLink(downLinkActive_3[l], true);
} else if(coreSwitches[coreSwIndex].isActive()) {
// From aggregation to edge switch (downward0
coreSwitches[coreSwIndex].setActive(false);
edAgSwitches[downEdgeSwIndex].setActive(true);
int l = findAggCoreLink(downAggrSwitch, coreSwitch);
downLinkActive_3[l] = updateLink(downLinkActive_3[l], false);
l = findEdgeAggLink(downEdgeSwitch, downAggrSwitch);
downLinkActive_2[l] = updateLink(downLinkActive_2[l], true);
} else if(edAgSwitches[downAggrSwIndex].isActive()) {
// From edge to destination (downward)
edAgSwitches[downAggrSwIndex].setActive(false);
endHosts[dstIndex].setActive(true);
int l = findEdgeAggLink(downEdgeSwitch, downAggrSwitch);
downLinkActive_2[l] = updateLink(downLinkActive_2[l], false);
downLinkActive_1[dstIndex] = updateLink(
downLinkActive_1[dstIndex], true);
} else if(edAgSwitches[downEdgeSwIndex].isActive()) {
// End of the path
edAgSwitches[downEdgeSwIndex].setActive(false);
//updateArray(switchActive[downEdgeSwIndex]);
endHosts[dstIndex].setActive(false);
downLinkActive_1[dstIndex] = updateLink(
downLinkActive_1[dstIndex], false);
flowDisplayDone[i] = true;
}
} else {
if(edAgSwitches[upEdgeSwIndex].isActive()) {
```

```
        // From aggregation to edge (downward)
        edAgSwitches[upEdgeSwIndex].setActive(false);
        //updateArray(switchActive[upEdgeSwIndex]);
        edAgSwitches[downEdgeSwIndex].setActive(true);
        int l = findEdgeAggLink(upEdgeSwitch, upAggrSwitch);
        upLinkActive_2[l] = updateLink(upLinkActive_2[l], false);
        l = findEdgeAggLink(downEdgeSwitch, downAggrSwitch);
        downLinkActive_2[l] = updateLink(downLinkActive_2[l], true);
        } else if(edAgSwitches[upAggrSwIndex].isActive()) {
        // From edge to destination (downward)
        edAgSwitches[upAggrSwIndex].setActive(false);
        endHosts[dstIndex].setActive(true);
        int l = findEdgeAggLink(downEdgeSwitch, downAggrSwitch);
        downLinkActive_2[l] = updateLink(
        downLinkActive_2[l], false);
        downLinkActive_1[dstIndex] =
        updateLink(downLinkActive_1[dstIndex], true);
        } else if(edAgSwitches[downEdgeSwIndex].isActive()) {
        // End of the path
        edAgSwitches[downEdgeSwIndex].setActive(false);
        endHosts[dstIndex].setActive(false);
        downLinkActive_1[dstIndex] =
        updateLink(downLinkActive_1[dstIndex], false);
        flowDisplayDone[i] = true;
        }
        }
        }
        }
    }
    frame_count = 0;
}
```

```
}

public void displayChanged(GLAutoDrawable drawable,
    boolean modeChanged, boolean deviceChanged) {
}

void drawHost(GL gl, Node node, boolean active,
    double space, double portion) {
    int host_no = node.getPodNum()*hosts_per_pod + node.getIndex();
    int pod = node.getPodNum();
    double width_diff = (double)(host_tex.getTextureWidth()
        -host_tex.getImageWidth());
    double height_diff = (double)(host_tex.getTextureHeight()
        -host_tex.getImageHeight());
    hostStartY = Double.parseDouble(props.getProperty(
        "HOST_Y_POS"));
    hostEndY = hostStartY+ portion;
    double hostStartX = pod*space+host_no*portion;
    double hostEndX = pod*space+(host_no+1)*portion;

    if(!active)
        gl.glBindTexture(GL.GL_TEXTURE_2D, host_tex.getName());
    else
        gl.glBindTexture(GL.GL_TEXTURE_2D, host_s_tex.getName());

    if(node.isSelected()) gl.glColor3f(0.63f, 0.63f, 0.63f);
    else gl.glColor3f(0.93f, 0.93f, 0.93f);

    gl.glBegin(GL.GL_QUADS);
    gl.glTexCoord2d(width_diff/host_tex.getTextureWidth()/2, //0
            (height_diff/ 2+host_tex.getImageHeight()-1)/
```

```
            host_tex.getTextureHeight()); //1
        gl.glVertex2d(hostStartX, hostStartY);

        gl.glTexCoord2d(width_diff/host_tex.getTextureWidth()/2, //0
            height_diff/host_tex.getTextureHeight()/2); //0
        gl.glVertex2d(hostStartX, hostEndY);

        gl.glTexCoord2d((width_diff/2+host_tex.getImageWidth()-1)
            /host_tex.getTextureWidth() , //1
            height_diff/host_tex.getTextureHeight()/2); //0
        gl.glVertex2d(hostEndX, hostEndY);

        gl.glTexCoord2d((width_diff/ 2+host_tex.getImageWidth()-1)
            /host_tex.getTextureWidth() , //1
            (height_diff/ 2+host_tex.getImageHeight()-1)
            /host_tex.getTextureHeight()); //1
        gl.glVertex2d(hostEndX, hostStartY);
        gl.glEnd();
}

void drawEdgeAggregation(GL gl, Node node, boolean active,
        double space, double portion, double portion_sw) {
        int sw_no = node.getPodNum()*edge_per_pod*2 + node.getIndex();
        int pod = node.getPodNum();
        int row = node.getIndex()/2;
        int col = pod*edge_per_pod+sw_no%edge_per_pod;
        double width_diff = (double)(switch_tex.getTextureWidth()
            -switch_tex.getImageWidth());
        double height_diff = (double)(switch_tex.getTextureHeight()
            -switch_tex.getImageHeight());
        double y_pos = Double.parseDouble(
```

```
    props.getProperty("EDGE_Y_POS"));
double add_pos = Double.parseDouble(
    props.getProperty("AGGR_Y_POS"));

double startX = pod*space+(2*col+1)*portion-portion_sw/2;
double endX = pod*space+(2*col+1)*portion+portion_sw/2;
double startY = y_pos+((portion_sw+add_pos)*row);
double endY = y_pos+portion_sw+((portion_sw+add_pos)*row);

if(active)
    gl.glBindTexture(GL.GL_TEXTURE_2D, switch_s_tex.getName());
else
    gl.glBindTexture(GL.GL_TEXTURE_2D, switch_tex.getName());

gl.glBegin(GL.GL_QUADS);
gl.glTexCoord2d(width_diff/switch_tex.getTextureWidth()/2,
        (height_diff/2+switch_tex.getImageHeight()-1)
        /switch_tex.getTextureHeight());
gl.glVertex2d(startX, startY);

gl.glTexCoord2d(width_diff/switch_tex.getTextureWidth()/2,
        height_diff/switch_tex.getTextureHeight()/2);
gl.glVertex2d(startX, endY);

gl.glTexCoord2d((width_diff/2+switch_tex.getImageWidth()-1)
    /switch_tex.getTextureWidth(),
        (double)(switch_tex.getTextureHeight()
        -switch_tex.getImageHeight())/switch_tex.getTextureHeight()/ 2);
gl.glVertex2d(endX, endY);

gl.glTexCoord2d((width_diff/2+switch_tex.getImageWidth()-1)
```

```
            /switch_tex.getTextureWidth(),
                (height_diff/2+switch_tex.getImageHeight()-1)
                /switch_tex.getTextureHeight());
        gl.glVertex2d(endX, startY);
        gl.glEnd();
}

void drawCore(GL gl, Node node, boolean active, double space,
        double portion, double portion_sw) {
        int sw_no = node.getPodNum();
        double width_diff = (double)(switch_tex.getTextureWidth() -
            switch_tex.getImageWidth());
        double height_diff = (double)(switch_tex.getTextureHeight() -
            switch_tex.getImageHeight());
        double y_pos = Double.parseDouble(
            props.getProperty("CORE_Y_POS"));

        double coreStartX = sw_no*space+(hosts_per_pod*sw_no+2) *
            portion-portion_sw/2;
        double coreStartY = y_pos-portion_sw;
        double coreEndX = sw_no*space+(hosts_per_pod*sw_no+2) *
            portion+portion_sw/2;
        double coreEndY = y_pos;

        if(active)
            gl.glBindTexture(GL.GL_TEXTURE_2D, switch_s_tex.getName());
        else
            gl.glBindTexture(GL.GL_TEXTURE_2D, switch_tex.getName());

        gl.glBegin(GL.GL_QUADS);
        gl.glTexCoord2d(width_diff/switch_tex.getTextureWidth()/2,
```

```
                (height_diff/2+switch_tex.getImageHeight()-1) /
                switch_tex.getTextureHeight());
        gl.glVertex2d(coreStartX, coreStartY);

        gl.glTexCoord2d(width_diff/switch_tex.getTextureWidth()/2,
                height_diff/switch_tex.getTextureHeight()/2);
        gl.glVertex2d(coreStartX, coreEndY);

        gl.glTexCoord2d((width_diff/2+switch_tex.getImageWidth()-1) /
            switch_tex.getTextureWidth(),
                (double)(switch_tex.getTextureHeight()-
                switch_tex.getImageHeight())/
                switch_tex.getTextureHeight()/2);
        gl.glVertex2d(coreEndX, coreEndY);

        gl.glTexCoord2d((width_diff/2+switch_tex.getImageWidth()-1)/
            switch_tex.getTextureWidth(),
                (height_diff/2+switch_tex.getImageHeight()-1)/
                switch_tex.getTextureHeight());
        gl.glVertex2d(coreEndX, coreStartY);
        gl.glEnd();
    }

    private static void setupFlowInformation() throws Exception {
        for(int i = 0; i < currFlowData.length; i++) {
            if(!currFlowData[i].isPathBroken() && !flowDisplayDone[i]) {
                //read flow data and show them
                Node srcHost = currFlowData[i].getSrcHostIndex();
                Node dstHost = currFlowData[i].getDstHostIndex();
                Node upEdgeSwch = currFlowData[i].getUpEdgeSwitch();
                Node upAggrSwch = currFlowData[i].getUpAggrSwitch();
```

```
                Node coreSwch = currFlowData[i].getCoreSwitch();
                Node downAggrSwch = currFlowData[i].
                getDownAggrSwitch();
                Node downEdgeSwIndex = currFlowData[i].
                getDownEdgeSwitch();
                int srcIndex = srcHost.getPodNum()*
                hosts_per_pod + srcHost.getIndex();
                int upEdgeSwIndex = upEdgeSwitch.getPodNum()
                *hosts_per_pod + upEdgeSwitch.getIndex();
                boolean samePod = false;
                if(srcHost.getPodNum() == dstHost.getPodNum())
                samePod = true;
                if((srcHost != null && dstHost != null) &&
                (samePod ——
                (upEdgeSwitch != null && upAggrSwitch != null &&
                coreSwitch != null
                && downAggrSwitch != null &&
                downEdgeSwIndex != null))) {
                endHosts[srcIndex].setActive(true);
                upLinkActive_1[srcIndex] = updateLink(
                upLinkActive_1[srcIndex], true);
                edAgSwitches[upEdgeSwIndex].setActive(true);
                }
            }
        }
}

static int findAggCoreLink(Node agg, Node core) {
        int pod = agg.getPodNum();
        int ndx = agg.getIndex();
        int temp1 = Integer.parseInt(
```

```
            props.getProperty("AGGR_"+pod+"_"+ndx+"_1"));
        int temp2 = Integer.parseInt(
            props.getProperty("AGGR_"+pod+"_"+ndx+"_2"));
        if(temp1 == core.getPodNum())
            return pod*hosts_per_pod + ((ndx%2==0)?0:2);

        else if(temp2 == core.getPodNum())
            return pod*hosts_per_pod + ((ndx%2==0)?1:3);
        else {
            return -1;
        }
    }

    static int findEdgeAggLink(Node edge, Node agg) {
        if(edge.getPodNum() != agg.getPodNum()) {
            return -1;
        }
        int pod = edge.getPodNum();
        int ndx = edge.getIndex();
        int r = (edge.getIndex()%2 ==
            agg.getIndex()%2)?(ndx%2*2):(ndx*2+1);
        return pod*hosts_per_pod + r;
    }

    void updateArray(boolean[] a) {
        boolean res = false;
        for(int i = 0; i < a.length-1; i++) {
            res = res — a[i];
        }
        a[a.length-1] = res;
    }
```

```
    static int updateLink(int a, boolean sign) {
        if(sign) {
            if(a < 10)
                return a+1;
        } else {
            if(a > 0)
                return a-1;
        }
        return a;
    }

}
```

Bibliography

[1] M Al-Fares, A Loukissas, and A Vahdat. A scalable, commodity data center network architecture. *Proceedings of the ACM SIGCOMM conference, 2008*, 2008.

[2] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network visualization with the VINT network animator nam. Technical Report 99-703b, University of Southern California, March 1999. revised November 1999, to appear in IEEE Computer.

[3] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network visualization with nam, the vint network animator. *Computer*, 33:63–68, November 2000.

[4] D. Fisher, D.A. Maltz, A. Greenberg, Xiaoyu Wang, H. Warncke, G. Robertson, and M. Czerwinski. Using visualization to support network and application management in a data center. In *Internet Network Management Workshop, 2008. INM 2008. IEEE*, pages 1 –6, oct. 2008.

[5] http://www.openflow.org/documents/openflow-spec v0.8.9.pdf. *OpenFlow Switch Specication.*

[6] Alexander Loukissas. Implementation and simulation of the two-level lookup. Master's thesis, UCSD, 2008.

[7] Niranjan Mysore, Pamboris, Farrington, Huang, Miri, Radhakrishnan, Subramanya, and Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. *Proceedings of the ACM SIGCOMM conference, 2009*, 2009.

[8] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an openflow switch on the netfpga platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 1–9, New York, NY, USA, 2008. ACM.