City University of Hong Kong
Department of Computer Science

BSCCS Final Year Project Report 2014-2015

(14CS040)

QoS Guaranteed SDN Controller Placement and Performance Analysis for Wide Area Networks

(Volume 1 of 1)

Student Name : WANG Mengqing

Student No. : 52209388

Programme Code : BScCS

For Official Use Only

Supervisor : Prof. JIA, Xiaohua

1st Reader : Dr. LEE, Chung Sing Victor

2nd Reader : Dr. CHAN Edward

Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

QoS Guaranteed SDN Controller Placement and Performance Analysis for Wide Area Networks

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name:	WANG Mengqing	Signature:	WANG Mengqing
Student ID:	52209388	Date:	12 Apr 2015

Contents

1 Abstract

Software Defined Network (SDN) has attracted a significant attention due to its notable advantages including centralized control, simplified algorithms and efficient load balancing. It has been recognized that SDN can be widely applied in various network environments including enterprise networks, home and small businesses and data centers. OpenFlow, which is an implementation for the flow control in SDN, allows the centralized controllers to poll the traffic on each network link of the data plane network and make decisions to redirect the packets to the uncongested links. In order to deploy a scalable SDN in wide area networks, a number of controllers must be deployed in a logically centralized control plane. This controller placement issue has aroused much attention as it will significantly affect the SDN's performance in wide area networks. The controller placement problem tries to place k controllers in a given network to optimize performance metrics such as propagation latency, load distribution, network reliability and failure resilience. However, the quality of service (QoS) is always a primary concern of the network operators in the placement of SDN controllers. Since the SDN controllers are responsible to provide services for routers, the response time of controllers is an important QoS parameter of network operators. In this paper, we introduce the QoS-Guaranteed Controller Placement problem: Given a network topology and a response time bound, how many controllers are needed, where to place them, and which routers are assigned to each of the controller. We propose three heuristic algorithms: incremental greedy algorithm, primal-dual-based algorithm and network-partition-based algorithm. The proposed algorithms are tested and compared on the Internet Topology Zoo, a dataset of public available network topologies. Simulation results show that the proposed three methods have the similar performance on all input topologies.

2 Acknowledgments

First of all, I would like to thank my project supervisor Professor Xiaohua Jia for his insightful guidance throughout the project. It is Professor Jia who guided me on how to conduct a scientific research from scratch. Without his help, I would not have achieved as that far and benefited as that much.

Second, I would like to thank Ms.Tracy Cheng, who works on similar projects as mine, for her insightful suggestions through discussions.

Third, I would like to thank my classmates Mr.Yanlin Zhang and Mr.Huayi Duan, who provide me with many useful resources on how to write a good scientific paper.

Finally, I want to thank my parents who have supported me all the way along my study. Without them, I would not have had such achievements.

3 Introduction

3.1 Background & Motivation

Nowadays, complexities are introduced to computer networks with the advent of more devices including routers, switches etc. In addition, as more policies are enforced among the devices, the hierarchical structures become harder to manage for network operators. Traditional approaches always rely on the smart hardware devices with all rules embedded, which make the network ossified and not scalable. The state-of-the-art Software Defined Networking (SDN) [1] has recently been proposed for addressing the scalability issue by separating the control logic with the physical data transmission devices. A centralized control plane is extracted for making decisions for data plane traffic such as which rule should be applied to the packets, which path should be chosen for a flow etc. Network operators can therefore deploy the programmed the control logic so that network traffic can be better managed in a scalable manner.

While SDN brings significant improvement to network flexibility, challenges remain as how to make SDN outperform the traditional network infrastructure in the wide area networks in terms of scalability. Directions including distributed controller placement, flow management, fault tolerance, topology update and traffic monitoring have been researched recently. In this paper, we will focus on the problem when SDN outperform the traditional network in terms of controller placement.

3.2 Overview

3.2.0.1 Control plane design Control plane has been proposed and designed in both traditional networks and SDN. In the traditional architecture, when the control plane is not decoupled from the physically forwarding device switch, the control topology takes the form of peer-to-peer, which means each forwarding device (switch itself) talks to the control plane and

they together make autonomous decisions based on a local view of the global state. After control plane is decoupled from the forwarding devices, the design of the communications among switches and the control plane turns into the client-server model [2]. Switches consult the centralized controllers for forwarding decisions which is much closer to a client-server communication mechanism.

After the control plane is decoupled from the data plane, which is currently in charge of handling packet forwarding and without any logic embedded in, the greatest concern is how to arrange the controllers in the control plane. The most popular forms for the arrangement include a star (single controller), a hierarchy (controllers connected in a full mesh) or a dynamic ring (controllers in a distributed hash table).

3.2.0.2 Placement metrics in WAN Applying SDN in wide area networks brings a lot of flexibility to the network functions. However, the scalability draws more concerns since one controller may not cater to the requirements of wide area networks. The requirements include low communication delay among switches and controllers, short control service queue, fault tolerant in face of link failures etc. In this sense, researches have been conducted to see how many controllers are needed in a certain network topology and where should they be placed to satisfy the requirements.

Although the design of control plane varies with respect to different performance metrics, we are mostly interested in the propagation delay between the switch and the controller as well as the service time of each controller in wide area networks. The propagation latency places fundamental limits in the resource availability, which we believe that if the delay factor can be set as a reasonable enough value, the link failure can be eliminated. Besides, the service time of each controller also needs to be taken into consideration since the sojourn time of a control packet during queuing in the controller will affect the total delay.

3.3 Aims & Objectives

This project targets at addressing the abovementioned problem in the following approaches:

1. Formulate the QoS-guaranteed controller placement problem into a linear programming problem
2. Propose heuristics algorithms to solve the linear program
3. Design the simulation experiments to evaluate the performance of the proposed heuristic algorithms

4 Literature Review

4.1 Current work and its limitations

To address the controller placement problem, there are some research works considering different performance metrics such as node-to-controller latency [2], network reliability [3], load of controllers [4], inter-controller delay [5], failure resilience [6], etc.

The controller placement problem was firstly introduced by [2]. This work aims to minimize the average latency between controllers and routers. It is pointed out that the average latency and the maximal latency cannot be optimized at the same time, and a K-means greedy method was employed to find the optimal placement that minimizes the average latency and k-center greedy method was proposed to minimize the maximal latency.

A fault tolerant controller placement was proposed in [3]. The author formulated the fault tolerant controller problem considering the network reliability constraint. A heuristic algorithm was proposed to computes placements with the required reliability. This work concludes that each node is required to connect to 2 or 3 controllers, which typically provide more than five nines reliability.

The load of controllers is taken into account for designing controller placement in [4]. The controller problem was extended to a capacitated problem where there is an upper bound for the load of each controller. The author proposed a greedy K-center algorithm to solve the problem.

The propagation delay among controllers is taken into consider in [5] for designing the control plane. A K-critical algorithm is developed to find the minimum number of controllers to satisfy a target communication between controllers and nodes, such as delay, latency, convergence time, etc.

Failure resilience is considered in [6] for placing controllers. This work concludes that whereas one controller is enough from a latency point-of-view, more controllers are needed to meet resilience requirement. They also consider inter-controller latency, load balancing between controllers, and trade-off considerations between latency and failure resilience. A framework for resilience pareto-based optimal controller placement is proposed that provides network operators with all pareto-optimal requirements.

The previous works on the controller placement problem focus on placing a given number of controllers to optimize a given performance metric such as propagation delay, controller load distribution, network reliability, etc. However, minimizing the number of controllers has not been considered and the QoS has not been guaranteed in the previous work. The response time of the request sent from a router to its controller is an important parameter of QoS. Therefore, we propose the QoS-guaranteed controller placement problem to determine: 1) the minimal number of controllers needed; 2) where to place them; 3) which router is under controller of each of them.

4.2 Controller Placement as Set Cover Problem

Set cover problems was introduced in combinatorics and it is a problem which promote the development of approximation algorithms. We will review the set cover problem in this section since it is a problem closed related to the system design of the controller placement problem.

Set cover problem has been defined as

Given an instance of (U, C), U represents a finite set of elements and C represents a collection of subsets. The relationship of U and C can be denoted as: for each element in U, it belongs to at least one subset in the family C.

$$\mathrm{U}{=}\cup\mathrm{S} \quad \text{where } \mathrm{S} \in \mathrm{C}$$

The set cover problem comes into two flavours, which are weighted set cover problem and unweighted set cover problem. In the unweighted set cover problem, the cost for taking a collection of sets is defined as the number of sets in the collection:

$$|C| = \sum_{S_i \in C} 1$$

In the weighted set cover problem, there is a weight function for each set S: w: $S- > R$. The total cost of the collection is defined as the summation of the total weight:

$$|C| = \sum_{S_i \in C} w(S_i)$$

The goal of the minimum set cover problem is to minimize $|C|$.

Minimum set cover problem is a model for many resource covering problems. Most of these problems only take the consideration of whether the resource has been selected or not, which means the cost of each resource weighs as binary (0 or 1). Therefore, unweighted set cover problem is comprehensively studied and we will illustrate this problem in detail in the following sections.

An example is used to illustrate the unweighted minimum set cover problem here. In the figure 1, an instance (U, T) is the set cover problem. U contains in total 12 points and T consisits of $T = \{T1, T2, T3, T4, T5, T6\}$. One possible solution for the minimum set cover is $C = \{T3, T4, T5\}$.

Minimum set cover problem has been used to solve many practical problems including shortest superstring problem and edge coverting problem. It is proved to be one of Karp's NP-complete problems and therefore to solve this problem, we need to develop heuristic algorithms which takes a certain approximation ratio to the optimal solution. There are two main approaches to tackle the minimum set cover problem, named greedy approximation algorithm and primal-dual

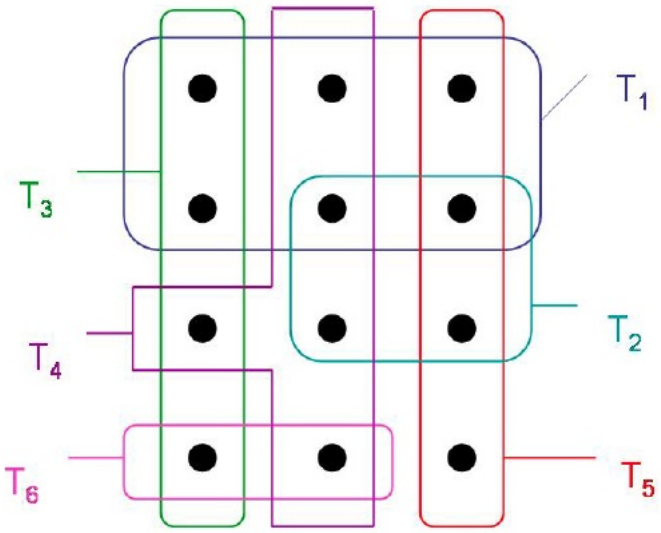

Figure 1: Set cover problem

based algorithm:

4.2.1 Greedy approximation algorithm

The idea of the greedy approximation algorithm is that in each step we pick the set with the greatest number of points yet uncovered. For the example in Figure 1, the greedy algorithm will pick $T1$ at first because it covers the greatest number of points. It will then pick $T4$, which leaves 3 points uncovered. Next $T5$ will be selected to cover 2 points and finally $T3$ will be picked to cover the last point. By the greedy approximation algorithm, the total cost $|C| = 4$. However, we have discussed that the optimal solution will return $|C| = 3$.

The greedy approximation algorithm is described here in Algorithm 1:

The greedy approximation algorithm for the minimum set cover problem has been analyzed and proved to be an $\alpha - approximation$ algorithm. The following theorem introduces how α is derived.

Input: Universe U, Sets S;

Output: Minimum size of sets C;

$R \leftarrow U, C \leftarrow \emptyset$;

while $R \neq \emptyset$ **do**

  select $I \in S$ where $|I \cap R|$ is maximized ;

  $R \leftarrow R - S$;

  $C \leftarrow C \cup \{S\}$;

end

return C

Algorithm 1: Greedy Set Cover

Theorem 1 *The greedy approximation algorithm for the minimum set cover problem is* $\alpha -$ *approximation, where*

$$\alpha = H(max|I| : I \in S)$$

$H(a)$ *is defined as the* a^{th} *harmonic number.*

4.2.2 Primal-dual based algorithm

A large proportion of the approximation algorithms is built on the Linear Programming (LP) method since the primal-dual attribute can yield the combinatorial algorithms with satisfying approximation ratios and running times. In this section, we will firstly review the basic linear programming theories, in which we focus on the LP-duality that gives the most algorithmic significance. Then we will introduce how approximation algorithms can be designed based on the primal-dual schema.

4.2.2.1 Linear Programming Linear programming is the problem to maximize or minimize a linear function which is subject to several linear constraints. The function to be optimized is

called *objective function*. Here is an example of the linear programming problem:

$$\begin{aligned} &minimize\ 5x_1 + 6x_2 \\ &subject\ to\ 3x_1 + 2x_2 \geq 10 \\ &x_1 + 2x_2 \geq 3 \\ &x_1 \geq 0, x_2 \geq 0 \end{aligned} \tag{1}$$

Any solution that satisfies the constraints of the linear program 1 is called a feasible solution. It's obvious that there are many feasible solutions for a linear program. What we focus on here is the optimal solution among all feasible solutions in order to satisfy the objective function. Let z^* denote the optimal solution of the linear program. the question that remains is how to put a lower bound on z^*. According to the linear program 1, we can obtain the relationship $5x_1 + 6x_2 \geq (3x_1 + 2x_2) + 2(x_1 + 2x_2) \geq 16$. To find out the lower bound on z^*, we only need to choose the appropriate multipliers for $3x_1 + 2x_2$ and $2x_1 + 4x_2$ such that the right hand side of the sum is as large as possible. In this sense, the problem of finding out the lower bound is transformed into the other problem which is formulated below:

$$\begin{aligned} &maximize\ 10y_1 + 6y_2 \\ &subject\ to\ 3y_1 + 2y_2 \leq 5 \\ &2y_1 + 4y_2 \leq 6 \\ &y_1 \geq 0, y_2 \geq 0 \end{aligned} \tag{2}$$

We call the linear program 1 *primal program* and the linear prgram 2 *dual program*. The transformation can be obtained systematically from the minimization problem to the maximization problem. The dual program will provide the lower bound for the optimal solution of the primal program. The reverse also holds which means the solution of the primal program serves as the upper bound for the optimal solution of the dual program. In this sense, if we can find the feasible solutions for primal and dual programs which yield the same objective

function values, we can conclude that they are both optimal solutions. This is the main idea of the $LP - duality\, theorem$. Let's formulate the general linear programming problem and the LP-duality theorem as below:

A general linear program can be defined as follows:

$$(LP) \quad minimize \sum_{j=1}^{n} c_j x_j \qquad (3)$$
$$subject\, to \sum_{j=1}^{n} a_{ij} x_j \geq b_i,\ i = 1, ..., m$$

$$x_j \geq 0,\ j = 1, ..., n \qquad (4)$$

A vector $x \in R^n$ is a feasible solution of the linear program if it satisfies constraints 3 and 4. $\sum_{j=1}^{n} c_j x_j$ is called the objective value of x. If $\sum_{j=1}^{n} c_j x_j^* \leq \sum_{j=1}^{n} c_j x_j$ for every x, x^* is therefore the optimal solution of the linear program.

Our objective to solve the linear programming problem is to find a lower bound for $\sum_{j=1}^{n} c_j x_j^*$. One approach is to transform the existing (primal) linear program into a dual linear program.

$$(D) \quad maximize \sum_{i=1}^{m} b_i y_i \qquad (5)$$
$$subject\, to \sum_{i=1}^{m} a_{ij} y_i \leq c_j,\ j = 1, ..., n$$

$$a_i \geq 0,\ i = 1, ..., m \qquad (6)$$

Theorem 2 (LP-Duality) *The primal program has finite optimum if and only if its dual has*

finite optimum. Moreover, if x^* and y^* are optimal solutions for the primal and dual program, respectively, then $c^T x^* = y^{*T} b$.

We can further conclude the complementary slackness conditions from the abovementioned LP-duality:

Theorem 3 (Complementary slackness conditions) *Let x be a feasible primal solution and y a feasible dual solution. Then, x and y are both optimal if and only if all of the following conditions are satisfied:*

Primal complementary slackness conditions:

$\forall j \in [1, n]$, *there must satisfy:* $x_j = 0 \parallel \sum_{i=1}^{m} a_{ij} y_i = c_j$

Dual complementary slackness conditions:

$\forall i \in [1, m]$, *there must satisfy:* $y_i = 0 \parallel \sum_{j=1}^{n} a_{ij} x_i = b_i$

The complementary slackness conditions play an important role in designing approximation algorithms, which we will show in the next section.

4.2.2.2 Approximation algorithms based on LP

From the introduction of the linear programming, it is not difficult to find out why linear programming plays the significant role in designing approximation algorithms for combinatorial optimization problems, which can be stated as integer programs. The linear relaxation on the integer problem can provide a lower bound of the optimal solution.

There are two common approaches in designing approximation algorithm named linear programming rounding and primal-dual technique.

- **Linear programming rounding**: One intuitive way to solve the integer program is that we firstly solve the program and round the fractional solution to the closest integers. This

technique is also called randomized rounding, which was introduced by [7]. It introduces the method the round the variables to 0,1 with a given probability. Since it is probabilistic, we can therefore calculate the expected cost, which is ρ times of the optimal value. Therefore, the approximation guarantee is derived from the cost of fractional and integral solutions.

- **Primal-dual technique**: This method can be applied to deriving both exact and approximate algorithms. The main idea is to derive the integral solutions for primal and the feasible solutions to the dual programs iteratively since any feasible solution to the dual returns the lower bound for the optimal solution for the primal. We can formulate the technique as follows. From the definition of LP-duality in (3) and (5) as well as the complementary slackness conditions, let $\alpha \geq 1$ and $\beta \geq 1$, the slackness condition can be relaxed as:

  Relaxed primal complementary slackness conditions

  $\forall j \in [1, n]$, there must satisfy: $x_j = 0 \parallel \frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$

  Dual complementary slackness conditions

  $\forall i \in [1, m]$, there must satisfy: $y_i = 0 \parallel b_i \leq \sum_{j=1}^{n} a_{ij} x_i \leq \beta b_i$

  An $\alpha\beta$-approximation algorithm can be derived if we can find x and y primal and dual feasible.

5 System Design

In this section, we first formulate the response time of a request from a router to a controller, which is performance metric considered in our controller placement problem. Then we formulate the proposed QoS-guaranteed controller placement problem.

The network is modeled by graph $G(V; E)$, where V represents network nodes, E represents the network links between nodes. Each node $i \in V$ represents a router. Each router i is associated with a request demand r_i. Each router's location is a potential candidate site for placing a controller. Suppose all controllers have the same processing capacity μ. Our goal is to find the minimal subset $C \in V$ to place a controller in each node of C.

Table 1: Notations of Variables

Notation	Meaning
V	set of routers
C	set of controllers
δ	upper bound of response time
r_i	the requesting rate of router i
t_i	response time of request from router i
$\bar{t}$	average response time of all routers
μ	service capacity of each controller
λ_j	traffic arrival rate of controller j
τ_j	mean service time of controller j
S_j	serving range of controller j
x_{ij}	variable indicates whether router i is assigned to controller j
y_j	variable indicates whether controller candidate j is open
a_j	variable indicates whether set S_j is selected
b_i	dual variable of router i

The main job of a controller in SDN is to set up routing information for flows upon request from routers [8]. The flow setup process works as follows: When the first packet of a new flow arrives at an OpenFlow switch, its header information is extracted and then matched against the flow table entries. If there is no match found for the packet, the packet is forwarded to the controller for processing. The controller will send the routing information to the router to help set up the flow table for the new flow of this packet [9]. The response time of a request sent from a router is defined as the time from the moment of sending out this request until the reply is received from its controller. Given a upper bound δ of the response time, our task is to place the minimum number of controllers in V such that the average response time for all routers is within the given bound δ.

We introduce two variables, X and Y, to respectively denote the assignment of routers to controllers and the placement of controllers. For router i and controller j, $x_{ij} = 1$ if router i is assigned to controller j and $x_{ij} = 0$ otherwise; $y_j = 1$ if there is a controller placed at node j and $y_j = 0$ otherwise. The response time of a flow setup request from a router has two components: 1) the round trip time from the switch to its controller, and 2) the service time of the controller for the request. We first take a look at the service time of a controller.

Each controller can be modeled as a M/M/1 queueing model. Suppose packets arrive according to a Poisson process and the processing times of the packets are independent and identically exponentially distributed. Let λ_j denote the total arrival rate of packets at the controller j, which is represented as:

$$\lambda_j = \sum_{\forall i \in S} r_i x_{ij}, \tag{7}$$

where r_i is the request rate of router i.

The packets in one controller are processed according to First-Come-First-Serve order. Each controller $j \in V$can only accept number of clients within its capacity μ, therefore it is required that: $\lambda_j < \mu$.

According to the queueing theory [10], the expected mean service time of controller j is defined as:

$$\begin{aligned} \tau_j &= \frac{1}{\mu - \lambda_j} \\ &= \frac{1}{\mu - \sum_i r_i x_{ij}} \end{aligned} \tag{8}$$

For each router i, let t_i denote the response time of its request, which can be represented as:

$$\begin{aligned} t_i &= 2t_{ij} + \tau_j \\ &= 2t_{ij} + \frac{1}{\mu - \sum_i r_i x_{ij}}, \end{aligned} \tag{9}$$

where t_{ij} denotes the network delay between router i and controller j. We assume the request and reply between the router and the controller always take the shortest path in the network. t_{ij} is the sum of the link delay of the shortest path between nodes i and j in the network.

Let $\bar{t}$ denote the mean response time, which is defined as:

$$\bar{t} = \frac{1}{|R|} \sum_{i \in R} t_i \tag{10}$$

In this paper, we define the QoS-guaranteed controller placement problem as follows:

Definition 1 *Given a network topology G, we are asked to place the minimal number of controllers subject to the response time constraint:* $\bar{t} \leq \delta$.

6 Heuristic Algorithm Design

Each router's location is a candidate site for placing a controller. We compute the service range of each candidate site. The service range of a candidate site include all routers whose requests have a response time less than or equal to δ from this controller candidate, which guarantees the QoS requirement. Let S_j denote the set of routers within the service range of candidate site $j \in V$. Our proposed QoS-guaranteed controller placement problem can be described as: Given a set V of n routers, a collection of subsets of V: $S = \{S_1, ..., S_n\}$, find a minimum subcollection of S that serves all routers of V. To solve this problem, we design and compare three heuristic algorithms: incremental greedy algorithm, primal-dual-based algorithm and network-partition-based algorithm.

6.1 Incremental Greedy Algorithm

The incremental greedy algorithm iteratively picks the largest set and removes the served routers. In each iteration, open the controller candidate j associated with the selected set S_j. Routers within the set S_j are assigned to this controller j. This iteration is repeated until every router is served by one controller.

The detail of greedy algorithm is illustrated in Algorithm 2. At the begining, we compute the service range of each candidate site. Suppose each candidate site is placed with a controller, the service range of a cadidate site include all routers whose requests have a response time less than or equal to δ from the controller at this candidate site. In each iteration, the candidate site with the largest service range is chosen to place a controller. Unserved routers within its service range are assigned to this controller. This iteration is repeated until all routers are being served.

Input: Graph $G = (V, E)$, Response time bound δ;

Output: X, Y;

$R \leftarrow V$, $C \leftarrow V$;

$y_j = 0,\ \forall j \in C$;

$x_{ij} = 0,\ \forall i \in R, j \in C$;

while $R \neq \emptyset$ **do**

  foreach $j \in C$ **do**

    Compute the service range of candidate site j;

  end

  Find candidate site j which has the largest service range and set $y_j = 1$;

  $C \leftarrow C - \{j\}$;

  foreach *router i within the service range of j* **do**

    $x_{ij} = 1$;

    $R \leftarrow R - \{i\}$;

  end

end

return X, Y

Algorithm 2: Incremental Greedy Algorithm

6.2 Primal-Dual-based Algorithm

The incremental greedy algorithm does not have a performance guarantee for all kinds of inputs. It's performance often varies on different input. The primal-dual schema is used for designing approximation algorithms for integer programming problems, which provides a approximation ratio compared with the optimal solution. The basic idea is to consider the LP-relaxation of the original integer program as the primal program. We derive the dual program of the primal program. Then we settle down the complementary slackness condition of the primal program and the dual program respectively. In the primal-dual schema, we start with initial feasible solutions to the primal and dual programs, it iteratively starts satisfying complementary slackness conditions. The current primal solution is used to determine the improvement to the dual solution, and vice versa. This algorithm is ended when a primal feasible solution is obtained and all complementary slackness conditions are satisfied.

To formulate the proposed controller placement problem as an integer program, let us assign a variable a_j for each set $S_j \in S$, which is allowed to be 0/1 values. The set S_j represents the set of routers within the service range of controller candidate j. a_j is set to 1 iff set S_j is selected, 0 otherwise. For each router $i \in V$, it is required that at least one of the sets containing it be selected. The controller placement problem can be formulated as:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{S_j \in S} a_j \\
\text{subject to} \quad & \sum_{S_j : i \in S_j} a_j \geq 1, \qquad \forall i \in V \\
& a_j \in \{0, 1\}, \qquad \forall j \in V
\end{aligned}
\tag{11}
$$

The LP-relaxation of this integer program is obtained by letting the domain of variables a_j be $a_j \geq 0$, which is the primal program. We introduce a variable b_i for each router $i \in V$, we obtain the dual program.

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in V} b_i \\ \text{subject to} \quad & \sum_{i: i \in S_j} b_i \leq 1, \qquad \forall S_j \in S \\ & b_i \geq 0, \qquad\qquad \forall i \in V \end{aligned} \tag{12}$$

The complementary slackness conditions are:

$$\begin{aligned} \text{Primal condition:} \quad & a_j \neq 0 \Rightarrow \sum_{i: i \in S_j} b_i = 1, \ \forall S_j \in S \\ \text{Dual condition:} \quad & b_i \neq 0 \Rightarrow \sum_{S_j: i \in S_j} a_j \leq f, \ \forall i \in V \end{aligned} \tag{13}$$

Define the frequency of an router to be the number of sets it is in. f denotes the frequency of the most frequent router. Set S_j is said to be tight if $\sum_{i: i \in S_j} b_i = 1$. The primal variable a_j is incremented integrally, which means to pick only tight sets in the system.

The primal-dual algorithm proceeds in iterations. In each iteration, we pick an unserved router, say i, and raise b_i until some set goes tight. Pick all tight sets and open the related controller candidates. For each tight set, assign routers within the set to the related controller. For routers covered by multiple tight sets, each of them can randomly choose a tight set and connect to its controller. Then the unselected sets and unserved routers are updated. This iteration is repeated until all routers are served. This primal-dual algorithm achieves an approximation

factor of f. The detail of this primal-dual algorithm is illustrated in Algorithm 3.

To help illustrate this primal-dual-based controller placement algorithm, we give an example. In Figure 2, there are some sets $\{N_1, N_{13}\}$, $\{N_2, N_{13}\}$, $\{N_3, N_{13}\}$, ... $\{N_9, N_{13}\}$, $\{N_{21}, N_{22}\}$. Each of these sets represents a service range of a controller candidate. Suppose the algorithm raises b_{13} in the first iteration. When b_{13} is raised to 1, all sets including N_{13} go tight. They are all picked in the system, thus routers $N_1, N_2, N_3, ..., N_9$ are served by the controllers of the tight sets. In the second iteration, b_{21} is raised to 1 and the set $\{N_{21}, N_{22}\}$ goes tight. Pick the set $\{N_{21}, N_{22}\}$ into the system and all routers are being served by at least one controller.

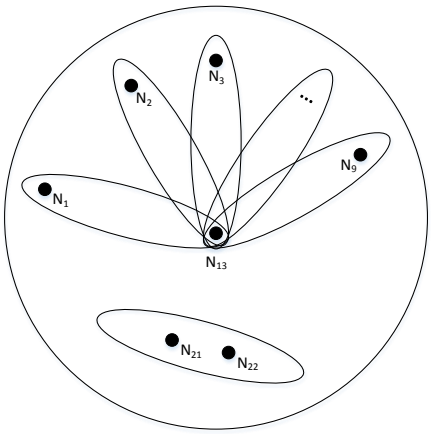

Figure 2: An expample of applying primal-dual-based algorithm

6.3 Network-Partition-based Algorithm

The above greedy algorithm and the primal-dual algorithm don't have a global view of the whole network. They create unbalanced placement and unbalanced load among controllers. Next we design a network-partition-based algorithm which considers the load balancing among all controllers.

Input: Graph $G = (V, E)$, Response time bound δ;

Output: X, Y;

foreach $j \in V$ **do**

    compute the service range and S_j ;

end

$S = \{S_j, \forall j \in V\}$;

$a_j = 0, \ \forall j \in S$;

$b_i = 0, \ \forall i \in V$;

while $V \neq \emptyset$ **do**

    Pick an unserved router, say i, and set $b_i \leftarrow b_i + 1$; Find all sets that go tight ;

    foreach *tight* S_j **do**

        Set $a_j = 1$ and open the controller j ;

        Assign all routers within S_j to controller j ;

        $V \leftarrow V - \{i | i \in S_j\}$;

    end

end

return X, Y

Algorithm 3: Primal-Dual-based Algorithm

The basic idea of network-partition-based method is to divide the network into partitions and place controllers in each partition seperately. First, the network is divided into partitions. In each partition, the largest propagation delay between any two routers is equal to the bound δ. In each partition, we use the incremental greedy method to place controllers. Since we limit the service range of each controller to one partition, the load distribution among all controllers is balanced.

The detail of the network-partition-based algorithm is described in Algorithm 4. First, the network is divided into grids. Any pair of routers within each grid has a communication delay less than δ. In the network partitioning, we first divide the network into vertical strips, where each strip is left closed and right open. In each strip, the propagation delay between the node on the left boundary and the node on the right boundary is equal to δ. Then each strip is divided into grids. In each grid, the propagation delay between the node on the top boundary and the node on the bottom boundary is equal to δ. After partitioning the network, we use the greedy method to place controllers in each partition.

Input: Graph $G = (V, E)$, Response time bound δ;

Output: X, Y;

$R \leftarrow V$, $C \leftarrow V$;

$y_j = 0,\ \forall j \in C$;

$x_{ij} = 0,\ \forall i \in R, j \in C$;

Divide the network graph into vertical strips ;

foreach *vertical strip* **do**

    Divide the vertical strip into horizontal grids ;

end

foreach *grid* **do**

    Use the incremental greedy algorithm to place controllers ;

end

return X, Y

Algorithm 4: Network-Partition-based Algorithm

7 Analysis of the controller placement

The proposed three heuristic methods are tested on the Internet Topology Zoo, a public dataset of network topologies. We first introduce the setup of the simulation, and then give the simultation results and analysis.

7.1 Experiment setup

The network topologies we collected are from a public repository called Internet Topology Zoo [11] which collects topologies and conducts statistical analysis on each of them, including location, bandwidth etc. There are in total 232 network topologies which are public now, among which 141 are at the Point-of-Presence (PoP) level, which means each node in the graph can be abstractly taken as a location for a router. In our scenario, each PoP location is also taken as a candidate location for placing a controller.

We further analyze the 141 topologies of at the PoP level. Among them, 124 of them are connected network topologies, which means they can be applied in the scope of SDN since each site can communicate with others either for requesting decisions from the controller or making decisions as a controller. Among the 124 connected network topologies, 74 of them are labeled with precise latitude and longitude coordinates for each node, which means we can therefore calculate the distance among nodes according to their geographical coordinates. There are two types of distance between two locations on earth. One is called Vincenty distance, which adopts iterative methods in measuring the distance; the other one is called great-circle distance, which is less accurate than Vincenty distance measurement. Here we follow a more precise pattern in measuring the distance, which is the Vincenty distance.

The parameters in the simulation experiments are defined and evaluated before the experiments. According to a flow setup rate measurement test [12] on the HP ProCurve 5406zl switch,

the results indicates that the switch completes roughly 275 flow setups per second. In the Internet Topology Zoo, a PoP may consist of a number of routers. Therefore the requesting demand of each router in our scenario is set as $r_i \in [1500, 3000]$ per second. A study [13] shows that a popular network controller (NOX) handles around 30k flow requests per second while maintaining a sub-10ms flow install time. Therefore, the processing rate of each controller in our problem is set as $u_j = 30000$ per second.

The response time of the request sent from a router to its controller consists of three components: the propagation time from the node to the controller (uplink), the service time (sojourn time) of the controller, the propagation time from the controller to the node (downlink). Let T_G denote the largest propagation delay for the network topology G. The value of the response time bound δ depends on the network topology. Since each input network topology is different from each other, the value of δ is also different for each topology. The upper bound of the response time δ in our problem is set as $\delta \in [2T_G + 5, 2T_G + 10]$ ms. We run the three heuristic algorithms seperately for network topology G under a given response time bound δ. Each simulation result is obtained through an average pf 10 running experiments.

7.2 Simulation results

The controller placement was studied for the three heuristic algorithms respectively. The results presented in this section are compared among different network sizes.

The result of simulation experiments indicates that, not surprisingly, the three heuristic algorithms achieves overall similar performance over most network topologies of different sizes ranging from 4 to 74 routers. In the following sections, we will analyze the performance of

each algorithm under different network sizes, network geographical diameters, controller load balancing and QoS constraints.

7.2.1 Network Size

The network topologies that we collect from Internet Topology Zoo have the sizes ranging from 4 to 74 routers. Figure 3 and 4 shows the number of controllers required based on the ascending order of the network sizes of topologies.

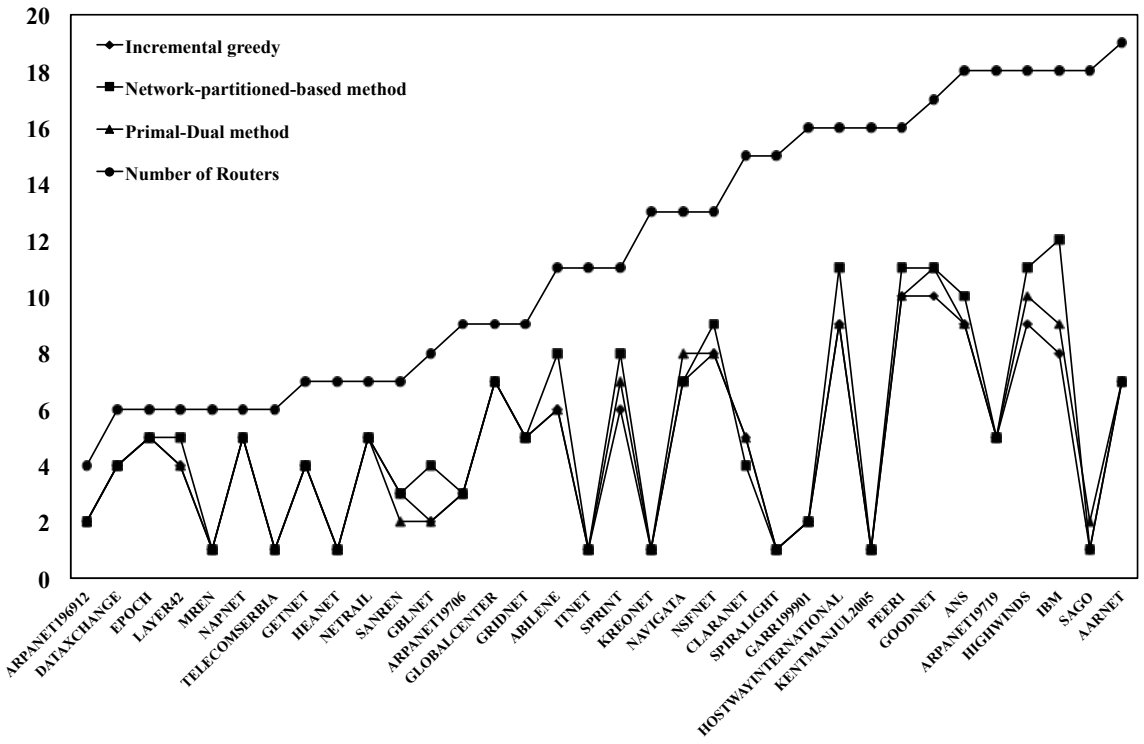

Figure 3: Overall number of controllers (part I)

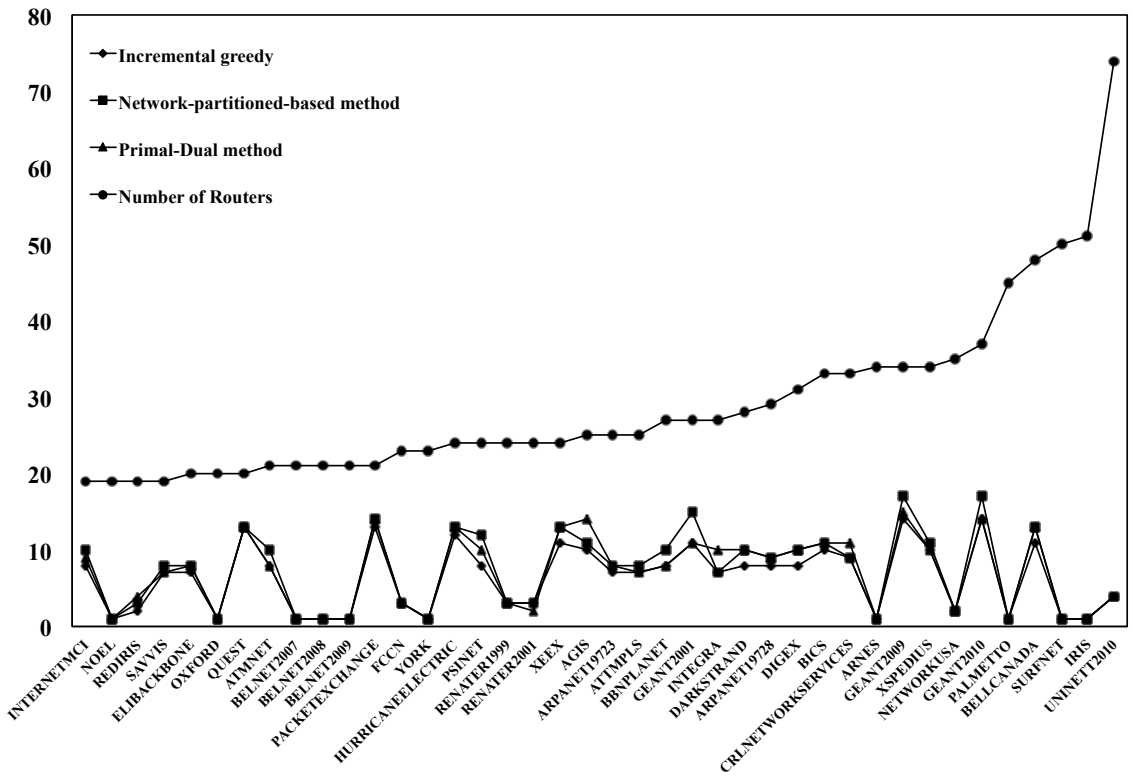

Figure 4: Overall number of controllers (part II)

We can conclude that the three proposed heuristic algorithms perform similarly for topologies of different sizes.

7.2.2 Network Geographical Diameters

In wide area networks, geographical distance is a main concern of the network communication efficiency. According to our calculation, the propagation delay between any two routers far exceeds the queuing delay for router-controller communication. Furthermore, the router-controller communication happens much less frequently. Therefore, the propagation delay among the routers will affect the result of how to place controllers. As we know, the propagation delay is determined by the geographical distance between any two routers. Hence, apart from the network size, we also investigate how the geographical diameter affect the controller placement under the three heuristic algorithms. The geographical diameter of a network topology is measured by the greatest communication distance among any two routers in the topology. Here, the communication distance between two routers is defined as the shortest link length between the two routers.

The results shown in the figure 5 and 6 below indicates the number of controllers output by three heuristic algorithms in the ascending order of geographical diameter of network topologies, which means $KentmanJul2005$ in figure 5 has the smallest geographical size and *packetexchange* in figure 6 has the largest geographical size.

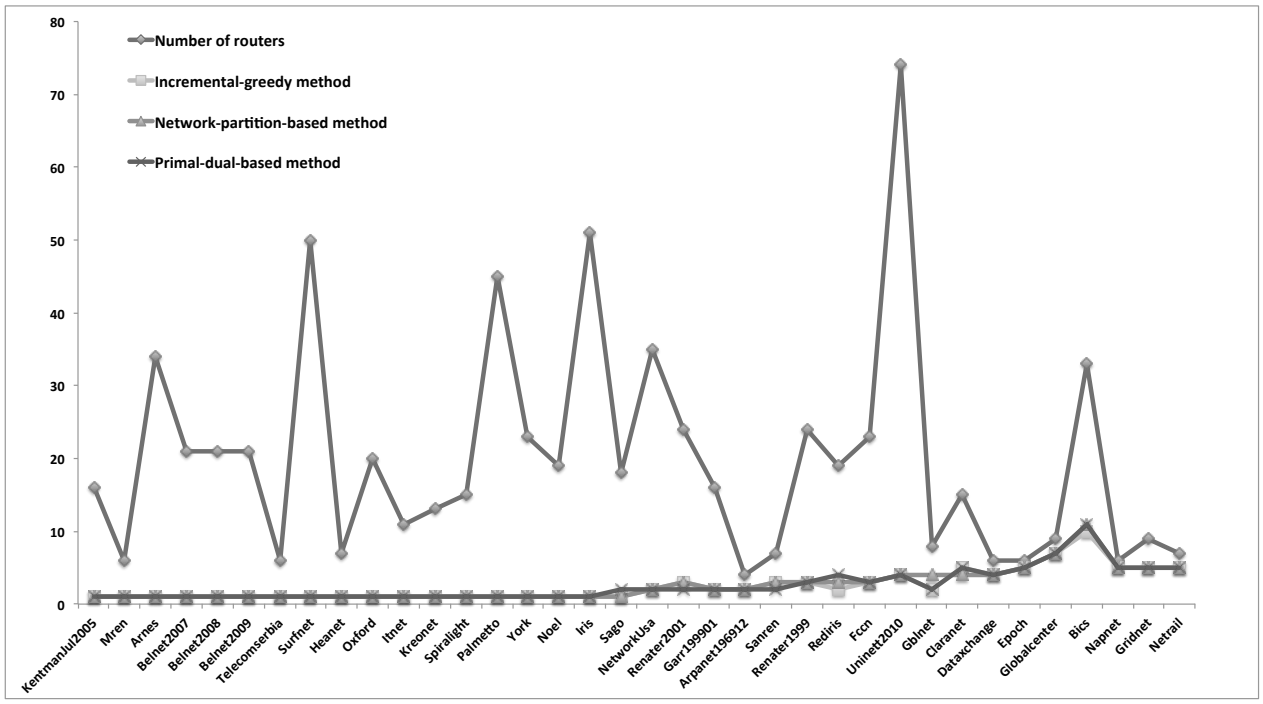

Figure 5: Overall number of controllers in the ascending order of geographical diameter(part I)

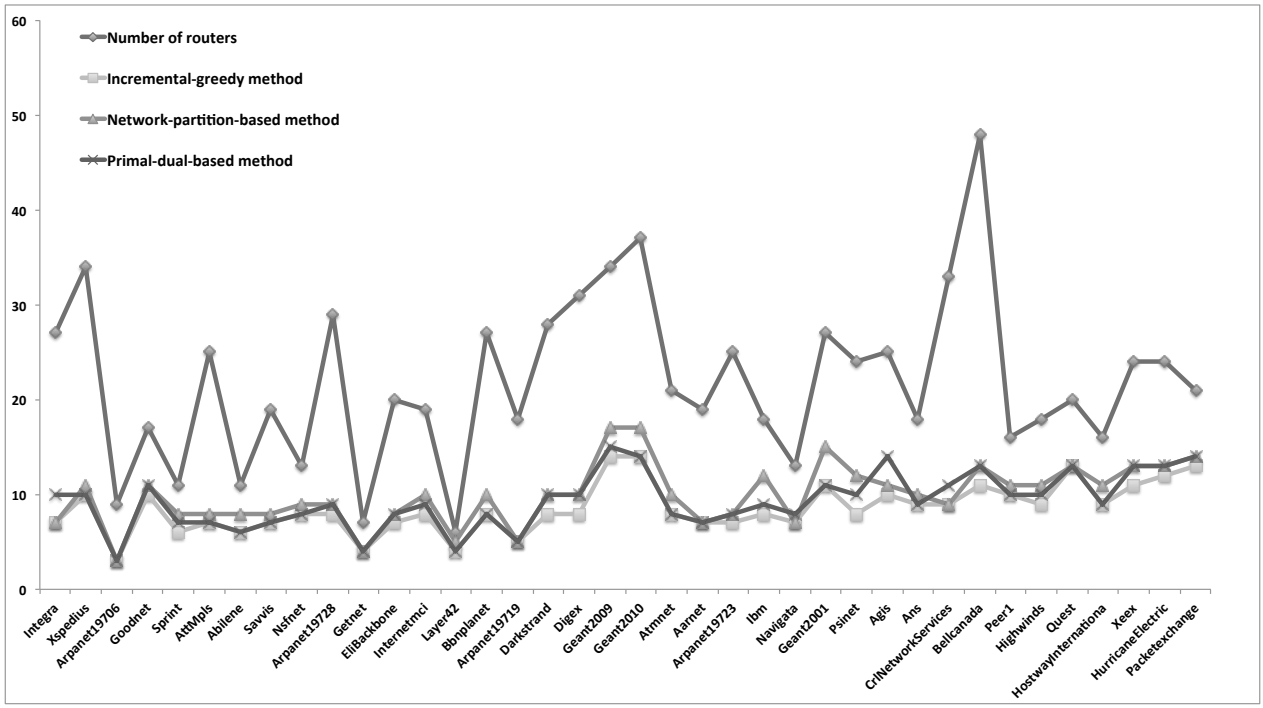

Figure 6: Overall number of controllers in the ascending order of geographical diameter(part II)

Observing from figure 5, the three algorithms perform similarly when the geographical size is from small to medium. When the geographical size turns larger, the network-partition-based algorithm tends to place more controllers than the other two algorithms do. This conclusion can be observed from topologies with larger geographical diameters such as *Geant*2010 and *Atmnet*. To explain why this algorithm achieves worse results, we need to investigate what dominates the result of this algorithm. In the network-partition-based algorithm, we firstly partition the networks into grids. It is not difficult to observe that more grids will be obtained when the network geographical size gets larger. Since each grid will require at least one controller to serve all the routers located in the grid, the number of controllers will be dominated by the number of grids in this algorithm, which is highly dependent on the network geographical size.

Although the network-partition-based algorithm assigns more controllers in some network topologies, it overall achieves better load balancing for each controller, which we will discuss in the following section.

7.2.3 Controller Load Balancing

In the QoS-guaranteed controller placement problem, we consider the respose time to be the QoS constraint. It means routers will be assigned to a controller as long as the summation of propagation delay and the queuing time spent on the controller does not exceed the QoS constraint. However, it will introduce a problem that the workload of each controller is not considered. Some controllers may need to serve more routers while some may not. If a controller is heavily-loaded, it will possibly introduce more communication errors, which we actually don't expect.

Although we did not formulate the problem as a load-balance-guaranteed one, the network-partition-based algorithm actually serves as a good insight into how to balance the load among all

controllers. Network-partition-based algorithm partitions the topology into grids and controllers in each grid will be selected inside each grid. This criteria guarantees that controllers will only serve the routers in a certain range without any influence of other grids. The experiment result below indicates that the controllers selected from the network-partition-based algorithm have more balanced load.

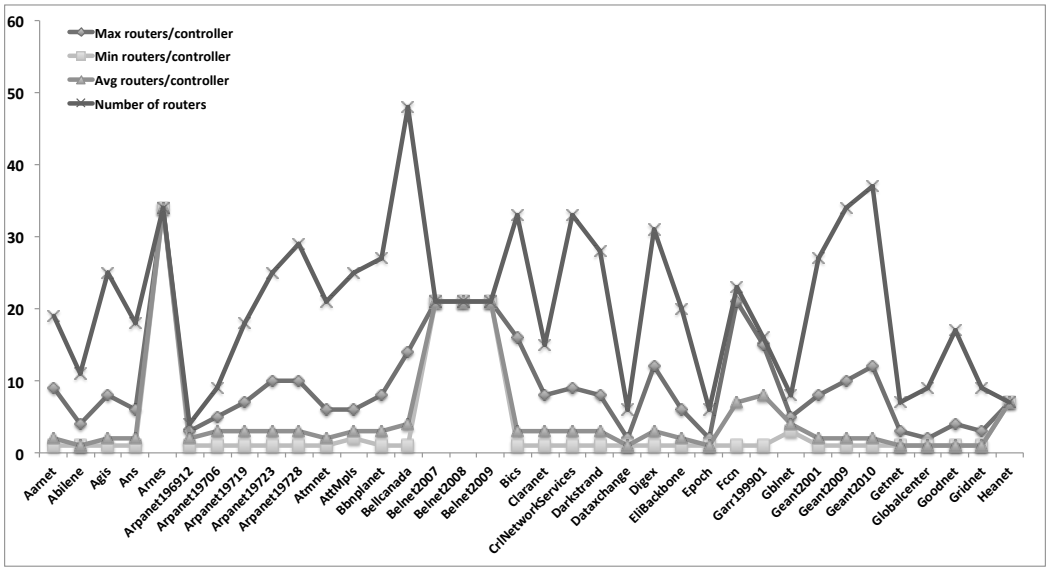

Figure 7: Incremental Greedy method (part I)

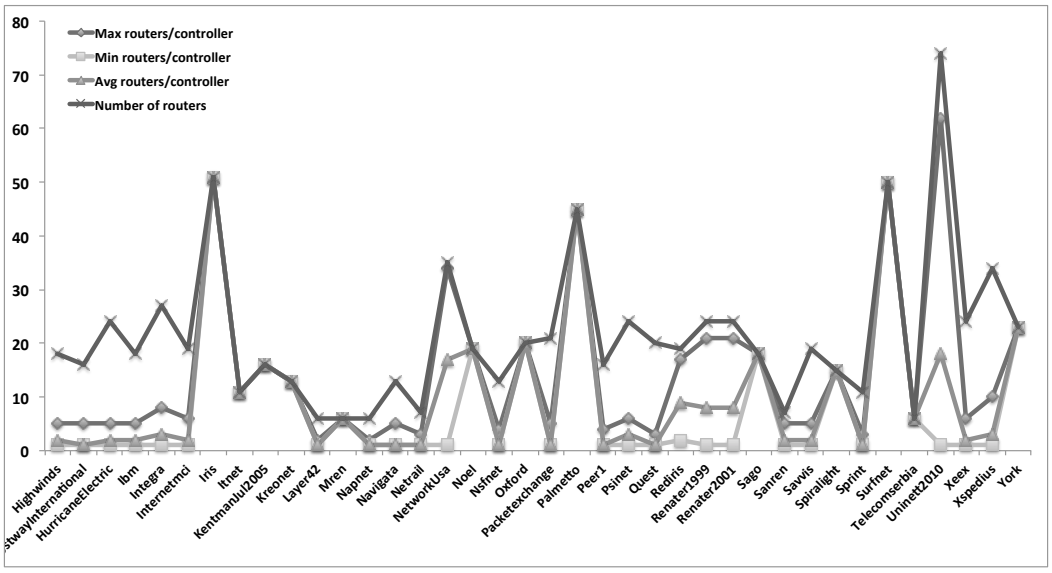

Figure 8: Incremental Greedy method (part II)

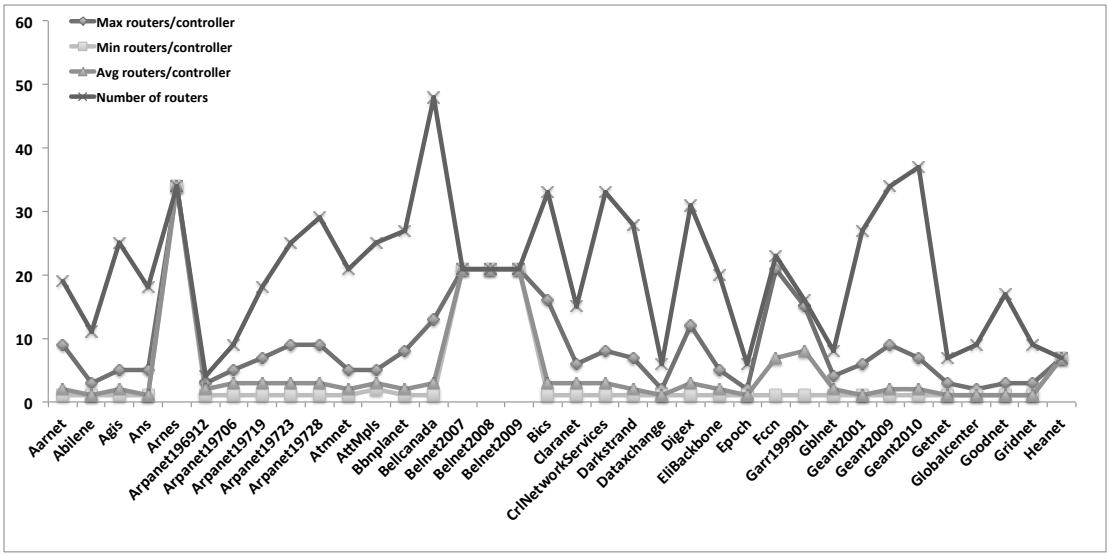

Figure 9: Network-partition-based method (part I)

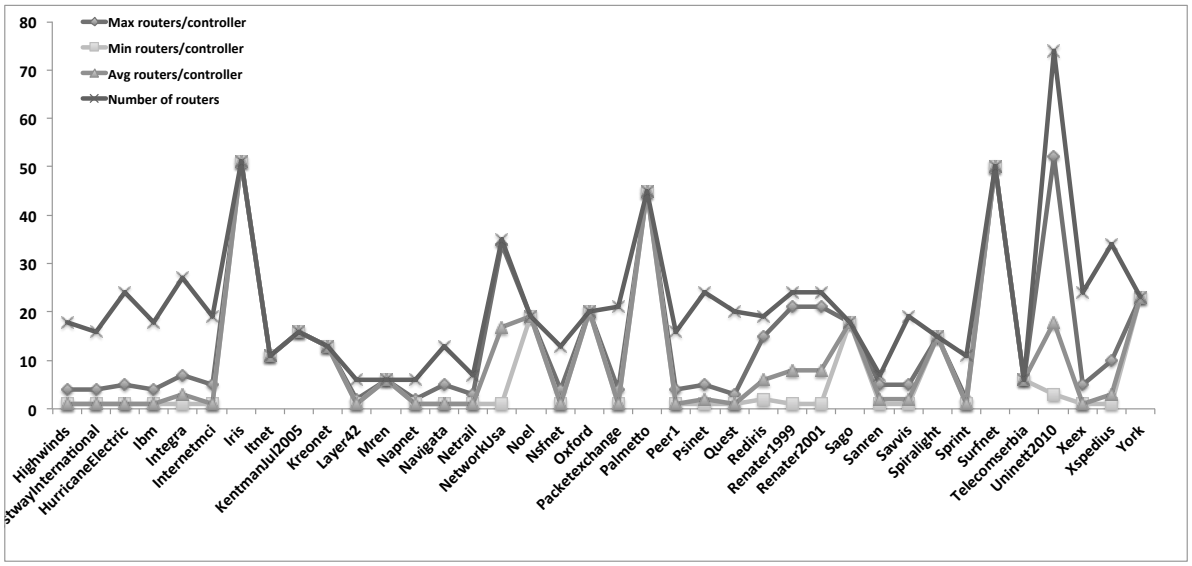

Figure 10: Network-partition-based method(part II)

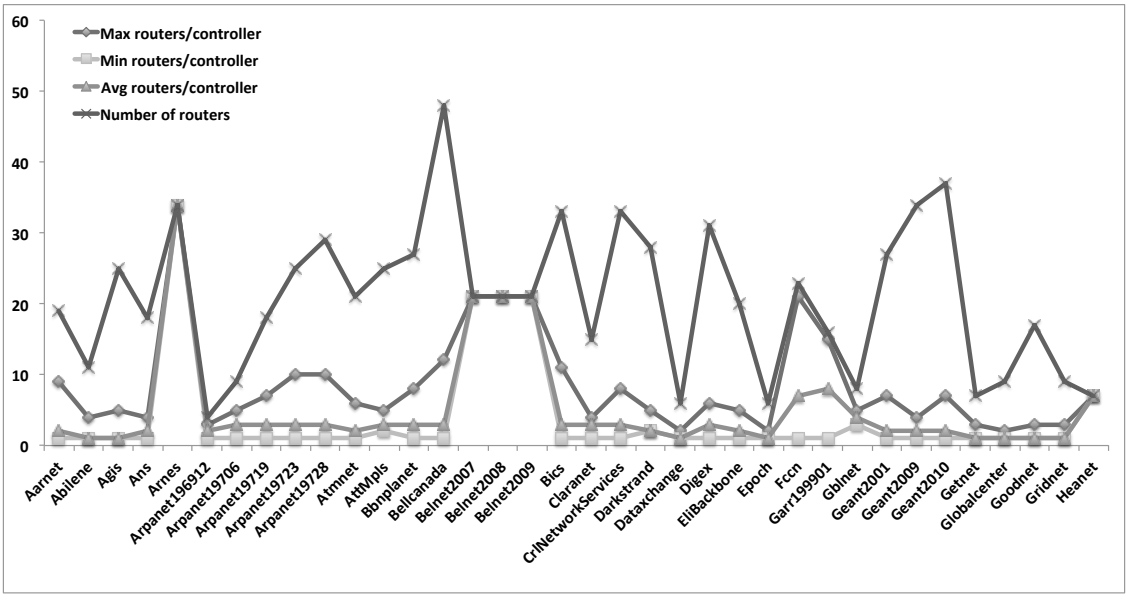

Figure 11: Primal-dual-based method method (part I)

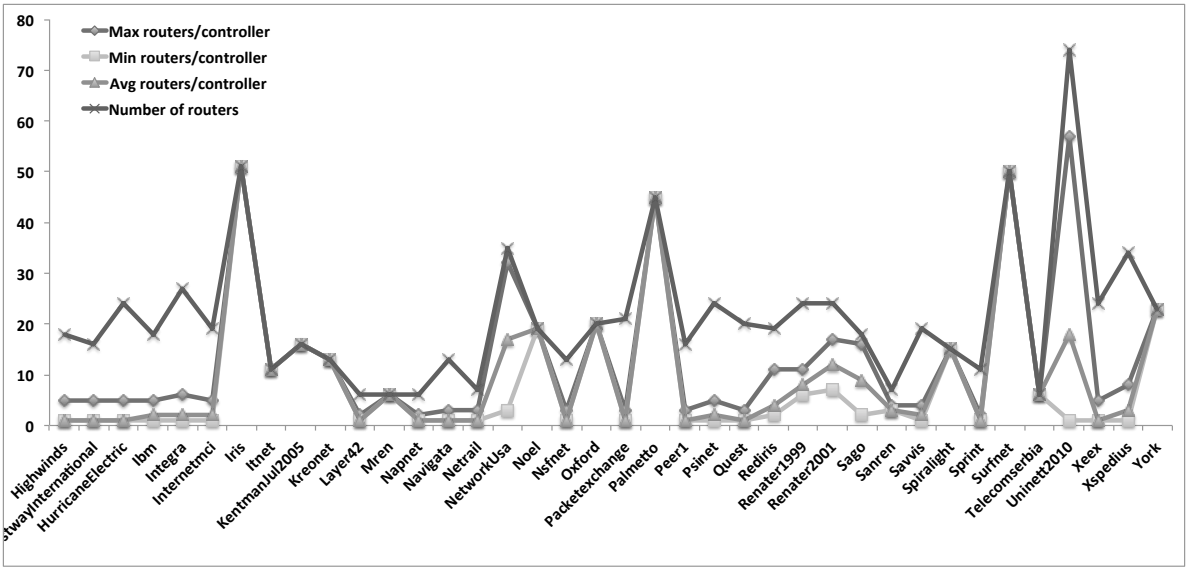

Figure 12: Primal-dual-based method (part II)

7.2.4 QoS Constraints

As one of the inputs of the problem, QoS constraints will affect the decision of how many controllers as well as where these controllers are placed. The higher the quality of service demands, more controllers are required.

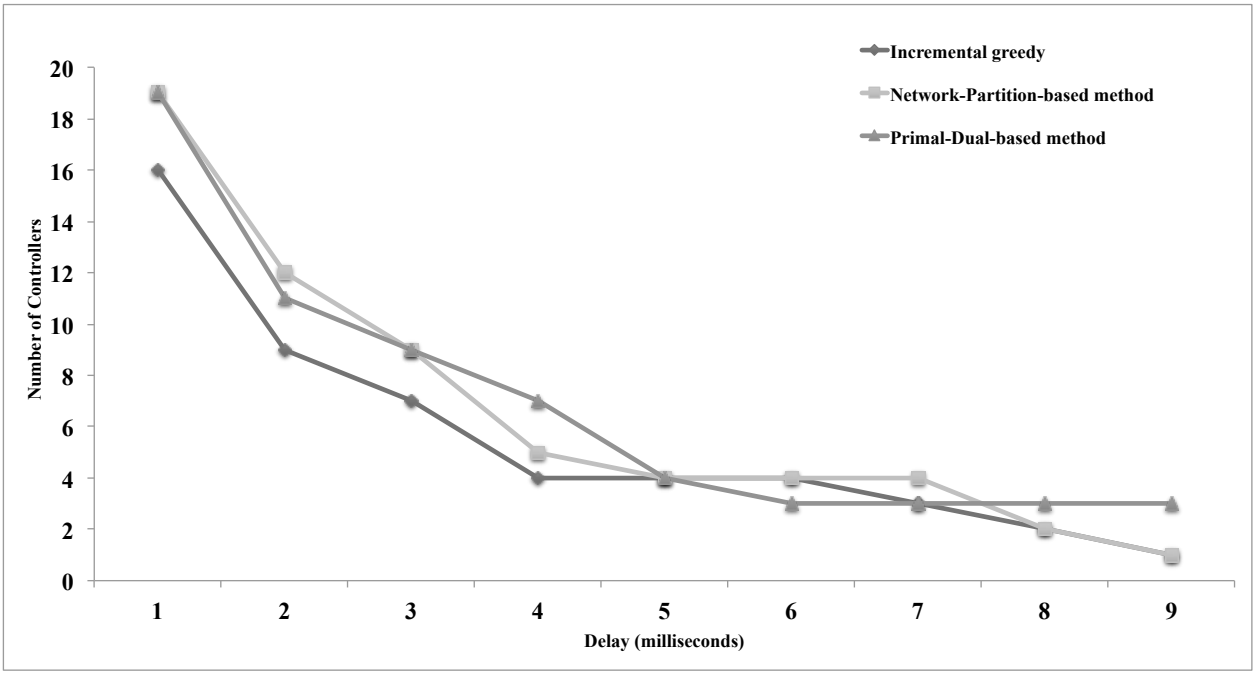

Figure 13: Number of controllers vary with QoS constraints

Figure 13 shows the performance of the proposed three methods under different value of δ in one topology. We choose the network topology named UNINETT2010, which has 74 nodes. For this network topology, we compute 9 different value of δ: 1 ms to 9 ms, as it is shown in the X-axis. Apart from the conclusion mentioned, we can also see that the performance of the three methods are very close to each other under various value of δ.

8 Conclusion and Future Work

8.1 Summary of Achievements

In previous works for controller placement, much concern has been focused on placing k controllers to optimize performance metrics such as propagation latency, load distribution, network reliability. However, in the actual applications, the number of controllers is unknown beforehand. Moreover, the quality of service (QoS) is always a primary concern of the network operators in the placement of SDN controllers. For this reason, we introduce the QoS-Guaranteed Controller Placement problem: Given a network topology and a response time bound, how many controllers need to be instantiated, where they are deployed, and which network nodes are under control of each of the controller, in order to guarantee the response time between a node and its subscribed controller is less than the threshold. We propose three heuristic algorithms to solve this controller placement problem with the required response time constraint. The proposed heuristic methods are compared on the Internet Topology Zoo, a public network topology dataset. The simulation results demonstrate that the proposed three heuristic methods have similar performance on all input topologies.

8.2 Critical Review

This is the first time that I work on a research project in the domain of computer networks. At the initial stage, I felt difficult for me to find relevant reference materials to approach the problem. My supervisor provided me with a lot of mathematical insights on the methodologies of different problems in networks. Apart from the meeting with my supervisor, I also read a lot of reference books on combinatorial optimization, which paved my way for the design of approximation algorithms.

This research project can be generalized as the following steps. First we formulate the SDN

problem into integer programs. After that we can perform linear programming relaxation on the integer programs. We can therefore design approximation algorithms by the linear programming relaxation. However, it has been well proved that there is a integrality gap known as approximation ratio to this solution. This interferes us from obtaining the optimal solution. In addition, approximation algorithms designed for another problem formulation may yield different outcome, which we will discuss in the following section.

8.3 Future Extension

This research work can be further investigated in the following perspectives.

8.3.1 Problem Formulation

We formulated this problem as the set cover problem. However, there are some other problems in the domain of operations research which may provide us with insights to solve the QoS-guaranteed controller placement problem. One typical problem is called $Facility\,Location\,Problem$, which is defined as the optimal placement of facilities to minimize the transportation costs. In the previous work, the number of facilities are fixed to be k. We will investigate the facility location problem in the both the domains of minimizing the number of facilities and optimal placement of facilities.

Another problem called $minimum\ cost\ multi-commodity\ flow\ problem$ is also similar to the QoS-guaranteed controller placement problem, which is defined as finding the cheapest way to send commodities from sources to destinations. If we view each router as both the source and the destination, the commodity flow should be routed through at least one controller. We will also investigate the feasibility of solving the min-cost multi-commodity flow network problem.

8.3.2 Fault-tolerance SDN

In this work, we discussed QoS-guaranteed controller placement problem in terms of the response time. There are some other concerns in the controller placement in SDN including how to build a fault-tolerant network topology so that the fault recovery time is less than a threshold. These problems will be further investigated in the future.

9 Monthly log

Month	Log
October	Study the project background including ow the original OpenFlow architecture performs load balancing, How the new proposed controlling mechanisms optimise the load balancing, How to set up models for derive the input-output relationship
November	Analyzed problem modeling in Network Traffic Engineering and defined the Fat-Tree Topology in the environment
December	Studied the analytical model of link stability analysis in Data Center Networks and the analytical model of minimum active switches in Data Center Networks
January	Studied the controller placement problem scope and worked out the greedy algorithm in QoS quaranteed controller placement problem; Prepared and processed the dataset and implemented the algorithm using Python
February	Formulate the problem into a linear programming problem and studied the design of an approximation algorithm in solving a linear programming problem
March	Proposed the other two heuristic algorithms and proved their efficiency and performance, implemented and refined all three heuristic algorithms, plotted the performance metrics diagrams
April	Analyzed the performance of the three heuristic algorithms

References

1. Marc Mendonca, Bruno Astuto A Nunes, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: past, present, and future of programmable networks. *hal-00825087*, 2013.

2. Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 7–12. ACM, 2012.

3. Francisco Javier Ros and Pedro Miguel Ruiz. Five nines of southbound reliability in software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 31–36. ACM, 2014.

4. Guang Yao, Jun Bi, Yuliang Li, and Luyi Guo. On the capacitated controller placement problem in software defined networks. 2014.

5. Yury Jimenez, Cristina Cervello-Pastor, and Aurelio J Garcia. On the controller placement for designing a distributed sdn control layer. In *Networking Conference, 2014 IFIP*, pages 1–9. IEEE, 2014.

6. David Hock, Matthias Hartmann, Steffen Gebert, Michael Jarschel, Thomas Zinner, and Phuoc Tran-Gia. Pareto-optimal resilient controller placement in sdn-based core networks. In *Teletraffic Congress (ITC), 2013 25th International*, pages 1–9. IEEE, 2013.

7. Prabhakar Raghavan and Clark D Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

8. Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On controller performance in software-defined networks. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, volume 54, 2012.

9. Sakir Sezer, Sandra Scott-Hayward, Pushpinder-Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for sdn? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43, 2013.

10. Donald Gross. *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

11. Simon Knight, Hung X Nguyen, Nick Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *Selected Areas in Communications, IEEE Journal on*, 29(9):1765–1775, 2011.

12. Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 254–265. ACM, 2011.

13. Arsalan Tavakoli, Martin Casado, Teemu Koponen, and Scott Shenker. Applying nox to the datacenter. In *HotNets*, 2009.