Mathematical Analysis of Traffic Engineering in Software Defined Network

Heewon Kim, Jiwon Seo, and Chohong Min

***Abstract*—Software-Defined Networking (SDN) allows network administrators to dynamically control and manage network behavior. In a seminal work, Agarwal et al. [1] introduced traffic engineering in SDN to improve network resource utilization, and Guo et al. suggested a further improvement [11] by optimizing the weight setting of the OSPF protocol. The optimization was known as an NP-complete problem, and the search for the optimum involved a brute-force search.**

In this paper, our main argument is that two types of unnecessary searches are performed for the optimization. One type occurs in links that do not carry traffic, and the other in links that do not share traffic with the maximum utilization link. Skipping the unnecessary ones, the computational cost can be reduced by about 30%. Our argument is supported by a complete mathematical analysis and is validated by examples from real-world ISP network topology.

***Index Terms*—traffic control, mathematical analysis, network theory (graphs)**

I. Introduction

Every time new IT trends surface, such as Big Data, Internet of Things (IoT) and Cloud Computing, there is a keyword that always appears. It's 'Network'. According to the Cisco Visual Networking Index Forecast and Service Adoption for 2014 to 2019 (Cisco VNI Report) [5], annual global IP traffic is expected to increase 3 times between 2014 and 2019. Also, according to the report, worldwide IP traffic will reach 168 EB per month in 2019, reaching 59.9 EB per month (exabytes, 1 billion gigabytes) in 2014. Data coming and going on a network is exploding, and naturally, network management technology has become an important issue.

Traffic engineering is always an essential consideration for large-scale networks that optimize network operation [3]. Open Shortest Path First (OSPF) is the most commonly used Internet routing protocol[18], and there are many studies on the optimal OSPF weight setting problem that aim to optimize the network performance [8], [9]. However, as the networking environment becomes increasingly demanding and rapidly changing, the network performance of the OSPF routing protocol reaches limits. Since existing network equipment and facilities are fixed, packet delay and losses cannot be avoided if traffic changes rapidly. Therefore, the environmental changes and increasing traffic demands have resulted in a need for a Software-Defined Network (SDN) [4], [17] .

Department of Mathematics, Ewha womans university, Seoul, Korea
E-mail: chohong@ewha.ac.kr

SDN allows network administrators to control and manage network behavior dynamically by separating the network control plane from the packet forwarding plane. To create an SDN environment, an open interface must be created to allow seamless communication for each networking command. Open Flow [17] is an element that makes up the SDN, and it is the standard interface responsible for communication between the control device and the networking switch. An SDN consists of two main components, one is the SDN controller and the other is the Forwarding Element. The controller is a logically centralized function [10], [14] that determines the forwarding path for each flow, and the forwarding element constructs the data plane for the network. The forwarding element forwarding the packets according to the forwarding path is determined in the controller, and since SDN is a new networking paradigm that simplifies network management and dramatically improves the network resource utilization, traffic engineering in SDN is an important issue [2], [15].

Agarwal et al. [1] considered unicast traffic engineering in the case where the SDN controller controls only a few SDN forwarding elements in the network. The rest of the network (non-forwarding elements) does standard hop-by-hop routing using OSPF. In this paper, we refer to the algorithm traffic engineering *Phase I*. By deploying forwarding elements, these produce improvements in the delay and a loss in performance of the network. However, in terms of the non-forwarding elements, a traditional OSPF routing protocol(a hop-by-hop shortest path) is not the best option. We can reduce the maximum link utilization by optimizing the OSPF weights.

Guo et al. [11] optimized the OSPF weights in SDN to balance the flow. Even though they obtain an improvement in network performance when compared to the traffic engi-

neering in a traditional OSPF network [9] and SDN/OSPF network with fixed weight setting [1], since the optimization of the weight setting is an NP-complete problem [20], they suggest a local search heuristic algorithm. The algorithm is a brute-force algorithm that tries all integer weights on the edge to find the minimum of the maximum utilization. We call them traffic engineering *Phase II*.

In this paper, we found many unnecessary searches in traffic engineering Phase II. We identify edges where the increase in the edge weight does not lead to an improvement in the minimum of the maximum utilization. As a result, we were able to reduce the computations by 30% in the existing method and obtained a considerable improvement in speed. Throughout this paper, we use standard definitions from Graph theory, and the main purpose of this paper is to gain a speed-up in running Phase II. In Section 2 and 3, we briefly review the survey of the traffic engineering Phase I and Phase II, respectively. We compare the maximum link utilization of the three algorithms: traditional OSPF, traffic engineering Phase I, and traffic engineering Phase II. In Section 4, we perform a mathematical analysis of the traffic engineering algorithms and present an improvement for Phase II. We first show that the function for the minimization is quite nasty to deal with. We also analyze the algorithm in Phase II and suggest an enhanced algorithm to speed-up Phase II by identifying unnecessary searches and skipping. In Section 5, we present experimental results using a real network topology [19] and compare the maximum link utilization and computational numbers. Section 6 provides the conclusion and summary of this paper as well as suggestions for future work.

II. Traffic engineering : Phase I

In this section, we briefly review the basic ideas of the traffic engineering introduced by Agarwal et al. [1]. The basic traffic engineering, which we call *Phase I*, will be improved in the next section in *Phase II*.

The network in consideration is represented by an undirected graph $G = (V, E)$. In this paper, we focus on unicast communications. One vertex is the destination, and the other vertices are sources. Each source vertex s has T_{sd} amount of data to be delivered to the destination vertex d. Each edge e has a capacity $c(e)$, the limit of data flow through it. Figure 1 illustrates a typical example from [1] .

A standard routing protocol, called OSPF[18], sends data through the shortest paths. A default distance at a source is the minimum number of *hops* to reach the destination. An important management of network is to keep the data flow on each edge under capacity. Otherwise, the network

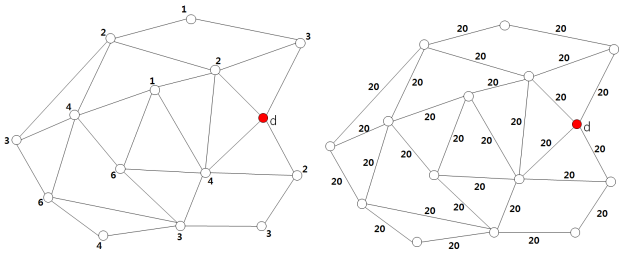

Figure 1: The network is represented by an undirected graph $G = (V, E)$ with T_{sd} on vertices (left) and the capacity $c(e)$ on edges (right).

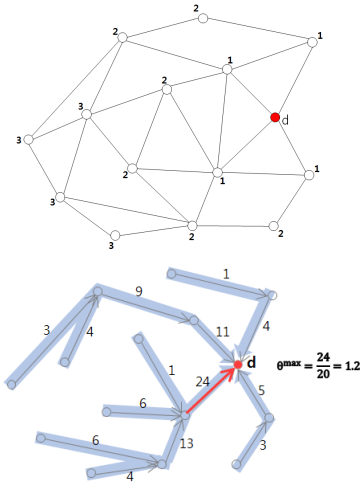

Figure 2: Hop-by-hop OSPF network : hop-by-hop distances on vertices (top) and the network flow (bottom). The data flow on each edge is the sum of T_{sd}'s in Figure 1. θ^{max} is attained at the edge marked red.

will experience packet delays and packet loss. Thus, the maximum-utilization-rate θ^{max}, defined below, is required to be smaller than 1.

$$\begin{cases} \text{utilization rate at edge } e, & \theta(e) & := \frac{\text{data flow at } e}{\text{capacity } c(e)} \\ \text{maximum utilization rate,} & \theta^{max} & := \max_{e \in E} \theta(e) \end{cases}$$

Figure 2 shows the hop-by-hop distance function and the OSPF network flow. In the network, θ^{max} is 1.2, and needs to be reduced. The traffic engineering places a smart capability to some of the vertices. The smart vertex can forward data to many vertices, and the forwarding can be programmed and controlled. The smart vertex is denoted FE (*Forwarding Element*) vertex and the network is called SDN (*Software Define Network*). Each FE vertex can send data to

the vertices whose hop distances are smaller than or equal to its distance. The edges that can flow data form a directed subgraph $G^{sub} = \left(V, E^{sub}\right)$. With the controllability on FE vertices, the data flow is divided into uncontrollable flow on some edges and controllable flow at the FE vertices. Figure 3 shows the directed subgraph (left) and uncontrollable traffic flow (middle).

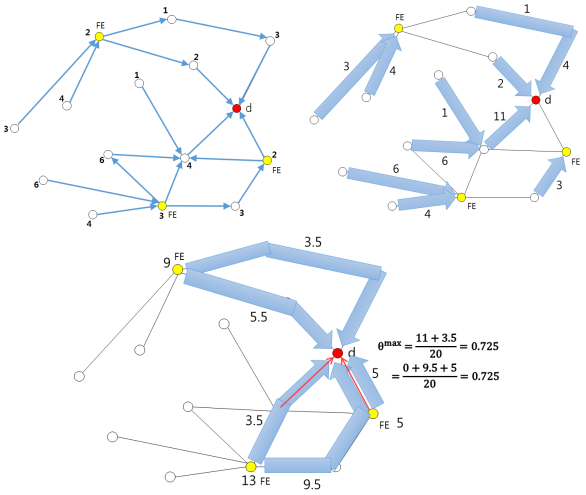

Figure 3: Traffic engineering, Phase-I : directed subgraph with T_{sd} on vertices (top left), uncontrollable traffic flow with $g\left(e\right)$ on edges (top right), and controllable traffic flow with I_{ud} on FE vertices (bottom). θ^{max} is attained at two edges marked red. Now the value is smaller than the non-SDN network in Figure 2.

Due to the controllability, there are many paths $P \in P_{ud}$ that connect the FE vertex u to the destination d. I_{ud} denotes the amount of data that needs to be delivered from u to d, and $x\left(P\right)$ denotes the amount of data that path P delivers. The unknown $x\left(P\right)'s$ are determined to minimize θ^{max} by the following linear programming.

$$\text{minimize } \theta^{max} \text{ subject to}$$
$$\begin{cases} g\left(e\right) + \sum_{e \in P} x\left(P\right) \leq \theta^{max} \cdot c\left(e\right), & \text{for each } e, \\ \sum_{P \in P_{ud}} x\left(P\right) \leq I_{ud}, & \text{for each } u, \\ x\left(P\right) \geq 0, & \text{for each } P. \end{cases}$$

Figure 3 shows the optimal value θ^{max} and the calculated $x\left(P\right)'s$. Note that θ^{max} is reduced from 1.2 to 0.725 by traffic engineering, Phase I.

III. Traffic Engineering : Phase II

In a previous section, we reviewed basic traffic engineering by Agarwal et al. that reduces θ^{max} from 1.2 to 0.725. We called this engineering Phase I. In this section, we review the advanced traffic engineering proposed by Guo et al.[11], and call it Phase II. The Phase II algorithm reduces θ^{max} to 0.55.

As illustrated in Figures 1 and 3, Phase I begins with hop-based distances, and then follows the calculations for subgraph and linear programming. The hop-based distances are the Dijkstra distances with uniform weight $W\left(e\right) \equiv 1$ on the edges.

The main idea of Phase II is to seek a further reduction in θ^{max} by increasing the freedom on the edge weight W. The series of operations of Phase I enables us to think of θ^{max} as a function of weight $W : E \to \mathbb{R}^{+}$. The operations are illustrated in Figure 1 and 3, and summarized below.

Algorithm 1 Traffic engineering, Phase-I:
$\theta^{max} = \theta^{max}\left(W\right)$

Input: weight, $W : E \to \mathbb{R}^{+}$(default $W \equiv 1$)
Output: $\theta^{max} \in \mathbb{R}^{+}$
1: calculate Dijkstra distance; $Dist : V \to \mathbb{R}^{+}$
2: calculate directed subgraph: $G^{sub} = \left(V, E^{sub}\right)$
3: solve the linear programming to minimize θ^{max}

The Phase-II algorithm by Guo et al. is a brute-force algorithm that tries all integer weights on each edge to find the minimum θ^{max}. The algorithm sequentially visits each edge and updates the integer-valued weight, whenever a new minimum is found. The algorithm ends when there is no change on the weight after a visit of all edges. The Phase II algorithm is summarized in Algorithm 2. Figure 4 shows the directed subgraph (top) and uncontrollable traffic flow (bottom left), and the controllable traffic with maximum link utilization (bottom right).

IV. Mathematical Analysis

A. Regularity of the function $\theta^{max}\left(W\right)$

In this section, we perform a mathematical analysis of the traffic engineering algorithms, and present an improvement in Phase II. Our first task is to reveal the regularity of the function $\theta^{max} = \theta^{max}\left(W\right)$ in Phase I. There are numerous optimization techniques, and a proper technique is determined for the regularity of the function to minimize. The following two examples show that the function is quite nasty to deal with.

Example 1. *$\theta^{max}\left(W\right)$ is not generally convex.*

Algorithm 2 Traffic engineering, Phase-II

Input: initial weight, $W : E \to \mathbb{R}^+$ (default $W = \frac{w_{max}}{2}$) range of weights to search, $1 : w_{max}$

Output: $\theta^{max} \in \mathbb{R}^+$

1: Perform Phase I to calculate $\theta^{max} = \theta^{max}(W)$
2: **repeat**
3: **for** each edge **do**
4: find its weight in $1 : w_{max}$ that minimizes θ^{max}.
5: update $W(e)$ with the argument of the minimum.
6: update $\theta^{max}(W)$ with the minimum value.
7: **end for**
8: **until** there is no change in the weights after a visit for all edges

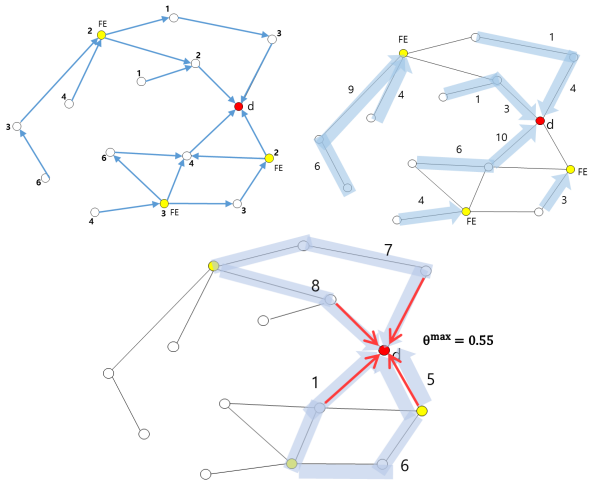

Figure 4: Traffic Engineering, Phase II: directed subgraph with T_{sd} on the vertices (top left), uncontrollable traffic flow with $g(e)$ on the edges (top right), and controllable traffic flow with I_{ud} on the FE vertices (bottom). θ^{max} is attained at the four edges marked in red. Now the value is smaller than the network using SDN Traffic Engineering Phase I in Figure 3.

We present a counter example based on the graph that appears in Section 2. The basic components of graph, a vertex set and an edge set are the same as $G = (V, E)$ in Figure 1 and FE, T_{sd}, and the capacity c are also the same. We first consider the weight function W_1, that gives the integer weight from 1 to 28 on each arbitrary edge where there are no two edges with the same weight. Second, W_2 is a constant weight function that gives the weight 10 to each edge equivalently. If the function $\theta^{max}(W)$ is convex, then

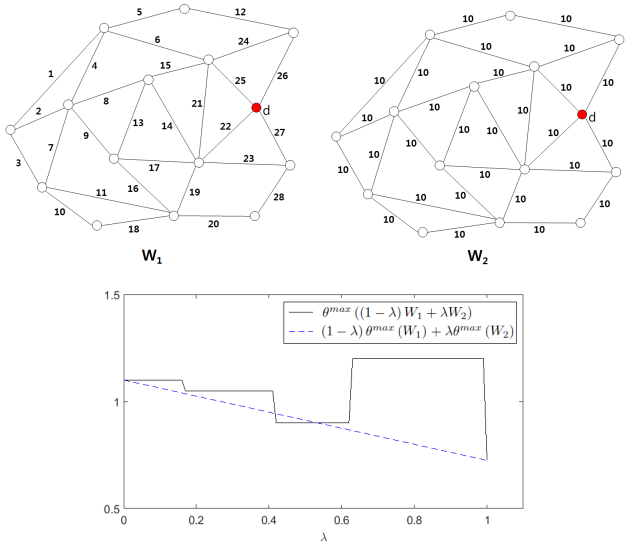

Figure 5: With the weight function W_1 which gives integer weight from 1 to 28 on each arbitrary edge (upper left) and a constant weight function W_2 (upper right), we can check that the function $\theta^{max}(W)$ is not generally convex. The solid line and dashed line in the graph below depict $\theta^{max}((1-\lambda)W_1 + \lambda W_2)$ and $(1-\lambda)\theta^{max}(W_1) + \lambda\theta^{max}(W_2)$, respectively, with respect to a change in λ.

$$\theta^{max}((1-\lambda)W_1 + \lambda W_2) \leq (1-\lambda)\theta^{max}(W_1) + \lambda\theta^{max}(W_2), \forall\lambda \in [0,1].$$

However, as we can see in Figure 5, there exists $\lambda \in [0,1]$ such that $\theta^{max}((1-\lambda)W_1 + \lambda W_2) > (1-\lambda)\theta^{max}(W_1) + \lambda\theta^{max}(W_2)$. Therefore, $\theta^{max}(W)$ is not generally convex.

Example 2. *$\theta^{max}(W)$ is not generally continuous.*

Fixing $W \equiv 1$ for all edges except one, say e^*. We slightly increase the weight of e^* from 0.01 to 4, we can find a jump discontinuity near $W(e^*) = 2$ which is shown in Figure 6. And therefore, $\theta^{max}(W)$ is not generally continuous either.

The two examples show that we cannot use the most of efficient optimization techniques, such as Bisection method, Gradient descent method, Newton method and their various analogues. Thus the brute-force search of Phase-II makes sense regarding the bad regularity of the function.

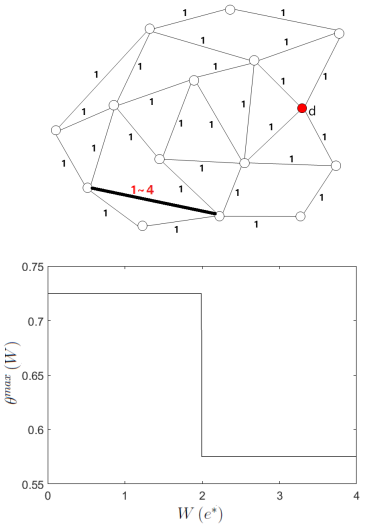

Figure 6: As we slightly change the weight of one edge (which is colored red in figure on the top) with all the other fixed, we can check that $\theta^{max}(W)$ is not continuous (bottom).

B. Enhancement of Phase II

Our next task is to analyze the Phase II algorithm. In the brute-force search, we found that there are many unnecessary searches. In this section, we identify the cases when the change from W to $\widetilde{W}$ results in $\theta^{max}(W) \leq \theta^{max}\left(\widetilde{W}\right)$. Such cases are not helpful in reducing θ^{max}. We propose skipping such unnecessary searches, and as a result, gain a speed-up in running Phase-II.

Let $W : E \to \mathbb{R}^+$ be the current weight, from which Phase-I calculates the following, as in Figure 1 and 2.

· Distance function : $Dist : V \to \mathbb{R}^+$
· Subgraph : $G^{sub} = \left(V, E^{sub}\right)$
· Linear programming : θ^{max}

Phase-II changes a weight $W(e)$, while the other weights are fixed. We denote $\widetilde{W}(e)$ as the new weight and the change will bring about an update on the above results.

· Updated Distance function : $\widetilde{Dist} : V \to \mathbb{R}^+$
· Updated Subgraph : $\widetilde{G^{sub}} = \left(V, \widetilde{E^{sub}}\right)$
· Updated Linear programming : $\widetilde{\theta^{max}}$

Throughout the paper, we use standard definitions in Graph theory. A path is a sequence of distinct edges connected by common vertices, and its length is the sum of the edge weights. The distance at a vertex is the minimum length of paths that connects the vertex to the destination. Given two vertices v, w, we define the set of paths connecting v to w in G and G^{sub} as $P_{v\to w}$ and $P^{sub}_{v\to w}$, respectively.

Our first result is that Phase-II does not have to try larger weights whenever the edge does not belong to E^{sub}.

Theorem 1. *If $e \in E - E^{sub}$ and $\widetilde{W}(e) \geq W(e)$, then $\widetilde{\theta^{max}} \geq \theta_{max}$.*

Proof. It is enough to show that $Dist = \widetilde{Dist}$, then follows $G^{sub} = \widetilde{G^{sub}}$ and $\theta^{max} = \widetilde{\theta^{max}}$. For any $v \in V$,

$$\begin{aligned} Dist(v) &= \min_{P \in P_{v\to d}} length(P) \\ &= \min\left(\min_{\substack{e \in P \\ P \in P_{v\to d}}} length(P), \min_{\substack{e \notin P \\ P \in P_{v\to d}}} length(P) \right) \\ &= \min_{\substack{e \notin P \\ P \in P_{v\to d}}} length(P) \end{aligned}$$

There is a shortest path in G^{sub} and $e \notin G^{sub}$. Only the weight of e increases, and the $\widetilde{length}(P) = length(P)$ when $e \notin P$, and $\widetilde{length}(P) > length(P)$ when $e \in P$. Now we prove $\widetilde{Dist}(v) = Dist(v)$.

From the construction of G^{sub}, there is a shortest path from v to d in G^{sub}. Since $e \notin G^{sub}$, $Dist(v)$ equals $length(P)$ for some $P \in P^{sub}_{v,d}$ that does not include e. Only the weight of e increases. Thus $\widetilde{length}(P) = length(P)$ when $e \notin P$, and $\widetilde{length}(P) > length(P)$ when $e \in P$.

$$\begin{aligned} \widetilde{Dist}(v) &= \min_{P \in P_{v\to d}} length(P) \\ &= \min\left(\min_{\substack{e \in P \\ P \in P_{v\to d}}} \widetilde{length}(P), \min_{\substack{e \notin P \\ P \in P_{v\to d}}} \widetilde{length}(P) \right) \\ &= \min\left(\min_{\substack{e \in P \\ P \in P_{v\to d}}} \widetilde{length}(P), \min_{\substack{e \notin P \\ P \in P_{v\to d}}} length(P) \right) \\ &= \min_{\substack{e \notin P \\ P \in P_{v\to d}}} length(P) = Dist(v) \end{aligned}$$

□

Theorem 1 identifies unnecessary searches when $e \in E - E^{sub}$. Now, let us turn our attention to cases when $e \in E^{sub}$. In the directed graph G^{sub}, a connected component is a

maximal set of vertices in which any two vertices are connected by at least a path. Throughout this paper, we do not include the destination in the connected components and intentionally separate G^{sub} into many components. Otherwise, there is only one connected component. Let us denote by C_{max}^{sub}, the connected component that contains the edge e^{max} at which θ^{max} occurs.

Lemma 1. *Either there is no controllable flow on e^{max}, or there exists an FE vertex u such that $\max_{e\in P}\theta(e)=\theta^{max}$, for every $P\in P_{u\to d}$.*

Proof. Assume there is a controllable path $P\in P_{u\to d}$ that passes through e^{max}. If there is another path $Q\in P_{u\to d}$ that has $\max_{e\in Q}\theta(e)<\theta^{max}$, we may decrease $X(P)$ and increase $X(Q)$ by the same amount to decrease $\theta(e^{max})$. Since the linear programming finds the optimal solution, there cannot be such FE vertex u. □

Theorem 2. *If $e\in E^{sub}-E\left(C_{max}^{sub}\right)$ and $\widetilde{W}(e)\geq W(e)$, then $\widetilde{\theta^{max}}\geq\theta^{max}$.*

Proof. First thing we need to show is that $Dist=\widetilde{Dist}$ on C_{max}^{sub}. For each vertex $v\in C_{max}^{sub}$, there is a shortest path P in C_{max}^{sub} that connects v to d. Hence $Dist(v)=length(P)$. Since $e\notin E\left(C_{max}^{sub}\right)$ by the assumption, $\widetilde{length}(P)=length(P)$. The weight increase, in general, increases all other lengths, so $\widetilde{Dist}(v)=Dist(v)$. From this observation, we can also find that the internal edges of C_{max}^{sub} stay the same after the weight increase.Therefore C_{max}^{sub} stays connected in $\widetilde{G^{sub}}$. Furthermore, the following is that since $\widetilde{Dist}\geq Dist$ in $V-C_{max}^{sub}$, there can be only an additional inbound edge to C_{max}^{sub} in $\widetilde{G^{sub}}$.

By Lemma 1, either there is no controllable flow on e^{max}, or there exists an FE vertex u such that $\max_{e\in P}\theta(e)=\theta^{max}$, for every $P\in P_{u\to d}$. In the former case, $g(e^{max})$ is the sum of all T_{sd}'s that come to e^{max}. Since there is only an additional inbound edge to C_{max}^{sub} in $\widetilde{G^{sub}}$, $\widetilde{g}(e^{max})\geq g(e^{max})$, and $\widetilde{\theta^{max}}\geq\theta^{max}$. In the latter case, there is no room to increase $X(P)$ in every $P\in P_{u\to d}$. Furthermore, $\widetilde{I_{u,d}}\geq I_{u,d}$ because of the inflow. Therefore $\widetilde{\theta^{max}}\geq\theta^{max}$. □

Using Theorem 1 and 2, we present a new algorithm that reduces the iteration number. In this algorithm, we do not make unnecessary searches for edges e where an increase in $W(e)$ does not lead to an improvement in utilization. Our algorithm is summarized in Algorithm 3, and the numerical result is illustrated in Figure 8. As we can see in the figure, the Enhancement of Phase-II shows the same improvement

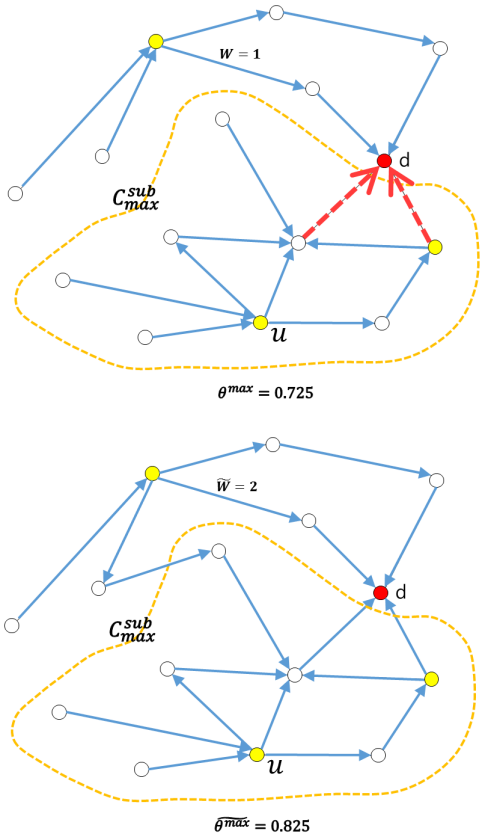

Figure 7: The first picture shows G^{sub} and the connected component C_{max}^{sub} in the example. As stated in Lemma 1, every path $P\in P_{u\to d}$ goes through one of e^{max}'s. When the weight increases outside the component, as stated in Theorem 2, C_{max}^{sub} stays the same and only the inbound edges can be added to it; see the second picture.

in utilization compared to that in Phase-II, but the number of iterations is considerably reduced. The total number of iterations of Phase-II is 820 and that for Phase II-E is 654. We can see that the Enhancement in the Phase-II algorithm reduces the total iteration count by about 20.3% compared to the Phase-II algorithm.

V. Experiments and Evaluation

In this section, we apply the presented algorithms to real-world network examples and validate the arguments in Theorems 1 and 2. Four examples are taken from Rocketfuel data [16]. The specifications of the examples are listed in Table I. For each example, the link capacity is taken as the

Algorithm 3 Enhancement of Phase-II

Input: weight, $W : E \to \mathbb{R}^+$(default $W \equiv 1$)
maximum value of weight, w_{max}
Output: $\theta^{max} \in \mathbb{R}^+$

Perform Phase I to calculate $\theta^{max} = \theta^{max}(W)$
repeat
  for each edges **do**
    if an edge is not in E^{sub} or C^{sub}_{max} **then**
      $w_{curr} \leftarrow$ current weight on the edge
      find the weight which minimizes $\theta^{max}(W)$, from 1 to w_{curr}
    else
      find the weight which minimizes $\theta^{max}(W)$, from 1 to w_{max}
    end if
    update W with the argument of the minimum
    update $\theta^{max}(W)$ with the minimum value
  end for
until there is no change in the weight after a visit for all edges

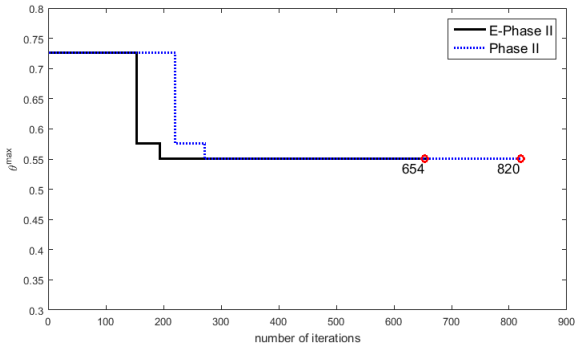

Figure 8: The utilization rate θ^{max} is plotted in each iteration using E-Phase II (solid line) and Phase-II (dotted line). The number of total iterations is posted for each algorithm.

number	name	nodes	links
1755	EBONE(Europe)	87	322
3257	Tiscali(Europe)	161	656
6461	Abovenet(US)	138	744
1239	Sprintlinks(US)	315	1944

Table I: Network examples from Rocketfuel

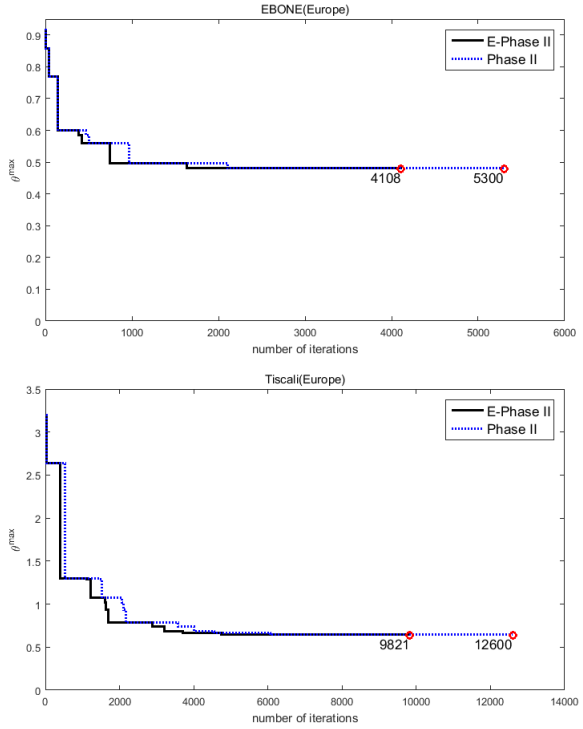

Figure 9: Experiments of smaller examples, EBONE (top) and Tiscali (bottom). The utilization rate θ^{max} is plotted in each iteration using E-Phase II (solid line) and Phase-II (dotted line). The number of total iterations is posted for each algorithm.

inverse of the link cost, and 10% of the nodes are randomly chosen and selected as forwarding elements, as in [11].

Theorems 1 and 2 state two types of unnecessary searches in the Phase II algorithm. We denote by Enhanced Phase II, the algorithm without unnecessary searches. Through the experiments, we intend to verify that Phase II and E-Phase II generate the same results and measure the speed-up from skipping unnecessary searches.

Figure 9 shows the results of the two smallest examples, EBONE and Tiscali. The solid line (E-Phase II) and dotted line (Phase II) undergo the same set of θ^{max} values. E-Phase II reaches each θ^{max} value earlier than Phase-II. The two algorithms end when there is no change in the visit of all edges. The computational costs are thus proportional to the number of iterations at the end. E-Phase II reduces the computational cost of Phase II by 22.5% and 22.1% in the two examples.

We remark that Phase II and E-Phase II spend about the same computational cost for each iteration. The only addition to E-Phase II is to calculate the connected component C^{sub}_{max}. It is straightforward to compute the connected component

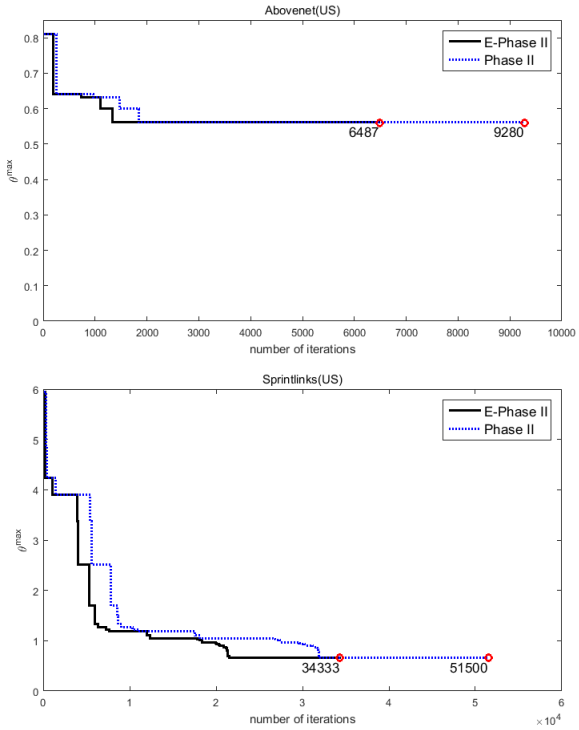

Figure 10: Experiments of larger examples, Abovenet (top) and Sprintlinks (bottom). The utilization rate θ^{max} is plotted in each iteration using E-Phase II (solid line) and Phase-II (dotted line). The number of total iterations is posted for each algorithm.

in linear time using a breadth-first-search. The addition was negligible compared to the other computations, such as with the Dijkstra algorithm and linear programming.

Figure 10 shows the results of the two larger examples, Abovenet and Sprintlinks. Also E-Phase II and Phase II, as stated in the theorems, generate the same utilization values. Now E-Phase II reduces the computational cost of Phase-II by 30.1% and 33.4%. We note that the amount of reductions become larger as the network size grows, in the tried examples.

ACKNOWLEDGEMENT

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2009-0093827)

VI. CONCLUSION AND FUTURE WORK

In this paper, we conducted a mathematical analysis on traffic engineering in SDN, mainly based on graph theory. In traffic engineering, since the optimal weight setting problem is NP-complete, previous studies proposed an algorithm based on brute-force, which tries all integer weights on the edge to find the minimum of the maximum utilization. In our research, we first observed that the optimal weight setting problem is not regular enough to exploit the existing efficient optimization solvers. After that, we conducted a mathematical analysis on one of the existing algorithms, proposed by Guo et al.[11]. As a result, we were able to improve the speed by skipping unnecessary searches. On the links that are not included in the set E^{sub} or the set C^{sub}_{max}, it is unnecessary to try a larger weight than the one that is currently set. Based on this theory, we proposed an improved algorithm, with a computation volume reduced by about 30%. There can be a further method to speed up the algorithm. For example, on the links contained in C^{sub}_{max}, it is better not to consider reducing the link weight. We understand this phenomenon intuitively, but we will leave it as future work. Furthermore, different to our work where we experimented on unicast communication network with one destination and several sources, if our theory is applied to multicast traffic engineering [12] that sends traffic to multiple destinations, we expect to see improvements as well.

REFERENCES

[1] S. Agarwal, M. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2211–2219, April 2013.

[2] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30, 2014.

[3] D. O. Awduche. Mpls and traffic engineering in ip networks. *IEEE Communications Magazine*, 37(12):42–47, Dec 1999.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, August 2007.

[5] VNI Cisco. Cisco visual networking index: Forecast and methodology, 2014–2019 white paper, 2015.

[6] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. On the effect of forwarding table size on sdn network utilization. In *INFOCOM, 2014 Proceedings IEEE*, pages 1734–1742. IEEE, 2014.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[8] M. Ericsson, M.G.C. Resende, and P. M. Pardalos. A genetic algorithm for the weight setting problem in ospf routing. *Journal of Combinatorial Optimization*, 6(3):299–333, 2002.

[9] B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 2, pages 519–528 vol.2, 2000.

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M.Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[11] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu. Traffic engineering in sdn/ospf hybrid network. In *2014 IEEE 22nd International Conference on Network Protocols*, pages 563–568, Oct 2014.

[12] L. H. Huang, H. C. Hsu, S. H. Shen, D. N. Yang, and W. T. Chen. Multicast traffic engineering for software-defined networks. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[13] M. Huang, W. Liang, Z. Xu, W. XU, S. Guo, and Y. Xu. Dynamic routing for network throughput maximization in software-defined networks. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S.Shenker. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[15] Y. Liu, Y. Li, Y. Wang, and J. Yuan. Optimal scheduling for multi-flow update in software-defined networks. *Journal of Network and Computer Applications*, 54:11–19, 2015.

[16] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, pages 231–236, New York, NY, USA, 2002. ACM.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[18] J. Moy. Ospf version 2. 1997.

[19] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, pages 133–145, New York, NY, USA, 2002. ACM.

[20] Y. Wang, Z. Wang, and L. Zhang. Internet traffic engineering without full mesh overlaying. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 1, pages 565–571 vol.1, 2001.