Efficient Heuristics for Energy-Aware Routing in Networks with Bundled Links

Gongqi Lin*, Sieteng Soh*, Kwan-Wu Chin**, Mihai Lazarescu*
*Department of Computing, Curtin University of Technology, Perth, WA, Australia
**University of Wollongong, Northfields Ave, Wollongong, NSW, Australia

Abstract—Current networks are typically over-provisioned to ensure low delays, redundancy and reliability. These Quality of Service (QoS) guarantees are typically achieved using high end, high power network equipments. Their use, however, has led to concerns regarding green house gas emissions, which garnered a lot of attention recently and have resulted in a number of global initiatives aim at reducing the carbon footprint of Internet Service Providers (ISPs). These initiatives have motivated ISPs and researchers to design novel network algorithms and hardware that scale the usage or active time of a network according to traffic load. To this end, this paper considers the problem of shutting down a subset of bundled links during off-peak periods in order to minimize energy expenditure. Unfortunately, identifying the cables that minimize this objective is an NP-complete problem. Henceforth, we propose several practical heuristics based on *Dijkstra*'s algorithm and *Yen*'s *k*-shortest paths algorithm. We evaluated our heuristics on the Abilene network - with both real and synthetic traffic matrices and several larger random topologies with various loads. Our results show that the proposed heuristics to be effective and efficient. Moreover, our approaches could potentially reduce the energy usage of cables used in the Abilene network by up to 56.7%, assuming the traffic demands recorded on September 5, 2004.

Index Terms— Green networks, Bundled links, *Dijkstra*'s algorithm, Optimization.

1. INTRODUCTION

Today's computer network infrastructures around the world consume non-negligible amount of energy. For example, the energy usage of the network infrastructures in Italy in 2006 exceeded 1.4 TWh, which is approximately 0.7% of the total energy usage [1]. Other examples include Verizon, where in 2006, it consumed 8.9 TWh (about 0.26% of USA

energy requirements), while Telecom France recorded 2 TWh [2]. British Telecom reported that the overall energy consumption for its network and estate during the 2008 financial year to be 2.6 TWh, making it the biggest single energy consumer in the country [3].

These energy consumption figures are expected to increase further given that today's networks are designed to support the maximum number of customers whilst meeting their Quality of Service (QoS) requirements without any considerations for energy efficiency. These goals are usually achieved by building many redundant links and adequately over-provisioning and engineering links to ensure low delays, and to absorb any rise in traffic resulting from link failures or key events. For instance, high-end IP routers use complex multi-rack architectures that are able to support increasing network functionalities and network traffic, which increases 2.5 times every 18 months [4]. These highly engineered links, however, are usually underutilized. In fact, the average link utilization in backbone networks of large Internet Service Providers (ISPs) is estimated to be around 30%-40% [5]. Further, the Global e-Sustainability Initiative (GeSI) reported that, by 2020, the CO_2 emissions of wired network devices (*e.g.*, routers, switches, *etc.*), and broadband access equipments are expected to reach 22% and 15%, respectively, of the overall network's CO_2 emissions [6]. The rapid increase in energy price and awareness of green house effect will eventually trigger more restrictive government policies on the energy footprint of the Information and Communication Technology (ICT) sector, which in turn will stimulate demands for effective energy efficient network solutions [7].

Although green networking research is still in its infancy, a number of interesting works have already been carried out; see Section 2. In this paper, we use Traffic Engineering (TE) to reroute traffic across fewest possible number of links and routers. As the energy consumption of backbone routers and their line cards is essentially independent of link load [8], it is natural to set all under-utilized routers and line cards during off-peak periods into sleep mode. In this respect, Intel Corporation has introduced the 0BASE-X concept [9], whereby line-cards are able to quickly switch from active mode, in which data can be transmitted rapidly, to idle mode to save energy, and vice versa. The concept is effective in reducing energy

consumption of links with low utilization. Note that a router typically contains more than one line-card slot, thus, reactivating a router from its sleep state requires significantly longer time as compared to powering on a single line card for a link [10].

This paper considers line cards that have the said active/idle toggling capability, and are connected by multiple physical cables. These cables form one logical bundled link [11] as standardized by the IEEE 802.1 AX [12]; that is, the Medium Access Control (MAC) layer treats each bundled link as a single link. An advantage of using bundled links is that they afford network operators an easy way to upgrade network capacity. However, during off-peak periods, where the full network capacity is not required, there is a clear incentive, in terms of energy cost reduction, to power off cables.

Our main contributions are as follows. We propose an efficient approach - Shortest Single Path First (SSPF), described in Section 4, to power off redundant cables as long as the remaining cables provide sufficient capacity to satisfy traffic demands. We build on the work in [13]. However, it is important to note that our work is different to [13] in two significant aspects. First, unlike the method in [13] that allows multiple paths, our approach routes each traffic demand using only a single path; see Section 2 for its advantages. Our extensive simulations in Section 5 show that this restriction does not reduce the effectiveness of our approach while significantly reducing time complexity as compared to the approach in [13]. Our heuristic approach only takes 0.385 seconds, as compared to 79.6 seconds using FGH [13] to find redundant cables to switch-off in the Abilene topology, which translates to a saving of 50%, versus 46.3% produced by FGH, in energy consumption. Second, the method in [13], while reducing energy, does not set an upper bound on link utilization. In contrast, we include the maximum link utilization $0 \leq U_T \leq 1.0$ as a constraint in our model; see Section 3.2.

In this paper, we have proposed three versions of SSPF: SSPF-1, SSPF-2 and SSPF-R. SSPF-1 and SSPF-2 are exactly the same except for heuristic functions that are used to determine candidate cables to be powered off; Section 4 and 5 compare the performance of these two versions. Given that both SSPF-1 and SSPF-2 may result in a *local minimum*, we have

proposed SSPF-R to overcome local minima and produce better results than SSPF-1 and SSPF-2.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 formulates the problem at hand. Section 4 describes three efficient heuristics to solve said problem. Section 5 presents our evaluation methodology and results. Section 6 concludes the paper.

2. RELATED WORK

Green networking research emanated from the seminal work of Gupta *et al*. [14]. The authors examined the energy consumption of networking devices and discussed their impact on network protocols if they are put to sleep. They showed that packets routed through networks with coordinated sleeping, called the network-wide approach, require protocol changes, whilst those with uncoordinated sleeping, called the link layer approach, only require local information. In a subsequent work, Gupta *et al*. [15] explore this idea in a wired LAN setting. However, as argued in [16], this approach is not applicable to backbone networks that have short packet interval times. Nevertheless, their works [14, 15] have inspired recent research on conserving energy in networks, including ours.

There has been a handful of works on energy-aware Traffic Engineering (TE), some of which uses distributed optimization [17, 18, 19, 20] while others utilize centralized optimization [13, 16, 21, 22, 23, 24]. Vasic *et al*. [17] present EATe, a technique that takes energy consumption into account while achieving the same traffic rates between source and destination nodes as energy oblivious approaches. However, they assume fixed end-to-end paths, which make their approach non-flexible and hard to operate. The authors in [18] propose an energy-aware source routing protocol for a cognitive packet network. However, their method is usable only for smart packet networks [18]. The authors of [19] propose a distributed method that selectively switches off links in an IP-based network to save power. Another distributed strategy, proposed in [20], generates an energy-aware network topology and weight metrics

for each time period. In their design [20], the network control and management system is responsible for populating a historical demand matrix for each time period.

In [23], the authors propose a centralized algorithm that exploits the algebraic connectivity of a network to find the set of links that can be put into standby state, thus generating an energy-aware network topology. Such topology-oriented solution is a planned operation, performed statically by a network administrator for each time period. Thus, when the algorithm incorrectly removes a sub-optimal link, it will never backtrack to correct its mistake for the period. Other centralized optimization solutions on energy-aware TE, like ours, use shortest paths routing [13, 16, 21, 22, 24]. However, these studies do not consider links with bundled cables except [13]. In [21], the authors formulate the problem of power consumption as multi-commodity minimum cost flow problems, and provide methods to switch off routers according to policies such as random, least link, least-flow and most-power. Their algorithms in [21], running on a centralized controller, select the minimum set of devices that must be switched on to meet current traffic demands. Further, they consider a scenario that reconfigures the network periodically, *e.g.*, every 30 minutes, to match the daily traffic variation in current backbone networks. Note that fewer network reconfigurations reduce potential power saving but mitigate latencies incurred when changing power state. Similar to the model in [21], we assume that a network operator changes network configuration infrequently, *i.e.*, only during the off-peak periods, to reduce the risk of network oscillations. The authors of [21] consider different node types, *i.e.*, access and backbone nodes. We, on the other hand, consider only backbone networks comprising of nodes that cannot be switched off. As described later, our work is closely related to [13]. The approach in [22] uses shortest paths routing protocol to find network elements (routers and weighted links) that can be switched off to minimize the total energy consumption while guaranteeing a given maximum link utilization (MLU) constraint. For their power saving problem [24], the authors use an integrated IP layer strategy that is compatible with Open Shortest Path First (OSPF). Their approach aims to generate a subset of IP links to be powered off, when network traffic decreases, subject to satisfying a given link load constraint.

The authors of [16] propose an intra-domain TE mechanism, called GreenTE, which maximizes the number of links that can be put into sleep to minimize power consumption under two performance constraints: maximum link utilization and packet delay. They modelled the problem as a mixed integer program and proposed a heuristic algorithm. For each demand d, their algorithm requires all (s_d, t_d) k-shortest paths as the input, and uses the AMPL/CPLEX solver [25] to generate routes for all traffic demands that use the minimum number of switch-on links/nodes, and hence minimizes power usage, subject to the given constraints. Note that the computational time of the algorithm increases significantly with increasing values of k. However, unlike our work and that in [13], GreenTE considers each link with only a single cable. Further, similar to [13], their model [16] may route a demand through multiple paths.

Fisher *et al*. [13] consider networks in which each pair of core routers is connected by multiple physical cables that form one logical bundled link [11]. They formulated an Integer Linear Programming (ILP) that maximizes the total number of powered-off cables, with the constraint that all traffic demands can be routed through remaining cables. The authors showed that the problem is NP-complete and proposed three heuristics: FGH, EGH, and BGH. Each algorithm iteratively selects one candidate cable to be deleted and use a Linear Program (LP) [13] to check if deleting the cable yields a feasible solution, *i.e.*, the traffic demands can be routed using remaining cables. They used AMPL/CPLEX [25] to solve the LP, which may route one traffic demand onto multiple, not necessarily disjoint, paths. Splitting traffic flow onto multiple paths increases packet delays as well as routing complexity; moreover, throughput of the Transmission Control Protocol (TCP) can be affected significantly due to out of order packets [26]. In contrast, our work restricts each demand to only a single path, which is simpler and more practical. For example, the Multiprotocol Label Switching (MPLS) is generally used with a working configuration that avoids splitting demands, and in most router implementations, packets that belong to a particular TCP session (*i.e.*, going to a specific destination in terms of IP address of the end computer) are routed on a specific shortest-path (even if multiple shortest paths are available)

[27]. Further, Proposition 4.1 in [27] shows that when the number of demands is much larger than the number of edges, which occurs in most practical cases, most demands can be routed through single paths. Apart from that, reference [13] does not consider MLU, and therefore the utilization of remaining links may reach 100%. In contrast, our model includes the threshold of MLU, $0 \leq U_T \leq 1.0$, as a constraint, which can be flexibly set according to best current practices, *e.g.*, 30% to 40%. Note that FGH, the fastest among the three algorithms in [13], may be stuck in a local minimum if it removes an incorrect or sub-optimal cable, and will never backtrack to correct its mistake. In contrast, our SSPF-R algorithm, described in Section 4.3, is able to avoid local minima. Further, FGH requires 50 ± 20 minutes and 14 ± 4 minutes to produce results for the Waxman and Hierarchical topologies [13] respectively. In contrast, our proposed approach, described in Section 4, is able to produce better energy savings in less than five seconds. Given that about 30% of traffic demands in the Abilene network change every five minutes [16], we need a more efficient, *i.e.*, faster algorithm, to adapt quickly to changing traffic conditions whilst achieving energy savings.

Our problem, described in Section 3, is a special case of the multi-commodity capacitated network design problem (MCND), a known NP-hard problem [28]. MCND's reduction, from the Satisfiability problem, uses as many commodities as there are clauses, while the authors, in [29], present a reduction of the Satisfiability problem to the two-commodity integral flow in directed graphs (D2CIF). The problem aims to minimize network costs, *e.g.*, energy, while satisfying traffic demand requirements and link capacity constraints. Several heuristics [30, 31, 32] and branch-and-cut methods [33, 34, 35] have been proposed to solve this problem. Reference [36] studies a 0-1 reformulation of MCND, and shows that extended linking inequalities, derived from variable disaggregation techniques, are equivalent to residual capacity inequalities. The authors of [36] provide a heuristic method that produces a lower bound for MCND with a value that is equivalent to one computed by a LP. Note that their method can be used to generate the initial routes for energy aware routing as an alternative to the LP approach used in [13] and shortest paths, as used in our algorithm, *i.e.*, Step 1 in Figure 1, described in Section 4.

3. NETWORK MODEL AND PROBLEM STATEMENT

3.1 Network Model

Consider a computer network that is represented by a weighted directed graph $G(V, E)$ where V is the set of n nodes, and E is the set of m links. Each node represents a router and each link (i, j) between nodes i and j represents a bundled link as a communication channel with a limited capacity/bandwidth $c_{ij}>0$. Each link (i, j) consists of $w_{ij}\geq 1$ cables. We call w_{ij} the bundle size of link (i, j); we generalize the model in [13] that assumes equal bundle size. Each cable can be switched-off independently. Let $n_{ij}\leq w_{ij}$ be an integer that represents the total number of powered-on cables in (i, j). Let $E_r\subseteq E$ be a set of links in which each $(i, j)\in E_r$ has $n_{ij}>0$. Let D be a set of all demands in $G(V, E)$, and (s_d, t_d, b_d) denote a traffic demand $d=1, 2, \ldots, |D|$ between source node $s_d\in V$ and terminal node $t_d\in V$, where b_d is the amount of traffic exchanged between these nodes. Let P_d be a (s_d, t_d) path that can be used to route the flow in demand d. Let x_{ij}^d be a binary variable that is set to 1 (0) when the traffic d is routed (not routed) through link (i, j), and $f_{ij}=\sum_{d\in D} b_d x_{ij}^d$ be the total flow on (i, j). Note that $n_{ij}=\lceil f_{ij}/(c_{ij}/w_{ij})\rceil$. Lastly, let $0\leq U_T\leq 1.0$ be the threshold of MLU, and r_{ij} be the remaining/spare capacity on link (i, j), computed as $r_{ij}=(n_{ij}/w_{ij})U_T c_{ij}-f_{ij}$.

3.2 Problem Statement

Given $G(V, E)$ and a traffic demand set D, where each demand is to be routed along a single path, the optimization problem is to generate (i) the minimum number of powered on or active cables, and (ii) the set of paths that satisfies traffic demands D using only these powered on cables, subject to a required maximum link utilization $0\leq U_T\leq 1.0$. The optimization problem is formalized in the following Integer Linear Programming (ILP) formulation:

Minimize $\sum_{(i,j)\in E} n_{ij}$ (1)

Subject to:

$$\sum_{(i,j)\in E} x_{ij}^{d} - \sum_{(j,i)\in E} x_{ji}^{d} = \begin{cases} 1, & i = s_d \\ -1, & i = t_d \\ 0, & Otherwise \end{cases}, \forall i \in V, d \in D \quad (2)$$

$$f_{ij} = \sum_{d\in D} b_d x_{ij}^{d} \le (n_{ij} / w_{ij}) U_T c_{ij}, \forall (i,j) \in E \quad (3)$$

$$0 \le n_{ij} \le w_{ij}, \forall (i,j) \in E \quad (4)$$

The objective in the ILP is to minimize the number of powered-on cables, as per Eq. (1). Constraint (2) ensures flow conservation, and requires each demand to be routed along a single path. Eq. (3) constraints the total flow on each link to be less than the capacity provided by active cables for a given link and a given threshold U_T, and Eq. (4) bounds the number of active cables to be less than the bundle size of each link.

As mentioned in [13], the presented formulation is equivalent to the simple two-commodity integral flow in directed graphs (simple D2CIF) [29] problem, which is NP complete. Therefore, in the following sections, we propose a number of heuristics to solve the ILP.

4. EFFICIENT SINGLE SHORTEST PATH FIRST HEURISTICS

This section describes our heuristic approach: Single Shortest Path First (SSPF). Subsection 4.1 describes two versions of SSPF: SSPF-1 and SSPF-2; subsection 4.3 describes its third version, SSPF-R. This section also provides an example to illustrate SSPF and presents the running time complexity analysis of all three versions.

4.1 SSPF

Our greedy heuristic approach, SSPF in Figure 1, produces a set of paths P that can be used to route all demands in D through all powered on cables in link set E_r, *i.e.*, P=$\{P_d \mid (s_d, t_d)$ path in $G(V, E_r)\}$. SSPF uses E_r and n_{ij} of each $(i, j)\in E_r$ to compute its energy saving. In addition, SSPF creates a First-In-First-Out (FIFO) history log, Q, that stores the sequence of removed cables. Specifically, Q is a sequence of a pair $((i, j), nc)$ that denotes the number of switched off cables, nc, in (i, j). As described in Section 4.3, our SSPF-R algorithm needs the information in the set Q to avoid local minima. SSPF initializes $Q=\Phi$, $E_r=E$, and $fix((i, j))=$

false firstly for each $(i, j) \in E$. Note that $fix((i, j))$=false means that it is still possible to turnoff one or more cables in (i, j) while satisfying all demands in D. Then, it executes the following three main steps.

In Step 1, SSPF uses each shortest path P_d to route the traffic flow of each demand d; P_d can be computed using *Dijkstra*'s algorithm [26] and we assume unitary link delay, *i.e.*, each path length is measured in hop count. We assume the network has sufficient link capacity to route all demands in D through their shortest paths. As an example, consider the network in Figure 4 with a set D containing eight traffic demands 1 to 8, (0, 2, 4.2), (0, 5, 1.05), (0, 6, 0.95), (0, 7, 2.25), (0, 10, 8.5), (4, 5, 3.35), (4, 6, 4.35), (10, 5, 1.55) respectively, with w_{ij}=2 and c_{ij}=10. Step 1 will generate eight (s_d, t_d) paths for all demands in D, *i.e.*, P={P_1=((0, 2)), P_2=((0, 2), (2, 5)), P_3=((0, 3), (3, 6)), P_4=((0, 1), (1, 4), (4, 7)), P_5=((0, 8), (8, 9), (9, 10)), P_6=((4, 5)), P_7=((4, 6)), P_8=((10, 5))}. Figure 4 shows the total flow f_{ij} for each link (i, j); see the number without bracket for each link.

In Step 2, SSPF first calculates the total number of cables for each link needed to route all demands in Step 1. Figure 4 shows the total number of needed cables, n_{ij}, for each link (i, j); see each integer in bracket. As an example, for link (4, 6), f_{46}=4.35, and n_{46} is calculated as $\lceil 4.35/(10/2) \rceil$=1; thus one cable in the link is unused. In essence, Step 2 aims to switch off the maximal number of unused cables from each link, whilst ensuring the remaining cables are capable of meeting all traffic demands. If n_{ij}=0, *i.e.*, link (i, j) is never used, the link is removed from E_r. As an example in Figure 4, the step removes link (9, 6) from E_r. This step also stores the number of switched off cables nc>0 for each link (i, j), *i.e.*, each pair $((i, j), nc)$, into Q; for the example in Figure 4, Q=(((9, 6), 2), ((0, 3), 1), ((3, 6), 1), ((2, 5), 1), ((0, 1), 1), ((1, 4), 1), ((4, 7), 1), ((4, 5), 1), ((4, 6), 1)).

In Step 3, SSPF iteratively selects a candidate link (i, j) from E_r and aims to switch off one of its cables. For this step, SSPF considers two different functions to determine the candidate link. SSPF version 1, called SSPF-1, uses the *argmax* function from [13], while its version 2, called SSPF-2, uses our heuristic function, *H-Select-e*(); both versions are exactly the same except for the two functions. The *argmax* function selects a non-fixed link (i, j), *i.e.*, $fix((i,$

j))=false, that has the largest spare capacity r_{ij} while *H-Select-e*() selects a non-fixed (i, j) with the smallest average flow per demand, *i.e.*, (i, j) with the smallest f_{ij}/θ, where $\theta \leq |D|$ is the total number of traffic demands that use link (i, j). The latter function assumes that demand with less flow is easier to re-route onto an alternative path. For Figure 4, *argmax* will select link (0, 2) because it has the largest r_{02}=(2/2)*1.0*10-5.25=4.75, and *H-select-e*() function will select link (0, 3) because it has the smallest f_{ij}/θ=0.95/1. Note that both functions may select a link that might lead to a *local minimum*. Therefore, we propose SSPF-R in Section 4.3 to heuristically restore all cables in a link and select a candidate cable in another link to avoid local minima.

For each selected link (y, z), the algorithm uses our Greedy Heuristic function, *GH-Flow*() in Figure 2, to check if deleting a cable is feasible, *i.e.*, the remaining n_{yz}-1 cables in (y, z) and all cables in the other links in E_r can still meet the flow of all demands. The function re-routes all paths that use (y, z) to all possible paths. One cable in (y, z) will be switched off if re-routing is feasible, and a record, ((y, z), 1), is created and stored in Q, and each $fix((i, j))$ is reset to false for $(i, j) \in E_r$. Any unused link (i, j), *i.e.*, n_{ij}=0, is removed from E_r. Further, this step ensures that the flow of each (s_d, t_d) demand d is routed only through a single path P_d in G. Step 3 is repeated until it is not possible to turn off any remaining cable, *i.e.*, $fix((i, j))$=true for all (i, j), and thus SSPF terminates after turning off all cables in Q from G. The details of *GH-Flow*() is described as follows.

Step 1 of *GH-Flow*() finds each P_d, for $d \in D$, that contains the candidate link (i, j), and adds the capacity of each link in P_d with the previously allocated b_d for demand d. As an example, when (2, 5) is selected, P_2=((0, 2), (2, 5)) is affected and thus the value of r_{02} and r_{25} is increased by b_2=1.05. If the path is disconnected when the cable is turned off, *i.e.*, n_{ij}-1=0, the step generates a new P_d. Following the previous example, n_{25}-1=0, *i.e.*, the link cannot carry any traffic when its only cable is switched off. As the result, P_2=((0, 2), (2, 5)) is disconnected, and thus the step generates a new shortest path P_2=((0, 1), (1, 4), (4, 5)) for the affected demand 2. Step 1 stores either the original or new P_d in a temporary path set TP; let

us call each path in TP for demand d as TP_d. Since removing the cable in (2, 5) only affects demand 2, TP=$\{TP_2\}$.

In Step 2, the function routes the flow of each affected demands in D through its corresponding paths in TP. Notice that the route of the flow from each unaffected demand remains unchanged. The function uses path $TP_d \in TP$ if it can be used to route demand d, and subtract the capacity of each link in the path by its flow, b_d. Following the above example, traffic demand 2 can be rerouted through a new path TP_2=((0, 1), (1, 4), (4, 5)) and thus the step subtracts b_d=1.05 from r_{01}, r_{14}, and r_{45}. However, if any links in the shortest path cannot support the demand, the function generates $k \geq 1$ shortest paths for demand d; this can be carried out using *Yen*'s algorithm [37]. Let SP_d be the set of k shortest paths for demand d. *GH-Flow*() aims to route the flow using the shortest possible path among the k paths. When there is more than one path with the same length, the function selects one randomly. If a flow in demand d can be routed using any of the k-shortest paths, we subtract the capacity of each link in the path by b_d. However, if none of the path has sufficient capacity to route the flow, the function knows that the deleted cable needs be turned on to meet all demands in D. As an example, assume *argmax* selects (0, 3), which disconnects P_3=((0, 3), (3, 6)), and SP_3 for demand 3 contains P_3=((0, 1), (1, 4), (4, 6)). Notice that demand 3, with b_3=0.95, cannot be rerouted through P_3 since r_{46}=5-4.35=0.65<0.95, and thus the step cannot switch off the cable in (0, 3). Function *GH-Flow*() returns false when at least one affected demand cannot be rerouted. Notice that the function does not roll-back the routes of any demands that have been successfully rerouted through their corresponding paths in TP or in SP to their original routes. As an alternative, the function may continue routing the flow of each remaining demand. Note that the running time of this alternative is longer, and therefore is not suggested. As shown in Figure 5, SSPF-1 generates a sequence of switched off cables Q=(((9, 6), 2), ((0, 3), 1), ((3, 6), 1), ((2, 5), 1), ((0, 1), 1), ((1, 4), 1), ((4, 7), 1), ((4, 5), 1), ((4, 6), 1), ((2, 5), 1), ((0, 2), 1)), and a path set P=$\{P_1$=((0, 2)), P_2=((0, 1), (1, 4), (4, 5)), P_3=((0, 3), (3, 6)), P_4=((0, 1), (1, 4), (4, 7)), P_5=((0, 8), (8, 9), (9, 10)), P_6=((4, 5)), P_7=((4, 6)), P_8=((10, 5))$\}$ to route the eight demands, where switched-off links are denoted by dashed lines. For the example, SSPF-

1 is able to switch off 12 cables of 28 total cables in the network, and thus saves 42.9% of energy usage.

4.2 Time Complexity

In Step 1, SSPF uses *Dijkstra*'s algorithm to generate all-pair shortest paths in $O(n^3)$, and route the flows in $O(|D|*m=mn^2)$. Note that n and m are the total number of nodes and links in G respectively, and $|D| \leq n(n\text{-}1)$ is the total number of traffic demands; thus Step 1 has a time complexity no more than $O(mn^2)$ since in general $m \geq n$. Step 2 requires searching all links in G and therefore has time complexity of $O(m)$. Step 3 for SSPF-1 uses the *argmax* function [13] which takes $O(m)$ to find each candidate link. On the other hand, Step 3 for SSPF-2 uses *H-Select-e*() that has $O(|D|)=O(n^2)$ time complexity. The *GH-Flow*() function has the worst case time complexity of $O(n^3(m+nlogn))$; the detailed time complexity analysis of the function is provided below. Since Step 3 is repeated m times, it has complexity of $O(m*(m + n^3(m+nlogn)))=O(mn^3(m+nlogn))$ for SSPF-1, and $O(m*(n^2 + n^3(m+nlogn)))=O(mn^3(m+nlogn))$ for SSPF-2; both versions require the same time complexity. Thus, SSPF has worst case time complexity of $O(n^3+m+mn^3(m+nlogn))=O(mn^3(m+nlogn))$.

The time complexity of function *GH-Flow*() is calculated as follows. The worst case of Step 1 requires running *Dijkstra*'s algorithm for all-pair (s_d, t_d) shortest paths, and thus takes $O(n^3)$ time. However, our simulation in Section 4 shows that on average, each cable deletion affects only 2% of demands. In the worst case, Step 2 is repeated $O(|D|=n^2)$ time. However, as in Step 1, the loop in Step 2 is repeated only for 2% of demands, far less than the worst case. If path TP_d has sufficient capacity to route the traffic demand then it only needs $O(m)$ time to update flow of all links on the path, since it has at most m links. Otherwise, we use *Yen*'s algorithm to generate k-shortest paths for demand d, which has time complexity $O(kn(m+nlogn))$. The worst case scenario is when the feasible path is the last of the ordered k-shortest paths; this case requires $O(km)$. Therefore this sub-step requires $O(kn(m+nlogn) + km)=O(kn(m+nlogn))$, and the worst case time complexity of Step 2 is $O(n^2*kn(m+nlogn))= O(kn^3(m+nlogn))$. Note that we can consider k a constant since our simulation in Section 5

uses $k \leq 10$. Thus, *GH-Flow*() has the worst case time complexity of $O(n^3 + n^3(m+nlogn)) = O(n^3(m+nlogn))$.

The time complexity of SSPF cannot be directly compared with FGH since the latter uses LP solver such as AMPL/CPLEX [25]. As reported in [13], FGH and its improved version – EGH and BGH need to call a LP solver up to $O(m^2)$ and $O(m^3)$ times respectively. In contrast, SSPF only uses *GH-Flow*(), which utilizes *Dijkstra*'s algorithm and *Yen*'s algorithm to generate (s_d, t_d) shortest paths and k (s_d, t_d)-shortest paths respectively for each traffic demand d. Thus, the time complexity of SSPF depends directly on the traffic matrix size $|D|=O(n^2)$. Our simulations in Section 5 show that SSPF runs significantly faster than FGH for various networks with up to 9900 traffic demands.

4.3 SSPF-R

We propose a heuristic algorithm called SSPF-R to improve the optimality of SSPF. Similar to FGH [13], either SSPF-1 or SSPF-2 may select a candidate link that leads to local minima. The authors of [13] proposed EGH and BGH to heuristically solve the problem. However, they showed that EGH and BGH do not improve FGH significantly whilst incurring a significantly higher time complexity, particularly for solving large networks, *i.e.*, Wax50 and Hier50 [13].

Our efficient SSPF-R, shown in Figure 3, greedily avoids local minima. The algorithm repeatedly assumes that the deleted cables, sequenced in Q, leads to a local minimum, and aims to correct the mistake by sequentially restoring each deletion in Q at a time, from the least recent deletion, while assuming that the remaining deleted cables were correct decisions that will lead to a global minimum. For each of constant number $\Re \leq |Q|$ iterations, Step 1 sets E_r and Q^* as a copy of E and Q respectively. Note that in our simulation, outlined in Section 5, we set $\Re$ to $|Q|/2$ since running SSPF-R is faster while producing the same results as compared to using $\Re=|Q|$.

Step 2 aims to correct each possible non-optimal cable deletion by sequentially restoring one cable in pair $((a, b), nc) \in Q$ at a time while assuming that the remaining deleted cables in Q^* were correct decisions, leading to a global minimum. If the link had no cable, *i.e.*,

disconnected, the step connects it back, *i.e.*, includes it in E_r. The step also initializes the status of all links, *i.e.*, setting each *fix*((*i*, *j*))=false. As an example, recall that SSPF-1 in Section 4.1 generates *Q*=(((9, 6), 2), ((0, 3), 1), ((3, 6), 1), ((2, 5), 1), ((0, 1), 1), ((1, 4), 1), ((4, 7), 1), ((4, 5), 1), ((4, 6), 1), ((2, 5), 1), ((0, 2), 1)). Step 2 sequentially attempts to restore one cable starting from pairs ((9, 6), 2) to ((2, 5), 1).

For each attempt to restore a cable in (*a*, *b*), Step 3 uses either *argmax* or *H-Select-e*() to select one cable from the candidate link (*y*, *z*)≠(*a*, *b*). Note that Step 3 in SSPF-R is similar to Step 3 in SSPF. If *GH-Flow*() function in Step 3 is able to re-route traffic flow from (*y*, *z*), the cable is deleted, and Step 3 is repeated using another candidate link (*y*, *z*)≠(*a*, *b*) until all remaining links in E_r are checked, *i.e.*, *fix*((*i*, *j*))=true for each link (*i*, *j*). Note that the status of link (*y*, *z*) is set to true if *GH-Flow*() fails to route affected traffic demands with one less cable in (*y*, *z*). Restoring a cable in (*a*, *b*) generates a better energy saving if *GH-Flow*() could turn-off more than one cable while restoring one cable in (*a*, *b*). For this case, Step 4 updates *Best_Q* with the better result, which will be returned when SSPF-R terminates.

To illustrate SSPF-R, consider Figure 5 that was generated by SSPF-1. Notice that SSPF-R finds that cable deletions in the sequence from ((9, 6), 2) to ((4, 5), 1) in *Q* are optimal and thus, the *Best_Q* equals *Q*. For ((4, 6), 1), Step 2 sets n_{46}=1+1=2, and *argmax* in Step 3 selects (0, 3), which affect P_3=((0, 3), (3, 6)) that was generated in SSPF-1. Thus, *GH-Flow*() generates a new path P_3=((0, 1), (1, 4), (4, 6)) that has sufficient capacity to route demand 3. In another iteration, *argmax* generates link (3, 6). However, since no route is affected by deleting one cable in (3, 6), *GH-Flow*() also returns true. Thus, SSPF-R is able to switch-off two cables, *i.e.*, one in (0, 3) and another in (3, 6) when one cable in (4, 6) is restored, with a gain of one that further reduces the energy saving result when using SSPF-1; see Figure 5 versus Figure 6.

The time complexity of SSPF-R is calculated as follows. The most time consuming step in SSPF-R is Step 3. As described in Section 4.2, *argmax* requires $O(m)$ while *H-Select-e*() requires $O(n^2)$; here SSPF-R uses either function. Step 3 also takes $O(n^3(m+n\log n))$; see its calculation in Section 4.2. This step is repeated $O(m)$ times for each deleted cable, and

SSPF-R considers $\Re$ consecutive cable. Thus, SSPF-R has computational time complexity of $O(\Re*m*(m+ n^3(m+nlogn)))$ when using *argmax*, and $O(\Re*m*(n^2+ n^3(m+nlogn)))$ when using *H-Select-e*() functions. In either case, its complexity is $O(\Re mn^3(m+nlogn)))$. Since $\Re \leq |Q| \leq m$, its worst case time complexity becomes $O(m^2n^3(m+nlogn)))$.

5 EVALUATION

In this section, we evaluate the performance of SSPF; namely, SSPF-1, SSPF-2, and SSPF-R. We first describe our experimental setup in Section 5.1. Then, in Section 5.2, we evaluate the performance of SSPF, and its versions, using both synthetic and realistic topologies for maximum link utilization constraint U_T=1.0. For this case, their performance is evaluated in terms of energy saving and running time for w_{ij}=1 (Section 5.2.1) and for variable w_{ij} (Section 5.2.2). Further, we analyse the effects of switching off link on link utilization (Section 5.2.3), and on path length (Section 5.2.4). In Section 5.2, we also compare the performances of our algorithms against the state-of-the-art technique – FGH [13]. To further evaluate the performances of our algorithms, in Section 5.3, we evaluate them on networks with variable U_T and bundle sizes. Finally, in Section 5.4, we use our techniques on the Abilene network with its 288 traffic demands over 24 hours period, each of which corresponds the traffic demand recorded at every five minutes; these datasets are obtained from [38].

5.1 Methodology

Table 1 summarizes the different network topologies used in our simulations, with nodes ranging from 12 to 100 and links from 28 to 434. We have included the three topologies used in [13]; *i.e.*, Abilene and two synthetic topologies - a two level hierarchical graph (Hier50), and the Waxman graph (Wax50). In the Waxman graph, the probability that two nodes are connected by a link decays exponentially by the distance between them. We also use the Abilene topology from [38]. Note that this Abilene topology (herewith called Real Abilene) has different link connections and capacities than the Abilene in [13]. To further evaluate the performance of SSPF and FGH, we used GT-ITM [39] to generate three random graphs Geo10, Geo30, and Geo50 that represent small, medium and large topologies respectively;

Geo50 contains 50 nodes and 434 links. Further, we also used GT-ITM [39] to generate a large hierarchical graph (Hier100) that has 100 nodes and 284 links.

Each link in the Abilene topology has capacity c_{ij}=10000 and in Wax50 c_{ij}=1000, while the capacity of links in Hier50 is either 1000 or 200; we assume the same capacity unit (*e.g.*, megabytes per second) for both link capacity and traffic demand. Each link in Real Abilene has either c_{ij}=9920 or c_{ij}=2480. For the three random topologies, each link has capacity c_{ij}=1000 and Hier100 has capacity c_{ij}=10000.

As shown in Table 1, the traffic demands for each topology range from 90 to 9900. For the three topologies, Abilene, Hier50 and Wax50, we used the traffic demands and flows provided by the authors of [13]. For each random graph and Hier100, we consider traffic demands between each (s_d, t_d) pair in the network; *i.e.*, Hier100 with 100 nodes has 100*(100-1)=9900 traffic demands. Each traffic flow is generated using the classical entropy model for urban traffic, as described in [40]. The model computes each traffic flow b_d=10*rn_1*rn_2, where $rn_1$and rn_2 are two random numbers between 0 and 1; thus, $0 \leq b_d \leq 10$. Finally, for Real Abilene, we used the traffic matrices from [38], measured every five minutes over a 24 hours period. We used Real Abilene and their 288 different traffic demands in Section 5.4.

We have implemented SSPF in *Java 6*. We computed the energy saving for a given topology by taking the percentage of the total number of off cables over the total number of cables. The power consumption of line-cards used in our simulations is specified in Table 2. For FGH, we used its implementation provided by the authors of [13]. The authors of [13] ran FGH on a Window machine and used the AMPL/CPLEX to solve the LP in [13]. In our simulation, we replaced AMPL/CPLEX with a Linux-based GLPK [41]. Note that FGH's running times reported in [13] are significantly slower compared to those generated in our simulations. For example, the authors of [13] reported that FGH for Wax50 required up to 50 ±20 minutes while in our simulation, the algorithm took only 5±2 minutes. However, the energy savings reported in [13] for FGH on the three topologies (*i.e.*, Abilene, Wax50, and Hier50) are equivalent to our results; we used the results obtained in our simulations for the

FGH's running time and energy saving. We compute the energy saving as the ratio between total powered-off cables and all cables in the network.

We ran all the algorithms on a Linux machine with *Fedora 10* (2.6.x kernel), 1024 MB memory and 28GB hard disk. For SSPF-1, SSPF-2 and SSPF-R, we set k=100, and SSPF-R uses the *argmax* function. We ran each algorithm five times, and calculated its average CPU time.

5.2 Performance Evaluation for U_T=1.0

5.2.1. Energy Savings and Running Times for w_{ij}=1

Table 2 shows the energy savings and running times for all versions of SSPF when each link has equal bundle size w_{ij}=1 and required link utilization U_T=1.0. We see that all algorithms are able to save energy ranging from 37.2% to 86.6%. SSPF-1 and SSPF-2 produce almost equivalent energy savings for all tested networks. However, SSPF-1 runs faster than SSPF-2 since the latter uses *H-Select-e*(), which has a time complexity of $O(|D|=n^2)$, in contrast to the *argmax* function in SSPF-1 which takes $O(m)$. As SSPF-2 may also produce better result for certain type of networks, *e.g.*, Wax50, we suggest running both alternative algorithms and use their best results. However, when faster running time is important, SSPF-1 is the better alternative. Table 2 shows that SSPF-R always produces better energy savings than SSPF-1 and SSPF-2. Since SSPF-R, as described in Section 4.3, runs either SSPF-1 or SSPF-2 repeatedly and selects the best result from all possible outcomes, SSPF-R is guaranteed to always produce at least the energy savings of SSPF-1 or SSPF-2. However, as a trade-off, the running time of SSPF-R is always slower than either SSPF-1 or SSPF-2.

To further evaluate the efficiency and effectiveness of our SSPF, we have compared it with FGH [13]. As described in Section 2, FGH allows a demand to be routed through more than one path while SSPF restricts its routing through only a single path, allowing simpler routing protocol. As shown in Table 2, FGH, in most cases, produces inferior results compared to all versions of SSPF while using significantly more computational time. FGH produces energy savings ranging from 1.9% to 25.3% worse than SSPF-1, and up to 28% worse than SSPF-2. Further, while producing better results, SSPF-1, SSPF-2, and SSPF-R take only, respectively,

0.21% to 13%, 0.49% to 13.09%, and 2.54% to 35.1% running time of FGH. Note that FGH failed to produce a result, denoted as 'N/A', for Hier100 after running for three hours.

To further evaluate the performances of our algorithms, we compare their results with the upper bound (UB) and lower bound (LB) of energy savings. To generate the bounds, like in [13], we first consider the linear-programming version of our ILP problem stated in Eq. (1) to (4), *i.e.*, replace $\sum_{(i,j)\in E} n_{ij}$ in Eq. (1) with $\sum_{(i,j)\in E} f_{ij}$ to minimize the total flow over all links. Then, we use GLPK [31] to obtain the minimum flow of each link while satisfying the constraints in Eq. (2) to (4). To obtain a UB on the energy saving, like in [13], we "round down" the number of cables for each link needed to carry the traffic obtained by the solution. For example, for f_{08}=8.5 in Figure 4, the upper bound of powered-off cables in e_{08} is $\lfloor f_{ij} / (c_{ij} / w_{ij}) \rfloor = \lfloor 8.5 / (10/2) \rfloor$=1. Note that as stated in [13], no flow assignment that satisfies all the demands can use fewer cables than those used in the upper bound. A lower bound is obtained similarly by "rounding up" the number of cables in each link, *i.e.*, for the example, the lower bound of powered-off cables for e_{08} is $\lceil f_{ij} / (c_{ij} / w_{ij}) \rceil = \lceil 8.5 / (10/2) \rceil$=2.

As shown in Figure 7 to 9, the energy savings produced by our SSPF algorithms and FGH are in between their LB and UB. In particular, the energy savings produced by SSPF-1 are between 3.7% at w_{ij}=10 and 20% at w_{ij}=1 off from the UB on Waxman network. Further, SSPF-1 could improve the energy saving generated by LB between 4.6% at w_{ij}=10 and 42% at w_{ij}=2 on Waxman network; for w_{ij}=1, LB could not power-off any cable.

5.2.2. Energy Savings and Running Times for Variable w_{ij}

We further evaluated SSPF-1 using the Abilene, Hier50 and Wax50 topologies when their bundle size, w_{ij}, increases from 1 to 10. For Abilene, as shown in Figure 7, the energy savings produced by the algorithm increases sharply when w_{ij} increases from one to three; *i.e.*, 50% to 82.1%. Similarly, for Hier50 and Wax50, increasing w_{ij} from 1 to 3 also significantly reduces the energy consumption. As a comparison, the figure shows the results for FGH; as shown, SSPF-1 slightly outperforms FGH in term of energy saving for all topologies. The result contradicts the intuition that the less restrictive problem (*i.e.*, to allow demand routed through one or more paths, and hence more path selection flexibility) would lead to a better

energy saving. We believe SSPF produces better results due to its novel approach. Further, consistent with the reported running time in Section 5.2 for w_{ij}=1, Figure 7, 8 and 9 show that SSPF-1 requires significantly less CPU time than FGH for w_{ij}>1; *i.e.*, on average 0.512 seconds versus 62.99 seconds for Abilene, 2.068 versus 315.62 for Hier50, and 2.33 versus 345.59 for Wax50.

5.2.3. Effects on Link Utilization

This section analyses the effects of using fewer cables, thus saving energy, on the average Link Utilization (LU), for U_T=1.0 and w_{ij}=1, calculated using the following Eq. (5):

$$Ave(LU) = (\sum_{(i,j)\in E}((f_{ij}w_{ij})/(n_{ij}c_{ij})))/m' \quad (5)$$

Note that m' is the total number of powered-on links, $((f_{ij}w_{ij})/(n_{ij}c_{ij}))$ is the link utilization of (i, j) that is ignored when n_{ij}=0 since the link is switched off when its cables are all off. Table 3 shows the MLU and average LU of the generated topology (after turning off cables) using SSPF and FGH; both approaches produce equivalent results. As a benchmark, we have compared the results with MLU and average LU, before turning off cables, when each traffic demand is routed through its shortest path (SP). Table 3 shows that the SP routing using all cables in the Abilene, Hier50, and Wax50 results in MLU of 65.5%, 100%, and 92.9%, respectively; the MLU for the other networks is less than 30%. Further, the average LU using SP ranges from 0.9% to 24.1%; low average link utilization is expected during off-peak period, and in general, all algorithms achieve high percentage of energy savings because the network has low link utilization. As shown in the table SSPF and FGH increase the MLU and average LU of the networks as compared to SP. The results are expected because when fewer cables are used to carry the same amount of flow, each cable carries more flow. However, we observe that it is possible to power-off higher percentage of cables in a network that has the higher average link utilization; see Wax50 and Geo10 with average LU of 24.1% and 0.9% (Table 3) but with energy savings of 63.3% and 57.1% (Table 2), respectively.

To further evaluate the effects of shutting down cables on the remaining link's utilization, we show in Figure 11(a), Figure 12(a) and Figure 13(a) the cumulative distribution function (CDF) of the link utilizations in Abilene, Wax50 and Hier100 networks, respectively using SSPF-1, SSPF-2, SSPF-R, FGH and SP routings. The results for other topologies are similar

and thus not shown here. All four energy-saving routing algorithms increase link utilization as compared to using SP, but to no more than 0.85 and 0.6 for Abilene and Hier100 respectively; FGH cannot obtain any results in a reasonable time for Hier100, and thus Figure 13(a) omits FGH. Among the four algorithms, SSPF-2 and SSPF-R perform the best for Abilene and Hier100, respectively; for Wax50, they produce similar results. Notice that, SSPF-2 increases the number of links with utilization above 0.4 from 3% to 14% as a tradeoff for reducing Abilene's energy by about 37%; see Figure 11(a) and Table 2. Similarly, for Hier100, SSPF-R increases the number of links with utilization above 0.1 from 3% to 23% while reducing energy usage using SP by 53%; see Figure 13(a) and Table 2. As expected, when the traffic demands are considerably larger than available link resources, *i.e.*, for Wax50 as shown in Figure 12(a), all energy-saving routing algorithms, while reducing energy usage by close to 60% (see Table 2), result in larger ratio of link utilizations as compared to SP. However, as will be discussed in Section 5.3.2, our SSPF approach allows different MLU settings, *e.g.*, no more than 0.4 as the standard practice in ISP, while maximizing energy savings.

5.2.4. Effects on Path Length

Table 4 shows the effect of turning off cables, using SSPF and FGH, on the average path length $L(P_d)$, for U_T=1.0 and w_{ij}=1, calculated as follows,

$$Ave(L(P_d)) = (\sum_{d \in D} L(P_d))/|D| \quad (6)$$

Note that $L(P_d)$ is the length of the path used to route demand d, *i.e.*, its hop counts. For FGH, as demands may be routed through multiple paths, we used the maximum path length. As shown in Table 4 turning off cables, either using SSPF or FGH, have a significant impact on the average path length; SSPF and FGH produced similar results. The results are expected since turning off cables forces some part of the traffic to be routed through longer (non-shortest) paths. Notice, however, that our simulation considers only hop counts as the path lengths, which do not reflect the actual path delays that include several other factors such as the queuing delays.

Figure 11(b), Figure 12(b) and Figure 13(b) show the CDF of the path lengths in Abilene, Wax50 and Hier100 networks using SSPF-1, SSPF-2, SSPF-R, FGH and SP routings, which evaluate the impact of energy savings in path delay. The results for other topologies are similar and thus not shown here. All energy-saving routings produce similar results except FGH that cannot obtain the results in reasonable time for Hier100, and Figure 13(b) omits FGH. For Abilene, as shown in Figure 11(b), while decreasing energy by 51.2%, the energy saving routings, *e.g.*, SSPF-R, reduces the percentage of routes that have delay of one hop from 37% using SP to 11%, and those with two hops from 74% using SP to 25%. Notice that the longest path using SP is five hops, while that using SSPF-R is 8 hops, an increase of 60%. However, the longest path in SSPF-R is shorter than the network diameter of Abilene, *i.e.*, 9 hops. As shown in Figure 12(b) and Figure 13(b), the effects of shutting down cables on path length on Wax50 and Hier100 are similar to in Abilene in Figure 11(b). Further, similar to Abilene, the longest path of routes using SSPF-R for Wax50 and Hier100 is also shorter than the network diameter.

5.3 Energy Savings for Different U_T

As discussed in the Section 5.2.3, SSPF affects link utilization as fewer links are used to carry traffic, In this section, we investigate the effect of using 10 different MLU bounds, *i.e.*, U_T between 0.1 and 1.0 – with 0.1 increments, on energy savings achievable using our SSPF approach, for bundle size w_{ij} between 1 and 10 while running SSPF-R; we obtained similar results using SSPF-1 and SSPF-2. Table 5 and 6 shows results for the Real Abilene and Hier100, respectively where UT_X denotes MLU bound X; *e.g.*, UT_0.4 means U_T=0.4, and $ND = \sum_{(i,j)\in E}(w_{ij} - n_{ij})$ be the total number of switched-off cables. The results for other topologies have the similar trend when the bundle size increases from 1 to 10 under different U_T, and thus not shown here. For Abilene, SSPF-R fails to save energy with $U_T \leq 0.3$; these results are also omitted. Further, for each algorithm, increasing U_T from 0.5 to 0.6, or from 0.7 to 0.8, or from 0.9 to 1.0 does not affect energy savings, and thus, Table 6 only shows the results for UT_0.4, UT_0.5, UT_0.7 and UT_0.9. For Hier100 network, SSPF-R fails to save energy with $U_T \leq 0.1$; thus we do not show its result in Table 6. Similar to the Abilene network,

other than U_T=0.2 and U_T=0.3, increasing U_T from 0.4 to 0.6, or from 0.7 to 1.0 does not affect energy savings. To this end, Table 6 only shows the results for UT_0.2, UT_0.3, UT_0.4 and UT_0.7.

As shown in Table 5 and 6, energy savings increase, for each U_T, when the bundle size increases from 1 to 10 since there are more idle cables. Notice that there is a large increase in energy saving, *i.e.*, by about 25% when the bundle size in Abilene is increased from 1 to 2. Similarly, a large increase in energy saving also occurs in Hier100 when the bundle size increases from 1 to 4. As shown in Table 5, our SSPF approach is able to reduce energy usage in Abilene between 37.8% and 86.5% when its bundle size is set between w_{ij}=1 and w_{ij}=10, respectively, even when each link utilization is set to no more than 40%, which is within the standard practice set by ISP [13]. Similarly, for MLU≤40%, SSPF-R is able to reduce the energy expenditure of Hier100 by 46.5% to 90.2%, for bundle sizes between 1 and 10, respectively. For both Abilene and Hier100, relaxing the link utilization constraint to higher values allow our SSPF algorithm to find better alternative paths, and thus further reducing energy usage.

Tables 5 and 6 also show that for each bundle size, there is negligible effect from using different U_T constraints on energy saving. As an example, Table 6 shows that, for w_{ij}=1, using significantly more restrictive U_T=0.2, as compared to U_T=0.7, only slightly decreases energy saving to 40.9% from 50.3%. This effect may be due to the low traffic levels. Recall that for Hier100, each traffic flow for demand d is computed as b_d=10*rn_1*rn_2, where rn_1 and rn_2 are two random numbers between 0 and 1; thus, 0≤b_d≤10 is a small value as compared to the capacity of each link, *i.e.*, c_{ij}=10000.

To see the effect of traffic levels, *i.e.*, different flow size in traffic demands, we generated five other traffic levels for Hier100 when w_{ij}=1. Specifically, we multiplied each b_d used to generate Table 6 for w_{ij}=1 with five scaling factors: 0.5, 2, 3, 4, 5. Thus, the smallest scale, *i.e.*, 0.5, sets 0≤b_d≤5, while the largest, *i.e.*, 5, generates 0≤b_d≤50; the former simulates lower traffic level while the latter assumes a more congested network as compared to the traffic level used in Table 6 with 0≤b_d≤10. As expected, Figure 14 shows that, for each constraint U_T,

SSPF-R produces higher energy savings for lighter traffic flows, *e.g.*, 51.2% for $0.5*b_d$ versus 46.5% for b_d with U_T=0.6, and lower energy savings in more congested networks, *e.g.*, only 39.16% for $3*b_d$. Further, consistent with our results in Table 5 and 6, the figure shows that for each traffic level, U_T does not significantly affect the energy savings produced by SSPF-R, *e.g.*, for $0.5*b_d$ the savings increase only from 41.96% for U_T=0.1 to 51.2% for U_T=1.0. We observed that the different traffic levels only affect the feasibility of traffic routings. For example, there is no feasible routing for demands using $0 \leq b_d \leq 50$ ($0 \leq b_d \leq 20$) with $U_T \leq 0.8$ ($U_T \leq 0.3$), *i.e.*, the SP routing, described in section 5.2.3, and SSPF-R fail to produce results due to insufficient link capacities for the given set of traffic demands.

5.4 Potential Energy Savings on Real Abilene

Figure 10 shows the potential energy savings on Real Abilene [38] using SSPF-1, SSPF-2, and SSPF-R. We ran each algorithm for 288 different traffic demands from [38]; each demand represents traffic traces recorded every five minutes over 24 hours on September 5, 2004. For this experiment, SSPF-1, SSPF-2, and SSPF-R each required on average 0.21, 0.24 and 0.547 CPU seconds, respectively, to produce results for each five minutes of traffic demand. As shown in Figure 10, SSPF-R consistently produces the best energy savings compared to SSPF-1 and SSPF-2. It is interesting to observe that SSPF-2 outperforms SSPF-1 from 0:00 through 11:00 hours and from about 22:30 through 24:00; at other times, SSPF-1 in most cases produced better results than SSPF-2. Further, energy savings produced by SSPF-R are always the same for each of the 288 different traffic demands. As another comparison, the figure shows that FGH in all (most) cases perform worse than SSPF-R (either SSPF-1 or SSPF-2).

6 Conclusion and Future Work

In this paper, we have proposed an efficient and effective heuristic approach, SSPF, to minimize network energy usage while satisfying all network traffic demands subject to a given maximum link utilization constraint. Our approach aims to switch off redundant cables in core routers using bundled links. However, unlike [13], our approach routes each traffic

demand onto one single path, which simplifies routing. We have used the Abilene topology - with both real and synthetic traffic matrices and several larger randomly generated topologies – with synthetic traffic matrices to evaluate its performances. The simulation shows that our approach could potentially save up to 56.7% of the energy expenditure incurred by the Real Abilene topology, as per the 24 hours traffic demands, measured every five minutes, obtained from [38]. Further, our heuristic significantly outperforms the approaches in [13], both in terms of their running times and energy savings. Re-routing traffic demands to minimize active cables using both FGH [13] and our approach may focus traffic flows to switched-on cables and/or use longer (s_d, t_d) paths. Consequently, our results increase the path length and/or the upper bound of the network's link utilization. As a future work, we plan to extend our approach to include constraints on the maximum path length. Further, we will also include node energy consumption in our power model, which is suitable for an environment where both node and link can be powered-off to save energy.

Acknowledgement

We thank Will Fisher [13] for providing us the FGH source code and data, and the anonymous reviewers for their valuable comments that have significantly improved our paper.

References

[1] C. Bianco, F. Cucchietti, G. Griffa, "Energy Consumption Trends in the Next Generation Access Network - A Telco Perspective," in INTELEC 2007, Rome, Italy, Sept. 2007, pp.737-742.

[2] S. N. Roy, "Energy Logic: A Road Map to Reducing Energy Consumption in Telecommunications Networks," in INTELEC 2008, San Diego, CA, USA, Sept. 2008.

[3] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures," in IEEE Communications Surveys & Tutorials, vol. 13, 2011, pp. 223-244.

[4] IT Wales, "Green Evangelist to Call for Big Changes in Computer Use to Aid Environment," inITWales Conf., Nov. 2007, http://www.itwales.com/997539.htm.

[5] J. Guichard, F. L. Faucheur, and J.-P. Vasseur, Definitive MPLS Network Designs, in Cisco Press, 2005.

[6] Global e-Sustainibility Initiative (GeSI), "SMART 2020: Enabling the Low Carbon Economy in the Information Age", http://www.theclimategroup.org/assets/ resources/publications/Smart2020Report.pdf.

[7] W. Vereecken, L. Deboosere, D. Colle, B. Vermeulen, M. Pickavet, B. Dhoedt, and P. Demeester, "Energy efficiency in telecommunication networks," in NOC 2008.

[8] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright,“ Power Awareness in Network Design and Routing,” in IEEE INFOCOM, Phoenix, AX, USA, April, 2008.
[9] R. Hays, “Active/idle Toggling with 0base-x for Energy Efficient Ethernet,” in IEEE P802.3az Energy Efficient Task Force, Atlanta, GA, USA, Nov. 2007.
[10] B. Awerbuch, D. Holmer, and H. Rubens, “The Pulse Protocol: Mobile Ad hoc Network Performance Evaluation,” in WNOC 2005, MD, USA, Jan. 2005.
[11] R. Doverspike, K. K. Ramakrishnan, and C. Chase, Structural Overview of ISP Networks, in Guide to Reliable Internet Services and Applications (C. Kalmanek, S. Misra, and R. Yang, eds.), Springer, 2010.
[12] IEEE, IEEE Standard 802.1 AX: Link Aggregation, 2008.
[13] W. Fisher, M. Suchara and J. Rexford. “Greening Backbone Networks: Reducing Energy Consumption by Shutting Off Cables in Bundled Links,” in ACM SIGCOMM Workshop on Green Networking, New Delhi, India, August,2010.
[14] M. Gupta, S. Singh, “Greening of the Internet,” in ACM SIGCOMM, Karlsruhe, Germany, August, 2003.
[15] M. Gupta, S. Singh, “Energy Conservation with Low Power Modes in Ethernet LAN Environments,” in IEEE INFOCOM (mini symposium), Anchorage, Alaska, USA, 2007.
[16] M. G. Zhang, C. Yi, B. Liu and B. Zhang, “GreenTE: Power-Aware Traffic Engineering,”in IEEE International Conference on Network Protocols (ICNP), Kyoto, Japan, October, 2010.
[17] N. Vasic and D. Kostic,“Energy-Aware Traffic Engineering,” in ACM SIGCOMM, New Delhi, India, August, 2010.
[18] E. Gelenbe and T. Mahmoodi. “Energy-aware Routing in the Cognitive Packet Network,” in Proc. of International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, 2011.
[19] A. P. Bianzino, L. Chiaraviglio and M. Mellia, “GRiDA: a Green Distributed Algorithm for Backbone Networks,” in IEEE Online Green Communications Conference, September 2011.
[20] S. S. W. Lee, P. K. Tseng and A. Chen. “Link Weight Assignment and Loop-free Routing Table Update for Link State Routing Protocols in Energy-aware Internet,” in Future Generation Computer Systems, 2011.
[21] L. Chiaraviglio, M. Mellia, and F. Neri, “Minimizing ISP Network Energy Cost: Formulation and Solutions,” in IEEE/ACM Transactions on Networking, 2011.
[22] E. Amaldi, A. Capone, L.G. Gianoli, L. Mascetti, “A MILP-based Heuristic for Energy-Aware Traffic Engineering with Shortest Path Routing,” in International Network Optimization Conference (INOC) 2011, Hamburg, Germany, June 13-16,2011.
[23] F. Cuomo, A.Cianfrani, M. Polverini and D.Mangione, “Network Pruning for Energy Saving in the Internet,”in Computer Networks, 2012.
[24] A. Cianfrani, V. Eramo, M. Listanti, M. Polverini and A. V. Vasilakos, “An OSPF-Integrated Routing Strategy for QoS-Aware Energy Saving in IP Backbone Networks,” in IEEE Transactions on Network and Service Management, Vol. 9, No.3, Sep. 2012.
[25] AMPL/CPLEX, <http://www.ampl.com/DOWNLOADS/index.html>.
[26] U. Ranadive and D. Medhi,“ Some Observations on the Effect of Route Fluctuation and Network LinkFailure on TCP,” in Proc. 10th IEEE International Conference on Computer Communications and Networks (ICCCN’01), pages 460–467, Scottsdale, AZ, October 2001.

[27] M. Pioro and D. Medhi, Routing, Flow, and Capacity Design in Communication and Computer Networks, in ELSEVIER, 2004.
[28] R. M. Karp, "On the Complexity of Combinatorial Problems," in Networks, 5 (1975), pp. 45-68.
[29] S. Even, A. Itai, and A. Shamir, "On the Complexity of Time Table and Multi-commodity Flow Problems," in IEEE Foundation of Computer Science (FOCS), Berkeley, CA, USA, October, 1975.
[30] D. Berger, B. Gendron, J.-Y. Potvin, S. Raghavan, P. Soriano, "TabuSearch for a Network Loading Problem with Multiple Facilities," in Journal of Heuristics 6 (2000),pp. 253–267.
[31] B. Gendron, J.-Y. Potvin, P. Soriano, "Diversification Strategies in Local Search for a Non-bifurcated Network Loading Problem," in European Journal of Operational Research 142 (2002),pp. 231–241.
[32] Y.K. Agarwal, "Design of Capacitated Multi-commodity Networks with Multiple Facilities," in Operations Research 50 (2002), pp. 333-344.
[33] A. Atamturk, "On Capacitated Network Design Cut-set Polyhedral," in Mathematical Programming 92 (2002) 425-437.
[34] F. Barahona, "Network Design Using Cut Inequalities," in SIAM Journal on Optimization 6 (1996), pp. 823-837.
[35] D. Bienstock, S. Chopra, O. Günlük, C. Tsai, "Minimum Cost Capacity Installation for Multicommodity Network Flows," in Mathematical Programming 81 (1998), pp. 177–199.
[36] A. Frangioni, B. Gendron, "0-1 Reformulations of the Multi-commodity Capacitated Network Design Problem," in Discrete Applied Mathematics 157 (6), pp. 1229-1241, 2009.
[37] J. Y. Yen, "Finding the K Shortest Loopless Paths in a Network," in Management Science, 17(11), 1971.
[38] Yin Zhang's Abilene Traffic Matrix, [Online]. Available: <http://www.cs.utexas.edu/~yzhang/research/AbileneTM>.
[39] GT-ITM: Georgia Tech Internetwork Topology Models (software), 1996. Available: <http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html>.
[40] B. Fortz and M. Thorup, "Optimizing OSPF/IS-IS Weights in a Changing World," in IEEE Journal on Selected Areas in Communications, 20(4), 2002,pp. 756-767.
[41] GNU Linear Programming Kit (GLPK), Available: <http://www.gnu.org/s/glpk/>.

Table 1: Network Topologies Used in Experiments

Name	Topology	Nodes	Edges	Demands
Abilene	Backbone	39	28	253
Hier50	Hierarchical	50	148	2450
Wax50	Waxman	50	169	2450
Hier100	Hierarchical	100	286	9900
Geo10	Random	10	28	90
Geo30	Random	30	136	870
Geo50	Random	50	434	2450
Real Abilene	Backbone	12	30	132

Table 2: Energy Saving for Various Networks with w_{ij}=1, U_T=1.0

Topology	Energy Saving (%)				CPU Time (Second)			
	SSPF-1	SSPF-2	SSPF-R	FGH	SSPF-1	SSPF-2	SSPF-R	FGH
Abilene	50	46.3	51.2	46.3	0.385	0.392	2.036	79.6
Hier50	37.8	37.2	37.8	37.2	3.591	4.394	41.3	313.7
Wax50	56.8	60.4	63.3	53.3	2.969	5.723	163.28	359.3
Geo10	53.6	53.6	57.1	46.4	0.143	0.144	0.252	1.1
Geo30	72.1	72.8	73.5	69.9	0.564	1.310	18.225	85.7
Geo50	86.4	85.9	86.6	84.8	2.875	17.872	490.3	1395.9
Hier100	49.7	49.7	50.3	N/A	13.462	55.86	514.24	N/A

Table 3: MLU (%) and Average Link Utilization (%) for Various Algorithms, w_{ij}=1 and U_T=1.0

Network	SSPF-1		SSPF-2		SSPF-R		FGH		SP	
	MLU	Ave	MLU	Ave	MLU	Ave	MLU	Ave	MLU	Ave
Abilene	80.35	16.33	83.25	16.32	98.13	16.77	80.48	15.22	65.5	4.9
Hier50	100	51.26	100	50.56	100	51.51	100	48.36	100	22.9
Wax50	99.95	72.9	100	61.8	100	71.48	100	77.26	92.9	24.1
Geo10	4.49	3.32	5.99	3.32	4.65	3.32	5.87	2.45	2.17	0.9
Geo30	61.17	14.98	68.87	19.63	69.25	21.32	53.34	16.24	14.86	1.9
Geo50	86.04	38.76	99.96	37.34	100	40.62	90.2	31.12	24.2	1.4
Hier100	58.84	14.82	67.15	15.18	59.96	16.83	N/A	N/A	26.25	2.9

Table 4: Average Path Length (hop) for Various Algorithms, w_{ij}=1 and U_T=1.0

Network	SSPF-1	SSPF-2	SSPF-R	FGH	SP
Abilene	4.37	4.32	4.53	4.42	2.05
Hier50	4.83	4.71	4.9	4.92	3.17
Wax50	4.16	4.23	4.3	4.26	2.2
Geo10	1.92	1.92	1.97	1.89	1.18
Geo30	5.5	5.5	5.61	5.47	1.36
Geo50	7.4	7.36	7.4	7.35	1.0
Hier100	13.83	13.72	13.83	N/A	5.81

Table 5: Energy Saving on the Abilene Network Using SSPF-R

Bundle Size w_{ij}	Total Cables	UT_0.4		UT_0.5		UT_0.7		UT_.0.9	
		ND	(%)	*ND*	(%)	*ND*	(%)	*ND*	(%)
1	82	31	37.8	36	43.9	39	47.6	42	51.2
2	164	103	62.8	111	67.7	115	70.1	116	70.7
3	246	164	66.7	189	76.8	192	78	193	78.4
4	328	244	74.3	253	77.1	270	82.3	275	83.2

5	410	323	78.8	342	83.4	347	84.6	350	85.1
6	492	395	80.3	409	83.5	425	86.3	429	87
7	574	474	82.6	480	83.6	503	87.6	508	88.1
8	656	556	84.8	557	84.9	580	88.4	584	88.9
9	738	629	85.2	634	85.9	658	89.1	663	90
10	820	709	86.5	711	86.7	716	87.3	740	90.5

Table 6: Energy Saving of Hier100 Network Using SSPF-R

Bundle Size	Total Cables	UT_0.2		UT_0.3		UT_0.4		UT_0.7	
w_{ij}		*ND*	(%)	*ND*	(%)	*ND*	(%)	*ND*	(%)
1	286	117	40.9	124	43.4	133	46.5	143	50.3
2	572	307	53.7	394	68.9	408	71	420	50.3
3	858	589	68.7	593	69.1	686	79.8	701	73.4
4	1144	871	76.1	876	76.6	880	83.8	981	81.7
5	1430	1156	80.8	1156	80.8	1162	81.6	1260	85.8
6	1716	1440	83.9	1441	84	1446	84.7	1541	88.1
7	2002	1723	86	1722	86	1728	86.7	1827	89.8
8	2288	2002	87.5	2004	87.6	2009	87.7	2096	91.3
9	2574	2287	88.8	2292	89	2293	89	2378	91.6
10	2860	2571	89.9	2574	90	2582	90.2	2658	92.4

Algorithm SSPF

$Q \leftarrow \phi$, $E_r \leftarrow E$, $fix((i,j)) \leftarrow$ false for each $(i,j) \in E_r$; //*initialize the linkstatus*

1) **For** each demand $d \in D$ **do**
   Generate shortest path P_d, and route its b_d through P_d;
   Store P_d into P;
   End-For
2) **For** each $(i,j) \in E_r$ **do**
   $n_{ij} \leftarrow \lceil f_{ij} / (c_{ij} / w_{ij}) \rceil$;
   If $n_{ij} < w_{ij}$ **then** // *there is unused cable*
   If $n_{ij}=0$ **then** //*no cable in the link is used*
   $E_r \leftarrow E_r - \{(i,j)\}$; // *turn-off all cables in* (i,j)
   $Q \leftarrow Q + \{((i,j), w_{ij}-n_{ij})\}$; // *turn off* $nc=(w_{ij}-n_{ij})$ *cables in* (i,j)
   End-If
   End-For
3) **While**$|E_r|>0$ **do**
   Use *argmax*() or *H-Select-e*() to select a link $(y,z) \in E_r$ that has $fix((y,z))$=false;
   If *GH-Flow*$(E_r,(y,z))$=true **then**
   $Q \leftarrow Q + \{(y,z),1\}$ //*Turn off one cable in* (y,z)
   $n_{yz} \leftarrow n_{yz}-1$
   If $n_{yz}=0$ **then**
   $E_r \leftarrow E_r - \{(y,z)\}$;
   Set $fix((i,j)) \leftarrow$ false $\forall (i,j) \in E_r$; // restart Step 3
   End-if
   Else
   $fix((i,j)) \leftarrow$ true;
   If $fix((i,j))$=true, for $\forall (i,j) \in E_r$ **then** // *check the status of each link in* E_r
   break; //*End while loop*
   End-Else
   End-While
   Delete all switched of cables in Q from their corresponding links in G.

Figure 1: Algorithm SSPF

***GH-Flow*(E_r,(i,j))**

Status ← true; // *The status of function*
1) **For** each demand $d \in D$ **do**
   If $P_d \in P$ contains link (i,j) **then**
   $r_{ij} \leftarrow r_{ij}+b_d$, $\forall (i,j) \in P_d$; // *returns the resource*
   If $(n_{ij}-1)=0$ **then** // *the path is disconnected when one cable in* (i,j) *is off*
   Replace P_d with a new (s_d, t_d) shortest path for $d$on $G(V, E_r)$;
   Store P_d into TP; //*a new or original path*P_d *after deleting a cable*
   End-If
   End-For
2) **For** each $TP_d \in TP$ **do**//*we need to consider the threshold of link utilization constraint* U_T
   If TP_d has enough capacity to route b_d for d **then**
   $r_{ij} \leftarrow r_{ij}-b_d$, $\forall (i,j) \in TP_d$; // *update the remaining capacity for each link in* P_d
   Replace $P_d \in P$ with TP_d;
   Else
   Generate the k-shortest (s_d, t_d) paths SP_d for demand d;
   For each P_x in SP_d **do**// *there are at most k different path* P_x
   If P_x has enough capacity to route b_d **then**
   $r_{ij} \leftarrow r_{ij}-b_d$, $\forall (i,j) \in P_x$;
   Replace $P_d \in P$ with P_x;
   Go to 2);
   End-If
   End-For
   Status ← false; // *rerouting is not possible*
   Go to 3);
   End-Else
   End-For
3) **Return** Status;

Figure 2: Function *GH-Flow*()

Algorithm SSPF-R

$Q^* \leftarrow \phi$, *Best_Q* ← Q;
For $x \leftarrow 1$ to $\Re$ **do**
1) $Q^* \leftarrow Q$, $E_r \leftarrow E$;
2) Remove x^{th} pair $((a, b), nc)$ from Q, *i.e.*, $Q^* \leftarrow Q^*-\{(a, b), nc\}$;
   $n_{ab} \leftarrow n_{ab}+1$; //*G is the output from SSPF-1 or SSPF-2 and add one cable to* (a, b)*;*
   If $n_{ab} = 1$**then** //(a, b)*was disconnected*
   $E_r \leftarrow E_r+\{(a, b)\}$; //*add the link*
   Set $fix((i,j)) \leftarrow$ false $\forall (i,j) \in E_r$;
3) **While** $|E_r| > 0$ **do**
   Call *argmax*() or *H-Select-e*(G) to select a link $(y, z) \neq (a, b)$ from E_r;
   If *GH-Flow*(E_r, (y, z))=true **then**
   $Q^* \leftarrow Q^*+\{(y, z), 1\}$;
   $n_{yz} \leftarrow n_{yz}-1$; // *Remove one cable every time*
   If $n_{yz}=0$ **then**
   $E_r \leftarrow E_r-\{(y, z)\}$; //*delete the link*
   Set $fix((i,j)) \leftarrow$ false $\forall (i,j) \in E_r$;
   Else
   $fix((y, z)) \leftarrow$ true;
   If $\forall (i,j) \in E_r$ has $fix((i,j))$=true **then**
   break; //*End while loop*
   End-Else
   End-While
4) **If** Q^* contains more deleted cables than *Best_Q* **then**
   Best_Q←Q^*
End-For

Figure 3: The SSPF-R Algorithm

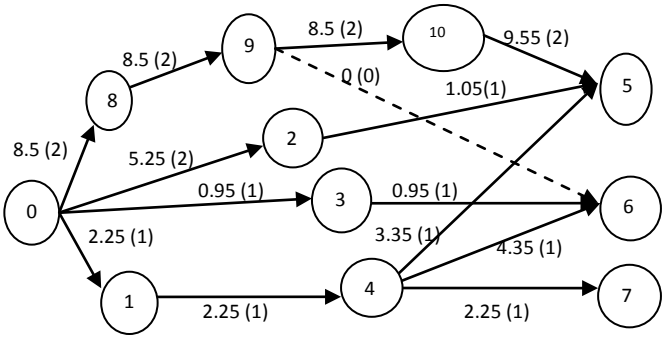


Figure 4: An Example Network

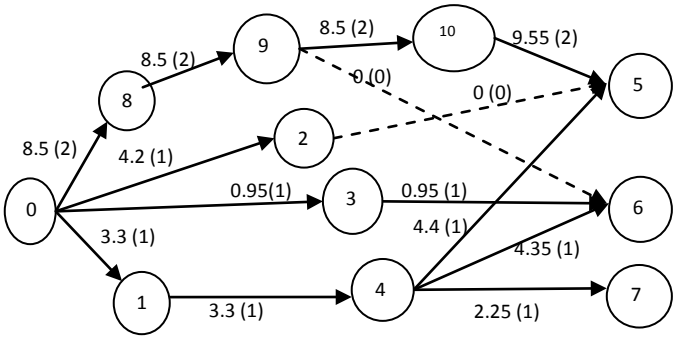


Figure 5: SSPF-1 Solution for Network in Figure 4

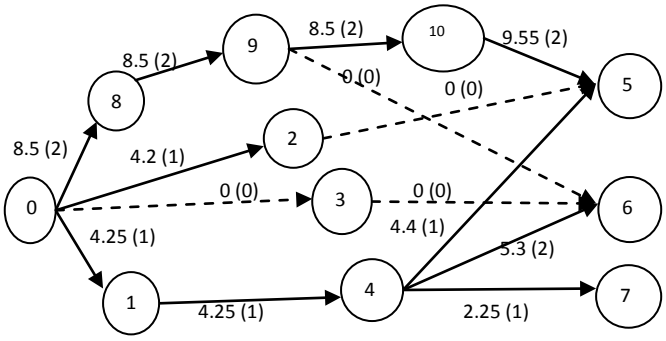


Figure 6: SSPF-R Solution Based on Figure 5

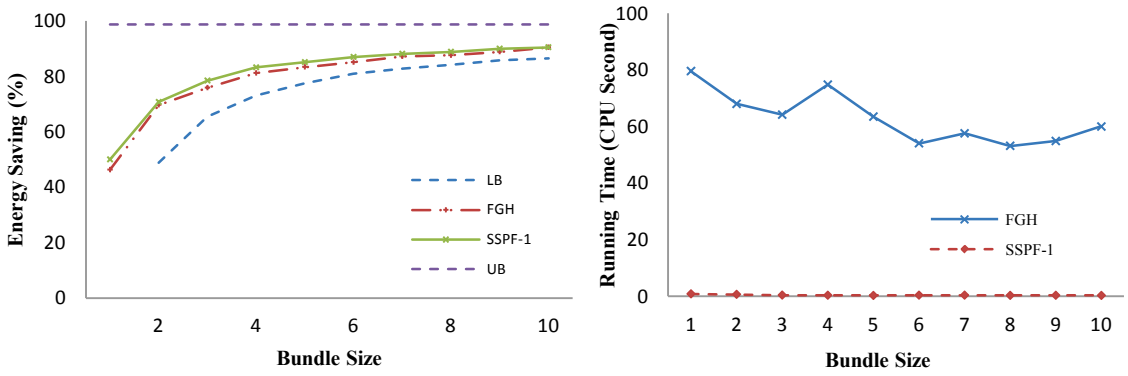


Figure 7: Energy Saving and Running Time on Abilene for Various Bundle Sizes

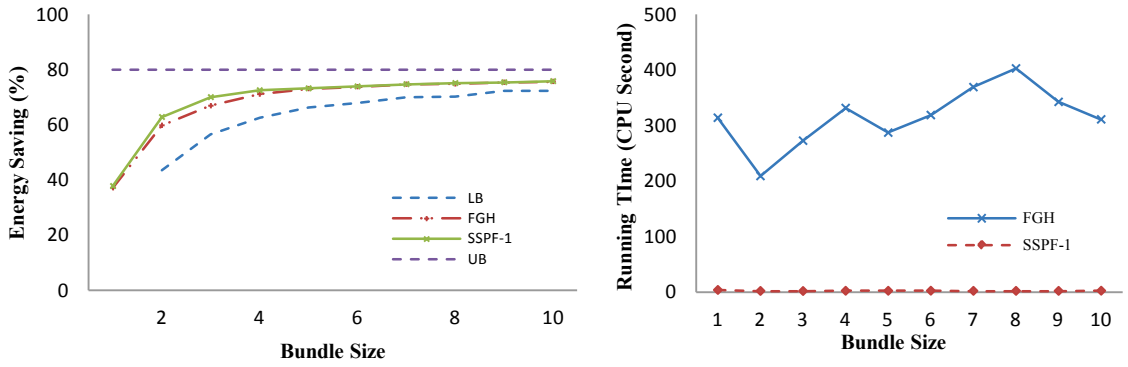


Figure 8: Energy Saving and Running Time on the Hier50 Network for Various Bundle Sizes

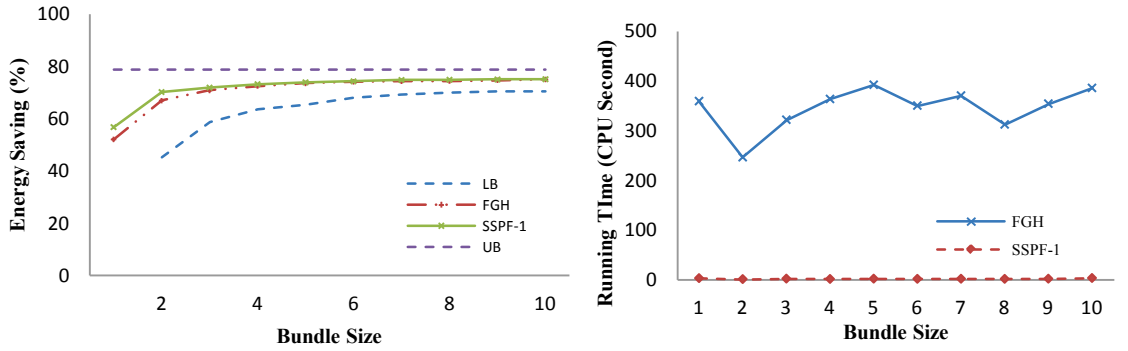


Figure 9: Energy Saving and Running Time on the Wax50 Network for Various Bundle Sizes

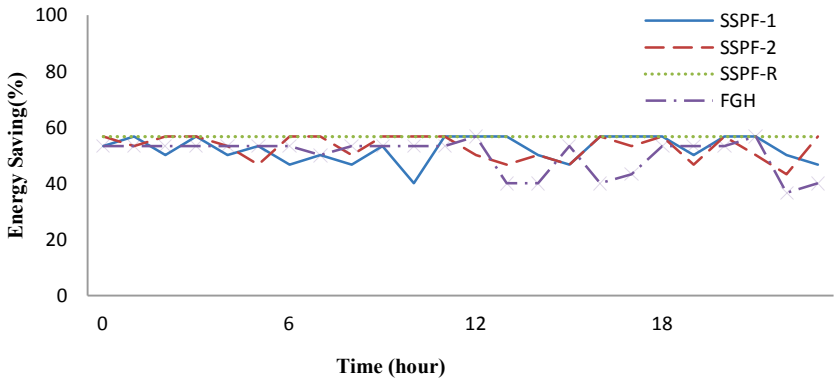
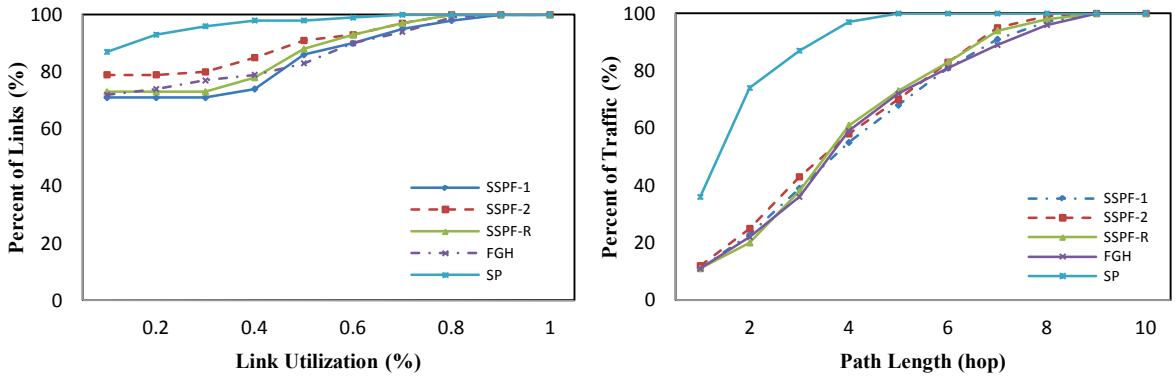
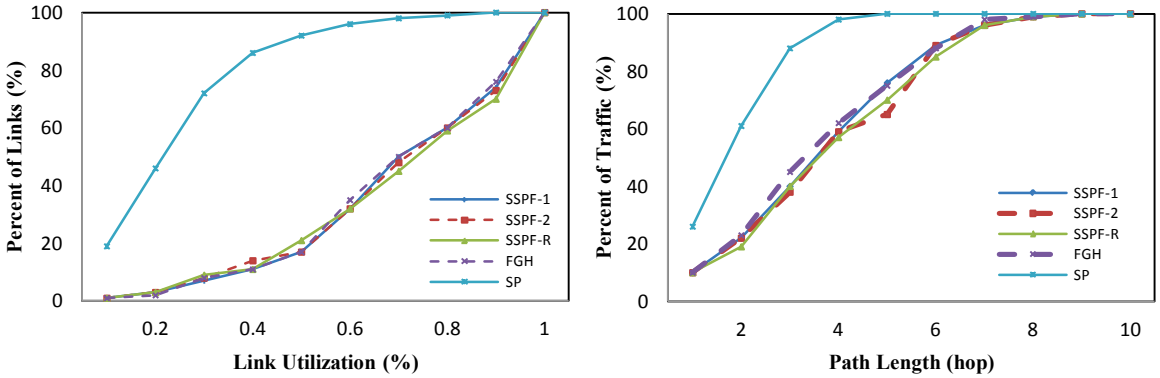


Figure 10: The Potential Energy Saving on Real Abilene on Sept. 5, 2004



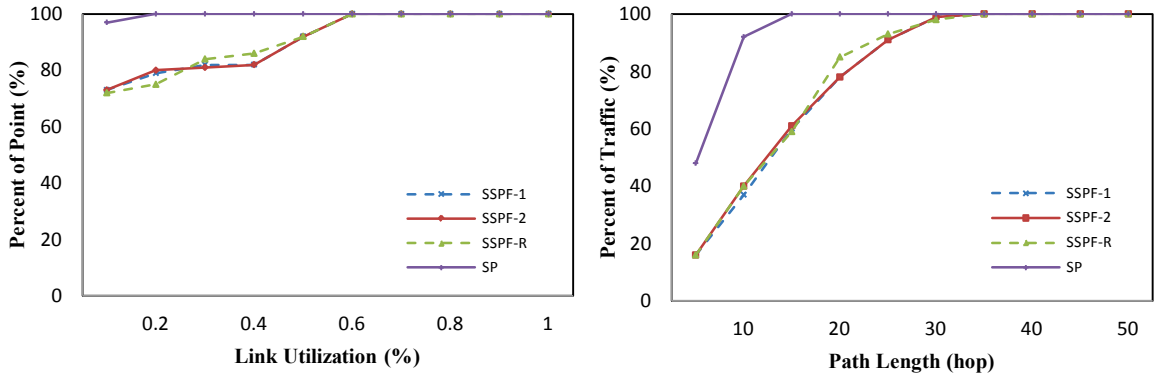
(a) (b)

Figure 11: CDF of Link Utilization and Packet Delay on the Abilene network



(a) (b)

Figure 12: CDF of Link Utilization and Packet Delay on the Wax50 network



(a) (b)

Figure 13: CDF of Link Utilization and Packet Delay on the Hier100 network

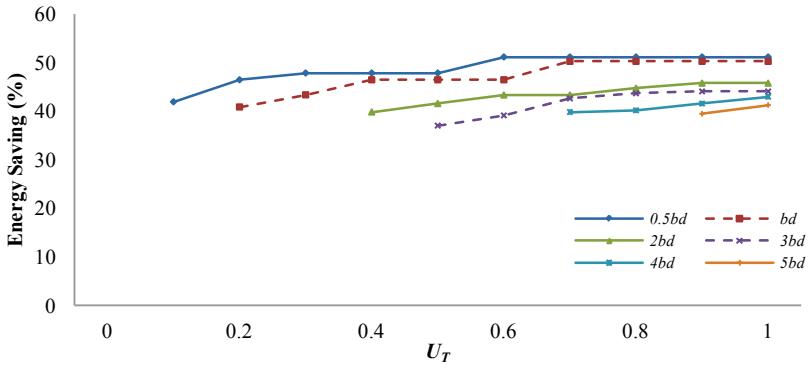


Figure 14: Impact of Different Traffic Levels and U_T on Energy Saving in Hier100