Computer Networks 79 (2015) 323–344

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Joint overlay routing and relay assignment for green networks ☆

Fatma Ekici [a], Didem Gözüpek [b,*]

[a] Scientific and Technological Research Council of Turkey (TUBITAK), Gebze, Kocaeli, Turkey
[b] Department of Computer Engineering, Gebze Technical University, Kocaeli, Turkey

ARTICLE INFO

Article history:
Received 14 May 2014
Received in revised form 24 November 2014
Accepted 10 January 2015
Available online 28 January 2015

Keywords:
Green networks
Routing
Optimization
Integer programming
Approximation algorithms
Heuristic algorithms

ABSTRACT

Power consumption of information and communication technologies (ICT) has increasingly become an important issue in the last years. Both energy costs and environmental concerns call for energy aware "green" networking solutions in wired networks. Overlay routing is an attractive method to enhance the performance and reliability of routing mechanisms without the need to change the standards of the current underlying routing. In this work, we focus on overlay routing in wired networks from an energy efficiency perspective. We formulate an optimization problem called *JORRA* (Joint Overlay Routing and Relay Assignment), which jointly determines the overlay routing paths and relay nodes. We consider issues such as the relay costs, whether the network elements can be put into sleep mode or not as well as the energy efficiency and reliability tradeoff for source and destination pairs in the network. We formulate *JORRA* as an integer linear program and prove that it is APX-Hard in addition to being NP-Hard even in its special cases. We then show that another special case of *JORRA* admits a 2-approximation algorithm. Moreover, we propose two polynomial time heuristic algorithms and demonstrate through performance evaluation that our heuristics are suitable for practical implementation.

1. Introduction

Environmental concerns have increased in recent years mainly due to the increase in greenhouse gas emissions. To this end, the usage of renewable energy sources (such as solar panels and wind turbines) as opposed to traditional ones (such as coal and fuel) has gained importance [1]. These environmental concerns also have a technological aspect. For instance, wired networks have traditionally been designed without considering energy efficiency. However, there is a continuous increase in their energy consumption and therefore, energy efficiency has an increasing importance in wired networks. Already in 2007 information and communications technologies (ICT) industry accounted for 2% of the global CO_2 emissions, same amount as global air travel [2]. Studies show that transmitting data through Internet takes more energy (in bits per Joule) than transmitting data through wireless networks [3]. This increasing energy consumption not only has an environmental cost but also a financial burden in terms of electricity costs and cooling equipments. For instance, powering wired networks in USA costs approximately 0.5–2.4 billion dollars per year [4].

Providing alternate paths for a set of source and destination pairs in a communication network achieves reliability and robustness against path failures. Overlay routing has been proposed in recent years as an effective method to achieve path diversity [5]. The alternate path that is different from the default (underlay) path is called an *overlay path*. To coordinate the communication over the alternate path, some nodes on the alternate path need to be

☆ This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under Grant No. 113E567.

* Corresponding author.
E-mail addresses: fatma.ekici@tubitak.gov.tr (F. Ekici), didem.gozupek@gtu.edu.tr (D. Gözüpek).

http://dx.doi.org/10.1016/j.comnet.2015.01.005
1389-1286/

equipped with extra functionality. These nodes are called *overlay nodes*, *relay nodes* or *infrastructure nodes*. Some works in the literature such as [6] propose a routing strategy that finds a path passing through the intermediate (relay) nodes assuming that the intermediate nodes are predetermined. Some other works such as [5] study the reverse problem and focus on the relay placement problem where an overlay path is a path that consists of two shortest paths, one from the source to the relay node and another from the relay node to the destination. The work in [7] follows a similar strategy except that the cost of the relay nodes is also taken into account. In other words, the set of overlay paths is given as input to their optimization problem. To the best of our knowledge, ours is the first study that jointly determines the alternate paths and the relay nodes by also taking the relay costs into account.

It is estimated that switches, hubs, and routers consume 6 TW h per year in the US and costs about $500 million per year [8]. Some recent studies show that traffic load of the routers has little effect on their energy consumption. The main cause of energy consumption is the switched-on network elements such as routers and interfaces. Network elements are usually powered on 24/7 in the idle mode, during which they consume a large amount of energy. Therefore, researchers have proposed to put the devices into low-energy sleep states [3]. However, not all networking equipment can be put into sleep mode due to hardware limitations or topological constraints. For instance, authors in [3] state that the Internet hardware in 2003 does not have sleeping capability. Moreover, some equipments such as gateways may take a long time to switch to the active mode from the sleep mode and therefore putting these devices into sleep mode may not be preferred [9]. Most studies on energy-aware wired networks [4,10,11] focus on cases where all equipments can potentially be put into sleep mode. However, modern energy-aware devices that have sleep functionalities will have to coexist with devices that do not have these capabilities since it is not feasible to quickly upgrade all of the Internet hardware at least for a considerable amount of time. In this paper, while determining the alternate paths, we suggest favoring the paths that pass through the nodes that cannot be put into sleep mode. These devices will have to be in the active state in any case; therefore, having the alternate paths utilize these nodes instead of other nodes that can be put into sleep mode help decrease the overall energy consumption in the network. Note here that we do not force the alternate paths to pass through the nodes that cannot be put into sleep mode; if it is more advantageous in terms of other criteria such as reliability, then the routing solution offered by our model may not pass through the nodes that cannot be put into sleep mode. Our model basically takes into account the potential savings from energy consumption that passing through these nodes can offer. To the best of our knowledge, this paper is the first study that takes the different sleeping capabilities of the networking equipment into account.

On the one hand, making the alternate and default paths as disjoint as possible is important in order to increase reliability and robustness. On the other hand, putting the nodes and links that are not on a default or alternate path into sleep mode helps decrease the energy consumption. Therefore, increasing the overlapping edges between default and alternate paths help decrease the energy consumption in the network. Furthermore, each source and destination pair may have a different reliability and fault tolerance requirement depending on the applications they execute; i.e., some pairs may tolerate more overlap with the default path and other alternate paths, whereas some other pairs may tolerate very few or no overlap. In this paper, we address the reliability and energy efficiency tradeoff by also considering the heterogeneous fault tolerance requirements of different source and destination pairs. To the best of our knowledge, previous works on energy-aware routing in wired networks [10–13] do not address these heterogeneous requirements.

The rest of this paper is organized as follows: Section 2 discusses related work and summarizes our contributions. Section 3 provides our problem formulation, whereas Section 4 presents our NP-hardness and inapproximability results as well as approximation algorithms for some special cases of our formulated problem. Section 5 introduces our proposed heuristic algorithm. Section 6 presents simulation results and Section 7 concludes the paper.

2. Related work and summary of contributions

2.1. Related work

The usage of path diversity to provide fault tolerance and load balancing is initially introduced in [14] as *dispersity routing*. Studies in [15] show that in 30–80% of the cases, an alternate path with significantly superior quality exists on the Internet. Another study [16] shows that more than 20% of Internet path failures are not recovered within 10 min. Advantages of alternate paths are investigated also by [17–19].

One way of achieving path diversity is *overlay routing*, which refers to the usage of alternate paths called overlay paths in addition to the default path between source and destination pairs. Overlay path passes through a strategically placed node called relay node, overlay node or infrastructure node. Relay nodes are equipped with extra functionality to coordinate the communication across the overlay path.

The work in [5] focuses on the problem of placing the relay nodes such that every pair has an overlay path that is as disjoint as possible from the default path and at the same time passes through a relay node. When it is not possible to achieve complete disjointness, a penalty metric is used for partially disjoint paths. The case where an overlay path consists of two shortest paths, one from the source node to the relay node and the other from the relay node to the destination node, is considered. In particular, authors focus on the problem of finding the positions of relays in the network such that every pair finds an overlay path that is maximally disjoint from the default path when the number of relay nodes is given.

The work in [6] proposes a routing strategy that routes traffic to the destination after ensuring that it passes through a pre-determined intermediate node. Their scheme is oblivious of and robust to any changes in the traffic distribution. The focus is on coping with traffic uncertainty; issues such as overlaps between different paths are not addressed.

The deployment and management of overlay nodes over the physical infrastructure has a non-negligible cost since the overlay nodes have to be equipped with extra functionality. The work in [7] is the first work in the literature that takes into account the cost associated with deploying these relay nodes. Given a set of overlay and underlay paths, they focus on the problem of finding a set of overlay nodes with minimum total cost such that the required routing properties are satisfied. In this paper, unlike the works in [5–7], we focus on an optimization problem that jointly finds overlay paths and relay nodes. To the best of our knowledge, this paper is the first one in the literature that focuses on such a joint optimization. Moreover, as in [7], we also take into account the costs associated with deploying the relay nodes.

Green networking is a recent paradigm that aims to address the increasing energy consumption of ICT sector [3,2,20]. Due to environmental and financial reasons, green networking concept presents an energy efficiency perspective on traditional networking paradigms. To the best of our knowledge, overlay routing concept has not previously been studied from a green networking perspective. Hence, to our knowledge, this paper is the first study in the literature that focuses on overlay routing for green networks.

Up until the emergence of the green networking paradigm, keeping network elements in always-on state even when they are inactive has been the main trend. Research in green networks shows that the main cause of energy consumption in wired networks is these switched-on and idle network elements. Therefore, researchers propose to put the unused network elements into sleep mode in order to save power [3,10,12,21,22]. Studies in green networks assume that all networking devices can be put into sleep mode. Nevertheless, some networking equipments, especially old equipments, do not have this functionality [3] because of hardware limitations, etc. To the best of our knowledge, this paper is the first study in the literature that takes into account the fact that some networking devices cannot be put into sleep mode.

Green overlay routing not only requires an energy efficient routing scheme but also a reliable communication over the overlay network. There are several studies on energy efficient routing but ours is the first study on energy efficient overlay routing. In essence, this paper presents a routing algorithm that combines both energy efficiency and resilience of the network. There are many studies in the literature about energy efficient routing. In [12], energy efficiency is presented as the minimization of the number of edges in a multicommodity integral flow problem. For this purpose the number of active links in the network is minimized. Since multi-commodity integral flow problem is NP-complete, two heuristics are suggested. The first one removes the less loaded edges and checks if there is still a feasible solution without that edge and continues until there is no feasible solution. Second heuristic differs from the first one by making the edge selection randomly. A similar, but distributed approach is presented in [23], where idle or underutilized links are switched off if this action does not affect the network functionality. The process of switching on/off the links is fully decentralized; i.e., it takes local decisions at random intervals and hence enables a more robust solution with respect to centralized approaches. Another work related to energy efficient routing uses a Steiner tree based algorithm [24], where authors show that the method drastically increases the number of sleeping nodes and links in a network. Their algorithm generates a Steiner tree connecting the source and destination nodes. In addition, their method calculates bypass routes that replace long inefficient hop-count routes to decrease the traffic congestion on the Steiner tree.

Energy saving on a network can also be obtained by partial shut down of certain network elements according to their loads and traffic patterns. This approach is advocated in [25], where the energy consumption of network elements that is independent of traffic load is questioned. It is suggested that the energy consumption should be proportional to the current traffic load on that network element. This approach requires traffic engineering, which is not always possible since it is expensive for some systems to represent different network states in time domain due to critical data that should not be lost in the network.

Energy efficient network design is also important like energy aware routing. In [26], these two problems are addressed jointly and expressed as a mixed nonlinear integer program. Energy aware routing is expressed as a nonlinear multicommodity flow problem, where links and nodes are powered off in order to reduce the overall network power consumption. In [11], energy consumption models for nodes and different types of links are constructed like in a real world system. A mixed integer program is designed by taking the traffic matrix and link capacities of a real world system. The work compares the energy saving model to the worst case scenario where no nodes or links can be shut down.

In addition to overlay routing, placing the relay nodes over the overlay network requires a strategic method. In many of the previous works [5,7,27], overlay node selection is done before determining the overlay paths. In [27], random placement, node degree-based and traffic-aware greedy heuristic algorithms are presented. Finally, it is concluded from the experiments that a hybrid approach combining greedy and random approaches provides the best tradeoff between computational efficiency and accuracy. In contrast to previous studies, we do not make the relay node selection at first stage. Instead, we make a joint optimization of path selection and relay node selection.

2.2. Summary of contributions

Our contributions can be summarized as follows:

1. To the best of our knowledge, this paper is the first work that focuses on joint optimization of alternate path finding and relay node selection.
2. To the best of our knowledge, this paper is the first work that takes into account and utilizes the fact that some networking devices may not have sleeping capability.

3. To the best of our knowledge, this paper is the first one in the literature that addresses the reliability and energy efficiency tradeoff in green networks by taking into consideration the heterogeneous fault tolerance requirements of different communication pairs.
4. To the best of our knowledge, this paper is the first work that focuses on overlay routing within the green networking paradigm.
5. Since we do not make any assumption on network topologies, our approach in this paper is applicable for networks with general topologies.

3. Problem formulation

We model the network as an undirected graph $G = (V, E)$, where V and E represent the nodes and links, respectively, of the network. We are given a set of source and destination pairs $Q = \{(s_1, d_1), (s_2, d_2), \ldots, (s_p, d_p), \ldots, (s_P, d_P)\}$, where $Q \subseteq V \times V$ and P is the total number of source and destination pairs. Here source and destination pairs are ordered because there is an implicit directionality in the decision variables defined in Table 2. This implication will be further elaborated on in the paper. We are also given a set of underlay paths, P_u, that altogether connects each source node to the corresponding destination node. Underlay paths are default paths that are derived from the underlying routing scheme. Furthermore, the following cost functions are also provided as input to our problem formulation:

- A weight function $W_1 : V - \bigcup_i \{s_i, d_i\} \to \mathbb{R}$ that indicate the cost associated with relay selection. Note that the set of source and destination nodes is excluded from the domain of W_1 because source and destination nodes cannot be selected as relay nodes (as also stated in [7]).
- A weight function $W_2 : V \to \mathbb{R}$ that indicate the cost associated with the energy consumption of the nodes.
- A weight function $W_3 : E \to \mathbb{R}$ that indicate the cost associated with the energy consumption of the links.

Table 1 outlines the input variables fed to our optimization problem. In essence, the variable t_{ij} is a function of the input variable t_i. In other words, the values for t_{ij} are actually enforced by the values for t_i; i.e., t_{ij} variables are not actually input variables. However, for better clarity, we state t_{ij} as input variable in Table 1. The fact that both ends of a link can be put into sleep mode implies that the link can be put into sleep mode; i.e., $t_{ij} \leqslant t_i + t_j$ since having $t_i = t_j = 0$ enforces that $t_{ij} = 0$. Moreover, the fact that any one end of a link cannot be put into sleep mode implies that the link cannot be put into sleep mode; i.e., $t_i \leqslant t_{ij}$ and $t_j \leqslant t_{ij}$ since having either $t_i = 1$ or $t_j = 1$ enforces that $t_{ij} = 1$. To summarize, the relationship between t_i and t_{ij} can be expressed as follows: $t_i \leqslant t_{ij} \leqslant t_i + t_j$ and $t_j \leqslant t_{ij}$.

Table 2 outlines the decision variables used by our integer linear programming formulation. The variable $\hat{x}_{ij}$ equals 1 if either $x_{ij} = 1$ or $x_{ji} = 1$. Besides, $\hat{x}_{ij} = 0$ if both $x_{ij} = 0$ and $x_{ji} = 0$. In other words, $\hat{x}_{ij} = 1$ if the edge between node i and j is used in either direction, whereas there is an implicit directionality from i to j in variable x_{ij}. To put it in another way, $\hat{x}_{ij} = \hat{x}_{ji}$; however, it is not necessarily true that $x_{ij} = x_{ji}$. Similar situation holds for the decision variables $\hat{x}_{ijp}, x_{ijp}$ and x_{jip}.

Each link belonging to any of the overlay paths consume energy that is equal to W_3^{ij}; hence, total energy consumption of the links that are part of the resulting overlay routing paths is equal to $\sum_{i=1}^{|V|}\sum_{j=i}^{|V|} W_3^{ij}\hat{x}_{ij}$. Besides, among the links that are not part of any overlay path, the ones that cannot be put into sleep mode have an additional energy consumption, which is equal to $\sum_{i=1}^{|V|}\sum_{j=i}^{|V|} t_{ij} W_3^{ij}(1 - \hat{x}_{ij})$. A similar situation exists for the nodes of the network: Each node belonging to any of the overlay paths consume energy that is equal to W_2^i; hence, total energy consumption of the nodes that are part of the resulting overlay routing paths is equal to $\sum_{i=1}^{|V|} W_2^i n_i$. Likewise, among the nodes that are not part of any overlay path, the ones that cannot be put into sleep mode have an additional energy consumption, which is equal to $\sum_{i=1}^{|V|} t_i W_2^i (1 - n_i)$.

As a result, the objective function of our integer linear programming (ILP) formulation for *JORRA* is as follows:

$$\min\left(\sum_{i=1}^{|V|}\sum_{j=i}^{|V|} W_3^{ij}\hat{x}_{ij} + \sum_{i=1}^{|V|}\sum_{j=i}^{|V|} t_{ij}W_3^{ij}(1-\hat{x}_{ij}) + \sum_{i=1}^{|V|} W_2^i n_i + \sum_{i=1}^{|V|} t_i W_2^i (1-n_i)\right) \tag{1}$$

Table 1
Table for input variables.

Input variable	Explanation
W_1^i	= The cost incurred if node i is selected as a relay
W_2^i	= The energy consumption of node i
W_3^{ij}	= The energy consumption of the link that connects node i and j
g^{ij}	$= \begin{cases} 1, & \text{if graph } G \text{ contains the edge that connects vertices } i \text{ and } j \\ 0, & \text{otherwise} \end{cases}$
t_i	$= \begin{cases} 1, & \text{if vertex } i \text{ cannot be put into sleep mode} \\ 0, & \text{otherwise} \end{cases}$
t_{ij}	$= \begin{cases} 1, & \text{if the edge that connects vertices } i \text{ and } j \text{ cannot be put into sleep mode} \\ 0, & \text{otherwise} \end{cases}$
u_{ij}	$= \begin{cases} 1, & \text{if the edge that connects vertices } i \text{ and } j \text{ is on an underlay path} \\ 0, & \text{otherwise} \end{cases}$
$k_p \in \mathbb{Z}$	= Maximum number of links that path p is allowed to share with other (overlay or underlay) paths
$r \in \mathbb{Z}$	= Upper bound (threshold value) for the total relay cost

Table 2
Table for decision variables.

Decision variable	Explanation
x_{ij}	$= \begin{cases} 1, & \text{if the edge that connects vertices } i \text{ and } j \text{ is selected in the overlay routing paths} \\ 0, & \text{otherwise} \end{cases}$
$\hat{x}_{ij}$	$= \begin{cases} 1, & \text{if either } x_{ij} = 1 \text{ or } x_{ji} = 1 \\ 0, & \text{otherwise} \end{cases}$
x_{ijp}	$= \begin{cases} 1, & \text{if the overlay path of pair } p \text{ goes from node } i \text{ to } j \text{ using the edge between them} \\ 0, & \text{otherwise} \end{cases}$
$\hat{x}_{ijp}$	$= \begin{cases} 1, & \text{if either } x_{ijp} = 1 \text{ or } x_{jip} = 1 \\ 0, & \text{otherwise} \end{cases}$
n_i	$= \begin{cases} 1, & \text{if node } i \text{ is selected in the overlay routing paths} \\ 0, & \text{otherwise} \end{cases}$
y_{ijp}	$= \begin{cases} 1, & \text{if the overlay path of pair } p \text{ shares the edge between node } i \text{ and } j \text{ with another overlay or underlay path} \\ 0, & \text{otherwise} \end{cases}$
s_{ijp}	$= \begin{cases} 1, & \text{if } u_{ij} = 1 \text{ or the overlay path of some pair } p' \neq p \text{ uses the edge between node } i \text{ and } j \\ 0, & \text{otherwise} \end{cases}$
z_i	$= \begin{cases} 1, & \text{if vertex } i \text{ is selected as a relay node} \\ 0, & \text{otherwise} \end{cases}$
m_{ip}	$= \begin{cases} 1, & \text{if vertex } i \text{ is not a source/destination node and is on the overlay path of pair } p \\ 0, & \text{otherwise} \end{cases}$
w_{ip}	$= \begin{cases} 1, & \text{if vertex } i \text{ is on the overlay path of pair } p \text{ and is selected as a relay node} \\ 0, & \text{otherwise} \end{cases}$

Resulting overlay links can only be among the links that exist in the network. We can model this natural requirement with the following constraint:

$$x_{ij} \leqslant g_{ij}; \quad \forall i,j \tag{2}$$

The following constraints model the relationships among the decision variables $\hat{x}_{ij}, x_{ij}$, and x_{ji}:

$$x_{ij} \leqslant \hat{x}_{ij} \leqslant x_{ij} + x_{ji}; \quad \forall i,j \tag{3}$$

$$\hat{x}_{ij} \geqslant x_{ji}; \quad \forall i,j \tag{4}$$

A node is part of the overlay routing paths if any of its incident edges is selected as part of the overlay paths. Besides, if none of the incident edges of a node is selected, then the node is not part of the overlay routing paths. This relationship of the variables $\hat{x}_{ij}$ with the variable n_i can be represented with the following set of constraints:

$$\hat{x}_{ij} \leqslant n_i \leqslant \sum_{j=1}^{|V|} \hat{x}_{ij}; \quad \forall i,j \tag{5}$$

Relationship between the variables x_{ij} and x_{ijp} also needs to be modeled. If $x_{ij} = 0$, then x_{ijp} variables have to be equal to zero for all pairs p. In addition, if $x_{ijp} = 0$ for all pairs p, then x_{ij} has to be equal to zero. We model these requirements with the following constraints:

$$x_{ijp} \leqslant x_{ij} \leqslant \sum_{p=1}^{P} x_{ijp}; \quad \forall i,j,p \tag{6}$$

For a particular pair p, the pertinent x_{ijp} variables need to form a path from the source node s_p to the destination node d_p of the pair p. The following constraints achieve this goal:

$$\sum_{j=1}^{|V|} x_{ijp} - \sum_{j=1}^{|V|} x_{jip} = \begin{cases} 1; & \text{if } i = s_p \\ -1; & \text{if } i = d_p \\ 0; & \text{otherwise} \end{cases} \tag{7}$$

Recall that the variable $\hat{x}_{ijp} = 1$ if either $x_{ijp} = 1$ or $x_{jip} = 1$. Besides, $\hat{x}_{ijp} = 0$ if both $x_{ijp} = 0$ and $x_{jip} = 0$. In other words, $\hat{x}_{ijp} = 1$ if the edge between node i and j is used in either direction, whereas there is an implicit directionality from i to j in the variable x_{ijp}. To put it in another way, $\hat{x}_{ijp} = \hat{x}_{jip}$; however, it is not necessarily true that $x_{ijp} = x_{jip}$. This relationship can be modeled by the following constraints:

$$\hat{x}_{ijp} \geqslant x_{jip}; \quad \forall i,j,p \tag{8}$$

$$x_{ijp} \leqslant \hat{x}_{ijp} \leqslant x_{ijp} + x_{jip}; \quad \forall i,j,p \tag{9}$$

The definition of the variable s_{ijp} states that $s_{ijp} = 1$ if $u_{ij} = 1$. Moreover, $s_{ijp} = 1$ if $\hat{x}_{ijp'} = 1$ for some $p' \neq p$. In all other cases, $s_{ijp} = 0$; in other words, if all of u_{ij} and $\hat{x}_{ijp'}$ variables equal zero, then $s_{ijp} = 0$. To put it in another way, $s_{ijp} = 1$ if the link between node i and node j is used by some underlay path or another overlay path different from path p. We can model these requirements by the following constraints:

$$u_{ij} \leqslant s_{ijp}; \quad \forall i,j,p \tag{10}$$

$$\hat{x}_{ijp'} \leqslant s_{ijp} \leqslant u_{ij} + \sum_{p'=1, p' \neq p}^{|P|} \hat{x}_{ijp'}; \quad \forall i,j,p \neq p' \tag{11}$$

The definitions of the variables $y_{ijp}, \hat{x}_{ijp}$ and s_{ijp} imply that $y_{ijp} = \hat{x}_{ijp} \times s_{ijp}$. In other words, $y_{ijp} = 1$ if the link between node i and node j is used by the path connecting pair p and it is shared with some underlay or another overlay path. We model this product of the decision variables by the following constraints:

$$y_{ijp} \leqslant \hat{x}_{ijp}; \quad \forall i,j,p \tag{12}$$

$$\hat{x}_{ijp} + s_{ijp} - 1 \leqslant y_{ijp} \leqslant s_{ijp}; \quad \forall i,j,p \tag{13}$$

Maximum number of links that path p is allowed to share with other overlay/underlay paths is k_p. This requirement can be modeled with the following constraint:

$$\sum_{i=1}^{|V|} \sum_{j=i}^{|V|} y_{ijp} \leqslant k_p; \quad \forall p \tag{14}$$

The definition of m_{ip} states that $m_{ip} = 1$ if vertex i is on the overlay path of pair p, and $m_{ip} = 0$ otherwise. The following constraint achieves this condition:

$$\hat{x}_{ijp} \leqslant m_{ip} \leqslant \sum_{j=1}^{|V|} \hat{x}_{ijp}; \quad \forall i, j, p \tag{15}$$

Note that a node that is not on any overlay path cannot be selected as a relay node. This requirement can be modeled as follows: $z_i \leqslant \sum_{j=1}^{|V|} \hat{x}_{ij}$. Furthermore, note that the definitions of the decision variables w_{ip}, m_{ip} and z_i imply that $w_{ip} = m_{ip} \times z_i$. We can model all of these requirements by the following set of constraints:

$$w_{ip} \leqslant z_i \leqslant \sum_{j=1}^{|V|} \hat{x}_{ij}; \quad \forall i, p \tag{16}$$

$$z_i + m_{ip} - 1 \leqslant w_{ip} \leqslant m_{ip}; \quad \forall i, p \tag{17}$$

The following constraint ensures that there is at least one relay node on each overlay path so that the communication over each overlay path can be coordinated:

$$\sum_{i=1, i \notin Q}^{|V|} w_{ip} \geqslant 1; \quad \forall p \tag{18}$$

Total cost of deploying relay nodes should not exceed a predetermined value denoted by the input variable r. The following constraint serves this purpose:

$$\sum_{i=1, i \notin \{s,d | (s,d) \in Q\}}^{|V|} W_1^i z_i \leqslant r \tag{19}$$

Finally, the constraint that all decision variables need to be binary decision variables can be stated as follows:

$$x_{ij}, \hat{x}_{ij}, x_{ijp}, \hat{x}_{ijp}, n_i, y_{ijp}, s_{ijp}, z_i, m_{ip}, w_{ip} \in \{0, 1\} \tag{20}$$

4. Computational complexity

4.1. Preliminaries

Approximation algorithms: Let Π be a minimization problem and $\rho \geqslant 1$. A (feasible) solution s of an instance I of Π is a *ρ-approximation* if its objective function value $O_\Pi(s)$ is at least a factor ρ of the optimal objective function value $O^*_\Pi(I)$ of I, i.e., $O_\Pi(s) \leqslant \rho O^*_\Pi(I)$. An algorithm *ALG* is said to be a *ρ-approximation algorithm* for a minimization problem Π if *ALG* returns a ρ-approximation for every instance I of Π supplied to it. A problem Π is said to be *ρ-approximable* if there is a polynomial-time ρ-approximation algorithm for it. Π is said to be *ρ-inapproximable* if there is no polynomial-time ρ-approximation algorithm for it unless $P = NP$. If there exists a constant ρ such that there is a ρ-approximation algorithm for a problem Π, then Π is said to be in *APX*. If there exists a constant ρ' such that Π is ρ'-inapproximable, then Π is said to be in APX-Hard. If a problem is both in *APX* and APX-Hard, then the problem is said to be APX-Complete.

Suppose that we have two optimization problems Π and Π' such that instances of one problem can be mapped onto instances of the other in a way that nearly-optimal solutions to instances of the latter problem can be transformed back to yield nearly-optimal solutions to the former. This way, if we have an approximation algorithm for problem Π', and an efficient approximation-preserving reduction from problem Π to problem Π', by composition we obtain an approximation for problem Π. More formally, an *approximation ratio preserving (polynomial time) reduction* from a minimization problem Π to a minimization problem Π' is a pair of algorithms (f, g) such that (a) f transforms every instance I of Π to an instance $I' = f(I)$ of Π', and (b) g transforms every ρ-approximation s' of $I' = f(I)$ to a ρ-approximation $g(s')$ of I. We denote this fact by $\Pi \preceq_{APX} \Pi'$. Π and Π' are said to be equivalent under approximation preserving reductions if $\Pi \preceq_{APX} \Pi'$ and $\Pi' \preceq_{APX} \Pi$. If $\Pi \preceq_{APX} \Pi'$, then if there exists a ρ-approximation algorithm for Π', we can then get a ρ-approximation algorithm for Π. Likewise, if Π cannot have a ρ-approximation algorithm unless $P = NP$, then Π' is also ρ-inapproximable.

Steiner tree problem: Given an undirected graph $G = (V, E)$ with nonnegative edge costs and whose vertices are partitioned into two sets, *required* and *Steiner*, Steiner Tree problem is to find a minimum cost tree in G that contains all the required vertices and any subset of the Steiner vertices [28].

Steiner forest/minimum point-to-point connection problem: Given an undirected graph $G = (V, E)$, a nonnegative cost function on edges and a collection of disjoint subsets of V referred to as a set of source and destination pairs, i.e., $Q = \{(s_1, d_1), (s_2, d_2), \ldots, (s_p, d_p), \ldots, (s_P, d_P)\}$ where $Q \subseteq V \times V$ and P is the total number of source and destination pairs, Steiner Forest problem is to find a minimum cost subgraph in which each pair of vertices belonging to the same set (s_p, d_p) is connected. We can restate the problem in the following way: Define a *connectivity requirement function* r that maps unordered pairs of vertices to $\{0, 1\}$ as follows:

$$r(u, v) = \begin{cases} 1; & \text{if } u \text{ and } v \text{ belong to the same set } \{s_p, d_p\}, \\ 0; & \text{otherwise} \end{cases} \tag{21}$$

Steiner Forest problem is to find a minimum cost subgraph F that contains a $u - v$ path for each pair (u, v) with $r(u, v) = 1$. In general, the solution will be a forest. Steiner Forest problem appears in the literature also under the name of Minimum Point to Point Connection problem [29].

Minimum cost edge-disjoint paths problem: Given an undirected graph $G = (V, E)$, a nonnegative cost function on edges and Q as defined previously, Minimum Cost Edge Disjoint Paths problem is to find a minimum cost subgraph in which each pair of vertices belonging to the same set (s_p, d_p) is connected and the paths connecting the source and destination pairs have no common edges. In other words, Minimum Cost Edge Disjoint Paths problem is to find a Steiner forest with the property that each path is mutually edge-disjoint.

4.2. Complexity of JORRA problem

Theorem 1. Steiner Tree $\preceq_{APX}$ *JORRA*.

Proof. We can show that Steiner Tree problem is a special case of *JORRA* problem as follows: Let $W_2^i = t_i = u_{ij} = 0\ \forall i, j$; i.e., energy consumption of the nodes equals zero, all nodes can be put into sleep mode, and there are no underlay paths. Furthermore, let $k_p = r = M\ \forall p$, where

M is a very large number. Moreover, let $s_p = s_{p'}$ and $d_p \neq d_{p'}$ $\forall p, p' \in \{1, 2, \ldots, P\}$; i.e., a single node is the source node for all pairs, whereas all destination nodes are distinct. In this special case, second, third and fourth terms of the objective function in (1) equal zero, constraints (10)–(14), which are related to the sharing of edges among the overlay paths, as well as constraints (15)–(19), which are related to relay costs, are redundant and hence can be removed from the formulation. This special case corresponds to the Steiner Tree problem, where the vertices corresponding to the source node and destination nodes are the required vertices and all other vertices are Steiner vertices. □

Corollary 1. *There exists a* $1 + (\ln 3)/2$*-approximation algorithm for the special case of the JORRA problem, where* $W_2^i = t_i = u_{ij} = 0$ $\forall i, j, k_p = r = M$ $\forall p$*, where* M *is a very large number, and* $s_p = s_{p'}$ *and* $d_p \neq d_{p'}$ $\forall p, p' \in \{1, 2, \ldots, P\}$.

Proof. This special case corresponds to the Steiner Tree problem, for which there is a $1 + (\ln 3)/2$-approximation algorithm [30]. □

Corollary 2. *JORRA problem is* NP-*Hard in the strong sense even when* $W_2^i = t_i = u_{ij} = 0$ $\forall i, j, k_p = r = M$ $\forall p$*, where* M *is a very large number, the underlying network is a complete graph with* $W_3^{ij} \in \{1, 2\}$ $\forall i, j$*, and* $s_p = s_{p'}$ *and* $d_p \neq d_{p'}$ $\forall p, p' \in \{1, 2, \ldots, P\}$.

Proof. The special case of Steiner Tree problem where the underlying graph is complete and all edge weights are either 1 or 2 is referred to in the literature as *STEINER*$(1, 2)$. Due to Theorem 1 and the fact that *STEINER*$(1, 2) \preceq_{APX}$ Steiner Tree, *STEINER*$(1, 2) \preceq_{APX}$ *JORRA*. Since *STEINER*$(1, 2)$ is NP-Hard in the strong sense [31], Corollary 2 holds. □

Corollary 3. *JORRA problem is* APX-*Hard even when* $W_2^i = t_i = u_{ij} = 0$ $\forall i, j, k_p = r = M$ $\forall p$*, where* M *is a very large number, the underlying network is a complete graph with* $W_3^{ij} \in \{1, 2\}$ $\forall i, j, s_p = s_{p'}$*, and* $d_p \neq d_{p'}$ $\forall p, p' \in \{1, 2, \ldots, P\}$.

Proof. As proved in Theorem 1 and Corollary 2, this special case corresponds to the *STEINER*$(1, 2)$ problem, which is APX-Hard [31]. □

Corollary 4. *There exists a 1.28-approximation algorithm for the special case of JORRA problem where* $W_2^i = t_i = u_{ij} = 0$ $\forall i, j, k_p = r = M$ $\forall p$*, where* M *is a very large number, the underlying network is a complete graph with* $W_3^{ij} \in \{1, 2\}$ $\forall i, j, s_p = s_{p'}$*, and* $d_p \neq d_{p'}$ $\forall p, p' \in \{1, 2, \ldots, P\}$.

Proof. This special case corresponds to the *STEINER*$(1, 2)$ problem, for which there exists a 1.28-approximation algorithm [31]. □

Corollary 5. *The special case of the JORRA problem where* $W_2^i = t_i = u_{ij} = 0$ $\forall i, j, k_p = r = M$ $\forall p$*, where* M *is a very large number, the underlying network is a complete graph with* $W_3^{ij} \in \{1, 2\}$ $\forall i, j, s_p = s_{p'}$*, and* $d_p \neq d_{p'}$ $\forall p, p' \in \{1, 2, \ldots, P\}$ *is APX-complete.* □

Proof. This result follows from Corollaries 3 and 4, which state that this special case is APX-Hard and in APX, respectively. □

Theorem 2. Steiner Forest $\preceq_{APX}$ *JORRA.*

Proof. We can show that Steiner Forest problem is a special case of *JORRA* problem as follows: Let $W_2^i = t_i = u_{ij} = 0$ $\forall i, j$; i.e., energy consumption of the nodes equals zero, all nodes can be put into sleep mode, and there are no underlay paths. Furthermore, let $k_p = r = M$ $\forall p$, where M is a very large number. This special case corresponds to the Steiner Forest problem, which also appears as Minimum Point to Point Connection problem in the literature, and hence Theorem 2 follows. □

Corollary 6. *There exists a* $2 - (1/P)$*-approximation algorithm for the special case of the JORRA problem, where* $W_2^i = t_i = u_{ij} = 0$ $\forall i, j, k_p = r = M$ $\forall p$*, where* M *is a very large number.*

Proof. As proved in Theorem 2, this special case corresponds to the Steiner Forest problem, for which there exists a $2 - (1/P)$-approximation algorithm [32]. □

Theorem 3. Minimum Cost Edge Disjoint Paths $\preceq_{APX}$ *JORRA.*

Proof. We can show that Minimum Cost Edge Disjoint Paths problem is a special case of *JORRA* problem as follows: Let $W_2^i = t_i = u_{ij} = 0$ $\forall i, j$; i.e., energy consumption of the nodes equals zero, all nodes can be put into sleep mode, and there are no underlay paths. Furthermore, let $r = M$, where M is a very large number, and $k_p = 0$ $\forall p$. In other words, overlay paths are not allowed to share any edge with other overlay paths; i.e., they need to be edge-disjoint. This special case corresponds to the Minimum Cost Edge Disjoint Paths problem and hence the theorem follows. □

Corollary 7. *JORRA problem is* NP-*Hard even when* $W_2^i = t_i = u_{ij} = 0$ $\forall i, j, r = M$*, where* M *is a very large number,* $k_p = 0$ $\forall p$ *and* $P = 2$.

Proof. As a consequence of Theorem 3, this special case corresponds to the Minimum Cost Edge Disjoint Paths problem with two pairs, i.e. $P = 2$. The work in [33] states that Minimum Cost Edge Disjoint Paths problem is NP-Hard even when $P = 2$; hence, the corollary follows. □

Corollary 8. *JORRA problem is NP-Hard even when* $W_2^i = t_i = u_{ij} = 0$ $\forall i, j, r = M$*, where* M *is a very large number,* $k_p = 0$ $\forall p$ *and the underlying network is a grid graph.*

Proof. As a consequence of Theorem 3, this special case corresponds to the Minimum Cost Edge Disjoint Paths problem in a grid graph. The work in [34] states that Minimum Cost Edge Disjoint Paths problem is NP-Hard even when the underlying graph is a grid graph; hence, the corollary follows. □

5. Heuristic algorithms

Our analytical findings in Section 4 prove that *JORRA* is a computationally very difficult problem. Therefore, designing efficient heuristic algorithms for *JORRA* is vital. To this end, we propose in this section two polynomial-time heuristic algorithms for *JORRA*.

Both heuristics consist of three consecutive phases: path selection, path correction and relay node selection. In path selection phase, we construct overlay paths as an initial temporary solution. In path correction phase, we check these overlay paths, which were constructed in the path selection phase, for (in)feasibility. If any of these overlay paths causes a constraint violation, we discard this path and find an alternative path. In path correction phase, we use Yen's k-shortest path algorithm [35] to find alternative paths. In relay node selection phase, we select relay nodes among the nodes constituting the overlay paths.

5.1. *Minimum Cost Overlay Path Algorithm (MCOPA)*

MCOPA is a greedy algorithm where most of the work is done in path correction phase. We first start by constructing a forest where all source and destination pairs are connected. Afterward, we choose the minimum energy paths, i.e., paths that have minimum total energy consumption of nodes and links for that particular source and destination pair, on the constructed forest as temporary overlay paths; this way, an initial solution, which completes the path selection phase, is obtained. This initial solution already satisfies the constraints specified in (2)–(13) since the selected temporary paths are on the input graph and they ensure the connectivity of the source and destination pairs. The remaining constraints (14)–(20) are not completely satisfied until the path correction and relay selection phases are finished.

After path selection, path correction is done on the temporary overlay paths. Path correction is mainly associated with constraint (14), which states that the maximum number of links that an overlay path p is allowed to share with other overlay and underlay paths is k_p. In order to satisfy this constraint, we check if any of the overlay paths violate this constraint and we detect the pairs causing the constraint violation. We correct the overlay paths of the violating pairs by finding alternative paths for those pairs by using Yen's k-shortest path algorithm [36].

After correcting all violations in overlay paths, *MCOPA* continues with relay selection phase and ensures that total relay cost does not exceed the given upper bound r. We select relay nodes among the nodes belonging to overlay paths and we do not select a node as a relay if that node is a source or destination of an overlay path. We also ensure that each source and destination pair has at least one relay node on the overlay path that connects them. Therefore, in relay selection phase we satisfy the constraints (15)–(20).

Algorithm 1 describes *MCOPA*, which takes as input the following: $G = (V, E)$, the set of source and destination pairs $Q = \{(s_1, d_1), \ldots, (s_p, d_p), \ldots, (s_P, d_P)\}$, the set of underlay paths $U = \{u_1, \ldots, u_p, \ldots, u_P\}$, the set of upper limits for edge sharing $K = \{k_1, \ldots, k_p \ldots, k_P\}$, the upper bound r for total relay cost and cost matrices $W_1 = [W_1^i], W_2 = [W_2^i]$ and $W_3 = [W_3^{ij}]$. W_1^i indicates the relay cost and W_2^i indicates the energy consumption of the node i. W_3^{ij} gives the energy consumption of the link between node i and node j. The output parameters are the set of overlay paths Φ and the set of relay nodes R. F indicates the forest where all source and destination pairs are connected.

Algorithm 1. Minimum Cost Overlay Path Algorithm (*MCOPA*).

Require: $G, Q, U, K, r, W_1, W_2, W_3$
Ensure: Φ, R
1: $\Phi \leftarrow \emptyset, R \leftarrow \emptyset, F \leftarrow \emptyset$
2: $(\Phi, F) \leftarrow$ *PathSelectionMCOPA* (G, Q, W_2, W_3)
▹ Described in Table 4
3: $\Phi \leftarrow$ *PathCorrectionMCOPA* (Φ, F, K, U)
▹ Described in Table 5
4: **if** $\Phi = \emptyset$ **then**
5: **return** $(\emptyset, \emptyset)$
6: **end if**
7: $R \leftarrow$ *RelaySelectionMCOPA* (Φ, Q, W_1, r)
▹ Described in Table 6
8: **return** (Φ, R)

In Line 2, path selection phase of *MCOPA* is executed. Path selection phase takes the graph D and the set of source and destination pairs Q as input. It constructs an initial solution by generating the set of initial overlay paths Φ. It also returns a forest F, which is later used in the path correction phase. The second phase is path correction, which is executed by calling *PathCorrectionMCOPA*(Φ, F, K) in Line 3. Path correction phase iteratively runs Yen's k-shortest path algorithm [35] for the pairs having constraint violation. If path correction phase fails, it returns an empty set. In this case, there is no need to continue with relay selection; hence we terminate the algorithm immediately in Line 5 and declare that no feasible solution is found. After path correction, relay node selection phase is executed in Line 7, which takes as input the set of corrected overlay paths Φ and upper bound r for total relay cost. If the relay selection phase cannot find a feasible solution, it returns an empty set. Consequently, if *MCOPA* algorithm returns an empty set for either Φ and/or R, it means the algorithm cannot find a feasible solution in that case. Detailed descriptions of the methods we use in Algorithm 1 are given as follows:

Undirected to directed graph conversion: We convert the undirected input graph G to a directed and edge weighted graph since Dijkstra's shortest path algorithm used in the path selection phase and Yen's k-shortest path algorithm do not handle the input graphs having both node weights and link weights. Therefore, we need an algorithm to convert an undirected graph with link and node weights to an equivalent directed graph having only link weights. Table 3 gives the outline of this conversion algorithm. In Line 2, U is the set of vertices for the directed graph. In other words, U is a copy of V, the vertex set of the original input graph. The set of edges L is initially empty. In Lines 6 and 7, for each undirected edge, two directed edges are created. Weight of a directed edge is the summation of the weight of the original edge and the weight of the source node of the directed edge. This calculation basically provides a new weight function for edges so that a suitable input is prepared for Dijkstra's algorithm, which assumes no weights on the vertices and uses the edge weights only. The function returns the directed graph in Line 11.

Path selection phase for MCOPA: Path selection phase for *MCOPA* creates an initial solution for the main algorithm *MCOPA* to begin with. Details of this phase are given in Table 4. It takes $G' = (V', E')$ and the set of source and destination pairs (Q) as input. It also takes node and link energy consumption matrices (W_2 and W_3) as input. The first step is to construct the minimum spanning forest of the input graph with respect to W_2 and W_3 in Line 3. Therefore, when we calculate the minimum spanning forest, we prune the graph G by selecting the links and nodes having lower energy consumption. After calculating *MSF* in Line 3, next step is to obtain an edge weighted digraph D

Table 3
Graph conversion algorithm.

1: **function** GraphConversion $(G' = (V', E'), W_2, W_3\})$
2: $\quad U \leftarrow V', L \leftarrow \emptyset$
3: $\quad$ **for all** $e \in E'$ **do**
4: $\quad\quad i \leftarrow$ source node of e
5: $\quad\quad j \leftarrow$ destination node of e
6: $\quad\quad d_1 \leftarrow$ directed edge from i to j with weight $(W_3^{ij} + W_2^{i})$
7: $\quad\quad d_2 \leftarrow$ directed edge from j to i with weight $(W_3^{ij} + W_2^{j})$
8: $\quad\quad L \leftarrow L \cup \{d_1, d_2\}$
9: $\quad$ **end for**
10: $\quad D \leftarrow (U, L)$
11: $\quad$ **return** D
12: **end function**

Table 4
Path selection phase for *MCOPA*.

1: **function:** PathSelectionMCOPA $(G' = (V', E'), Q = \{(s_p, d_p) | p = 1, \ldots, P\}, W_2, W_3)$
2: $\quad \Phi \leftarrow \emptyset, F \leftarrow \emptyset$
3: $\quad$ Construct the minimum spanning forest *MSF* on G' by using Prim's algorithm wrt W_2 and W_3.
4: $\quad D \leftarrow GraphConversion(MSF, W_2, W_3)$
5: $\quad$ **for all** $(s_p, d_p) \in Q$ **do**
6: $\quad\quad p \leftarrow$ Minimum energy path for (s_p, d_p) on D using Dijkstra's algorithm
7: $\quad\quad \Phi \leftarrow \Phi \cup \{p\}$
8: $\quad\quad$ Mark all edges of p as unremovable on *MSF*
9: $\quad$ **end for**
10: $\quad F \leftarrow MSF \setminus \{e \in MSF \mid e \text{ is not marked}\}$
11: $\quad H \leftarrow F \cup \{e \in E \mid e \text{ does not have sleep mode}\}$
12: $\quad$ **return** $(\Phi, GraphConversion(H, W_2, W_3))$
13: **end function**

from *MSF* which is undirected. Line 4 performs this task using the algorithm in Table 3. The reason for this conversion is to provide a graph in the right form for Dijkstra's shortest path algorithm. After conversion to a directed graph, we detect the minimum energy paths for each source and destination pair (s_p, d_p) and construct the set of overlay paths in Lines 6 and 7. Afterward, we construct the graph H in Lines 10 and 11 as follows: We first remove all edges and nodes except the ones on the minimum energy paths from *MSF* and obtain the Steiner Forest F in Line 10. We then add all edges and nodes of G' that cannot be put into sleep mode to F since these edges cause energy consumption in any case and hence they have to be part of the overlay graph when we calculate the total energy consumption. This way, we construct a graph H where all source and destination pairs are connected with minimum energy consumption paths. Note that H is not necessarily a forest since adding extra edges in Line 11 might cause some loops in graph H. H contains all the links and nodes that cannot be put into sleep mode. Moreover, H does not contain any link or node having sleep mode unless it resides on a minimum energy path connecting some source and destination pair. At the end of the phase, we return the directed version of H since the next step of *MCOPA* requires a directed graph where Yen's k-shortest path algorithm is used if necessary.

Path correction phase for MCOPA: Path correction phase (Table 5) initially checks if any of the overlay paths causes the violation of constraint (14). If there is no such path, the function terminates and we continue with relay selection. Otherwise, we find alternative paths for the paths causing constraint violations. Path correction phase takes as input the set of initial overlay paths Φ, which were already determined in the path selection phase. Other parameters are as follows: Graph G', in which we search for alternative paths, K, which is the set of k_p values for each source and destination pair and U, which is the set of underlay paths. Lines 3–6 identify the paths that violate constraint (14). At the end of Line 6, the set Ω consists of paths that violate constraint (14). Our idea here is to replace these paths with other possible paths such that the total number of paths that violate constraint (14) decreases. To this end, we calculate alternative paths for the constraint violating paths by using Yen's k-shortest path algorithm [36], which finds k shortest paths between a given source and destination where the first path is the shortest one, the second path is the second shortest one, etc. In lines 10–20 we first calculate κ shortest paths for path p in terms of energy consumption and we then select the minimum energy path among them such that the size of Ω is decreased. Note that Ω is the set of paths having constraint violation. The for loop in Line 12 processes the paths in the list of k-shortest paths for the path p. In Line 13 we change the path p that has constraint violation with the alternative path π. Changing the overlay path p might affect all other overlay paths in terms of constraint violations since the number of shared links might have been changed. Therefore, in the for loop starting from Line 14, we recalculate the number of violations for each overlay path and construct the set Ω', which is the new set of paths having constraint

Table 5
Path correction phase for *MCOPA*.

1: **function** PathCorrectionMCOPA ($\Phi = \{p_i | i = 1, \ldots, P\},\ G' = (V',\ E'),\ K = \{k_p | p = 1, \ldots, P\},\ U = \{u_p | p = 1, \ldots, P\}$)
2: $\Omega \leftarrow \emptyset$ $\triangleright$ Ω is the set of paths that violate constraint (14).
3: **for all** $p \in \Phi$ **do**
4: $s \leftarrow$ Number of links that p shares with the other paths in U and Φ
5: **if** $s > k_p$ **then** $\Omega \leftarrow \Omega \cup \{p\}$
6: **end for**
7: $\kappa \leftarrow \lceil |E'|/|V'| \rceil * P$
8: **while** $|\Omega| > 0$ **do**
9: **for all** $p \in \Omega$ **do** $\triangleright$ A is the list of the k-shortest paths in terms of energy consumption
10: $A \leftarrow \{\rho_i | \rho_i = i^{\text{th}}$ minimum energy path for $(s_p, d_p), i = \{1, \ldots, \kappa\}$
11: $bestAlternative \leftarrow p$
12: **for all** $\pi \in A$ **do** $\triangleright$ A is sorted in ascending order wrt. path length.
13: $\Omega' \leftarrow \emptyset, p \leftarrow \pi$
14: **for all** $t \in \Phi$ **do**
15: $s \leftarrow$ Number of links that t shares with the other paths in U and Φ
16: **if** $s > k_t$ **then** $\Omega' \leftarrow \Omega' \cup \{t\}$
17: **end for**
18: **if** $|\Omega'| < |\Omega|$ **then** $bestAlternative \leftarrow \pi, \Omega \leftarrow \Omega'$, **break**
19: **end for**
20: $p \leftarrow bestAlternative$
21: **end for**
22: **if** $|\Omega|$ has not been changed for κ iterations **then break**
23: **end while**
24: **if** $|\Omega| > 0$ **then return** $\emptyset$
25: **return** Φ
26: **end function**

violations. In Line 18, we compare the size of Ω' with the previous set containing paths that have violations. If the number of paths having constraint violation is smaller, we choose π as the best alternative for the path p and then we proceed with the next overlay path with constraint violation.

κ is the k value given as an input to Yen's k-shortest path algorithm. It is crucial to choose an appropriate k value so that the k-shortest path algorithm finds enough number of alternative paths. Intuitively, a constant value for k might not be suitable for all input graphs. As the size of the input graph increases, k should increase as well. On the other hand, a larger graph might not necessarily imply that there are many alternative paths between source and destination nodes if the graph is sparse. Density of the input graph is a more determinative factor on k rather than the number of edges. In a dense graph, it is more likely to find many alternative paths. Therefore, we take the graph density into consideration while choosing the value for k. We have experimentally seen that k values that are proportional to the average degree of G, which is also an indicator of graph density, incur more feasible solutions. The average degree of a graph G shows how many edges are in set E compared to the number of vertices in set V. Because each edge is incident to two vertices and is counted while calculating the degree of both vertices, the average degree of an undirected graph is $2 \times |E|/|V|$. Therefore, we choose κ proportional to $2 \times |E|/|V|$. Another factor that we should take into account while choosing k is the number of source and destination pairs P. When there is a high number of source and destination pairs, it is more likely for an overlay path to have shared links causing constraint violation. In this case, we need more alternative paths to correct constraint violations. Because of these reasons, we set κ to $\lceil |E|/|V| \rceil \times P$, which is used as the k value for Yen's k-shortest path algorithm.

We also use κ for the termination condition of the while loop in Line 8. If the size of Ω does not change for κ iterations, i.e. we cannot make any further improvements on the overlay paths having violation, we terminate the algorithm in Line 22. At the end of the phase, if we succeed to correct all paths having violation, we return the new set of overlay paths; otherwise, we return an empty set meaning that we could not find a feasible solution.

Relay selection phase for MCOPA: After determining overlay paths, the next step is to choose relay nodes. Each source destination pair should have at least one relay node that coordinates the communication among them along the overlay path. Relay node selection phase, which we outline in Table 6, is a simple greedy heuristic that selects relay nodes among nodes residing on overlay paths. In Line 3 we detect all nodes residing on an overlay path and collect them in the set Υ. Afterward, we exclude the nodes that are either source or destination node from Υ (Line 4). For all nodes on overlay paths, we define a metric, which we call *relay metric*, to be used in determining whether the node will be selected as relay or not (Line 6). Relay metric is denoted as r_η for node η, which is as follows:

$$r_\eta = \frac{W_1^\eta}{|p \in \Phi | p \ni \eta|} \tag{22}$$

where $p \ni \eta$ refers to a path p that contains node η. Hence, $|p \in \Phi | p \ni \eta|$ refers to the number of overlay paths that contain node η. Recall that W_1^η refers to the cost associated with selecting node η as a relay node. Our rationale for using Eq. (22) is the following: As the relay cost W_1^η of a

Table 6
Relay selection phase for *MCOPA*.

1: **function** RelaySelectionMCOPA
$(\Phi = \{p_i | i = 1, \ldots, P\}, Q = \{(s_p, d_p) | p = 1, \ldots, P\}, r)$
2: $R \leftarrow \emptyset$ ▷ R is the set of relay nodes, initially empty.
3: $\Upsilon \leftarrow \{\eta \in p | p \in \Phi\}$ ▷ Υ is the set of nodes on overlay paths.
4: $\Upsilon \leftarrow \Upsilon \setminus \{s, d | (s, d) \in Q\}$
5: **for all** $\eta \in \Upsilon$ **do**
6: $r_\eta \leftarrow (W_1^\eta / |\{p \in \Phi | p \ni \eta\}|)$
7: **end for**
8: $\Psi \leftarrow \emptyset$ ▷ Ψ is the set of overlay paths having at least one relay node.
9: **while** $|\Psi| < |\Phi|$ **do**
10: $\upsilon \leftarrow \arg\min_{x \in \Upsilon} \{r_x\}$
11: $R \leftarrow R \cup \{\upsilon\}$
12: $\Upsilon \leftarrow \Upsilon \setminus \{\upsilon\}$
13: $\Psi \leftarrow \Psi \cup \{p \in \Phi | p \ni \upsilon\}$
14: **if** $|\Psi| == |\Phi|$ **then break**
15: **for all** $\eta \in \Upsilon$ ▷ Recalculate relay metric for all nodes on the overlay graph.
16: $r_\eta \leftarrow (W_1^\eta / |\{p \in \Phi | p \notin \Psi \text{ and } p \ni \eta\}|)$
17: **end for**
18: **end while**
19: **if** $\sum_{\upsilon \in R} r_\upsilon > r$ **then return** $\emptyset$
20: **return** R
21: **end function**

node η increases, its chances of being selected as a relay node decreases. In addition, as the number of overlay paths that contain the node increases, its chances of being selected as a relay node increases. Since the goal is to find a set of relay nodes with as little total cost as possible such that each overlay path has at least one relay vertex, a node with a higher relay cost might be a better candidate to be a relay node than the node with a lower relay cost if it resides on many overlay paths. We choose the node with the minimum relay metric r_η (Line 10) and repeat relay metric calculation in each iteration. We repeat this process until all source and destination pairs have a relay node. At the end of each iteration, we recalculate in line 16 the relay metric for nodes that have not yet been selected as a relay node. We make this recalculation because in each iteration the number of pairs that a node can connect changes and this number might decrease or stay the same. This change affects the relay metric of nodes. Therefore, we update the relay metric of all nodes and make the relay selection over the updated values.

5.2. Minimum Cost Overlay Path Algorithm with Link Overlap Avoidance (MCOPA–LOA)

In our experimental analysis (see Section 6), we have seen that *MCOPA* does not perform well in terms of finding feasible solutions on sparse graphs. Therefore, it is necessary to design another algorithm that produces more feasible solutions. To this end, we propose (*MCOPA–LOA*), which is another greedy algorithm and an improved version of *MCOPA*. We outline (*MCOPA–LOA*) in Algorithm 2. Unlike *MCOPA*, *MCOPA–LOA* does not create a feasible forest in path selection phase (Line 3). Instead, it selects initial overlay paths in a way that avoids using any link that is already being used by an underlay path or overlay path. The algorithm does not completely forbid any link sharing but it discourages it by setting the energy cost of the already used links to a high value.

The algorithm has nearly the same steps of *MCOPA* except that the path selection phase (Line 3) does not create a feasible forest connecting all source and destination pairs. Another difference is that we convert the input graph to a directed graph at the very beginning and instead of the feasible forest, we give the full input graph D to the path correction phase (Line 3). In Line 6, the algorithm terminates if the path correction step is unsuccessful. Otherwise, it continues with relay node selection step (Line 8). Finally, it returns the set of overlay paths Φ and the set of relay nodes R.

Algorithm 2. *MCOPA–LOA*

Require: $G, Q, U, K, r, W_1, W_2, W_3$
Ensure: Φ, R
1: $\Phi \leftarrow \emptyset, R \leftarrow \emptyset$
2: $D \leftarrow GraphConversion\ (G, W_2, W_3)$
3: $\Phi \leftarrow PathSelectionMCOPALOA\ (D, Q, U)$
4: $\Phi \leftarrow PathCorrectionMCOPA\ (\Phi, D, K, U)$
5: **if** $\Phi = \emptyset$ **then**
6: **return** $(\emptyset, \emptyset)$
7: **end if**
8: $R \leftarrow RelaySelectionMCOPA(\Phi, Q, W_1, r)$
9: **return**(Φ, R)

Since path correction and relay node selection phases are the same as the ones used in *MCOPA*, we only give the details of path selection phase in Table 7.

– *Path selection phase for MCOPA–LOA:* Path selection phase of *MCOPA–LOA* is different from the one used in *MCOPA* in two aspects: First, *MCOPA–LOA* avoids the usage of a link by more than one path. Second, unlike *MCOPA*, *MCOPA–LOA* does not eliminate from the solution the network elements that have sleep mode and that are not on a minimum energy path between some source and destination pair.

First of all, we set the energy cost of all links on under-

Table 7
Path selection phase for *MCOPA–LOA*.

function PathSelectionMCOPALOA
$(G' = (V', E'), Q = \{(s_p, d_p) | p = 1, \ldots, P\}, U = \{u_p | p = 1, \ldots, P\})$
$\Phi \leftarrow \emptyset$
for all $u_p \in U$ **do**
Set the energy cost of links on u_p to HIGH-COST on G'
end for
while $|Q| > 0$ **do**
$\Psi \leftarrow \emptyset$ ▷ Ψ is the set of minimum energy paths on G' for all pairs in Q
for all $(s_i, d_i) \in Q$ **do**
$\pi \leftarrow$ minimum energy path for (s_i, d_i) on G'
$\Psi \leftarrow \Psi \cup \{\pi\}$
end for
$p \leftarrow \arg\min_{\pi \in \Psi} (PL(\pi))$ ▷ $PL(\pi)$ is the length of path π
$\Phi \leftarrow \Phi \cup \{p\}$
Set the energy cost of links on p to HIGH-COST on G'
$Q \leftarrow Q \setminus \{(s_p, d_p)\}$
end while
return Φ
end function

lay paths to a high cost so that we obstruct overlay paths to share any links with underlay paths (Line 4). Afterward, we start determining initial overlay paths for source and destination pairs until all pairs have an overlay path (Line 6). In Lines 9 and 10, we calculate the minimum energy paths for each (s_p, d_p) pair in Q. Each path in the set of minimum energy paths is a candidate overlay path. Afterward, among these minimum energy paths in Ψ, we take the path p with the shortest path length in terms of hop count and set this path as a definite overlay path for the corresponding source and destination pair (s_p, d_p) (Lines 12, 13). Since path p is selected to be a definite overlay path, we should aim to make the sharing of this path's links with other overlay paths as small as possible. For this purpose, in Line 14 we set the energy cost of all links on path p to a high cost value. Moreover, we remove (s_p, d_p) from Q since this pair has an overlay path and should not be considered in the next iterations. The function returns the set of initial overlay paths Φ as an output.

- *Path correction phase for MCOPA—LOA:* Path correction phase is the same as the one used in *MCOPA*. Details can be found in Table 5. The only difference is that in *MCOPA* we give a forest as input parameter, whereas in *MCOPA—LOA* we give the whole graph D as input. This property of *MCOPA—LOA* might cause it to find more alternative paths and hence increases the possibility of correcting an overlay path with violated constraint(s). On the other hand, using the whole graph in this phase causes the total energy cost to be higher since there is no elimination of the network elements with sleep mode.
- *Relay selection phase for MCOPA—LOA:* Relay node selection is the third and final step in *MCOPA—LOA*. If relay node selection returns an empty set, it means the algorithm has failed to find a feasible solution. Relay node selection phase is the same as the one used in *MCOPA*. Details can be found in Table 6.

5.3. Time complexity analysis of our proposed heuristics

5.3.1. Best case analysis

The best case for both *MCOPA* and *MCOPA—LOA* is the situation where the path selection phase creates overlay paths having no constraint violation. In this case, we would not need to execute the path correction phase and we would directly proceed to the relay selection. As for the relay selection, the best case occurs when we select a node as a relay node and this node already connects all source and destination pairs. Since only the path selection phase is specific to each heuristic, let us first determine the best case time complexity of the path selection phases. For the sake of simplicity, we assume that number of vertices is V, number of edges is E and number of pairs is P.

- *Path selection phase for MCOPA:* The basic operations that are done in this phase are the calculation of the minimum spanning forest on the input graph, conversion of the input graph to a digraph and running Dijkstra's algorithm for each source and destination pair. Other operations take constant time; hence, they can be ignored. We use Prim's algorithm for *MSF* calculation with a priority queue. Complexity of this calculation is $\Theta(E \log V)$ since we use a priority queue. Graph conversion algorithm in Table 3 basically consists of a single for loop where each iteration performs construction of two edges and adding these edges to the newly constructed graph. Since the for loop contains E iterations, the complexity of graph conversion algorithm is $\Theta(E)$. Running the shortest path algorithm has $\Theta(E \log V)$ complexity since we use Dijkstra's algorithm with priority queue. Overall, the best case complexity of the path selection phase is $\Theta(E \log V) + \Theta(E) + P\Theta(E \log V)$, which reduces to $\Theta(PE \log V)$.
- *Path selection phase for MCOPA—LOA:* Path selection phase of *MCOPA—LOA* mainly consists of a main while loop, which processes the list of the source and destination pairs Q. Size of the list Q decreases exactly by one in each iteration. Single iteration of this while loop means running the shortest path algorithm for P times. Therefore, the best case complexity of this phase is as follows:

$$\sum_{i=1}^{P} i\Theta(E \log V) = \frac{P(P-1)}{2}\Theta(E \log V) = \Theta(P^2 E \log V) \tag{23}$$

- *Path correction phase:* The best case complexity of path correction phase for both heuristics is $\Theta(1)$ because in the best case, overlay paths resulting from the path selection phase would cause no constraint violations. Therefore, complexity of the path correction phase has no impact on the best case complexity of *MCOPA* and *MCOPA—LOA*.
- *Relay node selection phase:* The best case of this phase occurs when a single relay node is enough for all source and destination pairs. It means that the while loop in Line 9 of the relay selection phase in Table 6 executes just once. The first step is the calculation of the relay metric for the nodes that reside on the overlay paths. This step, i.e. the for loop in Line 6, has $\Theta(V)$ complexity. Afterward, in the while loop we search for the node having the minimum relay metric, which also has $\Theta(V)$ complexity. Since the while loop executes once in the best case, the overall complexity of the relay selection function phase is $\Theta(V)$.

The best case time complexity of two heuristics is the summation of the complexity of three phases, namely path selection, path correction and relay selection. For *MCOPA* the best case complexity is $\Theta(PE \log V) + \Theta(1) + \Theta(V)$, which reduces to $\Theta(PE \log V)$, whereas for *MCOPA—LOA* the best case complexity is $\Theta(P^2 E \log V) + \Theta(1) + \Theta(V)$, which reduces to $\Theta(P^2 E \log V)$.

5.3.2. Worst case analysis

Worst case scenario for our heuristics occurs when all overlay paths obtained in the path selection phase have at least one violated constraint. In this case, we need to find another path for each pair in the path correction phase. Moreover, we should also consider the worst case of the relay node selection phase, which occurs when each source and destination pair has a distinct relay node that is not shared with any other pair. In the following, we determine the worst case complexity of the three phases of our heuristic algorithms.

- *Path selection phase for MCOPA:* Recall that path selection operation is executed for all source and destination pairs. Therefore, path selection phase entails an inevitable computational work, which is the same for both worst case and best case scenarios. To this end, the worst case complexity of the path selection phase for *MCOPA* is $\Theta(PE\log V)$.
- *Path selection phase for MCOPA—LOA:* Like *MCOPA*, path selection phase of *MCOPA—LOA* has the same worst case complexity in its best case since it executes the same operation for each source and destination pair. Therefore, the worst case complexity of this phase for *MCOPA—LOA* is $\Theta(P^2E\log V)$.
- *Path correction phase:* Path correction phase is the same for both heuristics. In the worst case, all source and destination pairs would have at least one constraint violation requiring the path correction phase to run for each overlay path between source and destination pairs. For each pair, we need to run Yen's k-shortest path algorithm and select the best alternative path among k-shortest paths and thereby reducing the number of pairs having constraint violation. Since we try all paths one by one in the for loop in line 12 of Table 5, in the worst case the for loop does not terminate until we get to the last path in the list A. Therefore, the path assignment and recalculation of the number of shared edges in the for loop in Line 12 executes κ times. Moreover, in Line 14 recalculation of the number of shared edges has $\Theta(P)$ complexity. Hence, if we denote the complexity of Yen's k-shortest path algorithm by C_{kSPA}, the worst case complexity of the outer for loop starting from Line 9 is $P(C_{kSPA} + \kappa\Theta(P))$. The while loop in Line 8 terminates when the size of the paths with violations reduces to zero. Moreover, if the size of this list does not change for κ iterations we terminate the while loop. The worst case of this while loop occurs when the size of the list reduces just by one and this reduction occurs once in $(\kappa - 1)$ iterations since κ iterations without any change would cause the loop to terminate. Therefore, we need to multiply $P(C_{kSPA} + \kappa\Theta(P))$ with $(\kappa - 1)P$, which results in $P^2\kappa(C_{kSPA} + \kappa\Theta(P))$.
  The worst case complexity of Yen's k-shortest path algorithm is $C_{kSPA} = \Theta(\kappa V(E + V\log V))$ [36]. When we plug in $\kappa = \lceil E/V\rceil \times P$ and $C_{kSPA} = \Theta(\kappa V(E + V\log V))$, the worst case complexity of path correction phase becomes:
$$
\begin{aligned}
&P^2\kappa(\Theta(\kappa V(E + V\log V)) + \Theta(\kappa P))\\
&\quad = P^3(E/V)(\Theta(EP(E + V\log V)) + \Theta((E/V)P^2))\\
&\quad = \Theta(E^2P^4((E + V\log V)/V)) + \Theta((E^2P^5)/V^2)\\
&\quad = \Theta(E^2P^4(E/V + \log V + P/V^2))
\end{aligned}
$$
- *Relay node selection phase:* The worst case of this phase occurs when each source and destination pair has its own distinct relay node and it does not share the relay node with any other pair. In other words, in the worst case, the while loop in Line 9 of the relay selection phase (see Table 2) executes P times. The first step of this phase is the calculation of the relay metric for the nodes that reside on the overlay paths. This step, i.e. the for loop in lines 5–7, has $\Theta(V)$ complexity. Because in each iteration we recalculate the relay metric in Line 16 for all nodes on the overlay graph, the while loop in line 9 has $\Theta(V)$ complexity for each iteration. Since we have P iterations of the while loop, in the worst case, complexity of the while loop becomes $\Theta(PV)$. Therefore, the overall worst case complexity is $\Theta(PV)$.

The worst case complexity of *MCOPA* is $\Theta(PE\log V) + \Theta(E^2P^4(E/V + \log V + P/V^2)) + \Theta(PV)$, which reduces to $\Theta(E^2P^4(E/V + \log V + P/V^2))$. For *MCOPA—LOA*, the worst case complexity is $\Theta(P^2E\log V) + \Theta(E^2P^4(E/V + \log V + P/V^2)) + \Theta(PV)$, where the middle term dominates and the complexity becomes the same as in *MCOPA*. Therefore, we conclude that both heuristics have $\Theta(E^2P^4(E/V + \log V + P/V^2))$ worst case complexity. Furthermore, for dense graphs where $E = \Theta(V^2)$, the worst case complexity becomes $\Theta(P^4V^5 + P^5V^2)$ for both heuristics.

6. Numerical evaluation

In this section, the main objective of experiments is to comparatively evaluate the performance of *MCOPA*, *MCOPA—LOA* and CPLEX solutions under various parameter settings. In particular, we compare average energy consumption values and number of feasible outputs of two heuristics and CPLEX output. We implemented both heuristics in Java. ILP formulation in (1)–(20) is also implemented in Java with CPLEX Java library. As the problem size gets larger, CPLEX running times become too high and we have to change *epgap* parameter to a higher value. Default value of *epgap* is 0.0001 and it can take any value between 0.0 and 1.0. We set this parameter to 0.05 in our experiments.

In order to evaluate the performance of heuristics and compare them with the CPLEX output, it is required to generate input graphs and energy consumption values that are similar to the real life situation. We first generate input graphs and afterward, we set the link and node energy consumption values on input graphs.

6.1. Input graph generation

In our experiments, we basically make use of two topology generators. The first one is a random generator that implements Waxman's topology model [37,38]. The second topology generator we use is INET topology generator [39], which reflects the characteristics of the Internet more closely than the Waxman generator [27]. Moreover, INET provides an AS-level Internet topology, whereas Waxman provides a router level topology.

Waxman topologies have a pre-determined number N of nodes, which are uniformly distributed over an $n \times n$

grid, which implies that there are $N = n^2$ nodes and the domain size is $n \times n$. The probability of the existence of a link depends on parameters α and β, where a higher α indicates a higher number of links (a denser output graph) and a higher β indicates that the density of long links is higher than the density of the short links. In our experiments with Waxman model, we take $\beta = 0.2$ and α varying between 0.1 and 1.0. We change α to obtain dense or sparse graphs and observe the impact of graph density on the performance of our heuristic algorithms.

INET topologies are similar to the AS-level topology of the Internet. They take as input the number of nodes, which should not be less than 3037. This number represents the number of *AS*s on the Internet in November 1997. Another important parameter for INET is the fraction of degree-one nodes, which slightly affects the density of the graph.

After generating the graph structure, the second step is to determine the link and node energy consumption values. Total energy consumption of the network consists of the energy consumption of the nodes and links. In order to assign realistic energy consumption values to links and nodes, we adopt the power benchmarking model presented in [40]. We consider a system consisting of network switches, some of which can be put into sleep mode. Moreover, the links between switches are Ethernet cables and as in [41], they can be put into sleep mode if both ends of the link can be put into sleep mode.

Energy consumption caused by cables can be ignored since cables are not active network elements; physically removing the cables from network has little effect on total energy consumption [42]. Therefore, we ignore the impact of cables in terms of energy consumption. On the other hand, while disconnecting the cable, if the associated network ports are also put into sleep mode, total energy consumption might decrease. Hence, we assume that the energy cost of a link is associated with the network ports where the link is connected. In summary, the energy consumption of a link is simply the sum of the energy consumptions of the associated network ports.

As mentioned by [43,40], total power consumption of a switch depends on the power consumption of the chassis, linecards and active ports on the linecards. Depending on the type of the switch, ports on the linecards of the switch can be put into sleep mode. A switch might have several linecards which can be put into sleep mode individually. However, for the sake of simplicity, we assume in our simulations that individual linecards cannot be put into sleep mode. For this reason, for a switch that is powered on, we add the power consumption of all linecards to the chassis power consumption of the switch. The work in [43] makes the same assumption for both Rack switches and Tier 2 switches; i.e. they include the cost of all linecards while computing the power consumption of the switch. In other words, in our simulations, a linecard is put into sleep mode only if the switch is completely put into sleep mode. We use a simplified version of the switch configuration described in [43]. We assume that there are two types of switches. Type 1 switch has one linecard with 48 ports and 146 W of chassis power consumption including the consumption of the linecard. Type 2 switch has six linecards, each having 24 ports and 39 W of power consumption. We include the cost of linecards to the cost of the chassis of Type 2 switch, which is 54 W. Hence, the chassis power consumption of Type 2 switch becomes 288 W. In our simulations, if a node has 48 or less incident links, we set the node as Type 1. Otherwise, we set the node as Type 2 and assign energy consumption values accordingly. As in [43], we assume that the linespeed of a port can be 10 Mbps, 100 Mbps or 1 Gbps. When we generate the input graphs, these three possible values for the linespeeds of the ports are equally likely. Power consumption values of the ports having the same linespeed are different for Type 1 and Type 2 switches. Typically, ports of Type 2 switches have higher power consumption. Detailed description of power consumption values can be seen in Table 8.

Energy consumption of a link is associated with the power consumption of the ports at the ends of the links. Therefore, the energy consumption of a 10 Mbps link is 0.24 W and 0.84 W, if both ends are Type 1 and Type 2, respectively. Likewise, energy consumption of a 10 Mbps link with one end Type 1 and the other end Type 2 is 0.54 W. When a link is put into sleep mode, we assume that both ports are put into sleep mode. If all ports of a switch are in sleep mode, the switch is also in sleep mode. If a switch does not have a sleep mode, none of its ports can be put into sleep mode. Recall that our problem formulation takes these facts into consideration.

6.2. Simulation results

In all experiments, we consider the case where the relay cost of each node is equal to each other; i.e., all relay costs are equal to one. Since there is one relay node for each communication pair, *r*, which is the upper bound (threshold value) for the total relay cost, equals *P*, which is the total number of communication pairs in our experiments. Moreover, we set the underlay path for each communication pair as the path with minimum number of hops.

In our first set of experiments, we execute our heuristic algorithms and the CPLEX implementations of our ILP formulation with 100 randomly generated input graphs that are based on the Waxman topology. We then comparatively evaluate their performance in terms of average energy consumption and number of feasible solutions. We evaluate the impact of the following parameters: α, η, probability of having sleep mode, number of pairs, and domain size.

To begin with, the parameter α, which takes values in the range $(0, 1]$, is the link density parameter that linearly affects the probability of the existence of a link between two arbitrary nodes in Waxman topology. Therefore, a

Table 8
Power consumption of chassis and port types used in our experiments.

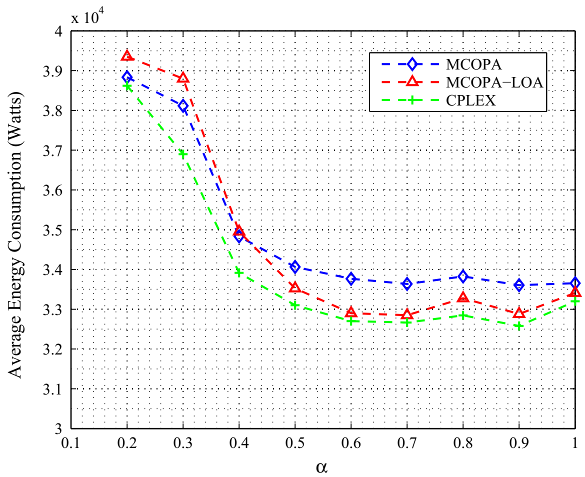
Configuration	Type 1 (in W)	Type 2 (in W)
$Power_{chassis}$	146	288
$Power_{10Mbps\ port}$	0.12	0.42
$Power_{100Mbps\ port}$	0.18	0.48
$Power_{1Gbps\ port}$	0.87	0.9

higher α implies an input graph with higher density. In order to evaluate the impact of k_p, we define the parameter $\eta = k_p/Length(u_p)$, where u_p is the underlay path for pair p. η takes values in the range $[0, 1]$ and represents the proportion of links on an overlay path that are allowed to be shared with underlay paths and other overlay paths. This way, it basically helps us to determine the upper limit k_p value for a pair p. While giving a k_p value as an input to the algorithms, it is more realistic that k_p takes a value proportional to the length of the underlay path of the pair p rather than being constant for all pairs. The relationship between η and k_p is as follows: $k_p = \eta \times Length(u_p)$, where u_p is the underlay path for pair p. Probability of having a sleep mode is a value in the $[0, 1]$ range and related only to the nodes. If a node has a sleep mode, it can be put into sleep mode if none of its ports are currently used; i.e., all of its ports are inactive. Number of pairs is a parameter that might make it difficult to find a feasible solution. It might also cause the average energy consumption to increase. Table 9 shows the details of the experimental setup related to Waxman topology generator. In our experiments, in order to test the effect of a parameter, as the parameter under consideration takes values in the range specified in Table 9, we set the other parameters to their middle values. This way, we make sure that only the parameter under consideration has an effect on test results. The middle values for α, η, and probability of sleep mode are 0.5, while the middle values for the number of pairs and domain size are 50 and 25, respectively. For example, when we test the effect of α, we set η and probability of sleep mode to 0.5 and run the tests with 50 pairs on a graph that is produced by Waxman generator on a 25 by 25 domain size. The same logic applies when we test other parameters.

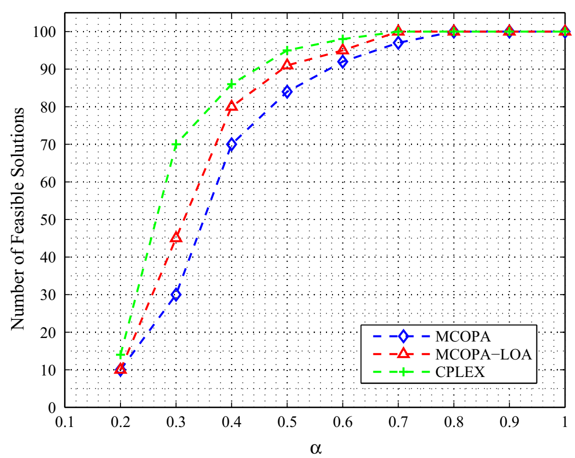
Table 9
Parameter values used in experiments with Waxman topology.

Parameter	Value range
α	$\{0.1, 0.2, \ldots, 1.0\}$
η	$\{0.1, 0.2, \ldots, 1.0\}$
Probability of sleep mode	$\{0.1, 0.2, \ldots, 1.0\}$
Number of pairs	$\{10, 20, 30, \ldots, 90, 100\}$
Domain size	$\{10 \times 10, 20 \times 20, 30 \times 30, 40 \times 40, 50 \times 50\}$

We illustrate in Fig. 1 the results of the experiments that are related to the first parameter in Table 9. Recall that α is a parameter that directly affects the density of the input graph; hence, a higher α might help the algorithms to find more feasible solutions. We define the overlay graph as the 2-tuple consisting of the set of links on overlay paths and the set of nodes on overlay paths. Nodes and links that cannot be put into sleep mode are also included in these sets since they contribute to the energy consumption irrespective of whether they are on the overlay paths. In fact, the energy consumption of the overlay graph that we just defined corresponds to the objective function in our ILP formulation. We compare in Fig. 1a the performance of our heuristics and CPLEX in terms of the average energy consumption of the resulting overlay graph. As expected, the average energy consumption becomes lower as α gets closer to 1.0. Since a higher α value causes a denser input graph, it becomes easier to find a shorter alternative path, leading to more feasible solutions on the average. This situation causes the total energy consumption of the overlay graph to decrease. The energy consumption does not decrease any more after $\alpha = 0.7$. The reason for this behavior is as follows: the input graph becomes dense enough after $\alpha = 0.5$; hence, adding excessive links does not change the length or the total energy consumption of the overlay paths. On the other hand, Fig. 1b shows the comparison of our heuristics and *CPLEX* solutions in terms of the number of feasible solutions. As expected, *CPLEX* always gives the highest number of feasible solutions. Besides, *MCOPA–LOA* always finds more feasible solutions than *MCOPA* until $\alpha = 0.7$. Starting from $\alpha = 0.7$, two heuristics find the same number of feasible solutions. Therefore, *MCOPA–LOA* has closer performance to *CPLEX*



(a) Average energy consumption with varying α



(b) Number of feasible solutions with varying α

Fig. 1. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying α with Waxman topology.

compared to *MCOPA* in terms of satisfying the ILP constraints.

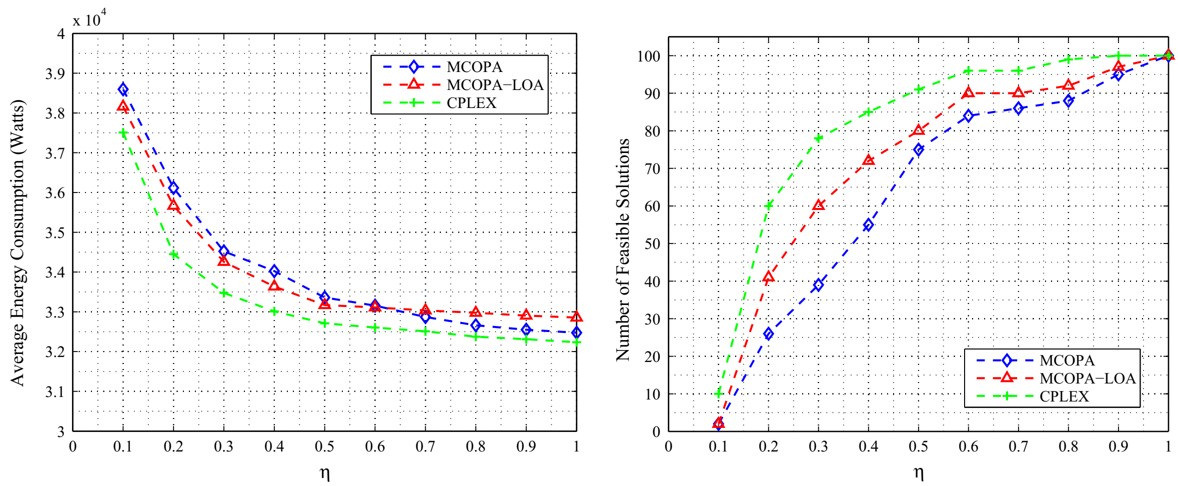
We investigate in Fig. 2 the effect of η. As we mentioned before, η determines the k_p value for a pair p. A lower value of η means that the overlay paths have less overlapping edges with the underlay and other overlay paths. As η increases, there is a nearly quadratic decrease in the average energy consumption. This decrease is more apparent on the curve belonging to *CPLEX* output. This result can be explained as follows: As η increases, k_p values also increase, allowing more shared links between overlay and underlay paths. Therefore, the produced overlay paths are shorter; hence, the energy consumption of the overlay graph is smaller. As for the number of feasible solutions, *MCOPA—LOA* always finds more feasible solutions than *MCOPA*, which can be seen in Fig. 2b. Also in Fig. 2b, as η gets higher, the number of feasible solutions increases. Because higher η values lead to higher k_p values, higher number of shared edges does not create constraint violations. Therefore, the number of feasible solutions found by heuristics and CPLEX increases.

The ratio of nodes that can be put into sleep mode is also an important parameter that has impact on the average energy consumption of the resulting overlay graphs. We investigate in Fig. 3a the average energy consumption resulting from two heuristics and *CPLEX* solutions. We see that having more nodes that have sleep mode causes a linear decrease on the average energy consumption. This behavior is expected since a higher ratio of nodes having sleep mode means saving energy from more nodes and links that are not being used. In Fig. 3a, *MCOPA* and *MCOPA—LOA* give akin average energy consumption; however, the average energy consumption of *MCOPA* is slightly lower than *MCOPA—LOA*. The reason for this difference is that *MCOPA—LOA* prunes in path selection phase the nodes having sleep mode, whereas *MCOPA—LOA* does not prune these nodes and construct paths by selecting nodes regardless of their capability of being put in sleep mode.

Fig. 3b shows the number of feasible solutions with varying probability of sleep mode. This figure shows us that the ratio of nodes having sleep mode does not have much impact on the number of feasible solutions found by *CPLEX* and *MCOPA—LOA*. However, *MCOPA* has a decreasing trend in terms of the number of feasible solutions. Since *MCOPA* prunes the nodes having sleep mode, it becomes more difficult to find alternative paths for an overlay path having constraint violation. This behavior causes less feasible solutions for *MCOPA*.

We show in Fig. 4a and b the impact of varying the number of pairs while the graph size and density stay constant. Number of pairs has a nearly linear effect on heuristics and CPLEX results in terms of average energy consumption. However, the gap between *MCOPA* and *MCOPA—LOA* increases as the number of pairs becomes higher. For lower values, we see that *MCOPA—LOA* results in less energy consumption than *MCOPA*. Both algorithms perform close to *CPLEX* solutions in terms of average energy consumption. Fig. 4b shows the performance in terms of the number of feasible solutions. *MCOPA—LOA* mostly leads to higher number of feasible solutions. Nevertheless, as the number of pairs increases, the number of feasible solutions produced by both heuristics become very close to each other. This behavior is caused by the difficulty of finding disjoint overlay paths for higher number of source and destination pairs.

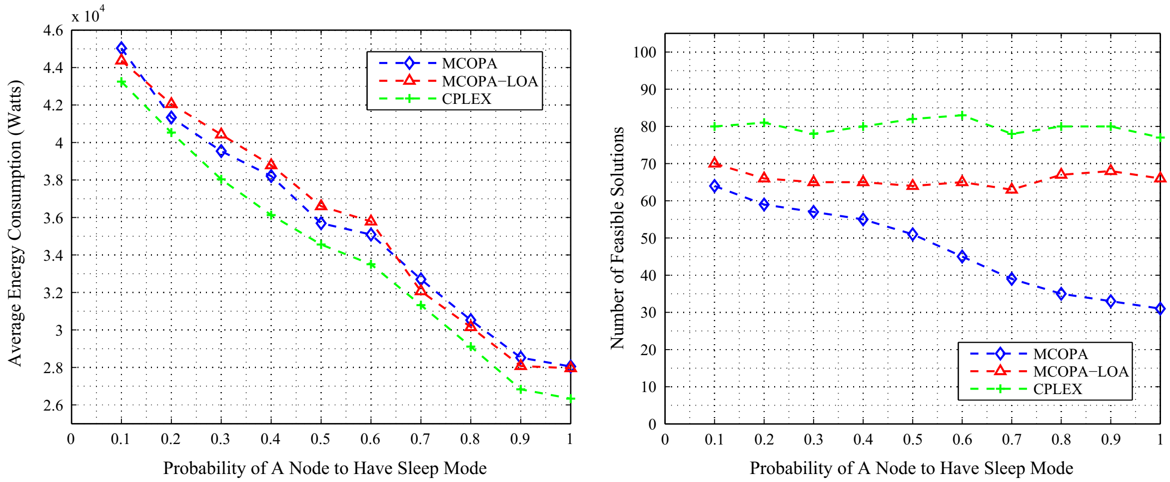
Fig. 5 displays the effect of the domain size while all other parameters stay constant. The domain size for Waxman topology is related to the size of the input graph. Fig. 5a shows that the average energy consumption increases quadratically as the domain size increases. As the network gets larger, the hop count of the paths between the source and destination pairs also gets larger. Therefore, the average energy consumption increases. Furthermore, as the number of nodes in the network gets larger, the number of nodes that cannot be put into sleep mode also gets larger. Hence, this increase also contributes



(a) Average energy consumption with varying η

(b) Number of feasible solutions with varying η

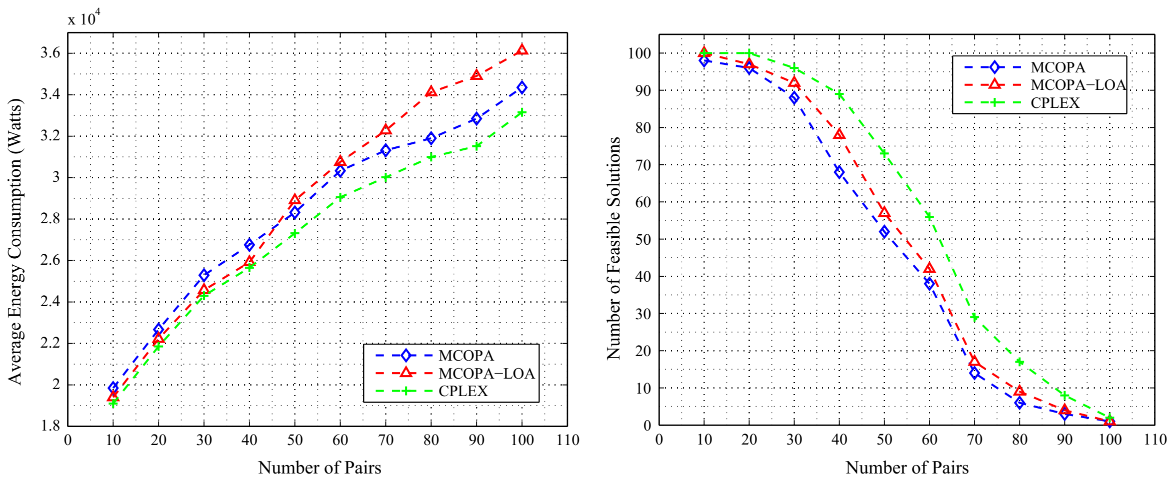
Fig. 2. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying η values with Waxman topology.



(a) Average energy consumption with varying sleep mode probability

(b) Number of feasible solutions with varying sleep mode probability

Fig. 3. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying sleep mode probability with Waxman topology.



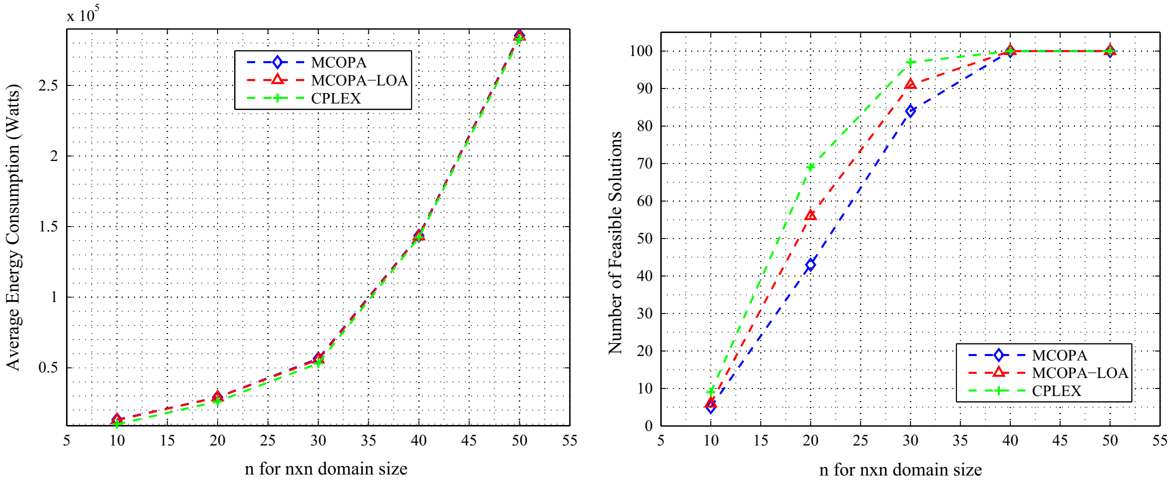
(a) Average energy consumption with varying number of pairs

(b) Number of feasible solutions with varying number of pairs

Fig. 4. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying number of pairs with Waxman topology.

to the increase in the average energy consumption. Again, *MCOPA—LOA* has better performance than *MCOPA* in terms of finding feasible solutions.

In the second set of experiments, we evaluate the performance of our heuristics on the input graphs that are generated by INET topology generator. We again run our experiments on 100 randomly generated input graphs. Note that the most basic difference between INET and Waxman generators is that Waxman generates router level topologies while INET generates AS-level topologies. Furthermore, while it is possible to produce dense graphs with Waxman, INET generates proportionally sparse graphs with nearly $2N$ links for N nodes given by the user. While generating graphs with INET, we give the number of nodes, the fraction of degree one nodes and the seed number as input. We choose the fraction of degree one nodes to be 0.1 because we aim not to generate very sparse graphs since observing the actual performance of our heuristics would be extremely difficult in such a situation. As in Waxman experiments, we compare the average energy consumption and number of feasible solutions of our heuristic algorithms and CPLEX solutions. We evaluate the impact of the following parameters: η, probability of having sleep mode, number of pairs, and number of nodes. Note that the only parameter that is different from the ones in Waxman experiments is the number of nodes in the input graph. INET enables user control on the number of nodes while in Waxman we cannot give this parameter as an input; i.e., we cannot predetermine the number of nodes in Waxman generator. Table 10 shows the details



(a) Average energy consumption with varying domain size

(b) Number of feasible solutions with varying domain size

Fig. 5. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying graph domain size with Waxman topology.

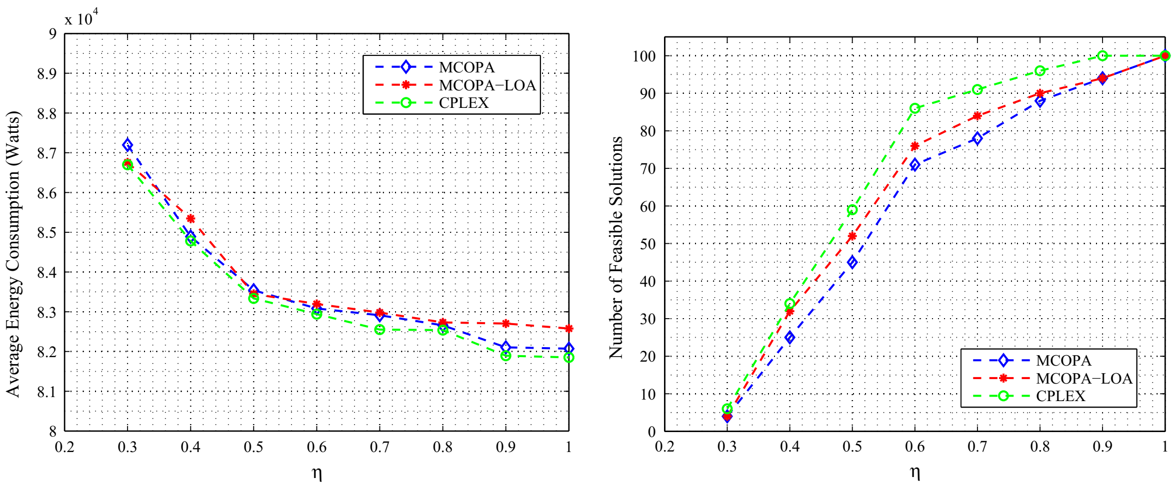
Table 10
Parameter values used in experiments with INET topology generator.

Parameter	Value range
η	$\{0.1, 0.2, \ldots, 1.0\}$
Probability of sleep mode	$\{0.1, 0.2, \ldots, 1.0\}$
Number of pairs	$\{50, 60, 70, \ldots, 130, 140\}$
Number of nodes	$\{3037, 3500, 4000, \ldots, 6000\}$

of the experimental setup related to the INET topology generator.

Fig. 6 displays the effect of η when we run heuristics and CPLEX on INET generated graphs with 4500 nodes and 100 pairs. We see in Fig. 6a and b that changing η has the same effect as in Waxman experiments. When η increases, we have more tolerance for overlapping edges and hence, an overlay path does not need to prefer a longer path to avoid sharing links with other paths. This situation enables the overlay paths to be shorter and explains the decrease in the average energy consumption in Fig. 6a. Furthermore, there is an important difference between INET results and Waxman results. In Waxman experiments with varying η, we have obtained feasible solutions after η is 0.1, while in INET experiments we start to obtain feasible solutions after η is 0.3. This behavior can be explained by the sparsity of INET generated graphs. Moreover, Inet's hierarchical tree structure causes the number of shared edges to be higher compared to Waxman topology. When the number of shares edges is higher, neither of the algorithms can find feasible solutions in case of lower η values.

Fig. 7 displays the impact of the probability of having sleep mode on the performance of our heuristics and CPLEX solutions. Fig. 7a shows that the average energy consumption decreases as the probability of sleep mode



(a) Average energy consumption with varying η

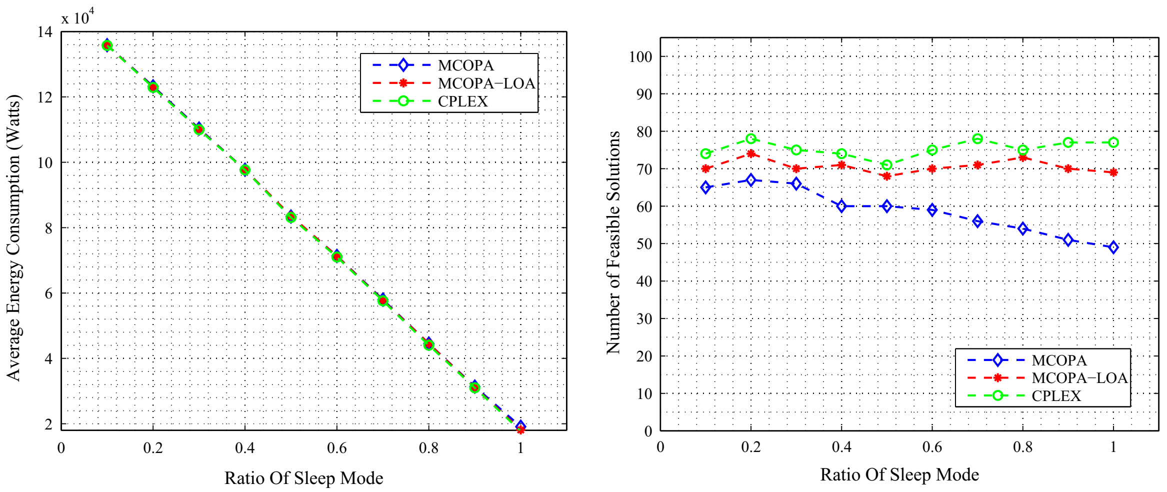
(b) Number of feasible solutions with varying η

Fig. 6. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying η values with Inet topology generator.

increases because higher number of nodes can save from power by means of the sleep state. Fig. 7B displays the number of feasible solutions with varying sleep mode probability. This probability does not have a significant impact except that it causes a decrease in *MCOPA* starting from 0.3. The reason for this behavior is the pruning of the nodes having sleep mode at the beginning steps of *MCOPA*.

Fig. 8 shows the impact of increasing number of communication pairs on INET generated networks. We vary the number of pairs between 50 and 140. Fig. 8a shows that average energy consumption linearly increases when the number of pairs increases. The gap between *MCOPA* and *MCOPA–LOA* increases as the number of pairs increases because *MCOPA–LOA* aims to find paths as disjoint as possible and this behavior increases the path lengths and average energy consumption. Fig. 8a shows the number of feasible solutions with varying number of pairs. In general, *MCOPA–LOA* gives higher number of feasible solutions than *MCOPA*. However, the gap between two heuristics decreases with higher number of pairs because finding disjoint paths becomes more difficult as the number of pairs increases.

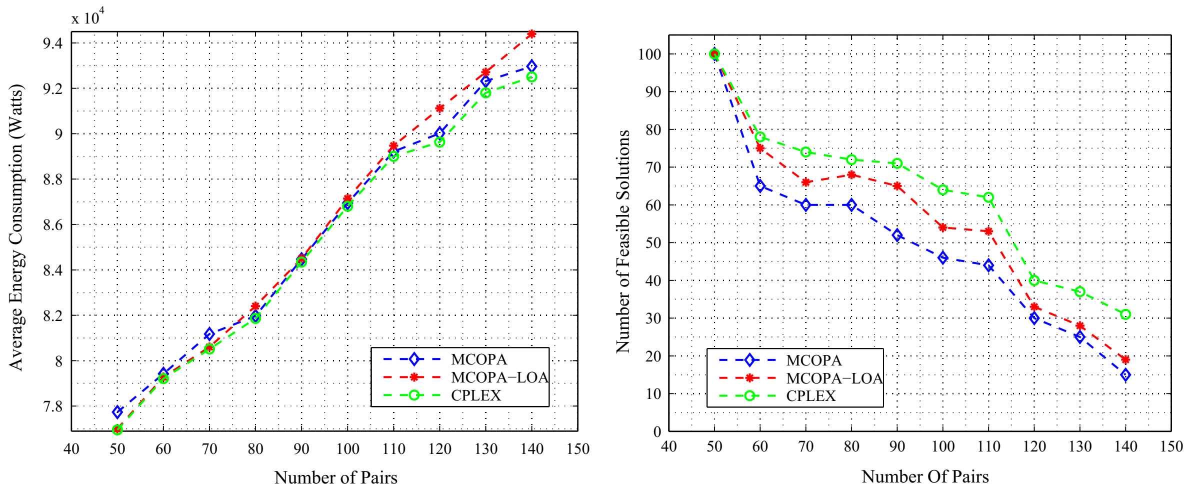
Unlike Waxman, the number of nodes is under user's control in INET topology generator. Fig. 9 displays the impact of increasing number of nodes while all other parameters are constant. Recall that INET topology generator runs when the number of nodes is greater than or equal to 3037. Starting from 3037, we increase the number of nodes until 6000. Fig. 9a shows that average energy consumption increases linearly as the number of nodes increases. The reason is the following: INET places the



(a) Average energy consumption with varying sleep mode probability

(b) Number of feasible solutions with varying sleep mode probability

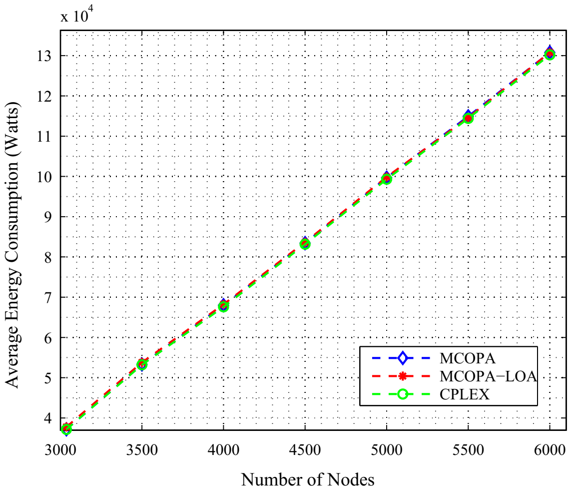
Fig. 7. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying sleep mode probability with Inet topology generator.



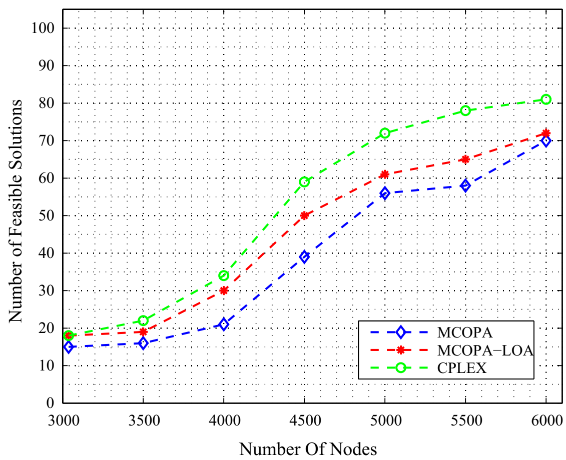
(a) Average energy consumption with varying number of pairs

(b) Number of feasible solutions with varying number of pairs

Fig. 8. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying number of pairs with Inet topology.



(a) Average energy consumption with varying number of nodes



(b) Number of feasible solutions with varying number of nodes

Fig. 9. Comparison of MCOPA, MCOPA–LOA and CPLEX outputs for varying number of nodes with Inet topology.

nodes on a 10,000 by 10,000 plane, which has the default size. When the number of nodes increases, network density does not change as long as the fraction of degree one nodes does not change. Instead, the trees in the tree structure of the INET generated network enlarge and this situation increases the probability of source and destination nodes to be further away from each other. This behavior causes the paths to become longer, which explains the linear increase in the average energy consumption. There is a fast increase in the number of feasible solutions until some point where it gets slower. The initial fast increase is due to the fact that as the number of nodes increases, the number of shared links tends to get lower. However, after some point the rate of increase decreases. Here, again the reason is that adding extra nodes to the graph using INET does not make the graph denser; it only enlarges the tree structure of the graph. The hierarchical tree structure of INET topology is responsible for this behavior. The performance gap between heuristics gets smaller with higher number of nodes because the number of pairs is constant and as the number of nodes increases, the number of shared links tends to get lower. Therefore, heuristics perform better and more closely.

To sum up, *MCOPA—LOA* performs in general better than *MCOPA* in terms of finding feasible solutions. However, the same observation is not true when we compare them in terms of average energy consumption; in some cases, *MCOPA* gives lower energy consumption than *MCOPA – LOA*. For instance when the input graph is sparse, the number of pairs is very high or η is very high, i.e., k_p values are high, *MCOPA* yields lower average energy consumption than *MCOPA—LOA*. Throughout our experiments we have seen that both heuristics perform very close to *CPLEX* in terms of average energy consumption. We have also observed that *MCOPA—LOA* yields more feasible solutions than *MCOPA*. Furthermore, both heuristics have low computational complexity, which makes them suitable for finding energy efficient overlay paths. Note also that the centralized nature of our proposed heuristic algorithms is in line with the recently emerging software defined networking paradigm [44], where data and control plane are separated such that decisions such as routing can be made in a logically centralized manner.

7. Conclusion

Overlay routing is an important concept for wired networks since it provides a more reliable routing mechanism. It supports and maintains the connection between source and destination pairs by finding alternative paths and relay nodes for each pair. Energy efficiency of the overlay network is as crucial as the energy efficiency of the underlying routing scheme. To the best of our knowledge, this study is the first in the literature that considers both energy efficiency and relay node selection on overlay networks.

In this study, we have investigated overlay routing on wired networks in terms of energy efficiency and relay selection. We have formulated an optimization problem called *JORRA* (Joint Overlay Routing and Relay Assignment) as an integer linear program, where the goal is to minimize the energy consumption. We have implemented our proposed formulation by using the optimization software *CPLEX*. Moreover, we have proved that *JORRA* is APX-Hard in addition to being NP-Hard in the strong sense even in its special cases. For this reason, we have designed two computationally efficient heuristic algorithms, namely *MCOPA* and *MCOPA—LOA*. *MCOPA—LOA* is an improved version of *MCOPA* and has a higher computational complexity. We have made experiments by using Internet like network topologies and demonstrated that our proposed algorithms provide very close performance to the *CPLEX* solutions. Furthermore, we have observed that *MCOPA—LOA* finds more feasible solutions than *MCOPA*.

As a future work, we plan to add extra constraints to our integer linear programming formulation such as ensuring that each link has a specific upper limit for the number of overlay paths that can use it or each link has a capacity and the total traffic demand of source and destination pairs using a particular link should not exceed its capacity. This way, heterogeneity in the reliability aspects of different links can be addressed. We also plan to propose a distributed algorithm for a distributed environment where a centralized server having all information about energy consumption values and underlay paths does not exist. In such a scenario, we plan to consider the case where there are some specialized servers in the network having local information as well as the situation with an entirely distributed environment where each node possesses information about its neighboring nodes.

References

[1] S. Ricciardi, F. Palmieri, U. Fiore, D. Careglio, G. Santos-Boada, J. Solé-Pareta, An energy-aware dynamic RWA framework for next-generation wavelength-routed networks, Comput. Networks 56 (10) (2012) 2420–2442.

[2] G. Fettweis, E. Zimmermann, ICT energy consumption-trends and challenges, in: International Symposium on Wireless Personal Multimedia Communications, 2008.

[3] M. Gupta, S. Singh, Greening of the internet, in: ACM SIGCOMM, 2003, pp. 19–26.

[4] W. Fisher, M. Suchara, J. Rexford, Greening backbone networks: reducing energy consumption by shutting off cables in bundled links, in: ACM SIGCOMM Workshop on Green Networking, 2010, pp. 29–34.

[5] M. Cha, S. Moon, C. Park, A. Shaikh, Placing relay nodes for intra-domain path diversity, in: IEEE International Conference on Computer Communications (INFOCOM), vol. 1, 2006, pp. 1–12.

[6] M. Kodialam, T. Lakshman, S. Sengupta, Efficient and robust routing of highly variable traffic, in: Workshop on Hot Topics in Networks (HotNets-III), 2004.

[7] R. Cohen, D. Raz, Cost effective resource allocation of overlay routing relay nodes, in: IEEE International Conference on Computer Communications (INFOCOM), 2011, pp. 3236–3244.

[8] K. Christensen, B. Nordman, Reducing the energy consumption of networked devices, IEEE 802.3 Tutorial.

[9] E. Goma, M. Canini, A. Lopez Toledo, N. Laoutaris, D. Kostić, P. Rodriguez, R. Stanojević, P. Yagüe Valentin, Insomnia in the access: or how to curb access network related energy consumption, in: ACM SIGCOMM, 2011, pp. 338–349.

[10] L. Chiaraviglio, M. Mellia, F. Neri, Reducing power consumption in backbone networks, in: IEEE International Conference on Communications (ICC), 2009, pp. 1–6.

[11] A. Bianzino, C. Chaudet, F. Larroca, D. Rossi, J. Rougier, Energy-aware routing: a reality check, in: IEEE GLOBECOM Workshops, 2010, pp. 1422–1427.

[12] F. Giroire, D. Mazauric, J. Moulierac, B. Onfroy, Minimizing routing energy consumption: from theoretical to practical results, in: IEEE International Conference on Green Computing and Communications (GreenCom), 2010, pp. 252–259.

[13] L. Chiaraviglio, I. Matta, Greencoop: cooperative green routing with energy-efficient servers, in: International Conference on Energy-Efficient Computing and Networking, 2010, pp. 191–194.

[14] N. Maxemchuk, Dispersity routing, in: IEEE International Conference on Communications (ICC), vol. 75, 1975, pp. 41–10.

[15] S. Savage, A. Collins, E. Hoffman, J. Snell, T. Anderson, The end-to-end effects of internet path selection, in: ACM SIGCOMM Computer Communication Review, vol. 29, 1999, pp. 289–299.

[16] N. Feamster, D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, Measuring the effects of internet path faults on reactive routing, in: ACM SIGMETRICS Performance Evaluation Review, vol. 31, 2003, pp. 126–137.

[17] S. Patek, R. Venkateswaran, J. Liebeherr, Enhancing aggregate QoS through alternate routing, in: IEEE Global Telecommunications Conference (GLOBECOM), vol. 1, 2000, pp. 611–615.

[18] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, et al., Detour: informed internet routing and transport, IEEE Micro 19 (1) (1999) 50–59.

[19] L. Subramanian, I. Stoica, H. Balakrishnan, R. Katz, OverQoS: offering internet QoS using overlays, vol. 33, 2003, pp. 11–16.

[20] A.P. Bianzino, C. Chaudet, D. Rossi, J.-L. Rougier, A survey of green networking research, IEEE Commun. Surv. Tutorials 14 (1) (2012) 3–20.

[21] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, D. Wetherall, Reducing network energy consumption via sleeping and rate adaptation, in: USENIX Symposium on Networked System Design and Implementation (NSDI), vol. 8, 2008, pp. 323–336.

[22] A.P. Bianzino, C. Chaudet, S. Moretti, J.-L. Rougier, L. Chiaraviglio, E. Le Rouzic, Enabling sleep mode in backbone IP-networks: a criticality-driven tradeoff, in: EEE International Conference on Communications (ICC), 2012, pp. 5946–5950.

[23] A.P. Bianzino, L. Chiaraviglio, M. Mellia, J.-L. Rougier, Grida: green distributed algorithm for energy-efficient IP backbone networks, Comput. Networks 56 (14) (2012) 3219–3232.

[24] H. Matsuura, Energy-saving routing algorithm using Steiner tree, in: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013, pp. 378–386.

[25] B. Addis, A. Capone, G. Carello, L. Gianoli, B. Sanso, Energy management through optimized routing and device powering for greener communication networks, vol. PP, 2013, pp. 1–1.

[26] R.G. Garroppo, S. Giordano, G. Nencioni, M.G. Scutella, Mixed integer non-linear programming models for green network design, Comput. Oper. Res. 40 (1) (2013) 273–281.

[27] S. Roy, H. Pucha, Z. Zhang, Y. Hu, L. Qiu, On the placement of infrastructure overlay nodes, IEEE/ACM Trans. Network. 17 (4) (2009) 1298–1311.

[28] V.V. Vazirani, Approximation Algorithms, Springer, 2001.

[29] C.-L. Li, S.T. McCormick, D. Simchi-Levi, The point-to-point delivery and connection problems: complexity and algorithms, Discr. Appl. Math. 36 (3) (1992) 267–292.

[30] G. Robins, A. Zelikovsky, Improved Steiner tree approximation in graphs, in: ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 770–779.

[31] M. Bern, P. Plassmann, The Steiner problem with edge lengths 1 and 2, Inform. Process. Lett. 32 (4) (1989) 171–176.

[32] M.X. Goemans, D.P. Williamson, A general approximation technique for constrained forest problems, SIAM J. Comput. 24 (2) (1995) 296–317.

[33] S. Even, A. Itai, A. Shamir, On the complexity of time table and multi-commodity flow problems, in: IEEE Annual Symposium on Foundations of Computer Science, 1975, pp. 184–193.

[34] M. Kramer, J. van Leeuwen, The complexity of wire-routing and finding minimum area layouts for arbitrary vlsi circuits, Adv. Comput. Res. 2 (1984) 129–146.

[35] J.Y. Yen, Finding the k shortest loopless paths in a network, Manage. Sci., Theory Ser. 17 (1971) 712–716.

[36] E. Martins, M. Pascoal, A new implementation of Yen's ranking loopless paths algorithm, Quart. J. Belg., Fr. Ital. Oper. Res. Soc. 1 (2) (2003) 121–133.

[37] I. Kaj, R. Gaigalas, Waxman Random Network Topology Generator. <http://www2.math.uu.se/research/telecom/software/stgraphs.html>.

[38] B. Waxman, Routing of multipoint connections, IEEE J. Sel. Areas Commun. 6 (9) (1988) 1617–1622.

[39] C. Jin, Q. Chen, S. Jamin, Inet: Internet Topology Generator. <http://topology.eecs.umich.edu/inet>.

[40] P. Mahadevan, P. Sharma, S. Banerjee, P. Ranganathan, A power benchmarking framework for network devices, in: Proceedings of the 8th International IFIP-TC 6 Networking Conference, NETWORKING '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 795–808.

[41] S. Ricciardi, D. Careglio, U. Fiore, F. Palmieri, G. Santos-Boada, J. Solé-Pareta, Analyzing local strategies for energy-efficient networking, in: Proceedings of the IFIP TC 6th International Conference on Networking, NETWORKING'11, 2011, pp. 291–300.

[42] R. Sohan, A. Rice, A.W. Moore, K. Mansley, Characterizing 10 Gbps network interface energy consumption, in: IEEE Conference on Local Computer Networks (LCN), 2010, pp. 268–271.

[43] P. Mahadevan, P. Sharma, S. Banerjee, P. Ranganathan, Energy aware network operations, in: Proceedings of the 28th IEEE International Conference on Computer Communications

Workshops, INFOCOM'09, IEEE Press, Piscataway, NJ, USA, pp. 25–30.
[44] F. de Oliveira Silva, J.H. de Souza Pereira, P.F. Rosa, S.T. Kofuji, Enabling future internet architecture research and experimentation by using software defined networking, in: IEEE European Workshop on Software Defined Networking (EWSDN), 2012, pp. 73–78.

Fatma Ekici received the B.S. degree (honors) and M.S. degree in computer engineering from Bogazici University, Istanbul, Turkey, in 2009 and 2014, respectively. She worked as a software engineer in Nortel Netas from 2009 to 2011. Since 2011, she is a researcher in the Sensor and Radar Division of the Scientific and Technological Research Council of Turkey (TUBITAK). Her main research interests are wireless networks, optimization, graph theory, multi-radar target tracking and data fusion.

Didem Gözüpek received the B.S. degree (high honors) in telecommunications engineering from Sabanci University, Istanbul, Turkey, in 2004, the M.S. degree in electrical engineering from the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 2005, and the Ph.D. degree in computer engineering from Bogazici University, Istanbul, Turkey, in 2012. She is an Assistant Professor with the Computer Engineering Department, Gebze Technical University, Kocaeli, Turkey. From 2005 to 2008, she worked as an R&D Engineer in a telecommunications company in Istanbul. Her main research interests are scheduling and resource allocation in communication networks, algorithmic graph theory, and approximation algorithms. Dr. Gözüpek received the CAREER Award from the Scientific and Technological Research Council of Turkey (TUBITAK) in 2014, the Dr. Serhat Özyar Young Scientist of the Year Honorary Award in 2013, and the Bogazici University Ph.D. Thesis Award in 2012. She was a finalist for the Google Anita Borg Memorial Scholarship in 2009.