Abstract—As devices and traffic in the Internet increase, power consumption becomes a critical issue for Information and Communication Technology (ICT) industry. The emerging technology, Software Defined Networking (SDN), can adaptively manage the network resources and control the power consumption. In this paper, we first investigate the time efficient energy aware routing in SDN enabled networks, and then propose a Time Efficient Energy Aware Routing (TEAR) algorithm. The goal of this work is to effectively reduce the number of used links for packet delivery so that the energy consumption can be decreased. Different from most existing schemes which only focus on static demands, the proposed TEAR algorithm considers both the temporal variation in demand and computational efficiency when finding routing paths. Specifically, the energy consumption problem for routing traffic in a network is formulated as an integer linear programming problem. Hence, the proposed TEAR scheme is capable of changing routes for flows dynamically based on the conditions of the network. The authors in [5] present an Optimizing Rule Placement for Energy-Aware Routing (ORPEAR) in SDN. ORPEAR determines traffic routes to minimize the number of used links by using SDN, but it suffers from high computational cost, which affects network performance dramatically in SDN, when demands join and leave dynamically, because packets are queued in routers to wait for the decision making of the controller.

To address the computation cost issue, in this paper, TEAR scheme is proposed to reduce power consumption in the network using SDN architecture while decreasing the computation time and considering rule (TCAM space) and link (bandwidth) capacities. Many studies in the literature only consider static demands and do not pay attention to the dynamic properties of flows coming and leaving over time. Adopting the dynamic routing capabilities of SDN, the proposed scheme can assign energy-efficient routes to new demands quickly to satisfy the requirement of time efficiency in practice. To better show the performance of the proposed scheme, real topologies and traffic logs in SNDlib [6] are used in the simulations. The results show that the proposed scheme outperforms the other schemes in terms of computation time and energy consumption of the network.

I. INTRODUCTION

Applications built on networks have revolutionized our everyday life. The increasing number of network applications generates considerable communication traffic in the network and also consumes quite a lot of energy. According to previous studies, the power consumption of ICT industry accounts for 2 to 10% worldwide power consumption [1] and is continuing to grow in the future. In Japan, ICT equipment is responsible for 4% of total electricity consumption, which is 1% of the total energy consumption of Japan [2].

Consequently, reducing energy consumption in ICT industry is an urgent issue. Previous studies show that network equipment such as routers and line cards are the major energy consuming devices [3], [4]. In practice, the number of unused links can be used as an indicator for energy consumption. Although turning off unused links or routers could reduce energy consumption, frequently turning on or off a router will shorten the life cycle of the router [5]. Therefore, to minimize network operating costs, it is crucial to reduce the energy consumption by reducing the number of used links while decreasing the opportunities of switching on or off the routers. However, in traditional networking, it is difficult to control traffic flexibly to achieve green traffic routing and maximize the number of unused links.

Software Defined Networking (SDN) is a new network architecture, which moves the functions of the control plane in a router into a central controller, and leaves the data plane in the router for packet forwarding. The controller can globally monitor flows of the whole network, and install or remove rules in the router in order to adaptively utilize available network resources. The flows are forwarded to the corresponding ports based on the matching rules in the routers. However, the Ternary Content Addressable Memory (TCAM) space is more restrictive than traditional routers because of more matching fields in OpenFlow. Thus, the TCAM space should be used more carefully. If a flow does not match any rule in the routers, the flow will be forwarded to the controller for further process instead of being dropped. Therefore, SDN is capable of changing routes for flows dynamically based on the conditions of the network. The authors in [5] present an Optimizing Rule Placement for Energy-Aware Routing (ORPEAR) in SDN. ORPEAR determines traffic routes to minimize the number of used links by using SDN, but it suffers from high computational cost, which affects network performance dramatically in SDN, when demands join and leave dynamically, because packets are queued in routers to wait for the decision making of the controller.

To address the computation cost issue, in this paper, TEAR scheme is proposed to reduce power consumption in the network using SDN architecture while decreasing the computation time and considering rule (TCAM space) and link (bandwidth) capacities. Many studies in the literature only consider static demands and do not pay attention to the dynamic properties of flows coming and leaving over time. Adopting the dynamic routing capabilities of SDN, the proposed scheme can assign energy-efficient routes to new demands quickly to satisfy the requirement of time efficiency in practice. To better show the performance of the proposed scheme, real topologies and traffic logs in SNDlib [6] are used in the simulations. The results show that the proposed scheme outperforms the other schemes in terms of computation time and energy consumption of the network. The contributions of this paper are summarized as follows:

- An integer linear programming problem is formulated to reduce the number of used links. It provides the theoretical foundation of the investigated problem.
- An algorithm, which considers dynamic traffic demands over time, is proposed to resolve the energy aware routing problem. The proposed algorithm is shown to
be much more time efficient than other schemes.

- Real topologies and traffic logs are used in the simulations to evaluate the performance of the proposed scheme. Energy consumption of the network can be reduced effectively by using the selected routes.

The paper is organized as follows. Section II shows the existing studies for energy-aware routing and SDN. Section III presents the proposed energy aware routing in software defined networking. Section IV evaluates the performance of the scheme by simulations. Section V describes some challenges of TEAR and the solutions for future work. The paper concludes in Section VI.

II. RELATED WORK

The authors in [7] point out that the network equipment account for 4.75% of the total energy consumption in India and provide a good motivation to make the network equipment more efficient. In [8], a novel algorithm is proposed to dynamically turn on/off the active network elements. Moreover, it consists of three models which are the optimizer model, the power control model and the routing model. [9] proposes a green network-aware VMs placement mechanism to minimize the required hardware to maximize the allocated VMs satisfying the resource requirements. Thus the energy consumption can reduced by using less hardware resource. Instead of focusing on the network equipment, how to route traffic efficiently is also an important aspect of energy saving.

Energy-aware routing has been studied by many researchers in recent years. The authors in [10] propose a heuristic routing algorithm to shut down idle network devices based on the static traffic matrix. [11] proposes a novel model to reduce power consumption by scheduling and routing deadline-constrained flows. [12] previously schedules the flow and maximizes the bandwidth utilization of links in order to reduce the energy consumption. Nevertheless, the above studies only focus on the static traffic by assigning the route of flows previously and just study on the traditional networking.

The network management can be accomplished conveniently and efficiently by configuring the applications on the controller in SDN. The authors in [13] propose an algorithm to detect and locate the links which are out of order and maintain the connectivity of network in SDN. [14] uses OpenDaylight controller to achieve load balancing and proposes an algorithm to fully utilize the distributed data storage. The authors in [15] jointly consider routing engineering and rule placement in order to use TCAM resources efficiently in SDN.

For energy saving, SDN is also widely used by many researchers to manage the network applications. [16] proposes a framework for energy-efficient routing algorithms to reduce the energy consumption of TCAM. The authors in [17] propose Hybrid Energy-Aware Traffic Engineering (HEATE) algorithm to turn off the underutilized links for SDN/IP hybrid environment. [18] presents a novel scheme to control the route of each flow based on the difference of network devices’ power-profile and uses OpenFlow protocol as well as Network-FPGA cards to deploy an energy-aware data center network.

[19] designs an energy efficient virtual machine placement and quality of service aware routing with elephant flow detection scheme based on SDN. However, the above studies including [5] lack the analysis of computation time, which is also a critical factor in SDN environment.

III. DYNAMIC ROUTING METHOD

In this section, the problem of energy-aware routing is formulated as an Integer Linear Programming (ILP) problem. The goal is to minimize the number of links used in the network subject to the link capacities and the rule capacities in the routers. Adopting the network management capabilities of SDN, the routes of flows can be chosen in order to reduce the number of used links.

A. Problem formulation

Let $G = (V, E)$ be an undirected weighted graph for a network topology where nodes in $V$ represent routers and links in $E$ represent physical links connected with the adjacent routers. Denote $d_{s,t}$ as a demand for traffic bandwidth from node $s$ to node $t$, where $d_{s,t} \geq 0$, $s, t \in V$, and $s \neq t$. Let $C_u$ denote the rule capacity in node $u$ and $L_{u,v}$ denote the link capacity on the link between $u$ and $v$. The goal is to find routing paths for a set of traffic demands with the minimal number of active links subject to the constraints of rule and link capacity. The notations are summarized in Table I and the problem is formulated as follows:

$$\min \sum_{(u,v) \in E} x_{u,v} \quad \text{(ILP)}$$

s.t. $\sum_{v \in N(u)} k_{v,u}^{u,v} - k_{u,v}^{u,v} = \begin{cases} 1, & \text{if } u = s \\ -1, & \text{if } u = t \\ 0, & \text{otherwise} \end{cases}$ \quad (1)$

$$\sum_{d_{s,t} \in D} \sum_{u \in V} d_{s,t} (k_{u,v}^{u,v} + k_{v,u}^{v,u}) \leq L_{u,v} x_{u,v}, \forall (u, v) \in E \quad (2)$$

$$\sum_{d_{s,t} \in D} \sum_{v \in N(u)} k_{u,v}^{u,v} \leq C_u, \forall u \in V \quad (3)$$

$$x_{u,v}, k_{u,v}^{u,v} \in \{0, 1\}, \forall (u, v) \in E, d_{s,t} \in D \quad (4)$$

The objective function minimizes the total number of links in use. Constraint 1 conserves that for each demand $d_{s,t}$, the number of flows entering and leaving a router must be the same except the source and destination. Constraints 2 and 3 are link and rule capacity constraints, respectively. For each link $(u, v)$, the total bandwidth requested by all demands on this link should not exceed the capacity of the link as shown in constraint 2. Similarly, the number of flows that use node $u$ as router should be less than or equal to the rule capacity of node $u$ as in constraint 3. Constraint 4 indicates that $x_{u,v}$ and $k_{u,v}^{u,v}$ are binary variables.

B. Dynamic Energy Aware Routing

The problem formulated in Section III-A is very challenging because a specific case of the formulated problem, called the minimum links routing problem, which only has
constraints on link capacity, is studied in [20] and has been proven to be NP hard. Since the problem is difficult to solve exactly and may even not be easy to find approximate to an insured factor, a heuristic algorithm named TEAR is proposed to provide a potential solution. The main idea of TEAR is that it prefers to assign the links that are already in use to new demands and shut down the unused links to reduce energy consumption. In fact, TEAR also considers the dynamic properties of demands and avoids re-routing the existing flows. The detail of the algorithm is shown in Algorithm 1. Whenever a new demand joins or an existing demand leaves, Algorithm 1 is performed. Specifically, in the proposed scheme, a graph $G$ is created according to the network topology. The algorithm can be partitioned into three parts:

1) Initialization: Let $W_{u,v}$ denote the weight of link $(u, v)$. $W_{u,v}$ is set to $|E|$ if it is not in use. Otherwise, it is set to 1. Initially, all of the links are not in use, i.e., $W_{u,v} = |E|$, $\forall (u,v) \in E$. Denote $\hat{C}_u$ and $\hat{L}_{u,v}$ as residual rule capacity and link capacity, respectively. Initially, $\hat{C}_u = C_u$, $\forall u \in V$ and $\hat{L}_{u,v} = L_{u,v}$, $\forall (u,v) \in E$.

2) New demands arrive: When a new demand joins the network (Line 7 in Algorithm 1), first create $G = G$ with weight $W_{u,v}$ on each link. Check the residual rule space and link capacity on $G'$. If the residual rule capacity of a node is empty, remove the node and all links connecting to it (Line 10). If the residual link capacity is not enough for the demand on a link, remove the link on $G$ (Line 13). Then, find the shortest path based on the weight of the links in $G'$ (Line 14). If the path is feasible (Line 16), subtract the requested capacity from the links on the shortest path (Line 17) and subtract one from the residual rule capacity on each node (Line 19) in $G$. Finally, set the weight of used links to 1 (Line 20).

3) Demands finish: When a demand leaves the network (Line 1-6), first add the released bandwidth back to the corresponding links and add one free entry to the residual rule capacity on each node along the route (Line 2-4). Then, check whether the link on the corresponding path is still in use. If no demands use the link (Line 5), turn the link off and set the link weight to $|E|$ (Line 6).

Algorithm 1 TEAR routing method

Input:
Demand $d_{s,t}$, an undirected weighted graph $G$, link capacity $L_{u,v}$, residual rule capacity $\hat{C}_u$, residual rule capacity $\hat{L}_{u,v}$

Output:
Routing solution

1: if $d_{s,t}$ is leaving the network then
2: Add the requested bandwidth $d_{s,t}$ back to links $(u,v), \forall (u,v) \in P_{d_{s,t}}$
3: Remove rule on node $n, \forall n \in P_{d_{s,t}}$
4: if $\hat{L}_{u,v} = L_{u,v}, \forall (u,v) \in P_{d_{s,t}}$ then
5: set $W_{u,v} = |E|$
6: else if $d_{s,t}$ is joining the network then
7: Create $G' = G$ with weight $W_{u,v}$ on each link
8: if $\hat{C}_u = 0, \forall u \in V$ then
9: Remove node $u$ and the links connecting to it from $G'$
10: if $\hat{L}_{u,v} < d_{s,t}, \forall (u,v) \in E$ then
11: Remove link $(u,v)$ from $G'$
12: Find the shortest path $P_{d_{s,t}}$ on $G'$ based on the weight
13: $W_{u,v}, \forall (u,v) \in E$
14: if $P_{d_{s,t}}$ is feasible then
15: Subtract the requested capacity $d_{s,t}$ from link $(u,v), \forall (u,v) \in P_{d_{s,t}}$
16: Update rule on node $n, \forall n \in P_{d_{s,t}}$
17: Set $W_{u,v} = 1, \forall (u,v) \in P_{d_{s,t}}$
18: return $P_{d_{s,t}}$
19: else
20: return Routing solution not exist

Note that the weight of unused links can be adjusted to control the level of preference for activating an unused link for data transmission. If the weight of unused links is too high, traffic demands may create congestion in some critical links. If it is set too low, links are easy to be activated for data transmission and may consume more energy. Moreover, the weight of used links is set to one. When choosing used links for data transmission, a demand prefers the path with the fewest hops in some sense. Of course, the constraints on link capacity and rule capacity on the nodes would also affect the selection of the routes.

TEAR algorithm is running on the top of an OpenFlow controller, which gathers network information including topology, rule capacity of routers, and link capacity as the inputs of TEAR algorithm. When a new demand joins the network, the

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>NOTATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G$</td>
<td>An undirected weighted graph</td>
</tr>
<tr>
<td>$V$</td>
<td>The set of total nodes</td>
</tr>
<tr>
<td>$E$</td>
<td>The set of total links</td>
</tr>
<tr>
<td>$d_{s,t}$</td>
<td>Traffic demand from node $s$ to node $t$</td>
</tr>
<tr>
<td>$N(u)$</td>
<td>The set of neighbors of node $u$</td>
</tr>
<tr>
<td>$x_{u,v}$</td>
<td>1 if link $(u,v)$ is in use, 0 otherwise</td>
</tr>
<tr>
<td>$D$</td>
<td>A set of demands</td>
</tr>
<tr>
<td>$k_{u,v}$</td>
<td>1 if demand $d_{s,t}$ is routed from $u$ to $v$, 0 otherwise</td>
</tr>
<tr>
<td>$C_u$</td>
<td>Rule capacity at node $u$</td>
</tr>
<tr>
<td>$\hat{C}_u$</td>
<td>Residual rule capacity at node $u$</td>
</tr>
<tr>
<td>$L_{u,v}$</td>
<td>Link capacity on link $(u,v)$</td>
</tr>
<tr>
<td>$\hat{L}_{u,v}$</td>
<td>Residual link capacity on link $(u,v)$</td>
</tr>
<tr>
<td>$W_{u,v}$</td>
<td>Link weight on link $(u,v)$</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
</tr>
<tr>
<td>$P_{d_{s,t}}$</td>
<td>Path of demand $d_{s,t}$</td>
</tr>
</tbody>
</table>
bandwidth broker in the architecture contacts the OpenFlow controller to request bandwidth reservation. The OpenFlow controller passes the requirement to TEAR algorithm, which calculates a path for the demand. Then, TEAR algorithm asks the OpenFlow controller to install rules in routers for energy aware routing. Since TEAR algorithm considers demands over time, it handles dynamic scenario well when new demands join the network or existing demands leave. Moreover, TEAR algorithm tends to assign the links that have been already in use to new demands such that the number of used links can be reduced.

IV. SIMULATIONS

Simulations are conducted to show the performance of the proposed method. The simulations are run on a machine equipped with 1.4GHz Intel Core i5 CPU and 4G RAM. The real topologies and traffic from SNDlib [6] are used as the input data. A variety of topologies are simulated and the results show better performance for TEAR compared to other algorithms. Due to space limitations, only the results for one of the topologies namely Germany50, shown in Fig. 1, is presented. There are 50 nodes and 88 links in the topology. The traffic was logged every five minutes over 24 hours (i.e., 288 time instants). The rule capacity is set to be 100 entries on each node and link capacity is set to be 1000 Mbps.

A. Data Modification

Since the recorded traffic is on backbone networks, most of the demands do not change frequently. In order to increase the dynamic property of the demands, the demands are cut randomly to mimic user behaviors. For each demand, a random length of traffic is cut as the preserved log followed by a random length of traffic removed. The preserving and removing operations are executed alternatively until the length of the traffic is less than a certain threshold.

For simplicity, the data rate of each demand is set to the maximum requested bandwidth during the life time of the demand. Ten thousand demands are selected randomly from the log for the simulations. Fig. 2(a) shows the total number of the selected demands at each time instant in the network and Fig. 2(b) shows the total data rate of the selected demands at each time instant.

B. Compared Algorithms

The performance of TEAR is compared to the algorithm ORPEAR in [5] and the original shortest path. ORPEAR divides the routing table into normal ports and a default port where the default port takes one entry in the routing table. The ORPEAR algorithm is briefed as follows:

• Step 1 - Find a feasible routing schedule: for each demand \( d_{s,t} \), find the shortest path for \( d_{s,t} \). Update the residual rule capacity and link capacity on all links along the route. If the rule capacity on a node is full, shrink the routing table by assigning the default port to the port which carries the highest number of flows.

• Step 2 - Reduce the number of used links: after finding a feasible routing schedule for all the demands, the algorithm then removes the link with the most residual capacity from the topology and re-run Step 1 until no feasible route schedule can be found.

Another algorithm to compare is the original shortest path for each demand, i.e., the least hop route, determined by Dijkstra’s algorithm.

C. Performance Evaluations

In order to perform dynamic demand routing, ORPEAR and the shortest path algorithm are triggered at the time instant when there are demands joining or leaving the network. The following metrics are used to compare the efficiency of the algorithms.

1) Processing time: the time of calculating the paths for all the demands throughout the simulation.

2) Power saving: the amount of unused links is transformed into power saving ratio by using the power mode [18] [21]. The power calculation can be accomplished by Table II and the following equation 5.

\[
P_{sw} = P_{chassis} + N_{rate} \times P_{rate} \]  

Where:

• \( P_{sw} \): Total power consumption of a switch.
• \( P_{chassis} \): The basic power consumption of a switch chassis.
• \( N_{rate} \): Number of ports in link state none active, 10Mbps, 100Mbps, 1Gbps.
3) Rule installation cost: the number of times that the controller updates rules in routers.

The processing time evaluation is shown in Fig. 3. Controller’s processing time is important for time-sensitive applications and the processing time is dominated by the time of finding feasible routes. TEAR performs much better than ORPEAR since TEAR does not reroute all the existing demands and only assigns routes to new demands. If ORPEAR is applied to the scenario with dynamic demands over time, ORPEAR has to find the shortest paths for all the demands when new demands arrive and then reschedules all the routes by removing the least loaded link one by one until no feasible route schedule can be found. It is much more time consuming and only appropriate for the condition with static demands.

Fig. 4 shows the simulation result about power saving over time. Note that less links in use are equivalent to more power saving. The result shows that TEAR performs similar to ORPEAR and much better than the original shortest path algorithm. Although TEAR and ORPEAR have similar results on power saving, ORPEAR spends much more time than TEAR in finding the routes.

Fig. 5 shows the rule installation cost over the simulation, i.e., the number of rule changes at each time instant when new demands arrive. Rule installations drive the controller to update the rule table in the routers and could decrease the transmission efficiency. TEAR and the original shortest path algorithm perform rule update only for new demands joining or old demands leaving the network. The original shortest path algorithm makes fewer rule updates because it chooses the path without considering the usage of links. ORPEAR reroutes all demands in order to reduce the number of used

\[ P_{rate} : \text{Power consumption of ports in link state none active, 10Mbps, 100Mbps, 1Gbps.} \]

Based on the number of unused links which was calculated by the algorithms, the total energy consumption of each switch can be obtained in order to compare the energy saving of each algorithm.
links when new demands arrive and changes almost all the rules for the routing. Apparently, it is not appropriate for the condition with dynamic demands.

From the experimental results, one can observe that ORPEAR is good in finding the routes for the demands using less links. However, it spends a lot of time generating the route schedule. The processing time is about 50 times longer than that of TEAR. In addition, the scheme needs to reroute all the demands when new demands arrive. It may result in higher rule updates and is not appropriate for the condition with dynamic demands over time.

The original shortest path algorithm does not consider the number of link used. Therefore, the number of unused links is pretty low. From the simulation results, the number of unused links when TEAR is used is, on average, 5 times more than that when the original shortest path algorithm is used. The benefit of the original shortest path algorithm is that it can reduce the number of rule updates since it chooses route with the fewest hops.

TEAR obviously outperforms the other schemes. It spends much less processing time assigning routes for the demands while still has pretty low energy consumption overall. Furthermore, the total number of rule updates in routers is also in a decent situation. These characteristics could be very helpful for practical applications which are time-sensitive and prefer low power consumption.

V. CHALLENGES

Although TEAR outperforms ORPEAR and the original shortest path algorithm in terms of computation time and energy consumption, there are some challenges of TEAR. Because TEAR intends to choose the links in use by setting the weight of the used links to one, the path of existed demands may be longer than the one in the original shortest path algorithm. Moreover, reusing the links in use will lead to congestion of traffic in some parts of the topology, and therefore QoS (Quality of Service) will be effected. Consequently, the average length of path and the average number of unacceptable demands are evaluated to comprehend the difference of three algorithms.

In Fig. 6, the original shortest path algorithm just finds the shortest path for each incoming demand without considering any constraint. On the other hand, TEAR and ORPEAR consider the usage of the nodes and links to achieve energy saving, thus the demands will be assigned to a longer path in
order to satisfy the requirement of the algorithms.

Fig. 7 shows the average number of unacceptable demands of three algorithms. The more unacceptable demands represents the more demands which can not find the feasible path. Since TEAR reuses the links and may cause some critical links and nodes to become over utilized, the number of unacceptable demands is higher than ORPEAR and the original shortest path algorithm.

For future work, load balancing will be considered by designing a new adaptation mechanism of link weight. In order to show the trade-off between energy saving and QoS, we will consider the utilization of flow table and the bandwidth of links and design a new model to adjust the variables in adaptation mechanism. Consequently, the network service providers can easily manage their network to achieve different preference and goals.

VI. CONCLUSION

Energy consumption in the ICT industry is a critical issue in recent years. This paper investigates the problem of energy aware routing in SDN enabled networks. SDN controllers can monitor the condition of the network and instruct the routers to forward arriving traffic demands in the way that the total number of used links is reduced. The solution, namely TEAR, is proposed to resolve the energy aware routing problem with dynamic demands over time. Simulations are conducted to show the performance of TEAR using real topology and traffic logs. The results show that TEAR can effectively reduce the number of used links while significantly outperforming the other schemes in computation time. It fulfills the basic requirement for time efficiency in contemporary network environment and the natural characteristic of demands arriving and finishing over time in practice.

ACKNOWLEDGMENT

This work is supported by MOST 105-2218-E-011-014 -MY3, 107-2218-E-002-048-, 106-2221-E-011-003-, and the Industrial Technology Research Institute (ITRI), Taiwan, Republic of China (ROC).

REFERENCES