Energy-Aware IP routing over SDN

Saptarshi Ghosh, Tasos Dagiuklas, Muddesar Iqbal
School of Engineering, Division of Computer Science & Informatics, London Southbank University, UK
Email: {ghoshs4, tdagiuklas , m.iqbal }@lsbu.ac.uk

***Abstract*— The routing protocols play a vital role in saving energy, especially by minimizing the time a packet takes to travel from source to destination. The aim of energy-aware routing protocols is to select a route that engages routers in such a way that the overall energy consumption is minimized. In this paper a relationship between resource utilization and energy consumption is stated, further, a resource-aware dynamic routing algorithm for SDN is proposed. The contribution of this paper is a queuing theory-based approach that measures the average waiting time of nodes and links based on their utilization and finds a path that costs the least time. The paper also proposes a framework for implementing routing algorithm over an SDN. Performance of the algorithm is verified using a GNS3 based implementation with an Opendaylight controller.**

Keywords—Resource aware routing, SDN

I. Introduction

Energy awareness techniques in routing algorithms are in the limelight of the research community for a while. For the last few decades, it is evident that Moor's law is broken, and devices are being more and more powerful. However, on the flip side, they are becoming more power hungry, and the advancement in battery capacity is not coping up with the rate. Therefore, designing energy efficient software has become a trend in the research community to meet the green objective. Contribution from several fields has made it a very rich domain. In [1], the authors present how energy savings can be optimized by offloading application using Microsoft's MAUI framework. But when the local energy is saved by executing an intense part of program remotely, communication cost comes in which is proportional to the routing time. Routing algorithm plays a vital role in the energy savings schemes. Routing protocols developed for homogeneous networks such as Ad-hoc On-Demand Distance Vector (AODV), doesn't work for the heterogeneous environment, as the resource utilization of network devices affects the efficiency. Hence, Resource-Aware Routing for Low powered and Lossy Networks (RPL), was standardized (RFC 6550) [2] which also formulates the node cost calculation metric. Link cost calculation is typically depending on the nature & type of the network, however, there are some generalized techniques discussed in [3][4].

Software Defined Networking (SDN) [5] is also becoming the de facto standard of the modern networking. It decouples the control and data plane. Control plane (CP) is a logically centralized entity hosted by one or many devices called Controllers, it instructs the traffic forwarding rules to the Data Plane (DP) which constitutes switches, which only forwards. CP bridges with the DP with OpenFlow [6] protocol and switches register the instructions in OpenFlow Tables.

This paper has designed and developed an energy-aware routing algorithm that exploits application offloading. Further, it proposes a resource-aware routing algorithm for SDN, which monitors the resource utilization of network devices (nodes) and channels (links), using a push agent and fetches topology and flow table information from the controller. Using *Link Queue Modelling* [10] and *Stochastic Network Calculus* [11], it guarantees a route that avoids busy nodes and uses unutilized ones. Results show the validity of the algorithm.

The rest of the paper is organized as follows, Section II presents the state of the art, section III describes the problem statement, Section IV introduces the algorithm, Implementation details and simulation results are shown in Section V and we conclude on Section VI.

II. Related Works

In battery powered networks such as WSN, energy-aware routing is one of the key areas of researchers. There is a wide spectrum of work that has been done on traditional wired & wireless networks. Han Bo in his paper [7] has applied energy awareness in SDN based WSN. Energy-aware routing optimizes total energy utilization of the network by prioritizing the power healthy devices like line powered routers [8] Or steering traffic in such a way that engages minimum network devices [9]. The other way of optimizing energy utilization is selecting devices for a traffic with higher efficiency. Therefore, the resource awareness idea comes in, where a routing path involves more underutilized devices.
Most researchers have contributed to the RPL protocol in Low powered Lossy networks. The authors in the articles [12], [13] addressed and solved some of the bottlenecks of native RPL by adding mixed mode operation, adaptively, hierarchical routing etc. and applied on heterogeneous wireless M2M & IoT domains. Advantageous over traditional networking for lowering down the control message overhead. Also, a push agent-based implementation would replace the negotiation mechanism used in RPL and a generalized resource metric to replace the threshold based discrete MOP domain. The work of D. Lee [14] resembles us, the author proposes a proactive k-

shortest path approach, and the only limitation for this solution would be dealing with a loosy network.

A. Contributions

This paper proposes a Temporal Resource-aware Routing Algorithm (STR-RA) for SDN. That contributes the following

a. An algorithm has designed and developed to determine the node and link utilization. This has been accomplished through a push agent (*Shellmon-client*) based mechanism where the agent runs on every Open-V-Switches (OVSs) of the SDN and updates a remote server (*SellMon-server*) about the node and link utilization. The collected data are normalized by the client.
b. A technique, called Stochastic Temporal Edge Normalization (*STEN*), has been introduced. It is a stochastic network calculus-based model that normalizes the node costs by distributing it to the edges in time domain.
c. Finally, using an optimization model proposed by [1], the relationship between optimal saved energy and efficient routing has been demonstrated both analytically and via simulation.

III. Problem Statement

In this section, the problem formulation and the mathematical modeling for the algorithm are presented.

A. System Model

Consider $G(V,E)$ is directed graph represent the network topology. The network connects the switches with the controller, which is not a part of this graph. $V=\{v_i|1<i<n\}$ is the vertex set and represents the open-flow switches (OVSs) and $E=\left\{e_{ij}^{(t)}|adj(v_i,v_j),\forall v_i,v_j\in V\right\}$. The function $adj(v_i,v_j)$ returns the weight associated with the edge at time instance t. For distinct vertices pair (i.e. v_i & v_j), the function $adj()$ returns the initial link cost and for identical vertices pairs (i.e. v_i & v_i) it represents a weighted self-loop and $adj()$ returns the node cost. Cost calculation and metrics are explained in later section. Since each vertex v_i represents an OVS, it connects a several hosts or end devices denoted by the set $H_i=\{h_{i,j}\}$. Each host $h_{i,j}$ typically contains its addressing information (i.e. IP and MAC).

B. Relationship between Energy and Routing

Assume, an application requires a total of E amount of energy to run locally. Without loss of generality, it is assumed that part of the application runs locally, and the rest is offloaded to it remotely; Then, E can be expressed as a sum of the energy consumed for local execution (E^l), remote execution (E^r) and data transfer (E^t). From the source's perspective $E^r=0$ as it is not utilizing the source's energy resources. Hence, the actual energy saved by offloading the application partially, is E^l-E^t, as the energy spent for data transfer acts as a penalty for the saved energy.

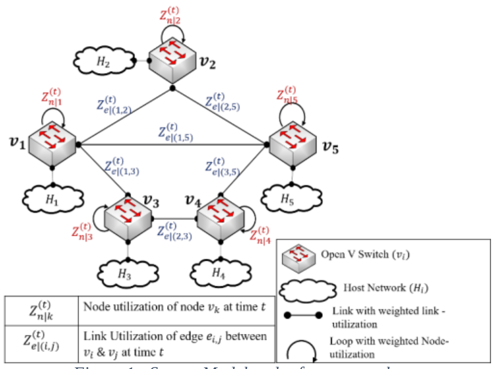

Figure 1 : System Model and reference topology

Originally proposed by Microsoft in their article of MAUI framework, formulates the optimal saved energy for a call graph in a distributed application, with a constrained latency. This section explains the original formulation and then state the scope of advancement which is addressed in the following sections.

The proposed solution is a 0-1 integer linear programming problem (IPP). The objective function maximizes the energy saved by executing a method remotely. The saved energy is the difference of the total energy cost of local execution, $(E_v^l \mid v\in V)$ and the total data transfer cost for executing the method, $\left(C_{u,v} \mid u,v\in V \text{ and } e_{u,v}\in E\right)$ .There are two constrains for the above objective function. First, the total time for the execution $T_v^l+T_v^r$ must be within a certain latency$L$. Where T_v^l & T_v^r are the local and remote execution time of $v\in V$. Second, only remote-based methods can be offloaded for remote execution. The formal representation is given below.

$$maximise \sum_{v\in V} I_v\times E_v^l - \sum_{e_{u,v}\in E} |I_u-I_v|\times C_{u,v} \tag{1}$$

$$such\ that, \left(\sum_{v\in V}(1-I_v)\times T_v^l + I_v\times T_v^r\right) + \sum_{e_{u,v}\in E}\left(|I_u-I_v|\times B_{u,v}\right)\le\ L \tag{2}$$

$$and, I_v\le r_v\ \forall v\in V \tag{3}$$

Where, I_v is an integer that is equal to 0 for local execution and 1 for rethe mote. r_v Represents methods marked as "*remotable*", and $B_{u,v}$ is the state transfer time from u to$v$.

It can be clearly inferred from equation 2 that the latency satisfiability constraint is linearly dependent on the execution time T_v^l & T_v^r and state transfer time $B_{u,v}$. further, remote execution and state transfer time is proportional to the network delay. Hence a routing protocol that selects the shortest path that takes least time to than its length or hop count, has a higher

Commented [TD1]: Title in the label is missing probability to meet the latency satisfiability constraint. Eventually optimizing the saved energy, defined as equation 1.

Stating the relationship between the routing protocol and energy savings, the following section discusses the design cost calculation and design of our proposed algorithms.

C. Cost calculation of nodes

Cost of a node is calculated by cumulating its various resources' utilization. In addition to the parameters used in the original draft of RFC 6550 (RPL), we have introduced a robust cost calculation function that incorporates more resource parameters such as (memory and CPU frequency, core count etc.). Various parameters and their symbols are listed below.

- f_c : Frequency of the CPU per core
- f_m: Frequency of the RAM
- N_c: Total number of Cores
- N_m: Total volume of RAM
- U_c: Percentage of processer utilization
- U_m: Percentage of memory utilization
- $\overline{U}_b$: Percent of utilized battery (100 for line sourced)
- R_b: Rate of battery usage (1 for line sourced nodes)

Let Z_n be the node utilization factor, a higher Z_n means less occupied node. Each node represents an OVS. In a virtualized heterogenous environment, resource allocation is unbounded. Therefore, a node of 20% resource utilization with a dual core CPU is equally busy, that of a 10% utilized with quad core. The same applies to memory utilization. Hence the percent of utilization is not enough to decide the load of the system, rather counting the free clocks. For a battery powered device, the fitness can be judged by how long the remaining power can last? There is no point of choosing a node that has adequate CPU and memory resource, but the battery is about to run out. Therefore, we introduced a cutoff periodU_b^{min}, as the remaining battery time approaches the cutoff, Z_n must be diminished significantly. Z_n Is expressed formally below.

$$Z_n = \alpha(f_c \times N_c)(1 - U_c) + \beta(f_m \times N_m)(1 - U_m) + \gamma\left(\frac{\overline{U}_b}{R_b} - U_b^{min}\right)^k \tag{4}$$

The first term is the total amount of unused CPU frequency, the second term as unused memory, the third term is of order k because, as $\frac{\overline{U}_b}{R_b}$ (i.e. battery time remaining) tends toU_b^{min}, the contribution of the term drops at order k, which is a free parameter. For our experiment we found a best match at, $k = 2$, α, β & γ Are weighing coefficients.

D. Cost calculation of edges

The edge cost is calculated by two factors: Link quality (L_q) and Energy cost (E_c) and expressed as (Eq. 5),

$$Z_e = L_q - E_c \tag{5}$$

The following sections describe each factor.

1) Link quality calculation

The link quality of an edge specifies the reliability of the channel. It considers the amount of free channel capacity, signal strength, and average contention. The formal expression is the same used in ARPANET [3] is given below.

$$L_q = \alpha\frac{C - B_a}{B_a} + \beta\frac{RSSI_{max} - RSSI}{RSSI} + \gamma N_c \tag{6}$$

Where,
C: Link capacity
B_a: Available bandwidth
$RSSI_{max}$: Maximum signal strength (RSSI) value*
$RSSI$: received signal strength value*
N_c : Average contention
α, β, γ: Weighing components
**for wired devices, $RSSI_{max}$ & $RSSI$ are set to 1*

The author [3] heuristically obtains the values of the weighing components are, $= 1, \beta = 1, \gamma = 10$, on their experiments.

2) Energy Cost Calculation

Energy cost is only calculated when the device is battery powered. The following set of the equation (eq. 7) is used for calculation of the energy cost as per IEEE 802.15.4 [15] (Low rate wireless networks).

$$E_c = \eta_{tx}\alpha_{tx} + \eta_{rx}\alpha_{rx} \tag{7}$$

Where, η_{tx} & η_{rx} are the normalized energy costs for transmission and reception respectively with α_{tx} & α_{rx} are weighing components, set to 0 when line powered and 1 when battery. η_{tx}& η_{rx} Can be further stated as (eq. 8 and eq. 9),

$$\eta_{tx} = [(C_{tx-data} + C_{rx-ack})E_{link}]^x \left[1 + \left(1 - \frac{E_{tx-res}}{E_{tx-init}}\right)\right]^y \tag{8}$$

$$\eta_{tx} = [(C_{rx-data} + C_{tx-ack})E_{link}]^x \left[1 + \left(1 - \frac{E_{rx-res}}{E_{tx-init}}\right)\right]^y \tag{9}$$

Where,
C_{tx}, C_{rx} : are the energy consumption during transmission and reception respectively.
$E_{tx-init}, E_{rx-init}$: are the initial energy of the transmitter and receiver.
E_{tx-res}, E_{rx-res} : are the remaining energy of the transmitter and receiver.
E_{link} : The expected number of transmission represented as follows (eq. 10).

$$E_{link} = \sum_{i=0}^{k} i(1 - PRR)^i PRR \tag{10}$$

Where, k is the maximum number of retransmission and PRR represents the packet reception rate of a link.

x & y : are the weighing factors, if $x = y = 0$ then the shortest path comprises minimum hops and if $x = 1, y = 0$ then the shortest path comprises minimum energy.

E. Queueing Model of the network

The basis of the proposed algorithm is the theory of stochastic network calculus (SNC) [11]. SNC renders a network as a collection of interconnected queues, where each node and edge are modeled as a queue. However, our proposed algorithm is a simplified use case of the theory.

In our reference graph $G(V,E)$ representing a network topology, there are switches represented as nodes and links as edges. Now each node has a weighted self-loop, represents the nodes cost Z_n and edges too weighted with edge costs Z_e. The cost calculations are explained in section C & D.

Assume if a packet arrives on the switch v_i at time T_0, called arrival time (AT) and after being processed it leaves at time T_k then the interval $(T_k - T_0) = T_q$ is called service time or queueing time (QT). The QT is proportional to the queue size which is proportional to the load of the system. Similarly, the edges can also be treated as a queue. We can generalize the two costs Z_n & Z_e and express them as QT. Therefore, a path which is an alternating sequence of nodes and edges can also be a sequence of queues and the path cost be the sum of QTs (i.e. the total time a packet takes to traverse from the source node to the destination). Figure 2 depicts the queueing model of figure 1 where the weights of each edge and self-loop becomes the length of the corresponding queues. Each queue a point of entry and exit called *rear* and *front* (denoted as hollow and solid circles respectively on the figure2). For depiction simplicity, it is assumed that the links are simplex, i.e. $e_{i,j}$ can only get data from v_i to v_j not vice versa.

The queueing system can be heterogeneous, i.e. each queue may run a different scheduling mechanism, and therefore, it is obvious to make a generalization. As mentioned earlier, the queue size is proportional to the processing load for the nodes and traffic load for the edges. The queue size also proportional to the QT, the mean of QT is also called average waiting time (AWT). Hence choosing a least time-consuming path can also be a sequence of queues such the sum of AWT is least among the possible alternatives, which inherently choose nodes and edges which are comparatively under-loaded. Here we present the relationship between AWT and Queue size.

1) AWT of nodes

Since the packets are arriving from many sources and the service time depends on the system load which depends on several random causes, therefor A & B has been chosen as distribution agnostic. Also, we assume the problem as an unbounded buffer problem with single server hence $k = \infty$ and$c = 1$. This makes the queuing model as $G/G/1$.

From the Little's rule,

$$W = W_q + \frac{1}{\mu} = \left(\frac{L_q}{\lambda} + \frac{1}{\mu}\right) = O(L_q) \tag{11}$$

Where,

W : AWT of the system

W_q: AWT of the queue

L_q : mean number of requests in the queue

λ : mean rate of interval

μ : mean service rate

From the approximated value of L_q for $G/G/1$ queues derived by Marchal,

$$L_q = O(\rho^2, \sigma_s^2, \sigma_a^2, \mu^2, \lambda^2) \tag{12}$$

Where,

ρ : Utilization of the server

σ_s^2, σ_a^2 : variance of the service & inter-arrival time respectively

Hence, from equation 4, 11 & 12,

$$W_{node} = O(L_q) = O(\rho^2) = O\left(\frac{1}{Z_n^2}\right) \tag{13}$$

Therefore, as the system goes busy, Z_n decreases and W (AWT) of the nodes increases quadratically (eq. 13).

2) AWT of edges

The AWT of edges are relatively simpler to calculate. Since the channel is FIFO, we consider the mean round trip time RTT as AWT which is inversely proportional to the edge cost. Therefore, from (eq. 5),

$$W_{edge} = O(RTT) = O\left(\frac{1}{Z_e}\right) \tag{14}$$

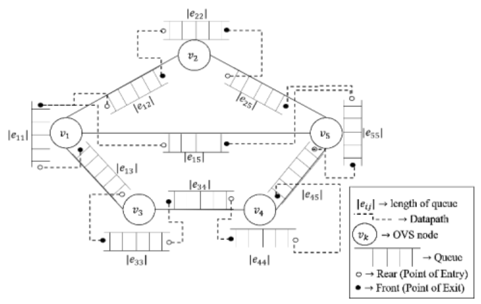

Figure 2 Queuing model of the network

F. Stochastic Temporal Edge Normalization (STEN) concept

Section E discussed the queue modeling of the graph. But there lies a problem finding the shortest path. All the shortest path algorithms assume the graph to be simple (i.e. no self-loop or

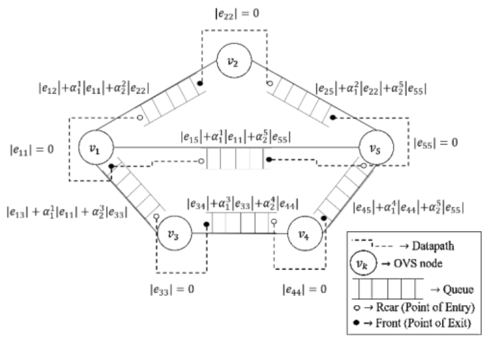
$|e_{22}| = 0$
$|e_{12}|+\alpha_1^1|e_{11}|+\alpha_2^2|e_{22}|$
$|e_{25}|+\alpha_1^2|e_{22}|+\alpha_2^5|e_{55}|$
$|e_{11}| = 0$
$|e_{15}|+\alpha_1^1|e_{11}|+\alpha_2^5|e_{55}|$
$|e_{55}| = 0$
$|e_{13}| + \alpha_1^1|e_{11}| + \alpha_2^3|e_{33}|$
$|e_{34}|+\alpha_1^3|e_{33}|+\alpha_2^4|e_{44}|$
$|e_{45}|+\alpha_1^4|e_{44}|+\alpha_2^5|e_{55}|$
v_1
v_2
v_3
v_4
v_5
$|e_{33}| = 0$
$|e_{44}| = 0$
→ Datapath
v_k → OVS node
→ Queue
○ → Rear (Point of Entry)
● → Front (Point of Exit)

Figure 3: After normalizing, node costs are diminished and get added with the link costs.

parallel edges). Our queue modeled graph has associated weighted self-loops. Before applying any of the shortest path algorithms, there is a need to normalize the loops by removing them. Once removed its weight must be distributed among the incident edges of the node where the loop was.

Figure 3 shows the normalized version of Figure 2. All the loops $|e_{ii}|$ are set to zero, instead, their values are distributed among the adjacent edged of the nodev_i. The coefficient α_j^i is a rational number between [0,1] that denotes a fraction of$|e_{ii}|$, such that $\sum_j \alpha_j^i = 1$. It specifies the next-hop probability of a switch v_i distributed over its incident edges. This edge normalization process is temporal as it changes time-to-time and stochastic because the fraction is probabilistic and distribution agnostic.

Once normalized, the graph will be re-aligned, the busy nodes will be put farther, and the free nodes will be put closer. Consequently, running any shortest path algorithm will choose a path with minimum path length, which in other words it comprises freer nodes than the busy ones.

The normalization function ℵ transforms a graph with self-loop to one with a normalized edge. ℵ Is defined formally below (eq. 15),

$$\begin{aligned} &\aleph\big(G(V,E)\big) \rightarrow G'(V,E') \\ &Such\ that,\ |e_{ij}| \rightarrow |e_{ij}| + \alpha_k^i|e_{ii}| + \alpha_k^j|e_{jj}| \\ &and, |e_{ii}| = 0\ \forall e \in E| \end{aligned} \tag{15}$$

IV. Proposed algorithm

This section discusses the proposed algorithm's design and analysis. Time complexity T of a routing algorithm can be expressed as (Eq. 16)

$$T = N_r(T_c + 2T_p + T_u) \tag{16}$$

Where $T_c, T_p \& T_u$ are mean cost calculation, propagation & update time respectively and N_r is the mean number of rerouting. Though in SDN, the complexity is far less as routing is performed by the controller, that simultaneously configures flow to the switches. This reduces $T_p = O(|V|^2)$ to$O(1)$. A proactive K-shortest path mechanism may suffer in a resource aware scenario, as the node costs changes frequently, it must recalculate the entire routing table again every time in a time complexity of $O(|V|^4)$. Thus, we opt for a purely reactive algorithm that selectively normalizes edges whose incident vertices' cost have changed.

1) Algorithm design

Algorithm 1: Shortest path generation for eligible pairs

Input: Graph $G(V,E)$ - Topology from SDN Controller
$Z_{n|v}^{(t)}$ & $Z_{e|l}^{(t)}$ - Utilization $\forall v \in V\ and\ \forall l \in E$ at time, t.

Output: Set of Routes $R_{ij}^{(t)}$

Steps:

```
1.  While (true) {
2.      Set $route \leftarrow r_{temp} \leftarrow \phi$
3.      Normalize $G : G' = \aleph(G)$
4.      For all vertex pair $(v_i, v_j) \in V(G') \times V(G')$ {
5.          If $(v_i, v_j) \in E'$ {
6.              If $\min(e'_{ik}) + \min(e'_{kj}) + \min(E') < |e'_{ij}|$
7.                  $route \leftarrow route \cup e'_{ij}$
8.              $\Delta e'_{ij}$ : change in edge weight
9.              If $\Delta e'_{ij} > \min(e'_{ik}) + \min(e'_{kj})$
10.                 $route \leftarrow route \cup e'_{ij}$
            }
11.         Else $r_{temp} \leftarrow r_{temp} \cup e'_{ij}$
        }
12.     If $route \neq \phi$
13.         $route \leftarrow route \cup r_{temp}$
14.     For all $(v_i, v_j) \in route$
15.         $R_{ij} \leftarrow R_{ij}\ U\ dijekstra(v_i, v_j)$
16.     For all $r_{ij} \in R_{ij}$ call *Algorithm 2*
17.     Sleep(Timeout)
    }
```

Algorithm 1 takes a graph $G(V,E)$ as input, normalizes it using equation (15), for all eligible node pairs it runs Dijkstra's single source shortest path algorithm. A route between v_i & v_j at time t is denoted as $r_{i,j}^{(t)}$. Each of which represents a sequence of nodes$\{v_k\}$. All such routes constitute the set$R_{ij}^{(t)} = \left\{r_{ij}^{(t)}\right\}$. A function $succ(v_k) = v_{k+1}|v_k \in r_{ij}^{(t)}, i \leq k < j$ on a node for a certain route returns the successor node, $succ(v_j) = \phi$. Algorithm 2 translates each $r_{ij}^{(t)} \in R_{ij}^{(t)}$ into a set flow entry$F_{ij}^{(t)} = \{f_k | k \in V\}$, to configure OVSs involve in the routerr_{ij}. Both source & destination IP addresses for flow match haven been used with output port as action. From Figure 3, at time t let an arbitrary route $r_{1,5}^{(t)} = \{v_1, v_2, v_5\}$, the corresponding flow entries will be (Table 1),

Table 1: Example of flow entries for $r_{1,5}^{(t)}$

OVS	Match		Action
	Source IP	Destination IP	
$v1$	H_1	H_5	$out: P(succ(v_1))$
v_2	H_1	H_5	$out: P(succ(v2))$

Where H_i is the set of IP addresses, local to OVS v_i. $P(v_j)$ returns the port number of v_i connectsv_j. The size of the flow set can be expressed as, $|F_{ij}^{(t)}| = (H_i \times H_j \times d)$, where d denotes the diameter of G. Hence, it can face space allocation problem for a network with large number of end-devices. A lookup table method such as Network Address Translation (NAT) can be a good solution to restrict the size at $O(V)$.

Algorithm 2: Configure OVSs with Flow entries

Input: Route $r_{ij} \in R_{ij}$

Output: Flow entry F_{ij}

Steps:

```
1.  For all $v_k$ in $r_{ij}${
2.      If $succ(v_k) \neq \phi${
3.          $ovs \leftarrow v_k$
4.          $sip \leftarrow H_i = \{h_i\}$
5.          $dip \leftarrow H_j = \{h_j\}$
6.          $port \leftarrow p(succ(v_k))$
7.          $ovs.addFlow($
                $nw_{src} = sip$
                $nw_{dst} = dip$
                $action = output: port$ )
        }
    }
```

2) *Complexity Reduction & Analysis*

Running Dijkstra's algorithm for all pair of vertices would cost$O(|V|^4)$. To reduce it, algorithm 1 only chooses those pair of vertices which are eligible. meaning they are potentially replaceable by an alternate path. The eligibility criteria are listed below,

a. If e_{ij} is an edge between two adjacent vertices (v_i, v_j) and the sum of minimum weighing incident edges of the subjected vertices and the minimum weighing edge of the entire graph is less than $|e_{ij}|$, i.e. (Eq. 17)

$$\min_{i,k \in E'}(|e'_{ik}|) + \min_{j,k \in E'}(|e'_{kj}|) + \min_{e}(E') < |e'_{ij}| \tag{17}$$

b. If the change in the value of a direct edge e'_{ij} ,denoted as $\Delta e'_{ij}$ exceeds the sum of minimum weighing incident edges of the subjected vertices. (Eq. 18)

$$\Delta e'_{ij} > \min_{i,k \in E'}(|e'_{ik}|) + \min_{j,k \in E'}(|e'_{kj}|) \tag{18}$$

c. All indirect vertex pair, i.e. $(v_i, v_j) \mid e_{ij} \notin E$ are eligible.

This doesn't reduce the asymptotic upper bound of the runtime, but the lower-bound significantly, when the eligible edges are few.

V. Implementation & Results

This section discusses the Implementation, methodology, and results. We implemented the test bed using GNS3 network emulator, OVSs are hosted by Docker containers. OpenDaylight (ODL) beryllium SR4 was used as an SDN Controller. MySQL Server is used for middleware & database management. We developed three apps (*Shellmon*, *route*, *TopoSense*) for the application layer.

A. *Experimental setup*

Each OVS runs *Shellmon Client* and sends event-driven resource updates to *Shellmon Server*. The *TopoSense* app retrieves topology and flow table information from ODL using *RESTConf* protocol from *nodes/topology* and *nodes/inventory* resources respectively and updates to the database. *Route-App* fetches data from the database, run algorithm 1 & 2, to generate a graph with resource information and shortest path for eligible edges. Each shortest path then gets configured to the OVS using OpenFlow packet out messages from the controller. Figure 4 depicts the complete data-flow.

B. *Methodology*

This section describes the methodology we followed in order during the experiment.

i. A non SDN (*Quagga* based) topology was built & configured with the reference topology (Figure 1). The end to end (E2E) throughput between two hosts has been tested using *iperf* while overloading an intermediate router with *stress* tool. The experimental result shows the throughput falls in a quadratic rate both for RIP & OSPF, which matches the expected result (Eq. 13).

ii. The proposed technique has been implemented using an SDN platform depicted in Figure 4, keeping the topology same. Results show a linear characteristic compared to the exponential rise.

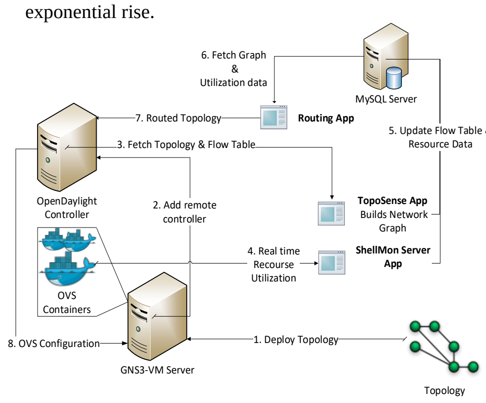

Figure 4: Experimental Setup and Dataflow Architecture

C. Results

Figure 6 shows the result, with CPU threads along horizontal axis vs the moving average plot of throughput achieved. a pair of the quadratic fitted curve also confirms the characteristic equation for RIP & OSPF.

The utilization vs E2E delay characteristics is shown in figure

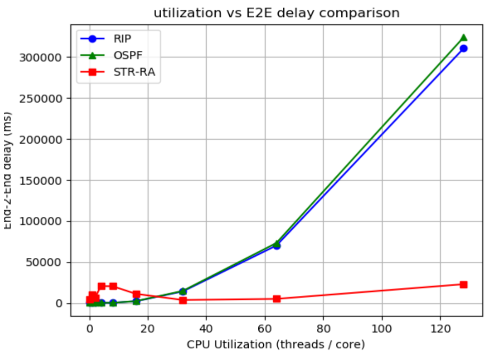

Figure 5 Comparison between RIP, OSPF & proposed STR-RA in Utilization vs Delay characteristics

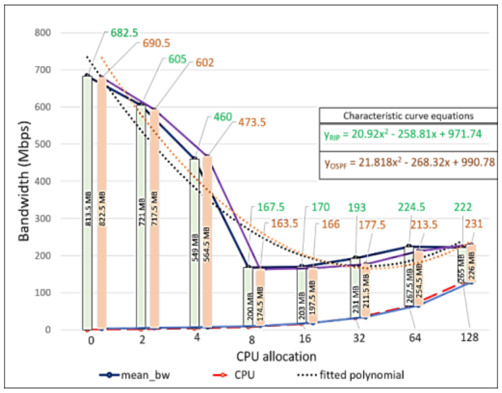

Figure 6: E2E Throughput falling under RIP & OSPF, when one of the intermediate router gets overloaded

5. The initial delay for the proposed STR-RA algorithm is due to control packer exchange between OVS & ODL and the apps to generate the data structure. The delay touches the minima immediately after the initialization. After thread count exceeds 60 it finds & switches to a different path with larger hop count that causes a slight hike.

VI. Conclusion

This paper states how energy total consumption of a network depends on the resource utilization of its devices. Therefore, a resource-aware routing protocol is proposed that monitors both the node and link utilization dynamically. The routing algorithm uses average waiting time of a path as a metric, which is modeled using stochastic network calculus. an SDN framework for the algorithm is proposed, which does dynamic translation of the shortest paths into flow entries. Finally, results confirm the performance comparing RIP & OSPF.

References

[1] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl, "MAUI", *Proceedings of the 8th international conference on Mobile systems, applications, and services - MobiSys '10*, 2010.

[2] "RFC 6550 - RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", *Tools.ietf.org*, 2012. [Online]. Available: https://tools.ietf.org/html/rfc6550. [Accessed: 30- Apr- 2018].

[3] K. Kowalik, B. Keegan and M. Davis, "RARE - Resource Aware Routing for mEsh", *2007 IEEE International Conference on Communications*, 2007.

[4] V. Gungor, C. Sastry, Z. Song and R. Integlia, "Resource-Aware and Link Quality Based Routing Metric for Wireless Sensor and Actor Networks", *2007 IEEE International Conference on Communications*, 2007.

[5] "RFC 7426 - Software-Defined Networking (SDN): Layers and Architecture Terminology", *Tools.ietf.org*, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7426. [Accessed: 30- Apr- 2018].

[6] *OpenFlow Switch Specification*. 2018, pp. 12-24.

[7] H. Bo, W. Muqing, Z. Min and L. Wenxing, "An energy aware routing algorithm for software defined wireless sensor networks", *2017 IEEE/CIC International Conference on Communications in China (ICCC)*, 2017.

[8] S. He, K. Xie, W. Chen, D. Zhang and J. Wen, "Energy-Aware Routing for SWIPT in Multi-Hop Energy-Constrained Wireless Network", *IEEE Access*, vol. 6, pp. 17996-18008, 2018.

[9] J. Manjate, M. Hidell and P. Sjodin, "Can Energy-Aware Routing Improve the Energy Savings of Energy-Efficient Ethernet?", *IEEE Transactions on Green Communications and Networking*, pp. 1-1, 2018.

[10] W. Jin, *A link queue model of network traffic flow*. 2013, pp. 3-8.

[11] M. Fidler and A. Rizk, "A Guide to the Stochastic Network Calculus", *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 92-105, 2015.

[12] J. Guo, P. Orlik, K. Parsons, K. Ishibashi and D. Takita, "Resource Aware Routing Protocol in Heterogeneous Wireless Machine-to-Machine Networks", *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015.

[13] J. Guo, P. Orlik and K. Ishibashi, "Resource aware hierarchical routing in heterogeneous wireless IoT networks", *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2016.

[14] D. Lee, P. Hong and J. Li, "RPA-RA: A Resource Preference Aware Routing Algorithm in Software Defined Network", *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015.

[15] "IEEE Standard for Low-Rate Wireless Networks."