DREAM: DistRibuted Energy-Aware traffic Management for Data Center Networks

Liang Zhou
University of California, Riverside
lzhou008@ucr.edu

Laxmi N. Bhuyan
University of California, Riverside
bhuyan@cs.ucr.edu

K. K. Ramakrishnan
University of California, Riverside
kk@cs.ucr.edu

ABSTRACT

Traffic consolidation has been proposed to save energy in data center networks. However, existing centralized traffic consolidation approaches focus on achieving optimal network energy saving, without considering the need to be responsive to traffic variability. In this paper, we present DREAM, a distributed flowlet-level traffic consolidation framework for achieving energy efficiency in data center networks. DREAM splits a TCP flow across multiple paths based on ECN feedback by adapting the path selection probability for sending a flowlet. This helps to choose a path while avoid congestion/queue build-up at switches. Distributed agents in DREAM are implemented in Open vSwitch at each server without modifying applications, TCP, or the hardware switches in the data center network. Testbed evaluations using traces from Wikipedia web service and Facebook MapReduce traffic prove that DREAM on average achieves at least 15.8% energy saving for the data center network, while state-of-the-art approaches such as CARPO and ElasticTree produce 11.6% and 8.4% energy saving, respectively. The packet drop ratio in DREAM is less than 0.01% while the best among the alternatives, ElasticTree, has at least 0.19% drop ratio. DREAM also has 30% lower application-level latency than state-of-the-art centralized traffic consolidation approaches.

CCS CONCEPTS

• **Networks → Network management**; **Data center networks**.

KEYWORDS

energy efficiency, distributed traffic consolidation, flowlet

ACM Reference Format:
Liang Zhou, Laxmi N. Bhuyan, and K. K. Ramakrishnan. 2019. DREAM: DistRibuted Energy-Aware traffic Management for Data Center Networks. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems (e-Energy '19), June 25–28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307772.3328291

1 INTRODUCTION

Energy consumption of data centers is significant enough to warrant effort to reduce it. In 2014, data centers in the U.S. consumed 70 billion kWh of energy, representing 1.8% of the U.S. electricity consumption [35]. Most of the energy is consumed in servers, but aggressive energy savings have been recently developed and applied in data centers [20, 22, 23, 37]. With energy-proportional servers, the fraction of energy consumed by the networking components can reach 50% on Google clusters, especially when the server utilizations are low, at 15% [1]. Hence, it is desirable to design practical techniques for the energy-proportional Data Center Networks (DCNs).

DCNs provide significant path redundancy and typically have low link utilizations [34]. Traffic consolidation [17, 38, 42] proactively shifts all traffic to a minimal subset of network devices and completely turns off unused links and switches. Previous works perform a centralized optimization periodically to decide the set of active switches and links based on current traffic demands. These techniques promise to yield considerable energy savings ranging from 25-62% [17].

However, state-of-the-start centralized traffic consolidation approaches such as CARPO and ElasticTree formulate the energy saving in DCNs as a linear programming model, which usually has high computational complexity. For example, it may take hours for the centralized linear programming model to find the minimal subset of active switches and links in large DCNs [17, 38]. Current traffic consolidation frameworks [17, 42, 44] run periodically, epoch by epoch, to adapt to traffic variation. The epoch length should be at least larger than the computation time of centralized optimizer. Otherwise, we suffer energy inefficiency or link congestion. Poor responsiveness of centralized traffic consolidation becomes worse when traffic bursts [21] occur.

Even heuristics to improve the scalability of such centralized approaches (e.g., a greedy bin packing algorithm [17]) takes more than 1000 seconds to find the active portion of DCN topology for a network topology with 10K hosts [38]. Orthogonal to the computation time of these centralized optimization-based approaches, the epoch length of these algorithms is still constrained by the frequency with which we can monitor traffic statistics. Typically, a centralized controller uses protocols such as Simple Network Management Protocol (SNMP) [8] or Openflow [25] to obtain flow statistics. Nevertheless, the polling frequency from hardware switches is limited, considering overheads. For example, a HP switch [11] may only refresh flow statistics every 20 seconds for Openflow protocol. Thus, the epoch length for centralized optimizer needs to be at least 20 seconds.

For improved responsiveness, and thereby higher energy savings, we propose DREAM, a DistRibuted Energy-Aware traffic Management framework for data center networks. In DREAM, the distributed agents and hosts work cooperatively to consolidate traffic to a portion (e.g., the left-most) of the DCN. It reacts to traffic bursts

e-Energy '19, June 25–28, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6671-7/19/06…$15.00
https://doi.org/10.1145/3307772.3328291

at the Round Trip Time (RTT) time scale. The network monitoring is based on data plane of the hardware switch rather obtaining values of flow counters of the forwarding plane through Openflow. The distributed agents are implemented as a shim layer, such as Open vSwitch [31]. We require no modification to the host protocol stack or the data center networking fabric. This makes our design easy for deployment.

In DREAM, we use the existing primitives in hardware switches such as Explicit Congestion Notification (ECN) marking [3, 14, 32] to report link congestion to DREAM's distributed agents in servers. The ECN marking in DREAM is also beneficial for controlling latency for an application's flow. The distributed agents monitoring ECN feedback reroute traffic when a path is congested, but at a flowlet granularity. DREAM reacts before significant queuing or packet loss occurs.

The basic scheduling unit in DREAM is flowlet (i.e., a burst of packets) [2]. This enables it to take advantage of the multiple active paths in the DCN without any modification to the host protocol stack. Flowlet scheduling also reduces fragmenting of the link capacity and allows for better energy savings. For each active path, a value representing the probability of sending next flowlet on that path is maintained. The selection probability value is increased additively after successfully sending a flowlet over an uncongested path, and decreases substantially when observing ECN feedback indicating congestion on the path. Increasing the selection probability of left-most path correspondingly reduces the selection probability for the right-most path, thus eventually resulting in traffic consolidation. Prior work [10] on proving that the equilibrium point for Additive Increase Multiplicative Decrease (AIMD) can be re-used to show that its use in our design can result in a stable operation.

Finally, we propose a packet encapsulation format in the Open vSwitch to achieve explicit path control. In prior works [17, 42, 44], the routing path is decided by the centralized controller. DREAM has to enforce explicit routing path in a distributed manner. Inspired by Xpath [18], we pre-install the forwarding rules based on a compressed path ID at switches. At the traffic source, the distributed agent encapsulates packets and inserts the path ID in an encapsulation header. Switches match on the path ID along with other header fields to route the flows appropriately. To minimize encapsulation overheads, we re-use primitives already in Open vSwitch.

DREAM is implemented in a testbed DCN with a leaf-spine topology. The testbed evaluation uses the Wikipedia traffic trace [36] and more bursty Facebook MapReduce traffic trace [9]. We show that DREAM on average achieves at least 15.8% DCN energy saving, while CARPO and ElasticTree produces 11.6% and 8.4% energy saving, respectively. The packet drop ratio in DREAM is less than 0.01% while the best among the alternatives, ElasticTree [17], has 0.19% drop ratio on Facebook trace and 0.85% drop ratio on Wikipedia trace. Finally, DREAM has at least 30% shorter application-level latency compared to the alternatives. Our major contributions include:

- To improve responsiveness and scalability, we consolidate traffic to a subset of the DCN in a distributed manner, to achieve energy saving. The distributed agent is implemented in Open vSwitch without any modification to the host protocol stack or the data center network switches.
- We propose to reroute traffic based on ECN feedback, with congestion information obtained every RTT. The more frequent, accurate feedback of incipient congestion on the precise path a flowlet traverses helps achieve much better network latency.
- Network traffic is scheduled at the granularity of flowlet to take advantage of multiple active paths in the DCN, without any host side modification. We propose a multi-path flowlet scheduling algorithm for the energy efficient DCN.
- We propose a packet encapsulation format for explicit path control in the network.
- We demonstrate the effectiveness of DREAM in a testbed implementation with production switches.

2 BACKGROUND AND MOTIVATION

Traffic consolidation shifts network flows to a minimal number of active switches and turns off idle switches and links to save power. State-of-the-art traffic consolidation frameworks [17, 42, 44] are not responsive to traffic variation because of the long computation times involved with the optimization in a centralized manner and also depends on the limited refresh rate for the flow-level statistics from network switches. Poor responsiveness results in link congestion or packet drops in the network as well as poor adaptation in terms of managing energy consumption. This motivates us to design a distributed energy-aware traffic management framework for the DCN. The distributed design can quickly adapt to traffic fluctuation in the DCN, and promises to fully take advantage of every energy saving opportunity when the network load ebbs. What is more, it also can move out traffic from congested paths in a more timely manner. In such cases, it reduces the packet drop rate and latency.

2.1 Energy Efficient Data Center Networks

Data centers host tens of thousands of servers and consume tens of Megawatts of power [37]. Power management on servers is an important component, and the use of techniques such as Dynamic Voltage and Frequency Scaling (DVFS) [20, 23, 37, 47] and Virtual Machine (VM) migration [12, 40, 41] already seek to save power. ElasticTree [17] and a number of other following works [42, 44, 45] seek to make the data center *network* energy proportional as well, to complement the power management on servers.

ElasticTree [17] converts the energy management of DCN into an augmented Multi Commodity Flow (MCF) problem and then solves it by using Linear Programming (LP). Typically, the LP program runs at the centralized Software Defined Network (SDN) [7] controller. The SDN controller leverages the Openflow [25] protocol to fetch traffic statistics from the hardware switches periodically. The future bit rate of flows is predicted based on the traffic history in the last epoch, while providing for a safety margin [17]. Based on these traffic statistics, the centralized optimizer in the SDN controller runs every epoch to determine which subset of the DCN should be active. The epoch length needs to be longer than optimizer's computation time and more importantly longer than the rate at which the traffic statistics are reported by the hardware switches. The centralized optimizer outputs the updated routing path for each flow to achieve optimal network energy savings, which is enforced by the centralized SDN controller.

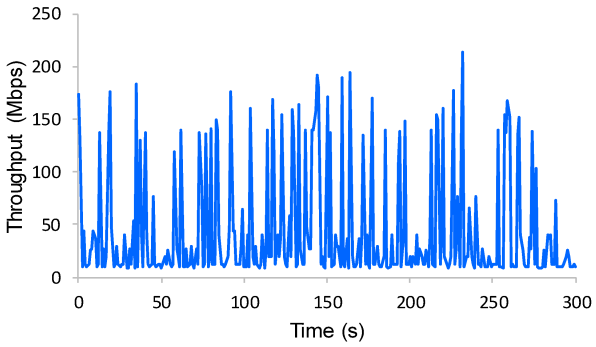


Figure 1: Traffic in the DCN has high variation, as shown for an example from the Microsoft search trace [3].

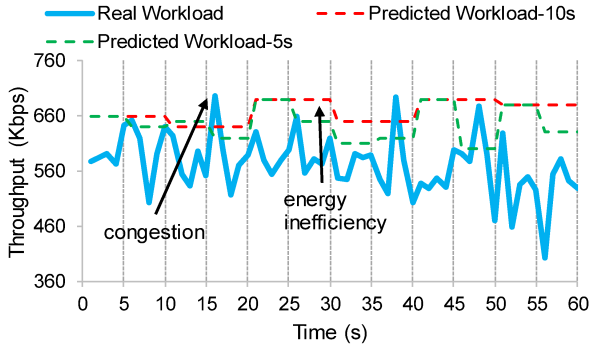


Figure 2: Poor responsiveness to traffic variation incurs link congestion or energy inefficiency. The network provisioning (i.e., predicted workload in the Fig.) with 5s epoch length has better energy saving than the one with 10s epoch length.

2.2 Traffic Variability and Responsiveness

Network traffic in data centers can be highly variable [6, 21]. Some applications, such as MapReduce [13], generate bursty traffic in a short time period, especially for the data shuffle phase. In Fig. 1, we plot the traffic variation of Microsoft Search trace [3] spanning a 5 min. interval. For example, the throughput at 34th second is 8.8Mbps but bursts up to 183Mbps at the next second.

In the ideal case for an energy proportional DCN, the energy consumption should adapt to the traffic variation. Otherwise, we suffer either link congestion or energy inefficiency. As we discussed in the previous subsection, the responsiveness of the energy saving framework depends on the epoch length for traffic scheduling. In Fig. 2, we give an example to show how the epoch length might impact energy saving as well as link congestion. The blue line represents one network flow in the data center. Its throughput varies over the 60 seconds shown. First, we set the epoch length for traffic consolidation at 10s. The red line is the predicted throughput based on the 90th-percentile [42] throughput in last epoch. Then, the centralized controller will reserve network bandwidth based on this predicted traffic data rate. Due to prediction errors, the observed flow's throughput might be larger than the predicted value, as shown in the interval 15s - 20s. But the centralized controller will reroute traffic only at the end of current epoch (i.e., at 20s). Packet drop and TCP retransmission could occur. A shorter epoch length promises to achieve better energy savings. We also plot the predicted workload for 5s epoch length (green line). During the interval 15s - 20s, when the rate of the flow decreases, the optimizer with 5s (smaller) epoch length will capture this energy saving opportunity, which will be missed if a longer epoch is used.

The epoch length is limited by two considerations: the computation time of the optimizer and the measurement frequency to obtain the traffic statistics. Modeling the traffic consolidation as a linear programming model has high computation complexity [38]. Although a heuristic algorithm has been proposed to accelerate this [17, 44], the computation time can still be high for large data center network topologies. Fig. 3 shows the computation time of linear programming and heuristic greedy bin packing [17] for different data center sizes. Greedy Bin Packing still takes more than 1000s for a medium size data center. We realize that this can be

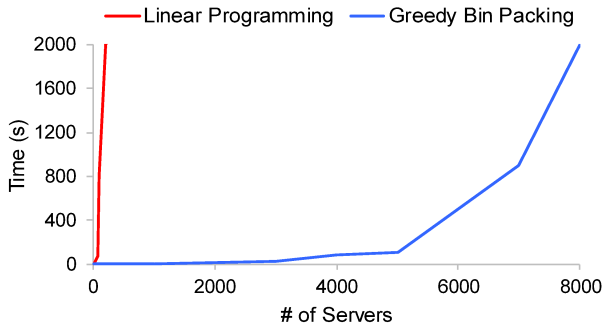


Figure 3: Computation time of linear programming and greedy bin packing for various network topology sizes.

speeded up with a faster server. However, the epoch length is still constrained by measurement frequency of traffic statistics. In the centralized design, the controller has to poll the hardware switches for traffic stats using protocols such as SNMP, sFlow and Openflow. The measurement frequency cannot be very high due to measurement overheads. For example, a hardware switch in our testbed only updates the traffic metrics for Openflow [25] protocol every 20 seconds [11]. That means the epoch length has to be 20 seconds or even more.

The distributed design is a promising approach to make the traffic consolidation more responsive to traffic variation in the DCN. DISCO [46] proposes a host level and link level distributed traffic consolidation. But they still use the centralized controller to implement the distributed designed, modeled as a number of sub-optimizers. We believe the granularity of deciding at the flowlet-level in DREAM is better than scheduling an entire flow in DISCO, as only a portion of the flow (i.e., a few flowlets) has to be rerouted when congestion occurs. This alleviates any potential traffic oscillation [19]. Another distributed design, REsPoNse [38], requires the full traffic history as prior knowledge to train their algorithm to

derive the always-on path offline, over which traffic is routed. But this depends on the assumption that the data center traffic pattern doesn't change significantly, which is not the case in real DCNs [6]. REsPoNse [38] relies on proactive link utilization feedback from routers to perform traffic scheduling, which is also difficult in current hardware switches. So, REsPoNse is only implemented in the CLICK software switch [27]. On the other hand, reusing ECN in DREAM is much more feasible for practical DCN deployment.

3 DREAM DESIGN

DREAM is a distributed traffic consolidation framework operating at a flowlet-level to achieve energy savings in data center networks. Its distributed and adaptive design makes it responsive to latency-sensitive applications and rapidly adjusts to bursty network traffic [21]. The flowlet-level scheduling by agents at endpoints takes advantage of the multiple paths potentially available in data center networks without any modifications to existing data center infrastructures. Flowlet-level scheduling with the much shorter decision-making epochs enables DREAM to fully utilize every energy saving opportunity, as the workload varies. Finally, DREAM has lower packet drop ratio and application-level latency by adopting ECN for flowlet path selection.

3.1 Overview

The design of DREAM is shown in Fig. 4. Instead of using a centralized controller to do data center wide traffic engineering [17, 42, 44], we use distributed agents at each end-system to work cooperatively to achieve energy saving in the DCN. The distributed agent at the servers is a shim layer between the host's protocol stack and the data center networking fabric. We use the Open vSwitch [31] virtual switch as an example of this shim layer, since it is widely used in virtualization environments for software-based switching and forwarding and network automation. The distributed design in DREAM has better scalability and is more responsive to bursty traffic. In addition, there is no need to modify host applications or the switches in the DCN.

For example, consider Fig. 4 where there are 4 paths ($P1$ to $P4$) between Host 1 and Host 2. To save energy in the DCN, we seek to consolidate traffic to a minimal number of paths, without violating performance (i.e., latency) requirements, and then turn off idle switches and links (e.g., the switch and links along path $P4$). At a high level, DREAM consolidates traffic to a part (e.g., 'left-most' for a Leaf-Spine topology) of the DCN topology. By 'left-most', we mean the active switches of a single layer at the structured DCN topology are chosen from left to right [17]. To avoid potentially large wake up times for OFF switches impacting flows, a few backup paths are reserved in the DCN topology, thus slightly over-provisioning DCN capacity. Path $P3$ in Fig. 4 works as a backup path.

Reducing network delay is very important for latency-sensitive applications. Similarly, avoiding packet drops is important to maintain TCP throughput [2, 3]. Hence, we leverage the commonly used ECN feedback for congestion response. We also use ECN to steer traffic to an alternate path when one is congested. Idle path such as $P4$ can be turned off to save DCN energy. But, there are still 3 active paths between Host 1 and Host 2 in Fig 4. When the end-systems are not the bottleneck, the ability to use multiple paths between a

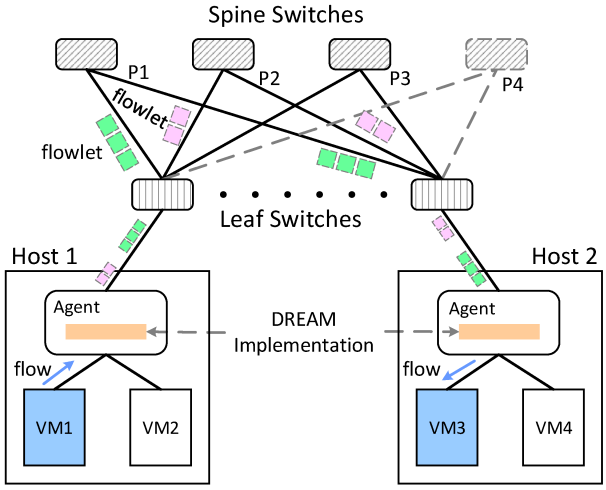


Figure 4: The design overview of DREAM.

sender-receiver pair can improve throughput [43]. DREAM splits a flow across multiple active paths in the DCN at the flowlet granularity. For example, we split the flow between VM1 and VM3 among active paths $P1$ and $P2$. A flowlet is a burst of packets of a flow [2], where the inter-flowlet time gap is much longer than the inter-arrival times of packets within a flowlet. The traffic sent on each DCN path is a group of flowlets rather than a larger flow. Thus, DREAM reduces the fragmenting of bandwidth capacity on each active path, thus improving multiplexing efficiency on each path (i.e., 'packing' more traffic).

At each distributed agent, we maintain a probability for selecting each active path. The path selection decision for an incoming flowlet is made based on the selection probability for each path. We dynamically change the path selection probability for each active path based on network conditions. For example, in a Leaf-Spine network, we indirectly consolidate traffic to the left-most paths by increasing the selection probability for the left-most path and reducing the selection probability for the right-most path. When ECN marks are received for traffic on a particular path, the selection probability for that path is reduced to mitigate congestion. Only a portion of flowlets are moved to different less-utilized path (that has a higher selection probability), rather than moving the entire flow, as used in prior distributed designs [38, 46]. This significantly reduces the oscillations [19] typically observed in distributed designs for DCN energy reduction.

Unlike previous centralized designs that do not consider the new arrivals of flows [17, 42, 44], DREAM explicitly deals with new arrivals of flows. DREAM first places a new flow on the right-most path (i.e., $P2$ in Fig. 4) and then gradually moves traffic to the left part of active DCN topology. This avoids congesting paths with existing flows and only migrates the new flows to the favored left-most paths as network conditions warrant. Previous works factor only existing flows as a variable in a linear programming formulation and do not deal with online flow arrivals.

Finally, we use packet encapsulation by the distributed agents to achieve explicit path control. In DREAM, the forwarding rules

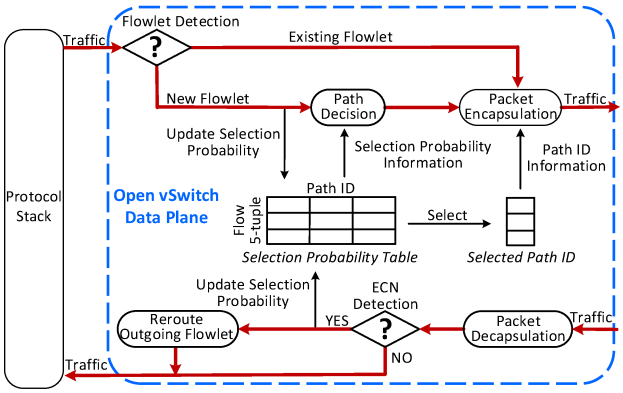


Figure 5: DREAM implementation in the data plane of Open vSwitch (OVS).

at hardware switches are set up by re-utilizing the schemes described in Xpath [18], with highly compressed rules to save Ternary Content-Addressable Memory (TCAM) resources in switches. The path IDs are inserted into the outer packet header by the distributed agent. Hardware switches forward packets accordingly, based on the path ID. For the sake of efficiency, we utilize the existing primitives in Open vSwitch for the packet encapsulation.

3.2 Distributed Agents

In DREAM, the distributed agent is implemented at the shim layer between host's protocol stack and the DCN's networking fabric, for the sake of easy deployment. Open vSwitch (OVS) has already implemented the Generic Routing Encapsulation (GRE) and VxLAN packet encapsulation for tunneling, which we utilize in DREAM. As all the traffic goes through the OVS data plane, it is an ideal place for the distributed agent of DREAM.

The basic scheduling unit in DREAM is a flowlet. Packet reordering is minimized if the idle interval between flowlets is more than the maximum difference between the delays across different paths [2]. Moreover, the TCP protocol stack has increasingly supported packet reordering to support multi-path, and we take advantage of it, when it does occur across flowlets.

In DREAM, we consolidate flowlets on a path until we observe ECN on that path. While ECN is usually used for network congestion feedback [3, 14, 32], we leverage it to steer packets to avoid congested paths in the DCN. ECN provides feedback of incipient congestion and thus reduces packet drops and latency by having ECN-enabled TCP connections respond to ECN marks. Hardware switches with ECN capability use an Active Queue Management (AQM) at the interface level to mark the *CE* bit, which is reflected back in the ECE bit of TCP ACK packets. Our distributed agent monitors the *ECE* bit in the TCP header to sense path congestion, and moves flowlets from congested paths to alternate under-utilized paths.

Fig. 5 shows the implementation of DREAM in the data plane of Open vSwitch. The probability of sending a flowlet on each possible path (columns) is maintained in a table for each flow identified by the 5-tuple (protocol type, src.IP, dst.IP, src.port and dst.port packet header fields). This selection probability is used to select the path ID which is stored as the Selected Path ID in an additional column. The selected path ID is for the currently identified flowlet of the flow. Note that each cell in the Selection Probability Table is a probability value but each cell in the column of Selected Path ID is an integer ID.

First, we introduce the workflow of the outgoing traffic from the host's protocol stack to the DCN. Based on the packet inter-arrival time, the Flowlet Detection module will decide if we have found a new flowlet. The packets of an existing flowlet will be directly encapsulated with path ID before sending it out on the wire. Upon finding a new flowlet, the selection probability of previously completed flowlet has to be updated. The details on how to change the selection probability are described in Section 3.3. We then choose a path for the new flowlet based on the path selection probability. As a last step, packets are encapsulated.

Second, the processing of incoming traffic from the DCN is shown at the bottom of Fig. 5. After decapsulating the packet, we check the ECN flags. If congestion has been detected on a path (indicated by the ECE bit of ACK packets), DREAM will update the selection probability of that path and reroute the flowlet to an alternate under-utilized path before forwarding the packet to the host's protocol stack. Otherwise, the incoming traffic is directly forwarded to the host's protocol stack.

3.3 Scheduling Algorithm

Whenever we detect a flowlet in DREAM, the distributed agent makes a decision to choose a path for this flowlet, with all the packets in the flowlet following the same path. DREAM schedules the flowlet on a path based on the selection probability, f_i for path i. We seek to set the value of f_i such that we don't observe ECN on the active routing path. In Fig. 6, we give the state machine for flowlet scheduling.

Over-utilized. We start our algorithm description with the system being in equilibrium, where the traffic is consolidated onto a few active paths and none of the paths see congestion (as observed with ECN). Traffic fluctuation is common in DCNs [6, 34]. When we sense that one of the active paths is congested, some traffic have to be moved to another under-utilized path. Unlike prior works [17, 42, 44], that move the entire TCP flow every time epoch, DREAM does this at the flowlet-level. This enables DREAM to be more responsive and dramatically eliminate traffic oscillation. A subset of the flowlets on a congested path are re-routed, thus alleviating congestion but also ensuring that there is no traffic oscillation [19]. Traffic are routed on uncongested paths by adapting the probability of sending the subsequent flowlets on those paths.

In DREAM, the moving average of ECN history is maintained for each path. e_{new} is the fraction of marked packets with ECN bits in the current time window. The ECN marking history e is smoothed using Exponentially Weighted Moving Average (EWMA) as $(1-\alpha)*e_{old}+\alpha*e_{new}$. α is the weight for new fraction of ECN markings. Thus, the distributed agents can react to link congestion at RTT timescales. Since the RTT in data center networks is usually a few microseconds [26], it enables DREAM to be very responsive. When the agent detects ECN on path i, the selection probability f_i is decreased by $X\%$. $X\%$ is defined as $f_i * e/2$.

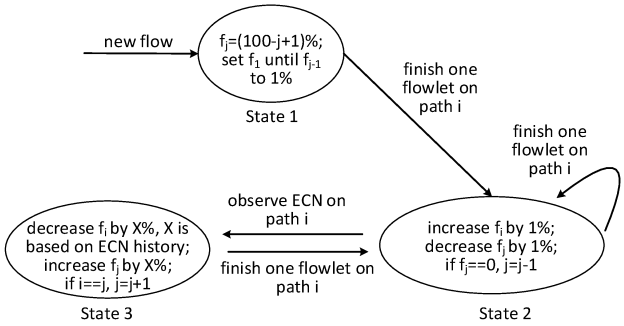


Figure 6: The state machine for flowlet scheduling.

Under-utilized. To save energy, we seek to consolidate traffic to one portion (e.g., left-most active paths) in the DCN. Let us consider a TCP flow with 4 groups of flowlets routing on 4 different active paths. The probability of sending the next flowlet on each of the paths is f_1, f_2, f_3 and f_4, respectively. For each path i (=1, 2, 3, 4), we increase the value of f_i if the distributed agent successfully sends a flowlet on path i without observing any congestion feedback signals. The probability increase follows the Additive Increase rule, i.e., increase f_i by a fixed amount (e.g., 1%). Since we guarantee that the sum of f_1, f_2, f_3 and f_4 is 100%, we correspondingly decrease the value of right-most active path (i.e., path 4) by 1%. In this design, flowlets are more likely to be routed on the left-most available active paths. One major concern is that the Additive Increase of f_i will finally result in queue build-up on path i. Then, we will revert to the case of over-utilized paths, in which we move a subset of flowlets to the under-utilized right most paths. The motivation of this design is to take advantage of every energy saving opportunity during the period when there is traffic variation and to react each traffic burst as well, just like TCP varies the congestion window.

New Arrivals. Unlike ElasticTree [17] and CARPO [42], which are offline solutions for a fixed set of flows, DREAM supports new flows for an online solution. Newly arriving flows are initially routed over the right-most active path j, to avoid congesting the heavily utilized left-most paths (1 to $j-1$) in the network. As the new flow's flowlets generate traffic, they gradually move to the left-most active paths (1 to $j-1$) at the probability of 1%, as long as these paths are not congested. If we successfully send a flowlet on path i, then we enter the under-utilized case for path i.

Finally, we show the state machine for our flowlet-level scheduling in Fig. 6. The state machine starts with the arrival of a new flow. First, we place all the flowlets on the right-most path j. The selection probability f_j for right-most active path j is (100-j+1)%. To gradually consolidate traffic to the left-most paths, we set f_1 until f_{j-1} to 1%. For any path i completing the transmission of one flowlet, we increase its probability value f_i by 1%. Correspondingly, the selection probability value of right-most path j decreases by 1%. When observing ECN feedback on path i, we enter state 3. Here, the distributed agent decreases the probability value f_i by X% depending on the smoothed value of the ECN feedback. After that, each probability value f except the right-most one f_j recovers its probability value to reach a dynamic balance.

Algorithm 1: Distributed Flowlet Scheduling

```
1  f_j = (100 − j + 1)%
2  while 1 ≤ i ≤ j − 1 do
3  |   f_i = 1%
4  end
5  while true do
6  |   if observe ECN on path i then
7  |   |   if i == j then
8  |   |   |   j = j + 1
9  |   |   end
10 |   |   f_i = f_i − X%
11 |   |   f_j = f_j + X%
12 |   end
13 |   if finish one flowlet on path i then
14 |   |   if f_i == 100% then
15 |   |   |   continue
16 |   |   end
17 |   |   if f_j == 0 then
18 |   |   |   j = j − 1
19 |   |   end
20 |   |   f_i = f_i + 1%
21 |   |   f_j = f_j − 1%
22 |   end
23 |   send packet and enforce path control
24 end
```

The pseudo code of this scheduling policy is given in Algorithm 1. In lines 1-4, we place the new arrival of flows on the right-most active path j. The path 1 to $j-1$ begin to enter the process of Additive Increase. The while loop between lines 5-24 is the send() function in the data plane of Open vSwitch. Whenever we observe ECN in line 6, the selection probability f_i is decreased. Lines 13-22 is for the Additive Increase of the selection probability. Prior work [10] on proving that the equilibrium point for Additive Increase Multiplicative Decrease (AIMD) can be reused to show that the algorithm will result in a stable operation. In our design, the throughput of TCP flow is only determined by the host side congestion window. Our distributed flowlet scheduling algorithm only adjusts the splitting ratio of flowlets among different active paths and consolidates the traffic to left-most paths.

3.4 Packet Encapsulation

In data centers, the switches locally decide the next hop using Equal Cost Multiple Path (ECMP) or Valiant Load Balancing (VLB) [15]. In energy-proportional networks, we need to route traffic over a minimal number of switches and links thus turning off idle networking components. For this, explicit path control is essential.

There are many approaches such as Myrinet [28] and Multi Protocol Label Switching (MPLS) [33] to achieve the explicit path control. MPLS uses distributed protocols such as CR-LDP [4] or RSVP-TE [5] to populate labels in the network to set policy-based paths, but can be complex [18]. Previous centralized traffic consolidation frameworks [17, 42] leverage the SDN controller to insert

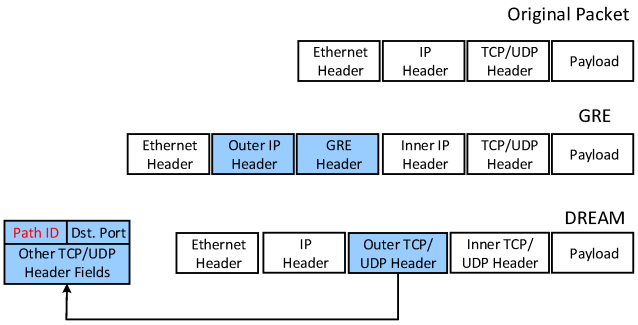


Figure 7: The packet encapsulation format in DREAM. Our encapsulation header is between the IP header and Inner TCP/UDP header. The original Src. port field of Outer TCP/UDP header is replaced by Path ID.

forwarding rules at the Openflow switches, but the refresh rate on the Openflow TX/RX counters in hardware switches can be a limiting factor for responsiveness.

Xpath [18] proposes an efficient way to do explicit path control in the DCN. It uses a compressed hierarchical path ID to save TCAM space in the switch. A data center operator pre-installs the forwarding rules at hardware switches based on this path ID. In our design, we use packet encapsulation at DREAM's distributed agents to achieve explicit path control by making path selection decisions and putting the path ID in the encapsulation header. Fig. 7 gives the packet format of our design. Similar to the existing encapsulation protocols such as GRE in Open vSwitch, DREAM's encapsulating packet header comes between the original IP header and the TCP/UDP header (shown as Inner TCP/UDP header). We directly copy the fields in inner TCP/UDP header to the outer TCP/UDP header, maintaining the rest of the layer 4 information, except for using the source port number in the outer header to store the path ID. Existing hardware switches match on this path ID, and require no modification. The original source port number is stored at inner TCP/UDP packet header which will be used by the end-system protocol stack after packet decapsulation at the destination server.

4 IMPLEMENTATION

4.1 Testbed

DREAM is implemented on a testbed shown on Fig. 8. In the testbed, we have 8 blade servers and 6 switches. The servers and switches are hosted at two different racks. In the network, we create a 2-layer leaf spine DCN topology. There are two types of switches in the network: HPE 3800 J9574A leaf switches and Arista 7050X-72Q spine switches. A pair of 2 servers as a group is connected to a leaf switch.

Each server has a 32 core AMD Opteron 6272 CPU, 64G memory, 1G Ethernet Network Interface Card (NIC), 50G SSD and 12 TB shared RAID file system. The TCP protocol on our CentOS 7.2 operating system is TCP Cubic. We also disable the Nagle algorithm on the operating system to have more accurate latency results. ECN is enabled in the Linux kernel. We take control of the host NIC by a modified version of Open vSwitch 2.4, in which we implement the

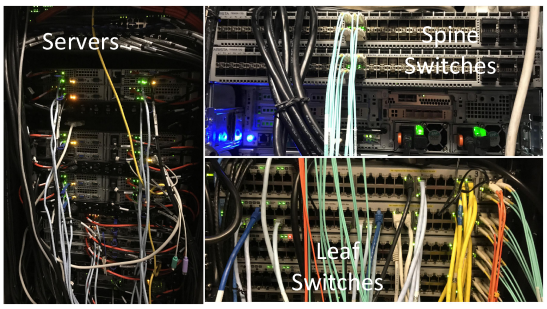


Figure 8: The servers and switches in DREAM.

distributed agents of DREAM. For Open vSwitch, most of codes of DREAM are implemented at the data plane module. We re-utilizes the sk_buffer primitives in Open vSwitch for efficient packet encapsulation and decapsulation. At the endpoints, flows are generated by a Python-based traffic generator with TCP Cubic connections.

On the switches, ECN is enabled. All the port queues share a memory of 12MB. Each queue can have up to 6.9MB memory. We set the minimum queue length threshold as 20 TCP segments and maximum threshold as 500 TCP segments. When the queue length is less than 20 TCP segments, no ECN bits are marked. When the queue length is between 20 segments and 500 segments, the ECN is marked probabilistically according to the queue length. Switches mark ECN bits on all the packets when the average queue length is larger than 500 TCP segments.

To implement the centralized approaches such as ElasticTree and CARPO, we use the ONOS [30] SDN controller to control all the software and hardware switches. There are 2 VLANs in the testbed to avoid affecting the accuracy of our measurement results. All the data traffic is routed over VLAN 20 and all the Openflow control messages are routed over VLAN 1. The Openflow protocol version we used is 1.3. The power consumption of switch is measured by using Watts Up power meter with a granularity of 0.01 watt.

4.2 Workloads

We use two application workloads that are typical of those observed in current data centers: Web service like Wikipedia and MapReduce as seen in Facebook to evaluate the benefit of DREAM.

Wikipedia Trace. The Wikipedia trace [36] contains about 280 Billion HTML page requests to Wikipedia database servers. As in [42], we extract all the requests to the English Wiki pages and group the requests into different prefix folders. For example, the URL https://en.wikipedia.org/wiki/American is grouped into folder A. Thus we have 61 folders (A-Z, a-z and 1-9). We assume that each folder is served by one database server. Thus, we have 61 flows in the virtual network. Three of them are shown on Fig. 9 (a) and (b). Fig. 9 (a) is the trace for 6 days and Fig. 9 (b) is the results for 60 seconds. The Wikipedia trace exhibits diurnal and day-of-week pattern over the 6 days.

Facebook MapReduce Trace. We use a MapReduce trace obtained from a 2009 Hadoop log file of a Facebook cluster [9]. The SWIM [9] benchmark allows us to replay jobs from the Hadoop log file and submit these jobs to actual clusters of machines. To get the

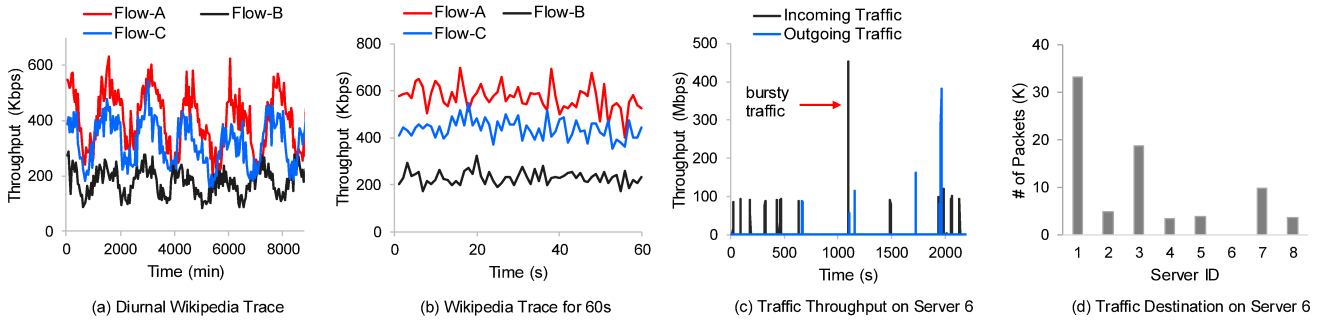


Figure 9: We use the Wikipedia web service [36] (part a and b) and Facebook MapReduce [9] (part c and d) trace to evaluate DREAM.

traffic matrix, we set up a 8 virtual machine cluster on a host with 64 cores and 64G memory. Each virtual machine has 2 CPU cores, 2G memory, a 140G hard disk and runs Ubuntu 14.04. In this cluster, we log the task information. Then we regenerate these tasks on our testbed using Python. The generated network traffic depends on the topology and environment we implement.

Fig. 9 (c) and (d) show the features of our MapReduce trace. The incoming and outgoing traffic at each slave node is negligible (about 4 Kbps) most of the time. However, as seen in Fig. 9 (c), there are very short time-scale spikes in the traffic. The burst in link throughput peaks for a few seconds, going up to 400Mbps. The burstiness of the trace comes from the shuffle phase in MapReduce framework. In Fig. 9 (d), we find that there is traffic between each pair of servers in the cluster.

5 EVALUATION RESULTS

In this section, we evaluate DREAM on the testbed described in Section 4.1. The Wikipedia web service [36] and the more bursty Facebook MapReduce trace [9] are used. For comparison, we also implement the baseline ECMP, as well as ElasticTree [17] and CARPO [42] designs. Baseline ECMP is implemented on a centralized ONOS [30] SDN controller. Both CARPO and ElasticTree are centralized approaches that periodically run a linear programming model to decide the routing paths for flows. We use GLPK programming language [24] to implement the linear programming model. The scheduling granularity for ElasticTree and CARPO is the single TCP flow-level.

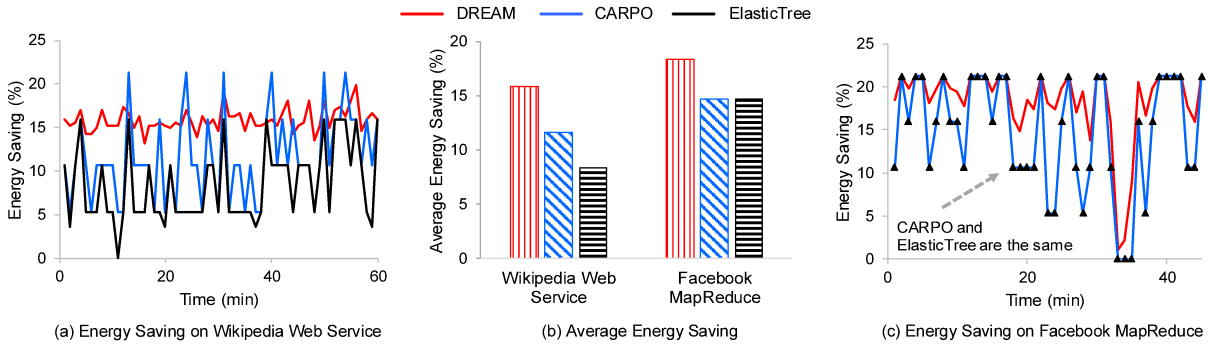
Due to the refresh rate of hardware stats in our testbed, which is 20 seconds, we set the epoch length for ElasticTree and CARPO at 25 seconds. CARPO considers the correlation between flows and avoids placing positively-correlated flows together compared to ElasticTree. Another major difference is that CARPO uses the 90th%tile of the last epoch to predict the flow's bit rate at the next epoch, while ElasticTree uses the peak value in the last epoch. In contrast, the epoch length for DREAM is at most one RTT as the hardware switch can mark ECN on each packet. Finally, we obtain the energy savings by using a Watts up power meter with granularity of 0.01 watt. The packet drop ratio is calculated by using the TX/RX drop counters at the switches and latency is measured at the application-level.

5.1 Energy Saving

We first compare DREAM with CARPO and ElasticTree with regard to the energy saving. All the energy saving results are normalized to the energy consumed with ECMP. In the first experiment, we evaluate DREAM and the other alternatives using the Wikipedia web service workload. Each server has 8 consistent flows to other servers in the testbed. Half of them are sent to the servers within the same Top of Rack (ToR) switch. The destination of remaining 4 flows are randomly selected outside the ToR switch. We use a traffic generator to generate the Wikipedia web service flows with TCP connections implemented on Linux. As the original Wikipedia trace has a low data rate for flows, we scale up its rate in our experiments, and utilize more than the Minimum Spanning Tree (MST) topology in the DCN.

The energy saving results on Wikipedia trace, as it varies over time (up to 60 minutes) is given on Fig. 10 (a) and the average results are shown on Fig. 10 (b). Among all the 3 frameworks, DREAM has the highest energy saving results throughout the experiment, except for 6 sample points. On average, the energy saving of DREAM is 15.9%. The improved energy saving with DREAM is due to it being more responsive to traffic variation because of its distributed design. When the traffic load ebbs, it immediately reconfigures the routing of flowlets and thus leaves more idle switches and links. On the other hand, the centralized CARPO and ElasticTree have longer 25 seconds epoch length. They miss a large number of opportunities to save energy as the traffic load decreases. The longer epoch length of the centralized approaches is constrained by the running time for the optimization and the refresh rate of TX/RX stats from the switches.

Of the alternatives, CARPO has better energy saving results compared to ElasticTree, because of its more aggressive data rate prediction and that it reserves less network bandwidth for future traffic variations. But this sacrifices congestion, reflected in packet drop rate and delay. On average, CARPO and ElasticTree save 11.6% and 8.6% DCN's energy, respectively. Only at a few time instants (note the six peaks), CARPO saves more energy compared to DREAM in Fig. 10 (a). Because the distributed agents in DREAM reroute flowlets to alternative paths while we observed ECN feedback as the flow's data rate increases. On the other hand, CARPO has to wait until the end of current epoch and wakes up an alternative path in next epoch. That is why CARPO saves more



(a) Energy Saving on Wikipedia Web Service (b) Average Energy Saving (c) Energy Saving on Facebook MapReduce

Figure 10: Because of good responsiveness and flowlet-level traffic scheduling, DREAM saves more DCN energy compared with CARPO and ElasticTree on both Wikipedia and Facebook trace.

energy at those six peaks. When the traffic ebbs at next epoch, CARPO reserves bandwidth based on large data rate predicted at last epoch, thus resulting in switches and links being left active and under-utilized.

The second experiment is to evaluate the schemes based on the more bursty Facebook MapReduce trace. In the MapReduce framework, the network throughput bursts only when the data shuffle phase occurs. Most of the time, traffic in the network are just the heartbeats between master and slave nodes. This is just a few Kbps in our trace. In order to have more traffic variability, we add the Wikipedia trace as background traffic in the second experiment. The energy saving results on the Facebook MapReduce trace is shown in Fig. 10 (c). Once again, DREAM saves the most DCN energy all the time. The average energy saving over 60 mins is 18.4%, which is 26% better than CARPO and ElasticTree. The average energy saving for CARPO and ElasticTree are similar, around 14%. Most of this is because of their poor responsiveness to bursty traffic.

5.2 Packet Drop

One major performance concern in traffic consolidation is that some links are heavily utilized while others are idle, to save energy. Higher link utilization increases the probability of packet drops and delay. Mitigating this requires scheduling schemes such as CARPO and ElasticTree to be able to reroute the flows quickly, especially for traffic bursts. Nevertheless, being responsive to bursty traffic is still challenging for CARPO and ElasticTree because of their long epoch length. In DREAM, we observe that queue build-up at switches mainly contributes to the high packet drop and delay. The ECN signal in our design detects the queuing at switches early and notifies the distributed agents before the queue at switches is full. At high link utilization, DREAM can reroute the bursty traffic in a timely manner to reduce packet drop and delay. Also, path redundancy in a DCN is exploited as we transmit the flowlets of one TCP connection across multiple active paths. In this section, we compare DREAM, CARPO and ElasticTree, in terms of packet drop rates. Then, we show the application-level latency results in the next section.

The average packet drop ratios are given on Fig. 11 (a). On both the Wikipedia web service and Facebook MapReduce trace, the packet drop ratio of DREAM is less than 0.01%, mainly because it adapts the flowlet transmission based on queue build-up information obtained from ECN feedback. In DREAM, traffic will be moved from a path if the queue length at switches on the path exceeds the threshold. On the Wikipedia web service workload, the packet drop probability for CARPO and ElasticTree is 1.23% and 0.85%, respectively. Compared with DREAM, they are much worse due to their poor responsiveness to traffic variation. CARPO has 44% worse packet drop ratio than ElasticTree because of its more aggressive data rate prediction for a flow. It uses the 90th-percentile data rate in last epoch to predict the rate rather than the peak value used in ElasticTree. On the Facebook MapReduce trace results, the packet drop ratio for CARPO and ElasticTree is around 0.2%. Although the average packet drop ratio results for CARPO and ElasticTree are lower compared with those on the Wikipedia trace, the packet drop ratio can be as high as 6.2% when the data shuffle phase in MapReduce happens. Because there is only heartbeat traffic in the network most of time. Packet drops don't occur for those heartbeats. The average packet drop ratio on Facebook MapReduce trace is lower (which is somewhat misleading), but the larger number of drops during the MapReduce data shuffle phase significantly impacts the application's performance.

5.3 Application-Level Latency

Finally, we show the application-level latency results in Fig. 11 (b), (c) and (d). In the experiments, we put the time stamp in the payload of packets originating from the sender of TCP flows. On the receiver side, our program echos the inserted time stamp without any additional processing or delay. Then the application-level perceived latency can be calculated simply as $(time_{new} - timestamp_{old})/2$.

Fig. 11 (b) is the CDF for application-level latency on Wikipedia web service trace. The average values are given on Fig. 11 (c). The latency in DCN mainly comes from the queuing time at switches. The baseline ECMP has the shortest latency results, with an average of 19.12 ms. Because of traffic consolidation in DREAM, CARPO and ElasticTree, all of them have longer latency than ECMP. The average latency for ElasticTree is 30.5 ms. CARPO has larger average latency result of 31.5 ms due to its aggressiveness. CARPO reserves a much smaller headroom bandwidth. In DREAM (which has an average of

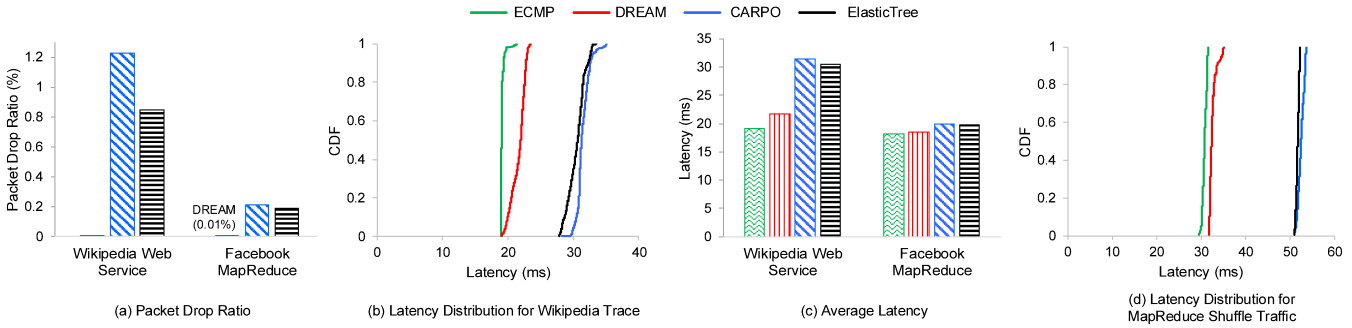


Figure 11: DREAM has the least average packet drop ratio on both Wikipedia web service trace and Facebook MapReduce trace. Part (b), (c) and (d) plot the application-level latency results. DREAM achieves the shortest latency among all the alternatives.

21.6 ms), we have early reaction to queue build-up by using ECN feedback. So, DREAM has 29% shorter average latency compared with ElasticTree.

The application-level latency results for the Facebook MapReduce trace are shown in Fig. 11 (c) and (d). As there is only a small amount of heartbeat traffic in the network most of time and all the schemes have the same active topology (i.e., the Minimum Spanning Tree), the average latency for ECMP, DREAM, CARPO and ElasticTree are almost the same in Fig. 11 (c). However, when the shuffle phase in Mapreduce happens, we observe the latency differences among the traffic consolidation alternatives. So, we focus on the CDF of the latency for MapReduce only during the shuffle phase in Fig. 11 (d). For bursty traffic, DREAM achieves at least 39% shorter latency compared to the other traffic consolidation schemes. Overall, the responsiveness and flowlet scheduling in DREAM produces much higher energy savings. The early reaction to queue build-up at switches by using ECN feedback reduces packet drops and application-level latency.

6 RELATED WORK

Gupta et al. [16] first proposes a position paper exploring possible opportunities to save Internet power. At the device level, idle components of a switch can be put into sleep. At the network level, traffic can be rerouted to a few links while letting other idle switches go to sleep. The following works [1] and [29] demonstrate that DCN can be energy-proportional at device level. GoogleP [1] combines flattened butterfly topology and rate adaption to dynamically change link speed and power consumption. Research [29] buffers packets at edge switches for a little while to have longer traffic gaps. In such a way, intermediate switches can have longer sleep time.

ElasticTree [17] proposes three centralized optimizers which consolidate traffic to the minimal subnet of the network and turn off the unused switches. Traffic consolidation in ElasticTree can have better power saving than device level techniques. However, it is impractical in production data centers because of its high computation complexity. GreenTE [44] proposes a heuristic algorithm to accelerate centralized traffic consolidation. EATe [39] solves a complex linear programming model at edge switch to consolidate traffic on pre-defined paths. Similarly, DISCO [46] has a few sub-optimizers in the centralized controller to mimic distributed traffic consolidation at per switch level. In REsPoNse [38], the author hopes to find energy-critical paths offline and schedule flows on those paths online. But it needs traffic history as prior knowledge. CARPO [42] considers traffic correlation while consolidating traffic. However, it still uses the complex centralized optimizer. Finally, FCTcon [45] designs the deadline aware traffic consolidation in DCN with network latency feedbacks.

7 CONCLUSION

We show in this paper that energy-aware traffic consolidation in DCNs is best performed in a distributed manner. State-of-the-art centralized traffic consolidation approaches have poor responsiveness to traffic variation in a DCN, especially for bursty traffic. The poor responsiveness results in much poorer energy savings, as well as higher packet drop rates and packet delays. Our distributed energy-aware traffic management framework, DREAM, consolidates traffic to a minimal portion of DCN at the flowlet granularity. Flowlet scheduling produces less fragmenting of the link capacity and allows for better energy savings. The distributed agent is implemented in Open vSwitch and utilizes ECN to sense the queue build-up at the intermediate switches. As reaction to ECN operates at RTT timescales, DREAM has good responsiveness to traffic variations and bursts. As a result, DREAM fully utilizes every energy saving opportunity and quickly reroutes flowlets to avoid packet drops and long delays. Testbed results using the Wikipedia trace and the Facebook MapReduce trace prove that DREAM on average saves at least 15.8% DCN energy, compared to existing schemes such as CARPO which saves only 11.6% and ElasticTree which saves 8.4% energy. The packet drop ratio in DREAM is less than 0.01% while the best among the alternatives, ElasticTree [17], has 0.19% drop ratio on Facebook trace and 0.85% drop ratio on Wikipedia trace. For application-level latency, DREAM achieves at least 30% shorter latency compared to the alternatives.

ACKNOWLEDGMENTS

This research was partly supported by NSF grants 1815643 and 1763929.

REFERENCES

[1] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. 2010. Energy Proportional Datacenter Networks. In *Proceedings of the 37th Annual*

International Symposium on Computer Architecture (ISCA '10). ACM, 338–347.

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed Congestion-aware Load Balancing for Datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. ACM, 503–514.

[3] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. ACM, 63–74.

[4] Loa Andersson, Ross Callon, Ram Dantu, and Paul Doolan. 2002. Constraint-Based LSP Setup using LDP. https://tools.ietf.org/html/rfc3212

[5] Daniel O. Awduche, Lou Berger, Der Hwa Gan, Tony Li, Vijay Srinivasan, and George Swallow. 2001. RSVP-TE: Extensions to RSVP for LSP Tunnels. https://tools.ietf.org/html/rfc3209

[6] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network Traffic Characteristics of Data Centers in the Wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. ACM, 267–280.

[7] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: Taking Control of the Enterprise. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '07)*. ACM, 1–12.

[8] Jeffrey D. Case, Mark Fedor, and Martin Lee Schoffstall. 1990. A Simple Network Management Protocol (SNMP). https://tools.ietf.org/html/rfc1157

[9] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. 2011. The Case for Evaluating MapReduce Performance Using Workload Suites. In *Proceedings of the 19th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '11)*. IEEE, 390–399.

[10] Dah-Ming Chiu and Raj Jain. 1989. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Comput. Netw. ISDN Syst.* 17, 1 (June 1989), 1–14.

[11] Hewlett-Packard Development Company. 2016. HP Switch Software OpenFlow v1.3 Administrator Guide. https://support.hpe.com/hpsc/doc/public/display?docId=emr_na-c04777809&docLocale=en_US

[12] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, 153–167.

[13] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113.

[14] Sally Floyd. 1994. TCP and Explicit Congestion Notification. *SIGCOMM Comput. Commun. Rev.* 24, 5 (Oct. 1994), 8–23.

[15] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM '09)*. ACM, 51–62.

[16] Maruti Gupta and Suresh Singh. 2003. Greening of the Internet. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '03)*. ACM, 19–26.

[17] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. 2010. ElasticTree: Saving Energy in Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI'10)*. USENIX Association, 17–17.

[18] Shuihai Hu, Kai Chen, Haitao Wu, Wei Bai, Chang Lan, Hao Wang, Hongze Zhao, and Chuanxiong Guo. 2015. Explicit Path Control in Commodity Data Centers: Design and Applications. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, 15–28.

[19] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. 2005. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '05)*. ACM, 253–264.

[20] Harshad Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast Analytical Power Management for Latency-critical Systems. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. ACM, 598–610.

[21] Fangming Liu, Jian Guo, Xiaomeng Huang, and John C. S. Lui. 2017. eBA: Efficient Bandwidth Guarantee Under Traffic Variability in Datacenters. *IEEE/ACM Trans. Netw.* 25, 1 (Feb. 2017), 506–519.

[22] Yanpei Liu, Stark C. Draper, and Nam Sung Kim. 2014. SleepScale: Runtime Joint Speed Scaling and Sleep States Management for Power Efficient Data Centers. In *Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA '14)*. IEEE Press, 313–324.

[23] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards Energy Proportionality for Large-scale Latency-critical Workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA '14)*. IEEE Press, 301–312.

[24] Andrew Makhorin. 2012. GNU Linear Programming Kit. https://www.gnu.org/software/glpk/

[25] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74.

[26] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, 537–550.

[27] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. 1999. The Click Modular Router. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles (SOSP '99)*. ACM, 217–231.

[28] Myrinet 1998. American National Standard for Myrinet-on-VME Protocol Specification. https://cms-docdb.cern.ch/cgi-bin/PublicDocDB/RetrieveFile?docid=2705&version=1&filename=Av26.pdf

[29] Sergiu Nedevschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, and David Wetherall. 2008. Reducing Network Energy Consumption via Sleeping and Rate-adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*. USENIX Association, 323–336.

[30] ONOS 2019. Open Network Operating System. https://onosproject.org/

[31] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. 2015. The Design and Implementation of Open vSwitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, 117–130.

[32] K. K. Ramakrishnan and Raj Jain. 1990. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Trans. Comput. Syst.* 8, 2 (May 1990), 158–181.

[33] Eric C. Rosen, Arun Viswanathan, and Ross Callon. 2001. Multiprotocol Label Switching Architecture. https://www.ietf.org/rfc/rfc3031.txt

[34] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, 123–137.

[35] Arman Shehabi, Sarah Josephine Smith, Dale A. Sartor, Richard E. Brown, Magnus Herrlin, Jonathan G. Koomey, Eric R. Masanet, Nathaniel Horner, Inês Lima Azevedo, and William Lintner. 2016. United States Data Center Energy Usage Report. https://eta.lbl.gov/publications/united-states-data-center-energy

[36] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. 2009. Wikipedia Workload Analysis for Decentralized Hosting. *Comput. Netw.* 53, 11 (July 2009), 1830–1845.

[37] Balajee Vamanan, Hamza Bin Sohail, Jahangir Hasan, and T. N. Vijaykumar. 2015. TimeTrader: Exploiting Latency Tail to Save Datacenter Energy for Online Search. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*. ACM, 585–597.

[38] Nedeljko Vasić, Prateek Bhurat, Dejan Novaković, Marco Canini, Satyam Shekhar, and Dejan Kostić. 2011. Identifying and Using Energy-critical Paths. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies (CoNEXT '11)*. ACM, Article 18, 12 pages.

[39] Nedeljko Vasić and Dejan Kostić. 2010. Energy-aware Traffic Engineering. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking (e-Energy '10)*. ACM, 169–178.

[40] Akshat Verma, Puneet Ahuja, and Anindya Neogi. 2008. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware (Middleware '08)*. Springer-Verlag New York, Inc., 243–264.

[41] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale Cluster Management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15)*. ACM, Article 18, 17 pages.

[42] Xiaodong Wang, Yanjun Yao, Xiaorui Wang, Kefa Lu, and Qing Cao. 2012. CARPO: Correlation-aware power optimization in data center networks. In *Proceedings of the 2012 IEEE International Conference on Computer Communications (INFOCOM '12)*. IEEE, 1125–1133.

[43] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*. USENIX Association, 99–112.

[44] Mingui Zhang, Cheng Yi, Bin Liu, and Beichuan Zhang. 2010. GreenTE: Power-aware traffic engineering. In *Proceedings of the 18th IEEE International Conference on Network Protocols (ICNP '10)*. IEEE, 21–30.

[45] Kuangyu Zheng and Xiaorui Wang. 2017. Dynamic Control of Flow Completion Time for Power Efficiency of Data Center Networks. In *Proceedings of the 37th IEEE International Conference on Distributed Computing Systems (ICDCS '17)*. IEEE, 340–350.

[46] Kuangyu Zheng, Xiaorui Wang, and Jia Liu. 2017. DISCO: Distributed traffic flow consolidation for power efficient data center network. In *Proceedings of the 2017 IFIP Networking Conference and Workshops (IFIP Networking '17)*. IEEE, 1–9.

[47] Liang Zhou, Chih-Hsun Chou, Laxmi N. Bhuyan, K. K. Ramakrishnan, and Daniel Wong. 2018. Joint Server and Network Energy Saving in Data Centers for Latency-Sensitive Applications. In *Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS '18)*. IEEE, 700–709.