Dynamic Switch-Controller Association and Control Devolution for SDN Systems

Xi Huang\(^1\), Simeng Bian\(^1\), Ziyu Shao\(^1\), Hong Xu\(^2\)
\(^1\)School of Information Science and Technology, ShanghaiTech University
\(^2\)NetX Lab @ City University of Hong Kong
Email: \{huangxi, biansim, shaozy\}@shanghaitech.edu.cn, henry.xu@cityu.edu.hk

Abstract—In software-defined networking (SDN), as data plane scale expands, scalability and reliability of the control plane have become major concerns. To mitigate such concerns, two kinds of solutions have been proposed separately. One is multi-controller architecture, i.e., a logically centralized control plane with physically distributed controllers. The other is control devolution, i.e., delegating control of some flows back to switches. Most of existing solutions adopt either static switch-controller association or static devolution, which may not adapt well to the traffic variation, leading to high communication costs between switches and controller, and high computation costs of switches. In this paper, we propose a novel scheme to jointly consider both solutions, i.e., we dynamically associate switches with controllers and dynamically devolve control of flows to switches. Our scheme is an efficient online algorithm that does not need the statistics of traffic flows. By adjusting some parameter \(V\), we can make a trade-off between costs and queue backlogs. Theoretical analysis and extensive simulations show that our scheme yields much lower costs and latency compared to static schemes, and balanced loads among controllers.

I. INTRODUCTION

Software-defined networking (SDN) holds great promises to improve network performance and management. The key idea of SDN is to decouple the control plane from the data plane [1]. Data plane can focus on performing basic functionalities such as packet forwarding at high speed, while the logically centralized control plane manages the network. Usually, switches at data plane send requests to control plane for processing some flow events, e.g., flow-install events.

The control plane is a potential bottleneck of SDN in terms of scalability and reliability. As the data plane expands, control plane may not be able to process the increasing number of requests if implemented with a single controller, resulting in unacceptable latency to flow setup. Reliability is also an issue since a single controller is a single point of failure, resulting in disastrous break-down of the control plane and the network.

Existing proposals to address such problems fall broadly into two categories. One is to implement the control plane as a distributed system with multiple controllers [2] [3]. Each switch then associate with certain controllers for fault-tolerance and load balancing [4] [5] [6] [7]. The second approach is to devolve partial loads of request processing from controllers to switches [8] [9] [10], reducing the workload of controllers.

For switch-controller association, the first category of solution, an obvious design choice is static switch-controller association [2] [3]. However, such static association may result in overloading of controllers and increasing flow setup latency due to its inflexibility to handle traffic variations. An elastic distributed controller architecture is proposed in [5], with an efficient protocol to migrate switches across controllers. However, it remains open how to determine the switch-controller association. Then [6] takes it one step further by formulating the switch-controller association problem as a deterministic optimization problem, i.e., an integer linear problem with prohibitively high computational complexity. A local search algorithm is proposed to find suboptimal associations within a given time limit (e.g., 30 seconds). In [7], the controller is modeled as an M/M/1 queue (Poisson arrival and exponential service). Under such an assumption, the switch-controller association problem with a steady-state objective function is formulated as a many-to-one stable matching problem with transfers. Then a novel two-phase algorithm is proposed to connect stable matching with utility-based game theoretic solutions, i.e., coalition formation game with Nash stable solutions.

For control devolution, the second category of solution, an obvious design choice is static devolution for certain functions and flows [8] [9] [10]; i.e., switches locally process requests that do not require network-wide state, such as link-layer discovery service. Static devolution mitigates the loads on control plane in some ways, but it could be inflexible in face of traffic variations. As an alternative, dynamic devolution allows switches to decide processing those requests locally or uploading them to control plane, depending on their amounts of loads at the moment. However, the design of dynamic devolution with respect to traffic variations remains open.

Several interesting questions thus arise, whose answers can potentially shape the design of SDN networks:

- Instead of deterministic switch-controller association with infrequent re-association [6] [7], can we directly perform dynamic switch-controller association in response to traffic variations? What is the benefit that we can obtain from a fine-grained control at the request level?
- How to perform dynamic devolution?
- How to make a trade-off between dynamic switch-controller association and dynamic control devolution?

In this paper, we consider a general SDN network with traffic variations, resulting in variations of requests that need to be handled. We assume each request can be either processed at switch (incurs computation costs) or be uploaded to certain controllers (incurs communication costs)\(^1\). We aim at reducing the computational loads at control plane by performing control devolution at data plane, reducing the communication cost by switch-controller association between data plane and control plane, and reducing the response time experienced by switches, which is mainly caused by queuing delay on controllers. Under such settings, we provide a new perspective and a novel scheme to answer those questions.

Our key results and contributions are summarized as follows:

\(^1\)The scenario that some requests can only be processed by controller is a straightforward extension of our model.
• **Problem Formulation**: To the best of our knowledge, this paper is the first to study the joint optimization problem of dynamic switch-controller association and dynamic control devolution.

• **Finer Granularity Control**: To the best of our knowledge, this paper is the first to perform the control decisions at the granularity of request-level. Note that request-level information such as time-varying queue backlog sizes and number of request arrivals present the actual time-varying state of data plane. Hence it helps for more accurate decision making of dynamic association and dynamic devolution when compared to coarse-grained control.

• **Online Algorithm Design**: We formulate a stochastic network optimization problem, aiming at minimizing the long-term average sum of communication cost and computational cost, while keeping time-average queue backlogs of both switches and controllers small\(^2\). By employing Lyapunov drift technique [11] and exploiting sub-problems structure, we develop an efficient greedy algorithm to achieve optimality asymptotically. Our algorithm is online, which means it does not need the statistics of traffic workloads and does not need the prior assumptions of traffic distribution.

• **Algorithm Analysis**: We show that our algorithm yields a tunable trade-off between \(O(1/V)\) deviation from minimum long-term average communication and computational cost and \(O(V)\) bound for average queue backlogs. We also find that the positive parameter \(V\) determines the switches’ willingness of uploading requests to controllers, i.e., performing switch-controller association.

• **Evaluation**: We conduct large-scale trace-driven simulations to evaluate the performance of our algorithm within two widely adopted data center networking topologies, i.e., canonical 3-tiered topology and Fat-tree topology. Simulation results verify the effectiveness and the trade-off of our algorithm. In addition, in the extreme case without control devolution, we compare our dynamic association scheme with other association schemes including Static, Random, and JSQ (Join-the-Shortest-Queue). The results verify the advantages of our scheme.

We organize the rest of paper as follows. We present the basic idea and formulation in Section II. Then we show our algorithm design and corresponding performance analysis in Section III. In Section IV, we present simulation results. We conclude this paper in Section V. Due to the page limit, all proofs and more simulation results are provided in our technical report [12].

### II. PROBLEM FORMULATION

In this section, we first provide a motivating example for the dynamic switch-controller association and dynamic control devolution. Then we introduce the system model and formulate the problem.

#### A. Motivating Example

The example of dynamic association and devolution is shown in Fig. 1.(a). In this example, there are 3 switches \(s_1, s_2, s_3\) and 2 controllers \(c_1, c_2\), and 1 global scheduler. Each switch or controller maintains a queue that buffers requests. During each time slot, each switch can serve 2 requests while each switch can serve only 1 request. There is a communication cost per request if switches upload requests to controllers, and a computational cost (2 per request on each switch) from local processing by switches themselves. At the beginning of time slot \(t\), \(s_1, s_2, s_3\) generates 3, 2, and 2 requests, respectively. The scheduler then collects system dynamics and decides a switch-controller association (could be (b) or (c)), aiming at minimizing the sum of communication cost (could be the number of hops, RTTs, etc.) and computational cost, as well as maintaining small queue backlog size. Each switch chooses to either locally process its requests or send them to controllers according to the scheduling decision.

Fig. 1. An example that shows the request-level scheduling process. There are 3 switches \((s_1, s_2, s_3)\), 2 controllers \((c_1, c_2)\), and 1 global scheduler. Each switch or controller maintains a queue that buffers requests. During each time slot, each switch can serve 2 requests while each switch can serve only 1 request. There is a communication cost per request if switches upload requests to controllers, and a computational cost (2 per request on each switch) from local processing by switches themselves. At the beginning of time slot \(t\), \(s_1, s_2, s_3\) generates 3, 2, and 2 requests, respectively. The scheduler then collects system dynamics and decides a switch-controller association (could be (b) or (c)), aiming at minimizing the sum of communication cost (could be the number of hops, RTTs, etc.) and computational cost, as well as maintaining small queue backlog size. Each switch chooses to either locally process its requests or send them to controllers according to the scheduling decision.

\(^2\)By applying Little’s law, small queue backlog implies small queuing delay or short response time.
\( j \in C \) maintains a queue backlog \( Q_i(t) \) that buffers requests from data plane. We denote \([Q_1(t), \ldots, Q_{\left| C \right|}(t)]\) as \( Q^{(1)}(t) \) and \([Q_i(t), \ldots, Q_{\left| S \right|}(t)]\) as \( Q^{(i)}(t) \). We use \( Q(t) \) to denote \([Q^{(1)}(t), Q^{(i)}(t)]\).

At the beginning of time slot \( t \), each switch \( i \in S \) generates some amounts \( 0 \leq A_i(t) \leq a_{\text{max}} \) of requests. Then each switch could choose to process its requests either locally or process the devoluted requests, while each controller \( j \in C \) has an available service rate \( 0 \leq B_j(t) \leq b_{\text{max}} \). We denote \([A_1(t), \ldots, A_{\left| S \right|}(t)]\) as \( A(t) \), \([B_1(t), \ldots, B_{\left| C \right|}(t)]\) as \( B(t) \), and \([U_1(t), \ldots, U_{\left| S \right|}(t)]\) as \( U(t) \). For \( i \in S \) and \( j \in C \), we assume that all \( A_i(t) \), \( B_j(t) \), and \( U_i(t) \) are i.i.d.; besides, their first and second raw moments are all finite.

Then the scheduler collects system dynamics information \((A(t), B(t), Q(t))\) during current time slot and makes a scheduling decision, denoted by an association matrix \( X(t) \in \{0, 1\}^{\left| S \right| \times \left| C \right|} \). Here \( X(t)_{i,j} = 1 \) if switch \( i \) will be associated with controller \( j \) during current time slot and 0 otherwise. An association is feasible if it guarantees that each switch is associated with at most one controller during each time slot. We denote the set of feasible associations as \( A \),

\[
A = \left\{ X \in \{0, 1\}^{\left| S \right| \times \left| C \right|} \mid \sum_{j \in C} X_{i,j} \leq 1 \text{ for } i \in S \right\}
\]

According to the scheduling decision, each switch \( i \) sends its request to controller \( j \) if \( X_{i,j} = 1 \). However, if \( \sum_{i \in S} X_{i,j} = 0 \), switch \( i \) appends its requests to local queue backlog. Then both switches and controllers serve as many requests in their associated with at most one controller during each time slot. An association is feasible if it guarantees that each switch is associated with at most one controller during each time slot.

We define the one-time-slot computational cost as

\[
x(t) = g(X, A(t)) \triangleq \sum_{i \in S} \alpha_i \left( 1 - \sum_{j \in C} X_{i,j} \right) \cdot A_i(t) \quad \text{ (5)}
\]

Given a series of associations \( \{X_0, X_1, \ldots, X_{t-1}\} \), the time-average expectation of computational cost is

\[
\hat{g}(t) \triangleq \frac{1}{t} \sum_{\tau=0}^{t-1} E \{x(\tau)\} \quad \text{ (6)}
\]

3) Queueing Stability: In this paper, we say that \( Q^*(t) \) is stable if

\[
\lim_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E \{Q_j^*(\tau)\} < \infty \text{ (7)}
\]

Likewise, \( Q^e(t) \) is stable if

\[
\lim_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} E \{Q_j^e(\tau)\} < \infty \text{ (8)}
\]

Queueing stability implies that both switches and controllers would process buffered requests timely, so that queueing delay is controlled within a small range.

Consequently, our problem formulation is given as follows

\[
\begin{align*}
\text{Minimize} & \quad \lim_{t \to \infty} \sup_{t \in \{0,1,2,\ldots\}} \left( \frac{1}{t} \sum_{\tau=0}^{t-1} E \{Q_j^*(\tau)\} \right) \\
\text{subject to} & \quad (2), (3), (9), (8)
\end{align*}
\]

III. ALGORITHM DESIGN AND PERFORMANCE ANALYSIS

To design a scheduling algorithm that solves problem (10), we adopt the Lyapunov optimization technique in [11]. We then employ the concept of opportunistically minimizing an expectation in [11], and we transform the long-term stochastic optimization problem (10) into the following drift-plus-penalty minimization problem at every time slot \( t \).

\[
\begin{align*}
\text{Minimize} & \quad V \cdot \left( \hat{f}(t, A(t)) + \hat{g}(X, A(t)) \right) + \sum_{j \in C} \sum_{l \in S} X_{i,j} \cdot A_i(t) + \sum_{i \in S} Q_i^e(t) \cdot \left( 1 - \sum_{j \in C} X_{i,j} \cdot A_i(t) \right) \quad \text{ (11)}
\end{align*}
\]

where \( V \) is a positive constant that makes the tradeoff between the total costs and total queue backlog size.

\footnote{For the intermediate derivation, we refer interested readers to our technical report [12].}

\footnote{The communication cost can be the number of hops or round trip times (RTT).}

\footnote{Note that we do not fix the serving principle, which can be FIFO, LIFO, etc. But in our simulation, we use FIFO.}
Then we design an optimal greedy algorithm\(^6\) that solves (12) and focuses on minimizing the second term of (12) only. Given \((i, j) \in S \times C\), we define

\[
\omega(i, j) = V W_{i,j} + Q_j^*(t) - V \alpha_i - Q_i^*(t) \tag{13}
\]

Then we design an optimal greedy algorithm\(^6\) that solves (12) as follows:

**Algorithm 1 Greedy Scheduling Algorithm**

**Input:** During time slot \(t\), the scheduler collects queue lengths information from individual controllers and switches, i.e., \(Q_i(t), Q_j^*(t)\), and \(A(t)\)

**Output:** A scheduling association \(X' \subseteq S \times C\)

1. Start with an empty set \(X' \leftarrow \emptyset\)
2. for each switch \(i \in S\) do
3. Split all controllers \(C\) into two sets \(I_1^j\) and \(I_2^j\), where
   \[J_1^j = \{j \in C \mid \omega(i, j) > 0\}\] and
   \[J_2^j = \{j \in C \mid \omega(i, j) \leq 0\}\]
4. if \(J_2^j = \emptyset\), then skip current iteration.
5. if \(J_2^j \neq \emptyset\), then choose controller \(j^* \in J_2^j\) such that
   \[j^* = \arg \min_{j \in C} \omega(i, j)\]
6. \(X' \leftarrow X' \cup \{(i, j^*)\}\)
7. end for
8. return \(X'\)

According to \(X'\), switches upload requests to controllers or append requests to their local queues. Then controllers and switches update their queue backlogs as in (2) and (3) after serving requests.

Remarks:
- Our algorithm is greedy since it greedily associates each switch with controllers that either have small queue backlog size or close to the switch, and otherwise it leaves all requests locally processed.
- For switch \(i\), given any controller \(j\) far enough from \(i\), i.e., \(W_{i,j} > \alpha_i\), switch \(i\) decides to upload requests to \(j\) only if \(\omega(i, j)\) is non-positive and smaller than any other. This requires switch \(i\) itself holds enough requests locally, i.e., \(Q_j^*(t) \geq V \cdot (W_{i,j} - \alpha_i) + Q_i^*(t)\). Then it will upload requests. Thus smaller \(V\) will invoke more effectively the willingness of switch \(i\) to upload requests to control plane.
- On the other hand, given any controller \(k\) close to switch \(i\), i.e., \(W_{i,k} < \alpha_i\), switch \(i\) will process requests locally if control plane holds large amounts of requests, i.e., \(Q_k^*(t) < V \cdot (W_{i,k} - \alpha_i) + Q_i^*(t)\). Thus given any large \(V\), controllers will have to hold great loads of requests before switches become willing to process requests locally.
- Therefore, the parameter \(V\) actually controls switches’ willingness to upload requests to controllers, i.e., performing switch-controller association. In other words, it controls the trade-off between communication cost and the computational cost incurred by uploading requests to control plane and locally processing, respectively.

Now we turn to time complexity analysis of our algorithm. Within each time slot, the algorithm runs \(|S|\) iterations in total. For each switch \(i\), it takes \(O(|C|)\) steps to split \(C\) into two disjoint sets. If the resulting \(J_2^j \neq \emptyset\), then the algorithm needs to calculate \(\omega(i, j)\) for each controller \(j\) (controllers in total) and picks up \(j^*\) for the minimum \(\omega(i, j)\). For each switch-controller pair \((i, j)\), calculating \(\omega(i, j)\) incurs only constant time according to (13). Therefore, during each time slot, our algorithm takes about \(O(|S| \times |C|)\) time to decide the optimal switch-controller association. In fact, our algorithm can also run in a parallel manner: i.e., for each switch \(i\), given \(W\), \(Q_j^*(t), Q_i^*(t),\) and \(A_i(t)\), the scheduler can decide its associated controller independently from other switches.

Next we characterize the performance of our algorithm. We suppose \(g^*\) and \(f^*\) are the infimum time-average computational cost and communication cost we want to achieve, respectively. We also suppose \(d_{\text{max}} = \max_{i,j} (E(B_j^i(t)), E(U_j^i(t)), E(A_j^i(t)))\). We have the following theorem on the \(O(V)\) \(O(V)\) trade-off between costs and queue backlogs:

**Theorem 1:** Given the parameters \(V > 0, \epsilon > 0\), and constant \(K = \frac{1}{2} d_{\text{max}} \cdot (|C| + |S| + |S|^2)\), then the queue vector process \(Q(t)\) is stable; besides, time-average expectation of communication cost and computational cost, as well as queue backlogs on switches and controllers satisfy:

\[
\limsup_{t \to \infty} \left( \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{j \in C} E \left\{ Q_j^*(\tau) \right\} + \sum_{i \in S} E \left\{ Q_i^*(\tau) \right\} \right) \leq K + \frac{V \cdot (f^* + g^* + K)}{\epsilon} \tag{14}
\]

IV. SIMULATION RESULTS

A. Basic Settings

**Topology:** In this section, we evaluate our Greedy scheduling algorithm under two widely adopted topologies: Canonical 3-Tiered topology and Fat-tree [13] topology. We show two of their instances in Fig. 2 and Fig. 3, respectively.

To make our performance analysis comparable in both topologies, we construct a Fat-tree network and a Canonical 3-Tiered network with almost the same number of switches. Regarding the Canonical 3-Tiered topology, we set the number of core switches as 26. Accordingly, the total number of switches is 702. Regarding the Fat-tree topology, we set the port number as 24 and thusly there are 720 switches in total. Note that the two resulting topologies are also comparable to the size of commercial data centers [14].

In both topologies, we deploy controllers on the hosts (one controller for every two pods\(^7\)), which are denoted by the blue circles in Fig. 2 and Fig. 3.

**Traffic Workloads:** We conduct trace-driven simulations, where the flow arrival process on each switch follows the distribution of flow inter-arrival time in [14] with mean 1700 μs.

\(^7\)See the proof in the appendix of our technical report [12].

\(^6\)For more detail, please refer to our technical report [12].
which is drawn from measurements in real-world data centers. We then set the length of each time slot as 10ms. Accordingly, the average flow arrival rate on each switch is about 5.88 flows per time slot.

In fact, there do exist hot spots within pods in real-world data center networks, where the switches have significantly high flow arrival rates. In our simulation, we pick the first pod as a hot spot and all switches there have a significantly high flow arrival rate, i.e., 200 flows per time slot. As for controllers, we set their individual capacity as 600 flows per time slot. That is consistent with the capacity of a typical NOX controller [15].

**Costs:** Given any network topology, we define the communication cost \( W_{i,j} \) between switch \( i \) and controller \( j \) as the length (number of hops) of shortest path from \( i \) to \( j \). Then we set a common computation cost \( \alpha \) for all switches, which equals to the average hop number between switches and controllers of its underlying topology. In Fat-tree topology, \( \alpha = 4.13 \); while in 3-Tiered topology, \( \alpha = 4.81 \).

### B. Evaluation of Greedy Algorithm

Fig. 4 (a) presents how the summation of long-term average communication cost and computational cost changes with different \( V \) in Fat-tree and 3-Tiered topologies. As \( V \) varies from 0 to \( 3.0 \times 10^7 \), we can observe that the total cost goes down gradually. This is consistent with our previous theoretic analysis. The intuition behind such decline is as follows. Since \( V \) controls the switches’ willingness of uploading requests. For switches that are close to controllers (their communication cost is less than the average), large \( V \) makes them unwilling to process requests locally unless the controllers get too heavy load. As \( V \) increases, those switches will choose to upload requests to further reduce the costs since for those switches, communication costs are less than the computation cost. Another observation we make is that the total cost of 3-Tiered topology is more than Fat-tree’s. The reason is that 3-Tiered has a higher computational cost (\( \alpha = 4.81 \) compared to 4.13) and it cost more when switches process requests locally.

Fig. 4 (b) shows the curve of total queue backlog size with different values of \( V \). From the figure, we notice that total queue backlog size increases until \( V \) reaches about \( 0.75 \times 10^7 \) and \( 1.5 \times 10^7 \) in Fat-tree and 3-Tiered topologies, respectively. This is also consistent with the \( O(V) \) queue backlog size bound in (14). Recall our analysis in **Total Cost:** larger \( V \) invokes most switches to spend more time uploading requests to control plane. However, control plane’s service capacity is fixed and requests will keep accumulating. Thus when \( V \) becomes sufficiently large, control plane will eventually hold most of requests in the system. This explains the increasing queue backlog size in Fig. 4 (b).

### C. Comparison with Other Dynamic Association Schemes

In this subsection, we consider the extreme case by setting common computational cost \( \alpha = 2.0 \times 10^{28} \) for all switches. This means the cost of local processing requests are prohibitively high and each time slot switches choose to upload requests to controllers. Thus our greedy algorithm degenerates into a dynamic switch-controller association algorithm. We compare its performance with three other schemes: **Static, Random and JSQ (Join-the-Shortest-Queue).** In static scheme, each switch \( i \) chooses the controller \( j \) with minimum communication cost \( W_{i,j} \) and then fixes this association for all time slots. In random scheme, the scheduler randomly picks up a controller for each switch at each time slot. In JSQ scheme, it randomly picks up one switch \( i \) without replacement round by round until all switches have chosen the target controllers. At each round, the selected switch \( i \) chooses the controller \( j \)
Fat-tree topology remains unchanged. Both schemes under Canonical 3-Tiered topology and Fat-tree topology, respectively.

Fig. 6 presents a comparison among the four schemes in terms of communication cost under Canonical 3-Tiered topology and Fat-tree topology, respectively. First, the communication cost under Static is the minimum among all schemes, which is consistent with its goal of minimizing the overall communication cost. Greedy cuts down the communication cost with increasing $V$. Eventually, when $V$ is sufficiently large, communication cost stops decreasing and remains unchanged. Both Random and JSQ exhibit much higher communication costs, compared to Static and Greedy.

Fig. 7 presents a comparison among the four schemes in terms of the variance of queue backlog size under Canonical 3-Tiered topology and Fat-tree topology, respectively. In fact, smaller queue backlog size variance indicates better capability of load balancing. The variance of Static grows exponentially with time, showing that Static is incompetent in load balancing. The reason is that Static greedily associates switches with their nearest controllers, ignoring different controllers’ loads. When it comes to Random and JSQ, the variance is almost 0, which shows the two schemes’ advantage in load balancing. While the variance of Greedy is in between the other three: it increases at the beginning and then remains stable after about thousands of time slots.

Fig. 7 presents a comparison among the four schemes in terms of the average queue backlog size under Canonical 3-Tiered topology and Fat-tree topology, respectively. The observations are very similar to those from Fig. 6.

In summary, among four schemes, Static is on the one end of performance spectrum: it minimizes communication cost while incurring extremely large queue backlogs; both Random and JSQ are on the other end of performance spectrum: they minimize the average queue backlog while incurring much large communication costs. In contrast, our Greedy scheme achieves a trade-off between minimization of communication costs and minimization of queue backlogs. Through a tunable parameter $V$, we can achieve different degrees of balance between cost minimization and latency (queue backlog) minimization.

V. CONCLUSION

In this paper, we studied the joint optimization problem of dynamic switch-controller association and dynamic control devolution for SDN networks. We formulated the problem as a stochastic network optimization problem, aiming at minimizing the long-term average cost of both communication and computational cost while maintaining low time-average queue backlogs. We proposed an efficient online greedy algorithm, which yields a long-term average sum of communication cost and computational cost within $O(1/V)$ of optimality, with a trade-off in an $O(V)$ queue backlog size for any positive control parameter $V$. Extensive simulation results showed the effectiveness and optimality of our online algorithm, and the ability to maintain a tunable trade-off compared to other dynamic association schemes.

ACKNOWLEDGMENT

This work was supported in part by NSFC grant No.61302114, and Hong Kong RGC CRF-C7036-15G.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008.
[2] T. Koponen, M. Casado, N. Gude, J. Stibring, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hamu et al., “Onix: A distributed control platform for large-scale production networks.” in Proceedings of OSDI, 2010.
[3] A. Tootoonchian and Y. Ganjali, “Hyperflow: A distributed control plane for openflow,” in Proceedings of the 2010 internet network management conference on Research on enterprise networking, 2010.
[4] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, “Logically centralized?, state distribution trade-offs in software defined networks,” in Proceedings of ACM HotSDN, 2012.
[5] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, “Towards an elastic distributed sdn controller,” in Proceedings of ACM HotSDN, 2013.
[6] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, “Pratyastha: An efficient elastic distributed sdn control plane,” in Proceedings of ACM HotSDN, 2014.
[7] T. Wang, F. Liu, J. Guo, and H. Xu, “Dynamic sdn controller assignment in data center networks: Stable matching with transfers,” in Proceedings of IEEE INFOCOM, 2016.
[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “Devflow: scaling flow management for high-performance networks,” in Proceedings of ACM SIGCOMM, 2011.
[9] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications,” in Proceedings of ACM HotSDN, 2012.
[10] K. Zheng, L. Wang, B. Yang, Y. Sun, Y. Zhang, and S. Uhlig, “Lazycell: Scalable network control for cloud data centers,” in Proceedings of IEEE 35th International Conference on Distributed Computing Systems (ICDCS), 2015.
[11] M. J. Neely, “Stochastic network optimization with application to communication and queueing systems,” Synthesis Lectures on Communication Networks, vol. 3, no. 1, pp. 1–211, 2010.
[12] X. Huang, S. Bian, Z. Shao, and H. Xu, “Dynamic switch-controller association and control devolution for sdn systems,” 2016, https://arxiv.org/abs/1702.03065.
[13] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” in Proceedings of ACM SIGCOMM, 2008.
[14] T. Benson, A. Akella, and D. A. Maltz, “Network traffic characteristics of data centers in the wild,” in Proceedings of ACM IMC, 2010.
[15] A. Tootoonchian, S. Gorbonov, Y. Ganjali, M. Casado, and R. Sherwood, “On controller performance in software-defined networks,” in Proceedings of ACM Hot-ICE, 2012.