Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Packet-in request redirection: A load-balancing mechanism for minimizing control plane response time in SDNs☆

Rui Xia [a], Haipeng Dai [a,*], Jiaqi Zheng [a,*], Hong Xu [b], Meng Li [a], Guihai Chen [a,*]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210023, China
[b] Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong Special Administrative Region of China

ARTICLE INFO

Keywords:
Software defined networking (SDN)
Distributed control plane
Load balancing
Lyapunov optimization
Approximation algorithm

ABSTRACT

A distributed control plane is more scalable and robust in software defined networking. This paper focuses on controller load balancing using packet-in request redirection, that is, given the instantaneous state of the system, determining whether to redirect packet-in requests for each switch, such that the overall control plane response time (CPRT) is minimized. To address the above problem, we propose a framework based on Lyapunov optimization. First, we use the drift-plus-penalty algorithm to combine CPRT minimization problem with controller capacity constraints, and further derive a non-linear program, whose optimal solution is obtained with brute force using standard linearization techniques. Second, we present a greedy strategy to efficiently obtain a solution with a bounded approximation ratio. Third, we reformulate the program as a problem of maximizing a non-monotone submodular function subject to matroid constraints. We implement a controller prototype for packet-in request redirection, and conduct trace-driven simulations to validate our theoretical results. The results show that our algorithms can reduce the average CPRT by 81.6% compared to static assignment, and achieve a 3× improvement in maximum controller capacity violation ratio.

1. Introduction

Software defined networking (SDN) provides a logically centralized controller that decouples the routing logic from the underlying forwarding elements. Since SDN delivers a flexible network management platform and enables operators to deploy new network functions rapidly, it has been widely used in the areas of data centers [2–4], WAN [5–7], and edge computing [8–11].

As the scale of SDN expands, a single controller needs to take charge of more and more switches, which leads to overload on processing capacity. Especially, controllers at edge have limited computing and bandwidth resources, and are prone to reach the performance bottleneck. Besides, unstable wireless connections in edge networks result in long communication delays between edge devices and a single controller. Thus, the controller cannot timely get the status of the data plane, and the edge devices suffer from a long response time from the controller. Thus, [12–14] introduce the distributed controller system where multiple controllers jointly manage switches in the data plane.

One crucial problem in a distributed controller system is to minimize the control plane response time (CPRT), which can be regarded as the elapsed time from the arrival to the completion for a network event. The distribution of the data plane traffic is uneven [15], *e.g.*, traffic reaches peak volume at around 8PM [16], and switches at different layers of hierarchical topologies significantly vary in flow arrival rates. Therefore, some controllers sometimes endure excessive network events and experience long CPRT, which leads to the deterioration of the network transfer performance. Moreover, long CPRT has an impact on the agility of the control plane, and consequently the network changes cannot be detected and handled timely.

In this paper, we address the problem of CPRT minimization using packet-in request redirection among controllers. *Packet-in requests* are the messages sent by switches when network events happen. In our scenario, each switch is statically assigned to a fixed controller [17–19]. Formally, given the instantaneous states of the system (*e.g.*, controller

☆ This paper is an extended version of work published in [1]. This work was supported in part by the National Key R&D Program of China under Grant No. 2018YFB1004704, in part by the National Natural Science Foundation of China under Grant No. 61502229, 61872178, 61832005, 61672276, 61872173, 61802172, and 61321491, in part by the Natural Science Foundation of Jiangsu Province under Grant No. BK20181251, in part by the Fundamental Research Funds for the Central Universities under Grant 021014380079, in part by funding from the Research Grants Council of Hong Kong (GRF 11209520) and from CUHK (4937007, 4937008, 5501329, 5501517).

* Corresponding authors.
E-mail addresses: xiarui@smail.nju.edu.cn (R. Xia), haipengdai@nju.edu.cn (H. Dai), jzheng@nju.edu.cn (J. Zheng), hongxu@cuhk.edu.hk (H. Xu), menson.smail.nju.edu.cn (M. Li), gchen@nju.edu.cn (G. Chen).

https://doi.org/10.1016/j.sysarc.2022.102590
Received 16 February 2022; Received in revised form 13 May 2022; Accepted 25 May 2022

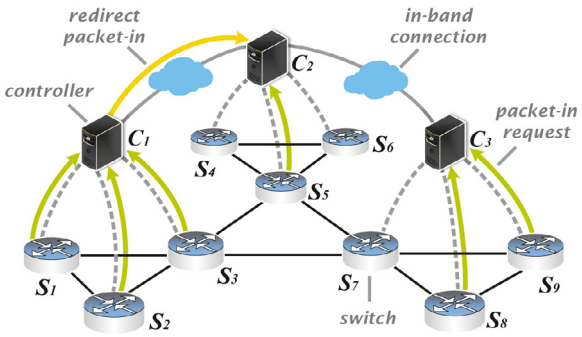


Fig. 1. Packet-in request redirection scheme.

load, packet-in requests arrival rate, *etc.*), our goal is to determine whether to redirect packet-in requests, such that the overall CPRT for all switches is minimized, while confining to the capacity constraints of the controllers.

There are two lines of work on CPRT minimization. First, [20–23] attempt to dynamically assign switches to controllers. For example, in Fig. 1, controller C_1 is overloaded, and switch S_3 can be reassigned to C_2 or C_3. This leads to unavoidable deployment cost during controller migration (*e.g.*, switch-controller connection setup). Among these works, [21] proposes an algorithm with a non-constant approximation ratio, which requires several iterations to achieve a Nash stable solution. However, our algorithm obtains a result with a constant approximation ratio in one round. Second, [24] proposes flow redirection in the data plane. For example, in Fig. 1, flows arriving at S_3 can be redirected to S_4, and S_4 reports packet-in requests. In this way, C_2 substitutes C_1 for processing S_3's packet-in requests. This incurs the expense of pre-installing wildcard rules in the flow tables of S_3, S_5, and S_4 for establishing the path between S_3 and S_4. Besides, [24] needs to predict the arrival distributions of new flows, while our work observes the instantaneous states and feed them to an online algorithm. Moreover, existing work neglects the capacity constraint of buffers in a controller. Most of them leverage the queuing theory and ensure that the average processing rate is greater than the average arrival rate. The above constraint on buffer capacity is weak and prone to cause buffer overflow. Note that the overflow leads to packet loss, which significantly affects the tail response time of packet-in requests. Our work defines the constraint of buffer capacity explicitly in the optimization problem and have a fine-grained control over the queue length in the buffer.

In this paper, we propose a new scheme based on packet-in request redirection. We depict it in Fig. 1, where C_1 meets a spike in packet-in requests and causes long CPRT for S_1, S_2, and S_3. In our scheme, we redirect the packet-in requests of one switch (*e.g.*, S_1) to C_2, and harness underutilized processing capacity of C_2 to handle these requests. On one hand, our scheme does not require controller migration which is needed in dynamic switch assignment. On the other hand, our scheme avoids pre-installing wildcard rules to establish paths for flow redirection. This is because we can utilize the connection among controllers, *e.g.*, in Fig. 1, C_1 and C_2 are connected through the in-band connection [25]. Note that our network model tracks the round-trip time (RTT) between two controllers, *e.g.*, RTT between C_1 and C_2 in the above example. Furthermore, we consider different capacity constraints of controllers in our scheme, and achieve a 3× improvement in maximum controller capacity violation ratio compared to the related works on CPRT minimization. We argue that our scheme has little effect on the traffic overhead among controllers. For example, using the reactive flow caching scheme proposed in SoftRing [2], a large proportion of packet-in messages carry only the first 128 Bytes of a table-miss flow.

We face two main technical challenges when dealing with the packet-in request redirection problem for CPRT minimization. The first challenge is that we need to make the redirection decision for each switch without a priori knowledge of the arrival rates of packet-in requests. The algorithm should capture the instantaneous state of the system, and make the redirection decisions by jointly considering the CPRT minimization and the capacity constraints of controllers in an online manner. The second challenge is to efficiently obtain a redirection decision with performance guarantee, for the online algorithm cannot endure the long-time waiting for decision making.

We propose a framework based on Lyapunov optimization to address the above two challenges. First, we substitute virtual queues for capacity constraints of controllers, and combine a queue stability problem with the CPRT minimization problem using the drift-plus-penalty (DPP) algorithm. Then, on each time slot, we derive a non-linear integer program from the DPP algorithm, whose optimal solution can be obtained with brute force using linearization techniques. Second, we find that time overhead of obtaining redirection decisions is significant, if simply leveraging Lyapunov optimization. Thus, we employ two techniques for different purposes, which cooperatively ease the time complexity of DPP algorithm in Lyapunov optimization. For faster running speed, we use the greedy strategy to efficiently get an initial solution, and recursively optimize it with the multiple knapsack problem. We theoretically prove the approximation ratio of the greedy strategy, which is related to the system parameters. To further pursue a *constant* approximation ratio, we prove that the reformulated problem falls into the scope of the problem of maximizing a non-monotone submodular function subject to matroid constraints. Both the second and the third ways provide approximate results, but each one has special advantages under different settings, *e.g.*, the second way has a faster running time, while the third way has a more optimal performance improvement ratio on average.

We construct a distributed controller system with Floodlight [26] by leveraging the `SyncManager` module. For packet-in request redirection, we temporarily connect two controllers with a network channel, which is kept active in a customized time interval to reduce connection setup time for subsequent request redirection. We periodically measure the round-trip time between any two controllers and report it to our algorithm. Then, we conduct simulations to evaluate our proposed algorithms. The results show our algorithms can reduce the average CPRT by 81.6% compared to static matching, and are 3× better than previous works on CPRT minimization in terms of capacity constraints.

2. Related work

CPRT minimization. Previous works on CPRT Minimization problem consist of two types of treatments, *i.e.*, dynamic controller assignment [20–23,27,28] and flow redirection in the data plane [24,29,30]. However, our approach is free from the controller migration, and does not need to pre-install wildcard rules for establishing paths in the execution of flow redirection. [21,22] focus on the dynamic controller assignment mechanism, which have been already implemented in [4,31]. Wang et al. [21] considered the switch-controller assignment problem as a stable matching problem with transfer, and proposed a two-phase algorithm, which combined the matching theory with coalitional games. Wang et al. [24] minimized the maximum value of CPRT through flow redirection, which needed to pre-install wildcard rules on switches. Wang et al. [29] studied the low delay route deployment problem, which concerned the QoS performance degradation when the control channels between switches and the controller were overloaded. Huang et al. [28] offloaded control plane's workloads to data plane, and dynamically made the switch-controller association and control devolution based on predicted request arrivals. Chai et al. [32] studied the capacitated controller deployment problem, which achieves the tradeoff between CPRT and the cost of controllers.

Controller placement and assignment. The placement and assignment of controllers are of vital importance to make the most of a distributed controller system. Our problem can adapt the existing works [17–19,33–36] to provide an initial controller-switch assignment, and further reduces CPRT as the system is up and running. Heller et al. [37] were the first to reveal two specific questions: how many controllers are needed, and where should they go? Lange et al. [19] presented POCO, a framework provided operators with Pareto optimal placements with respect to different performance metrics. Yao et al. [17] defined a capacitated controller placement problem, which attached importance to the load of controllers. Moreover, Das et al. [38] provided a comprehensive survey on the controller placement problem, and illustrated its significance.

Connections among distributed controllers. Distributed controllers are physically distributed across the network, but should have a consistent view of the data plane. Some SDN controllers [39–42] design various coordination protocols to logically connect the distributed controllers. Other works [43–48] propose network models of distributed controllers and optimize synchronization overhead. These network models focus on the communication cost among controllers, which is consistent with the model of packet-in request redirection. Namely, the network conditions among controllers affect the decision of redirection. Poularakis et al. [44] learned the SDN synchronization problem by considering two different objectives, *i.e.*, synchronization levels and application performance. Zhang et al. [43] analyzed and quantified the influence of synchronization levels on the average cost of the constructed routing paths. Muqaddas et al. [48] studied the pattern of inter-controller traffic under different consistency levels, and developed empirical models to quantify the traffic. Sakic et al. [49] used Stochastic Activity Networks (SAN) to model a distributed control plane connected by Raft, and analyzed the response time and availability metrics.

3. Models and problem statement

In this section, we first present the basic models of SDN and and network event handling, and list the related notations in Table 1. Then, we introduce the packet-in request redirection model, and list the related notations in Table 2. Finally, we define the CPRT minimization problem (CMP).

3.1. Network model in SDN

We consider the network topology in SDN as a two-tier structure, *i.e.*, the control and data plane, which are communicated through the OpenFlow protocol. The control plane consists of K controllers, which are denoted as $C = \{c_1, \dots, c_K\}$. Since these controllers may locate in different racks or sites, we use $D(j_1, j_2)$ to represent the communication delay between controller c_{j_1} and controller c_{j_2}. Namely, $D(j_1, j_2)$ is the time cost for a packet-in request to redirect from c_{j_1} to c_{j_2}, and return to c_{j_1} after processing. The processing rates of controllers are denoted as $\{\alpha_1, \dots, \alpha_K\}$. The data plane consists of N switches, which are denoted as $S = \{s_1, \dots, s_N\}$. During the configuration of SDN, a network operator can associate a switch with one and only one controller for network events handling. We denote $A(i)$ as the associated controller of s_i. Besides, we consider a discrete time model where the arrival rate of packet-in requests for a switch can be precisely measured. The duration of a time slot is an alternative variable, denoted as δ. For all switches in S, we denote $\boldsymbol{\lambda(t)} = [\lambda_1(t), \dots, \lambda_N(t)]$ as the vector of average arrival rates of packet-in requests on slot t. We present a toy example of our model on slot t in Fig. 2, where there exist four switches, *i.e.*, $\{s_1, s_2, s_3, s_4\}$, and two controllers, *i.e.*, $\{c_1, c_2\}$. The processing rates of c_1 and c_2 are α_1 and α_2, respectively, and the communication delay between them is $D(1,2)$. For a switch, *e.g.*, s_1, its arrival rate of packet-in requests is $\lambda_1(t)$, and its associated controller $A(1)$ is c_1.

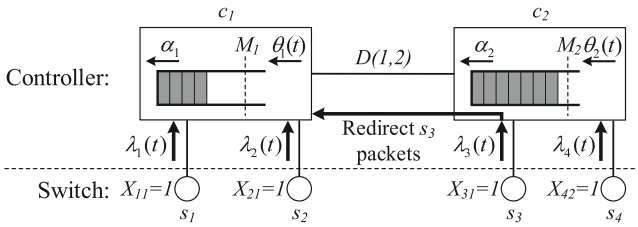


Fig. 2. Model of the packet-in request redirection.

Table 1
Basic notations.

Symbol	Meaning
S	$S = \{s_1, \dots, s_N\}$, set of switches
$A(i)$	Associated controller for s_i
C	$C = \{c_1, \dots, c_K\}$, set of controllers
t	$t \in \{1, 2, \dots, T\}$, time slots
δ	Duration of a time slot
$D(j_1, j_2)$	Communication delay between c_{j_1} and c_{j_2}
α_j	Processing rate of c_j
$\boldsymbol{\lambda(t)}$	Vector of switches' arrival rates of packet-in requests
M_j	Capacity constraint of c_j

Table 2
Notations for redirection model.

Symbol	Meaning
$\boldsymbol{Q(t)}$	Vector of controllers' waiting queue length
$X_{ij}(t)$	Indicator of whether to redirect s_i's packet-in requests to c_j on slot t
$\theta_j(t)$	Total arrival rates of packet-in requests for c_j on slot t
$C_i^T(t)$	Transmission cost of processing s_i's packet-in requests on slot t
$C_i^Q(t)$	Queueing cost of processing s_i's packet-in requests on slot t
$C_i(t)$	Total time cost of processing s_i's packet-in requests on slot t

3.2. Network event handling model

When a network event happens, a switch reports the event to its associated controller. Thus, s_i sends packet-in requests to $A(i)$ with the rate of $\lambda_i(t)$ on slot t. When receiving packet-in requests from s_i, $A(i)$ may process them itself or immediately redirect them to another controller. If s_i's requests are processed on $A(i)$, $A(i)$ will push these requests into its FIFO waiting queue. $A(i)$ continuously pull a request from its queue and process it using the global network information. We use vector $\boldsymbol{Q(t)} = [Q_1(t), \dots, Q_K(t)]$ to represent the waiting queue length of controllers in C at the *start* of time slot t. Furthermore, c_j has a capacity constraint of the waiting queue length, which is denoted as M_j. As shown in Fig. 2, the waiting queue length of c_1, *i.e.*, $Q_1(t)$, is four, and does not exceed the capacity constraint, *i.e.*, M_1.

3.3. Packet-in request redirection model

We try to redirect packet-in requests to reduce their CPRT. We denote a decision variable as $X_{ij}(t)$, which is an indicator of whether to redirect s_i's packet-in requests to c_j on slot t. Note that all packet-in requests from one switch on slot t are not divided into multiple parts, for some flexible NICs [50] can directly forward a packet based on its header fields and speed up the operation of packet-in request redirection. Consequently, we refer $\theta_j(t)$ as the total arrival rates of packet-in requests for c_j on slot t, and express it as

$$\theta_j(t) = \sum_{i=1}^{N} X_{ij}(t)\lambda_i(t). \tag{1}$$

The waiting queue length of c_j on slot $t+1$, *i.e.*, $Q_j(t+1)$, can be estimated as $Q_j(t+1) = \max\{Q_j(t) + (\theta_j(t) - \alpha_j)\cdot\delta, 0\}$. For example, $X_{31}(t)$ is set to one in Fig. 2, and thus s_3's packet-in requests are redirected to c_1. We derive that $\theta_1(t) = \sum_{i=1}^{3}\lambda_i(t)$ and $\theta_2(t) = \lambda_4(t)$.

For packet-in requests from a switch, their CPRT consists of two parts, *i.e.*, the transmission cost and the queueing cost. First, the transmission cost is derived from the packet dissemination between the associated controller and the packet-in requests processing controller. We denote the *transmission cost* of s_i on slot t as $C_i^T(t)$, which is represented as

$$C_i^T(t) = \sum_{j=1}^{K} X_{ij}(t) D(A(i), j). \tag{2}$$

In Fig. 2, we redirect s_3's packet-in requests to c_1. The processing results of these requests are returned back c_2, and directly sent to s_3. Thus, the transmission cost of s_3's packet-in requests, *i.e.*, $C_3^T(t)$, is $D(1,2)$. Besides, for s_1, s_2, and s_4, the transmission costs are all zero, for their packet-in requests do not need to be redirected.

Second, the queueing cost is generated by the packet-in requests processing controller, whose waiting queue is not empty. The packet-in requests will be queued until all earlier arrived requests have finished processing. Since the accurate waiting queue length is hard to estimate, we use the *worst* case of the queueing length on slot t, which is at most $Q_j(t)+\delta\cdot\theta_j(t)$ for c_j. Intuitively, the above worst case seems to batch all packet-in requests of s_i in the associated controller $A(i)$ and redirects the packet-in requests at the end of a slot. However, the implementation of request redirection is in a streaming mode. A packet-in request of s_i will be promptly redirected to its processing controller c_j. The worse case aims to estimate the upper bound of queueing cost and does not affect the implementation of redirection. Therefore, we denote $C_i^Q(t)$ as the *queueing cost* of s_i's packet-in requests, and express it as

$$C_i^Q(t) = \sum_{j=1}^{K} X_{ij}(t) \cdot \frac{Q_j(t) + \delta \cdot \theta_j(t)}{\alpha_j}. \tag{3}$$

In Fig. 2, the queueing costs of s_1, s_2, and s_3 are all $Q_1(t) + \delta \cdot \theta_1(t)$, while that of s_4 is $Q_2(t) + \delta \cdot \theta_2(t)$.

Finally, CPRT of the packet-in requests from switch s_i on slot t is the summation of the above two costs, and we denote it as

$$C_i(t) = C_i^T(t) + C_i^Q(t). \tag{4}$$

Note that, the redirection decision is affected by two factors: the inter-controller communication delay and the real-time queue length. Intuitively, the redirection decision prefers a controller which has small communication delay and is lightly loaded.

3.4. CPRT minimization problem

The distributed control plane for request redirection is a two-layer structure. The upper layer is a central controller, which monitors the real-time metrics of controllers and switches, *e.g.*, $\boldsymbol{Q(t)}$ and $\boldsymbol{\lambda(t)}$. The lower layer consists of multiple worker controllers, which receive packet-in requests and process them. The central controller instructs worker controller to redirect packet-in requests.

The CPRT minimization problem (CMP) aims to minimize the overall CPRT by updating the redirection decision at the start of each slot. After gathering metrics from controllers and switches, the central controller constructs a new instance of CMP and calculates a new redirection decision. At the start of the next slot, the central controller updates the redirection decision of worker controllers to the new one. An instance of CMP is formulated as follows:

$$\min_{X_{ij}(t)} \quad \sum_{t=1}^{T}\sum_{i=1}^{N} C_i(t)$$

$$s.t. \quad Q_j(t) \le M_j, \quad \forall j \tag{5}$$

$$\sum_{j=1}^{K} X_{ij}(t) = 1, \quad \forall i, t \tag{6}$$

$$X_{ij}(t) \in \{0,1\}, \quad \forall i, j$$

Note that $X_{ij}(t)$s ($i = 1,\dots,N; j = 1,\dots,K; t = 1,\dots,T$) are the decision variables. Constraint (5) forces the waiting queue length not to exceed the controller's capacity. Constraint (6) ensures that packet-in requests will be processed by only one controller.

CMP can be viewed as a variant of generalized assignment problem (GAP), which is NP-hard. Furthermore, if we can observe the distribution function of $\boldsymbol{\lambda(t)}$, a common way is to minimize the expectation of the objective function in CMP to obtain the *offline optimal result*. However, an offline algorithm is not suitable for scenarios such as the data center, where network traffic volume is rapidly changing.

4. Scheme based on Lyapunov optimization

This section goes step by step in explaining our design of the online analysis scheme. Note that Lyapunov optimization can provide control over waiting queues of controllers and make a compromise between CPRT minimization and controller capacity constraints using the drift-plus-penalty (DPP) algorithm. Thus, we leverage Lyapunov optimization in our model and prove that the gap between our online decision and the offline optimal result is bounded. Specifically, we first transform each controller capacity constraint in the time average form, and further convert it to a queue stability problem. Then, we use the DPP algorithm to solve CMP, and define the redirection decision problem (RDP) derived from the DPP algorithm.

4.1. Conversion of capacity constraints

In CMP, waiting queue length of each controller should not exceed its capacity, *i.e.*, $Q_j(t) \le M_j$ on slot t.

Definition 4.1. For each $c_j \in C$, we define $\overline{Q_j}(T)$ as the time average of $Q_j(t)$ over the first T slots:

$$\overline{Q_j}(T) = \frac{1}{T}\sum_{t=1}^{T} Q_j(t),$$

and define $\overline{Q_j}$ as the limiting value of $\overline{Q_j}(T)$, when $T \to \infty$.

Due to traffic fluctuation, it is infeasible to guarantee that the waiting queue length would not exceed the capacity on each time slot. Thus, we leverage its time average, and limit it within the capacity constraint, *i.e.*, $\overline{Q_j} \le M_j$. We convert the controller capacity constraint into the time average form, and further transform it into a queue stability problem.

Definition 4.2 (*Virtual Queue*)**.** we define a virtual queue $Z_j(t)$ for each $j \in \{1,\dots,K\}$, with update equation:

$$Z_j(t+1) = \max\{Z_j(t) + Q_j(t+1) - M_j, 0\}. \tag{7}$$

We propose the following theorem, which shows that the rate stability of the virtual queue $Z_j(t)$ can infer $\overline{Q_j} \le M_j$.

Theorem 4.1. *For each $c_j \in C$, we define the rate stability of the virtual queue $Z_j(T)$ as*

$$\limsup_{T\to\infty} \mathbb{E}[Z_j(T)]/T = 0,$$

which can ensure the constraint of $\overline{Q_j} \le M_j$.

Proof. Using the update equation of $Z_j(t)$ in Eq. (7), we get its lower bound:

$$Z_j(t+1) \ge Z_j(t) + Q_j(t+1) - M_j. \tag{8}$$

Then, we add up In Eq. (8) from time slot 1 to T, and dividing T at the both sides, we have $\mathbb{E}[Z_j(T)]/T \ge -M_j + \sum_{t=1}^{T}\mathbb{E}[Q_j(t)]/T$. When $T \to \infty$,

$$\overline{Q_j} \le M_j + \limsup_{T\to\infty} \mathbb{E}[Z_j(T)]/T = M_j. \quad \square$$

4.2. Lyapunov Optimization and DPP algorithm

Definition 4.3 (*Lyapunov Function*)**.** We let $\boldsymbol{Z(t)} = [Z_1(t), \dots, Z_K(t)]$ be the vector of all virtual queue backlogs, and define the Lyapunov function $L(\boldsymbol{Z(t)})$ as follows:

$$L(\boldsymbol{Z(t)}) = \frac{1}{2}\sum_{j=1}^{K} Z_j^2(t).$$

We use Eq. (7) to compute a bound on the shift in the Lyapunov function from one time slot to the next:

$$\begin{aligned} L(\boldsymbol{Z(t+1)}) - L(\boldsymbol{Z(t)}) &= \frac{1}{2}\sum_{j=1}^{K}(Z_j^2(t+1) - Z_j^2(t)) \\ &\le \sum_{j=1}^{K}\Big[\frac{1}{2}[Q_j(t+1) - M_j]^2 + Z_j(t)[Q_j(t+1) - M_j]\Big]. \end{aligned} \tag{9}$$

Definition 4.4 (*Lyapunov Drift*)**.** We define $\Delta(\boldsymbol{Z(t)})$ as the conditional Lyapunov drift for time slot t:

$$\Delta(\boldsymbol{Z(t)}) = \mathbb{E}[L(\boldsymbol{Z(t+1)}) - L(\boldsymbol{Z(t)})|\boldsymbol{Z(t)}].$$

Theorem 4.2. *We bound the value of $\Delta(\boldsymbol{Z(t)})$ as follows:*

$$\Delta(\boldsymbol{Z(t)}) \le B + \sum_{j=1}^{K} Z_j(t)\cdot\mathbb{E}[(Q_j(t+1) - M_j)|\boldsymbol{Z(t)}], \tag{10}$$

where B is a constant value.

Proof. Using In Eq. (9), $\Delta(\boldsymbol{Z(t)})$ is bounded by

$$\begin{aligned} \Delta(\boldsymbol{Z(t)}) &= \mathbb{E}[L(\boldsymbol{Z}(t+1)) - L(\boldsymbol{Z(t)})|\boldsymbol{Z(t)}] \\ &\le \mathbb{E}\Big\{\sum_{j=1}^{K}\big[\frac{1}{2}[Q_j(t+1) - M_j]^2 + Z_j(t)[Q_j(t+1) - M_j]\big]|\boldsymbol{Z(t)}\Big\}. \end{aligned}$$

Here, for each $c_j \in C$, we let

$$\epsilon_j = \max\{\epsilon' : Q_j(t) + \epsilon' \le M_j\}, \text{for all } t = 1, 2, \dots$$

The values of $\{\epsilon_1, \dots, \epsilon_K\}$ exist for they can be generated from a feasible redirection decision. Therefore, we get the bound as follows:

$$\sum_{j=1}^{K}\mathbb{E}\Big\{\frac{1}{2}[Q_j(t+1) - M_j]^2|\boldsymbol{Z(t)}\Big\} \le \sum_{j=1}^{K}\max\{M_j^2, \epsilon_j^2\} \triangleq B.$$

Thus, the theorem follows. □

Definition 4.5 (*Drift Plus Penalty*)**.** We define the drift plus penalty (DPP) expression on slot t:

$$DPP(t) = \Delta(\boldsymbol{Z(t)}) + V\cdot\mathbb{E}[C(t)|\boldsymbol{Z(t)}], \tag{11}$$

where $V \ge 0$ is a parameter that represents the weight on how much we emphasize CPRT Minimization.

We combine CPRT minimization with the virtual queue stability problem, which indicates controllers' capacity constraints. We further bound the DPP expression in Eq. (11) with the following theorem.

Theorem 4.3. *We bound the value of $DPP(t)$ as follows:*

$$\begin{aligned} DPP(t) \le B' + \mathbb{E}\Big\{ & V\cdot\sum_{i=1}^{N}\sum_{j=1}^{K} X_{ij}(t)\big[D(A(i), j) + \frac{Q_j(t)}{\alpha_j}\big] \\ & + \delta\cdot\sum_{j=1}^{K}\theta_j(t)\big[\sum_{i=1}^{N}\frac{V\cdot X_{ij}(t)}{\alpha_j} + Z_j(t)\big]\Big|\boldsymbol{Z(t)}\Big\}, \end{aligned}$$

where $B' = B + \sum_{j=1}^{K} Z_j(t)[Q_j(t) - \alpha_j\cdot\delta - M_j]$.

Proof. Substitute Eqs. (2), (3), (4), and Inequality (10) to Equality (11), and the result follows. □

Algorithm 1: Analysis Scheme for CMP

Input: Time slot interval δ, weighted parameter V, capacity constraints of controllers $\{M_1, ..., M_K\}$
Output: Redirection decision $\boldsymbol{X(t)}$
1 **for** *the start of each time slot* $t \in \{1, ..., T\}$ **do**
2 Get $\boldsymbol{Q(t)}$, and update $\boldsymbol{Z(t)}$.
3 Get the packet-in requests arrival rates $\boldsymbol{\lambda(t)}$ perceived during the previous time slot.
4 Construct a new instance of RDP.
5 Get the current redirection decision $\boldsymbol{X(t)}$ by solving the program in the new instance of RDP.

Rather than directly minimize the DPP expression in Eq. (11), our strategy aims to minimize the bound provided by Theorem 4.3. We describe the analysis scheme for CMP in Algorithm 1. In Step 1–3, the algorithm makes redirection decision at the start of each time slot, when the leader controller learns the system states, *i.e.*, $\boldsymbol{Q(t)}$, $\boldsymbol{\lambda(t)}$. In Step 4, we construct an instance of redirection decision problem (RDP), which attempts to minimize the upper bound given in Theorem 4.3 and is presented as follows:

$$\begin{aligned} \min_{X_{ij}(t)}\ & V\cdot\sum_{i=1}^{N}\sum_{j=1}^{K} X_{ij}(t)\big[D(A(i), j) + \frac{Q_j(t)}{\alpha_j}\big] \\ & + \delta\cdot\sum_{j=1}^{K}\theta_j(t)\big[\sum_{i=1}^{N}\frac{V\cdot X_{ij}(t)}{\alpha_j} + Z_j(t)\big] \\ s.t.\ & \sum_{j=1}^{K} X_{ij}(t) = 1, \quad \forall i \\ & X_{ij}(t) \in \{0, 1\}, \quad \forall i, j. \end{aligned} \tag{12}$$

In Step 5, we get the redirection decision, *i.e.*, $\boldsymbol{X(t)}$, by solving RDP. However, the objective function in Eq. (12) is nonlinear, for the values of $\{\theta_1(t), \dots, \theta_K(t)\}$ are related to $\boldsymbol{X(t)}$. In the next section, we try to study RDP using three different algorithms, which provide the optimal or approximate solutions to the problem.

Theorem 4.4. *The gap between the online decision provided by our analysis scheme and the offline optimal result is at most $O(1/V)$. Furthermore, we can ensure that the time average of each waiting queue length $\overline{Q_j}$ is not over M_j.*

Proof. For a better flow of this paper, we move the proof to Appendix A. □

5. Algorithms for redirection decision

In this section, we address redirection decision problem (RDP) using three different algorithms. We start by converting RDP to a mixed integer program (MIP) problem, which can provide the optimal solution to the problem. Following that, we propose two approximation algorithms, which are much efficient than the above one. For faster running speed, we greedily make an initial redirection decision, and recursively optimize it with the multiple knapsack problem. To pursue a constant approximation ratio, we resort to submodular optimization.

5.1. Optimal solution using MIP

RDP has quadratic terms in the objective function, and standard linear program approaches cannot be used. However, we can apply standard linearization techniques to obtain an MIP formulation, which only consists of linear objective function and constraints. We denote

Algorithm 2: RDP-MIP Algorithm for RDP

Input: Waiting queue length $\boldsymbol{Q}(t)$, packet-in requests arrival rates $\lambda(t)$, virtual queue length $\boldsymbol{Z}(t)$
Output: Redirection decision $\boldsymbol{X}(t)$
1 Linearize the program in RDP by involving new variables with constraints expressed in Inequation (13), (14), and (15).
2 Get the redirection decision $\boldsymbol{X}(t)$ by solving the MIP after substituting new variables into RDP.

the algorithm using MIP as RDP-MIP, and describe it in Algorithm 2. In Step 1, we introduce a set of new $\{0,1\}$ variables as follows:

$$Y_{ilj}(t) \le X_{ij}(t), \tag{13}$$

$$Y_{ilj}(t) \le X_{lj}(t), \tag{14}$$

$$Y_{ilj}(t) \ge X_{ij}(t) + X_{lj}(t) - 1, \tag{15}$$

where $i,l \in \{1,\dots,N\}, j \in \{1,\dots,K\}$. We observe that $Y_{ilj}(t)$ is equal to one, if and only if both $X_{ij}(t)$ and $X_{lj}(t)$ are set to one; otherwise, it will be zero. Furthermore, it is obvious that $X_{ij}(t) = Y_{iij}(t)$. Thus, we reformulate RDP as an MIP, which is presented as follows:

$$\begin{aligned}
\min_{X_{ij}(t),Y_{ilj}(t)} \quad & V \cdot \sum_{i=1}^{N}\sum_{j=1}^{K} Y_{iij}(t)\Big[D(A(i),j) + \frac{Q_j(t)}{\alpha_j}\Big] \\
& + \delta \cdot \sum_{j=1}^{K}\sum_{i=1}^{N}\sum_{l=1}^{N} \lambda_l(t)\Big[\frac{V \cdot Y_{ilj}(t)}{\alpha_j} + Z_j(t)Y_{llj}(t)\Big] \\
s.t. \quad & \sum_{j=1}^{K} Y_{iij} = 1, \quad \forall i \\
& Y_{ilj}(t) \in \{0,1\}, X_{ij}(t) \in \{0,1\}, \quad \forall i,l,j \\
& (13), (14), (15).
\end{aligned}$$

Existing LP solvers can be directly used to solve an MIP. These solvers mainly apply branch-and-bound techniques and are primarily fit to small-scale scenarios. In the next two subsections, we aim at solving RDP in large-scale instances. We attempt to pursue high time-efficient algorithms with bounded approximation ratios to the optimal solution.

5.2. Algorithm with recursive optimization

We reorder the terms in RDP, and construct it in a more understandable way named as transformed RDP (TRDP):

$$\min_{X_{ij}(t)} \quad \sum_{i=1}^{N}\sum_{j=1}^{K} X_{ij}(t) \cdot Cost(i,j) + \sum_{i=1}^{N}\sum_{j=1}^{K} X_{ij}(t) \cdot \theta'_j(t) \tag{16}$$

$$s.t. \quad \sum_{j=1}^{K} X_{ij}(t) = 1, \quad \forall i,j \tag{17}$$

$$X_{ij}(t) \in \{0,1\}, \quad \forall i,j.$$

The expressions of $Cost(i,j)$ and $\theta'_j(t)$ in TRDP are presented as follows:

$$Cost(i,j) = V\Big[D(A(i),j) + \frac{Q_j(t)}{\alpha_j}\Big] + \delta \cdot Z_j(t)\lambda_i(t), \tag{18}$$

$$\theta'_j(t) = \frac{\delta \cdot V}{\alpha_j} \sum_{i=1}^{N} X_{ij}(t)\lambda_i(t).$$

The underlying insight of TRDP is that we consider two types of cost, when redirecting a switch's (s_i) packet-in requests to a controller (c_j), or namely, setting the value of the decision variable ($X_{ij}(t)$) to one.

Definition 5.1 (*Type-one Cost*)**.** The type-one cost, denoted as $Cost(i,j)$, only relates to the system parameters, *i.e.*, the communication delay between controllers, and the known instantaneous system states, *i.e.*, $\boldsymbol{Q}(t)$, $\boldsymbol{Z}(t)$, and $\lambda(t)$.

Algorithm 3: RDP-G Algorithm for RDP

Input: Waiting queue length $\boldsymbol{Q}(t)$, packet-in requests arrival rates $\lambda(t)$, virtual queue length $\boldsymbol{Z}(t)$
Output: Redirection decision $\boldsymbol{X}(t)$
1 Reconstruct the terms, and transform RDP into TRDP.
2 For switch s_i and controller c_j, compute $Cost(i,j)$.
3 Get the initial redirection decision $\boldsymbol{X}^{(0)}(t)$ using the strategy expressed in Equation (19).
4 $f \leftarrow 0$. Construct a instance of MKP using $\boldsymbol{X}^{(0)}(t)$, and get $\boldsymbol{X}^{(f+1)}(t)$ by solving the new instance.
5 *impr_ratio* $\leftarrow$ the improvement ratio of $\boldsymbol{X}^{(f)}(t)$ to $\boldsymbol{X}^{(0)}(t)$ in terms of the objective function for TRDP in Equation (16).
6 **while** *impr_ratio* $\ge$ *a predefined threshold* **do**
7 $f \leftarrow f+1$. Construct MKP using $\boldsymbol{X}^{(f)}(t)$, and get $\boldsymbol{X}^{(f+1)}(t)$ by solving the MKP.
8 Update the value of *impr_ratio*.
9 **return** $\boldsymbol{X}^{(f)}(t)$.

Definition 5.2 (*Type-two Cost*)**.** The type-two cost, denoted as $\theta'_j(t)$, has relation to the decision variables, *i.e.*, $\boldsymbol{X}(t)$.

The values of the type-one costs can be thought as constants when we solve TRDP. However, the values of the type-two costs are varied as the decision variables change, and thus cannot be viewed as constants. When redirecting packet-in requests to a same controller c_j, all switches share an identical type-two cost $\theta'_j(t)$, while the type-one cost for each switch is unique, *e.g.*, $Cost(i,j)$ for s_i.

Using the implication of TRDP, we propose the greedy algorithm with recursive optimization, which is denoted as RDP-G in Algorithm 3. In Step 3, $\boldsymbol{X}^{(0)}(t) = \{X_{ij}^{(0)}(t) | i \in \{1,\dots,N\}, j \in \{1,\dots,K\}\}$ is the initial redirection decisions, each of which is expressed as follows:

$$X_{ij}^{(0)}(t) = \begin{cases} 1\ , j = \arg\min_j [Cost(i,j) \cdot C_j] \\ 0\ , else, \end{cases} \tag{19}$$

where $C_j = \frac{1}{\alpha_j}\sum_{i=1}^{N}\frac{1}{Cost(i,j)}$. The intuition of the initial redirection decision is the following: for each switch, we aim at assigning the controller with the minimum type-one cost, yet keeping an eye on the effects of other switches.

In Step 4, we construct an instance of the multiple knapsack problem (MKP) [51] with the following two steps. First, substituting $\boldsymbol{X}^{(0)}(t)$ into Eq. (1), we obtain the aggregated arrival rates of switches for a controller, *e.g.*, c_j, as follows:

$$\theta_j^{(0)} = \sum_{i=1}^{N} X_{ij}^{(0)}(t)\lambda_i(t). \tag{20}$$

Second, we substitute Eq. (20) into TRDP, and get the following integer linear program (ILP):

$$\min_{X_{ij}(t)} \quad \sum_{i=1}^{N}\sum_{j=1}^{K} X_{ij}(t)\Big(Cost(i,j) + \frac{\delta \cdot V(\theta_j^{(0)} + \gamma)}{\alpha_j}\Big) \tag{21}$$

$$s.t. \quad \sum_{j=1}^{K} X_{ij}(t) = 1, \quad \forall i$$

$$\sum_{i=1}^{N} X_{ij}(t) \cdot \lambda_i(t) \le \theta_j^{(0)}(t) + \gamma, \quad \forall j, \tag{22}$$

where $\gamma > 0$ is the maximum violation of the aggregated arrival rates of packet-in requests for each controller, compared to the initial redirection decision, *i.e.*, $\boldsymbol{X}^{(0)}(t)$. We add Constraint (22) to limit the aggregated arrival rates for a controller. In this way, we can get the upper bound of the type-two cost for each controller, and use this

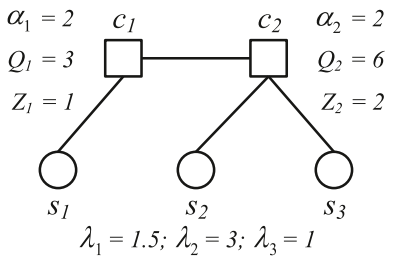


Fig. 3. A toy example for Algorithm 3.

bound to acquire an improved solution. The value of the cost term in Eq. (21), *i.e.*, $Cost(i,j)+\delta\cdot V(\theta_j^{(0)}+\gamma)/\alpha_j$, is only a constant. Thus, the above ILP can be viewed as an instance of MKP, whose optimal solution is denoted as $\boldsymbol{X}^{(1)}(t)$.

In Step 6, we recursively construct a new instance of MKP based on the current solution until the improvement ratio of the objective function for TRDP in Eq. (16) is within a predefined threshold, and obtain the final redirection decision.

Toy example. In Fig. 3, we provide a toy example to illustrate Algorithm 3. s_1 is directly connected to c_1, while s_2 and s_3 are connected to c_2. We list α, Q, Z, and λ in the figure. Besides, $D(1,2)$, δ, and V are set to one. Using Algorithm 3, we get $Cost(1,1) = 3$, $Cost(2,1) = 5.5$, $Cost(3,1) = 3.5$, $Cost(1,2) = 6$, $Cost(2,2) = 9$, and $Cost(3,1) = 5$, and derive that $C_1 = 0.400$ and $C_2 = 0.239$. According to Eq. (19), X_{11}, X_{22}, and X_{31} are set to one, and total cost in Eq. (16) becomes 17.7, while the cost of default setting, where X_{11}, X_{22}, and X_{32} are set to one, is 19.95. Moreover, we use MKP to further optimize the result, and set $\gamma = 2$ in Eq. (21). Consequently, X_{11}, X_{21}, and X_{32} are set to one, and the cost of the optimized result becomes 16.7.

Obviously, $\boldsymbol{X}^{(0)}(t)$ is a feasible solution to the new instance of MKP, so that $\boldsymbol{X}^{(1)}(t)$ is superior to $\boldsymbol{X}^{(0)}(t)$ in terms of the objective function in Eq. (21). Even though we cannot ensure that $\boldsymbol{X}^{(1)}(t)$ is a more optimal solution to TRDP compared to $\boldsymbol{X}^{(0)}(t)$, we will show that the approximation ratio of the initial redirection decision, *i.e.*, $\boldsymbol{X}^{(0)}(t)$, is already a constant in the following theorem. We denote C_{max} as $\max\{C_1,\dots,C_K\}$, and C_{min} as $\min\{C_1,\dots,C_K\}$.

Theorem 5.1. *For RDP-G, the approximation ratio of the initial redirection decision, i.e., $\boldsymbol{X}^{(0)}(t)$ is bounded by $(1+\eta)\left[1+\delta\cdot V\max\{\boldsymbol{\lambda}(t)\}C_{max}\right]$, where $1+\eta = C_{max}/C_{min}$.*

Proof. For a better flow of this paper, we move the proof to Appendix B. □

5.3. Algorithm using submodular optimization

In this subsection, we try to use techniques in submodular optimization to solve RDP. The following analysis is based on the transformed expression in TRDP, which has been discussed in Section 5.2.

First, we define a type of set functions called *supermodular*.

Definition 5.3 (*Supermodular*)**.** Let W be a finite set of elements (ground set). A set function $f : 2^W \to \mathbb{R}$ is called supermodular if for all subsets $A, B \subseteq W$ with $A \subseteq B$ and every element $e \in W\backslash B$, it holds that

$$f(A\cup\{e\}) - f(A) \le f(B\cup\{e\}) - f(B).$$

Next, we introduce the way to construct the ground set and set function for our problem, respectively.

Definition 5.4 (*Ground Set*)**.** The ground set W is defined as $W = \{e_{11},\dots,e_{1K},e_{21},\dots,e_{2K},\dots,e_{N1},\dots,e_{NK}\}$. The subsets $W_{i\bullet}$ and $W_{\bullet j}$ are denoted as $\{e_{i1},\dots,e_{iK}\}$ and $\{e_{1j}, ...,e_{Nj}\}$, respectively, for $i \in \{1,\dots,N\}$ and $j \in \{1,\dots,K\}$.

Algorithm 4: RDP-SO Algorithm for RDP

Input: Waiting queue length $\boldsymbol{Q}(t)$, packet-in requests arrival rates $\boldsymbol{\lambda}(t)$, virtual queue length $\boldsymbol{Z}(t)$
Output: Redirection decision $\boldsymbol{X}(t)$
1 Construct the set function $f(U)$ and ground set W.
2 Get $\boldsymbol{X}(t)$ by the algorithms for maximizing the non-monotone submodular function, *i.e.*, $\widehat{f}(U) = f^{ub} - f(U)$, subject to matroid constraints.

Definition 5.5 (*Set Function*)**.** we define the set function of the problem as follows:

$$f(U) = \sum_{e\in U} c(e) + \delta\cdot V\left\{\sum_{j=1}^{K}\left\{|U\cap W_{\bullet j}|\cdot\sum_{e\in U\cap W_{\bullet j}}\lambda(e)\right\} + \sum_{i=1}^{N}\left\{\mathbb{1}(|U\cap W_{i\bullet}| = 0)\cdot P\right\}\right\}, \tag{23}$$

where $U \subseteq W$, $c(e_{ij}) = Cost(i,j)$, $\lambda(e_{ij}) = \lambda_i(t)/\alpha_j$, $\mathbb{1}(\bullet)$ is an indicator function, and P is a penalty factor.

To obtain a redirection decision, we select a subset of W, which is denoted as U, and set the corresponding decision variables in U to one. For example, if we let $U = \{e_{11}, e_{21}\}$, then setting $X_{11}(t) = 1$ and $X_{21}(t) = 1$. Furthermore, Constraint (17), which ensures a feasible solution, can be formulated as a partition matroid constraint, *e.g.*, $|U\cap W_{i\bullet}| \le 1$.

Theorem 5.2. *The set function $f(U)$ is supermodular.*

Proof. For a better flow of this paper, we move the proof to Appendix C. □

We denote the algorithm using submodular optimization as RDP-SO, and describe it in Algorithm 4. In Step 2, we let $\widehat{f}(U) = f^{ub} - f(U)$, where f^{ub} indicates an upper bound to the highest possible value of $f(U)$, such that $\widehat{f}(U)$ is a non-negative, non-monotone and submodular function. Moreover, we would like to obtain its maximum value in an efficient way. The problem mentioned above has been well-studied in theoretical computer science, which can be seamlessly involved into our algorithm. Thus, we obtain the following theorem.

Theorem 5.3. *Algorithm 4 provide a solution to RDP with the approximation ratio, i.e., $\widehat{f}^{SOL}/\widehat{f}^{OPT} \ge \beta$, where β can be 0.372 in [52].*

Theorem 5.4. *For a set $U \subseteq W$ and a switch $s_i \in S$, if the condition $|U\cap W_{i\bullet}| = 0$, holds, we can derive that*

$$f(U) > f(U\cup\{e\}),\quad \forall e \in W_{i\bullet}, \tag{24}$$

when the penalty parameter, i.e., P, is large enough.

Proof. In the case that U has no element in $W_{i\bullet}$, we note that the penalty term in $f(U)$ will have a non-zero value. Thus, we can choose P large enough. In this way, picking any element in $W_{i\bullet}$ will cause a decreasing of $f(U)$. Therefore, In Eq. (24) holds for the case that the penalty parameter, *i.e.*, P, is large enough. □

Theorem 5.4 shows that the set function defined in Eq. (23) tends to the minimal value, when the subset U maps to a feasible solution to TRDP. Furthermore, if U achieves the minimum value of the set function, U will correspond exactly to the optimal solution to TRDP.

6. Discussion

Communication between controllers. A redirected packet-in request goes the other controller first and then comes back to its own controller. Our network model in Section 3.1 captures the communication

delay between controllers c_{j_1} and c_{j_2} with $D(j_1, j_2)$. However, real network deployment has more constraints on the communication between distributed controllers. Network providers usually divide the global network into small network domains, and deploy multiple controllers in a domain. Real-time network states are shared within one domain, and cannot easily shared across network boundaries. Thus, a packet-in request cannot be processed by a controller in other network domains. Though two controllers in different network domains have an in-band connection, the communication delay of the two controllers is manually set to infinity.

Packet loss is another issue on the communication between controllers. Temporary network congestion drops redirected packet-in requests, and the dropped requests suffer a long response time. Thus, the control plane loses the benefits of redirection even though some controllers are lightly loaded. To relieve the effect of unstable network condition, we dynamically configure the communication delay between two controllers. If a packet is dropped between two controllers, we will increase the communication delay between them, which avoids consequent redirection. When the network condition becomes stable, we set the communication delay to the normal value.

Buffers in controllers. In practice, a controller has finite buffers, and packet-in requests will be dropped when the buffer violates the controller's capacity. The optimization based on the DPP expression in Eq. (11) combines CPRT minimization with the stability of controllers' virtual queues. We use parameter V to adjust the proportion of the two objectives. During runtime, a controller reserves some extra buffers to cope with the instantaneous violation of buffer capacity. We need to make a trade-off between the size of the reserved buffers and the value of parameter V. For example, if the reserved buffers are sufficient, parameter V can put more emphasis on CPRT minimization. We experimentally study the choice of parameter V in Section 7.3.

The central controller monitors the queue length on each time slot. The instance of CMP obtains the real-time queue length and updates the redirection decision. Though the capacity constraint is in the time average format (Definition 4.1), the redirect decision is based on the real-time queue length. Consequently, the online optimization scheme provides a fine-grained control over the queue length of buffers in controllers, and avoids the instantaneous violation of buffer capacity.

Benefit for edge networks. Packet-in request redirection is suitable for load balancing of controllers in edge networks. The inter-connectivity of integrated edge–cloud environments varies from wireless links with limited bandwidth and transmission distance in the edge layer, to powerful cellular area and backbone networks in the cloud layer [53]. When an edge controller is overloaded, we can redirect packet-in requests of an edge device to cloud controllers. Note that the size of a packet-in request is small (128 Bytes), and the bandwidth overhead between edge and cloud networks is marginal. On the contrary, neither dynamic controller assignment nor flow redirection in the data plane is fit for edge networks. For dynamic controller assignment, an edge device cannot arbitrarily assign to another edge controller, for the delay of wireless connection has a large variance. Thus, the assignment of edge devices and controllers is tightly bound after initial deployment. For flow redirection in the data plane, flow redirection occupies valuable bandwidth resource in wireless edge networks.

Request redirection to multiple controllers. Redirecting packet-in requests to multiple controllers incurs extra performance overhead. A controller needs to divide the packet-in requests into multiple parts, *e.g.*, partitioning with hashing. The partitioning needs to deserialize packet-in requests, which is time-consuming. In contrast, when redirecting all packet-in requests to a single controller, some flexible NICs can directly forward a packet based on its header fields without packet deserialization. Consequently, request redirection to a single controller is more lightweight. Recently, SmartNICs supported by data processing units (DPUs) can execute more complex operations. Those SmartNICs may accelerate the request redirection to multiple controllers. We will consider this situation in future work.

7. Evaluation result

In this section, we first introduce our approach to implementing a controller prototype based on Floodlight [26] to support packet-in request redirection. Then, we conduct simulations to verify the performance of our proposed algorithms with three other algorithms.

7.1. Simulation setup

Implementation. Our implementation of packet-in request redirection uses Floodlight [26], an easy-to-extend controller in Java. Using the `SyncManager` module provided by Floodlight, we enable multiple Floodlight controllers to behave as a distributed control plane and cooperatively handle network events. Furthermore, we force the Floodlight controller to process LLDP packets that are not generated by itself, such that we can detect the links between two switches, which are not assigned to a same controller. When the packet-in requests need to be redirected between two controllers, we set up a network channel with pre-allocated network buffer, which is kept active and carefully maintained in a customized time interval to reduce extra setup time for subsequent request redirection.

Topology and testbed. We conduct our simulations on the two wide-used topologies, *i.e.*, fat-tree [54] and VL2 [55]. For fat-tree, the network consists of 8 pods and possesses 80 switches and 128 hosts in total. For VL2, the degree of intermediate switches (D_I) and the degree of aggregate switches (D_A) are set to 20, and thus the network possesses 130 switches and 100 racks with three hosts each. We deploy 10 controllers in the network for network events handling, and the processing rate of each controller is 18k packet-in requests per second [31], which is sufficient to accommodate the peak of total packet-in requests from all switches. One switch is statically assigned to one controller.

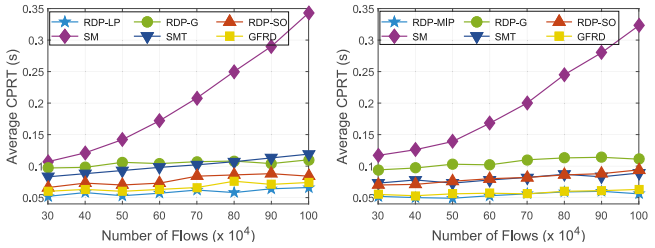
For edge networks, we use the MANIAC mobile ad hoc network in [56], which contains 14 nodes (edge devices). We deploy 10 edge controllers and 1 cloud controller. The processing rate of an edge controller is 4k packet-in requests per second, while that of a cloud controller is 18k packet-in requests per second. The total processing capacity of edge controllers is sufficient to process all flows in the networks. The cloud controller can help all edge controllers to process temporarily overloaded requests. The connection between an edge device and its assigned controller is wireless, while the connection between an edge controller and the cloud controller is wired.

We run experiments on three cloud servers with 32 GB of RAM and 32 physical cores. We construct the topologies using Mininet [57] in a desktop PC, and serve cloud servers as controllers. We deploy four controller instances in one server. Consequently, controllers in a same server have low delay, while those in different servers suffer long delay.

System parameters and settings. We set the duration of a time slot, *i.e.*, δ, to five minutes, considering that the packet-in requests arrival rates slightly change in that time period. Besides, we extend δ to 20 minutes for RDP-MIP in VL2 for the fast growing scale of MIP. The distribution of packet-in requests arrival rates follows the data captured in the real-world data center [15]. Moreover, the system perceives the instantaneous arrival rate for each switch, and updates the values in $\lambda(t)$ every five minutes.

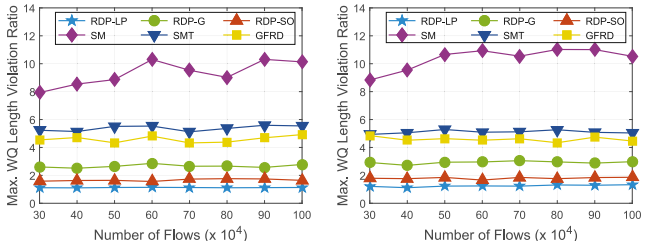
7.2. Comparison algorithms

We not only compare our algorithms, *i.e.*, RDP-MIP, RDP-G, and RDP-SO but involve three other comparison schemes. **(1) SM:** The static matching (SM) scheme, where the association between controllers and switches is fixed. We consider the results of **SM** as the baseline. **(2) SMT:** Wang et al. [21] proposed a controller load balance scheme named as stable matching with transfer (SMT), where a switch can dynamically change its associated controller. **(3) GFRD:** Wang et al. [24] considered flow redirection in the data plane to achieve the same goal in [21], and presented the greedy flow redirecting (GFRD) algorithm.



(a) Average CPRT in fat-tree. (b) Average CPRT in VL2.

Fig. 4. Average CPRT for different topologies.



(a) Maximum queue length violation ratio in fat-tree. (b) Maximum queue length violation ratio in VL2.

Fig. 5. Maximum queue length violation for different topologies.

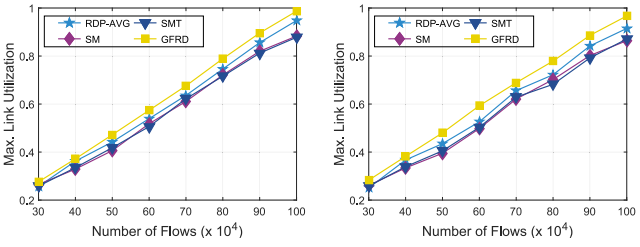
7.3. Performance comparison

Average CPRT. Our simulation results show in the worst case (largest number of flows), RDP algorithms can reduce average CPRT by 81.7% compared to SM, with similar effects to SMT and GFRD. Figs. 4(a) and 4(b) show that average CPRT of RDP algorithms, SMT, and GFRD fluctuates slightly, while that of SM linearly increases with the number of flows. Since SM does not execute workload balancing, some controllers suffer from requests overload at the peak of requests arrival rate. Moreover, we observe that on average RDP-MIP outperforms RDP-G and RDP-SO by 46.1% and 28.4%, respectively.

Maximum waiting queue length violation ratio. We specify the values of capacity constraints of waiting queues $\{M_1, \ldots, M_K\}$ within the range of 100 MB to 400 MB at random. Our simulation results show that RDP algorithms can well control the backlog in each controller without seriously violating the capacity constraints. Figs. 5(a) and 5(b) show the maximum waiting queue length violation ratios of RDP algorithms are around 3× smaller than those of SMT and GFRD. This is because SMT and GFRD operate load balancing on all controllers and neglect the capacity difference.

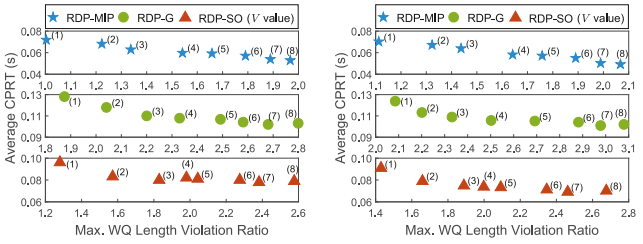
Maximum link utilization. We represent RDP-AVG as the average value of maximum link utilization for RDP-MIP, RDP-G, and RDP-SO. Our simulation results in Figs. 6(a) and 6(b) show that the increasing ratio of maximum link utilization of RDP algorithms is less than 8% compared to SM in both fat-tree and VL2. For the cases that the numbers of flows are small, the impact of packet-in request redirection is negligible. It is because the packet-in messages only carry the first 128 Bytes of the table-miss flows, and cost little link resource overhead in the network.

Weighted parameter and running time. Fig. 7(a) shows that the tradeoff between average CPRT and the maximum waiting queue length violation ratio can be achieved by varying the weighted parameter V. We observe that the decline rate of average CPRT slows down as V increases, so that we can select a moderate value of V, *e.g.*, around three. Fig. 8 shows that on average the running time of RDP-G and RDR-SO is less than five seconds, however, that of RDP-MIP rapidly increases as the scale of the MIP expands. Therefore, RDP-MIP is not suitable for use as the number of switches exceeds 100. In this case, RDP-G and RDP-SO can efficiently obtain a solution and are of practical use.



(a) Maximum link utilization of all links in fat-tree. (b) Maximum link utilization of all links in VL2.

Fig. 6. Maximum link utilization of all links with packet-in request redirection.



(a) Impact on weighted parameter V in fat-tree. (b) Impact on weighted parameter V in VL2.

Fig. 7. Impact on weighted parameter V for different topologies.

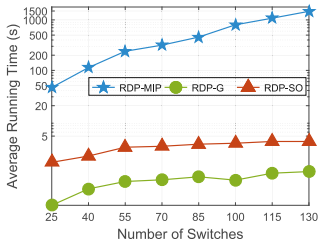


Fig. 8. Average running time of RDP algorithms.

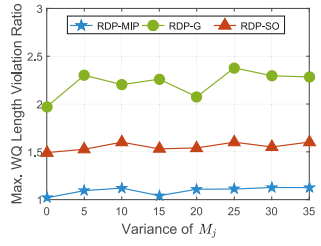


Fig. 9. Impact on controller capacity constraints.

Controller capacity constraints. We let the distribution of controller capacity constraints, *i.e.*, $\{M_1, \ldots, M_K\}$, comply with the normal distribution, of which the mean is 200 MB, while the variance is a variable and within the range of $[0, 35]$. Fig. 9 shows that the maximum waiting queue violation ratios for RDP-MIP, RDP-G, and RDP-SO are on average 1.09, 2.22, and 1.56, respectively. Thus, RDP algorithms can provide accurate control on packet-in request redirection to meet the capacity constraints of controllers.

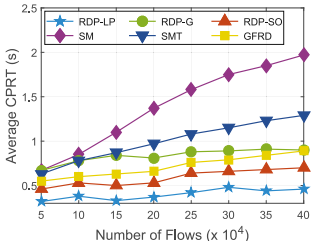


Fig. 10. Average CPRT in edge networks.

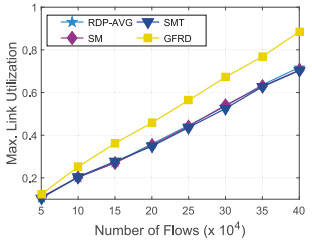


Fig. 11. Maximum link utilization of all links in edge networks.

7.4. Performance in edge networks

Average CPRT. Fig. 10 presents the average CPRT in edge networks. RDP algorithms can reduce average CPRT by 76.7% compared to SM in the worst case. The average CPRT of SMT and GFRD goes up with the increase of the number of flows. For SMT, the switch-controller re-assignment enlarges the communication delay between an edge device and its assigned controller. For GFRD, the flow redirection in the data plane needs extra hops in the wireless network, which takes 43.7% more time compared to RDP algorithms.

Maximum link utilization. Our simulation results in Fig. 11 show that RDP-AVG has little overhead of bandwidth usage. The connection between an edge controller and the cloud controller is wired, and the request redirection between them occupies little bandwidth resources of the wired connection. However, the increasing ratio of maximum link utilization of GFRD is 19.6% compared to the RDP algorithms. GFRD requires more bandwidth resources to reroute flows in the data plane, and the flow redirection aggravates the utilization of the wireless links in the edge.

8. Conclusion

The key novelty of this paper is on studying the first scheme for CPRT minimization using packet-in request redirection. The key contribution of this paper is building the packet-in request redirection model, developing an online framework, and implementing a controller prototype for packet-in request redirection. The key technical depth of this paper is in involving Lyapunov optimization into online redirection decision making and proposing two approximation algorithms using the greedy strategy and submodular optimization. Our simulation results show that our proposed algorithms reduce average CPRT by 81.6% compared to static matching, and achieve a 3× improvement in maximum waiting queue length violation ratio compared to the related works on CPRT minimization.

Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.sysarc.2022.102590.

Appendix A. Proof of Theorem 4.4

Proof. Assuming the distribution of $\lambda(t)$ is known, we consider $\boldsymbol{X}^*(t)$ as the offline optimal result, which is the optimal solution to the problem of minimizing the expectation of overall CPRT on slot t, such that $\mathbb{E}[C(t)] = C^*$. We denote $\boldsymbol{Q}^*(t)$ as the queue backlogs for $\boldsymbol{X}^*(t)$. Our algorithms attempt to minimize the objective function in RDP, and thus

$$\begin{aligned}&\Delta(\boldsymbol{Z}(t)) + V \cdot \mathbb{E}[C(t)]\\ &\le B' + V \cdot C^* + \sum_{j=1}^{K} Z_j(t) \cdot \mathbb{E}[(Q_j^*(t+1) - M_j)|\boldsymbol{Z}(t)]\\ &\le B' + V \cdot C^*. \end{aligned} \tag{25}$$

Note that $\boldsymbol{X}^*(t)$ is a feasible solution, so that $\mathbb{E}[(Q_j^*(t+1) - M_j)|\boldsymbol{Z}(t)] \le 0$. Add up In Eq. (25) from $t = 1$ to T, and divide $V \cdot T$ at the both sides:

$$\frac{1}{V \cdot T}\sum_{t=1}^{T}\Delta(\boldsymbol{Z}(t)) + \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[C(t)] \le \frac{B'}{V} + C^*. \tag{26}$$

Since $L(\boldsymbol{Z}(\boldsymbol{0})) = 0$ and $L(\boldsymbol{Z}(t)) \ge 0$, we imply that

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[C(t)] \le C^* + \frac{B'}{V}.$$

Therefore, we prove that the gap between our online decision and the offline optimal result is at most $O(1/V)$.

For the time average of each waiting queue $\overline{Q_j}$, we first bound the Lyapunov function using In Eq. (26):

$$L(\boldsymbol{Z}(\boldsymbol{T})) \le T(B' + V \cdot C^*).$$

Furthermore, we can derive that

$$\mathbb{E}[Z_j^2(T)] \le T(B' + V \cdot C^*).$$

Therefore, we get the lower and upper bounds of $\mathbb{E}[Z_j(t)]/T$:

$$0 \le \mathbb{E}[Z_j(T)]/T \le \sqrt{\mathbb{E}[Z_j^2(T)]}\Big/T \le \sqrt{(B' + V \cdot C^*)/T}.$$

When $T \to \infty$, we have $\limsup_{T\to\infty}\mathbb{E}[Z_j(T)]/T = 0$, which infers our target, *i.e.*, $\overline{Q_j} \le M_j$, using Theorem 4.1. □

Appendix B. Proof of Theorem 5.1

Proof. For the redirection decision in Eq. (19), we use shorthand to represent its type-one and type-two costs:

$$A = \sum_{i=1}^{N}\sum_{j=1}^{K} X_{ij}^{(0)}(t) Cost(i,j), \quad B = \sum_{i=1}^{N}\sum_{j=1}^{K} X_{ij}^{(0)}(t)\theta_j'(t).$$

We denote $\widehat{SOL}$ as the value of the objective function derived from the initial redirection decision, and specifically, $\widehat{SOL}$ is equal to $A + B$. Moreover, we define another greedy redirection decision as follows:

$$X_{ij}'(t) = \begin{cases} 1\ , j = \arg\min_j Cost(i,j) \\ 0\ , else, \end{cases}$$

and its type-one costs as $A' = \sum_{i=1}^{N}\sum_{j=1}^{K}[X_{ij}'(t) Cost(i,j)]$.

Lemma B.1. *Given $1+\eta = C_{max}/C_{min}$, the ratio of A to A' is within $1+\eta$, that is, $A/A' \le 1+\eta$.*

Proof. Since the initial redirection decision is greedily selected based on $Cost(i,j) \cdot C_j$, we have

$$\sum_{j=1}^{K}\sum_{i=1}^{N} X_{ij}^{(0)}(t)\left[Cost(i,j) \cdot C_j\right] \le \sum_{j=1}^{K}\sum_{i=1}^{N} X_{ij}'(t)\left[Cost(i,j) \cdot C_j\right].$$

Extracting the term of C_j in above inequation, we obtain

$$C_{min}\sum_{i=1}^{N}\sum_{j=1}^{K}X_{ij}^{(0)}(t)Cost(i,j)\le C_{max}\sum_{i=1}^{N}\sum_{j=1}^{K}X'_{ij}(t)Cost(i,j).$$

We imply that $A/A'\le C_{max}/C_{min}=1+\eta$, and the lemma holds. □

We denote OPT as the optimal solution to RDP. Note that $OPT\ge A'$, and we can bound the approximation ratio as follows:

$$\widehat{SOL}/OPT\le\widehat{SOL}/A'\le(A+B)/A'.$$

Lemma B.2. *Ratio B to A is within $\delta\cdot V\max\{\lambda(t)\}C_{max}$.*

Proof. Using the definition of B, we derive that

$$\begin{aligned}
B&=\delta\cdot V\sum_{j=1}^{K}\frac{1}{\alpha_j}\sum_{i=1}^{N}\sum_{l=1}^{N}X_{ij}^{(0)}(t)X_{lj}^{(0)}(t)\cdot\lambda_l(t)\\
&\le\delta\cdot V\sum_{j=1}^{K}\frac{1}{\alpha_j}\Big(\sum_{i=1}^{N}X_{ij}^{(0)}(t)\Big)\Big(\sum_{l=1}^{N}X_{lj}^{(0)}(t)\cdot\lambda_l(t)\Big)\\
&\le\delta\cdot V\max\{\lambda(t)\}\sum_{j=1}^{K}\Big\{\frac{1}{\alpha_j}\Big(\sum_{i=1}^{N}X_{ij}^{(0)}(t)\Big)^2\Big\}\\
&\overset{*}{\le}\delta\cdot V\max\{\lambda(t)\}\cdot\sum_{j=1}^{K}\Big\{\frac{1}{\alpha_j}\Big(\sum_{i=1}^{N}X_{ij}^{(0)}(t)Cost(i,j)\Big)\Big(\sum_{l=1}^{N}\frac{1}{Cost(l,j)}\Big)\Big\}\\
&\le\delta\cdot V\max\{\lambda(t)\}\sum_{j=1}^{K}\Big\{\Big(\sum_{i=1}^{N}X_{ij}^{(0)}(t)Cost(i,j)\Big)\cdot C_j\Big\}\\
&\le\delta\cdot V\max\{\lambda(t)\}C_{max}A.
\end{aligned}$$

The inequation $(*)$ is caused by the Cauchy Schwarz's inequality. This completes the proof. □

Combining Lemmas B.1 and B.2, we have

$$\begin{aligned}
\widehat{SOL}/OPT&\le(A+B)/A'\le(A/A')\cdot\big(1+\delta\cdot V\max\{\lambda(t)\}C_{max}\big)\\
&\le(1+\eta)\big(1+\delta\cdot V\max\{\lambda(t)\}C_{max}\big).
\end{aligned}$$

Thus, the theorem follows. □

Appendix C. Proof of Theorem 5.2

Proof. We assume that $A\subseteq B\subseteq W$, and there exists a new element $e\in W\backslash B$. We denote $E(e,A)$ as the event, that the value of $\sum_{i=1}^{N}\mathbb{1}(|A\cap W_{i\bullet}|\neq 0)$ keeps *unchanged* after adding the element e, and express it formally as follows:

$$\sum_{i=1}^{N}\mathbb{1}(|A\cap W_{i\bullet}|\neq 0)=\sum_{i=1}^{N}\mathbb{1}(|\{A\cup\{e\}\}\cap W_{i\bullet}|\neq 0).$$

Namely, when $E(e,A)$ happens, an element in $W_{i\bullet}$, where the element e is, has already been selected by A. On the contrary, we denote $\overline{E(e,A)}$ as the event that the value of $\sum_{i=1}^{N}\mathbb{1}(|A\cap W_{i\bullet}|\neq 0)$ is incremented by one after adding the element e. Similarly, we define $E(e,B)$ and $\overline{E(e,B)}$ for the set B with the same meanings as $E(e,A)$ and $\overline{E(e,A)}$.

We denote the marginal return of adding the element e to the subset A as $\Delta f(e|A)=f(A\cup\{e\})-f(A)$. For each $c_j\in C$, the mathematical formula of the switches associated with c_j can be presented as $U\cap W_{\bullet j}$. Thus, for the corresponding switches in $U\cap W_{\bullet j}$, we express the summation of their type-two costs as follows:

$$g(U)=\sum_{j=1}^{K}|U\cap W_{\bullet j}|\cdot\sum_{e\in U\cap W_{\bullet j}}\lambda(e).$$

Combining $\{E(e,A),\overline{E(e,A)}\}$ with $\{E(e,B),\overline{E(e,B)}\}$, we have four conditions to discuss.

Condition One. $E(e,A)\wedge E(e,B)$. In this case, both events incur identical type-one cost increments. Whereas the type-two cost increments for set A and set B are different, we present the gap of marginal returns for set A and set B as follows:

$$\begin{aligned}
\Delta f(e|B)-\Delta f(e|A)=\delta\cdot V\Big\{&\big[g(B\cup\{e\})-g(B)\big]\\
&-\big[g(A\cup\{e\})-g(A)\big]\Big\}.
\end{aligned}$$

Specifically, the value of marginal return for $g(U)$ is

$$g(U\cup\{e_{ij}\})-g(S)=|U\cap W_{\bullet j}|\cdot\lambda(e_{ij})+\sum_{e\in U\cup W_{\bullet j}}\lambda(e).\tag{27}$$

For a new element $e_{ij}\in W$, the first part of RHS in Eq. (27) is the cost increment for already selected elements. Note that, only the elements within $U\cap W_{\bullet j}$ have cost increment, and the costs of the rest remain unchanged. The second part is the type-two cost for the new element.

Assuming $A\subseteq B$, we have $|A\cap W_{\bullet j}|\le|B\cap W_{\bullet j}|$. Furthermore, we can derive $(A\cap W_{\bullet j})\subseteq(B\cap W_{\bullet j})$, and thus

$$g(B\cup\{e_{ij}\})-g(B)\ge g(A\cup\{e_{ij}\})-g(A).$$

The supermodularity of $f(U)$ is proved in the condition one.

Condition Two. $\overline{E(e,A)}\wedge E(e,B)$. In this case, $\Delta f(e|B)$ is as same as that in the condition one, while $\Delta f(e|A)$ has a little difference, and is expressed as

$$\Delta f(e|A)=\delta\cdot V\Big[g(A\cup\{e\})-g(A)\Big]-P,$$

where P is a penalty parameter. The penalty value will be added to the cost, whenever the intersection of U and $W_{i\bullet}$ is empty. Namely, the current selected set U is not a feasible solution to TRDP, when P is attached to the objective function. Using the result in the condition one, we infer that

$$\begin{aligned}
\Delta f(e|B)&\ge\delta\cdot V\Big[g(A\cup\{e\})-g(A)\Big]\\
&\ge\delta\cdot V\Big[g(A\cup\{e\})-g(A)\Big]-P=\Delta f(e|A).
\end{aligned}$$

The supermodularity of $f(U)$ is proved in the condition two.

Condition Three. $\overline{E(e,A)}\wedge\overline{E(e,B)}$. In this condition, both $\Delta f(e|B)$ and $\Delta f(e|A)$ have the term of P, specifically,

$$\Delta f(e|A)=\delta\cdot V\Big[g(A\cup\{e\})-g(A)\Big]-P,\tag{28}$$

$$\Delta f(e|B)=\delta\cdot V\Big[g(B\cup\{e\})-g(B)\Big]-P.\tag{29}$$

Removing the term of P in Eqs. (28) and (29), we see that this condition is analogous to the condition one.

Condition Four. $E(e,A)\wedge\overline{E(e,B)}$. Assuming $A\subseteq B$, we assert that this condition can never happen. □

References

[1] R. Xia, H. Dai, J. Zheng, H. Xu, M. Li, G. Chen, Packet-in request redirection for minimizing control plane response time, in: IEEE IPDPS, 2020.

[2] C. Hu, K. Hou, H. Li, R. Wang, P. Zheng, P. Zhang, H. Wang, SoftRing: Taming the reactive model for software defined networks, in: IEEE ICNP, 2017.

[3] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, S. Shenker, SCL: Simplifying distributed SDN control planes, in: USENIX NSDI, 2017.

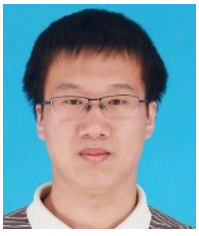
[4] A. Krishnamurthy, S.P. Chandrabose, A. Gember-Jacobson, Pratyaastha: an efficient elastic distributed sdn control plane, in: ACM HotSDN, 2014.

[5] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven WAN, in: ACM SIGCOMM, 2013.

[6] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, B.K. Naidu, C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, A. Vahdat, B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN, in: ACM SIGCOMM, 2018.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, B4: Experience with a globally-deployed software defined wan, ACM SIGCOMM (2013).

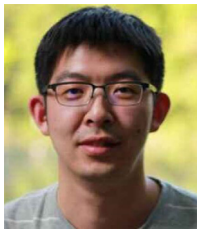
[8] D. Espinel Sarmiento, A. Lebre, L. Nussbaum, A. Chari, Decentralized SDN control plane for a distributed cloud-edge infrastructure: A survey, IEEE Commun. Surv. Tutor. 23 (1) (2021) 256–281.
[9] Q. Qiaofeng, P. Konstantinos, I. George, T. Leandros, SDN controller placement at the edge: Optimizing delay and overheads, in: IEEE INFOCOM, 2018.
[10] E. Ahvar, S. Ahvar, S.M. Raza, J. Manuel Sanchez Vilchez, G.M. Lee, Next generation of SDN in cloud-fog for 5G and beyond-enabled applications: Opportunities and challenges, Network 1 (1) (2021) 28–49.
[11] B. Alzahrani, N. Fotiou, Enhancing internet of things security using software-defined networking, J. Syst. Archit. 110 (2020) 101779.
[12] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, et al., Onix: A distributed control platform for large-scale production networks, in: USENIX OSDI, 2010.
[13] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with DIFANE, in: ACM SIGCOMM, 2010.
[14] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al., ONOS: towards an open, distributed SDN OS, in: ACM HotSDN, 2014.
[15] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: ACM IMC, 2010.
[16] W. John, S. Tafvelin, T. Olovsson, Trends and differences in connection-behavior within classes of internet backbone traffic, in: International Conference on Passive and Active Network Measurement, Vol. 4979, Springer, 2008, pp. 192–201.
[17] G. Yao, J. Bi, Y. Li, L. Guo, On the capacitated controller placement problem in software defined networks, IEEE Commun. Lett. 18 (8) (2014) 1339–1342.
[18] A. Sallahi, M. St-Hilaire, Optimal model for the controller placement problem in software defined networks, IEEE Commun. Lett. 19 (1) (2015) 30–33.
[19] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, M. Hoffmann, Heuristic approaches to the controller placement problem in large scale SDN networks, IEEE Trans. Netw. Serv. Manag. 12 (1) (2015) 4–17.
[20] M.F. Bari, A.R. Roy, S.R. Chowdhury, Q. Zhang, M.F. Zhani, R. Ahmed, R. Boutaba, Dynamic controller provisioning in software defined networks, in: IEEE CNSM, 2013.
[21] T. Wang, F. Liu, J. Guo, H. Xu, Dynamic SDN controller assignment in data center networks: Stable matching with transfers, in: IEEE INFOCOM, 2016.
[22] X. Huang, S. Bian, Z. Shao, H. Xu, Dynamic switch-controller association and control devolution for SDN systems, in: IEEE ICC, 2017.
[23] G. Cheng, H. Chen, Z. Wang, S. Chen, DHA: Distributed decisions on the switch migration toward a scalable SDN control plane, in: IFIP Networking, 2015.
[24] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, Y. Sun, Minimizing controller response time through flow redirecting in SDNs, IEEE/ACM Trans. Netw. 26 (1) (2018) 562–575.
[25] L. Schiff, S. Schmid, P. Kuznetsov, In-band synchronization for distributed SDN control planes, ACM SIGCOMM Comput. Commun. Rev. 46 (1) (2016) 37–43.
[26] Floodlight, 2016, http://www.projectfloodlight.org/floodlight/.
[27] J. Cui, Q. Lu, H. Zhong, M. Tian, L. Liu, A load-balancing mechanism for distributed SDN control plane using response time, IEEE Trans. Netw. Serv. Manag. 15 (4) (2018) 1197–1206.
[28] X. Huang, S. Bian, Z. Shao, H. Xu, Predictive switch-controller association and control devolution for SDN systems, in: IEEE/ACM IWQoS, 2019.
[29] P. Wang, H. Xu, L. Huang, J. He, Z. Meng, Control link load balancing and low delay route deployment for software defined networks, IEEE J. Sel. Areas Commun. 35 (11) (2017) 2446–2456.
[30] H. Xu, J. Liu, C. Qian, H. Huang, C. Qiao, Reducing controller response time with hybrid routing in software defined networks, Comput. Netw. 164 (2019) 106891.
[31] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an elastic distributed SDN controller, in: ACM HotSDN, 2013.
[32] R. Chai, X. Yang, C. Du, Q. Chen, Network cost optimization-based capacitated controller deployment for SDN, Comput. Netw. 197 (2021) 108326.
[33] M. He, A. Basta, A. Blenk, W. Kellerer, Modeling flow setup time for controller placement in sdn: Evaluation for dynamic flows, in: IEEE ICC, 2017.
[34] S. Petale, J. Thangaraj, Failure-based controller placement in software defined networks, IEEE Trans. Netw. Serv. Manag. 17 (1) (2020) 503–516.
[35] Y. Fan, L. Wang, X. Yuan, Controller placements for latency minimization of both primary and backup paths in SDNs, Comput. Commun. 163 (2020) 35–50.
[36] T. Hu, Q. Ren, P. Yi, Z. Li, J. Lan, Y. Hu, Q. Li, An efficient approach to robust controller placement for link failures in software-defined networks, Future Gener. Comput. Syst. 124 (2021) 187–205.
[37] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: ACM HotSDN, 2012.
[38] T. Das, V. Sridharan, M. Gurusamy, A survey on controller placement in SDN, IEEE Commun. Surv. Tutor. 22 (1) (2020) 472–503.
[39] OpenDayLight, 2014, https://www.opendaylight.org.
[40] A. Tootoonchian, Y. Ganjali, HyperFlow: A distributed control plane for OpenFlow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, 2010, pp. 1–6.
[41] S. Hassas Yeganeh, Y. Ganjali, Kandoo: A framework for efficient and scalable offloading of control applications, in: ACM HotSDN, 2012.
[42] ONOS, 2014, http://onosproject.org.
[43] Z. Zhang, L. Ma, K.K. Leung, F. Le, More is not always better: An analytical study of controller synchronizations in distributed SDN, IEEE/ACM Trans. Netw. 29 (4) (2021) 1580–1590.
[44] K. Poularakis, Q. Qin, L. Ma, S. Kompella, K.K. Leung, L. Tassiulas, Learning the optimal synchronization rates in distributed SDN control architectures, in: IEEE INFOCOM, 2019.
[45] S.-C. Lin, P. Wang, I.F. Akyildiz, M. Luo, Towards optimal network planning for software-defined networks, IEEE Trans. Mob. Comput. 17 (12) (2018) 2953–2967.
[46] M. Aslan, A. Matrawy, Adaptive consistency for distributed SDN controllers, in: International Telecommunications Network Strategy and Planning Symposium, 2016, pp. 150–157.
[47] E. Sakic, F. Sardis, J.W. Guck, W. Kellerer, Towards adaptive state consistency in distributed SDN control plane, in: IEEE ICC, 2017.
[48] A.S. Muqaddas, P. Giaccone, A. Bianco, G. Maier, Inter-controller traffic to support consistency in ONOS clusters, IEEE Trans. Netw. Serv. Manag. 14 (4) (2017) 1018–1031.
[49] E. Sakic, W. Kellerer, Response time and availability study of RAFT consensus in distributed SDN control plane, IEEE Trans. Netw. Serv. Manag. 15 (1) (2018) 304–318.
[50] A. Kaufmann, S. Peter, N.K. Sharma, T. Anderson, A. Krishnamurthy, High performance packet processing with flexnic, in: ACM ASPLOS, 2016.
[51] C. Chekuri, S. Khanna, A polynomial time approximation scheme for the multiple knapsack problem, SIAM J. Comput. 35 (3) (2005) 713–728.
[52] A. Ene, H.L. Nguyen, Constrained submodular maximization: Beyond 1/e, in: IEEE FOCS, 2016.
[53] K. Alwasel, D.N. Jha, F. Habeeb, U. Demirbaga, O. Rana, T. Baker, S. Dustdar, M. Villari, P. James, E. Solaiman, R. Ranjan, IoTSim-Osmosis: A framework for modeling and simulating IoT applications over an edge-cloud continuum, J. Syst. Archit. 116 (2021) 101956.
[54] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: ACM SIGCOMM, 2008.
[55] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, in: ACM SIGCOMM, 2009.
[56] A. Hilal, J.N. Chattha, V. Srivastava, M.S. Thompson, A.B. MacKenzie, L.A. DaSilva, P. Saraswati, CRAWDAD dataset vt/maniac (v. 2011-07-21), 2011, Downloaded from https://crawdad.org/vt/maniac/20110721.
[57] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: ACM Hotnets, 2010, pp. 1–6.



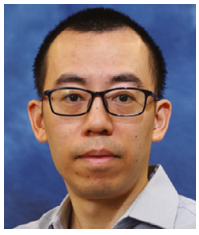
Rui Xia received the B.S. degree at the Department of Software Engineering, Nankai University, China, in 2017. He is currently working toward the Ph.D. degree at the Department of Computer Science, Nanjing University, China. His current research interests include software-defined network and network function virtualization.

Haipeng Dai received the B.S. degree in the Department of Electronic Engineering from Shanghai Jiao Tong University, Shanghai, China, in 2010, and the Ph.D. degree in the Department of Computer Science and Technology in Nanjing University, Nanjing, China, in 2014. His research interests are mainly in the areas of data mining, Internet of Things, and mobile computing. He is an associate professor in the Department of Computer Science and Technology in Nanjing University. His research papers have been published in many prestigious conferences and journals such as ACM UbiComp, IEEE INFOCOM, VLDB, IEEE ICDE, ACM WWW, ACM SIGMETRICS, ACM MobiHoc, ACM MobiSys, IEEE ICNP, IEEE ICDCS, ACM/IEEE TON, IEEE JSAC, IEEE TPDS,

IEEE TMC, IEEE TKDE, IEEE TDSC, and IEEE TIFS. He is an IEEE and ACM member. He serves/ed as Poster Chair of the IEEE ICNP'14, Track Chair of the ICCCN'19 and the ICPADS'21, TPC member of the ACM MobiHoc'20–21, IEEE INFOCOM'20–22, IJCAI'21–22, IEEE SC'22, IEEE ICDCS'20–21, IEEE ICNP'14, IEEE IWQoS'19–21, and IEEE IPDPS'20–22. He received Best Paper Award from IEEE ICNP'15, Best Paper Award Runner-up from IEEE SECON'18, and Best Paper Award Candidate from IEEE INFOCOM'17.



Jiaqi Zheng is currently a Research Assistant Professor from Department of Computer Science and Technology, Nanjing University, China. His research area is computer networking, particularly data center networks, SDN/NFV, machine learning system and online optimization. He was a Research Assistant at the City University of Hong Kong in 2015 and collaborated with Huawei Noah's Ark Lab. He visited CIS center at Temple University in 2016. He received the Best Paper Award from IEEE ICNP 2015, Outstanding Doctoral Dissertation Award from ACM SIGCOMM China 2018, the First Prize of Jiangsu Science and Technology Award in 2018, Outstanding Doctoral Dissertation Award from CCF, Jiangsu Province and Nanjing University in 2019. He is a member of ACM and IEEE.



Hong Xu is an Associate Professor in Department of Computer Science and Engineering, The Chinese University of Hong Kong. His research area is computer networking and systems, particularly big data systems and data center networks. From 2013 to 2020 he was with City University of Hong Kong. He received his B.Eng. from The Chinese University of Hong Kong in 2007, and his M.A.Sc. and Ph.D. from University of Toronto in 2009 and 2013, respectively. He was the recipient of an Early Career Scheme Grant from the Hong Kong Research Grants Council in 2014. He received three best paper awards, including the IEEE ICNP 2015 best paper award. He is a senior member of IEEE and ACM.

Meng Li received his B.E. degree in Computer Science from Nanjing University, Jiangsu, China, in 2016. He is currently a Ph.D. student in Nanjing University. His research interests are in the area of data mining.

Guihai Chen received B.S. degree in computer software from Nanjing University in 1984, M.E. degree in computer applications from Southeast University in 1987, and Ph.D. degree in computer science from the University of Hong Kong in 1997. He is a professor and deputy chair of the Department of Computer Science, Nanjing University, China. He had been invited as a visiting professor by many foreign universities including Kyushu Institute of Technology, Japan in 1998, University of Queensland, Australia in 2000, and Wayne State University, USA during Sept. 2001 to Aug. 2003. He has a wide range of research interests with focus on sensor networks, peer-to-peer computing, high-performance computer architecture and combinatorics.