PrePass: Load Balancing with Data Plane Resource Constraints using Commodity SDN Switches (Extended Version)

Haibo Wang[1] *Hongli Xu[1] Liusheng Huang[1] Chen Qian[2]
[1]*University of Science and Technology of China* [2]*University of California Santa Cruz*

Abstract

Due to versatility and universality of network applications, network layer load balancing is crucial to ensure operational efficiency in a network with a variety of workloads. Software defined networking (SDN) is a promising technique for load balancing. However, limited resources (*e.g.*, TCAM and computing capacity) on SDN switches bring critical challenges for load balancing. On one hand, though some solutions (*e.g.*, ECMP) satisfy resource constraints, these methods do not work well especially for network asymmetry and traffic dynamics. On the other hand, most previous works (*e.g.*, DevoFlow, Presto, LetFlow, and LocalFlow) achieve load balancing with additional hardware or software resources, which increase the system cost and limit the applicability. Thus, this paper tries to deal with the following challenge: *how to achieve load balancing without additional device and software on commodity switches while dealing with network/traffic uncertainties*? To this end, we design and implement PrePass, which uses wildcard entries for some aggregated flows to satisfy the flow table size constraint, and perform reactive routing for newly arrived flows to achieve load balancing. We define the problem of load balancing with flow table size constraint, and prove its NP-hardness. We then present an efficient algorithm based on randomized rounding, and analyze that our algorithm can achieve constant bi-criteria approximation under most practical situations. To make our problem more robust, an extended version is also studied. We implement PrePass on a real SDN testbed. The experimental results and extensive simulation results show that our proposed method can satisfy different resource constraints on switches, and only increase the link load ratio by about 5%-10% compared with per-flow routing scheme under various traffic scenarios.

1 Introduction

Network layer load balancing is an important technique that uses routing decisions to avoid potential congestion on certain links and increase the network bandwidth utilization. Load balancing is important for many types of networks including ISP, data centers [5], WAN [20], and enterprise networks [11]. In particular, modern networks should support an increasingly diverse set of workloads, ranging from small latency-sensitive flows (*e.g.*, search or RPCs [18]) to bandwidth-hungry large flows (*e.g.*, big data analytics [10], VM migration [22], or video [26]). To better serve a diversity of flows, load balancing is crucial to ensure operational efficiency and suitable application performance.

As a recent trend, Software Defined Networking (SDN) has been widely used in modern networks. In a typical SDN, the controller monitors the network and determines the forwarding paths of traffic flows. The switches carry out different operations (*e.g.*, forwarding or dropping) for flows/packets based on the rules installed by the controller. Since the controller is able to implement centralized and reactive control for each flow through the packet header reporting mechanism [29], an SDN can provide fine-grained flow management and route control, which help to improve the network bandwidth utilization compared with traditional network load balancing [20] [37] [38]. However, limited resources on SDN switches pose additional difficulty for load balancing. Specifically, two types of resources are of major concerns. **1) Memory resource.** Since Ternary Content Addressable Memory (TCAM) is expensive and power hungry, the size of a TCAM-based flow table is often limited (*e.g.*, 1500 entries on HP 5406zl switch [11]). **2) Computing resource.** Switches have special hardware for packet forwarding processing. In addition, as specified in the OpenFlow standard [29], each OpenFlow capable switch has a software implemented OpenFlow agent (OFA) for computing other than packet processing, which typically runs on a low-end CPU with limited processing power [35]. *These resource constraints certainly limit how individual flows can be controlled in the network for best load balancing*.

A natural and common way for load balancing is ECMP-based multi-path forwarding [4] [17]. This mechanism distributes the traffic of different flows on multiple equal-cost paths. However, it may perform poorly for dynamic flow traffic, which is common in today's networks. To deal with this weakness, the weighted multiple paths, called WCMP [43], is designed. It assigns different weights for paths based on link load distribution. These solutions of multi-path forwarding have three main disadvantages. First, it may perform poorly in asymmetric topologies, which are common in today's networks due to heterogeneous network components or link/device failures [6] [43]. Second, since the multipath forwarding is usually implemented based on flow hashing, it may cause link congestion when hash collisions occur [5] [8] [12] [31]. Third, these methods often route flows based on the traffic prediction. Due to traffic dynamics, it may still lead to load imbalance without immediate flow rerouting.

There have been many SDN-based load balancing solutions proposed to conquer the disadvantages of ECMP-like multi-path forwarding. Most, if not all, of them may incur problems of resource constraints on commodity switches. To deal with the resource constraints, they typically require additional hardware or software to increase the flow table size and/or computing capacity of the SDN data plane for load balancing. As a result, they all increase the network deployment cost and complexity.

- *Methods constrained by limited memory resource:* Some flow-level scheduling algorithms do not consider flow table size constraint on switches, such as Hedera [5]. To deal with the limited flow table size, the controller pre-deploys default paths (such as ECMP) for all flows, and then reroutes some elephant flows for better performance, such as DevoFlow [11], Planck [31], and HS [36]. They all require the traffic statistics information of each individual (or elephant) flow for rerouting and load balancing. However, additional hardware/software needs to be deployed for traffic statistics of individual flows, because those information for the flows through default paths are unavailable only through the flow table. For example, HS [36] mirrors flow traffics to servers for traffic analysis and Opensketch [39] requires additional software modules on switches for traffic measurement.

- *Methods constrained by limited computing resource:* Some methods are proposed to achieve load balancing via flow-level or even subflow-level control. For example, Presto [19] implements the flowcell (fixed-size units) scheduling for load balancing. However since commodity switches have no such computing resource, Presto requires deploying one virtual switch (vswitch) for every physical switch. It is because a vswitch has more powerful processing capacity compared with a physical switch [35]. Moreover, subflow-level control also requires massive flow entries, which conflict with the memory resource constraint.

Many previous load balancing methods require additional hardware or software, which brings a number of weaknesses for network deployment and applications. First, when additional hardware is required, it increases the system cost and limits the applicability. For example, Presto requires to deploy one vswitch for each physical switch, which leads to a higher deployment cost and worse scalability. Second, many current commodity switches (*e.g.*, HP 5460zl [11]) do not support the required software programs, such as flowlet control. Even though the future switches can support these functions, it will introduce huge cost for switch update and replacement, which may not be necessary. We summarize and compare the existing load balancing approaches in Table 1.

This paper tries to answer the following question: *how to perform load balancing without additional device or software on commodity switches, while conquering network asymmetry and traffic dynamics*? Our important observation is that load balancing can be achieved by controlling only a part ($< 50\%$) of the flows in the network and let the other flows follow aggregate paths, such as ECMP paths. In this work we design and implement PrePass for load balancing. PrePass periodically computes and installs wildcard forwarding rules (*i.e.*, proactive routing) for a part of aggregated flows (not all flows) while considering the flow table size constraint and flows' spatial distribution. The remaining flows will be assigned per-flow paths. In other words, the controller will perform the reactive routing control of each newly arrived flow to deal with traffic dynamics. We note that our scheme is different from the previous default path scheme (*e.g.*, DevoFlow [11], and HS [36]), in which all flows will follow default paths and some of them will be rerouted when congestion occurs. PrePass has some significant advantages:

1. PrePass installs wildcard rules for some (not all) flows, which helps to reduce the number of flow entries on each SDN switch. These wildcard rules are determined by the network traffic and the flow table size constraint.

2. As the controller performs reactive routing for other flows, the routing performance (*e.g.*, load balancing or network throughput) will still be efficient even with traffic dynamics, which will be validated by experiments in Section 6.

3. Our method only requires that each SDN switch performs the normal matching-and-forwarding operations, without additional control or computing requirements, which is compatible to commodity switches.

In this paper, we design and implement PrePass for load balancing. Specifically, we define the problem of load balancing with flow table size constraint (LB-FTS), and prove its NP-hardness. We present the detailed algorithm based on the randomized rounding method. The analysis shows that the proposed algorithm can achieve the constant bi-criteria approximation under most practical situations. This paper also discusses some practical issues to enhance the practicability of PrePass. We implement PrePass on our SDN testbed. The experimental results and extensive simulation results show that PrePass can satisfy the different resource constraints on switches, and only increases the link load ratio by about 5%-10% compared with a per-flow routing scheme that uses unlimited data plane resources, under various and dynamic traffic scenarios.

2 Background and Motivation

The goal of this paper is to design a simple and efficient load balancing scheme that conforms the flow table size (FTS) and computing capacity constraints on commodity switches. This section presents the key insights underlying our design.

2.1 Load Balancing with FTS Constraint

Due to the high cost and power consumption of TCAM, commodity switches can only support flow tables with limited size (usually less than 5,000 flow entries on commodity switches [41], *e.g.*, 1,500 on HP 5406zl witch [11]), which becomes the bottleneck of routing performances, especially in large-scale networks. For example, even in a moderate-size network [7], the number of flows may reach 10^6, which is usually far more than the flow table size. Thus, it is impossible to install a rule for each individual flow. One natural way, like RLJD [9], discards some flows so as to satisfy the flow table size constraint, which will reduce the network throughput and experience quality. To accommodate all flows, some wildcard rules are installed to match more flows. For example, we can perform prefix aggregate routing instead of per-flow routing, where flows with the same address prefix will share the same path [11] [31] [36]. Since many flows will match one wildcard flow entry and/or follow a single path, it may result in load imbalance. Moreover, the controller can not acquire the traffic statistics information of each individual flow. Thus, it is difficult to achieve load balancing through flow rerouting.

An SDN switch cannot support all flows with per-flow rules with the following two reasons. (1) TCAMs consume lots of ASIC space and power. Specifically, an OpenFlow rule is described by 10 header fields, which costs total 288 bits [29], while an Ethernet forwarding descriptor is 60 bits, thus OpenFlow entries use more states than Ethernet forwarding entries and are impractical to support large quantities of flow entries due to the limited TCAM size on commodity switches. (2) It takes an unacceptable long time to collect statistics for a large flow table. For example, the statistics-pulling latency for the 5406zl is less than 1 second when the flow table has fewer than 5600 entries [11]. But this can be increased to 2.5 seconds if there are 13K flow table entries [11], which is too long for some practical application link flow scheduling [5].

2.2 Switch Computing Capacity Constraint

The key insight for an SDN is to separate data plane and control plane, and let different parts perform their own functions. The core function of an SDN switch is to forward packets, leaving the control plane in charge of complex decision and control functions. Since most commodity switches run on a low-end CPU, it only has finite computing capacity, which should be mainly responsible for flow entry setup and statistics collection, etc. The testing results in [11] have shown that the switch can complete only 275 flow setups per second even without any traffic load. Due to limited computing capacity, the statistics-pulling interferes with flow setup. When traffic statistics are pulled once a second, collecting counters for just 4500 entries, the controller can only install 150 flow entries per second. In other words, statistics pulling will significantly degrade the entry setup functions (from 275 to 150 flow setups per second). Thus, to make the SDN switch work efficiently, we expect less extra computing overhead on each switch other than necessary operations, *e.g.*, flow entry setup and statistics collection.

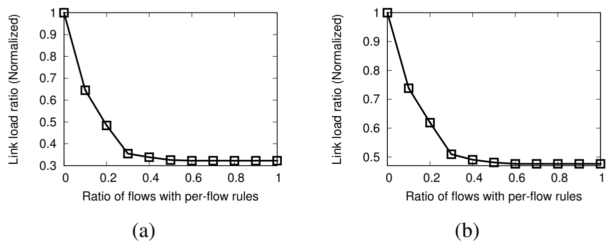


Figure 2: Ratio of flows with per-flow rules vs. Link load ratio (normalized). *Left plot*: Data Center Topology; *right plot*: Campus Topology

2.3 Network Uncertainties

Modern networks are filled with the following two main uncertainties. (1) Asymmetric topologies are common in practical situations due to link failures and heterogeneous switching equipment (*e.g.*, a different number of ports, forwarding speed, *etc.*). For instance, it is reported that data centers may experience frequent link cuts (averagely 40.8 times per day in a production data center [16]). (2) Traffic dynamics also contribute to network uncertainty as well. Previous studies [40] show that traffic in production data centers will be highly dynamic, and the link may be heavy-loaded once a few elephant flows burst.

In the following, we will illustrate how different uncertainties lead to load imbalance. An example of route configuration is shown in Figure 1(a), in which all links have the unique link capacity, 10, and flows f_2 and f_3 share the same link V3→V4. Under this configuration, all links are congestion-free. First, we observe the impact of the link failure (or topology asymmetry). When link V0→V2 fails, flow f_1 should be updated to a new path V0→V3→V4, as shown in Figure 1(b). Otherwise, this flow will be disrupted. Second, traffic dynamics also influence the routes selection for load balancing. For instance, the original route configuration is shown in Figure 1(a). When flow f_3 changes its traffic intensity from 3 to 8, as shown in Figure 1(c), flows f_1 and f_2 have to choose path V0→V2→V4 to avoid link congestion and achieve load balancing.

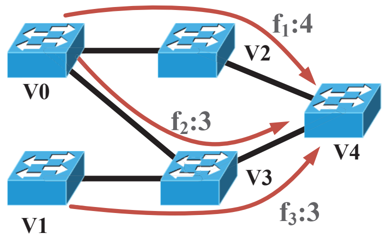
2.4 Let Partial Flows Balance the Load

Recall that all flows with per-flow rules can achieve better network performance and offer fine-grained management, while taking the rick of violating the FTS constraint. Moreover, aggregate (or default) paths can reduce the number of required flow entries, and only provide coarse-grained management. A natural way is to jointly optimize the per-flow routing and aggregate path routing under the constraint of

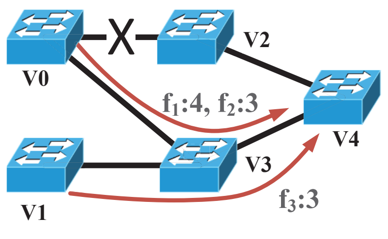
[1]Presto requires extra soft edge(*i.e.*, vSwitch and hypervisor) [19].

Schemes	Handle dynamics?	Satisfy FTS constraint?	Extra hardware/software?	Switch computing overhead[4]
ECMP [17]	No	Yes	No	Low
WCMP [43] DRB [8] RBDP [42]	Partially			
RLJD [9] Hedera [5]	Yes	No	No	Low
DevoFlow [11] DRILL [15] HS [36]	Yes	Yes	Yes	Low
LocalFlow [33] HULA [21] Presto[1] [19] CONGA [6] LetFlow [34]	Yes	Yes	Yes	High
PrePass	**Yes**	**Yes**	**No**	**Low**

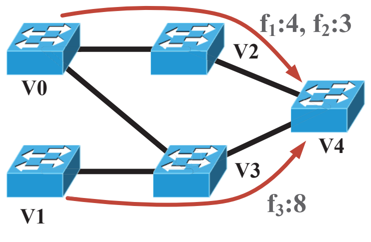
Table 1: Comparison of existing load balancing schemes



(a) Original Route Configuration



(b) link failure happens V0→V2



(c) Dynamic traffic of f_3 from 3 to 8

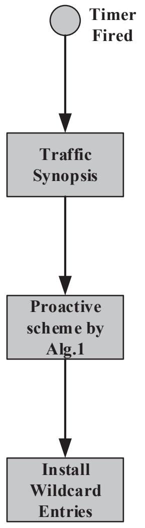
Figure 1: Problems caused by topology asymmetry and traffic dynamics

flow table size. The tradeoff between two schemes can make routing more resilient while satisfying the FTS constraint.

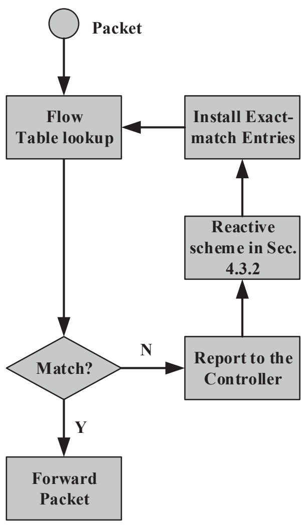
In the following, we will show that, even though the controller only installs per-flow rules for partial flows and let other flows follow the aggregate routes, the network performance will still be preserved. In the simulations, we adopt two topologies, one for campus networks and the other for data center networks, which will be explained in details in Section 6.3.1. We generate 60,000 flows on both topologies (data mining workload in Section 6.3.1), and use the OSPF method to generate the aggregate paths for flows. Figure 2 shows that the link load ratio will be reduced when per-flow rules for more flows are installed. We find that there is no need to install per-flow rules for all flows to achieve better routing performance. Specifically, Figures 2(a) and 2(b) show that the network performance can be almost optimized if the controller only installs per-flow rules for 30%-40% flows on both topologies. These simulation results motivate us to solve the load balancing with flow table size constraint.

2.5 System Workflow of PrePass

Due to a large quantity of flows in modern networks and the limited size of flow tables, it is impractical to install per-flow rules for every flow. Recall that only partial flows with per-flow rules will also help to achieve better load balancing in Section 2.4. Motivated by the simulation results in Figure 2, we study the optimal deployment of wildcard rules and per-



(a) Proactive routing scheme triggered by timer



(b) Reactive routing scheme triggered by new-arrival packets

Figure 3: Illustration of PrePass's workflow

flow rules for load balancing. Thus PrePass adopts two routing schemes. (1) Proactive scheme. The controller *periodically* calculates the aggregated routes for a set of flows (also called macroflows [24] in Definition 1). (2) Reactive scheme. The controller will dynamically determine the per-flow paths for remaining flows to achieve load balancing.

Definition 1 (Macroflow)**.** *A macroflow usually includes mul-*

tiple flows that share a same path and only cost one (wildcard) flow entry on each switch along the path.

A natural idea is to jointly optimize the deployment of these two kinds of rules. However, since the wildcard rules usually serve the aggregated flows with the same address prefix, multiple flows will be affected even if we only update one wildcard rule. For instance, if the controller modifies the forwarding port of a wildcard rule, all flows matching this wildcard rule have to be rerouted, or some flows may be disrupted. Thus the aggregate paths should not be frequently updated. On the contrary, the per-flow rule is fine-grained, reacting to the traffic dynamics in time to meet the different application requirements, *e.g.*, congestion-free and load balancing.

To this end, we separate **PrePass's workflow into two main parts: (1) proactive routing scheme triggered by timer, as shwon in Figure 3(a); and (2) reactive routing scheme triggered by new-arrival flows, as shown in Figure 3(b)**. For the proactive routing scheme, when a period (*e.g.*, 10min) is fired, PrePass first estimates the traffic synopsis based on the long-term traffic statistics [28]. Then, the controller executes Alg. 1 to determine how to install wildcard rules (Secion 4). To be more practical, since the traffic size estimation may not be accurate, we will discuss how to deal with this case in Appendix 5.

Now let's introduce the reactive routing scheme in Figure 3(b). During system running, when a packet arrives at a switch, the switch looks up the flow table. If there is a flow entry matching this packet header, the packet will be processed according to the action field of the flow entry. Otherwise, the switch will report the packet header to the controller for path selection. Accordingly, the controller will determine the route path through dynamic flow routing in Section 4.3.2.

3 Problem Formulation

3.1 Network Model

A software defined network (SDN) discussed in this paper consists of a set of SDN switches, $V = \{v_1, v_2, ..., v_n\}$, with $n = |V|$; and a logically centralized controller. The switches comprising the data plane are responsible for the packet forwarding function. When a flow arrives at a switch, if there is no matched rule for this flow, the controller will determine the route path for this flow in a logically centralized manner. The network topology can be modeled by a graph $G(V,E)$, where E is the set of links connecting two switches in the network.

3.2 Definition of Load-Balancing with Flow Table Size Constraint (LB-FTS)

In an SDN, the flow table size of each switch v_i is denoted by $\beta(v_i)$, and the capacity of each link $e \in E$ is denoted by $c(e)$. We consider a set of macroflows in the network $\Gamma = \{\gamma_1, ..., \gamma_m\}$, with $m = |\Gamma|$, and the traffic amount of a macroflow γ is denoted by $s(\gamma)$. For example, all flows from one ingress switch to another egress switch can be regarded as a macroflow. Since all flows in a macroflow γ match the same wildcard rule on each switch, the controller can gather the traffic statistics information of this macroflow by the Counter field in the flow entry. We also assume that the controller knows the number of flows $|\gamma|$ in the macroflow γ through long-term traffic statistics for two reasons. (1) The controller can compute the average flow size $s(f)$ through statistics collection of per-flow entries, and get the estimated number of flows as $|\gamma| = \frac{s(\gamma)}{s(f)}$. (2) The number of flows in each macroflow can also be predicted by the state-of-the-art prediction techniques. For example, Peng *et al.* [28] shows time-advance and high accuracy in terms of traffic forecasting. Thus, *all required information as the input of LB-FTS can be supported by the current commodity switches, without requiring extra resources on control/data planes*. To be more practical, there may be prediction errors of $|\gamma|$ and $s(\gamma)$, which will be discussed in Section 4.3.3 and Appendix 5, respectively. Note that our simulation results in Section 6 show that our proposed algorithm can better tolerate these estimation errors.

Each macroflow $\gamma \in \Gamma$ can be denoted by $\gamma = \{f_1, ..., f_{|\gamma|}\}$, where the number of flows included in macroflow γ is $|\gamma|$. Moreover, there is a feasible route path set, denoted by P_γ, which is determined based on the management policies and performance objectives (*e.g.*, k shortest paths). For flows in the same macroflow, since they usually have the same source and destination, these flows usually have the same feasible route paths. Without confusion, we denote P_f as the feasible route path set for flow $f \in \gamma$ and P_f is same as P_γ. Some important notations are shown in Table 2.

The LB-FTS problem will choose a subset of macroflows Γ' using proactive routing scheme, and deploy a feasible path for each macroflow in Γ'. We note that the flows will be directly forwarded if there exist matched wildcard rules at switches by proactive scheme. Otherwise, the new-arrival packet will be reported to the controller. We expect to (1) reserve enough flow entries for remaining flows using reactive routing scheme and (2) remain much more bandwidth on each link for load balancing. We formulate the problem as:

$$\min \quad \lambda$$

$$\begin{cases}
\sum\limits_{p \in P_f} y_f^p + z_\gamma = 1 & \forall f \in \gamma, \gamma \in \Gamma \\
\sum\limits_{p \in P_\gamma} y_\gamma^p = z_\gamma & \forall \gamma \in \Gamma \\
\sum\limits_{\substack{f: f \in \gamma, \gamma \in \Gamma \\ p: v \in p, p \in P_f}} y_f^p + \sum\limits_{\substack{\gamma: \gamma \in \Gamma \\ p: v \in p, p \in P_\gamma}} y_\gamma^p \le \beta(v) & \forall v \in V \\
\sum\limits_{\substack{\gamma: \gamma \in \Gamma \\ p: e \in p, p \in P_\gamma}} y_\gamma^p \cdot s(\gamma) \le \lambda c(e) & \forall e \in E \\
y_f^p, z_\gamma, y_\gamma^p \in \{0, 1\} & \forall p \in P_f, f \in \gamma, \gamma \in \Gamma
\end{cases} \tag{1}$$

Note that, variable y_f^p denotes whether the flow $f \in \gamma$ selects the path $p \in P_f$ or not, and y_γ^p denotes whether the macroflow

$\beta(v)$	flow table size of switch v
$c(e)$	the capacity of link e
$c_{\min}$	the minimum capacity of all links
γ	macroflow, including a set of flows
$s(.)$	traffic size of macroflow/flow
$P_.$	feasible path set for a macroflow/flow
n	the number of switches

Table 2: Some Important Notations

γ selects the path $p \in P_\gamma$ as its aggregate path or not. In addition, z_γ represents whether the macroflow γ chooses the proactive scheme or not, and λ is the load balancing factor for the macroflows. The first set of equations means that either macroflow γ chooses the proactive routing scheme or all flows in this macroflow choose the reactive scheme. The second set of equations ensures that there is exact one route path selected as the aggregate path once the macroflow chooses proactive scheme. The third set of inequalities means that the total number of flow entries for both proactive and reactive routing schemes doesn't exceed the flow table size of each switch. The fourth set of inequalities ensures the balance of link load for flows choosing the proactive routing scheme. Our objective is to minimize the load balancing factor (λ), *i.e.*, min λ.

Theorem 1. *The LB-FTS problem is NP-hard.*

Proof. We consider a special case in which each macroflow contains infinite flows, which requires that all flows have to choose the proactive routing scheme. In other words, if the controller deploys per-flow rules for all flows in one macroflow, it may violate the flow table size constraint. Thus all flows in the same macroflow should be regarded as one flow. Under the above circumstance, our LB-FTS problem becomes the unsplittable multi-commodity flow with minimum congestion problem [13], which is NP-hard. Since the special case of the problem is NP-hard, the LB-FTS problem is NP-hard too. □

4 Algorithm Description

Due to the NP-hardness, it is difficult to solve the LB-FTS problem in polynomial time. In this section, we propose a rounding-based algorithm for LB-FTS, and then analyze the approximation performance of the proposed algorithm. After that, we give some discussion to enhance the algorithm.

4.1 Algorithm Description

In this section, we propose a rounding-based wildcard rule configuration algorithm in PrePass for the LB-FTS problem. For each macroflow (or flow), there is a feasible path set. Actually, the number of feasible paths connecting two terminals could be exponential to the network diameter. To achieve the trade-off between algorithm complexity and network performance, same as [9] [36], we only construct some of feasible paths for each flow/macroflow. These paths may be the shortest paths between terminals, which can be found by depth-first search. Since Eq. (1) is an integer linear program, it is difficult to be solved directly. The proposed algorithm includes two steps, relaxing the problem and rounding to the integer solution, respectively. In the first step, we construct a linear program as a relaxation of the LB-FTS problem. Specifically, LB-FTS assumes that the traffic of each flow/macroflow is splittable and can be routed through multiple feasible paths. Since the relaxed version of LB-FTS, denoted by LP_1, is a linear program, we can solve it in polynomial time by a linear program solver. Assume that the optimal solution for LP_1 is $\{\widetilde{y}_f^p, \widetilde{y}_\gamma^p, \widetilde{z}_\gamma\}$, and the optimal result is $\widetilde{\lambda}$. Since LP_1 is the relaxation of the LB-FTS problem, $\widetilde{\lambda}$ is the lower-bound result for LB-FTS.

In the second step, we determine how to deploy aggregate paths for some chosen macroflows. For each macroflow γ, we obtain an integer solution $\widehat{z}_\gamma$ using the randomized rounding method [30]. More specifically, we set $\widehat{z}_\gamma = 1$, which means that an aggregate path will be deployed for macroflow γ, with probability $\widetilde{z}_\gamma$. If $\widehat{z}_\gamma = 0$, this means that all flows in macroflow γ will be routed with reactive scheme. According to the first set of equations in Eq. (1), we have the following equation after the randomized rounding for $\widehat{z}_\gamma$:

$$\sum_{f\in\gamma, p\in P_f} y_f^p = 1 - \widehat{z}_\gamma = \begin{cases} 0, \text{ if } \widehat{z}_\gamma = 1 \\ 1, \text{ if } \widehat{z}_\gamma = 0 \end{cases} \tag{2}$$

According to the second set of equations in Eq. (1), there is a fractional result $\widetilde{y}_\gamma^p$ for each feasible path $p \in P_\gamma$, and the sum of all these fractional results is $\sum\limits_{p\in P_\gamma} y_\gamma^p = \widetilde{z}_\gamma$. Since we will only select one feasible path by randomized rounding, we need to normalize the expectation of these variables. For each feasible path $p \in P_\gamma^p$, we set $\bar{y}_\gamma^p = \frac{\widetilde{y}_\gamma^p}{\widetilde{z}_\gamma}$. Then, we get the integer solution $\widehat{y}_\gamma^p$ through randomized rounding [30]. In particular, $\widehat{y}_\gamma^p$ is set as 1 with probability $\bar{y}_\gamma^p$. The PrePass routing algorithm is formally described in Alg. 1.

Algorithm 1 Rouding-Based Algorithm for LB-FTS

```
1:  Step 1: Solving the relaxed LB-FTS problem
2:  Construct a linear program $LP_1$ based on Eq. (1)
3:  Obtain the optimal solution $\widetilde{y}_f^p$, $\widetilde{y}_\gamma^p$ and $\widetilde{z}_\gamma$
4:  Step 2: Aggregate routing for selected macroflows
5:  Derive an integer solution $\widehat{z}_\gamma$ by randomized rounding
6:  for each macroflow $\gamma \in \Gamma$ do
7:      if $\widehat{z}_\gamma$ = 1 then
8:          for each feasible path $p \in P_\gamma$ do
9:              Compute $\bar{y}_\gamma^p = \frac{\widetilde{y}_\gamma^p}{\widetilde{z}_\gamma}$
            Derive an integer solution $\widehat{y}_\gamma^p$ by randomized rounding
10:         for each feasible path $p \in P_\gamma$ do
11:             if $\widehat{y}_\gamma^p$ = 1 then
12:                 Install wildcard entries on switches along $p$
```

4.2 Performance Analysis

Before analyzing the approximation performance of PrePass, we give the following lemma according to the rounding operations.

Lemma 2. *The probability that each macroflow γ selects a feasible path $p \in P_\gamma$ as its aggregate path is the same as the solution of the linear program LP_1.*

Proof. By the algorithm description, macroflow γ chooses the proactive routing scheme with probability $\widetilde{z}_\gamma$, which is the result of the linear program LP_1. Furthermore, for each feasible path $p \in P_\gamma$, it is selected as the aggregate path with probability $\frac{\widetilde{y}_\gamma^p}{\widetilde{z}_\gamma}$ by the Algorithm 1. Combining with the two steps, we conclude that the feasible path p is selected as the aggregate path with probability $\widetilde{z}_\gamma \cdot \frac{\widetilde{y}_\gamma^p}{\widetilde{z}_\gamma} = \widetilde{y}_\gamma^p$, which is same as the solution of linear program LP_1. □

In the following, we analyze the approximation performance of PrePass. Assume that the minimum capacity of all the links is denoted by $c_{\min}$. We define a variable α as follow:

$$\alpha = \min\{\beta(v), v \in V; \frac{\widetilde{\lambda} \cdot c_{\min}}{s(\gamma)}, \gamma \in \Gamma\} \quad (3)$$

Under most practical situations, since the flow intensity is usually much less than the link capacity, it follows that $\alpha \gg 1$. As PrePass is a randomized algorithm, we analyze the expected data plane resource cost. We give two famous lemmas for probability analysis.

Lemma 3 (Chernoff Bound)**.** *Given n independent variables: $x_1, x_2, ..., x_n$, where $\forall x_i \in [0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^n x_i]$. Then, $\mathbf{Pr}\left[\sum_{i=1}^n x_i \geq (1+\varepsilon)\mu\right] \leq e^{\frac{-\varepsilon^2\mu}{2+\varepsilon}}$, where ε is an arbitrary positive value.*

Lemma 4 (Union Bound)**.** *Given a countable set of n events: $A_1, A_2, ..., A_n$, each event A_i happens with possibility $\mathbf{Pr}(A_i)$. Then, $\mathbf{Pr}(A_1 \cup A_2 \cup ... \cup A_n) \leq \sum_{i=1}^n \mathbf{Pr}(A_i)$.*

Link Capacity Performance. We give the following lemma to show the link capacity performance of out algorithm.

Lemma 5. *After the rounding process, the load of each link will not exceed the constraint $\widetilde{\lambda} \cdot c(e)$ by a factor of $\varepsilon + 1 = \frac{4\log n}{\alpha} + 3$.*

Before analyzing the link resource performance, we define a random variable x_γ^e to denote the traffic amount on link e from macroflow γ.

Definition 2. *For each macroflow γ and each link e, a random variable x_γ^e is defined as:*

$$x_\gamma^e = \begin{cases} s(\gamma), & \text{with probability} \quad \sum_{p:e\in p, p\in P_\gamma} \widetilde{y}_\gamma^p \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

According to the definition, $x_{\gamma_1}^e$, $x_{\gamma_2}^e$... are mutually independent. The expected link load on link e is:

$$\mathbb{E}\left[\sum_{\gamma\in\Gamma} x_\gamma^e\right] = \sum_{\gamma\in\Gamma} \mathbb{E}[x_\gamma^e] = \sum_{\gamma\in\Gamma} \sum_{p:e\in p, p\in P_\gamma} \widetilde{y}_\gamma^p \cdot s(\gamma) \leq \widetilde{\lambda} \cdot c(e) \quad (5)$$

Combining Eq. (5) and the definition of α in Eq. (3), we have

$$\begin{cases} \frac{x_\gamma^e \cdot \alpha}{\widetilde{\lambda} \cdot c(e)} \in [0,1] \\ \mathbb{E}\left[\sum_{\gamma\in\Gamma} \frac{x_\gamma^e \cdot \alpha}{\widetilde{\lambda} \cdot c(e)}\right] \leq \alpha. \end{cases} \quad (6)$$

Then, by applying Lemma 3, assume that ε is an arbitrary positive value. It follows

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{x_\gamma^e \cdot \alpha}{\widetilde{\lambda} \cdot c(e)} \geq (1+\varepsilon)\alpha\right] \leq e^{\frac{-\varepsilon^2\alpha}{2+\varepsilon}} \quad (7)$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{x_\gamma^e \cdot \alpha}{\widetilde{\lambda} \cdot c(e)} \geq (1+\varepsilon)\alpha\right] \leq e^{\frac{-\varepsilon^2\alpha}{2+\varepsilon}} \leq \frac{F}{n^2} \quad (8)$$

where F is the function of network-related variables (such as the number of switches n, *etc.*) and $F \to 0$ when the network size grows.

The solution for Eq. (8) is expressed as:

$$\varepsilon \geq \frac{\log\frac{n^2}{F} + \sqrt{\log^2\frac{n^2}{F} + 8\alpha\log\frac{n^2}{F}}}{2\alpha}, \quad n \geq 2 \quad (9)$$

Lemma 6. *After the rounding process, the load of each link will not exceed the constraint $\widetilde{\lambda} \cdot c(e)$ by a factor of $\varepsilon + 1 = \frac{4\log n}{\alpha} + 3$.*

Proof. Set $F = \frac{1}{n^2}$. Eq. (8) is transformed into:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{x_\gamma^e \cdot \alpha}{\widetilde{\lambda} \cdot c(e)} \geq (1+\varepsilon)\alpha\right] \leq \frac{1}{n^4}, \text{where } \varepsilon \geq \frac{4\log N}{\alpha} + 2$$

By applying Lemma 4, we have,

$$\begin{aligned} &\mathbf{Pr}\left[\bigvee_{e\in E} \sum_{\gamma\in\Gamma} \frac{x_\gamma^e \cdot \alpha}{\widetilde{\lambda} \cdot c(e)} \geq (1+\varepsilon)\alpha\right] \\ \leq &\sum_{e\in E} \mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{x_\gamma^e \cdot \alpha}{\widetilde{\lambda} \cdot c(e)} \geq (1+\varepsilon)\alpha\right] \\ \leq &n^2 \cdot \frac{1}{n^4} = \frac{1}{n^2}, \quad \varepsilon \geq \frac{4\log n}{\alpha} + 2 \end{aligned} \quad (10)$$

Note that the third inequality holds, because there are at most n^2 links in a network. The approximation factor of our algorithm is $\varepsilon + 1 = \frac{4\log n}{\alpha} + 3$. □

Flow Table Resource Performance. We give the following lemma to show the flow table resource performance.

Lemma 7. *After the rounding process, the number of flow entries on each switch will not exceed the constraint $\beta(v)$ by a factor of $\rho + 2 = \frac{3\log n}{\alpha} + 4$.*

Before analyzing the flow table resource performance, we define a random variable w_γ^v to denote whether there deploys a flow entry for macroflow γ on switch v or not.

Definition 3. *For each macroflow γ and each switch v, a random variable w_γ^v is defined as:*

$$w_\gamma^v = \begin{cases} 1, & \textit{with probability} \sum_{p:v\in p, p\in P_\gamma} \widetilde{y}_\gamma^p \\ 0, & \textit{otherwise.} \end{cases} \tag{11}$$

According to the definition, $w_{\gamma_1}^v$, $w_{\gamma_2}^v$... are mutually independent. The expected number of occupied flow entries on switch v is:

$$\begin{aligned}\mathbb{E}\left[\sum_{\gamma\in\Gamma} w_\gamma^v\right] = \sum_{\gamma\in\Gamma}\mathbb{E}[w_\gamma^v] &= \sum_{\gamma\in\Gamma}\sum_{p:v\in p,p\in P_\gamma}\widetilde{y}_\gamma^p \\ &\le \beta(v) - \sum_{\gamma\in\Gamma}\sum_{f:f\in\gamma;p:p\in P_f,v\in p}\widetilde{y}_f^p \\ &\le \beta(v)\end{aligned} \tag{12}$$

Combining Eq. (12) and the definition of α in Eq. (3), we have

$$\begin{cases}\frac{w_\gamma^v\cdot\alpha}{\beta(v)} \in [0,1] \\ \mathbb{E}\left[\sum_{\gamma\in\Gamma}\frac{w_\gamma^v\cdot\alpha}{\beta(v)}\right] \le \alpha.\end{cases} \tag{13}$$

Then, by applying Lemma 3, assume that ρ is an arbitrary positive value. It follows

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{w_\gamma^v\cdot\alpha}{\widetilde{\lambda}\cdot c(e)} \ge (1+\rho)\alpha\right] \le e^{\frac{-\rho^2\alpha}{2+\rho}} \tag{14}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{w_\gamma^v\cdot\alpha}{\beta(v)} \ge (1+\rho)\alpha\right] \le e^{\frac{-\rho^2\alpha}{2+\rho}} \le \frac{F}{n^2} \tag{15}$$

where F is a variable defined in the above analysis.

The solution for Eq. (15) is expressed as:

$$\rho \ge \frac{\log\frac{n^2}{F} + \sqrt{\log^2\frac{n^2}{F} + 8\alpha\log\frac{n^2}{F}}}{2\alpha}, \quad n\ge 2 \tag{16}$$

Lemma 8. *After the rounding process, the number of flow entries on each switch will not exceed the constraint $\beta(v)$ by a factor of $\rho + 2 = \frac{3\log n}{\alpha} + 4$..*

Proof. Set $F = \frac{1}{n}$. Eq. (15) is transformed into:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{w_\gamma^v\cdot\alpha}{\beta(v)} \ge (1+\rho)\alpha\right] \le \frac{1}{n^3}, \text{where } \rho \ge \frac{3\log n}{\alpha} + 2$$

By applying Lemma 4, we have,

$$\begin{aligned}&\mathbf{Pr}\left[\bigvee_{v\in V}\sum_{\gamma\in\Gamma}\frac{w_\gamma^v\cdot\alpha}{\beta(v)} \ge (1+\rho)\alpha\right] \\ =&\mathbf{Pr}\left[\bigvee_{v\in V}\sum_{\gamma\in\Gamma}\frac{w_\gamma^v}{\beta(v)} \ge (1+\rho)\right] \\ \le&\sum_{v\in V}\mathbf{Pr}\left[\sum_{\gamma\in\Gamma}\frac{w_\gamma^v\cdot\alpha}{\beta(v)} \ge (1+\rho)\alpha\right] \\ \le& n\cdot\frac{1}{n^3} = \frac{1}{n^2}, \quad \rho \ge \frac{3\log n}{\alpha} + 2\end{aligned} \tag{17}$$

Note that the third inequality holds, because there are at most n switches in a network. After considering the occupied flow entries of flows with per-flow rules, we have,

$$\begin{aligned}&\mathbf{Pr}\left[\bigvee_{v\in V}[\sum_{\gamma\in\Gamma}\frac{w_\gamma^v}{\beta(v)} + \sum_{\gamma\in\Gamma}\sum_{f:f\in\gamma;p:p\in P_f,v\in p}\frac{\widetilde{y}_f^p}{\beta(v)}] \ge (2+\rho)\right] \\ \le& \frac{1}{n^2}, \quad \rho \ge \frac{3\log n}{\alpha} + 2\end{aligned} \tag{18}$$

The approximation factor of our algorithm is $\rho + 2 = \frac{3\log n}{\alpha} + 4$. $\square$

Approximation Factors. By above analysis, we can conclude that:

Lemma 9. *The traffic load will hardly be violated by a factor of $\frac{4\log n}{\alpha} + 3$, and the flow table constraint will not be violated by a factor of $\frac{3\log n}{\alpha} + 4$.*

In most practical situations, PrePass can reach almost the constant bi-criteria approximation. For example, let $\widetilde{\lambda}$ and n be 0.4 and 1000. Observing the practical flow traces, the maximum intensity of a macroflow may reach 100Mbps. In today's networks, the link capacity can be 10Gbps [4]. The flow table size is usually 4000. Under these circumstances, $\alpha = \frac{\widetilde{\lambda}\cdot c_{\min}}{s(\gamma)}$ will be 40. Thus, the approximation factors for the link capacity and the flow table constraint are 3.7 and 4.5, respectively. In other words, PrePass can achieve the constant bi-criteria approximation for the LB-FTS problem under many practical situations.

4.3 Discussion

This section provides some discussion to improve the practicability of PrePass.

4.3.1 Port Statistics Collection

During system running, the controller should master the real-time traffic load on each link to better deal with traffic dynamics. The openflow standard specifies the OFPT_PORT_STATUS interface for port traffic statistics collection [23]. Since each link connects with two ports on different switches, we can collect port traffic statistics only from a subset of switches to reduce the controller overhead. To this end, we use the minimum set cover algorithm [14] for port traffic statistics collection. Specifically, the controller determines from which switches the port statistics information will be collected and sends a set of the OFPT_PORT_STATUS requests to specified switches. After receiving these requests, the switch will report the statistics information of all ports to the controller. As a result, the controller knows the accurate traffic information of all ports (links). Since the number of ports on each switch is usually small (*e.g.*, 24 or 48), the overhead (or delay) for port statistics collection is very low (or small). For better trade-off between the statistics accuracy and collection overhead, we trigger the port statistics collection with a fixed period, *e.g.* 2s.

4.3.2 Dynamic Flow Routing

When a new flow arrives at an SDN switch, this switch reports the packet header to the controller, which will dynamically determine its route path. In this paper, we introduce a simple and efficient routing mechanism, in which the controller chooses the least-congestion route path with flow table size constraint for each new-arrival flow. As described in Section 4.3.1, the controller has known the link traffic load (or the link load ratio) by collecting the port statistics information at the beginning of each period. Since the flow table of each switch is updated by the controller, the controller can derive the number of occupied flow entries on a switch and determines whether this switch can accommodate this flow or not. For each path p, the congestion of this path is the maximum load ratio of all links on this path. The controller adopts the Dijkstra algorithm [27] to explore the route path with the least congestion for this flow, and installs rules on switches along this path. Under this situation, the controller can immediately determine the route path for new-arrival flows, which helps to deal with the dynamics of the traffic.

4.3.3 Dealing with Flow Estimation Deviation

When a flow f arrives at an SDN switch, if there is no matched flow entry or the flow table is saturated, the controller can not install a flow entry on this switch for flow f. Due to flow estimation deviation, the total number of required flow entries may exceed the flow table size. When it happens, we will aggregate some arrived flows with reactive routing scheme into one flow entry using the wildcard rule. Given a flow f, we determine its corresponding macroflow, denoted by γ, choose a least-congestion path for this macroflow, and update the flow tables on different switches. First, we add a wildcard rule on each switch along this aggregate path. Then, we will remove all flow entries for flows in this macroflow on different switches. As a result, the number of required flow entries on switches will be reduced.

4.3.4 Maximizing the Number of Controllable Flows

The previous section studies the efficient deployment of aggregate paths for macroflows based on the long-term traffic statistics. To make our problem more robust and generalized, this section studies the case in which the controller does not know the traffic size of each flow. Under this case, we expect to adjust/control as many individual flows as possible, which contribute to load balancing. For ease of expression, we denote PrePassE as the algorithm for this extended case. Due to space limit, the problem definition and algorithm description are relegated to Appendix 5.

5 Algorithm Extension

The previous section studies the efficient deployment of aggregate paths for macroflows based on the long-term traffic statistics. To make our problem more robust and generalized, this section studies the case in which the controller does not know the traffic size of each flow. Under this case, we expect to adjust/control as many individual flows as possible, which contribute to load balancing.

5.1 Definition of Load-Balancing with Maximum Adjustable Flows (LB-MDF)

Compared with the proactive routing scheme, the reactive routing scheme (with per-flow rules) can provide fine-grained control for each flow. In the following, if one flow is forwarded through the reactive scheme, we call this as an adjustable flow. To achieve better load balancing, we expect to remain as many adjustable flows as possible while taking the limited flow table size into account. We formulate the load balancing with maximum adjustable flows (LB-MDF) problem as:

$$\max \sum_{f\in\gamma,\gamma\in\Gamma,p\in P_f} y_f^p$$

$$\begin{cases}\sum_{p\in P_f} y_f^p + z_\gamma = 1 & \forall f\in\gamma,\gamma\in\Gamma\\ \sum_{p\in P_\gamma} y_\gamma^p = z_\gamma & \forall\gamma\in\Gamma\\ \sum_{\substack{f:f\in\gamma,\gamma\in\Gamma\\ p:v\in p,p\in P_f}} y_f^p + \sum_{\substack{\gamma:\gamma\in\Gamma\\ p:v\in p,p\in P_\gamma}} y_\gamma^p \le \beta(v) & \forall v\in V\\ y_f^p, z_\gamma, y_\gamma^p\in\{0,1\} & \forall p\in P_f, f\in\gamma,\gamma\in\Gamma\end{cases} \tag{19}$$

The first set of equations means that either macroflow γ chooses the proactive routing scheme or all flows in this macroflow choose the reactive scheme. The second set of equations ensures there is exact one route path selected as the default path once the macroflow chooses the proactive scheme. The third set of inequalities means that the total number of flow entries does not exceed the flow table size constraint on each SDN switch.

Theorem 10. *The LB-MDF problem is NP-hard.*

The proof has been relegated to A.

5.2 Algorithm Description

In this section, we propose a rounding-based wildcard rule configuration algorithm called PrePassE for the LB-MDF problem. Since Eq. (19) is an integer linear program, it is difficult to solve it directly, thus we first relax the integer linear program by replace the fourth constraints with $y_f^p, z_\gamma, y_\gamma^p \in [0,1]$. That is, each flow is splittable and can be forwarded to several paths. After the relaxation, Eq. (19) becomes a linear program, and we can solve it in polynomial time by a linear program solver. Note that $\{\widetilde{y}_f^p, \widetilde{y}_\gamma^p, \widetilde{z}_\gamma\}$ is the optimal solution for the linear program. Since the linear program is the relaxation of the LB-MDF problem, the optimal result $\widetilde{\theta} = \sum_{f\in\gamma,\gamma\in\Gamma,p\in P_f} \widetilde{y}_f^p$ is the upper-bound result for LB-MDF.

In the second step, we determine how to deploy an aggregate path for each macroflow. We obtain an integer solution $\widehat{z}_\gamma$ using the rounding method [30]. More specifically, we set $\widehat{z}_\gamma$ = 1, which means that an aggregate path is deployed for macroflow γ, with the probability $\widetilde{z}_\gamma$. If $\widehat{z}_\gamma$ = 0, this means that all the flows in macroflow γ will be routed with per-flow rules. According to the first set of equations in Eq. (19), we have the following equation after randomized rounding:

$$\sum_{f\in\gamma,p\in P_f} y_f^p = \begin{cases} 0, \text{ if } \widehat{z}_\gamma = 1 \\ 1, \text{ if } \widehat{z}_\gamma = 0 \end{cases} \tag{20}$$

According to the second set of equations in Eq. (19), there is a fractional result $\widetilde{y}_\gamma^p$ for each feasible path $p \in P_\gamma$, and the sum of all these fractional results is: $\sum_{p\in P_\gamma} y_\gamma^p = \widetilde{z}_\gamma$. Since we need to select one path by randomized rounding, it is required to normalize the expectation of these variables. For each feasible path $p \in P_\gamma^p$, $\bar{y}_\gamma^p = \frac{\widetilde{y}_\gamma^p}{\widetilde{z}_\gamma}$. Then, we get the integer solution $\widehat{y}_\gamma^p$ through randomized rounding [30]. In particular, $\widehat{y}_\gamma^p$ is set as 1 with the probability $\bar{y}_\gamma^p$. PrePassE is formally described in Alg. 2.

Algorithm 2 Rouding-Based Algorithm for LB-MDF

```
1: Step 1: Solving the relaxed LB-MDF problem
2: Construct a relaxed linear program based on Eq. (19)
3: Obtain the optimal solution $\widetilde{y}_f^p$, $\widetilde{y}_\gamma^p$ and $\widetilde{z}_\gamma$
4: Step 2: Proactive routing for selected macroflows
5: Derive an integer solution $\widehat{z}_\gamma$ by randomized rounding
6: for each macroflow $\gamma \in \Gamma$ do
7:     if $\widehat{z}_\gamma$ = 1 then
8:         for each feasible path $p \in P_\gamma$ do
9:             Compute $\bar{y}_\gamma^p = \frac{\widetilde{y}_\gamma^p}{\widetilde{z}_\gamma}$
10:        Derive an integer solution $\widehat{y}_\gamma^p$ by randomized rounding
11:        for each feasible path $p \in P_\gamma$ do
12:            if $\widehat{y}_\gamma^p$ = 1 then
13:                Install wildcard entries on switches along $p$
```

5.3 Performance Analysis

In this section, we analyze the approximation performance of PrePassE in terms of the number of adjustable flows and the flow table resource. We first define a variable η as follows:

$$\eta = \min\{\beta(v), v \in V; \widetilde{\theta}/|\gamma|, \gamma \in \Gamma\} \tag{21}$$

Under most practical situations, it follows that $\eta \gg 1$. As PrePassE is a randomized algorithm, we analyze the expected data plane resource cost.

Lemma 11. *The proposed PrePassE algorithm can guarantee that the total number of adjustable flows will not be less than the upper bound of LB-MDF by a factor of* $(1 - 2\sqrt{\frac{\log n}{\eta}})$*,*

The proof of Lemma is relegated to Appendix B. In most practical situations, the approximation factor is constant. In a network with n switches, the number of macroflows is $n(n-1)$. We assume that only 30% of the flows are adjustable (with per-flow rules) and the number of flows in each macroflow obeys poisson distribution. Note that the expected number of flows in macroflow γ is $|\bar{\gamma}|$, and we have $|\gamma| \sim P(|\bar{\gamma}|)$. According to poisson distribution and mathematical computation, $Pr(|\gamma| > 4|\bar{\gamma}|) < 0.0012$, thus we have $\max\{|\gamma|, \gamma \in \Gamma\} \le 4|\bar{\gamma}|$, and η is $0.3 \cdot \frac{1}{4} \cdot n(n-1)$. For example, the number of switches is usually more than 100, and the number of flow entries on each switches is about 4000 [11]. Thus $\eta = \min\{\beta(v), v \in V; \widetilde{\theta}/|\gamma|, \gamma \in \Gamma\}$=750, and the approximation factor is $(1 - 2\sqrt{\frac{\log n}{\eta}})$ = 0.84. For a large-scale network with 1000 switches, the approximation factor is 0.92. In summary, PrePassE can achieve constant approximation performance in terms of the number of adjustable flows compared with the upper bound of LB-MDF.

Lemma 12. *The proposed PrePassE algorithm can achieve the approximation factor of* $\frac{3\log n}{\eta} + 4$ *for the flow table size constraint.*

The proof of Lemma 12 is similar to that of Lemma 8. Due to space limit, we omit the proof here, which can be found in Appendix C. Also, the approximation factor is constant in most practical situations, and the analysis is similar to that in Section 4.2.

6 Performance Evaluation

To demonstrate the feasibility and efficiency of the proposed algorithms, we conduct the experiment and simulation-based evaluation.

6.1 Performance Metrics and Benchmarks

We have designed PrePass for proactive and reactive routing sch-emes, respectively. For ease of description, we denote PrePass as the integrated two algorithms. We evaluate PrePass through testbed implementation and large-scale network simulations. We adopt four metrics for performance evaluation. The first metric is the number of adjustable flows (NAF), which refers to the number of flows that can be adjusted/controlled by the controller through per-flow rules. The larger NAF means that we can control/adjust more individual flows, which are of benefit to load balancing in general. Since the packet header of each adjustable flow will be reported to the controller, the number of adjustable flows greatly reflects the controller overhead. Since this paper studies the load balancing under strict flow table size constraint by deploying aggregate paths for partial macroflows, the second metric is link load ratio (LLR). LLR is defined as $\max\{f(e)/c(e), e \in E\}$, where $f(e)$ is the traffic load of link e. The smaller LLR means better load balancing effect. Another important data-plane resource is flow entries. We adopt the cumulative distribution function (CDF) of occupied flow entries on all switches as the third metric. When the number of required flow entries exceeds the flow table size, or the link is congested, flows may be dropped. Thus we adopt throughput as the final metric, which is defined as the total traffic amount of flows that are successfully forwarded in the network with flow table size constraint.

To evaluate the performance of our proposed algorithms, we adopt four benchmarks listed as follows.

1. OSPF [25]. Each switch constructs the shortest path to each destination. That is, each switch will install a flow entry for each switch.

2. ECMP [17]. Each switch constructs multiple (*e.g.*, 3 in our evaluaion) equal-cost paths to each another switch. We should note that the ECMP method requires group entries when there are multiple equal-cost paths to the destination in an SDN. We will compare it with our algorithms in terms of the link load ratio and throughput.

3. RLJD [9]. In RLJD, flows are forwarded with per-flow rules. Since RLJD discards flows if there are not enough flow entries, we will compare it with our algorithm in terms of network throughput. Moreover, we use RLJD-M to represent that the flow table size is infinite. That is, each flow will be routed with per-flow rules. We compare RLJDM with our proposed algorithm by evaluating the link load ratio and CDF of occupied flow entries.

4. Presto [19], which applies OVS [2] on the edge switches to enhance the flow table and the processing capacity of edge switches. Though Presto can provide flowcell scheduling in a network, we assume that the controller schedules each flow (not flowcell) on its ingress switch for fair performance comparison.

6.2 System Implementation and Testing

6.2.1 Implementation on the Platform

We implement PrePass, OSPF, RLJD, Presto and ECMP on a small-scale testbed. Our testbed is mainly composed of three parts: a controller, eight virtual switches and seven virtual

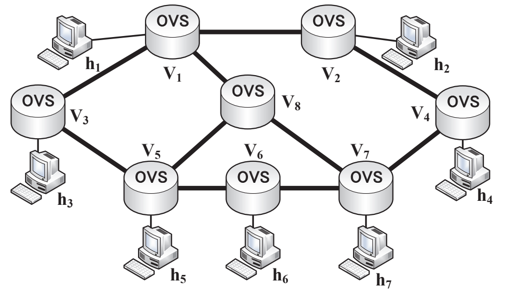


Figure 4: Topology of the SDN platform. We implement our proposed algorithm and other four benchmarks on a real testbed. Our testbed is mainly composed of three parts: a controller, eight virtual switches and seven virtual machines.

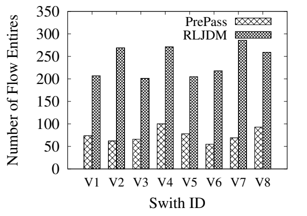


Figure 5: Number of Occupied Flow Entries on Switches

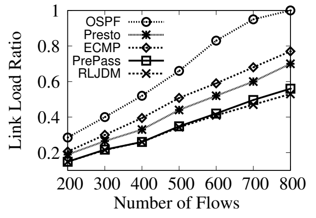


Figure 6: Number of Flows vs. Link Load Ratio (LLR)

No. of flows	200	300	400	500	600	700	800
NAF	200	226	213	203	196	195	191

Table 3: Number of adjustable flows (NAF) by PrePass

machines (acting as hosts). Each virtual switch is implemented using the open virtual switch (OVS, version 2.4.0) [2], which supports OpenFlow v1.3 standard [29]. Each virtual switch and the connected Kernel-based Virtual Machines (KVM) are implemented on a server with a core i5-3470 processor and 8GB of RAM. Since the controller will not participate in data forwarding, it is not explicitly shown in Figure 4 for simplicity. We use Ryu 4.17 [3] as the controller software running on a server with a core i5-3470 processor and 16GB of RAM.

In our implementation, we generate different quantities of flows on a small-scale topology. Specifically, each virtual machine (or host) is installed with a traffic generator, and requests flows according to a Poisson processing from randomly chosen hosts with different ports. Moreover, the flow size is drawn from a large cluster running *data mining* jobs [17]. To estimate the fabric's load balancing performance, we expect flows to traverse the fabric. Since the traffic between two hosts attached with the same switch doesn't traverse the fabric, it has no impact on the fabric's load. To this end, we deploy one host for each switch and use source IP, destination IP and source port to identify each flow. We also identify a marcoflow with the same source IP and destination IP. Then, each host is able to generate different numbers of flows in a network. All the links have the same link bandwidth, 10Gbps.

6.2.2 Experimental Results

We run three sets of experiments on the SDN platform. In each experiment, we set the flow table size on each switch is 100. In the first testing, we generate 600 flows in the network, and observe the number of required flow entries on all switches. The testing results in Figure 5 show that the maximum number of occupied flow entries is 100 and 286 for PrePass and RLJDM, respectively. In addition, PrePass satisfies the flow table size constraint on each switch. That's because PrePass deploys aggregate paths for partial macroflows with proactive scheme to reduce the occupied flow entries. Specifically, PrePass can averagely reduce the number of occupied flow entries by $\frac{239-75}{239}$=68% compared with RLJDM. In the second testing, we observe the number of adjustable flows and link load ratio by changing the number of flows in an SDN. The results in Table 3 show that (1) when the number of flows increases (to 300 flows), the number of adjustable flows increases too; and (2) when the number of flows increases (more than 300 flows), the number of adjustable flows decreases. That's because more macroflows will be forwarded through aggregate paths with more and more flows in a network. We observe link load ratio (LLR) by changing the number of flows from 200 to 800. As shown in Figure 6, LLR of PrePass is close to that of RLJDM (within 5%), while PrePass significantly reduces the link load ratio by 21%, 26% and 44% compared with Presto, ECMP, and OSPF, respectively. That's because PrePass deploys aggregate paths with the objective to minimize LLR of aggregate paths and dynamic route flows with per-flow rules for other flows.

6.3 Simulations

6.3.1 Simulation Setting

We select two practical topologies. The first topology is the fat-tree [4], denoted by (a), containing 80 switches and 128 servers. It has been widely applied in many data centers. The second topology, denoted by (b), is a campus network containing 100 switches and 200 servers from [1]. For both topologies, each link has a uniform capacity, 5Gbps. We conduct extensive simulations with realistic workloads based on the empirical traffic patterns in practical networks. Specifically, we adopt two traffic workloads. The first distribution is derived from traffic traces from a practical data center [6] and represents a large *enterprise workload*. The second distribution is from a large cluster running *data mining* jobs [17]. Both workloads are heavy-tailed: a small fraction of flows contributes most of the traffic amount. In particular, the data mining distribution has a very heavy tail with 95% of the traffic amount from about 3.6% of elephant flows larger than 20MB, while 20% of the flows account for 80% of the traffic for enterprise workloads. Same as [40], we generate flows between random senders and receivers with varying traffic loads. The flow table size is different for various switches. By default, the flow table size on each switch is set to a moderate value (*e.g.*, 4000 [11]) in our simulations.

6.3.2 Routing Performance and Occupied Flow Entries

We first observe the number of adjustable flows (NAF) by changing the number of flows in a network. Figures 7 and 8

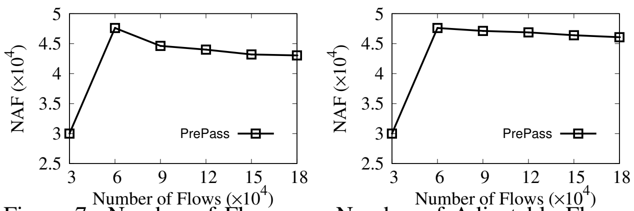


Figure 7: Number of Flows vs. Number of Adjustable Flows in Topology (a). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

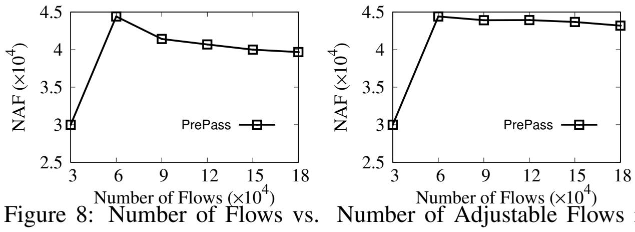


Figure 8: Number of Flows vs. Number of Adjustable Flows in Topology (b). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

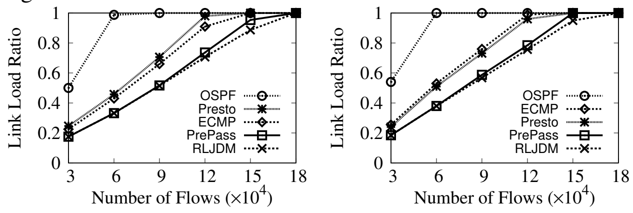


Figure 9: Number of Flows vs. Link Load Ratio (LLR) in Topology (a). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

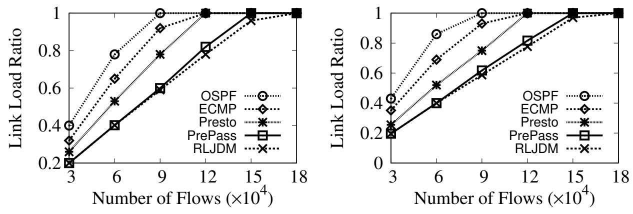


Figure 10: Number of Flows vs. Link Load Ratio (LLR) in Topology (b). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

show that NAF of PrePass increases as the number of flows increases. But when the number of flows increases to 6×10^4, the number of adjustable flows is stable in a certain interval. For example, Figure 7(a) shows that NAF increases from 3×10^4 to 4.7×10^4 when the number of flows increases from 3×10^4 to 6×10^4. After then, NAF remains in the interval $[4.4\times10^4, 4.7\times10^4]$. The reason is as follow. When there are less flows, the flow table is enough to accommodate all flows with per-flow rules. However, when the number of flows keeps increasing, due to the limited flow table size, more and more flows will pass through the aggregate paths. As a result, NAF is stable in a certain interval. From these two figures, we also conclude that the controller overhead keeps stable with more and more flows in a network.

The second set of simulations studies the link load ratio performance when the number of flows increases. We compare PrePass with OSPF, RLJDM, Presto and ECMP under different numbers of flows ranging from 3×10^4 to 18×10^4. The link load ratio (LLR) results on topology (a) are shown in Figure 9. We observe that LLR increases with more and

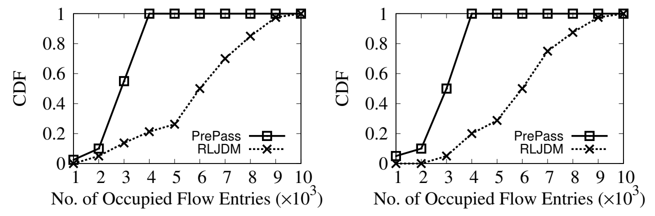


Figure 11: Number of Occupied Flow Entries vs. CDF in Topology (a). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

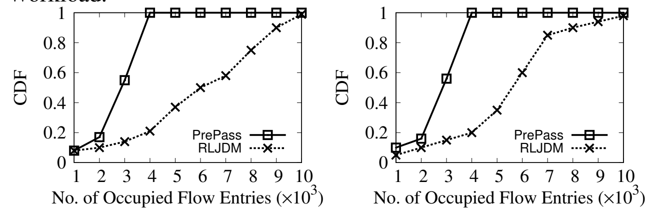


Figure 12: Number of Occupied Flow Entries vs. CDF in Topology (b). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

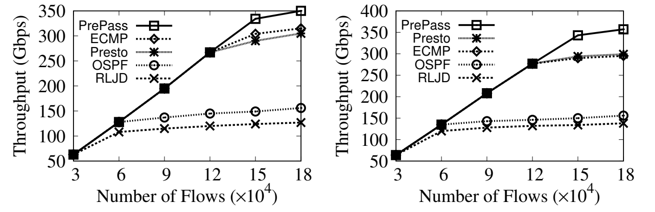


Figure 13: Number of Flows vs. Throughput in Topology (a). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

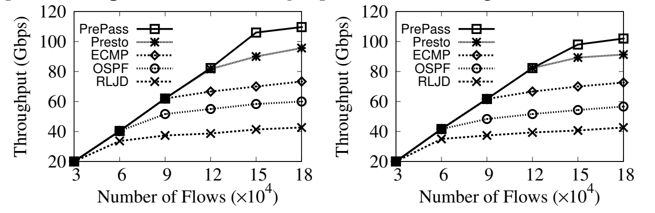


Figure 14: Number of Flows vs. Throughput in Topology (b). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

more flows for all five algorithms. In comparison, ECMP can reduce LLR by 10% compared with Presto for the enterprise workload, while for the data mining workload, Presto reduces LLR by about 6% compared with ECMP. PrePass can achieve almost the same LLR as RLJDM, especially when the number of flows is less than 9×10^4. For example, when the number of flows is 9×10^4 with the enterprise workload by Figure 9, OSPF occurs congestion (or LLR is 1), while the link load ratios of PrePass, RLJDM, Presto and ECMP are 0.52, 0.52, 0.72 and 0.66, respectively. As a result, PrePass can reduce LLR by 27.7% and 21.2% compared with Presto and ECMP, respectively. However, simulation results on topology (b) are quite different. ECMP has higher LLR than Presto, RLJD and PrePass. For example, when the number of flows is 6×10^4 with the data mining workload, LLRs of ECMP, Presto, RLJD and PrePass are 0.69, 0.53, 0.39 and 0.39, respectively. That accords with previous researches [19], which have shown that ECMP performs less efficiently on asymmetric topology like topology (b). For both topologies, PrePass achieves close LL-R (within 5%) to RLJDM and performs better than Presto and ECMP, even when the number of flows is large (*e.g.*, 15×10^4) and most of the flows pass through aggregate paths.

The third set of experiments evaluates the cumulative distribution function (CDF) of occupied flow entries on al-l switches. The horizontal ordinate is the number of flows ranging from 3×10^4 to 18×10^4. We compare the number of occupied flow entries for PrePass with RLJDM. As shown in Figures 11 and 12, PrePass needs no more than 4×10^3 flow entries, while RLJDM uses at most 10×10^3 flow entries. What's more, most of switches need 2×10^3-4×10^3 entries by PrePass. But for RLJDM, more than 79% of the switches exceed the flow table size constraint. That's because PrePass has deployed aggregate paths for partial macroflows, and flows in these macroflows will pass through their aggregate paths without requiring extra flow entries. Thus, PrePass can significantly reduce the number of occupied flow entries, which ensures the flow table size constraint on each switch.

The fourth set of simulations observes the network throughput performance when the flow table size is constant (*i.e.*, 4000). The results are shown in Figures 13 and 14. We observe that the throughput increases as the number of flows increases for all five algorithms. But RLJD and OSPF can accommodate much less traffic compared with Presto, ECMP and PrePass. That's because OSPF has high link load ratio and RLJD is constricted by the flow table size. ECMP has similar throughput as Presto in topology (a), but much less throughput compared with Presto in topology (b). For example, when the number of flows is 15×10^4, ECMP increases throughput by 4.8% compared with Presto in terms of the enterprise workload by Figure 13, But in topology (b) by Figure 14(a), Presto increases throughput by 27.7% compared with ECMP. Moreover, PrePass achieves much more throughput than other four benchmarks on both topologies and workload-s. For example, in Figure 14 with the enterprise workload, PrePass increases the throughput by 22.2%, 52.7%, 96.7% and 103.3% compared with Presto, ECMP, RLJD and OSPF, when the number of flows is 15×10^4. That's consistent with the above results that PrePass not only achieves similar LLR to RLJDM, but also satisfies the flow table size constraint.

From the above simulations in Figures 7-14, we can draw some conclusions. First, PrePass can reduce LLR by 30%, 40% and 63% compared with Presto, ECMP and OSPF for both two topologies and workloads. Second, PrePass satisfies the flow table size constraint while it only increases the link load ratio by about 5% compared with RLJDM.

6.3.3 Performance under Traffic Dynamics

The aggregate path deployment is based on the long-term traffic statistics collection and prediction. Due to the flow dynamics, the prediction error is unavoidable. Thus, it's necessary to evaluate the routing performance when the prediction error exists. There are two kinds of prediction errors, the number of flows in a macroflows and the traffic size of a macroflow. The proactive scheme ensures that one macroflow needs one and only one wildcard entry on each switch along the path, no matter how many flows the macroflow includes. Thus we only consider the error of traffic size prediction.

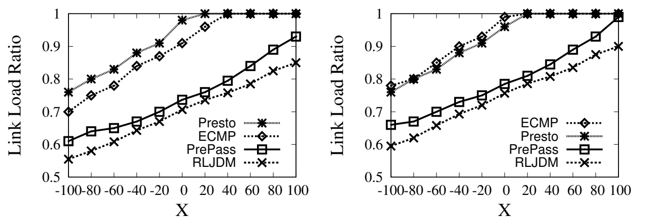


Figure 15: Prediction Error Parameter X vs. LLR in Topology (a). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

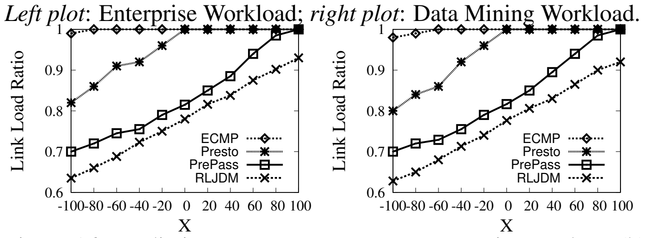


Figure 16: Prediction Error Parameter X vs. LLR in Topology (b). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

Without loss of generality, we will capture the prediction error with a single parameter X. Specifically, note that $f(\gamma)$ and $\bar{f}(\gamma)$ are the actual traffic size and predicted traffic size of macroflow γ respectively. We will simulate the case that the predicted traffic size of macroflow γ obeys the uniform distribution from (1-$X\%$)$\cdot\bar{f}(\gamma)$ to $\bar{f}(\gamma)$ or from $\bar{f}(\gamma)$ to (1-$X\%$)$\cdot\bar{f}(\gamma)$. The parameter X can be positive or negative, representing under-prediction and over-prediction respectively.

This section evaluates the impacts of traffic dynamics. According to Figures 7 and 8, when the number of flows is 12×10^4, the number of adjustable flows is about 4.5×10^4. It means the ratio of adjustable flows to all flows is 37.5%. Since most of the flows are forwarded through aggregate paths, we choose 12×10^4 to conduct our simulations by changing the prediction error parameter X from -100 to 100. The simulation results in terms of LLR are shown in Figures 15 and 16. When the error parameter X is 0, the gap between RLJDM and PrePass is the smallest. When $|X|$ increases, the gap increases too. For example, the gap between PrePass and RLJDM increases from 5% to 10% when X increases from 0 to 100 by Figure 15(a). What's more, PrePass performs better than Presto and ECMP, even when the prediction error parameter X is 100. That's because PrePass routes partial flows with per-flow rules to highly utilize the link capacity and reduce the prediction error caused by aggregate paths. In a word, PrePass can well adapt to the flow traffic dynamics and uncertainty, and achieve the similar LLR (less than 10%) to RLJDM even when the prediction error parameter X is 100.

6.3.4 Performance for the PrePassE Algorithm

For simplicity, the integrated routing algorithm, combing PrePassE and reactive scheme, is also denoted by PrePassE. Since we focus on the load balancing in this paper, this section mainly observes how our PrePassE algorithm changes the routing performance. The first set of simulations observes the link load ratio performance and the results are shown in Figures 17 and 18. Two figures show that PrePassE can reduce LLR by about 25%, 29.5%, and 60% compared with Presto, ECMP and OSPF, respectively. Moreover, our

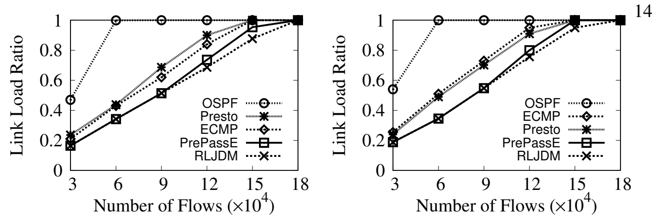


Figure 17: Number of Flows vs. Link Load Ratio (LLR) in Topology (a). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

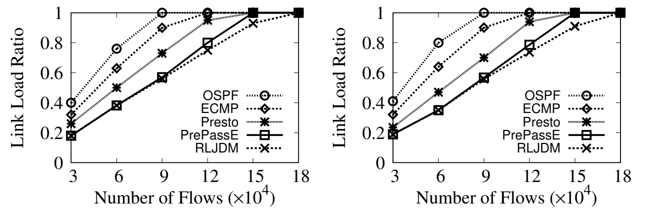


Figure 18: Number of Flows vs. Link Load Ratio (LLR) in Topology (b). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

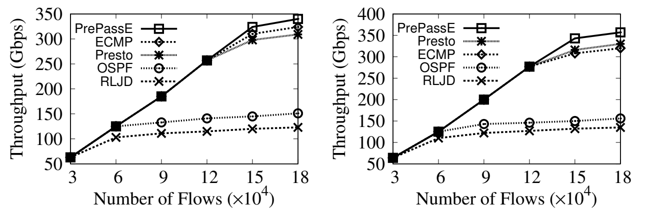


Figure 19: Number of Flows vs. Throughput in Topology (a). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

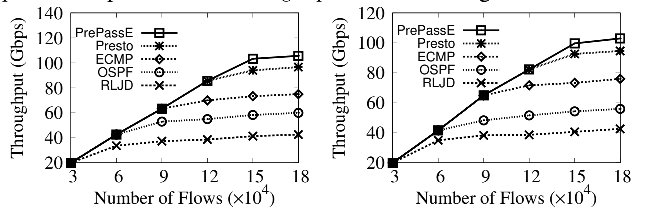


Figure 20: Number of Flows vs. Throughput in Topology (b). *Left plot*: Enterprise Workload; *right plot*: Data Mining Workload.

PrePassE algorithm can achieve almost the same LLR as RLJDM when the number of flows is less than 9×10^4. When the number of flows exceeds 9×10^4, PrePassE increases a little LLR compared with RLJDM. For example, when the number of flows is 12×10^4 and 15×10^4, PrePassE increases LLR by about 7% and 10% respectively from Figure 17(a). The second set of simulations studies the performance of network throughput. As shown in Figures 19 and 20, PrePassE can improve network throughput compared with four benchmarks. In particular, PrePassE increases throughput by 8.0%, 35.9%, 81.9%, and 146.7% compared with Presto, ECMP, OSPF, and RLJD, when the number of flows is 15×10^4. We note that Presto needs additional virtual switches for each physical switch. From these simulation results, we can conclude that our PrePassE algorithm can achieve close LLR to RLJDM, which required unlimited flow table size, and improve throughput than all other benchmarks, even if the controller is oblivious to the traffic size of each flow for deployment of aggregate paths.

7 Conclusion

In this paper, we study how to achieve load balancing by combining proactive routing and reactive routing in an SDN, to satisfy data plane resource constraints. We formulate the load balancing with flow table size constraint (LB-FTS) problem as an integer linear program. A rounding-based algorithm, PrePass, with bounded approximation factors is proposed to solve the LB-FTS problem. Some practical issues are discussed to enhance PrePass. We implement the proposed algorithms on an SDN testbed for experimental studies and use simulations for large-scale investigation. The experimental results and extensive simulation results show that our proposed method can satisfy the different resource constraints on switches, and only increase the link load ratio by about 5%-10% compared with per-flow routing scheme under various and dynamic traffic scenarios.

References

[1] The network topology from the monash university. `http://www.ecse.monash.edu.au/twiki/bin/view/InFocus/LargePacket-switchingNetworkTopologies`.

[2] Open vswitch: open virtual switch. `http://openvswitch.org/`.

[3] Ryu sdn framework. `http://osrg.github.io/ryu/`.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, number 4, pages 63–74. ACM, 2008.

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–19, 2010.

[6] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, G. Varghese, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Computer Communication Review*, number 4, pages 503–514. ACM, 2014.

[7] S. Banerjee and K. Kannan. Tag-in-tag: Efficient flow table management in sdn switches. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 109–117. IEEE, 2014.

[8] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz. Per-packet load-balanced, low-latency routing for clos-based data center networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 49–60. ACM, 2013.

[9] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. On the effect of forwarding table size on sdn network utilization. In *INFOCOM, 2014 Proceedings IEEE*, pages 1734–1742.

[10] L. Cui, F. R. Yu, and Q. Yan. When big data meets software-defined networking: Sdn for big data and big data for sdn. *IEEE network*, 30(1):58–65, 2016.

[11] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review*, number 4, pages 254–265. ACM, 2011.

[12] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2130–2138. IEEE, 2013.

[13] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 184–193. IEEE, 1975.

[14] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[15] S. Ghorbani, B. Godfrey, Y. Ganjali, and A. Firoozshahian. Micro load balancing in data centers with drill. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, page 17. ACM, 2015.

[16] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, number 4, pages 350–361. ACM, 2011.

[17] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, number 4, pages 51–62. ACM, 2009.

[18] E. Haleplidis, J. H. Salim, S. Denazis, and O. Koufopavlou. Towards a network abstraction model for sdn. *Journal of Network and Systems Management*, 23(2):309–327, 2015.

[19] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 2015.

[20] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *ACM SIGCOMM*, pages 15–26, 2013.

[21] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, page 10. ACM, 2016.

[22] J. Liu, Y. Li, and D. Jin. Sdn-based live vm migration across datacenters. In *ACM SIGCOMM Computer Communication Review*, number 4, pages 583–584. ACM, 2014.

[23] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM Computer Communication Review*, 38(2):69–74, 2008.

[24] R. Narayanan, S. Kotha, G. Lin, A. Khan, S. Rizvi, W. Javed, H. Khan, and S. A. Khayam. Macroflows and microflows: Enabling rapid network innovation through a split sdn data plane. In *Software Defined Networking, 2012 European Workshop on*, pages 79–84. IEEE, 2012.

[25] M. Y. O. Network. Ospf network design solutions. 2003.

[26] H. Owens II and A. Durresi. Video over software-defined networking (vsdn). *Computer Networks*, 92:341–356, 2015.

[27] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[28] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu. Hadoopwatch: A first step towards comprehensive traffic forecasting in cloud computing. In *INFOCOM, 2014 Proceedings IEEE*, pages 19–27. IEEE, 2014.

[29] B. Pfaff et al. Openflow switch specification v1.3.0, 2012.

[30] P. Raghavan and C. D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[31] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. Planck: Millisecond-scale monitoring and control for commodity networks. *ACM SIGCOMM Computer Communication Review*, 44(4):407–418, 2015.

[32] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM (JACM)*, 22(1):115–124, 1975.

[33] S. Sen, D. Shue, S. Ihm, and M. J. Freedman. Scalable, optimal flow routing in datacenters via local link balancing. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 151–162. ACM, 2013.

[34] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *NSDI*, pages 407–420, 2017.

[35] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen. Scotch: Elastically scaling up sdn control-plane using vswitch based overlay. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 403–414. ACM, 2014.

[36] H. Xu, H. Huang, S. Chen, and G. Zhao. Scalable software-defined networking through hybrid switching. In *Proc. IEEE INFOCOM*, 2017.

[37] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, and H. Wang. Incremental deployment and throughput maximization routing for a hybrid sdn. *IEEE/ACM Transactions on Networking*, 2017.

[38] H. Xu, Z. Yu, X.-Y. Li, L. Huang, C. Qian, and T. Jung. Joint route selection and update scheduling for low-latency update in sdns. *IEEE/ACM Transactions on Networking*, 2017.

[39] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.

[40] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury. Resilient datacenter load balancing in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 253–266. ACM, 2017.

[41] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang. On the effect of flow table size and controller capacity on sdn network throughput. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–6. IEEE, 2017.

[42] G. Zhao, H. Xu, S. Chen, L. Huang, and P. Wang. Deploying default paths by joint optimization of flow table and group table in sdns. In *Proc. IEEE ICNP*, 2017.

[43] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat. Wcmp: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, page 5. ACM, 2014.

A Proof of Lemma 10

Proof. We prove the NP-hardness by showing that the 0-1 knapsack problem [32] is a special case of our LB-MDF problem. Considering a special case of LB-MDF, in which the network only has one switch u (or all switches have infinite flow entries except that only one switch has limited flow table size). There are k macroflows $\{\gamma_1, \gamma_2, ..., \gamma_k\}$, in which each macroflow γ_i includes $|\gamma_i|$ flows. The flow table size on the switch u is $\beta(u)$. We construct a knapsack whose capacity is $\beta(u)-k$, and each item i whose weight and value are $w_i = |\gamma_i| - 1$ and $v_i = w_i = |\gamma_i| - 1$, respectively. The 0-1 knapsack problem is to maximize the total values of items in the knapsack while the total weight is less than or equal to the knapsack capacity. Assume that the optimal solution of the 0-1 knapsack problem is denoted by $B = \{b_1, b_2, ..., b_k\}$, where b_i is a binary variable representing whether the item i is selected (b_i=1) or not (b_i=0), and we have $\sum_{i=1}^{k} b_i \cdot w_i \le \beta(u) - k$. If the item i is selected by the optimal solution in the 0-1 knapsack problem, macroflow γ_i will choose proactive routing, otherwise, reactive routing. Thus the macroflow γ_i will occupy $b_i \cdot w_i + 1$ flow entries. The total number of occupied flow entries for all macroflows is $\sum_{i=1}^{k}(b_i \cdot w_i + 1) \le \beta(u) - k + k = \beta(u)$. Thus our constructed example of 0-1 knapsack can satisfy the flow table size constraint. Similar analysis for the objective to maximize the number of adjustable flows. Thus the optimal result B for 0-1 knapsack problem is the optimal result for the special case of LB-MDF problem. Obviously, the 0-1 knapsack problem is NP-hard [32]. Since the special case of our LB-MDF problem is NP-hard, LB-MDF is NP-hard too. □

B Proof of Lemma 11

To prove the lemma 11 , we first introduce a widely used formula.

Lemma 13 (Chernoff Bound)**.** *Given n independent variables: $x_1, x_2, ..., x_n$, where $\forall x_i \in [0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^{n} x_i]$. Then, $\mathbf{Pr}[\sum_{i=1}^{n} x_i \le (1-\xi)\mu] \le e^{\frac{-\xi^2\mu}{2}}$, where ξ is an arbitrarily positive value in [0,1].*

Since all flows in a macroflow adopt the same (proactive/reactive) routing scheme, we define a random variable π_γ to denote the number of adjustable flows in the macroflow γ.

Definition 4. *For each macroflow γ, a random variable π_γ is defined as:*

$$\pi_\gamma = \begin{cases} |\gamma|, & with\ probability\ \ 1 - z_\gamma \\ 0, & otherwise. \end{cases} \tag{22}$$

According to the definition, π_{γ_1}, π_{γ_2}... are mutually independent. The expected number of ajustable flows are:

$$\begin{aligned} \mathbb{E}\left[\sum_{\gamma\in\Gamma} \pi_\gamma\right] &= \sum_{\gamma\in\Gamma} \mathbb{E}[\pi_\gamma] = \sum_{\gamma\in\Gamma} |\gamma| \cdot (1 - z_\gamma) \\ &= \sum_{\gamma\in\Gamma} \sum_{p\in P_f, f\in\gamma} \widetilde{y}_f^p = \widetilde{\theta} \end{aligned} \tag{23}$$

Combining Eq. (23) and the definition of η in Eq. (21), we have

$$\begin{cases} \frac{\pi_\gamma \cdot \eta}{\widetilde{\theta}} \in [0,1] \\ \mathbb{E}\left[\sum_{\gamma\in\Gamma} \frac{\pi_\gamma \cdot \eta}{\widetilde{\theta}}\right] \le \eta. \end{cases} \tag{24}$$

Then, by applying Lemma 13, assume that ξ is an arbitrary positive value. It follows

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{\pi_\gamma \cdot \eta}{\widetilde{\theta}} \le (1-\xi)\eta\right] \le e^{\frac{-\xi^2\eta}{2}} \tag{25}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{\pi_\gamma \cdot \eta}{\widetilde{\theta}} \ge (1-\xi)\eta\right] \le e^{\frac{-\xi^2\eta}{2}} \le \frac{F}{n} \tag{26}$$

where F is the function of network-related variables (such as the number of switches n, *etc.*) and $F \to 0$ when the network size grows.

The solution for Eq. (26) is expressed as:

$$\xi \ge \sqrt{\frac{2\log n - 2\log F}{\eta}}, \quad n \ge 2 \tag{27}$$

Lemma 14. *After the rounding process, the number of adjustable flows will not less than the upper bound of LB-MDF by a factor of $1 - \xi = 1 - 2\sqrt{\frac{\log n}{\eta}}$.*

Proof. Set $F = \frac{1}{n}$. Eq. (26) is transformed into:

$$\mathbf{Pr}\left[\sum_{\gamma\in\Gamma} \frac{\pi_\gamma \cdot \eta}{\widetilde{\theta}} \le (1-\xi)\eta\right] \le \frac{1}{n^2}, \text{where } \xi \ge 2\sqrt{\frac{\log n}{\eta}}$$

The approximation factor of our algorithm is $1 - \xi = 1 - 2\sqrt{\frac{\log n}{\eta}}$. □

C Proof of Lemma 12

Before analyzing the flow table resource performance, we define a random variable b_γ^v to denote whether there deploys a flow entry on switch v for macroflow γ.

Definition 5. *For each macroflow γ and each switch v, a random variable b_γ^v is defined as:*

$$b_\gamma^v = \begin{cases} 1, & with\ probability \quad \sum_{p:v\in p, p\in\mathscr{P}_\gamma} \widetilde{y}_\gamma^p \\ 0, & otherwise. \end{cases} \tag{28}$$

According to the definition, $b_{\gamma_1}^v$, $b_{\gamma_2}^v$, ... are mutually independent. The expected number of occupied flow entries on switch v is:

$$\begin{aligned} \mathbb{E}\left[\sum_{\gamma\in\Gamma} b_\gamma^v\right] &= \sum_{\gamma\in\Gamma} \mathbb{E}[b_\gamma^v] = \sum_{\gamma\in\Gamma} \sum_{p:v\in p, p\in\mathscr{P}_\gamma} \widetilde{y}_\gamma^p \\ &\le \beta(v) - \sum_{\gamma\in\Gamma} \sum_{f:f\in\gamma; p:p\in\mathscr{P}_f, v\in p} \widetilde{y}_f^p \end{aligned} \tag{29}$$

Combining Eq. (29) and the definition of α in Eq. (3), we have

$$\begin{cases} \frac{b_\gamma^v \cdot \eta}{\beta(v)} \in [0,1] \\ \mathbb{E}\left[\sum_{\gamma \in \Gamma} \frac{b_\gamma^v \cdot \eta}{\beta(v)}\right] \leq \eta. \end{cases} \tag{30}$$

Then, by applying Lemma 3, assume that δ is an arbitrary positive value. It follows

$$\mathbf{Pr}\left[\sum_{\gamma \in \Gamma} \frac{b_\gamma^v \cdot \eta}{\widetilde{\lambda} c(e)} \geq (1+\delta)\eta\right] \leq e^{\frac{-\delta^2 \eta}{2+\delta}} \tag{31}$$

Now, we assume that

$$\mathbf{Pr}\left[\sum_{\gamma \in \Gamma} \frac{b_\gamma^v \cdot \eta}{\beta(v)} \geq (1+\delta)\eta\right] \leq e^{\frac{-\delta^2 \eta}{2+\delta}} \leq \frac{\mathscr{F}}{n^2} \tag{32}$$

where $\mathscr{F}$ is a variable defined in Subsection 4.2.

The solution for Eq. (32) is expressed as:

$$\delta \geq \frac{\log \frac{n^2}{\mathscr{F}} + \sqrt{\log^2 \frac{n^2}{\mathscr{F}} + 8\eta \log \frac{n^2}{\mathscr{F}}}}{2\eta}, \quad n \geq 2 \tag{33}$$

Lemma 15. *After the rounding process, the number of flow entries in each switch will not exceed the constraint $\beta(v)$ by a factor of $\delta + 2 = \frac{3\log n}{\eta} + 4$..*

Proof. Set $\mathscr{F} = \frac{1}{n}$. Eq. (15) is transformed into:

$$\mathbf{Pr}\left[\sum_{\gamma \in \Gamma} \frac{b_\gamma^v \cdot \eta}{\beta(v)} \geq (1+\delta)\eta\right] \leq \frac{1}{n^3}, \text{where } \delta \geq \frac{3\log n}{\eta} + 2$$

By applying Lemma 4, we have,

$$\begin{aligned} &\mathbf{Pr}\left[\bigvee_{v \in V} \sum_{\gamma \in \Gamma} \frac{b_\gamma^v \cdot \eta}{\beta(v)} \geq (1+\delta)\eta\right] \\ =&\mathbf{Pr}\left[\bigvee_{v \in V} \sum_{\gamma \in \Gamma} \frac{b_\gamma^v}{\beta(v)} \geq (1+\delta)\right] \\ \leq&\sum_{v \in V} \mathbf{Pr}\left[\sum_{\gamma \in \Gamma} \frac{b_\gamma^v \cdot \eta}{\beta(v)} \geq (1+\delta)\eta\right] \\ \leq& n \cdot \frac{1}{n^3} = \frac{1}{n^2}, \quad \delta \geq \frac{3\log n}{\eta} + 2 \end{aligned} \tag{34}$$

Note that the third inequality holds, because there are at most n switches in a network. After considering the occupied flow entries of aggregate paths, we have,

$$\begin{aligned} &\mathbf{Pr}\left[\bigvee_{v \in V} [\sum_{\gamma \in \Gamma} \frac{b_\gamma^v}{\beta(v)} + \sum_{\gamma \in \Gamma} \sum_{f: f \in \gamma; p: p \in \mathscr{P}_f, v \in p} \frac{\widetilde{y}_f^p}{\beta(v)}] \geq (2+\delta)\right] \\ &\leq \frac{1}{n^2}, \quad \delta \geq \frac{3\log n}{\eta} + 2 \end{aligned} \tag{35}$$

The approximation factor of our algorithm is $\delta + 2 = \frac{3\log n}{\eta} + 4$. □