Traffic-Aware Consistent Flow Migration in SDN

Ilora Maity, Student Member, IEEE Sudip Misra, Senior Member, IEEE,
and Chittaranjan Mandal, Senior Member, IEEE *

Abstract

In this paper, we present a traffic-aware consistent approach for flow migration in Software Defined Networking (SDN). The proposed scheme considers heterogeneous traffic to determine a consistent flow migration schedule. In a large-scale network, majority of traffic flows are latency sensitive. These flows change path frequently to accommodate new traffic flows. The challenge is to reduce the time required to modify the flow-path of latency sensitive flows. Existing solutions do not consider specific flow characteristics to decide a consistent traffic flow migration schedule. In this work, we propose a coalition graph game based strategy while prioritizing traffic flows based on latency sensitivity. The proposed scheme significantly reduces the migration duration of latency sensitive traffic flows. In particular, the average traffic flow migration duration is 15.43% less than existing timed two-phase update solution.

Keywords: Software Defined Networking (SDN), Flow Migration, Latency, Network Traffic, Coalition Graph Game.

1 Introduction

Software Defined Networking (SDN) separates control logic from forwarding elements which makes it appropriate for large-scale networks [1], [2]. The devices connected to these networks are huge in number and generate a massive number of traffic flows. Majority of these flows are short in size and latency sensitive [3]. Moreover, new traffic flows are generated frequently and the existing flows are required to migrate paths to accommodate these new flows. Therefore, traffic flow migration is an important aspect of network update in SDN [4]. In addition, completing traffic flow migration in minimal time is important. However, the migration process should not disrupt the consistency of the traffic flows [5].

During traffic flow migration, SDN controllers send update packets to the required SDN switches to change flow-path from an initial path to a final path and the switches install new flow-rules accordingly. Consistent traffic flow migration guarantees that the packets of a flow are either handled by old rules or new rules, not both. Moreover, a traffic flow is primarily classified into two categories — latency sensitive and throughput sensitive. Typically, long-lived traffic flows such as the flows involving VM migrations, data center synchronization, and data backup are throughput sensitive. On the other hand, traffic flows involving web browsing, email sending, social networking, or any other ephemeral flows are latency sensitive. The number of latency sensitive traffic flows are more than 80% of the total volume of flows in the network [3]. Existing flow migration solutions [6], [7], [8] do not consider the heterogeneous latency requirements of the traffic flows. Therefore, there is a need for a traffic-aware and consistent scheme for efficient migration of flows.

In this work, we propose a traffic flow migration scheme for SDN, which prioritizes flows based on latency sensitivity. The proposed approach for traffic-aware consistent flow migration in SDN, named COSMOS, consists of two primary modules — (a) coalition graph formulation, and (b) consistent flow migration. Initially, COSMOS generates a coalition graph to identify related flows. The coalitions are extracted from the graph in the order of their priorities. Thereafter, a feasible migration schedule is determined, and respective controllers update corresponding switches. This approach reduces the average waiting time for latency sensitive flows during flow migration.

The primary contributions of our work are as follows:

- We formulate the utility value for each traffic flow participating in the migration process. In addition, we design a coalition graph game to determine the traffic-aware flow migration schedule.
- We propose an algorithm to construct the coalition graph and define the coalitions.
- We design an algorithm for consistent, latency sensitive, and rule-space efficient traffic flow migration in SDN.

Simulation results depict that COSMOS reduces the average migration duration of the traffic flows.

The remainder of this paper is organized as follows. Section 2 discusses the existing approaches for traffic flow-path update in SDN. In Section 3, we discuss the system architecture and problem statement. Section 4 describes the coalition graph game-based approach. Section 5 depicts the experimental results. Finally, Section 6 concludes the proposed work and discusses directions for future work.

*The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India (Email: {imaity, sudipm, chitta}@iitkgp.ac.in).

2 Related Work

In this section, we discuss the existing literature related to traffic flow migration in SDN. Reitblatt *et al.* [6] introduced a two-phase update approach for ensuring consistency. In this approach, the first phase updates internal switches and the final phase updates ingress switches. Updated ingress switches install tags that contain the version numbers to incoming packets so that packets are entirely processed by a single version of rules. After the exhaustion of all the old packets, a garbage collection phase deletes the older rules. However, this approach wastes rule-space due to the storage of two versions of each rule for the entire flow migration duration. Canini *et al.* [9] presented a transactional flow-path update policy where either all updates are complete or none is initiated. McGeer *et al.* [10] proposed a buffered update technique which stores the incoming packets at the controller end during an ongoing update. Additional flow-rules are installed in each switch in order to redirect packets to the control plane. Mizrahi *et al.* [7] proposed a time-triggered approach for flow update which specifies starting time for each update phase. However, precise time synchronization depends entirely on the characteristics of particular switches. CURE [11] prioritizes switches based on workload and schedules update accordingly. This approach implements a queueing scheme to ensure consistency. CURE considers a control plane having a centralized controller.

Synthesis: From the exhaustive study of existing literature, it is evident that there exists a need for a consistent flow migration scheme for SDN, which reduces migration duration for latency sensitive traffic flows. Existing solution approaches do not consider the diverse traffic characteristics of traffic flows. Moreover, an unplanned schedule disrupts the operations of latency sensitive applications. Therefore, in this work, we consider flow-specific requirements to generate a delay-aware traffic flow migration schedule for SDN.

3 System Model

In this section, we discuss the SDN architecture and the problem statement for latency sensitive consistent update in SDN. Table 1 summarizes the key notations used in this work.

As shown in Figure 1, SDN involves heterogeneous devices including sensors, actuators, and mobile devices. These devices generate flows of data packets which are transferred to SDN switches through gateways. The latency sensitivity of a traffic flow is determined by the type of applications for which the flow is generated. The examples of latency sensitive applications are financial trading applications, online multiplayer games, and video conferencing applications.

SDN switches store packet forwarding instructions as flow-rules in table like structures called flow-tables [12]. Switches use Ternary Content Addressable Memory (TCAM) to store the flow-rules. The flow-rules are ternary strings of 0's, 1's, and *'s. The rule-space of each switch is limited due to the high manufacturing cost of TCAM. Each switch is managed by a controller which installs flow-rules for new traffic flows and updates existing rules.

Table 1: Summary of key notations

Symbol	*Definition*
C	Set of controllers
S	Set of switches
R_j	Set of flow-rules of $s_j \in S$
R^{max}	Rule storage capacity of $s_j \in S$
l_{ij}	Link between s_i and s_j
w_{ij}	Capacity of link l_{ij}
c_{ij}	Load of link l_{ij}
F	Set of traffic flows
$\alpha(f_i)$	Latency sensitivity index of $f_i \in F$
$S(f_i)$	Set of switches in the path of $f_i \in F$ before migration
$F^{'}$	Set of to-be-migrated traffic flows
E	Set of edges in coalition graph
$S^{'}(f_i)$	Set of switches in the path of $f_i \in F^{'}$ after migration
T_{f_i}	Flow setup time of $f_i \in F^{'}$
N_j	Number of overlapping flows of $f_j \in F^{'}$

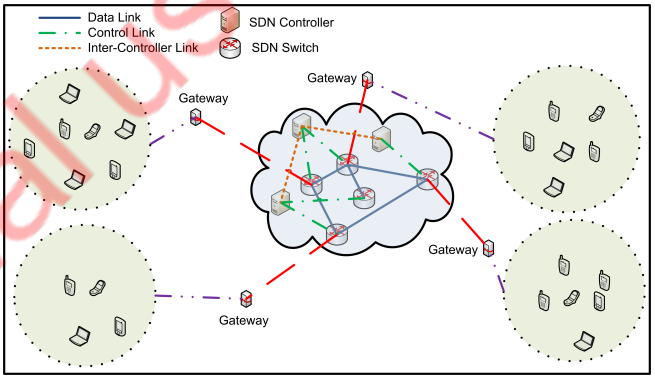


Figure 1: SDN Architecture

During traffic flow migration in SDN, controllers insert new flow-rules into the switches in the new path and delete old rules. Typically, large-scale networks require guaranteed service with minimum latency. Therefore, a flow migration schedule should ensure that all traffic flows are migrated consistently. Flow-level consistency is maintained when the packets of a traffic flow follow either old rules only or new rules only, after the initiation of network update.

Let C and S denote the set of controllers and the set of switches, respectively. The set of flow-rules for switch $s_j \in S$ is represented as R_j. The data link between s_i and s_j is denoted by l_{ij} and its capacity is denoted by w_{ij}.

F denotes the set of existing traffic flows in the network. A traffic flow $f_i \in F$ is denoted by a tuple $<src(f_i), dest(f_i), P(f_i), S(f_i), \alpha(f_i)>$, where $src(f_i)$ denotes the source, $dest(f_i)$ is the destination, $P(f_i)$ is the set of packets of f, $S(f_i) \subset S$ denotes the the set of switches along the path of f_i before migration, and $0 \leq \alpha(f_i) \leq 1$ signifies the latency sensitivity index (LSI) for f_i. A high $\alpha(f_i)$ indicates that f_i is highly latency

sensitive.

Let $F^{'} \subset F$ denote the set of to-be-migrated traffic flows. A traffic flow f_i is a member of $F^{'}$ if $\gamma(s_j) = 1, \exists s_j \in S(f_i)$, and $S(f_i) \neq S^{'}(f_i)$, where $S^{'}(f_i)$ is the set of SDN switches in the new path of f_i after migration. We assume that the source and destination of a traffic flow $f_i \in F^{'}$ do not change after migration. Therefore, $S(f_i) \cap S^{'}(f_i) \neq \phi$.

Let us consider that the network update procedure for traffic flow migration starts at time t_0. After t_0, a packet is termed old if it is handled by a to-be-updated switch. Otherwise, the packet is termed new. Therefore, the migration of a traffic flow $f_i \in F$ is consistent when an old packet in $P(f_i)$ follows old path only and a new packet in $P(f_i)$ follows new path only. We express consistent traffic flow migration as:

$$\Psi(f_i) = \begin{cases} 1 & \text{if the migration of } f_i \text{ is consistent,} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

At t_0, each switch in the new path receives an UPDATE signal from its master controller. Therefore, the set of to-be-updated switches are represented as:

$$\gamma(s_j) = \begin{cases} 1 & \text{if } s_j \in S \text{ received UPDATE signal,} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

The total setup time of a traffic flow f_i is given by:

$$T_{f_i} = \sum_{k=1}^{|S^{'}(f_i)|} \tau_k \gamma(s_k), \forall f_i \in F^{'}, \quad (3)$$

where τ_k is the average flow setup time of s_k.

3.1 Illustrative Example

Figure 2 depicts an example for traffic flow migration in SDN. We consider five switches s_1, s_2, s_3, s_4, and s_5. In addition, we consider two traffic flows f_1 and f_2. The old and new paths for each traffic flow are depicted in the figure as red dotted and green solid lines, respectively. Therefore, $\gamma(s_2) = 1$, $\gamma(s_4) = 1$, and $\gamma(s_3) = 1$ for the migration of f_1 and f_2. We consider link capacity as 1 Gbps for all links. The bandwidths of f_1 and f_2 are 1 Gbps each. We consider three different scenarios.

Scenario 1 considers f_1 as the traffic flow having the highest latency sensitivity. Controller schedules the update order as $f_1 \rightarrow f_2$. Switches s_2 and s_4 send Packet-In messages to the controller. For consistency, old packets of f_1 should be processed by s_3 and new packets should be processed by s_4 before reaching the destination. Moreover, due to this update order the link l_{45} becomes congested as it is part of both the new path of f_1 and the old path of f_2.

For scenario 2, let f_2 be the most latency sensitive traffic flow. Controller schedules the update order as $f_2 \rightarrow f_1$.

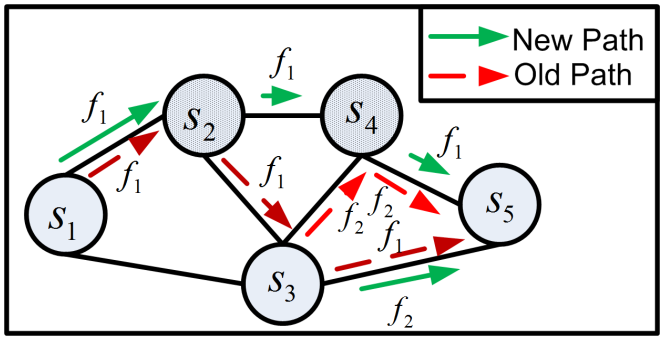


Figure 2: Illustrative Example

Switch s_3 sends Packet-In message to the controller for installation of new rules corresponding to the new path. For consistency, old packets of f_2 should be processed by s_4 and new packets should reach the destination directly. Moreover, due to this update order, the link l_{35} becomes congested as it is part of both the new path of f_2 and the old path of f_1.

In scenario 3, f_1 and f_2 have the same latency sensitivity. Therefore, if we update s_2, s_4, and s_3 together then we need to consider processing of the old packets consistently.

So, we need to group correlated to-be-updated traffic flows based on their latency sensitivity to avoid link congestion and packet inconsistency. Therefore, we design a coalition graph game which forms coalitions of traffic flows to generate an optimal migration schedule.

3.2 Problem Formulation

The migration of a flow f_i involves the update of each switch s_j in the set $S^{'}(f_i)$ having $\gamma(s_j) = 1$. Moreover, the update of f_i is consistent when the packets of f_i follow either new rules or old rules only, after the initiation of the network update. In addition, majority of the traffic flows in large-scale networks are latency-sensitive and are required to be updated within a short duration. Therefore, it is required to design a consistent flow migration scheme, which is aware of latency sensitivity of traffic flows and is free from link congestion.

In this work, we formulate a coalition graph game with non-transferable utility to minimize the total update completion time of the traffic flows in $F^{'}$ while maintaining flow-level consistency. Here, the players of the game are the to-be-migrated traffic flows. Each coalition $A_k \subset F^{'}$ denote the set of traffic flows $\{f_1, f_2, \ldots, f_{|A_k|}\}$ which are migrated simultaneously. Within a coalition, the traffic flow having the highest LSI is termed as the coalition-head. Therefore, a coalition-head has $|A_k| - 1$ children nodes, which are termed as coalition members. A coalition structure is a set of coalitions defined as:

$$V_A = \{A_1, A_2, \ldots, A_M\}, \quad \text{where} \bigcup_{k=1}^{M} A_k = F^{'}, A_i \cap A_j = \phi, \forall i \neq j \quad (4)$$

The controllers try to maximize the cumulative payoff obtained from the utility functions of the coalitions. Let $U(A_i, V_A)$ denote the utility value of a coalition $A_i \in V_A$ and $u_j(.)$ denote the utility value of a player $f_j \in A_i$. The marginal utility of each traffic flow f_j increases with decrease in the migration duration of the flow. Mathematically,

$$\frac{\partial u_j(.)}{\partial T_{f_j}} < 0 \quad (5)$$

The utility function $u_j(.)$ varies linearly with the LSI and the number of overlapping flows of each flow so that a high number of flows are migrated at a time depending on their traffic characteristics. Therefore, we get:

$$\frac{\partial u_j(.)}{\partial \alpha(f_j)} > 0 \text{ and } \frac{\partial u_j(.)}{\partial N_j} > 0 \quad (6)$$

Therefore, we utility function of a traffic flow f_j as:

$$u_j(.) = N_j \left(\alpha(f_j) - \frac{T_{f_j}}{T^{max}} \right), \quad (7)$$

where T^{max} is a constant denoting the maximum allowable duration for the migration of a traffic flow without violating the quality of service (QoS) constraints. N_j is the number of *overlapping flows*. The *overlapping flows* of f_j are the traffic flows in the coalition having at least one link in the old (new) path that overlaps with a link in the new (old) path of f_j. Hence, the utility function $U(A_i, V_A)$ is formulated as:

$$U(A_i, V_A) = \begin{cases} \sum_{f_j \in A_i} u_j(.) & \text{if } |A_i| > 1, \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

The total utility of all the coalitions in a coalition structure V_A is given by:

$$U(V_A) = \sum_{i=1}^{M} U(A_i, V_A) \quad (9)$$

Therefore, the coalition graph formulation is described mathematically as:

$$\underset{V_A}{\text{Maximize}} \quad U(V_A) \quad (10)$$

subject to

$$\Psi(f_i) = 1, \forall f_i \in F^{'}, \quad (11)$$

$$R_j \leq R^{max}, \quad \forall s_j \in S^{'}(f_i), \forall f_i \in F^{'}, \text{ where } \gamma(s_j) = 1, \quad (12)$$

$$T_{f_j} \leq T^{max}, \forall f_i \in F^{'}, \quad (13)$$

$$c_{ij} \leq w_{ij}, \forall s_i, s_j \in S \quad (14)$$

Equation (11) expresses the flow-level consistency constraint for all traffic flows in the network during update. Equation (12) represents the capacity constraint of switches, where R^{max} is the rule storage capacity of a switch. The rule storage capacity of a switch is defined as the maximum number of flow-rules that can be stored in the flow tables of the switch. In this work, we consider homogeneous switches which have the same amount of TCAM and rule storage capacity. Equation (13) ensures that the flow setup time for each traffic flow does not exceed the maximum allowable flow migration duration. Equation (14) denotes the link capacity constraint for all data links, where c_{ij} is the load of link l_{ij}.

The objective, however, is a combinatorial problem having high complexity when the number of traffic flows increases. Therefore, we design a heuristic algorithm for solving the problem.

4 COSMOS: The proposed scheme

In this section, we discuss the proposed scheme, COSMOS, in details. Based on the aforementioned coalition graph game formulation, we construct a coalition graph $G = (F^{'}, E)$, where $F^{'}$ is the set of to-be-migrated flows and E is the set of edges that denotes dependency between the to-be-migrated traffic flows. There exists an edge between $f_i \in F^{'}$ and $f_j \in F^{'}$ if f_i is an overlapping flow of f_j. We extract the feasible coalitions one by one from the coalition graph starting from the coalition containing the coalition-head with the highest LSI. Finally, we propose a consistent flow migration algorithm to migrate the traffic flows.

4.1 Coalition Graph Formation

The to-be-updated traffic flows, which are the players of the coalition graph game, form the coalition graph based on the utility function defined in Equation (9). In this game, each player is interested in finding the overlapping flows in its coalition to improve its utility. Therefore, we consider that the proposed coalition graph game is hedonic, which implies that a player has a preference for the choice of coalition. The preference relation is defined as:

***Definition* 1** (Preference Relation)**.** *The relation $V_A \succ_{F^{''}} V_B$ denotes that the way V_A partitions $F^{''}$ is preferred to the way V_B partitions $F^{''}$, where $F^{''} \subseteq F^{'}$ is a set of players.*

The coalitions are updated periodically based on merge and split rules as follows:

***Definition* 2** (Merge Rule)**.** *Merge any set of coalitions $\{A_1, A_2, \ldots, A_k\}$ where $\{\bigcup_{i=1}^{k} A_i\} \succ_{F^{''}} \{A_1, A_2, \ldots, A_k\}$,*

$F'' = \bigcup_{i=1}^{k} A_i$. *Therefore,* $\{A_1, A_2, \ldots, A_k\} \rightarrow \bigcup_{i=1}^{k} A_i$.

***Definition* 3** (Split Rule)**.** *Split any set of coalitions* $\bigcup_{i=1}^{k} A_i$ *where* $\{A_1, A_2, \ldots, A_k\} \succ_{F''} \{\bigcup_{i=1}^{k} A_i\}$, $F'' = \bigcup_{i=1}^{k} A_i$. *Therefore,* $\bigcup_{i=1}^{k} A_i \rightarrow \{A_1, A_2, \ldots, A_k\}$.

We propose the Coalition Graph Formation Algorithm (CGFA), which uses merge and split rules to generate the coalition graph.

Algorithm 1 Coalition Graph Formation Algorithm (CGFA)

INPUT: $F^{'}$
OUTPUT: $\{E, V_A\}$
PROCEDURE:
1: **for all** $f_i \in F^{'}$ **do**
2: $A_i \leftarrow A_i \cup \{f_i\}$
3: $A_i\_head \leftarrow f_i$ ▷ `Coalition head`
4: $V_A \leftarrow V_A \cup \{A_i\}$ ▷ `Coalition structure`
5: **end for**
6: **do**
7: Sort V_A in descending order of the α of the coalition heads
8: Store the ordered list of coalitions in V_A
9: $V_A^{initial} \leftarrow V_A$
10: **for all** $A_j \in V_A$ **do**
11: Merge or split A_j using Definition 2 and Definition 3
12: **if** $f_k \in A_j$ overlaps with $f_l \in A_j$ and $(f_k, f_l) \notin E$ **then**
13: $E \leftarrow E \cup (f_k, f_l)$
14: **end if**
15: Select the flow with the highest LSI as A_j_head and sort A_j in descending order of LSI of the member flows
16: **end for**
17: Update V_A
18: **while** $V_A \neq V_A^{initial}$
19: **return** $\{E, V_A\}$

Algorithm 1 describes the proposed algorithm, i.e., CGFA. Initially, each traffic flow forms an individual coalition. CGFA sorts the coalitions in descending order based on the LSI of the coalition-heads. Next, new coalitions are formed using Definitions 2 and 3. For each coalition, the traffic flow having the highest LSI is marked as the coalition-head. This process is repeated until no further update of V_A is possible.

Figure 3 shows an example of coalition graph generation process for a network having 11 switches and 5 traffic flows. For this example, we consider the Sprint topology [13] as the network topology. Figure 4 depicts the Sprint topology. In this example, $F^{'} = \{1, 2, 3, 4, 5\}$, $E = \{(5, 1), (2, 1), (3, 4)\}$. The possibility of parallel edges between nodes 3 and 4 is eliminated by line 12 of Algorithm 1. Initially, each traffic flow forms a coalition with utility 0 according to Equation (8). The sets of to-be-updated switches for the five traffic flows are $\{4, 5, 6, 7\}$, $\{4, 0, 7\}$, $\{7, 6\}$, $\{6, 7\}$, and $\{8, 7\}$, respectively. Let the average flow setup time for each switch is 1 ms and $T_{max} = 10$. Therefore, according to merge and split rules, the players in set $F^{'}$ are divided into two coalitions $\{1, 2, 5\}$ having utility $2(0.6 - \frac{4}{10}) + 2(0.4 - \frac{3}{10}) + 2(0.3 - \frac{2}{10}) = 0.8$ and $\{3, 4\}$ having utility $1(0.8 - \frac{2}{10}) + 1(0.8 - \frac{2}{10}) = 1.2$. The coalition-heads for coalitions $\{1, 2, 5\}$ and $\{3, 4\}$ are 1 and 4, respectively. The LSI of coalition-heads 1 and 4 are 0.6 and 0.8, respectively. Therefore, the ordered coalition-structure is $V_A = \{\{3, 4\}, \{1, 2, 5\}\}$.

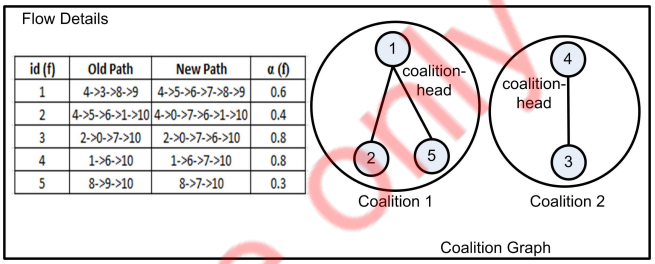


Figure 3: Coalition Graph Generation Example

4.2 Consistent Flow Migration

Algorithm 2 describes the process of traffic flow migration. The Consistent Flow Migration Algorithm (CFMA) selects the coalitions from V_A one-by-one. Initially, CFMA processes the old packets in the selected coalition and starts queuing the new packets at the switches. This step ensures packet-level consistency. After processing the all the old packets, new rules are installed and old rules are deleted. This step addresses rule-space constraint of the switches as only a single version of a flow-rule is stored at a time. After the modification of all the required rules, CFMA processes the queued packets.

Algorithm 2 Consistent Flow Migration Algorithm (CFMA)

INPUT: V_A
OUTPUT: F^p ▷ `Set of migrated traffic flows`
PROCEDURE:
1: **for all** $A_i \in V_A$ **do**
2: **for all** $f_j \in A_i$ **do**
3: Process old packets
4: Queue new packets
5: **end for**
6: **for all** $f_j \in A_i$ **do**
7: Install new rules
8: Delete old rules
9: **end for**
10: **for all** $f_j \in A_i$ **do**
11: Process queued packets
12: $F^p \leftarrow F^p \cup \{f_j\}$
13: **end for**
14: **end for**
15: **return** F^p

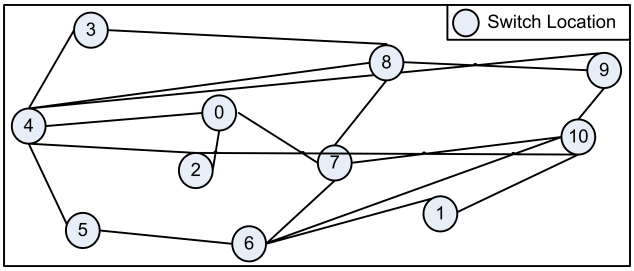


Figure 4: Sprint Topology

Table 2: Simulation parameters

Parameter	Value
Topology	Sprint [13]
Number of traffic flows	$200 - 1000$
Number of switches	11
Number of controllers	3
Maximum controller-to-switch delay	4.865 ms [11]
Maximum end-to-end network delay	0.262 ms [11]
Maximum time interval between dispatch of two consecutive update messages	5.24 ms [11]
Packet arrival rate per switch	$0.005 - 0.025$ mpps [11]
Average packet service rate per switch	0.03 mpps [11]
T^{max}	$1 - 250$ ms [14]

5 Performance Evaluation

5.1 Simulation Settings

We evaluate the performance of COSMOS by implementing a discrete event simulator in MATLAB. We use the Sprint topology [13], which is shown in Figure 4. The Sprint topology has 11 switches and 18 links. Table 2 represents the simulation parameters. The number of traffic flows in the network is varied between 200 and 1000. For each flow, we select the source-destination pair randomly.

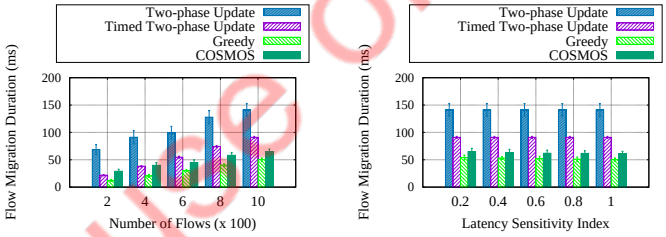
5.2 Benchmark schemes

We compare the performance of COSMOS with three benchmark schemes — two-phase update [6], timed two-phase update [7], and Greedy approach. The first two benchmark schemes consider no flow priority and schedule all traffic flows together for migration. The two-phase update scheme updates the ingress switches after updating the internal switches. In the timed two-phase update, the update of each phase starts at a pre-determined time instant. In both of these approaches, all flows complete migration only when all the switches are updated. In the Greedy approach, flows with similar LSI values are grouped together and the groups are migrated one-by-one according to the average LSI values of the groups starting with the group having the maximum average LSI. On the other hand, COSMOS considers flow-specific QoS requirements while preparing the migration schedule and migrates the flows consistently as described in CFMA.

5.3 Performance Metrics

We consider the following metrics to analyze the performance of COSMOS:

- *Flow migration duration*: The migration duration of a traffic flow is the interval between migration start time and migration end time of the corresponding coalition.
- *Queueing delay*: The queueing delay of a packet is its waiting time in the switch queue.
- *QoS violated flows*: QoS violated flows are traffic flows which do not satisfy link capacity constraint stated in Equation (14) or has a migration completion time greater than T^{max}.



(a) Effect of Traffic Load (b) Effect of LSI

Figure 5: Flow Migration Duration

5.4 Result and Discussion

5.4.1 Flow Migration Duration

From Figure 5a, we observe that the average flow migration duration for COSMOS is 55.17% and 15.43% less than the two-phase update and timed two-phase update, respectively. This is because, COSMOS migrates the flows incrementally, resulting in reduced waiting time for each traffic flow. However, the average flow migration duration for COSMOS is higher than the Greedy approach because COSMOS estimates overlapping flows and schedules overlapping flows together to address the link capacity constraint. Figure 5b depicts the effects of LSI on the average flow migration duration for 1000 flows. We observe that the average flow migration duration for both COSMOS and Greedy decreases as the latency sensitivity

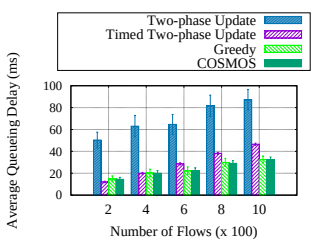


Figure 6: Queueing Delay

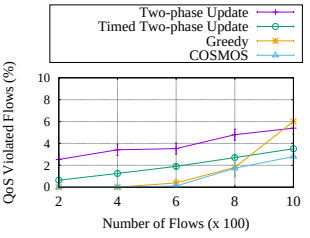


Figure 7: QoS Violated Flows

of migrating traffic flows increases. However, the change of flow latency sensitivity has no effect on the migration duration of benchmark schemes.

5.4.2 Queueing Delay

From Figure 6, we infer that average queueing delay for COSMOS is 65.84%, 18.44%, and 1.08% less than the two-phase update, timed two-phase update, and Greedy approach, respectively. This result implies that although COSMOS queues packets for maintaining consistency, the packets are processed with a consistent rate.

5.4.3 QoS Violated Flows

From Figure 7, we observe that the number of QoS violated flows in COSMOS is 76.34% less than the same using two-phase update, 53.50% less than the same using timed two-phase update, and 18.90% less than the same using Greedy approach. This is due to the fact that COSMOS migrates the traffic flows in order of their latency sensitivity so that each flow fulfills the specific QoS demand. Additionally, COSMOS considers the link capacity constraint by scheduling the overlapping flows together. On the other hand, the two-phase update schemes complete migration only after all ingress switches are updated in the last phase. In the Greedy approach, the number of QoS violated flows increases with the number of flows as higher number of overlapping flows increases the occurrence of link congestion. Therefore, the benchmark schemes are not able to meet QoS demands in realistic networks.

6 Conclusion

In this paper, a traffic-aware scheme for consistent flow migration in SDN was presented. A coalition graph game was designed to divide the traffic flows into multiple coalitions based on heterogeneous latency sensitivity of the flows. We proposed a priority based approach to migrate the traffic flows in each coalition consistently. The proposed scheme reduces the average flow migration duration by 15.43% and the number of QoS violated flows by 53.50% compared to timed two-phase update.

The future extension of this work involves addressing the influence of different network topologies, including data center networks (DCNs) on traffic flow migration. We also plan to modify our scheme considering loss sensitive and jitter sensitive traffic flows along with latency sensitive flows.

References

[1] A. Mondal, S. Misra, and I. Maity, "Buffer Size Evaluation of OpenFlow Systems in Software-Defined Networks," *IEEE Systems Journal*, pp. 1–8, 2018, doi: 10.1109/JSYST.2018.2820745.

[2] S. Bera, S. Misra, and M. S. Obaidat, "Mobility-Aware Flow-Table Implementation in Software-Defined IoT," in *Proc. IEEE GLOBECOM*, Dec 2016, pp. 1–6.

[3] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proc. ACM SIGCOMM*. New York, NY, USA: ACM, 2010, pp. 267–280.

[4] A. Basta, A. Blenk, S. Dudycz, A. Ludwig, and S. Schmid, "Efficient Loop-Free Rerouting of Multiple SDN Flows," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 948–961, Apr. 2018.

[5] S. Vissicchio, L. Vanbever, L. Cittadini, G. G. Xie, and O. Bonaventure, "Safe Update of Hybrid SDN Networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1649–1662, Jun. 2017.

[6] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for Network Update," in *Proc. ACM SIGCOMM*, New York, NY, USA, 2012, pp. 323–334.

[7] T. Mizrahi, E. Saat, and Y. Moses, "Timed Consistent Network Updates in Software-Defined Networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3412–3425, Dec. 2016.

[8] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella, "Simultaneously Reducing Latency and Power Consumption in OpenFlow Switches," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 1007–1020, Jun. 2014.

[9] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "Software Transactional Networking: Concurrent and Consistent Policy Composition," in *Proc. HOT SDN*. New York, NY, USA: ACM, 2013, pp. 1–6.

[10] R. McGeer, "A Safe, Efficient Update Protocol for Openflow Networks," in *Proc. HOT SDN*, New York, NY, USA, 2012, pp. 61–66.

[11] I. Maity, A. Mondal, S. Misra, and C. Mandal, "CURE: Consistent Update With Redundancy Reduction in SDN," *IEEE Trans. Commun.*, vol. 66, no. 9, pp. 3974–3981, Sep. 2018.

[12] "OpenFlow Switch Specification (Version 1.5.1): Open Networking Foundation," Mar. 2015.

[13] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[14] S. F. Abedin, M. G. R. Alam, S. M. A. Kazmi, N. H. Tran, D. Niyato, and C. S. Hong, "Resource Allocation for Ultra-Reliable and Enhanced Mobile Broadband IoT Applications in Fog Network," *IEEE*

Trans. Commun., vol. 67, no. 1, pp. 489–502, Jan. 2019.