

Flow Update Model Based on Probability Distribution of Migration Time in Software-defined Networks

Reo Uneyama, *Graduate Student Member, IEEE*, Takehiro Sato, *Senior Member, IEEE*, and Eiji Oki, *Fellow, IEEE*

Abstract—In a software-defined network (SDN), routes of packet flows need to be updated in situations such as maintenance and router replacement. Each flow is migrated from its old path to new path. The SDN update has an asynchronous nature; the time when the switches process commands by the controller varies depending on flows. Therefore, it is difficult to control an order of flow migrations, and packets can be lost by congestion. Existing models divide the time axis into rounds and assign migrations to these rounds. However, congestion caused by multiple migrations in the same round is uncontrollable. Based on the probability distribution of time required for each migration, congestion can occur. This paper proposes a flow update model which minimizes the expected amount of excessive traffic by shifting the probability distributions. The time axis is divided into time slots which are fine-grained than rounds, so that each probability distribution is shifted. The proposed model assigns the time when the controller injects a command of flow migration to time slots. The proposed model is formulated as an optimization problem to determine the command times to minimize the expected amount. This paper introduces two methods to compute the expected amount. This paper also introduces a two-stage scheduling scheme (2SS) that divides the optimization problem into two stages. 2SS suppresses the computation time from $\mathcal{O}(|T|^{|F|-1})$ to $\mathcal{O}(|T|^{\frac{|F|-1}{2}})$ at the cost of including at most 0.12% error. 2SS suppresses the amount of excessive traffic than an existing model by at most 71.2%.

Index Terms—Software-defined network, network update problem, capacity consistency, probability distribution, two-phase commit

I. INTRODUCTION

Network operators need to update configurations of their networks according to situations, such as maintenance, router replacement, and failures of network component. There are situations that require maintenance, router replacement, and response to failures. Updating the network is necessary in such situations. In the network before the advent of software-defined network (SDN), each router controls routes of packets. Therefore, when the network needs to be updated, settings of all routers need to be changed. In contrast, an SDN is an architecture which enhances flexibility of the network [1]. An SDN decouples controls of forwarding packets from the data plane, and integrates the controls into a logically centralized

controller (control plane). This decoupling allows the network to be updated flexibly since forwarding can be controlled independently of hardware constraints in the data plane. OpenFlow [2], [3] is one of the communication protocols which realize an SDN. OpenFlow switches perform actions to packets that match forwarding rules installed by the controller.

A network update problem is the problem of finding how to update a route of each packet flow when its paths in pre-update and post-update are given. Network update problems are divided into three main categories; connectivity consistency, policy consistency, and capacity consistency [4]. Connectivity consistency ensures that packets are delivered to their respective destinations. Packet loss may occur by forwarding loops [5] or blackholes [6]. Forwarding loop is a loop in paths of packets in intermediate states. Blackhole is a phenomenon that packets cannot be transferred and are lost, and the sender is not notified of these losses. The aim of connectivity consistency ensures loop-freedom and blackhole-freedom. Requirements in network update often go beyond connectivity. For example, policy consistency guarantees requirement to force a given node (Waypoint), e.g. a firewall, to be passed through. This enforcement is called waypoint enforcement (WPE) [7]. Since computer networks have limited capacity, it is also important that the capacity is not exceeded. When the capacity is exceeded, losing packet is called congestion. The aim of capacity consistency is to suppress packet loss by congestion. This paper deals with capacity consistency.

Updating an SDN is asynchronous; the time which it takes for flow migration varies depending on flows. Therefore, a flow of the initial state and another flow of the final state may exist on the same link at the same time. In this case, if the total amount of traffic exceeds the capacity of the link, the congestion occurs and packets are lost. Round-based model [8]–[11] addressed the asynchronous nature of updating an SDN. The model divides time axis into rounds that ensure enough time required for flow migrations. The controller commands migration of each flow at the beginning of each round. Since each migration is always completed in one round, migrations assigned to different rounds do not affect each other.

Since an order in which flows are migrated in the same round is uncontrollable, the round-based model cannot suppress the amount of excessive traffic in the round. Given that time required for each flow migration follows a certain probability distribution, the total amount of traffic exceeds the link capacity with the probability derived from the probability

This work was supported in part by JSPS KAKENHI Grant Numbers 21H03426 and 23H03382, and JST, PRESTO Grant Number JPMJPR23P4, Japan. Reo Uneyama, Takehiro Sato, and Eiji Oki are with the Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan (e-mail: uneyama.reo.38s@st.kyoto-u.ac.jp; takehiro.sato@i.kyoto-u.ac.jp; and oki@i.kyoto-u.ac.jp).

distributions. If the probability distributions can be shifted in the time axis, the amount of excessive traffic can be suppressed.

This paper proposes a flow update model which minimizes the expected amount of excessive traffic by shifting the probability distributions. In the proposed model, time axis is divided into time slots which are fine-grained than rounds. By shifting the probability distributions along time slots, the probability of occurring congestion can be suppressed. As a result, the expected amount of excessive traffic is suppressed. The proposed model is formulated as an optimization problem. This paper presents algorithms which compute the expected amount of excessive traffic from the shifted probability distributions. This paper also presents a two-stage scheduling scheme (2SS), which divides the optimization problem into two stages. The aim of introducing 2SS is to reduce the computation time to obtain an efficient schedule of suppressing the expected amount of excessive traffic. 2SS reduces the computation time from $\mathcal{O}(|T|^{|F|-1})$ to $\mathcal{O}(|T|^{\frac{|F|-1}{2}})$, instead of allowing that the optimal solution is not always obtained. Numerical results show that the expected amount of excessive traffic with 2SS has at most 0.12% error compared to the optimal amount. 2SS suppresses the expected amount of excessive traffic than an existing model by at most 71.2%.

The remainder of this paper is organized as follows. Section II describes the related works. Section III reveals issues of the round-based model in updating an SDN, and describes the motivation of shifting probability distributions. In Section IV, the proposed model is formulated. Section V introduces a method that suppresses the computation time taken to solve the formulated problem. Section VI presents methods and algorithms which compute the amount of excessive traffic. Numerical results are shown in Section VII, and Section IX concludes this paper.

II. RELATED WORKS

In this section, we introduce related works. The overview of network update problem is introduced in Section II-A. In Section II-B, works on congestion-aware network update problem are introduced.

A. Network update problem

Network update problems are divided into three main categories; connectivity consistency, policy consistency, and capacity consistency [4]. Connectivity consistency includes two categories; loop freedom [5], [12], [13] and blackhole freedom [6], [14]. Loop freedom ensures that packet flows do not use paths having loops in intermediate states. Blackhole freedom ensures that packets are transferred without disappearing. Policy consistency includes two categories; per-packet consistency (PPC) [15], [16] and waypoint enforcement (WPE) [7], [17]. PPC ensures that every packet passes through the initial or the final paths, each of which is guaranteed to satisfy the requirements such as security and service level agreement (SLA) compliance. PPC is a strict constraint since an intermediate path cannot be used. On the other hand, WPE relaxes the constraint of routing; it allows a packet to pass

through any path as long as it passes through specific nodes (e.g. a firewall) while network updating. It is not always necessary for the packet's path to exactly match the initial path or the final path. Network providers can choose PPC or WPE according to the requirements of the network. The two-phase commit technique [10], [15], [18] is one of techniques which guarantee PPC. Capacity consistency includes two categories; congestion aware [19] and latency aware [20]–[22]. Congestion aware consistency suppresses the amount of packet loss. In latency aware consistency, the amount of packet loss is suppressed under the condition that the latency that flows take to pass through links is guaranteed.

This paper addresses the congestion-aware network update problem which aims to suppress the amount of packet loss.

B. Congestion-aware network update problem

In this section, related works on congestion-aware network update problem are introduced.

Wen et al. [8] assumed a scenario in which congestion can be avoided by limiting the amount of demand for each flow. They presented an update scheduling that minimizes the amount of limitation of flow demands in network updates. Flows are unsplittable and can take intermediate routes in addition to the initial and final routes. In this work [8], the importance weight of each flow is defined and the weighted sum of the limitation amount of flow demands is minimized.

Zheng et al. [9] introduced an update method which minimizes transient congestion in the network, in which there are unsplittable flows and splittable flows. The transient congestion is defined as the maximum link congestion relative to its capacity during updating the networks. Unsplittable flows must use single path routing during update; splittable flows can use multipath routing. The aim in this work [9] is to find the optimal routing for all intermediate states that minimizes the transient congestion.

Luo et al. [10] presented an SDN update based on a two-phase commit. The two-phase commit is an update method in which flows do not take intermediate routes and are migrated from the initial routes to the final routes. They assumed that flows are unsplittable. The aim in this work [10] is to minimize the number of rounds, which are migration steps, while guaranteeing congestion-free or suppressing the amount of packet loss as small as possible.

Hong et al. [11] presented software-driven wide area network (SWAN), which enables an efficient and flexible inter-datacenter wide area network (WAN). They assumed that each flow is splittable and each link has an extra capacity which is called scratch capacity. SWAN can avoid congestion by dividing demands of flows and using the scratch capacity to migrate flows little by little in multiple steps. It is guaranteed that network update is completed by $\lceil \frac{1}{s} \rceil - 1$ steps when the ratio of the scratch capacity to the link capacity is s .

Jin et al. [23] presented a network update approach of dynamic scheduling that the migration ordering is decided while network updating. They pointed out that the existing network update methods based on static scheduling are slow since migration ordering is decided in advance. When there are

some congestion-avoidable orderings, the presented approach dynamically finds the ordering which suppresses congestion among those orderings. To find the ordering dynamically, they introduced a dependency graph, which allows for flexible and fast network update.

Zheng et al. [24] addressed congestion-free routing problem of multiple flows in Timed SDNs, in which individual node updates can be scheduled at specific times. Their approach is based on a time-extended network and a dependency graph. The time-extended network makes capturing the dynamic process of flow migration visible.

He et al. [25] presented an algorithm which guarantees three consistencies of network update problem in a continuously changing network. They pointed that “Update algebra,” which can handle multiple updates, guarantees only loop freedom and blackhole freedom. In this work [25], congestion freedom is also considered in the continuous network update based on an operation dependency graph. This graph enables us to find the relationship between update operation and the activity ratio of link.

Hezaveh et al. [26] presented an algorithm of continuous network update considering priority of each flow. They pointed that network update in SDN is more frequent than that in non-SDN since dynamic update of network application is easy. In this work [26], flow priority is considered in addition to the conventional model of continuous network update.

Compared to the related works [8]–[11], the proposed model in this paper takes the timeslot-based model. The related work [9] aims to minimize the maximum ratio of transmission capacity to link capacity. These works consider the transmission capacity of each link, whose unit is Mbps. When we evaluate the amount of excessive traffic, congestion time needs to be considered in addition to the transmission capacity. Our work finds the amount of excessive traffic integrated over congestion time. This paper statically migrates flow paths relative to the related work [23]. The aim of the proposed model is to suppress the amount of excessive traffic by shifting probability distributions. In the related work [24], the link delay is considered by using a time-extended network. Continuous network update is addressed in the related works [25], [26], whereas our work addresses single network update. Our work does not address the link delay and continuous network update, and they are future work.

This paper compares the proposed model (the timeslot-based model) with the round-based model. This is because the purpose of the proposed model is to suppress the excessive traffic amount from the round-based model by the ability to flexibly shift the flow migration time. The differences between this work and some existing works related to the round-based model are summarized in Table I. In this work, each flow takes only the initial path and the final path to apply the two-phase commit. Also, we assume that flows are unsplittable since numerical evaluation is performed in a basic problem setting to evaluate the effectiveness of the timeslot-based model. (To extend the proposed model to splittable flows is future work.) Therefore, we choose the model in [10] as the benchmark model among the works which adopt the round-based model.

TABLE I: Comparison of works related to the round-based model.

	Round-based or timeslot-based	A flow takes only initial path and final path	Splittable flow or unsplittable flow
[8]	Round		Unsplittable
[9]	Round		Both
[10]	Round	✓	Unsplittable
[11]	Round	✓	Splittable
This work	Timeslot	✓	Unsplittable

III. MOTIVATION OF SHIFTING PROBABILITY DISTRIBUTIONS

This paper assumes the application of the two-phase commit in an SDN update problem. Each flow is migrated from the initial route to the final route without using another route. The two-phase commit is performed in the following steps [4]. Initially, all packets are tagged with “old tag” and all switches are subject to the rule matching the “old tag.” In a first step, the controller commands each switch to forward packets with “new tag” to the new path. Next, the controller commands each source switch to send the packet with the “new tag.” Finally, the controller commands each switch to delete the old path. Congestion occurs when the old path of one flow and the new path of another flow exist on a link at the same time, and the total amount of traffic exceeds the link capacity.

Fig. 1 shows an example of network update in which congestion may occur. There are ten switches $S1, S2, \dots, S10$ and two flows, namely Flow 1 and Flow 2. The capacity of each link is ten and the demand of each flow is six. The solid and dashed lines represent the initial route and the final route of flows, respectively. Since the final state of Flow 1 and the initial state of Flow 2 share link $(S4, S5)$, if Flow 1 is rerouted before Flow 2, congestion with the overload amount of two occurs at link $(S4, S5)$. By rerouting Flow 2 before Flow 1, congestion is avoided at link $(S4, S5)$.

In network updates in SDN, congestion may not be avoided only by considering the update order. In Fig. 1, if the controller orders a route change to Flow 2 before Flow 1, it is possible that the migration of Flow 1 is completed first. This is because there are some delays between the time at which the controller orders the switches to update the route and the time at which the route is updated. The delays include queuing delays [27] that occur when the controller issues a command and the switch receives it, and propagation delays between the controller and the switch. Whereas the propagation delay is a constant value, the queuing delay varies depending on the wait state of the migration command. Therefore, the time taken for each flow to migrate is indeterminate and varies according to some probability distribution. Since the migration time varies depending on flows, it is not possible to synchronize migration times between flows. The asynchronous nature of SDN due to these delays is an issue in updating an SDN.

In round-based model [8]–[11], the amount of packet loss due to congestion is minimized by performing flow migration over multiple rounds, rather than updating all flows at once. This model sets the duration of each round so that it ensures that the flow migration is completed without the affection of the asynchronous nature of SDN. Congestion is avoided as

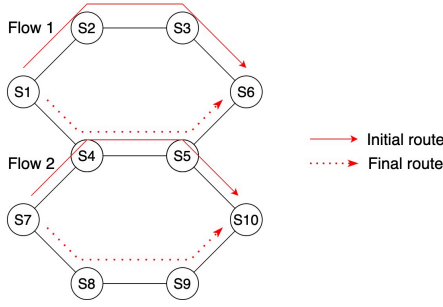


Fig. 1: Example of network in which congestion may occur.

much as possible by determining the time per round by based on only the maximum value of that time, and ordering reroute at the computed round. In the example in Fig. 1, Flow 2 is rerouted in round 1 and Flow 1 is rerouted in round 2 to avoid congestion. Since enough time is reserved for each round, migrations are always completed in the order commanded; the probability distributions of time required for flow migrations do not need to be considered.

It is uncertain in what order the updates are performed for multiple flow migrations that occur within the same round. The total amount of traffic exceeds the capacity with a certain probability, which can be derived from the probability distributions. For example, if the distribution of time required for flow migration is uniformly distributed and the update in Fig. 1 is performed in one round, The total amount of traffic exceeds the capacity with a probability of $\frac{1}{2}$.

In contrast, if the distribution of the time required for flow migration can be shifted without being bounded by rounds, it is possible to increase the probability that a flow in the final state (Flow 1) moves to the link after a flow in the initial state (Flow 2) moves to another link, which leads to a decrease in the amount of excessive traffic. Thus, this paper addresses the problem of determining scheduling of flow migrations that minimizes the amount of excessive traffic when the probability distribution of the time it takes for the flow migration is given.

IV. PROPOSED MODEL

In the proposed model, the time axis is divided into time slots that are fine-grained than rounds. In the round-based model, the migration operation of each flow is completed in one round, whereas in the proposed timeslot-based model, the migration operation is performed over multiple slots. This allows for flexible adjustment of time at which the controller orders the switches to reroute the flow. We compute the expected amount of excessive traffic derived from probability distributions of time required for flow migrations.

Fig. 2 shows an example of network in which there are three flows. Fig. 2a is the initial state of the network and Fig. 2b is the final state of the network. There are three flows, namely Flow 0, Flow 1, and Flow 2, and the demand of each flow is six. The capacity of each link is ten. The final route of Flow 0 shares the same links with the initial routes of Flow 1 and Flow 2. The final route of Flow 1 shares the same link with the initial route of Flow 2. If the flow migrations are performed

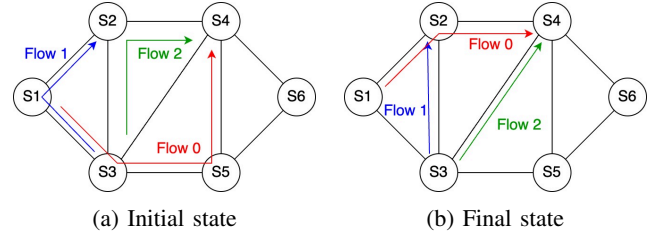
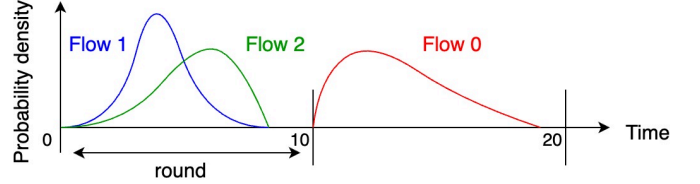
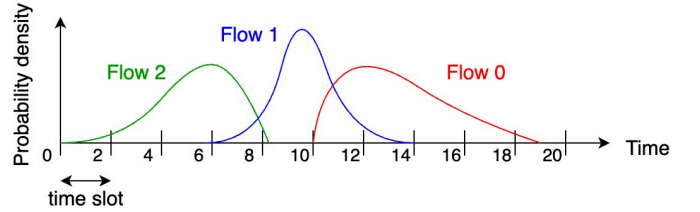


Fig. 2: Example of network in which there are three flows.



(a) Round-based model.



(b) Timeslot-based model.

Fig. 3: Shifting flow migration times.

in the order of Flow 2, Flow 1, and Flow 0, the congestion does not occur.

Fig. 3 shows an example of updating the network shown Fig. 2 using an existing model (the round-based model) and the proposed model (the timeslot-based model). The length of round is set to ten, and the length of time slot is set to two. The upper limit of time is set to 20, i.e., the number of rounds is set to two. The length of a unit time corresponds to real time and can be defined by the network provider. In the round-based model, Flow 1 and Flow 2 are assigned to the first round, and Flow 0 is assigned to the second round. In the timeslot-based model, each round is divided into several slots (five time slots in Fig. 3), and migration time of Flow 1 is shifted. As a result, the probability of migrations occurring in the order of Flow 2, Flow 1, and Flow 0 increases, and the expected amount of excessive traffic can be suppressed compared to the round-based model. The proposed model can shift a migration time to a time when migrations of other flows are less likely to occur.

Parameters used in the proposed model are defined as follows. We consider an SDN that is featured by directed graph $G = (V, E)$, where V is a set of switches and E is a set of links. $F = \{0, 1, \dots, |F| - 1\}$ denotes a set of flows in the network. D_f denotes a demand of flow $f \in F$. $P_f(t)$ denotes a probability density function of time required for migration of flow f . T denotes the upper limit of time at which the controller can command migrations. t_f denotes a time index of flow f , and $t = (t_0, \dots, t_{|F|-1})$ is

TABLE II: List of notations.

Parameters	Descriptions
V	Set of switches
E	Set of links
G	Directed network graph $G = (V, E)$
F	Set of flows. $F = \{0, 1, \dots, F - 1\}$
D_f	Demand of flow f
$P_f(t)$	Probability density function of time required for migration of flow f
T	upper limit of time at which controller can command migrations
t_f	Time index of flow f
$\mathbf{t}$	Vector of t_f . $\mathbf{t} = (t_0, \dots, t_{ F -1})$
$\mathcal{T}$	Set of t_f . $\mathcal{T} = \{0, \dots, T\}$
$\mathcal{T}$	Set of $\mathbf{t}$. $\mathcal{T} = \{\mathbf{t} \mid t_f \in \mathcal{T}, f \in F\}$
τ	Integer that indicates how many time slots one round is divided into
C_e	Capacity of link e
Variables	Descriptions
$x_{t_f}^f$	Binary decision variable that is set to one if the controller orders flow f to be rerouted at t_f , and zero otherwise
$l_{\mathbf{t}}^e$	Expected amount of excessive traffic in link e when controller orders flows to be rerouted at $\mathbf{t}$

a vector of t_f . $\mathcal{T} = \{0, \dots, T\}$ is a set of time indexes, and $\mathcal{T} = \{\mathbf{t} \mid t_f \in \mathcal{T}, f \in F\}$ is a set of $\mathbf{t}$. τ denotes an integer that indicates how many time slots one round is divided into. C_e denotes the capacity of link $e \in E$.

Decision variables used in the proposed model are defined as follows. $x_{t_f}^f$ is a binary variable that is set to one if the controller orders flow f to be rerouted at t_f , and zero otherwise. $l_{\mathbf{t}}^e$ is the expected value of the amount of excessive traffic in link e when the controller orders flows to be rerouted at $\mathbf{t}$.

The parameters and variables are summarized in Table II.

The proposed model aims to minimize the amount of excessive traffic. Since computation of the amount is nonlinear, the amount of $l_{\mathbf{t}}^e$ is computed for each $\mathbf{t} \in \mathcal{T}$ in advance. The way of computing $l_{\mathbf{t}}^e$ is described in Section VI. Then, a table storing $l_{\mathbf{t}}^e$ is made for each link $e \in E$. The expected amount of excessive traffic across the network is obtained by adding up the tables of each link. This paper calls the adding up table “sum amount table (SAT)” hereinafter. The $\mathbf{t}$ that has the smallest amount of excessive traffic is chosen among all rows of the SAT. If there are two or more $\mathbf{t}$ at which the amount of excessive traffic is minimum, $\mathbf{t}$ that the latest $t_f \in \mathbf{t}$ is earliest is chosen. Assume that the width of time slot is one in the network update shown in Fig. 1. Since congestion does not occur in the initial state, the amount of excessive traffic is the same between $\mathbf{t} = (1, 2)$ and $\mathbf{t} = (0, 1)$. This is because time relationship among flows at $\mathbf{t} = (0, 1)$ is the same as that at $\mathbf{t} = (1, 2)$; updating at $\mathbf{t} = (0, 1)$ is overall earlier than that at $\mathbf{t} = (1, 2)$. In this case, $\mathbf{t} = (0, 1)$ is chosen.

Table III shows an example of SAT in the network shown in Figs. 2 and 3. The left column is $\mathbf{t}$, which is a vector of command time of flow f (denoted by t_f). Since the time slot size is two, each element of $\mathbf{t}$ is a multiple of two. The right column is the sum of $l_{\mathbf{t}}^e$ for link e . The purpose of the proposed model is to choose the pattern of command times $\mathbf{t}$ which minimizes the excessive traffic amount from SAT.

A formulation of the proposed model is introduced below. For each link e , the amount of excessive traffic $l_{\mathbf{t}}^e$ is computed for each $\mathbf{t} \in \mathcal{T}$, and a table storing the amounts is made. The SAT is made by adding up these tables. The proposed model finds the smallest value among all rows of the SAT.

TABLE III: Example of SAT in network shown in Figs. 2 and 3

Command times $\mathbf{t} = (t_0, t_1, t_2)$	Sum of excessive traffic amount for all links
(0, 0, 0)	$\sum_{e \in E} l_{(0,0,0)}^e$
(0, 0, 2)	$\sum_{e \in E} l_{(0,0,2)}^e$
(0, 0, 4)	$\sum_{e \in E} l_{(0,0,4)}^e$
$\vdots$	$\vdots$
(20, 20, 20)	$\sum_{e \in E} l_{(20,20,20)}^e$

Note: calculated values are placed in the right column.

As mentioned before, $x_{t_f}^f$ is set to one when flow f is commanded migration at t_f . Since every flow is migrated only once during the network update, the sum of $x_{t_f}^f$ for each f is one. By multiplying each element in the SAT by $x_{t_f}^f$, the amount of excessive traffic at specific $\mathbf{t}$ can be taken from the SAT. Therefore, when there are three flows, the optimization problem that minimizes the expected amount of excessive traffic is formulated as follows:

$$\min \sum_{\mathbf{t} \in \mathcal{T}} \left(\prod_{f \in F} x_{t_f}^f \sum_{e \in E} l_{\mathbf{t}}^e \right), \quad (1a)$$

$$\text{s.t.} \quad \sum_{t_f \in \mathcal{T}} x_{t_f}^f = 1, \forall f \in F, \quad (1b)$$

$$x_{t_f}^f \in \{0, 1\}, \forall t_f \in \mathcal{T}, \forall f \in F. \quad (1c)$$

This optimization problem (1a)–(1c) is nonlinear since it includes multiplication of decision variables. This problem is linearized by replacing $\prod_{f \in F} x_{t_f}^f$ with $z_{\mathbf{t}}$. We also add a term to the objective function to get the earliest $\mathbf{t}$. α should be a sufficiently small number relative to the amount of excessive traffic.

$$\min \sum_{\mathbf{t} \in \mathcal{T}} z_{\mathbf{t}} \left\{ \left(\sum_{e \in E} l_{\mathbf{t}}^e \right) + \alpha \times \max(\mathbf{t}) \right\}, \quad (2a)$$

$$\text{s.t.} \quad \sum_{t_f \in \mathcal{T}} x_{t_f}^f = 1, \forall f \in F, \quad (2b)$$

$$x_{t_f}^f \geq z_{\mathbf{t}}, \forall \mathbf{t} \in \mathcal{T}, \forall f \in F, \quad (2c)$$

$$z_{\mathbf{t}} \geq \sum_{f \in F} x_{t_f}^f - |F| + 1, \forall \mathbf{t} \in \mathcal{T}, \quad (2d)$$

$$x_{t_f}^f \in \{0, 1\}, \forall t_f \in \mathcal{T}, \forall f \in F, \quad (2e)$$

$$z_{\mathbf{t}} \in \{0, 1\}, \forall \mathbf{t} \in \mathcal{T}. \quad (2f)$$

The scheme that directly solves the optimization problem is called OPT.

V. TWO-STAGE SCHEDULING SCHEME

In this section, a scheme that suppresses the computation time for solving the optimization problem is introduced. As described in Section IV, the optimization problem (2a)–(2f) for the proposed model is solved after the SAT is made. The amount of excessive traffic is calculated at each $\mathbf{t} \in \mathcal{T}$ to make the SAT. To suppress the number of calculation times, we introduce a two-stage scheduling scheme (2SS).

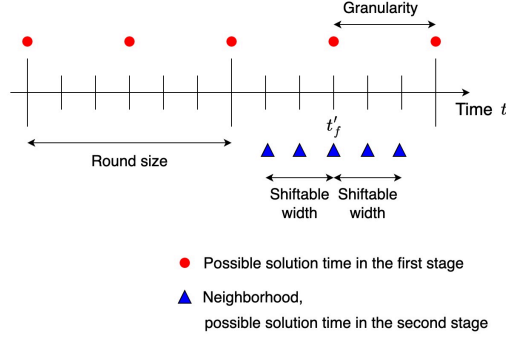


Fig. 4: Example of granularity and shiftable width in two-stage scheduling scheme (2SS).

The process of solving the optimization problem is divided into two stages in 2SS. The computation in each stage is described as follows. In the first stage, the optimization problem is solved under the condition that elements of $\mathcal{T}$ are restricted to multiples of n . We call the value of n *granularity* hereinafter. By solving the optimization problem in the first stage, the time point at which the amount of excessive traffic is the minimum among times that are multiples of n is found. Let t'_f denote the obtained time point. In the second stage, the optimization problem is solved under the condition that elements of $\mathcal{T}$ are restricted to around of t'_f . We define *neighborhood* as set of time point, whose element has difference of not more than w with t'_f . The difference w is called *shiftable width* hereinafter. By solving the optimization problem in the second stage, the time point at which the amount of excessive traffic is not more than that at t'_f is found in the neighborhood.

Fig. 4 shows how to determine time point at which controller can command migrations in 2SS. The granularity is three and the shiftable width is two. Red circle points and blue triangle points are possible solution times in the first stage and the second stage, respectively.

The computation complexities of OPT and 2SS are compared in the following. If we compute the amount of excessive traffic at each t , the computation complexity in OPT is $\mathcal{O}(|\mathcal{T}|^{|F|})$. However, when all elements of t are positive, the results of previous calculations can be used. For example, the amount of excessive traffic at $t = (t_0, t_1, \dots, t_{|F|})$ is computed by using that at $t' = (t_0 - \min(t), t_1 - \min(t), \dots, t_{|F|} - \min(t))$. Therefore, the computation complexity of OPT is $\mathcal{O}(|\mathcal{T}|^{|F|}) - \mathcal{O}((|\mathcal{T}| - 1)^{|F|})$, thus $\mathcal{O}(|\mathcal{T}|^{|F|-1})$.

The number of calculation times per one flow in 2SS is described as follows when we compute the amount of excessive traffic at each t . Let $|\mathcal{T}|$ denote the number of time slots. The number of calculation times in the first stage is $\frac{|\mathcal{T}|}{n} + 1$. When the shiftable width is set to n , the number of calculation times in the second stage is $2n + 1$. Therefore, if we compute the amount of excessive traffic at each t , the overall number of calculation times, $f(n)$, is expressed by the following equation.

$$\begin{aligned} f(n) &= \left(\frac{|\mathcal{T}|}{n} + 1 \right)^{|F|} + (2n + 1)^{|F|} \\ &= \sum_{k=0}^{|F|} |F| C_k \left(\left(\frac{|\mathcal{T}|}{n} \right)^k + (2n)^k \right) \\ &\geq 2 \sum_{k=0}^{|F|} |F| C_k (2|\mathcal{T}|)^{\frac{k}{2}}. \end{aligned} \quad (3)$$

The condition for equality is $n = \sqrt{\frac{|\mathcal{T}|}{2}}$. This value is rounded to an integer n' as follows:

$$n' = \arg \min_n (f(\lfloor n \rfloor), f(\lceil n \rceil)). \quad (4)$$

Computing on the setting where granularity is n' requires the least number of computation times. In this case, the computation complexity of 2SS is $\mathcal{O}(|\mathcal{T}|^{\frac{|F|}{2}})$. Since we use the amount of previous calculations when all elements of t are positive, the computation complexity of 2SS is $\mathcal{O}(|\mathcal{T}|^{\frac{|F|-1}{2}})$.

VI. COMPUTATION METHODS OF AMOUNT OF EXCESSIVE TRAFFIC

Computation methods of the expected amount of excessive traffic, l_t^e , are presented.

For simplicity, consider an example in which four flows, namely Flow 0, Flow 1, Flow 2, and Flow 3, cause congestion. The demand of each flow is one and the capacity of each link is two. In the initial state, Flow 0 and Flow 1 share a link. The initial amount of traffic in this link is two. In the final state, Flow 0 and Flow 1 have been migrated to another link, and Flow 2 and Flow 3 have been migrated to this link. Let m_f be time point at which the route of flow f is migrated.

There are $4! = 24$ possible orders in which flows are migrated. For example, assume that flows are migrated in order (Flow 2, Flow 3, Flow 0, Flow 1). Flow 2 and Flow 3 are migrated at time m_2 and m_3 , respectively. Therefore, the overload amount of one occurs in the link in time interval $[m_2, m_3]$. When Flow 3 is migrated to the link, the amount of traffic increases by one and the overload amount in the link is two in time interval $[m_3, m_0]$. When Flow 0 is migrated to another link, since the amount of traffic decreases by one, the overload amount in the link is one in time interval $[m_0, m_1]$.

The expected amount of excessive traffic can be computed from the expected time of each overload. Specifically, if flows are migrated in order (2, 3, 0, 1), the expected amount of excessive traffic is calculated by the following equation:

$$1 \times \mathbb{E}[m_3 - m_2] + 2 \times \mathbb{E}[m_0 - m_3] + 1 \times \mathbb{E}[m_1 - m_0], \quad (5)$$

where $\mathbb{E}[m_{f'} - m_f]$ denotes the expected value of time interval between $m_{f'}$ and m_f .

Similar calculations are performed for all orders of flows, and the results are added up to obtain the expected amount of excessive traffic when the four flows cause congestion. In calculating (5), the expected value of time of each overload (e.g. $\mathbb{E}[m_1 - m_0]$) needs to be computed.

Algorithm 1 Calculation of I_e , index_list and $D'_{e,f}$

Input: $F_e^{\text{before}}, F_e^{\text{after}}, D_f$
Output: $I_e, \text{index_list}, D'_{e,f}$

```

1:  $I_e \leftarrow 0$ 
2: for each  $f$  in  $F_e^{\text{before}}$  do
3:    $I_e \leftarrow I_e + D_f$ 
4: end for
5:  $F_e^{\text{all}} \leftarrow F_e^{\text{before}} \cup F_e^{\text{after}}$ 
6: index_list  $\leftarrow$  All orders of  $F_e^{\text{all}}$ 
7:  $F_e^{\text{const}} \leftarrow F_e^{\text{before}} \cap F_e^{\text{after}}$ 
8: for each  $f$  in  $F_e^{\text{const}}$  do
9:    $D'_{e,f} \leftarrow 0$ 
10: end for
11:  $F_e^{\text{before}} \leftarrow F_e^{\text{before}} - F_e^{\text{const}}$ 
12: for each  $f$  in  $F_e^{\text{before}}$  do
13:    $D'_{e,f} \leftarrow -D_f$ 
14: end for
15:  $F_e^{\text{after}} \leftarrow F_e^{\text{after}} - F_e^{\text{const}}$ 
16: for each  $f$  in  $F_e^{\text{after}}$  do
17:    $D'_{e,f} \leftarrow D_f$ 
18: end for
```

Based on this example, computing the expected amount of excessive traffic is generalized as follows. All orders of flow migrations causing congestion in link e are enumerated. For each order, the differences between adjacent migration times are calculated, multiplied by the corresponding overload amount, and added up. The sum of the results is the expected amount of excessive traffic in link e .

Some parameters are required to compute the expected amount of excessive traffic in link e . The procedure of obtaining the parameters is shown in Algorithm 1. In advance, we get a set of flows $F_e^{\text{before}} \subset F$, which are migrated from link e , and a set of flows $F_e^{\text{after}} \subset F$, which are migrated into link e . By using F_e^{before} , F_e^{after} , and D_f ($\forall f \in (F_e^{\text{before}} \cup F_e^{\text{after}})$), the required parameters, I_e , index_list, and $D'_{e,f}$, are computed. Here, I_e represents the amount of traffic in link e in the initial state. index_list represents a set of all orders of flow migrations. $D'_{e,f}$ represents a signed demand, which is defined as follows. When flow f is migrated from link e to another link, $D'_{e,f}$ is $-D_f$. When flow f continues to pass link e in both initial route and final route, $D'_{e,f}$ is zero. When flow f is migrated from another link to link e , $D'_{e,f}$ is $+D_f$.

By using the output of Algorithm 1, we compute the amount of excessive traffic. This paper introduces two methods to compute the expected value of the time. The first method, which obtains the theoretical amount of excessive traffic by integrating the probability density functions, is described in Section VI-A. The second method, which approximates the amount of excessive traffic by iteratively generating random numbers, is described in Section VI-B.

A. Computation of expected amount of excessive traffic from probability density function (Theoretical value)

In this section, a method, which computes the expected amount of excessive traffic, is introduced. The amount of excessive traffic is derived from the probability density function of time from when the controller commands flow migration to when the flow is rerouted to a new path. As mentioned before, to compute the expected amount of excessive traffic,

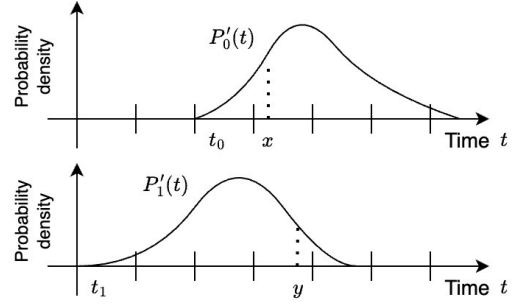


Fig. 5: Time relationship of probability density functions when there are two flows.

the expected value of overloaded time needs to be computed. The expected value of overloaded time is computed from a probability density function of time point at which each flow is migrated. The probability density function is obtained by translating $P_f(t)$ by time point at which flow f is commanded to migrate. Specifically, the probability density function of flow f is $P_f(t - t_f)$ when flow f is commanded to be migrated at t_f . This translated function of flow f is defined as $P'_f(t)$. The expected value of the time point of flow f which is commanded to be migrated at t_f is computed by the following equation:

$$\int_{t_f}^{\infty} t P_f(t - t_f) dt = \int_0^{\infty} t P'_f(t) dt. \quad (6)$$

The expected time of each overload is computed as follows.

For simplicity, consider the case where there are two flows, namely Flow 0 and Flow 1, in the same link. Time relationship of the probability density functions when there are two flows is shown in Fig. 5. x and y correspond to the migration times for Flow 0 and Flow 1, respectively. $E[m_1 - m_0]$ is computed by (7). Time $y - x$ is multiplied by each probability density function $P'_f(t)$ and is integrated. Note that there is an order constraint here; in order to calculate $m_1 - m_0$, Flow 0 must be migrated before Flow 1 is migrated. Therefore, the integral interval of x is $[0, y]$ and that of y is $[0, \infty]$.

$$E[m_1 - m_0] = \int_0^{\infty} \left(\int_0^y (y - x) P'_0(x) dx \right) P'_1(y) dy. \quad (7)$$

From here, as in the beginning part of Section VI, the case where four flows cause congestion is considered. Assume that an order of flow migrations is (Flow 2, Flow 3, Flow 0, Flow 1).

When computing $E[m_1 - m_0]$, it is necessary to take into account an order of not only migrations of Flow 0 and Flow 1 but also other flow migrations. Migration of Flow 3 is earlier than migration of Flow 0, and migration of Flow 2 is earlier than migration of Flow 3. Therefore, at first, the probability density function of Flow 2 is integrated from zero to time at which Flow 3 is migrated, and it is multiplied by the probability density function of Flow 3. Then, the result is integrated from zero to time at which Flow 0 is migrated, and

it is multiplied by the probability density function of Flow 0. Finally, an integral computation similar to (7) is performed.

$$\begin{aligned} E[m_1 - m_0] &= \int_0^\infty \left(\int_0^y (y - x) \cdot P'_0(x) \right. \\ &\quad \left. \left\{ \int_0^x P'_3(w) \left(\int_0^w P'_2(z) dz \right) dw \right\} dx \right) \cdot P'_1(y) dy. \end{aligned} \quad (8)$$

When computing $E[m_0 - m_3]$, it is necessary to take into account an order of migrations of Flow 0 and Flow 3 in addition to other flow migrations. Migration of Flow 2 is performed before Flow 3 is migrated, and migration of Flow 1 is performed after Flow 0 is migrated. Therefore, the probability density function of Flow 2 is integrated from zero to time at which Flow 3 is migrated, and it is multiplied by the probability density function of Flow 3. The probability density function of Flow 1 is also integrated from time at which Flow 0 is migrated to infinity, and it is multiplied by the probability density function of Flow 0. $E[m_0 - m_3]$ is calculated in the following equation:

$$\begin{aligned} E[m_0 - m_3] &= \int_0^\infty \left\{ \int_0^x (x - w) \cdot P'_3(w) \left(\int_0^w P'_2(z) dz \right) dw \right\} \\ &\quad \cdot P'_0(x) \left(\int_x^\infty P'_1(y) dy \right) dx. \end{aligned} \quad (9)$$

Based on the above calculations of $E[m_1 - m_0]$ and $E[m_0 - m_3]$, the expected value of the difference between the i th flow migration time and the $(i+1)$ th flow migration time is derived.

First, consider probabilities of migrations before the i th migration. This process iteratively integrates the probability density function of a flow from zero to time at which the next flow is migrated, namely t_{next} , and multiplies by the probability density function of the next migrated flow. Specifically, a variable A , whose initial value is one, is multiplied by the probability density function, and integrated from zero to t_{next} . The result is again stored in A . This procedure is iterated in the order of flows starting with the oldest migration.

Second, consider probabilities of migrations after the $(i+1)$ th migration. This process iteratively integrates the probability density function of a flow from time at which the previous flow is migrated to infinity, and multiplies by the probability density function of the previous migrated flow. Specifically, a variable B , whose initial value is one, is multiplied by the probability density function, and integrated from t to infinity. The result is again stored in B . This procedure is iterated in the order of flows starting with the newest migration.

From A and B , the expected value of overloaded time is computed by the following equation:

$$\begin{aligned} E[m_{\text{index}[i+1]} - m_{\text{index}[i]}] &= \int_0^\infty \left(\int_0^y (y - x) \times \left(P'_{\text{index}[i]}(x) \times A \right) dx \right) \\ &\quad \times \left(P'_{\text{index}[i+1]}(y) \times B \right) dy, \end{aligned} \quad (10)$$

where $\text{index}[i]$ represents the i th flow.

There is a possibility that congestion has already occurred in the initial state. In this case, time interval between 0 and time at which the first flow is migrated also needs to be considered. Since the first migrated flow is being considered, there are no flows to be migrated before the flow. Therefore, by using only B , the expected value of overloaded time is computed by the following equation:

$$E[m_{\text{index}[0]}] = \int_0^\infty y \times \left(P'_{\text{index}[0]}(y) \times B \right) dy. \quad (11)$$

The expected amount of excessive traffic can be computed from the overloaded time obtained in (10) and (11). The procedure of computation of the excessive traffic amount is shown in Algorithm 2. The amount of traffic in link e is denoted by J_e . In the initial state, the value of J_e is the same as that of I_e . We extract a flow order, index , one by one from index_list , in which all orders of flow migrations are stored. In the extracted flow order, at first, we check whether excessive traffic exists in the initial state, i.e., whether $J_e > C_e$. If excessive traffic exists, the expected amount of excessive traffic in the initial state is computed by multiplying the overload amount $J_e - C_e$ by the overloaded time obtained in (11). The result is added to l_t^e .

After excessive traffic in the initial state is checked, $D'_{e,f}$ is added to J_e in the order specified by index . The expected value of overloaded time is computed by (10) at each time at which J_e exceeds link capacity C_e . The expected amount of excessive traffic until the next flow is migrated is computed by multiplying the overload amount $J_e - C_e$ by this overloaded time. The result is added to l_t^e . This procedure is iterated for all index in index_list . The obtained l_t^e is the expected amount of excessive traffic at t in link e .

B. Approximation of expected amount of excessive traffic by generating random numbers

1) *Approximation algorithm:* It can take time to compute the expected amount of excessive traffic by using the method in Section VI-A. To suppress the computation time, a method which approximates the amount of excessive traffic by generating random numbers is introduced. The procedure of approximation of the excessive traffic amount is shown in Algorithm 3. As in Section VI-A, a flow order, index , is extracted one by one from index_list . We check whether the amount of traffic in the initial state exceeds the capacity of link e and whether J_e exceeds the capacity of link e by adding $D'_{e,f}$ to J_e . When the link capacity is exceeded, random numbers are generated for times required for flow migrations, which are contained in index , by function "GETDELAY." The random numbers are generated according to the probability density functions. The procedure of function "GETDELAY" is presented in Section VI-B2. Time point at which flow f is migrated is obtained by adding time point, at which the controller commands the switches to migrate flow f , to time that it takes to migrate flow f . Specifically, m_f ($\forall f \in (F_e^{\text{before}} \cup F_e^{\text{after}})$)

Algorithm 2 Computation of expected amount of excessive traffic

Input: I_e , index_list, F , $D'_{e,f}$, P_f , t_f , C_e
Output: expected value of loss l_t^e

```

1: for  $f$  in  $F$  do
2:    $P'_f(t) \leftarrow P_f(t - t_f)$ 
3: end for
4:  $l_t^e \leftarrow 0$ 
5: for each index in index_list do
6:    $J_e \leftarrow I_e$ 
7:   for  $i = -1$  to (length of index)-2 do
8:     if  $i \neq -1$  then
9:        $J_e \leftarrow J_e + D'_{e,f}$ 
10:    end if
11:    if  $J_e > C_e$  then
12:       $A \leftarrow 1$ 
13:      if  $i \geq 1$  then
14:        for each  $f$  in (index[0], index[1], ..., index[i-1]) do
15:           $A \leftarrow \int_0^t A \times P'_f(t) dt$ 
16:        end for
17:      end if
18:       $B \leftarrow 1$ 
19:      if  $i \leq (\text{length of index}) - 3$  then
20:        for each  $f$  in (index[(length of index)-1], index[(length of index)-2], ..., index[i+2]) do
21:           $B \leftarrow \int_t^\infty B \times P'_f(t) dt$ 
22:        end for
23:      end if
24:      if  $i = -1$  then
25:         $l_t^e \leftarrow l_t^e + (J_e - C_e) \times \int_0^\infty y \times (P'_{\text{index}[0]}(y) \times B) dy$ 
26:      else
27:         $l_t^e \leftarrow l_t^e + (J_e - C_e) \times \int_0^\infty \left( \int_0^y (y-x) \times (P'_{\text{index}[i]}(x) \times A) dx \right) \times (P'_{\text{index}[i+1]}(y) \times B) dy$ 
28:      end if
29:    end if
30:  end for
31: end for

```

is computed by adding t_f to the generated random number. When the large/small relationship of m_f matches the flow order, index, time intervals at which an overload occurs are computed by taking difference of m_f . The process from the random number generation to the difference calculation is iterated a certain number of times. The expected value of overloaded time is computed by taking the average of the differences. The expected amount of excessive traffic is calculated by multiplying the overload amount, and the result is added to l_t^e .

2) *Generation of random numbers according to probability density functions:* “GETDELAY” in Algorithm 3 is a function which generates random numbers according to arbitrary probability density functions. The procedure of generating random numbers is shown in Algorithm 4.

Variable U which follows a uniform distribution below is considered.

$$P(U \leq u) = u, 0 \leq u \leq 1. \quad (12)$$

Also, let $F(x)$ be a certain cumulative distribution function and consider a variable $X = F^{-1}(U)$. For the cumulative distribution function of X , the following equation holds:

$$\begin{aligned} P(X \leq x) &= P(F^{-1}(U) \leq x) \\ &= P(U \leq F(x)) \end{aligned}$$

Algorithm 3 Approximation of expected amount of excessive traffic

Input: I_e , index_list, F , $D'_{e,f}$, P_f , t_f , C_e
Output: expected value of loss l_t^e

```

1:  $l_t^e \leftarrow 0$ 
2: for each index in index_list do
3:    $J_e \leftarrow I_e$ 
4:   for  $i = -1$  to (length of index)-2 do
5:     if  $i \neq -1$  then
6:        $J_e \leftarrow J_e + D'_{e,f}$ 
7:     end if
8:     if  $J_e > C_e$  then
9:        $sum \leftarrow 0$ 
10:      for  $k = 1$  to 5000 do
11:        for each  $f$  in  $F$  do
12:           $x_f \leftarrow t_f + \text{GETDELAY}(f)$ 
13:        end for
14:        if  $x_{\text{index}[0]} < x_{\text{index}[1]} < \dots < x_{\text{index}[(\text{length of index})-1]}$  then
15:          if  $i = -1$  then
16:             $sum \leftarrow sum + x_{\text{index}[0]}$ 
17:          else
18:             $sum \leftarrow sum + (x_{\text{index}[i+1]} - x_{\text{index}[i]})$ 
19:          end if
20:        end if
21:      end for
22:       $l_t^e \leftarrow l_t^e + (J_e - C_e) \times (sum/5000)$ 
23:    end if
24:  end for
25: end for

```

$$= F(x). \quad (\because (12)) \quad (13)$$

From (13), it is observed that the cumulative distribution function of X , which is obtained by substituting U for the inverse function of $F(x)$, is $F(x)$. Therefore, a random number according to a certain probability density function can be generated by substituting a uniform distribution for the inverse function of the cumulative distribution function, which is generated from the probability density function.

By function “CDF_FROM_PDF” in Algorithm 4, a probability density function is integrated and added together for each microinterval. Each time adding the result of integration, the result is plotted and general shape of a cumulative distribution function is obtained. The general shape is approximated to linear function $Q_f(t)$.

By function “RANDOM_FROM_CDF” in Algorithm 4, random number y is generated from the uniform distribution, and $Q_f(t) = y$ is solved for t . The result t is the random number according to the probability density function.

3) *Comparison of theoretical values and approximated values:* In Section VI-B, the approximating method of the expected amount of excessive traffic is presented to alleviate time required for computing the theoretical value of that. This section examines whether the results obtained by the approximation are credible values. To evaluate the accuracy of the approximation that generates random numbers, the number of iterations that the random number is generated is changed, and the percentage of error from the theoretical values is calculated.

The network model in Fig. 1 is used for this evaluation. The capacity of each link is set to ten. The demand of each flow is set to six. T , the upper limit of time point at which the controller can command migration, is set to six. The width of time slot is set to one. The probability distribution of time

Algorithm 4 GETDELAY

```

1: function CDF_FROM_PDF( $P_f$ )
2:    $cdf \leftarrow [0]$ 
3:    $dt \leftarrow [0]$ 
4:   while  $cdf[-1] < 1$  do
5:      $append(cdf[-1] + \int_{dt[-1]}^{dt[-1]+0.1} P_f(t) dt)$  to  $cdf$ 
6:      $append(dt[-1] + 0.1)$  to  $dt$ 
7:   end while
8:    $Q_f(t) \leftarrow \text{approximate}(dt, cdf)$ 
9:   return  $Q_f(t), dt[-1]$ 
10: end function
11: function RANDOM_FROM_CDF( $Q_f(t), t_{\max}$ )
12:    $y \leftarrow$  random number between 0 and 1
13:    $solve Q_f(t) = y$  for  $t$  ( $0 \leq t \leq t_{\max}$ )
14:   return  $t$ 
15: end function
16: function GETDELAY( $f$ )
17:    $Q_f(t), t_{\max} \leftarrow$  CDF_FROM_PDF( $P_f$ )
18:    $t \leftarrow$  RANDOM_FROM_CDF( $Q_f(t), t_{\max}$ )
19:   return  $t$ 
20: end function

```

TABLE IV: Error average ratio in approximation relative to theoretical value.

Number of iterations	Trials				
	1st	2nd	3rd	4th	5th
500	6.538%	5.335%	6.595%	8.556%	6.936%
1000	4.059%	5.061%	5.733%	5.514%	5.289%
1500	4.162%	3.795%	3.692%	4.364%	4.386%
3000	3.216%	2.506%	2.813%	3.505%	2.453%
5000	2.292%	1.834%	1.764%	1.952%	1.408%
10000	1.254%	1.539%	1.853%	1.585%	1.061%

required for each flow migration is set to a uniform distribution whose range is $[0, 4]$.

Table IV shows the error average ratio of values in the approximated SAT relative to those in the theoretical SAT examined five times. The ratio is obtained by calculating an error ratio between the theoretical and approximated values for each row of the SATs and averaging them among the SATs. When the number of iterations is 1500, the error average ratio is at most 4.5%.

VII. NUMERICAL EVALUATION

This section compares the excessive traffic ratio in the proposed model with that in the benchmark model, which adopts the round-based model. In the proposed model, two schemes, OPT and 2SS, are evaluated. The evaluation metric is the ratio of excessive traffic amount, which is the ratio of the total amount of excessive traffic to the total amount of traffic across update duration. The optimization problem is solved with CPLEX Interactive Optimizer 22.1.0.0 [28] on a computer equipped with Apple M1 chip and 16 GB of RAM.

A. Settings of flow paths

This experiment is performed in scenarios where deadlocks occur. Deadlock is a state that congestion is unavoidable no matter how the controller adjusts the time to send the migration commands. When two or more flows exchange links to pass through in the initial and final states, deadlocks occur in the links.

We introduce how to create scenarios where deadlocks occur. At first, the source node (hereinafter v_s^f) and the

destination node (hereinafter v_d^f) are randomly chosen for each flow. Next, paths of flows in the initial and final states are decided by solving the following optimization problem.

Decision variables used in the optimization problem are defined as follows. $a_{i,j}^f$ is a binary variable that is set to one if flow f passes through link (i, j) in the initial state, and zero otherwise. $b_{i,j}^f$ is a binary variable that is set to one if flow f passes through link (i, j) in the final state, and zero otherwise. p_i^f is a positive integer variable that represents ordering of nodes which flow f passes through in the initial state. q_i^f is a positive integer variable that represents ordering of nodes which flow f passes through in the final state.

Let us assume that three flows, namely Flow 0, Flow 1, and Flow 2, cause one type of deadlock. The optimization problem that determines the initial and final states with a deadlock is formulated as follows:

$$\min \sum_{f \in F} \sum_{(i,j) \in E} (a_{i,j}^f + b_{i,j}^f), \quad (14a)$$

$$\text{s.t.} \quad \sum_{j:(i,j) \in E} a_{i,j}^f - \sum_{j:(j,i) \in E} a_{j,i}^f = 0, \forall f \in F, \quad \forall i \in V \setminus \{v_s^f, v_d^f\}, \quad (14b)$$

$$\sum_{j:(i,j) \in E} a_{i,j}^f - \sum_{j:(j,i) \in E} a_{j,i}^f = 1, \forall f \in F, i = v_s^f, \quad (14c)$$

$$\sum_{j:(i,j) \in E} a_{i,j}^f - \sum_{j:(j,i) \in E} a_{j,i}^f = -1, \forall f \in F, i = v_d^f, \quad (14d)$$

$$\sum_{j:(i,j) \in E} a_{i,j}^f \leq 1, \forall f \in F, \forall i \in V, \quad (14e)$$

$$\sum_{j:(j,i) \in E} a_{j,i}^f \leq 1, \forall f \in F, \forall i \in V, \quad (14f)$$

$$p_i^f - p_j^f + |V| \times a_{i,j}^f \leq |V| - 1, \forall f \in F, \forall (i, j) \in E, \quad (14g)$$

$$\sum_{j:(i,j) \in E} b_{i,j}^f - \sum_{j:(j,i) \in E} b_{j,i}^f = 0, \forall f \in F, \quad \forall i \in V \setminus \{v_s^f, v_d^f\}, \quad (14h)$$

$$\sum_{j:(i,j) \in E} b_{i,j}^f - \sum_{j:(j,i) \in E} b_{j,i}^f = 1, \forall f \in F, i = v_s^f, \quad (14i)$$

$$\sum_{j:(i,j) \in E} b_{i,j}^f - \sum_{j:(j,i) \in E} b_{j,i}^f = -1, \forall f \in F, i = v_d^f, \quad (14j)$$

$$\sum_{j:(i,j) \in E} b_{i,j}^f \leq 1, \forall f \in F, \forall i \in V, \quad (14k)$$

$$\sum_{j:(j,i) \in E} b_{j,i}^f \leq 1, \forall f \in F, \forall i \in V, \quad (14l)$$

$$q_i^f - q_j^f + |V| \times b_{i,j}^f \leq |V| - 1, \forall f \in F, \forall (i, j) \in E, \quad (14m)$$

$$\sum_{f \in F} D_f a_{i,j}^f \leq C_{(i,j)}, \forall (i, j) \in E, \quad (14n)$$

$$\sum_{f \in F} D_f b_{i,j}^f \leq C_{(i,j)}, \forall (i, j) \in E, \quad (14o)$$

$$\sum_{(i,j) \in E} (a_{i,j}^f + b_{i,j}^f - 2a_{i,j}^f b_{i,j}^f) \geq 1, \forall f \in F, \quad (14p)$$

$$\sum_{(i,j) \in E} a_{i,j}^0 a_{i,j}^1 b_{i,j}^2 \geq 1, \quad (14q)$$

$$\sum_{(i,j) \in E} b_{i,j}^0 b_{i,j}^1 a_{i,j}^2 \geq 1, \quad (14r)$$

$$a_{i,j}^f \in \{0, 1\}, \forall f \in F, \forall (i, j) \in E, \quad (14s)$$

$$b_{i,j}^f \in \{0, 1\}, \forall f \in F, \forall (i, j) \in E, \quad (14t)$$

$$p_i^f \in \{1, 2, \dots, |V|\}, \forall f \in F, \forall i \in V, \quad (14u)$$

$$q_i^f \in \{1, 2, \dots, |V|\}, \forall f \in F, \forall i \in V. \quad (14v)$$

The objective function (14a) prevents flow paths from becoming unnecessarily long. Since flow is unsplittable, we define the following five constraints in the initial state. Equation (14b) represents that the number of egress and that of ingress are the same at each node except source and destination. Equation (14c) represents that the number of egress is one and that of ingress is zero about flow f in the source node. Equation (14d) represents that the number of egress is zero and that of ingress is one about flow f in the destination node. Equations (14e) and (14f) represent that flow is unsplittable. Under these constraints (14b)–(14f), a sub loop that does not contain the source and destination nodes may be generated. Equation (14g) prevents a path from generating a sub loop. Equations (14h)–(14m) are flow constraints in the final state, which correspond to (14b)–(14g). Equations (14n) and (14o) are link capacity constraints. Equation (14p) represents that paths in the initial state and those in the final state are different from each other. Equations (14q) and (14r) cause a deadlock of Flow 0, Flow 1, and Flow 2.

These two constraints (14q) and (14r) are examples of deadlock constraints. In our experiment, flows which cause deadlocks are randomly determined and constraint equations are formulated by the following procedure. First, the number of deadlocks and flow combinations which can cause each deadlock are randomly generated. Second, each flow combination is shuffled and divided into two parts. Finally, the deadlock constraints are added. Equations (14q) and (14r) mean that the former part contains Flows 0 and 1, and the latter part contains Flow 2. As another example, consider that the former part contains Flows 0 and 2, and the latter part contains Flows 1 and 3. In this case, (14q) and (14r) are replaced with the following equations:

$$\sum_{(i,j) \in E} a_{i,j}^0 a_{i,j}^2 b_{i,j}^1 b_{i,j}^3 \geq 1, \quad (15a)$$

$$\sum_{(i,j) \in E} b_{i,j}^0 b_{i,j}^2 a_{i,j}^1 a_{i,j}^3 \geq 1. \quad (15b)$$

These procedure is iteratively implemented for the number of deadlocks.

B. Benchmark model

The benchmark model corresponding to the work [10] can be realized as a specific case ($\tau = 1$) of the proposed model.

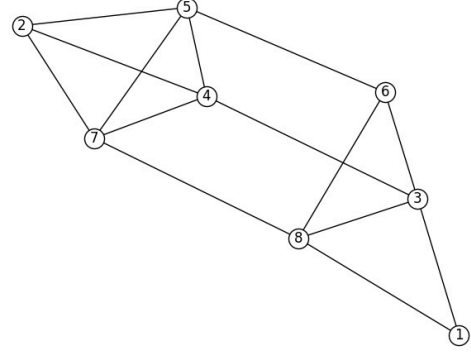


Fig. 6: Microsoft's inter-datacenter WAN [8], [9], [23].

TABLE V: Paths and demands of flows.

Flow	Initial path	Final path
F_0	$1 \rightarrow 3 \rightarrow 4 \rightarrow 2$	$1 \rightarrow 8 \rightarrow 7 \rightarrow 2$
F_1	$1 \rightarrow 3 \rightarrow 4 \rightarrow 5$	$1 \rightarrow 8 \rightarrow 7 \rightarrow 5$
F_2	$1 \rightarrow 8 \rightarrow 6$	$1 \rightarrow 3 \rightarrow 6$
F_3	$3 \rightarrow 8 \rightarrow 7$	$3 \rightarrow 4 \rightarrow 7$
F_4	$8 \rightarrow 7 \rightarrow 4$	$8 \rightarrow 3 \rightarrow 4$

While the flow migration can be commanded at the beginning of rounds in the benchmark model, the migration can be commanded at every time slot split from round in the proposed model. For example, when the length of round is eight and the number of round divisions (denoted by τ) is four, the length of time slot is two. The command time is chosen from $0, 8, 16, \dots, 8n$ in the benchmark model, and it is chosen from $0, 2, 4, 6, 8, 10, \dots, 8n$ in the proposed model. The proposed model with $\tau = 1$ is equivalent to the benchmark model.

C. Result in Microsoft's inter-datacenter WAN with given flow paths

In this section, we compare the performances of the benchmark model and the proposed model in Microsoft's inter-datacenter WAN which is used in related works [8], [9], [23]. Its topology is shown in Fig. 6. Table V shows paths and demands of flows used in related work [8]. The probability density functions of Flows 0, 1, $\dots$, 4 are given by (16a)–(16e) shown in Fig. 7, respectively. Since integrating each probability density function from 0 to 8 yields 1, the round size is set to 8. The capacity of each link is set to 60 Mb/s. The demand of each flow is set to 25 Mb/s. T , the upper limit of time point when the controller can command flows to be migrated, is set to 16, i.e., the number of rounds is set to three. The granularity and the shiftable width when $\tau = 3$ and 4 are set to two. The granularity and the shiftable width when $\tau = 6, 9$, and 12 are set to three.

$$P_0(t) = \frac{1}{\sqrt{2\pi \times 0.25}} \times \frac{1}{t} \times \exp\left(-\frac{(\log t)^2}{2 \times 0.25}\right), \quad (16a)$$

$$P_1(t) = \frac{1}{\sqrt{2\pi \times 0.125}} \times \frac{1}{t} \times \exp\left(-\frac{(\log t - 0.8)^2}{2 \times 0.125}\right), \quad (16b)$$

$$P_2(t) = \frac{1}{\sqrt{2\pi \times 0.1}} \times \frac{1}{t} \times \exp\left(-\frac{(\log t - 1)^2}{2 \times 0.1}\right), \quad (16c)$$

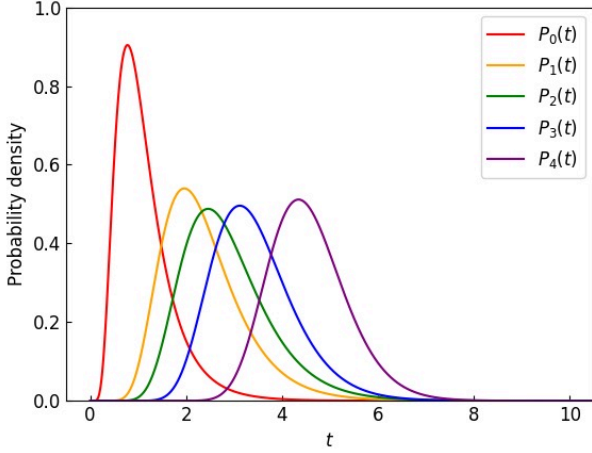


Fig. 7: Probability density functions of $P_0(t)$ – $P_4(t)$.

$$P_3(t) = \frac{1}{\sqrt{2\pi \times 0.0625}} \times \frac{1}{t} \times \exp\left(-\frac{(\log t - 1.2)^2}{2 \times 0.0625}\right), \quad (16d)$$

$$P_4(t) = \frac{1}{\sqrt{2\pi \times 0.03125}} \times \frac{1}{t} \times \exp\left(-\frac{(\log t - 1.5)^2}{2 \times 0.03125}\right). \quad (16e)$$

Fig. 8 shows the expected excessive traffic ratio. The excessive traffic ratio in the benchmark model ($\tau = 1$) is 2.96%. In the proposed model (OPT and 2SS), when $\tau = 3$, the ratio is 1.52%. This means that, by dividing one round into three time slots, the excessive traffic amount decreases by 48.8%. In the proposed model with 2SS, when $\tau = 4, 6, 9$, and 12, the ratios are 1.5%, 1.5%, 1.38% and 1.36%, respectively. In the proposed model with OPT, when $\tau = 4, 6$, and 9, the ratio are 1.38%, 1.42%, and 1.38%, respectively. When τ increases, the excessive traffic ratio can increase. It is because the optimal command time can be moved away from $\mathcal{T}$ factors by changing the division of round. Note that, the excessive traffic ratio of OPT when $\tau = 12$ cannot be computed due to insufficient memory.

Fig. 9 shows the computation time required to obtain the command time of each flow when the excessive traffic amount. The computation time of the benchmark model is 11.42 s. The proposed model with 2SS when $\tau = 3, 4, 6, 9$, and 12 takes 11, 15, 36, 56, and 105 times longer than the benchmark model, respectively. From Section V, the computation complexity of OPT is $\mathcal{O}(|\mathcal{T}|^4)$. The computation times of OPT when $\tau = 3, 4, 6$, and 9 are 342.07 s, 984.25 s, 4310.54 s, and 18614.97 s, respectively, which are consistent with the computation complexity. On the other hand, the computation times of 2SS when $\tau = 3, 4, 6, 9$, and 12 are 120.36 s, 168.78 s, 408.19 s, 642.5 s, and 1202.08 s, respectively. Since the granularity n is defined by (4), it is not possible to take the minimum value of (3). Therefore, the computation times of 2SS do not strictly follow $\mathcal{O}(|\mathcal{T}|^{\frac{|F|-1}{2}})$.

Since the difference of the excessive traffic ratios of OPT and 2SS is at most 0.12% in this evaluation, we compare

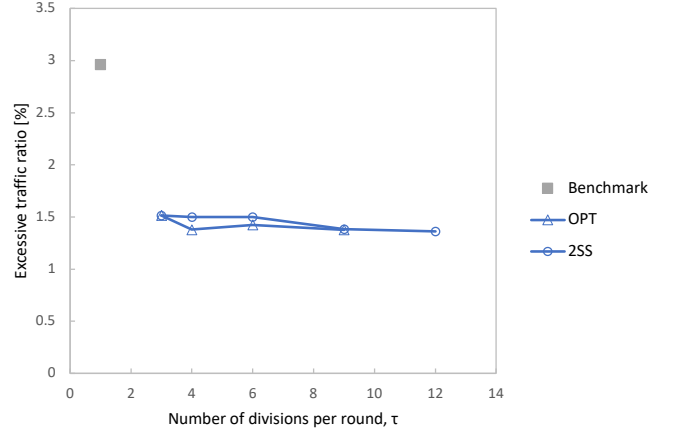


Fig. 8: Ratio of excessive traffic in Microsoft's inter-datacenter WAN when setting of flows is Table V.

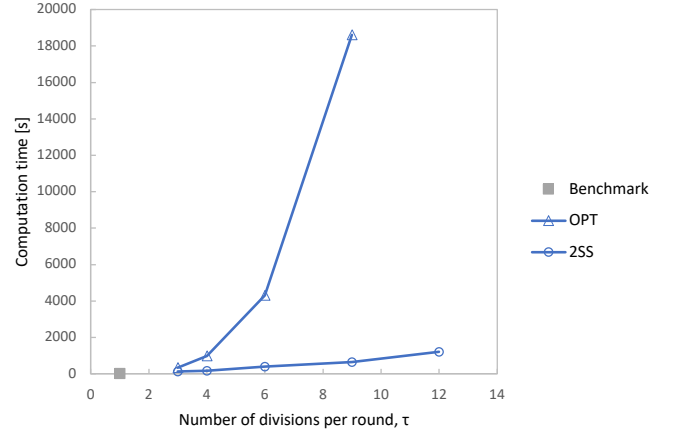


Fig. 9: Computation time in Microsoft's inter-datacenter WAN when setting of flows is Table V.

the benchmark model and the proposed model with 2SS, hereinafter.

D. Result in Microsoft's inter-datacenter WAN with random flow paths

In this section, the performance of the proposed model is evaluated by generating the path of each flow by the optimization problem (14a)–(14v). τ , an integer that indicates how many time slots one round is divided into, is set to 3, 6, and 12. The granularity n when $\tau = 3, 6$, and 12 are set to 2, 3, and 3. The other settings are the same as Section VII-C. We examine five scenarios.

Fig. 10 shows the excessive traffic ratio. In the fourth scenario, the proposed model with 2SS ($\tau = 3$) obtains the highest percentage of reduction of excessive traffic compared to the benchmark model, which is 71.2%. The difference of the amounts by 2SS ($\tau = 3$) and 2SS ($\tau = 6$) is the largest in the third scenario, which is 33.3%. The difference of the amounts by 2SS ($\tau = 6$) and 2SS ($\tau = 12$) is the largest in the fifth scenario, which is 7.87%. On the other hand, in

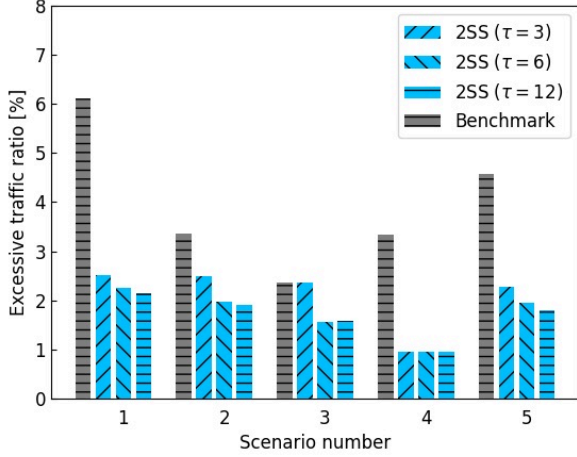


Fig. 10: Ratio of excessive traffic in Microsoft's inter-datacenter WAN when probability distributions are (16a)–(16e).

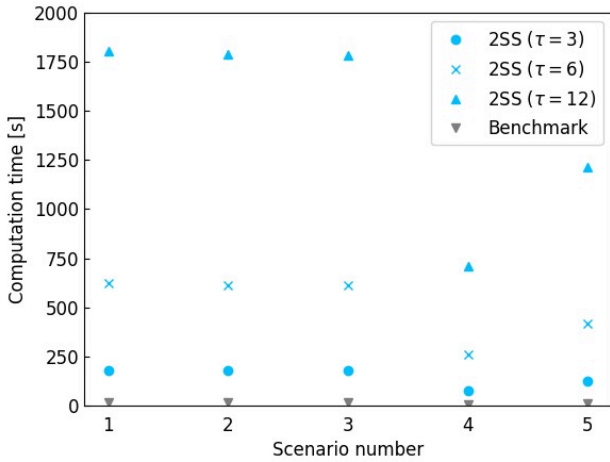


Fig. 11: Computation time in Microsoft's inter-datacenter WAN when probability distributions are (16a)–(16e).

fourth scenario, the difference of the amounts by 2SS ($\tau = 3$) and 2SS ($\tau = 12$) is 0.12%. Also, in the third scenario, the amount of 2SS when $\tau = 12$ is larger than that of 2SS when $\tau = 6$ by 1.17%. This is because 2SS can yield suboptimal solutions. Increasing τ does not always lead to decreasing the excessive traffic amount.

Fig. 11 shows the computation time. The benchmark model attains the smallest computation time, followed by the proposed model with 2SS ($\tau = 3$), 2SS ($\tau = 6$), and 2SS ($\tau = 12$). The computation time in 2SS with $\tau = 3$ is 10.3 times larger than that in the benchmark model. The computation times in 2SS with $\tau = 6$ and $\tau = 12$ are 3.39 and 9.75 times larger than that in 2SS with $\tau = 3$, respectively.

Next, we perform the same experiment as above changing the probability density functions to (17a)–(17e) shown in Fig. 12.

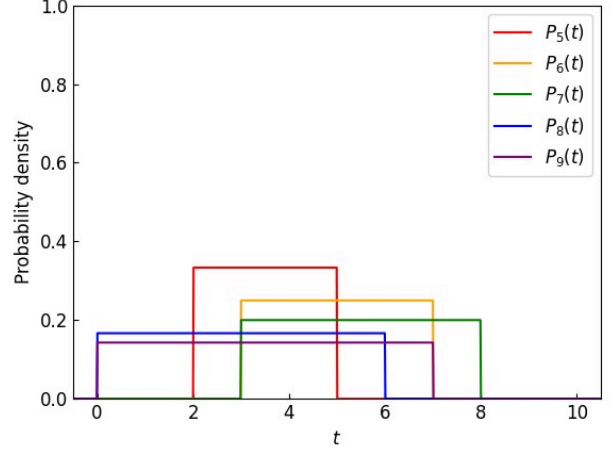


Fig. 12: Probability density functions of $P_5(t)$ – $P_9(t)$.

$$P_5(t) = \begin{cases} 0 & (t < 2), \\ \frac{1}{3} & (2 \leq t \leq 5), \\ 0 & (5 < t), \end{cases} \quad (17a)$$

$$P_6(t) = \begin{cases} 0 & (t < 3), \\ \frac{1}{4} & (3 \leq t \leq 7), \\ 0 & (7 < t), \end{cases} \quad (17b)$$

$$P_7(t) = \begin{cases} 0 & (t < 3), \\ \frac{1}{5} & (3 \leq t \leq 8), \\ 0 & (8 < t). \end{cases} \quad (17c)$$

$$P_8(t) = \begin{cases} 0 & (t < 0), \\ \frac{1}{6} & (0 \leq t \leq 6), \\ 0 & (6 < t). \end{cases} \quad (17d)$$

$$P_9(t) = \begin{cases} 0 & (t < 0), \\ \frac{1}{7} & (0 \leq t \leq 7), \\ 0 & (7 < t). \end{cases} \quad (17e)$$

Fig. 13 shows the excessive traffic ratio. In the fifth scenario, the proposed model with 2SS ($\tau = 3$) obtains the highest percentage of reduction of excessive traffic compared to the benchmark model, which is 11.2%. The difference of the amounts by 2SS ($\tau = 3$) and 2SS ($\tau = 6$) is the largest in the fourth scenario, which is 8.42%. The difference of the amounts by 2SS ($\tau = 6$) and 2SS ($\tau = 12$) is the largest in the fifth scenario, which is 2.06%. Since the uniform distribution does not have its peak value, the variance in the excessive traffic amount when using $P_5(t)$ – $P_9(t)$ is less than that when using $P_0(t)$ – $P_4(t)$ by shifting the probability distributions. This is because the reduction of excessive traffic in Fig. 13 is less than that in Fig. 10.

Fig. 14 shows the computation time. This result has almost the same trend as Fig. 11.

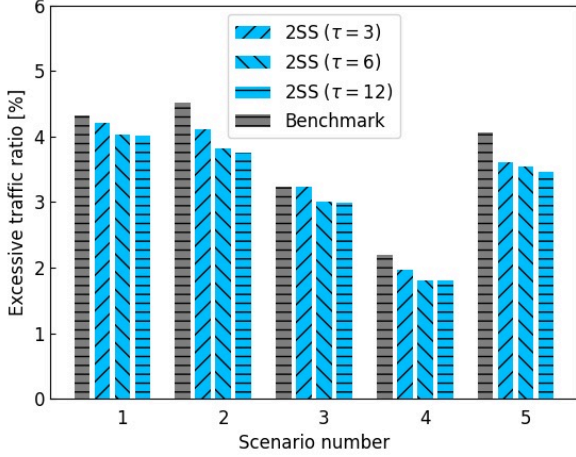


Fig. 13: Ratio of excessive traffic in Microsoft's inter-datacenter WAN when probability distributions are (17a)–(17e).

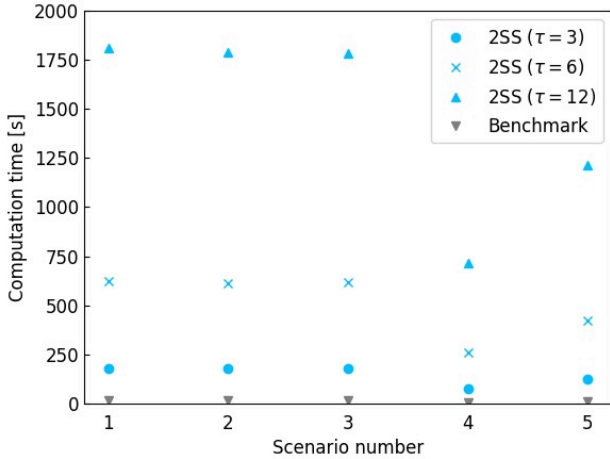


Fig. 14: Computation time in Microsoft's inter-datacenter WAN when probability distributions are (17a)–(17e).

E. Result in NSFNET with random flow paths

In this section, we compare the performances of the benchmark model and the proposed model in NSFNET shown in Fig. 15 [29]. The other settings are the same as Section VII-D. We examine five scenarios.

Fig. 16 shows the excessive traffic ratio. In the second scenario, the proposed model with 2SS ($\tau = 3$) obtains the highest percentage of reduction of excessive traffic compared to the benchmark model, which is 46.9%. The difference of the amount by 2SS ($\tau = 3$) and 2SS ($\tau = 6$) is the largest in the third scenario, which is 18.69%. The difference of the amount by 2SS ($\tau = 6$) and 2SS ($\tau = 12$) is the largest also in the third scenario, which is 6.7%.

Fig. 17 shows the computation time. The result trend in Fig. 17 is the same as Fig. 11. The benchmark model attains the smallest computation time, followed by the proposed

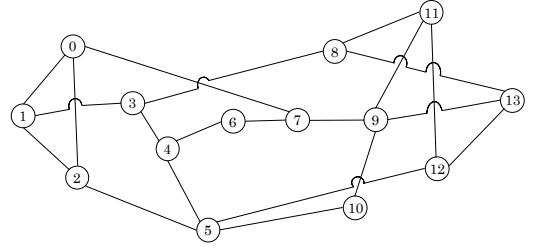


Fig. 15: NSFNET [29].

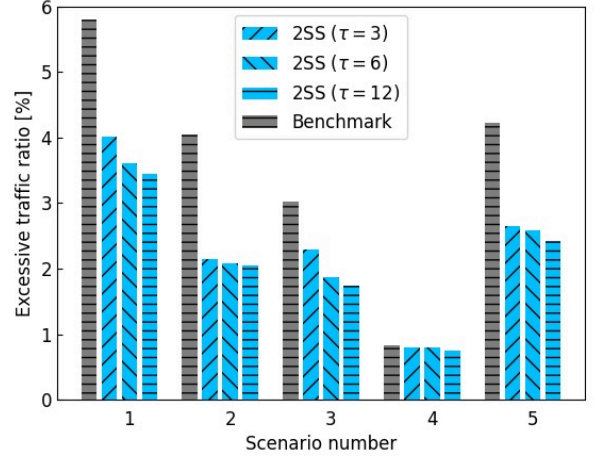


Fig. 16: Ratio of excessive traffic in NSFNET when probability distributions are (16a)–(16e).

model with 2SS ($\tau = 3$), 2SS ($\tau = 6$), and 2SS ($\tau = 12$).

Next, we perform the same experiment as above changing the probability density functions to (17a)–(17e) shown in Fig. 12.

Fig. 18 shows the excessive traffic ratio. As with Fig. 16, the proposed model suppresses the excessive traffic ratio compared to the benchmark model. In the fourth scenario, the proposed model with 2SS ($\tau = 3$) obtains the highest percentage of reduction of excessive traffic compared to the benchmark model, which is 43%.

Fig. 19 shows the computation time. This result is almost the same as Fig. 17. The computation time does not depend on the probability distribution since the excessive traffic amounts are computed by generating random numbers.

VIII. DISCUSSION

As mentioned in Section III, the proposed model assumes that the probability distributions of times required for flow migrations are known. The delay time between the controller and the switch is influenced by various factors: transmission delay, propagation delay, processing delay, queuing delay, load of each switch, and so on. In real networks, the probability distribution of delay times can have a certain width due to the variations in packet processing and queuing [30], [31]. To implement the proposed model on a real network, it is necessary to obtain the probability distributions of delay times in advance by experiments.

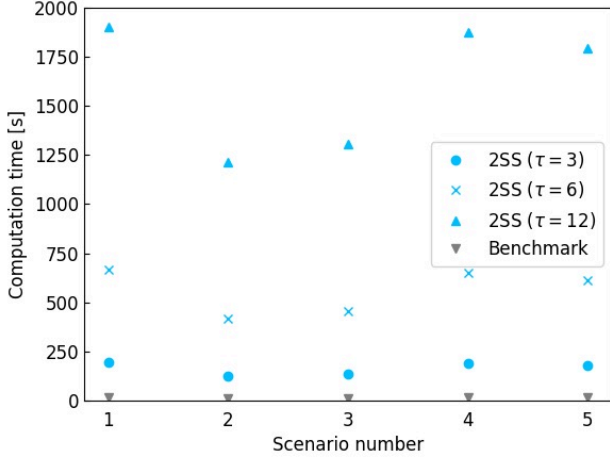


Fig. 17: Computation time in NSFNET when probability distributions are (16a)–(16e).

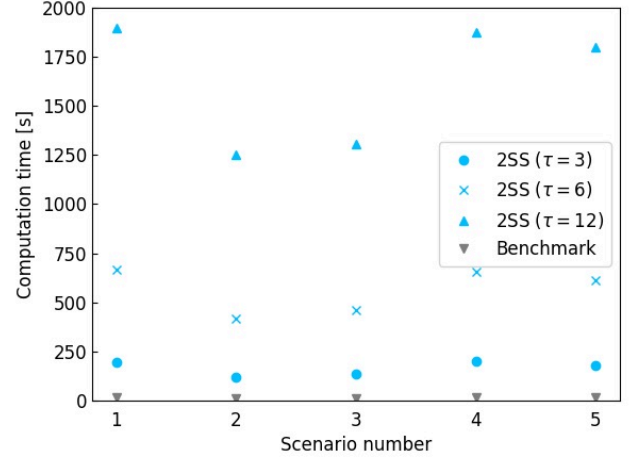


Fig. 19: Computation time in NSFNET when probability distributions are (17a)–(17e).

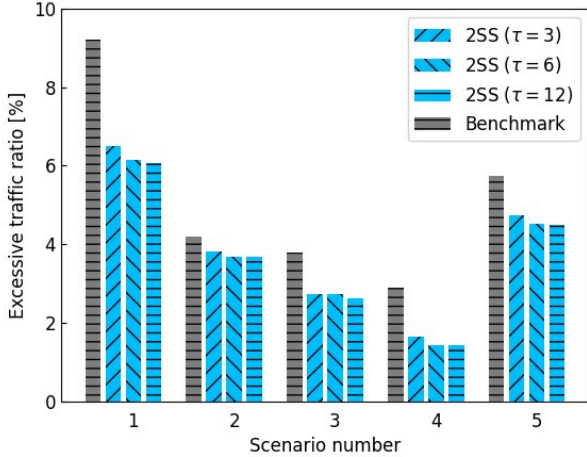


Fig. 18: Ratio of excessive traffic in NSFNET when probability distributions are (17a)–(17e).

For example, when OpenFlow [3] is used as the SDN protocol, the probability distributions can be obtained as follows. Let T_1 denote the time at which the controller sends a flow migration command to the switch, and T_2 denote the time at which the switch receives and applies the flow entry. T_1 is the time when the OpenFlow controller sends a *Flow-mod* message to add or delete a flow entry to the switch. The controller sends a *Barrier request* message to the switch to know whether the flow table processing is completed. T_2 is the time when the switch sends a *Barrier reply* message to notify the controller that the flow entry is applied. These messages can be captured by packet capturing software. The distribution of $T_2 - T_1$ can be statistically obtained by performing packet captures multiple times.

IX. CONCLUSION

This paper proposed the flow update model which minimizes the amount of excessive traffic by shifting the proba-

bility distributions of times required for flow migrations. In the proposed model, time axis is divided into smaller time slots than rounds. Each probability distribution is shifted by assigning time, when the controller commands each flow to be migrated, to time slots. The proposed model was formulated as an optimization problem, which determines time at which the controller commands migrations to the switches. Since the optimization problem is formulated as a nonlinear program, this paper presented two computation methods. Numerical evaluation showed that the proposed model can suppress the amount of excessive traffic compared to the round-based model. When the round size is divided into three time slots, the reduction is at most 46.9% in NSFNET where there are five flows.

There are some problems that need to be solved in the proposed model. The first one is that it takes time to compute the amount of excessive traffic; it can take three hours to compute the amount of excessive traffic in NSFNET where there are six flows and upper limit of command time is six. Future work includes the improvement of the computation method of the amount of excessive traffic and evaluating the proposed model in a network with a large number of flows. The second one is that the proposed model does not consider the existence of remaining flows on the old routes after their migrations. Another future work is to develop a model that also takes into account congestion caused by such flows.

REFERENCES

- [1] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426 (Informational), Internet Engineering Task Force, Jan. 2015. [Online]. Available: <http://www.ietf.org/rfc/rfc7426.txt>
- [2] K. Suzuki, K. Sonoda, N. Tomizawa, Y. Yakuwa, T. Uchida, Y. Higuchi, T. Tonouchi, and H. Shimonishi, "A survey on OpenFlow technologies," *IEICE Trans. Commun.*, vol. E97.B, no. 2, pp. 375–386, 2014.
- [3] A. Nygren, B. Pfaff, B. Lantz, B. Heller, C. Barker, C. Beckmann, et al., "OpenFlow switch specification version 1.5.1," *Open Networking Foundation Tech. Rep.*, 2015.

- [4] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1435–1461, 2019.
- [5] K.-T. Foerster, A. Ludwig, J. Marcinkowski, and S. Schmid, "Loop-free route updates for software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 328–341, 2018.
- [6] K.-T. Foerster, R. Mahajan, and R. Wattenhofer, "Consistent updates in software defined networks: On dependencies, loop freedom, and blackholes," in *2016 IFIP Netw. Conf. (IFIP Netw.) and Workshops*, 2016, pp. 1–9.
- [7] A. Ludwig, M. Rost, D. Foucard, and S. Schmid, "Good network updates for bad packets: Waypoint enforcement beyond destination-based routing policies," in *Proc. 13th ACM Workshop Hot Topics Netw.*, ser. HotNets-XIII. New York, NY, USA: Association for Computing Machinery, 2014, pp. 1–7.
- [8] C. Wen, P. Yang, Q. Liu, J. Luo, and L. Yu, "Minimizing congestion impairment of network update in SDN: A flow-based solution," in *2018 10th Int. Conf. Wireless Commun. and Signal Process. (WCSP)*, 2018, pp. 1–6.
- [9] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *2015 IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, 2015, pp. 1–10.
- [10] L. Luo, H. Yu, S. Luo, and M. Zhang, "Fast lossless traffic migration for SDN updates," in *2015 IEEE Int. Conf. Commun. (ICC)*, 2015, pp. 5803–5808.
- [11] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM 2013 Conf. SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 15–26.
- [12] J. Yu, J. Bi, G. Zhang, X. Fan, and Z. Li, "Loop-free rerouting of multiple policies with overlaps," *IEEE Commun. Lett.*, vol. 22, no. 7, pp. 1370–1373, 2018.
- [13] H. Zhou, X. Gao, J. Zheng, and G. Chen, "Scheduling relaxed loop-free updates within tight lower bounds in SDNs," *IEEE/ACM Trans. Netw.*, vol. 28, no. 6, pp. 2503–2516, 2020.
- [14] K.-T. Foerster and S. Schmid, "Distributed consistent network updates in SDNs: Local verification for global guarantees," in *2019 IEEE 18th Int. Symp. Netw. Comput. and Applicat. (NCA)*, 2019, pp. 1–4.
- [15] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. ACM SIGCOMM 2012 Conf. Applicat., Technol., Archit., and Protocols Comput. Commun.*, ser. SIGCOMM '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 323–334.
- [16] J. Hua, X. Ge, and S. Zhong, "Foum: A flow-ordered consistent update mechanism for software-defined networking in adversarial settings," in *IEEE INFOCOM 2016 - 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
- [17] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently secure network updates," in *Proc. 2016 ACM SIGMETRICS Int. Conf. Meas. and Modeling Comput. Sci.*, ser. SIGMETRICS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 273–284.
- [18] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, ser. HotSDN '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 49–54.
- [19] S. Brandt, K.-T. Förster, and R. Wattenhofer, "On consistent migration of flows in SDNs," in *IEEE INFOCOM 2016 - 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
- [20] J. Zheng, B. Li, C. Tian, K.-T. Foerster, S. Schmid, G. Chen, and J. Wux, "Scheduling congestion-free updates of multiple flows with chronicle in timed SDNs," in *2018 IEEE 38th Int. Conf. Distributed Comput. Syst. (ICDCS)*, 2018, pp. 12–21.
- [21] K.-T. Foerster, "On the consistent migration of splittable flows: Latency-awareness and complexities," in *2018 IEEE 17th Int. Symp. Netw. Comput. and Applicat. (NCA)*, 2018, pp. 1–4.
- [22] K.-T. Foerster, L. Vanbever, and R. Wattenhofer, "Latency and consistent flow migration: Relax for lossless updates," in *2019 IFIP Netw. Conf. (IFIP Netw.)*, 2019, pp. 1–9.
- [23] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, p. 539–550, aug 2014.
- [24] J. Zheng, B. Li, C. Tian, K.-T. Foerster, S. Schmid, G. Chen, J. Wu, and R. Li, "Congestion-free rerouting of multiple flows in timed SDNs," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 968–981, 2019.
- [25] X. He, J. Zheng, H. Dai, C. Zhang, G. Li, W. Dou, W. Rafique, Q. Ni, and G. Chen, "Continuous network update with consistency guaranteed in software-defined networks," *IEEE/ACM Trans. Netw.*, vol. 30, no. 3, pp. 1424–1438, 2022.
- [26] A. R. Hezaveh and M. Nobakht, "On priority-based scheduling for network updates in SDN," in *2023 IEEE 48th Conf. Local Comput. Netw. (LCN)*, 2023, pp. 1–4.
- [27] T. Qu, D. Guo, J. Xu, and Z. Liu, "Delay updating in software-defined datacenter networks," in *2017 IEEE/ACM 25th Int. Symp. Quality Service (IWQoS)*, 2017, pp. 1–6.
- [28] "IBM ILOG CPLEX optimization studio," <https://www.ibm.com/products/ilog-cplex-optimization-studio>, (Accessed on 22 August 2023).
- [29] "NSFNET, A Partnership for high-speed networking: Final report," https://www.merit.edu/wp-content/uploads/2019/06/NSFNET_final-1.pdf, (Accessed on 31 January 2023).
- [30] K. Igai and E. Oki, "A simple link-utilization estimation scheme based on RTT measurement," in *2011 1st International Symposium on Access Spaces (ISAS)*, 2011, pp. 266–270.
- [31] K. Igai and E. Oki, "A link utilization estimation scheme using RTT measurement considering processing delay jitter and available timer granularity," *World Telecommunications Congress (WTC)*, 2012.



Reo Uneyama is pursuing the M.E. degree at Graduate School of Informatics, Kyoto University, Kyoto, Japan. He received the B.E. degree from Kyoto University, Japan, in 2023. His research interests include optimization, software-defined network, and network update problem.



Takehiro Sato received the B.E., M.E. and Ph.D. degrees in engineering from Keio University, Japan, in 2010, 2011 and 2016, respectively. He is currently an associate professor in Graduate School of Informatics, Kyoto University, Japan. His research interests include design and control methods for optical and virtualized networks. From 2011 to 2012, he was a research assistant in the Keio University Global COE Program. From 2012 to 2015, he was a research fellow of Japan Society for the Promotion of Science. From 2016 to 2017, he was a research

associate in Graduate School of Science and Technology, Keio University, Japan.



Eiichi Oki is a Professor at Kyoto University, Kyoto, Japan. He was with Nippon Telegraph and Telephone Corporation (NTT) Laboratories, Tokyo, from 1993 to 2008, and The University of Electro-Communications, Tokyo, from 2008 to 2017. From 2000 to 2001, he was a Visiting Scholar at Polytechnic University, Brooklyn, New York. His research interests include routing, switching, protocols, optimization, and traffic engineering in communication and information networks.