A Study on Approaches for Stable Distributed Systems in Unstable Network Environments

Submitted to

Graduate School of Information Science and Technology

Osaka University

January 2015

Yuichi SUDO

List of Related Publications

Journal Papers

1. Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "Loosely-stabilizing leader election in a population protocol model," *Theoretical Computer Science*, vol. 444, pp. 100-112, 2012.

2. Yuichi Sudo, Kunio Hato, Junichi Murayama, "Performance evaluation for cloud-computing systems by audit measurement," *IEICE Transactions on Information and Systems* , vol. J97-D, No. 7, pp. 1148-1157, 2014.

3. Yuichi Sudo, Toshimitsu Masuzawa, Gen Motoyoshi, and Tutomu Murase, "Pseudo polynomial time algorithms for optimal longcut route selection," *IEICE Transactions on Information and Systems* , vol. E98-D, No. 3, 2015 (to appear).

Conference Papers

4. Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "Loosely-stabilizing leader election in population protocol model," in *Proceedings of the 16th International Conference on Structural Information and Communication Complexity*, SIROCCO '09, pp. 295–308, 2009.

5. Yuichi Sudo, Daisuke Baba, Junya Nakamura, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "An agent exploration in unknown undirected graphs with whiteboards," in *Proceedings of the Third International Workshop on Reliability, Availability, and Security*, WRAS '10, pp. 8:1–8:6, 2010.

6. Gen Motoyoshi, Yuichi Sudo, Tutomu Murase, and Toshimitsu Masuzawa, "Advantages of optimal longcut route for wireless mobile users," in *IEEE International Conference on Communications* , ICC '11, pp. 1-6, 2011.

7. Yuichi Sudo, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, "Loosely-stabilizing leader election on arbitrary graphs in population protocols," in *The 18th International Conference on Principles of Distributed Systems* , OPODIS '14, 2014 (to appear).

Technical Reports

8. Yuichi Sudo, Junya Nakamura, Yukiko Yamauchi, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, “Loosely-stabilizing leader election in population protocol model,” in *IPSJ SIG Technical Report*, vol. 2009-AL-124, no. 5, pp. 1-8, 2009.

9. Yuichi Sudo, Gen Motoyoshi, Tutomu Murase, and Toshimitsu Masuzawa, “Optimal longcut route selection for wireless mobile user ,” in *IEICE Technical Report*, vol. 109, *RCS2009-269*, pp. 71-76, 2010.

10. Yuichi Sudo, Daisuke Baba, Junya Nakamura, Fukuhito Ooshita, Hirotsugu Kakugawa, and Toshimitsu Masuzawa, “Time and space efficient graph exploration by a mobile agent using whiteboard,” in *IEICE Technical Report*, vol. 109, *COMP2009-58*, pp. 57-64, 2010.

11. Yuichi Sudo, Kunio Hato, Hidetsugu Kobayashi, and Eiji Kuwana, “Performance measurement of cloud resources for inter-cloud computing,” in *IEICE Technical Report*, vol. 112, *IN2012-153*, pp. 87-92, 2013.

12. Yuichi Sudo, Kunio Hato, Hidetsugu Kobayashi, and Eiji Kuwana, “Evaluation of performance measurement method of cloud resources for inter-cloud computing,” in *IEICE Technical Report*, vol. 112, *IN2012-156*, pp. 13-18, 2013.

13. Yuichi Sudo and Kunio Hato, “Performance analysis of public clouds,” in *IEICE Technical Report*, vol. 113, *IN2013-60*, pp. 13-18, 2013.

List of Unrelated Publications

Conference Papers

14. Hu Bo, Yuichi Sudo, Kunio Hato, Yuuichi Murata, and Junichi Murayama, “Cost reduction evaluation of sharing backup servers in inter-cloud,” in *Proceedings of the 19th Asia-Pacific Conference on Communications*, APCC ’13, pp. 256-261, 2013

Technical Reports

15. Kunio Hato, Yuichi Sudo, Hu Bo, Tsuyoshi Kondoh, Yuuichi Murata, and Junichi Murayama, “Resource Information Exchange Scheme for Inter-Cloud Systems,” in *Proceedings of the Society Conference of IEICE*, vol. 2012, *Communicatoin(2)*, pp. 57-58, 2012.

16. Tsuyoshi Kondoh, Yuichi Sudo, and Kunio Hato, "Memory Over Commitment Detection Using Latency Measurement," in *Proceedings of the IEICE General Conference*, vol. 2013, *Communicatoin(2)*, pp. 147, 2013.

17. Kunio Hato, Yuichi Sudo, Hu Bo, Tsuyoshi Kondoh, Yuuichi Murata, Junichi Murayama, Hidetsugu Kobayashi, and Eiji Kuwana "Evaluation of Resource Information Exchange Scheme for lnter-Cloud Systems," in *Proceedings of the IEICE General Conference*, vol. 2013, *Communicatoin(2)*, pp. 94-95, 2013.

18. Yuichi Murata, Kunio Hato, Yuichi Sudo, Hu Bo, Tsuyoshi Kondoh, Junichi Murayama, Hidetsugu Kobayashi, and Eiji Kuwana "Resource Data Model of Inter-cloud for Standardization," in *Proceedings of the IEICE General Conference*, vol. 2013, *Communicatoin(2)*, pp. 124-125, 2013.

19. Tsuyoshi Kondoh, Yuichi Sudo, and Kunio Hato, "Reduction of VM Lead Time in Massive Scale Out," in *IEICE Technical Report*, vol. 113, *IN2013-30*, pp. 29-32, 2013.

20. Yuichi Sudo and Kunio Hato, "Blocking off reflective DoS attacks by dynamic packet Filter," in *IEICE Technical Report*, vol. 113, *IN2013-181*, pp. 223-228, 2013.

21. Yuichi Sudo, Takahiro Hamada, Yuichi Murata, and Hideo Kitazume, "Detection of malicious database queries based on matching with HTTP requests," in *IEICE Technical Report*, vol. 114, *IN2014-65*, pp. 99-104, 2014.

Abstract

Distributed systems consisting of numerous nodes and links such as the Internet, sensor networks, and ad hoc networks are in wide spread use all over the world. Since distributed systems have numerous nodes and links, there is always non-negligible possibility that some nodes or links are crashed, or the performances of some nodes are drastically degraded. Hence, many distributed systems are unstable in terms of functionality or performance. On the other hand, users of distributed systems need stable services in most cases. Therefore, the author has tackled with developing methods for the users to enjoy stable services of the systems. In this thesis, the author presents four contributions to ensure fault-tolerance, communication performance, and computing performance of distributed systems.

First, the author introduces a novel concept of *loose stabilization*, which is an extension of *self-stabilization*. Self-stabilizing systems proposed by Dijkstra in 1974, refer to the distributed systems satisfying that (i) starting from any configurations, the system eventually converges to a safe configuration (convergence), and (ii) once the system reaches a safe configuration, the system keeps its specification forever (closure). A self-stabilizing system has high fault-tolerance: the system can recover from any transient fault (memory crash, topology change, and so on). However, owing to the strict requirements of self-stabilization (convergence and closure), it is known that there exists no self-stabilizing system for some problems. For example, no self-stabilizing algorithm exists for the leader election problem in the population protocol model (the PP model), which is a common model of mobile sensor networks unless the exact number of nodes is available. To circumvent this difficulty, the author introduces the concept of *loose-stabilization*, which relaxes the closure requirement without impairing the fault-tolerance. To show effectiveness of loose-stabilization, the author presents loosely-stabilizing algorithms that solve the leader election problem in the PP model. Specifically, the author presents a loosely-stabilizing leader election algorithm on complete graphs in the PP model as the first contribution (Chapter 2), and two loosely-stabilizing leader election algorithms on arbitrary graphs in the PP model as the second contribution (Chapter 3).

Next, the author introduces the study of finding the best route from the current location to the destination location in terms of *communication quality* for mobile users. They use various mobile networks such as cellar networks (e.g. W-CDMA and LTE) and wireless LANs (e.g. WiFi) in urban areas, hence the wireless environments (communication speeds) of the users highly depend on their routes. As the third contribution (Chapter 4), the author formulates this problem as *the optimal longcut route selection problem* and proves its NP-hardness. Furthermore, the author proposes two pseudo-polynomial algorithms for the problem, and evaluates their execution time by a theoretical (asymptotical) analysis and an empirical (simulating) experiments.

Finally, the author introduces the study on a trustworthy measurement method of computational performance of virtual machines (VMs) in cloud computing systems. Virtual machines in cloud computing systems generally show unstable and time-dependent performances. Hence, frequent and continuous measurements of VM performance are necessary to understand the VM performance of each cloud computing service. However, frequent and continuous measurements of VM performance by each user impose a large cost on the user. On the other hand, if cloud service providers measure VM performances of their own systems and publish the performance information continuously, the user can avoid the measurement cost but may not be confident with the performance information since the provider may fabricate it. As the fourth contribution (Chapter 5), the author proposes a method for users to get trustworthy information about the real performance of VMs with low costs. In this method, cloud service providers publish the performance information of their VMs at regular intervals, and users of their services infrequently measure the performance of VMs and try to detect the exaggeration of the published information. The experimental results show that the users can detect, in most of the cases, the exaggeration of several percentages on the performance the providers make.

Contents

List of Figures

List of Tables

List of Codes

Chapter 1

Introduction

Distributed systems consist of numerous nodes and communication links between nodes. For example, the Internet, peer to peer networks, sensor networks, grid computing systems, and cloud computing systems are examples of distributed systems. One of the major issues in this field is to provide stable functions and performance of a distributed system. Since a distributed system has numerous nodes, there is always non-negligible possibility that some nodes or links are crashed or the performance of some nodes are drastically degraded, which causes unstability of the system in terms of functionality or performance. For example, many applications for a distributed system are implemented assuming the existence of a single leader node that coordinates the behavior of all the nodes of the system. Hence, the applications cannot behave correctly when the leader node is crashed.

So far, the author has tackled with providing stable services in unstable distributed systems by realizing fault tolerance or guaranteeing communication performance or computational performance. Specifically, to realize fault tolerance of a distributed system, the author proposed a novel variation of self-stabilization that relaxes the closure requirements of the original self-stabilization without impairing its fault tolerance in practice. The author also dealt with optimizing communication quality of mobile networks: the author formalizes and solves the problem to find the route from the starting point to the destination point that maximizes communication quality of a mobile user in an urban area. Furthermore, the author tackled with obtaining trustworthy information of computational performance of virtual machines in cloud computing systems. The author present a novel method by which the users can get trustworthy information about the performance of virtual machines of each cloud computing system with low cost.

The rest of this chapter presents the overviews of these studies each of which corresponds to

Chapter 2, 3, 4, or 5.

1.1 Loosely-stabilizing Leader Election on Complete Graphs in Population Protocols

Chapter 2 focuses on improving fault tolerance of distributed systems. To achieve high fault tolerance, the author introduces a novel concept of *loose-stabilization* that is a variant of self-stabilization.

Self-stabilization [1] is a property of a distributed system: Even when any number and any type of transient faults (e.g. memory crash and topology changes) hit the system, it can autonomously recover from the faults. The notion of self-stabilization is described as follows: (i) starting from any arbitrary initial configuration, a self-stabilizing system eventually reaches a *safe configuration* (*convergence*), and (ii) once a self-stabilizing system reaches a safe configuration, then it keeps its specification forever (*closure*). Although self-stabilizing systems provide excellent fault-tolerance as mentioned above, designing self-stabilizing protocols is difficult and, what is worse, might be impossible due to the severe requirements of self-stabilization. To circumvent these difficulty and impossibility, many researchers have tried to relax the severe requirement of self-stabilization and proposed many variants such as probabilistic stabilization [2], quasi-stabilization [3], and weak stabilization [4]. However, these variants of self-stabilization still have room for improvement. There exists neither a probabilistic stabilizing protocol nor a weak-stabilizing protocol for some problems (e.g. leader election in the population protocol model [5] introduced later). On the other hand, quasi-stabilization heavily impairs fault tolerance of the original self-stabilization since it needs a kind of initialization of a distributed system after transient faults happen.

In this chapter, the author introduces a novel concept of loose-stabilization, a new variant of self-stabilization. All existing variants of self-stabilization relaxes the convergence requirement to the best of the author's knowledge. On the other hand, loose-stabilization relaxes the closure requirement: loose-stabilization requires that, after the system reaches a safe configuration, the system keeps its specification for a sufficiently long time, though not forever as in the original self-stabilization. For example, if the system keeps its specification for an exponentially long time with respect to the number of nodes, this relaxation does not impair the fault tolerance of self-stabilization at all in practical perspective.

To verify the effectiveness of loose-stabilization, this chapter presents a loosely-stabilizing algorithm for the leader election problem on complete graphs in the population protocol model

[5] (the PP model). The PP model is a common model of mobile sensor networks of anonymous mobile sensing devices where two devices communicate with each other only when they come sufficiently close to each other. Self-stabilizing leader election is a important problem in the PP model partly because many population protocols in the literature work on the assumption that a unique leader exists [5, 6, 7]. However, it was proved that no self-stabilizing algorithm exists for the leader election problem on complete graphs in the PP model unless every node (i.e. mobile devices) has a knowledge of the exact number of nodes [7]. Therefore, the proposed algorithm shows effectiveness of loose-stabilization: loose-stabilization allows a solutions for the problem that is known to have no self-stabilizing solution.

1.2 Loosely-stabilizing Leader Election on Arbitrary Graphs in Population Protocols

Chapter 3 focuses on loosely-stabilizing leader election on arbitrary graphs in the PP model. A new leader must be created when no leader exists in the system and the number of leaders must be reduced to one when multiple leaders exist. On a complete graph where all nodes can communicate with each other, every node can confirm the existence of a leader node since every node directly communicates with the leader node if it exists. Furthermore, every leader node can detect the existence of another leader node since any pair of leader nodes can communicate with each other. However, on an arbitrary graph, detecting the absence or multiplicity of leader nodes is harder because not every pair of nodes can communicate with each other directly.

Then, the author presents two loosely-stabilizing algorithms: one uses the unique node identifiers and the other uses random numbers to solve leader election on arbitrary graphs. The algorithm using node-identifiers adopts the traditional“minimum ID selection” approach while the algorithm using random numbers adopts a novel approach we call “virus war mechanism”. Given upper bounds N of n and Δ of the maximum degree of nodes, both algorithms keep the unique leader for $\Omega(Ne^N)$ expected steps after reaching a loosely-safe configuration. The former algorithms reaches a loosely-safe configuration within $O(mN\Delta \log n)$ expected steps while the latter algorithms does within $O(mN^3\Delta^2 \log N)$ expected steps where m is the number of edges of the graph. These two algorithms also show another evidence of effectiveness of the loose-stabilization since no self-stabilizing algorithm exists for the leader election problem on arbitrary graphs in the PP model even if the unique node identifiers and random numbers are available.

1.3 Optimal Longcut Route Selection

Chapter 4 focuses on communication quality of mobile users who use mobile networks for connecting to the Internet. Users of wireless mobile devices (e.g. smart phones, tablets, and laptops) need the Internet access not only when they are staying at home or office, but also when they are traveling. Consequently, when such users search a route from their current location to their destination, they may prefer the route with a good wireless communication environment to one with the shortest travel time. In this chapter, the author formulates the above situation as the optimization problem called *optimal longcut route selection* (OLRS), which requires to find the route that maximizes the communication quality during travel subject to a travel time constraint. First, the author proves that OLRS is NP-hard. Therefore, it is impossible to devise a polynomial time algorithm for this problem as long as P $\neq$ NP. Then, the author presents two pseudo-polynomial time algorithms for OLRS. Based on the author's theoretical (asymptotical) analysis, one of the two is better in terms of the worst-case execution time, and the other one is better for execution time of most inputs (i.e. a graph representing urban area, the starting point, the destination point, and wireless environment of each point and edge). The simulation experiment shows that both the two algorithms find the optimal solution within practically short time for sufficiently large inputs, and the latter algorithm is far faster than the former algorithm on average.

1.4 Performance Evaluation for Cloud Computing Systems by Audit Measurements

Chapter 5 focuses on computational performance of virtual machines in cloud computing systems. Recently, the market of cloud computing services that provide their users with virtual machines through internet services has grown significantly. Although cloud service providers present the specifications of their virtual machines (VMs), it is known that in many cases the real performance of the VMs deviates far from the specification. To understand the VMs' performance of cloud computing services exactly, we need frequent and continuous benchmarking on the VMs of their service. It requires high cost for users when the users directly measure the performance of the services that they are interested in. On the other hand, when the cloud service providers measure their services themselves and publish the result data, the trustworthiness of the result data is not guaranteed for users since there may exist dishonest providers who fabricate the published data to be overestimated. In this paper, we propose a method by which users get trustworthy information about the real performance of VMs of each cloud computing service with low costs. In the proposed method, cloud service providers publish the performance data of their VMs at regular intervals while users of their services infrequently measure the performance of VMs and try to detect the exaggeration of the published information. The experimental results show that the users can detect, in most of the cases, the exaggeration of several percentages on the performance the providers make.

Chapter 2

Loosely-stabilizing Leader Election on Complete Graphs in Population Protocols

2.1 Introduction

A distributed system is a collection of autonomous computational entities (processes) connected by communication links. Fault tolerance of distributed systems has attracted more and more attention since distributed systems are prone to faults. A *self-stabilizing system* [1, 8] has a desirable property that, even when any transient fault (e.g. memory crash) hits the system, it can autonomously recover from the fault. The notion of self-stabilization is described as follows: (i) starting from any arbitrary initial configuration, a system eventually reaches a *safe configuration* (*convergence*), and (ii) once a system reaches a safe configuration, then it keeps its specification forever (*closure*). Although self-stabilizing systems provide excellent fault-tolerance as mentioned above, designing self-stabilizing protocols is difficult and, what is worse, might be impossible due to the severe requirements of self-stabilization.

Many researchers have tried to relax the severe requirement of self-stabilization and proposed a lot of variants. *Probabilistic self-stabilization* [2] guarantees convergence to a safe configuration with probability 1 starting from any arbitrary configuration. *Quasi-stabilization* [3] guarantees convergence to a safe configuration only when all processes in the system start with the program counters of value 0. *Weak-stabilization* [4] guarantees that starting from any arbitrary configuration, there exists an execution that reaches a safe configuration. Devismes et al. [9] investigated the relations among self, probabilistic and weak stabilization. A notable characteristic common to all the above variants is that they relax only the convergence requirement but not the closure requirement of self-stabilization. As we shall see later, the author relaxes the closure requirement to introduce a novel notion of loose-stabilization.

In this chapter, we adopt *Population Protocol* (PP) model [10, 5, 6, 7, 11, 12] as a distributed system model. The PP model is one of the abstract models that represent wireless sensor networks of anonymous mobile sensing devices. In this model, two devices communicate with each other only when they come sufficiently close to each other (we call this event an *interaction*). For example, population protocol model can represent a flock of birds such that each bird is equipped with a sensing device of small transmission range. In such a sensor network, each device can communicate with another device only when the corresponding birds come sufficiently close to each other.

Self-stabilizing leader election in population protocol model of complete networks is an important problem and has been considered by several papers. Angluin et al. [7] prove that this problem is unsolvable if we can use no information about the network size, in other words, if a protocol must work on the complete networks of finite but any arbitrary size.[1] Cai et al. [11] prove that the exact information of the network size is necessary (and sufficient) to solve the problem. In other words, for any two distinct positive integers n and n', there exists no self-stabilizing leader election protocol that works on both the complete network of size n and the one of size n'. Fischer and Jiang [12] use external entity (a kind of failure detector) to solve the problem with no knowledge of the network size.

Contribution of This Chapter To circumvent difficulty and impossibility in designing self-stabilizing protocols, the author introduces a novel notion of *loose-stabilization*, which relaxes the closure requirement of self-stabilization. To the best of the authors' knowledge, this is the first trial to relax the closure requirement and not the convergence requirement. Intuitively, the notion of loose-stabilization is described as follows: (i) starting from any arbitrary configuration, a system reaches a *loosely-safe configuration* within a short time (*convergence*), and (ii) once a system reaches a loosely-safe configuration, then it keeps its specification for a long time (*loose-closure*). In other words, we relaxes the closure requirement by allowing a system to deviate from its specification even after a loosely-safe configuration but only after a long period satisfying the

[1] They prove this impossibility for a certain class of topology, called non-simple class. This class includes complete networks, directed line networks, and connected networks with a certain degree bound etc..

specification. The requirement of fast convergence is added to guarantee that the specification should be satisfied in most of the system running time. Actually, the loose-stabilization is practically equivalent to self-stabilization if the specification is kept for a significantly long time (e.g. exponential order with the network size) after the loosely-safe configuration.

From a practical perspective, the notion of loose-stabilization suits to the purpose of fault-tolerance better than self-stabilization. Self-stabilization has great importance for networks prone to faults, where probability of fault occurrence is not negligible and faults occur repeatedly and intermittently: self-stabilizing protocols can recover from faults and work correctly during the fault-free periods. Since the length of the fault-free period is commonly estimated by, for example, MTBF (mean time between faults), the *permanent closure* of self-stabilization (to permanently satisfy the specification after convergence) seems to be an exaggerated requirement. To such a situation, loose-stabilization satisfying the specification after convergence in a sufficiently long period (compared to the MTBF) is particularly appropriate.

To show effectiveness and feasibility of loose-stabilization, a loosely-stabilizing leader election protocol is presented in the PP model of complete networks. The protocol uses the knowledge of an upper bound, say N, of the network size: the protocol works correctly on any complete network of size N or less. Starting from any arbitrary configuration, the protocol elects a unique leader within $O(nN \log n)$ expected steps, and then, keeps the unique leader for $\Omega(Ne^N)$ expected steps where n is the actual network size. This result discloses an evidence that introduction of the loose-stabilization can circumvent impossibility results on self-stabilization; the self-stabilizing leader election in the PP model of complete networks cannot be solved even in a probabilistic way without knowledge of the exact network size (as mentioned above). The proposed protocol uses $O(\log N)$ space per device while most of prior papers on population protocols usually do not allow each devices to use more than constant space (with respect to n). However, the importance of the proposed protocol is never impaired by this fact because the above impossibility holds even if each device can use infinite space.

2.2 Preliminaries

In this section, the definitions of the population protocol model and the concept of loose-stabilization are given. Throughout this chapter, we use the notation $\mathrm{pre}_l(s)$ for describing the prefix of sequence s with length l.

A *population* consists of a collection of finite state sensing devices called *agents*. Each agent has its own state and updates the state by communication with other agents in pairs,[2] called *interactions.* We represent a population by simple directed graph $G(V,E)$: vertex set $V = \{0,1,\dots,n-1\}$ $(n \geq 2)$ represents the set of agents, and edge set $E \subseteq V \times V$ represents the set of possible interactions. If $(u,v) \in E$, agents u and v can interact (or communicate) with each other in such a way that u serves as an *initiator* and v serves as a *responder*. In this chapter, we assume that a population $G(V,E)$ is a directed complete graph, that is, the edge set E is equals to $\{(u,v) \mid u,v \in V,\ u \neq v\}$.

A *protocol* $P(Q,Y,O,T)$ consists of a finite set Q of states, a finite set Y of output symbols, an output function $O : Q \to Y$, and a transition function $T : Q \times Q \to Q \times Q$. The *output of an agent* is determined by O: when the state of an agent is $p \in Q$, the output of the agent is $O(p)$. When an interaction between two agents happens, T determines the next states of the two agents, the initiator and the responder. For agent u with state p and agent v with state q, the equation $T(p,q) = (p',q')$ indicates that the states of u (the initiator) and v (the responder) become p' and q' respectively after the interaction (u,v).

A *configuration* is a mapping $C : V \to Q$ that specifies the states of all agents in a population. We denote by $\mathcal{C}_{\mathrm{all}}(P)$ the set of all configurations of P. Let C and C' be configurations, and let u and v be distinct agents. We say that C changes to C' by an interaction $r = (u,v)$, denoted by $C \xrightarrow{r} C'$, if we have $(C'(u), C'(v)) = T(C(u), C(v))$ and $C'(w) = C(w)$ for all $w \in V$ except u and v.[3]

An *interaction sequence* $\gamma = (u_0,v_0),(u_1,v_1),\dots$ is an infinite sequence of interactions. For each $t \geq 0$, we denote u_t and v_t by $\gamma_1(t)$ and $\gamma_2(t)$ respectively, and denote (u_t,v_t) by $\gamma(t)$. We call $\gamma(t)$ *the interaction at time t in γ*. We say that agent v *joins in* interaction $\gamma(t)$ when $v \in \{\gamma_1(t), \gamma_2(t)\}$.

An *execution* is a infinite sequence of configurations. Given an interaction sequence γ and an initial configuration C_0, the execution of protocol P is uniquely defined as $\Xi_P(C_0,\gamma) = C_0, C_1, \dots$ such that $C_t \xrightarrow{\gamma(t)} C_{t+1}$ for all $t \geq 0$.

A scheduler determines which interaction happens at each time t $(t \geq 0)$. In this chapter, we consider a uniformly random scheduler: the interaction at each time is chosen uniformly at random from all possible interactions. We represent the choice of this scheduler by the interaction sequence Γ: each $\Gamma(t)$ is a random variable such that $\Pr(\Gamma(t) = (u,v)) = 1/|E| = 1/(n(n-1))$ for any arbitrary interactions $(u,v) \in E$ and for any integer $t \geq 0$.

[2] This means that an agent can communicate simultaneously with only one agent.

[3] This definition implies that interactions between two agents happen sequentially, that is, exactly one pair of agents interact at any time.

2.2.1 Specification

In this section, we introduce the concept of *specification* and define the specification of leader election problem.

For protocol $P(Q, Y, O, T)$ and configuration $C \in \mathcal{C}_{\mathrm{all}}(P)$, we view the composite function $O \circ C : V \to Y$ as the output of C and denote it by $O(C)$. For a sequence of configurations $T = C_0, C_1, \dots$, we define output sequence $OT_P(T)$ as $O(C_0), O(C_1), \dots$.

A *specification* $SP(Y)$ is a set consisting of sequences of functions $V \to Y$ (We omit Y from the notation $SP(Y)$ when it is clear from the context). Let $\Xi = C_0, C_1, \dots$ be an execution of protocol P. We consider that execution Ξ satisfies specification SP if and only if $OT_P(\Xi) \in SP$ holds. When $OT_P(\mathrm{pre}_{t+1}(\Xi)) \in SP$ holds, Ξ is considered to satisfy SP until time t. (Note the index: $\mathrm{pre}_{t+1}(\Xi) = C_0, C_1, \dots, C_t$.) In this chapter, we assume that $X \in SP \Rightarrow \mathrm{pre}_l(X) \in SP$ holds for any specification SP and any positive integer l.

Definition 1 (Leader Election Problem)**.** *We denote by le the set of all assignment $\omega : V \to \{F, L\}$ such that for some $v_l \in V$, $\omega(v_l) = L$ and for all $v \neq v_l$, $\omega(v) = F$. The leader election specification $LE(\{F, L\})$ is defined as $LE(\{F, L\}) = \{T = w^k \mid w \in le,\ 1 \leq k \leq \infty\}$ where w^k is the sequence of consecutive assignments w with length k, that is,*

$$w^k = \underbrace{w, w, \dots, w}_{k}.$$

Informally, $LE\{F, L\}$ requires that any legitimate execution has one *static* leader agent with the output symbol L and $n-1$ non-leader (follower) agents with the output symbol F through its all configurations. Here, “static” means that the leader must continue to be a leader and any other agent must not become a leader during the execution.

This specification does not require termination detection because the population protocol model lacks the concept of termination. Since interactions happen infinite times, the execution continues forever and never terminates.

2.2.2 Loose-stabilization

In this section, we define the notion of *loose-stabilization.* For the proof of the impossibility result,

Firstly, we define *holding time* $HT_P(\Xi, SP)$ for protocol $P(Q, Y, O, \delta)$, execution Ξ of P and specification $SP(Y)$. This represents how long Ξ satisfies SP from time 0. If $OT_P(\Xi) \in SP$ holds, then we define $HT_P(\Xi, SP) = \infty$. If $OT_P(\mathrm{pre}_1(\Xi)) \notin SP$ holds, then we define $HT_P(\Xi, SP) = 0$. Otherwise, we have some t such that $OT_P(\mathrm{pre}_t(\Xi)) \in SP$ and $OT_P(\mathrm{pre}_{t+1}(\Xi)) \notin SP$. (Such t is unique.) Then we define $HT_P(\Xi, SP) = t$.

Secondly, we define *convergence time* $CT_P(\Xi, \mathcal{C})$ for a set $\mathcal{C} \subseteq \mathcal{C}_{\mathrm{all}}(P)$ of configurations. This represents how long it takes for $\Xi = C_0, C_1, \dots$ to reach a configuration in $\mathcal{C}$. If $C_0 \in \mathcal{C}$ holds, then we define $CT_P(\Xi, \mathcal{C}) = 0$. If $C_t \in \mathcal{C}$ does not hold for any time $t \geq 0$, then we define $CT_P(\Xi, \mathcal{C}) = \infty$. Otherwise, we have some t such that $C_t \notin \mathcal{C}$ and $C_t \in \mathcal{C}$. (Such t is unique.) Then, we define $CT_P(\Xi, \mathcal{C}) = t$.

We denote $\mathbf{E}[HT_P(\Xi_P(C, \Gamma), SP)]$ by $EHT_P(C, SP)$ for any configuration $C \in \mathcal{C}_{\mathrm{all}}(P)$, where $\mathbf{E}[X]$ denotes the expected value of random variable X. Similarly, we denote $\mathbf{E}[CT_P(\Xi_P(D, \Gamma), \mathcal{C})]$ by $ECT_P(D, \mathcal{C})$ for any configuration $D \in \mathcal{C}_{\mathrm{all}}(P)$.

Definition 2 (Loose-stabilization)**.** *Let α and β be real numbers. A protocol $P(Q, Y, O, T)$ is (α, β)-loosely-stabilizing for specification $SP(Y)$ if a nonempty set $\mathcal{S}$ of configurations exists such that:*

$$\max_{C \in \mathcal{C}_{\mathrm{all}}(P)} ECT_P(C, \mathcal{S}) \leq \alpha,$$

$$\min_{C \in \mathcal{S}} EHT_P(C, SP) \geq \beta\,.$$

Intuitively, loose-stabilization requires that any execution starting from any configuration reaches a loosely-safe configuration (i.e. a configuration in $\mathcal{S}$) within a short time, and after that, the execution satisfies the specification for a long time. An (α, β)-loosely-stabilizing protocol is quite useful if β is sufficiently large (e.g. exponential order with n) and α is relatively small (e.g. low polynomial order with n).

2.3 Protocol P_{LE}

In this section, the author presents a leader election protocol $P_{LE}(Q, \{F, L\}, O, T)$, which uses the knowledge of an upper bound N of the network size n. The protocol has a design parameter s. When s is adequately set depending on N, it is $(O(nN \log n), \Omega(Ne^N))$-loosely-stabilizing for LE.

Each agent has one *leader bit* and a *timer* that takes an integer value in $[0, s]$, i.e. $Q = \{-, l\} \times \{0, 1, \dots, s\}$. For state p, we denote the first element (leader bit) of p by $p.leader$ and the second element (timer) of p by $p.time$. The output function O is defined as follows: if the leader bit of an agent is l, then the output of the agent is L, otherwise F. We call an agent with the leader bit l ($-$) a leader (non-leader, respectively). We describe the transition function T by pattern rules in Code 2.1. Given any pair of states (p, q), the pair of the next states $T(p, q)$ is defined as follows: (i) if (p, q) matches the left side of exactly one rule, $T(p, q)$ is determined by

Code 2.1 Loosely-stabilizing Leader Election P_{LE}

```
R1.  $((l,*),(l,*))$  $\rightarrow$  $((l,s),(-,s))$
R2.  $((l,*),(-,*))$  $\rightarrow$  $((l,s),(-,s))$
R3.  $((-,*),(l,*))$  $\rightarrow$  $((-,s),(l,s))$
R4.  $((-,0),(-,0))$  $\rightarrow$  $((l,s),(-,s))$
R5.  $((-,i),(-,j))$  $\rightarrow$  $((-,f),(-,f))$
                    $(0 \le i,j \le s,\ f = \max(i,j)-1)$
```

the right side of the rule, and (ii) if there are two or more matched rules, we apply the rule with smallest rule number among them. The symbol $*$ means "don't care", that is, $*$ matches any value of the timer. Note that this five rules are collectively exhaustive.

If two leaders interact, the initiator remains a leader and the responder becomes a non-leader (R1). If a leader and a non-leader interact, the leader bits of both the agents do not change (R2, R3). In every interaction in which one or two leaders join, the timers of both the agents are reset to the full timer value s (R1, R2, and R3). We call this event *timer reset.* A new leader is created only when two non-leaders with timer value 0 interact (R4). We call this event *timeout.* If two non-leaders interact where either or both the agents have non-zero timer, then at least one of the two agents decrements its timer value by 1 (R5). R5 plays another role of *propagating the higher timer value*: intuitively, when two non-leaders interact, the timer of a lower value is set to the other (higher) value (minus 1).

In a configuration containing at least one leader, timeout rarely happens because of frequent occurrences of timer reset and propagation of higher timer value. On the other hand, in a configuration containing no leader, timeout happens in a relatively short time because of no possibility of timer reset. Hence, starting from any configuration, removing leaders by R1 or creating a leader by R4 eventually bring the population to a configuration with exactly one leader. The following two properties hold clearly.

Lemma 1. *Once a configuration with one or more leaders is reached, the number of leaders cannot become 0 thereafter.*

Lemma 2. *Once a unique leader is elected, specification LE holds until the next timeout happens.*

As a set of loosely-safe configuration, we adopt $\mathcal{S}_{\mathrm{half}}$, which consists of all the configurations in which exactly one leader exists and the timer value of every agent is greater than or equal to $s/2$. From the above explanation for P_{LE}, one can intuitively observe the following two properties:

starting from any configuration, the population reaches a configuration in $\mathcal{S}_{\text{half}}$ within a relatively short time (*convergence*), and once a configuration in $\mathcal{S}_{\text{half}}$ is reached, the specification (the unique and static leader) is kept for an extremely long time (*loose-closure*). In Section 2.4, the author rigorously proves how fast P_{LE} converges to a loosely-safe configuration, and how long P_{LE} maintains the specification of leader election after a loosely-safe configuration is reached.

2.4 Analysis and Proofs

Assume that we set design parameter s so that s is a multiple of 96 and $s \geq \max(3n, 96(2\ln n + \ln 24))$. (In what follows, we use the notation s^* for $s/96$.) In this section, we prove that under this assumption, P_{LE} is $(O(ns\log n), \Omega(se^{s^*}))$-loosely-stabilizing for LE and $\mathcal{S}_{\text{half}}$. To claim it, we prove the following two expressions:

$$\max_{C \in \mathcal{C}_{\text{all}}(P_{LE})} ECT_{P_{LE}}(C, \mathcal{S}_{\text{half}}) = O(ns\log n), \tag{2.1}$$

$$\min_{C \in \mathcal{S}_{\text{half}}} EHT_{P_{LE}}(C, LE) = \Omega\left(s \cdot e^{s^*}\right). \tag{2.2}$$

In this section, we omit P_{LE} from some expressions when the protocol under consideration is clear from the context; for example we denote $\mathcal{C}_{\text{all}}(P_{LE})$, $OT_{P_{LE}}$ and $ECT_{P_{LE}}$ simply by $\mathcal{C}_{\text{all}}$, OT and ECT respectively. And, we use the following four subsets $\mathcal{L}_{\text{one}}$, $\mathcal{L}$, $\mathcal{C}_{\text{half}}$ and $\mathcal{L}_{\text{half}}$ of $\mathcal{C}_{\text{all}}$ in addition to $\mathcal{S}_{\text{half}}$;

$$\begin{aligned}
&\mathcal{L}_{\text{one}} = \{C \in \mathcal{C}_{\text{all}} \mid \#l(C) = 1\}\,, \\
&\mathcal{L} = \{C \in \mathcal{C}_{\text{all}} \mid \#l(C) \geq 1\}\,, \\
&\mathcal{C}_{\text{half}} = \left\{C \in \mathcal{C}_{\text{all}} \mid \forall v \in V,\ C(v).\mathit{time} \geq \frac{s}{2}\right\}\,, \\
&\mathcal{L}_{\text{half}} = \mathcal{L} \cap \mathcal{C}_{\text{half}}\,,
\end{aligned}$$

where $\#l(C)$ represents the number of leaders in configuration C, i.e. $\#l(C) = |\{v \in V | v.\mathit{leader} = l\}|$. The set $\mathcal{L}_{\text{one}}$ represents the set of all configurations in which exactly one leader exists while $\mathcal{L}$ represents the set of all configurations in which one or more leaders exist. The set $\mathcal{C}_{\text{half}}$ represents the set of all configurations in which the timer value of every agent is greater than or equal to $s/2$. Note that $\mathcal{S}_{\text{half}}$ is equal to $\mathcal{L}_{\text{one}} \cap \mathcal{C}_{\text{half}}$.

In the rest of this section, we introduce the notion of *epidemic* (presented in [6]) and *virtual agents* in Section 2.4.1. Using these tools, we prove (2.2) in Section 2.4.2 and (2.1) in Section 2.4.3.

2.4.1 Epidemic and Virtual Agents

In this section, we introduce the notion of *epidemic* (presented in [6]) and *virtual agents.*

To begin with, the notion of epidemic is introduced. Let C_0 be a configuration in $\mathcal{L}_{\mathrm{one}}$, and let $v_l \in V$ be the unique leader in C_0. Let γ be an interaction sequence. The *epidemic function* $I_{C_0,\gamma}(t)$ ($t = 0, 1, \dots$) that returns a set of agents is defined as follows: $I_{C_0,\gamma}(0) = \{v_l\}$, and $I_{C_0,\gamma}(t) = I_{C_0,\gamma}(t-1) \cup Add_{C_0,\gamma}(t-1)$ for any $t \geq 1$ where we define

$$Add_{C_0,\gamma}(i) = \begin{cases} \{\gamma_1(i), \gamma_2(i)\} & \text{if } \gamma_1(i) \in I_{C_0,\gamma}(i) \ \vee \ \gamma_2(i) \in I_{C_0,\gamma}(i) \\ \emptyset & \text{otherwise} \end{cases}$$

for any integer $i \geq 0$. We say that if $v \in I_{C_0,\gamma}(t)$, then v is *infected* at time t in the epidemic starting from C_0 and spreading under γ, otherwise v is *infection-free* at time t in the epidemic. At time 0, only v_l is infected. An infection-free agent becomes infected when it interacts with an infected agent. Once an agent becomes infected, it remains infected thereafter. We define the *infected time* $T_{C_0,\gamma}(v)$ of agent $v \neq v_l$ as an integer $i \geq 0$ that satisfies $v \notin I_{C_0,\gamma}(i)$ and $v \in I_{C_0,\gamma}(i+1)$. We naturally define $T_{C_0,\gamma}(v_l) = 0$.

In the following, we define the *virtual agent* $VA_{C_0,\gamma}(v)$ of each agent $v \in V$. We assume that all the agents eventually become infected, that is, $I_{C_0,\gamma}(t') = V$ holds for some $t' \geq 0$. The virtual agent $VA_{C_0,\gamma}(v)$ is not defined if no such t' exists for C_0 and γ. Let v be any agent other than v_l. We define the *parent* of v as the agent that infects v in time $T_{C_0,\gamma}(v)$. This parent-child relation makes the rooted spanning tree uniquely, the root of which is v_l. In this tree, the path from v_l to v, $v_l = w_0 \to w_1 \to w_2 \to \cdots \to w_m = v$, uniquely exists. The virtual agent $VA_{C_0,\gamma}(v)$ is a virtual entity that migrates from v_l to v through the path. This notion is formalized as *the location of the virtual agent* $L_{C_0,\gamma}(v,t)$ ($t \geq 0$), which is defined as follows:

$$L_{C_0,\gamma}(v,t) = \begin{cases} v_l & (0 \leq t \leq t_1) \\ w_i & (t_i + 1 \leq t \leq t_{i+1}, \ 1 \leq i \leq m-1) \\ v & (t \geq t_m + 1 = T_{C_0,\gamma}(v) + 1), \end{cases}$$

where $t_i = T_{C_0,\gamma}(w_i)$. For the leader agent v_l, we define $L_{C_0,\gamma}(v_l, t) = v_l$ for any $t \geq 0$.

Let v be an agent in V.[4] For simplicity, we denote the virtual agent $VA_{C_0,\gamma}(v)$ by v' here. We say that the virtual agent v' joins in interaction $\gamma(t)$ if agent $L_{C_0,\gamma}(v,t)$ joins in $\gamma(t)$, and we define indicator variable $VJ_{C_0,\gamma}(v,t)$ for any $t \geq 0$ as follows: if v' joins in $\gamma(t)$, then $VJ_{C_0,\gamma}(v,t) = 1$,

[4]Note that v can be v_l.

otherwise $VJ_{C_0,\gamma}(v,t) = 0$. *The number of virtual interactions* of v, denoted by $VI_{C_0,\gamma}(v,t)$, is defined as follows:

$$VI_{C_0,\gamma}(v,t) = \sum_{i=0}^{t-1} VJ_{C_0,\gamma}(v,i) \ .$$

Intuitively, $VI_{C_0,\gamma}(v,t)$ is the number of interactions in which v' joins between time 0 and time $t-1$.

One can observe that the virtual agent v' brings a large timer value to v with high probability when v' reaches v through the infecting path. Actually, the timer of v at time $T_{C_0,\gamma}(v)+1$ (just after v is infected) is at least $s - VI_{C_0,\gamma}(v, T_{C_0,\gamma}(v)+1)$. The following lemma trivially holds.

Lemma 3. *Let C_0 be a configuration in $\mathcal{L}_{\text{one}}$ and let γ be an interaction sequence. Let $\Xi(C_0,\gamma) = C_0, C_1, \ldots$. The following predicate holds for any integer $t \geq 0$:*

$$I_{C_0,\gamma}(t) = V \quad \Rightarrow \quad \forall v \in V,\ C_t(v).time \geq s - VI_{C_0,\gamma}(v,t).$$

If timeout happens, another new leader is created. This leader may change v_l to be a non-leader. Note that the above lemma holds even if such a situation happens, and v_l becomes a non-leader.

In addition to the number of virtual interactions, we define *the number of real interactions* of v as $RI_\gamma(v,t) = \sum_{i=0}^{t-1} RJ_\gamma(v,t)$, where $RJ_\gamma(v,t)$ is indicator variable such that if v joins in $\gamma(t)$ then $RJ_\gamma(v,t) = 1$, otherwise $RJ_\gamma(v,t) = 0$. Intuitively, $RI_\gamma(v,t)$ is the number of interactions in which v joins between time 0 and time $t-1$.

2.4.2 Expected Holding Time

In this section, we prove (2.2). For a positive integer t, a configuration $C_0 \in \mathcal{C}_{\text{all}}$ and an interaction sequence γ, we define indicator variable $TO_{C_0,\gamma}(t)$ as follows: if timeout happens in execution $\Xi_{P_{LE}}(C_0,\gamma)$ between time 0 and time $t-1$, that is, at least one of the interactions $\gamma(0), \gamma(1), \ldots, \gamma(t-1)$ causes timeout in $\Xi(C_0,\gamma)$, then $TO_{C_0,\gamma}(t) = 1$, otherwise $TO_{C_0,\gamma}(t) = 0$. For convenience, we define $TO_{C_0,\gamma}(0) = 0$. As a sufficient condition for (2.2), we focus on the following inequality for any configuration C_0 in $\mathcal{S}_{\text{half}}$:

$$\Pr\left(TO_{C_0,\Gamma}(2ns^*) = 0 \ \wedge\ C_{2ns^*} \in \mathcal{S}_{\text{half}}\right) \geq 1 - 2n \cdot e^{-s^*}, \tag{2.3}$$

where $\Xi(C_0,\Gamma) = C_0, C_1, \ldots, C_{2ns^*}, \ldots$.

Lemma 4. *Expression* (2.2) *holds if* (2.3) *holds for any configuration C_0 in $\mathcal{S}_{\text{half}}$.*

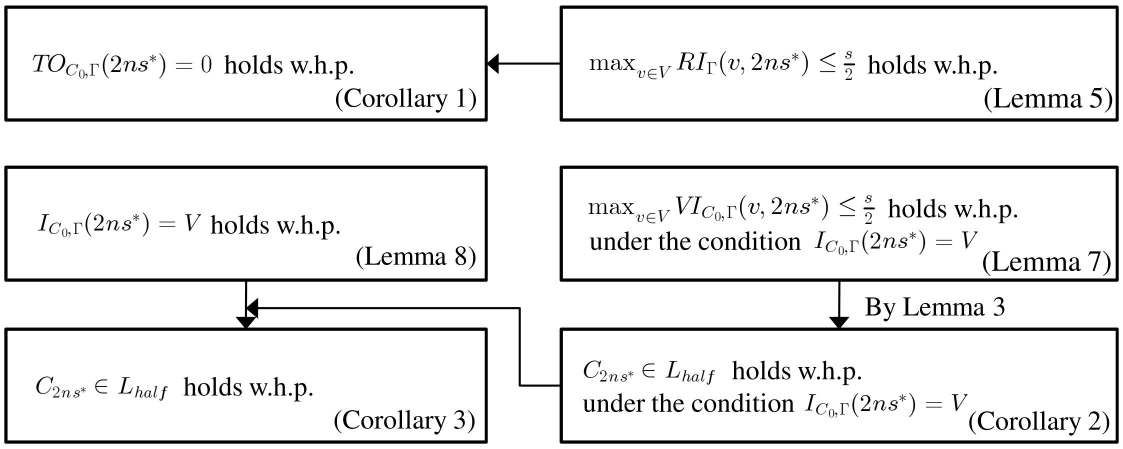

$TO_{C_0,\Gamma}(2ns^*) = 0$ holds w.h.p.
(Corollary 1)
$\max_{v \in V} RI_\Gamma(v, 2ns^*) \leq \frac{s}{2}$ holds w.h.p.
(Lemma 5)
$I_{C_0,\Gamma}(2ns^*) = V$ holds w.h.p.
(Lemma 8)
$\max_{v \in V} VI_{C_0,\Gamma}(v, 2ns^*) \leq \frac{s}{2}$ holds w.h.p.
under the condition $I_{C_0,\Gamma}(2ns^*) = V$
(Lemma 7)
By Lemma 3
$C_{2ns^*} \in L_{half}$ holds w.h.p.
(Corollary 3)
$C_{2ns^*} \in L_{half}$ holds w.h.p.
under the condition $I_{C_0,\Gamma}(2ns^*) = V$
(Corollary 2)

Figure 2.1: The structure of the proof for (2.3): w.h.p. means "with high probability".

Proof . Assume that (2.3) holds for any configuration C_0 in $\mathcal{S}_{\mathrm{half}}$. Then, from Lemma 2, the following inequality holds:

$$EHT(C_0, LE) \geq \left(1 - 2n \cdot e^{-s^*}\right)\left(2ns^* + \min_{C \in \mathcal{S}_{\mathrm{half}}} EHT(C, LE)\right) \quad .$$

Since C_0 is *any* configuration in $\mathcal{S}_{\mathrm{half}}$, we have

$$\min_{C \in \mathcal{S}_{\mathrm{half}}} EHT(C, LE) \geq \left(1 - 2n \cdot e^{-s^*}\right)\left(2ns^* + \min_{C \in \mathcal{S}_{\mathrm{half}}} EHT(C, LE)\right) \quad .$$

Solving this inequality gives us (2.2). □

In the following, we show that (2.3) holds for any configuration $C_0 \in \mathcal{S}_{\mathrm{half}}$. The structure of the proof is shown in Figure 2.1. Firstly, we bound the number of real interactions probabilistically (Lemma 5), from which we get the result that the probability of $TO_{C_0,\Gamma}(2ns^*) = 0$ is sufficiently close to 1 (Corollary 1). Secondly, we bounds the number of virtual interaction probabilistically (Lemma 7), from which and Lemma 3 together, we prove that $C_{2ns^*} \in \mathcal{L}_{\mathrm{half}}$ holds with sufficiently high probability under the condition $I_{C_0,\Gamma}(2ns^*) = V$ (Corollary 2). We also prove that $I_{C_0,\Gamma}(2ns^*) = V$ holds with sufficiently high probability (Lemma 8). By Corollary 2 and Lemma 8, we get the result that the probability of $C_{2ns^*} \in \mathcal{L}_{\mathrm{half}}$ is sufficiently close to 1 (Corollary 3). The combination of Corollaries 1 and 3 directly leads to (2.3).

Lemma 5. $\Pr(\max_{v \in V} RI_\Gamma(v, 2ns^*) \leq \frac{s}{2}) \geq 1 - n \cdot \exp(-16s^*)$.

Proof . At each time t, any agent v joins in $\Gamma(t)$ with probability $\frac{2}{n}$. Hence, $RI_\Gamma(v, 2ns^*) \sim B(2ns^*, \frac{2}{n})$. As one of Chernoff bounds, $\Pr(Y \geq R) \leq 2^{-R}$ holds for any binomial random variable Y and any real number $R \geq 6 \cdot \mathbf{E}[Y]$ [13, (4.3)]. Since $\frac{s}{2} \geq 6\mathbf{E}[RI_\Gamma(v, 2ns^*)] = \frac{s}{4}$ and $2^{-1/2} < e^{-1/4}$ [5], we obtain

$$\Pr\left(RI_\Gamma\left(v, 2ns^*\right) \geq \frac{s}{2}\right) \leq 2^{-s/2} < \exp\left(-\frac{s}{4}\right) = \exp(-16s^*)\,.$$

We achieve the lemma by summing up all the above probabilities with respect to $v \in V$. □

Corollary 1. *The inequality* $\Pr(TO_{C_0,\Gamma}(2ns^*) = 0) \geq 1 - n \cdot \exp(-16s^*)$ *holds for any configuration* C_0 *in* $\mathcal{C}_{\text{half}}$.

Proof . Since $C_0 \in \mathcal{C}_{\text{half}}$, timeout happens by time $2ns^* - 1$ (i.e. $TO_{C_0,\Gamma}(2ns^*) = 1$) only when some agent joins in at least $\frac{s}{2} + 1$ interactions between time 0 and time $2ns^* - 1$. Hence, the corollary follows from Lemma 5. □

Next, in Lemma 7, we bounds the number of virtual interactions probabilistically. Apparently, it seems that $VJ_{C_0,\Gamma}(v, t)$ and $RJ_\Gamma(v, t)$ have the same probability distribution for any $v \in V$ and $t \geq 0$. However, this is not true. Surprisingly, the interaction at time t influence the location of virtual agent of v at the same time t. Hence, $\Pr(VJ_{C_0,\Gamma}(v, t) = 1) = \Pr(L_{C_0,\Gamma}(v, t) \in \{\Gamma_1(t), \Gamma_2(t)\})$ is not equal to $\frac{2}{n}$ and very hard to calculate. Therefore, we must take a different approach for Lemma 7 from that of Lemma 5. To begin with, we introduce the following lemma as a tool.

Lemma 6. *Let* C_0 *be a configuration in* $\mathcal{L}_{\text{one}}$ *and let* $X(i, p)$ *be a binomial random variable such that* $X(i, p) \sim B(i, p)$. *Then, the following expression holds for any* $v \in V$ *and any integers* $t \geq n$ *and* $j \geq 0$*:*

$$\Pr(VI_{C_0,\Gamma}(v, t) \geq j + n - 1 \mid I_{C_0,\Gamma}(t) = V) \leq \Pr(X(t, 4/n) \geq j)\,.$$

Proof . Assume $I_{C_0\Gamma}(t) = V$ and let $v_l \in V$ be the unique leader in C_0. We define the *infecting time set* IT as $\bigcup_{v \in V \setminus \{v_l\}} \{T_{C_0,\Gamma}(v)\}$, and the *non-infecting time set* NIT as $\{0, 1, \ldots, t-1\} \setminus IT$. Let v be any agent in V, and let $NVI = \sum_{t' \in NIT} VJ_{C_0,\Gamma}(v, t')$. Since $|IT| = n-1$, the inequality $VI_{C_0,\Gamma}(v, t) \leq NVI + n - 1$ immediately follows. Therefore, it is sufficient for our proof to show $\Pr(NVI \geq j \mid I_{C_0,\Gamma}(t) = V) \leq \Pr(X(t, 4/n) \geq j)$.

Let t' be any integer in $\{0, 1, \ldots, t-1\}$ and let $m = |I_{C_0,\Gamma}(t')|$. Consider the case $t' \in NIT$. Then, the interaction $\Gamma(t')$ must be an interaction such that both agents $\Gamma_1(t')$ and $\Gamma_2(t')$ belong to $I_{C_0,\Gamma}(t')$ or both the agents belong to $V \setminus I_{C_0,\Gamma}(t')$. Otherwise, some infection-free agent

[5] $2^{-1/2} \approx 0.707$ and $e^{-1/4} \approx 0.778$

becomes infected at time t', contradicting $t' \notin IT$. And, $L_{C_0,\Gamma}(v,t') \in I_{C_0,\Gamma}(t')$ clearly holds by the definition of virtual agents. Thus, letting ${}_0C_2 = {}_1C_2 = 0$, we have

$$\begin{aligned}\Pr(VJ_{C_0,\Gamma}(v,t') = 1 \mid I_{C_0,\Gamma}(t) = V \ \wedge\ t' \in NIT) &= \frac{m-1}{{}_mC_2 +_{n-m} C_2} \\ &\le \frac{4}{n}\ .\end{aligned}$$

See Lemma 15 in Section 2.5 for the last inequality.

Note that this upper bound $4/n$ of the probability is independent from any interaction at any time other than t'. Hence, for any set S consisting of $t-n+1$ distinct integers in $[0,t-1]$, we have

$$\Pr\left(NVI \ge j \mid I_{C_0,\Gamma}(t) = V \ \wedge\ NIT = S\right) \le \Pr\left(X\left(t-n+1, \frac{4}{n}\right) \ge j\right)\ .$$

Therefore, the following inequality holds and so does the lemma.

$$\begin{aligned}\Pr(NVI \ge j \mid I_{C_0,\Gamma}(t) = V) &\le \Pr\left(X\left(t-n+1, \frac{4}{n}\right) \ge j\right) \\ &\le \Pr\left(X\left(t, \frac{4}{n}\right) \ge j\right)\ .\end{aligned}$$

□

Lemma 7. *Let C_0 be a configuration in $\mathcal{L}_{\text{one}}$. The following inequality holds:*

$$\Pr\left(\max_{v\in V} VI_{C_0,\Gamma}(v, 2ns^*) \le \frac{s}{2} \ \mid\ I_{C_0,\Gamma}(2ns^*) = V\right) \ge 1 - n\cdot\exp\left(-\frac{8s^*}{3}\right). \qquad (2.4)$$

Proof . Let v be any agent, and let $X(i,p)$ be an binomial variable such that $X(i,p) \sim B(i,p)$. By Lemma 6 and the assumption $s \ge 3n$, we have

$$\begin{aligned}&\Pr\left(VI_{C_0,\Gamma}(v, 2ns^*) \ge \frac{s}{2} \mid I_{C_0,\Gamma}(2ns^*) = V\right) \\ \le& \Pr\left(VI_{C_0,\Gamma}(v, 2ns^*) \ge \frac{s}{6} + n - 1 \mid I_{C_0,\Gamma}(2ns^*) = V\right) \qquad \because \frac{s}{2} \ge \frac{s}{6} + n - 1 \\ \le& \Pr\left(X\left(2ns^*, \frac{4}{n}\right) \ge \frac{s}{6}\right)\ .\end{aligned}$$

As one of Chernoff bounds, $\Pr(Y \ge (1+\epsilon)\mathbf{E}[Y]) \le \exp(-\epsilon^2\mathbf{E}[Y]/3)$ holds for any binomial random variable Y and any real number ϵ $(0 \le \epsilon \le 1)$ [13, (4.2)]. It follows that $\Pr(X(2ns^*, 4/n) \ge 16s^*) \le \exp(-8s^*/3)$. (We set $\epsilon = 1$.) We obtain (2.4) by summing up all the probabilities with respect to $v \in V$. □

The following corollary is directly obtained from Lemmas 3 and Lemma 7.

Corollary 2. *Let C_0 be a configuration in $\mathcal{L}_{\mathrm{one}}$ and let $\Xi(C_0,\Gamma) = C_0, C_1, \dots$. Then,* $\Pr(C_{2ns^*} \in \mathcal{L}_{\mathrm{half}} \mid I_{C_0,\Gamma}(2ns^*) = V) \geq 1 - n \cdot \exp(-8s^*/3)$ *holds.*

Lemma 8. $\Pr(I_{C_0,\Gamma}(2ns^*) = V) \geq 1 - n \cdot \exp(-s^*)$ *holds for any $C_0 \in \mathcal{L}_{\mathrm{one}}$.*

Proof . For each k $(2 \leq k \leq n)$, we define $T(k)$ as integer t such that $|I_{C_0,\Gamma}(t-1)| = k-1$ and $|I_{C_0,\Gamma}(t)| = k$, and define $T(1) = 0$. Intuitively, $T(k)$ is the first time at which there exists k infected agents in the population. Let $X_{\mathrm{pre}} = T(\lceil \frac{n+1}{2} \rceil)$ and $X_{\mathrm{post}} = T(n) - T(n - \lceil \frac{n+1}{2} \rceil + 1)$. Angluin et al. found in [6] that $T(k)$ and $T(n) - T(n-k+1)$ have the same probability distribution for any k $(1 \leq k \leq n)$. Hence, so do X_{pre} and X_{post}. And, $X_{\mathrm{pre}} + X_{\mathrm{post}} \geq T(n)$ holds because $\lceil \frac{n+1}{2} \rceil \geq n - \lceil \frac{n+1}{2} \rceil + 1$. We denote $T(n - \lceil \frac{n+1}{2} \rceil + 1)$ by T_{half} and let $X_v = \max(T_{C_0,\Gamma}(v) - T_{\mathrm{half}}, 0)$ for any agent v. Informally, X_v is the number of interactions that occurs between time T_{half} and the time at which agent v becomes infected. Consider the case $v \notin I_{C_0,\Gamma}(T_{\mathrm{half}})$. At any time $t \geq T_{\mathrm{half}}$, at least $n - \lceil \frac{n+1}{2} \rceil + 1$ $(\geq \frac{n}{2})$ agents are infected. Therefore, each interaction at time $t \geq T_{\mathrm{half}}$ infects v with the probability of at least $\frac{1}{{}_nC_2} \cdot \frac{n}{2} \geq \frac{1}{n}$, and hence, we have $\Pr(X_v > ns^*) \leq (1 - \frac{1}{n})^{ns^*} \leq e^{-s^*}$. Since the number of infection-free agent at time T_{half} is at most $\frac{n}{2}$, $\Pr(X_{\mathrm{post}} > ns^*) \leq \Pr(\bigvee_{v \in V}(X_v \geq ns^*)) \leq \frac{n}{2} \cdot \exp(-s^*)$ holds. By the equivalence of the distribution of X_{pre} and X_{post}, we have

$$\Pr\left(I_{C_0,\Gamma}(2ns^*) \neq V\right) \leq \Pr\left(X_{\mathrm{pre}} > ns^*\right) + \Pr\left(X_{\mathrm{post}} > ns^*\right) \leq n \cdot e^{-s^*} \quad .$$

□

Corollary 2 and Lemma 8 together lead to the following corollary.

Corollary 3. *Let C_0 be a configuration in $\mathcal{L}_{\mathrm{one}}$ and let $\Xi(C_0,\Gamma) = C_0, C_1, \dots,$ $C_{2ns^*}, \dots$. Then,* $\Pr(C_{2ns^*} \in \mathcal{L}_{\mathrm{half}}) \geq 1 - n \cdot \exp(-8s^*/3) - n \cdot \exp(-s^*)$ *holds.*

Theorem 1. *Expression* (2.2) *holds.*

Proof . By the assumption $s \geq 96(2 \ln n + \ln 24)$, we have $\exp(-\frac{s}{4}) + \exp(-\frac{s}{36}) \leq \exp(-\frac{s}{96})$. Therefore, (2.3) holds for any configuration $C_0 \in \mathcal{S}_{\mathrm{half}}$ from Corollaries 3 and 1. We achieve (2.2) by Lemma 4. □

2.4.3 Expected Convergence Time

Next, we show (2.1) to complete our proof. The following inequality clearly holds:

$$\begin{aligned} &\max_{C \in \mathcal{C}_{\mathrm{all}}} ECT(C, \mathcal{S}_{\mathrm{half}}) \\ &\qquad \leq \max_{C \in \mathcal{C}_{\mathrm{all}}} ECT(C, \mathcal{L}) + \max_{C \in \mathcal{L}} ECT(C, \mathcal{L}_{\mathrm{half}}) + \max_{C \in \mathcal{L}_{\mathrm{half}}} ECT(C, \mathcal{S}_{\mathrm{half}}). \end{aligned} \tag{2.5}$$

Therefore, it suffices to show that each term in the right side of (2.5) belongs to $O(ns \log n)$. We show that the first term belongs to $O(ns \log n)$ in Lemma 9, and the second and the third term belongs to $O(ns)$ in Lemma 10 and Lemma 14 respectively.

Lemma 9. $\max_{C \in \mathcal{C}_{\mathrm{all}}} ECT(C, \mathcal{L})$ *belongs to* $O(ns \log n)$.

Proof . We define $\nu(C, i)$ $(0 \le i \le s)$ as the number of agents with timer value i in configuration C, i.e. $\nu(C, i) = |\{v \in V \mid C(v).\mathit{time} = i\}|$. For any integer i, j $(0 \le i \le s,\ 1 \le j \le n)$ we denote by $\mathcal{W}_{i,j}$ the set of all configurations in which there exists no leader, the maximum timer value of all agents is i, and $\nu(C, i) = j$ holds. Note that $\mathcal{W}_{0,j}$ is empty when $j \ne n$.

Let $w_{i,j}$ be $\max_{C \in \mathcal{W}_{i,j}} ECT(C, \mathcal{C}_{\mathrm{all}} \setminus \mathcal{W}_{i,j})$. The execution get out of $\mathcal{W}_{i,j}$ if one of j agents with the largest timer value joins in an interaction. Furthermore, after that, the execution never comes back to $\mathcal{W}_{i,j}$ again. Since one of j agent is selected with probability $\frac{j(j-1)+2j(n-j)}{n(n-1)} \ge \frac{j}{n}$, we have $w_{i,j} \le \frac{n}{j}$, and thus

$$\max_{C \in C_{\mathrm{all}}} ECT(C, \mathcal{L}) \le w_{0,n} + \sum_{i=1}^{s} \sum_{j=1}^{n} w_{i,j} \le 1 + ns \cdot H(n) = O(ns \log n)$$

holds where H is the harmonic function. □

In Section 2.4.1, the definitions of epidemic and virtual agents stand on the assumption that there exists *exactly* one leader in the initial configuration C_0 (i.e. $C_0 \in \mathcal{L}_{\mathrm{one}}$). However, this assumption can be relaxed as follows: there exists *at least* one leader in C_0 (i.e. $C_0 \in \mathcal{L}$). With defining v_l as any arbitrary leader in C_0, we can redefine $I_{C_0,\gamma}(t)$ and $VI_{C_0,\gamma}(v, t)$ for any $C_0 \in \mathcal{L}$ in the same manner as Section 2.4.1. Then, Corollary 3 holds not only for any $C_0 \in \mathcal{L}_{\mathrm{one}}$ but also for any $C_0 \in \mathcal{L}$.

Lemma 10. $\max_{C \in \mathcal{L}} ECT(C, \mathcal{L}_{\mathrm{half}})$ *belongs to* $O(ns)$.

Proof . Let C_0 be an configuration in $\mathcal{L}$ and let $\Xi(C_0, \Gamma) = C_0, C_1, \ldots$. By Corollary 3, we have

$$\begin{aligned}\Pr\left(C_{2ns^*} \in \mathcal{L}_{\mathrm{half}}\right) &\ge 1 - 2n \cdot \exp\left(-s^*\right) \\ &\ge 1 - \frac{2n}{24n^2} \qquad \because s^* \ge 2\ln n + \ln 24 \\ &\ge \frac{1}{2} \ .\end{aligned}$$

Since $C_{2ns^*} \in \mathcal{L}$ holds by Lemma 1, we have

$$\max_{C \in \mathcal{L}} ECT(C, \mathcal{L}_{\mathrm{half}}) \le 2ns^* + \frac{1}{2} \max_{C \in \mathcal{L}} ECT(C, \mathcal{L}_{\mathrm{half}}).$$

Solving this inequality gives us $\max_{C \in \mathcal{L}} ECT(C, \mathcal{L}_{\mathrm{half}}) \le 4ns^* = O(ns)$. □

In the rest of this section, we prove Lemma 14. Informally, the lemma holds for the following reason.

- When we can ignore occurrence of timeout, at most n leaders kill each other, and one unique leader is elected within $O(n^2)$ expected interactions. (We shall see this fact in Lemma 12.)
- From Corollaries 1 and 3, one can observe that timeout happens only extremely rarely when the execution starts from a configuration in $\mathcal{L}_{\mathrm{half}}$. Hence, the probability that timeout happens by time $O(n^2)$ is negligible. Furthermore, although the execution may get out of $\mathcal{L}_{\mathrm{half}}$, it comes back to $\mathcal{L}_{\mathrm{half}}$ by time $O(ns)$ with very high probability (Corollary 3).

In what follows, we show the formal proof. To avoid complicated analysis of conditional probability, we introduced a protocol P'_{LE}, which eliminate timeout mechanism from P_{LE}. Specifically, P'_{LE} is the protocol obtained from P_{LE} by replacing rule R4 in the transition function T with the following rule R4':

$$\text{R4'} \qquad ((-,0),(-,0)) \;\to\; ((-,0),(-,0)) \;.$$

The state set of P_{LE} and P'_{LE} is identical, and hence, so do the $\mathcal{C}_{\mathrm{all}}(P_{LE})$ and $\mathcal{C}_{\mathrm{all}}(P'_{LE})$.

Lemma 11. *Let C_0 be a configuration in $\mathcal{C}_{\mathrm{all}}$ and let γ be an interaction sequence. Let $\Xi_{P_{LE}}(C_0,\gamma) = C_0, C_1, \dots$ and $\Xi_{P'_{LE}}(C_0,\gamma) = C_0, D_1, D_2, \dots$. The following predicate holds for any $t \ge 0$:*

$$D_t \in \mathcal{L}_{\mathrm{one}} \wedge C_t \in \mathcal{C}_{\mathrm{half}} \wedge TO_{C_0,\gamma}(t) = 0 \quad \Rightarrow \quad C_t \in \mathcal{S}_{\mathrm{half}}.$$

Proof . Assume that $D_t \in \mathcal{L}_{\mathrm{one}}$, $C_t \in \mathcal{C}_{\mathrm{half}}$ and $TO_{C_0,\gamma}(t) = 0$ holds. Note that executions $\Xi_{P_{LE}}(C_0,\gamma)$ and $\Xi_{P'_{LE}}(C_0,\gamma)$ have no difference until timeout happens. By the assumption $TO_{C_0,\gamma}(t) = 0$, we have the equality $C_t = D_t$, and hence, $C_t \in \mathcal{L}_{\mathrm{one}} \cap \mathcal{C}_{\mathrm{half}} = \mathcal{S}_{\mathrm{half}}$ holds. □

Lemma 12. $\max_{C\in\mathcal{L}} ECT_{P'_{LE}}(C,\mathcal{L}_{\mathrm{one}}) = (n-1)^2$.

Proof . We prove this lemma in the almost same way as [5]. The number of leaders decrease by 1 when two leaders have an interaction. In each interaction, two of i leaders have an interaction with probability $\frac{{}_iC_2}{{}_nC_2}$. Hence, we have

$$\max_{C\in\mathcal{L}} ECT_{P'_{LE}}(C,\mathcal{L}_{\mathrm{one}}) = \sum_{i=2}^{n} \frac{{}_nC_2}{{}_iC_2} = (n-1)^2$$

□

In what follows, we use an integer $r = \lceil \frac{3n^2}{2ns^*} \rceil \cdot 2ns^*$.

Lemma 13. *Let C_0 be an configuration in $\mathcal{L}_{\mathrm{half}}$ and let $\Xi_{P_{LE}}(C_0,\Gamma) = C_0, C_1, \dots$. Then, the following inequality holds:*

$$\Pr(C_r \in \mathcal{S}_{\mathrm{half}}) \geq \frac{1}{2}.$$

Proof . Let $\Xi_{P'_{LE}}(C_0,\Gamma) = C_0, D_1, D_2, \dots$. By Lemma 11, it suffices to show $\Pr(D_r \in \mathcal{L}_{\mathrm{one}} \wedge C_r \in \mathcal{C}_{\mathrm{half}} \wedge TO_{C_0,\Gamma}(r) = 0) \geq \frac{1}{2}$. By lemma 12 and Markov's inequality, we have

$$\begin{aligned}\Pr(D_r \notin \mathcal{L}_{\mathrm{one}}) &= \Pr(CT(\Xi_{P'_{LE}}(C_0,\Gamma),\mathcal{L}_{\mathrm{one}}) > r)\\ &\leq \frac{ECT_{P'_{LE}}(C_0,\Gamma)}{r} \leq \frac{(n-1)^2}{3n^2} \leq \frac{1}{3}\,.\end{aligned}$$

Next, we show a lower bound of $\Pr(C_r \in \mathcal{C}_{\mathrm{half}} \wedge TO_{C_0,\Gamma}(r) = 0)$. By Lemma 5 and Corollary 3, we have the following inequality:

$$\Pr\left(C_{2ns^*} \in \mathcal{L}_{\mathrm{half}} \wedge TO_{C_0,\Gamma}(2ns^*) = 0\right) \geq 1 - 2n \cdot e^{-s^*}\ .$$

Hence, we have

$$\begin{aligned}\Pr\left(C_r \in \mathcal{L}_{\mathrm{half}} \wedge TO_{C_0,\Gamma}\left(r\right) = 0\right) &\geq \left(1 - 2n \cdot e^{-s^*}\right)^{\lceil 3n^2/2ns^* \rceil}\\ &\geq 1 - 2n \cdot e^{-s^*} \cdot \left\lceil \frac{3n^2}{2ns} \right\rceil \qquad \text{(see Lemma 16 in Section 2.5)}\\ &\geq 1 - \frac{1}{12n}\left(\frac{3n}{2s^*} + 1\right) \geq 1 - \frac{1}{12}(1+1) = \frac{5}{6}\ ,\end{aligned}$$

where we use the assumption $s^* \geq 2\ln n + \ln 24$ for the third inequality. Thus, we have $\Pr(D_r \in \mathcal{L}_{\mathrm{one}} \wedge C_r \in \mathcal{C}_{\mathrm{half}} \wedge TO_{C_0,\Gamma}(r) = 0) \geq 1 - (\frac{1}{3} + \frac{1}{6}) = \frac{1}{2}$. □

Lemma 14. $\max_{C \in \mathcal{L}_{\mathrm{half}}} ECT(C, \mathcal{S}_{\mathrm{half}})$ *belongs to $O(ns)$.*

Proof . Let C_0 be an configuration in $\mathcal{L}_{\mathrm{half}}$ and let $\Xi_{P_{LE}} = C_0, C_1, \dots$. By Lemmas 1 and 13, we have

$$\max_{C \in \mathcal{L}_{\mathrm{half}}} ECT(C, \mathcal{S}_{\mathrm{half}}) \leq r + \frac{1}{2}\left(\max_{C \in \mathcal{L}} ECT(C, \mathcal{L}_{\mathrm{half}}) + \max_{C \in \mathcal{L}_{\mathrm{half}}} ECT(C, \mathcal{S}_{\mathrm{half}})\right).$$

Solving this inequality gives $\max_{C \in \mathcal{L}_{\mathrm{half}}} ECT(C, \mathcal{S}_{\mathrm{half}}) \leq 6n^2 + 8ns^* = O(ns)$. since we have $\max_{C \in \mathcal{L}} ECT(C, \mathcal{L}_{\mathrm{half}}) \leq 4ns^*$ in the proof of Lemma 10. □

Thus, we obtain (2.1) from Lemmas 9, 10, 14 and (2.5). The following theorem is directly derived from (2.1) and (2.2).

Theorem 2. *P_{LE} is $(O(ns\log n), \Omega(se^{s/96}))$-loosely-stabilizing for behavior LE and $\mathcal{S}_{\mathrm{half}}$ if $s \geq \max(3n, 96(2\ln n + \ln 24))$ holds and s is a multiple of 96.*

Recall that P_{LE} knows an upper bound N of n. When we set s to be $\max(96N, 96(2\ln N + \ln 24))$, P_{LE} realize $(O(nN\log n), \Omega(Ne^N))$-loose-stabilization for behavior LE and $\mathcal{S}_{\mathrm{half}}$. That is, P_{LE} realizes fast convergence to a loosely-safe configuration (low polynomial order time) and extremely long maintenance of its specification (exponential order time).

2.5 Complementary Lemmas

Lemma 15. *Let n and m be integers $(0 \le m \le n)$ and let ${}_0C_2 = {}_1C_2 = 0$. Then, the following inequality holds:*

$$\frac{m-1}{{}_mC_2 +{}_{n-m} C_2} \le \frac{4}{n}. \tag{2.6}$$

Proof . If $m \ge \frac{n}{2}$, then (2.6) follows from $\frac{m-1}{{}_mC_2+{}_{n-m}C_2} \le \frac{m-1}{{}_mC_2} = \frac{2}{m} \le \frac{4}{n}$. If $m < \frac{n}{2}$, then $n-m-1 > m-1$ holds. Thus, the following inequalities hold:

$$\begin{aligned}\frac{m-1}{{}_mC_2 +{}_{n-m} C_2} &= \frac{2(m-1)}{m(m-1)+(n-m)(n-m-1)}\\ &< \frac{2(m-1)}{m(m-1)+(n-m)(m-1)}\\ &= \frac{2}{n} < \frac{4}{n}.\end{aligned}$$

Therefore, (2.6) holds in all cases. □

Lemma 16. *$(1-p)^r \ge 1-rp$ holds for any real number $p \le 1$ and any integer $r \ge 1$.*

Proof . We show this lemma by induction with respect to r. If $r=1$, the equality $(1-p)^r = 1-rp$ trivially holds. Assume that $(1-p)^r \ge 1-rp$ holds when $r=k$. Then, we have

$$\begin{aligned}(1-p)^{k+1} = (1-p)(1-p)^k &\ge (1-p)(1-kp)\\ &\ge 1-(k+1)p+kp^2 \ge 1-(k+1)p.\end{aligned}$$

Therefore, $(1-p)^r \ge 1-rp$ holds for any integer $r \ge 1$. □

2.6 Conclusion

In this chapter, the author introduced a novel concept of loose-stabilization and presented a loosely-stabilizing leader election protocol in the PP model of complete networks. The basic strategy of this protocol is described as follows: if two leaders interact each other then one of the two becomes a non-leader, and if two non-leaders with the timer value 0 interact each other then one of the two becomes a leader. The timer value of each agent is controlled so that the timer value of every agent keeps relatively high when at least one leader exists in the population, while the highest timer value among all agents is monotonically non-increasing when no leader exists. Thus, starting from any arbitrary configuration, the protocol reaches a configuration in $\mathcal{S}_{\mathrm{half}}$ within $O(nN \log n)$ expected steps, and then, it keeps the unique leader for $\Omega(Ne^N)$ expected steps, where n is the actual network size and N is a known upper bound of n. The proposed protocol has practical significance from the following reason: the protocol can be practically considered to attain self-stabilization because of exponentially long time of keeping a unique leader while the self-stabilizing leader election in the PP model of complete networks is impossible without the knowledge of the exact network size [11].

The future work is to apply the notion of loose-stabilization to other problems that are known unsolvable or too costly in a self-stabilizing fashion.

Chapter 3

Loosely-stabilizing Leader Election on Arbitrary Graphs in Population Protocols

3.1 Introduction

Self-stabilizing leader election (SS-LE) requires that starting from any configuration, a system reaches a safe-configuration in which a unique leader is elected, and after that, the system has the unique leader forever. Self-stabilizing leader election is important in the population protocol model (the PP model) because (i) many population protocols in the literature work on the assumption that a unique leader exists [5, 6, 7], and (ii) self-stabilization tolerates any finite number of transient faults and this property suits systems consisting of numerous cheap and unreliable nodes. (Such systems are the original motivation of the PP model.) However, there exists strict impossibility of SS-LE in the PP model: no protocol can solve SS-LE for complete graphs, arbitrary graphs, trees, lines, degree-bounded graphs and so on unless the exact size of the graph (the number of agents n) is available [7]. This impossibility holds even if we strengthen the PP model by assigning unique identifies to agents, allowing agents to use random numbers, introducing memory of communication links (mediated population protocols [14]), or allowing more than two agents (k agents) to interact at the same time (the PP_k model [15]).

Accordingly, many studies of SS-LE took either one of the following two approaches. One approach is to accept the assumption that the exact value of n is available and focus on the space complexity of the protocol. Cai et al. [11] proved that n states of each agent is necessary and sufficient to solve SS-LE for a complete graph of n agents. Mizoguchi et al.[16] and Xu et al. [17] improved the space-complexity by adopting the mediated population protocol model and the PP_k model respectively. The other approach is to use *oracles*, a kind of failure detectors. Fischer and Jiang [12] took this approach for the first time. They introduced oracle $\Omega?$ that informs all agents whether at least one leader exists or not and proposed two protocols that solve SS-LE for rings and complete graphs by using $\Omega?$. Beauquier et al. [18] presented an SS-LE protocol for arbitrary graphs that uses two copies of $\Omega?$. Canepa et al. [19] proposed two SS-LE protocols that use $\Omega?$ and consume only 1 bit of each agent: one is a deterministic protocol for trees and the other is a probabilistic protocol for arbitrary graphs although the position of the leader is not static and moves among the agents.

In chapter 2, the author took a new approach to solve SS-LE. The author introduced the concept of loose-stabilization, which relaxes the closure requirement of self-stabilization: we allow protocols to deviate from the specification after following it for a sufficiently long time. Concretely, starting from any initial configuration, the system must reach a loosely-safe configuration within a relatively short time; after that, the specification of the problem (the unique leader) must be kept for a sufficiently long time, though not forever. The author then proposed a loosely-stabilizing protocol that solves leader election on complete graphs using only an upper bound N of n, not using the exact value of n. Starting from any configuration, the protocol reaches a loosely-safe configuration within $O(nN \log n)$ expected steps. After that, the unique leader is kept for $\Omega(Ne^N)$ expected steps. Since the specification is kept for an exponentially long time, we can say this loosely-stabilizing protocol is practically equivalent to a self-stabilizing leader election protocol. Furthermore, this protocol works on any complete graph whose size is no more than N while protocols using the exact value of n work only on the complete graph of size n.

Contribution of This Chapter In this chapter, we consider loosely-stabilizing leader election for *arbitrary undirected graphs.* We consider two settings: (i) the population with agent-identifiers where state-transition is deterministic (as in [20][1]) and (ii) the population consisting of anonymous agents where the agents can use random numbers for state-transition (as in [19]). As mentioned above, no self-stabilizing protocol can solve SS-LE for arbitrary graphs, even in these settings, unless the exact value of n is available. For each setting, we propose two protocols P_{ID} and

[1] Strictly speaking, the model of this chapter with identifiers is stronger than the model in [20]. We use identifiers to compare their values while Guerraoui et al. [20] only allow equality-test of identifiers and prohibited any other calculation of identifiers such as value-comparing.

P_{RD} respectively. The proposed protocols use just upper bounds N of n and Δ of the maximum degree of nodes, and does not use the exact number of n. To elect the unique leader, we take "the minimum ID selection" approach for P_{ID} utilizing the identifiers of agents while we take a novel approach we call "virus war mechanism" for P_{RD} utilizing random numbers.

Given upper bounds N of n and Δ of the maximum degree of nodes, both protocols keep the unique leader for $\Omega(Ne^N)$ expected steps after reaching a loosely-safe configuration. Protocol P_{ID} reaches a loosely-safe configuration within $O(mN\Delta \log n)$ expected steps while P_{RD} does within $O(mN^3\Delta^2 \log N)$ expected steps where m is the number of edges of the graph. Both protocols consume only $O(\log N)$ bits of each agent's memory. We can say this space complexity is small because even space optimal self-stabilizing protocols that use exact value of n consume $O(\log n)$ bits of each agent [11, 16]. For simplicity, the proposed protocols are presented for undirected graphs. However, they work on *directed* graphs with slight modification which is discussed in the conclusion.

Angluin et al. [5] proves that for any population protocol P working on complete graphs, there exists a protocol that simulates P on any arbitrary graph. However, this simulation can be achieved assuming that all the agents have the common initial states at the start of the execution. Since we cannot assume the specific initial states (This is the essence of self-stabilization), we cannot translate the loosely-stabilizing algorithm for complete graphs presented in 2 to a loosely-stabilizing algorithm that works for arbitrary graphs.

3.2 Preliminaries

This section defines the model of this chapter. The model includes both agent-identifiers and random numbers while protocols P_{ID} and P_{RD} use only one of them. In what follows, we denote set $\{z \in \mathbb{N} \mid x \le z \le y\}$ by $[x, y]$.

A *population* is a simple and weakly-connected directed graph $G(V, E, \text{id})$ where V ($|V| \ge 2$) is a set of *agents*, $E \subseteq V \times V$ is a set of directed edges and *id* defines unique identifiers of agents. Each edge represents a possible *interactions* (or communication between two agents): If $(u, v) \in E$, agents u and v can interact with each other where u serves as an *initiator* and v serves as a *responder*. Each agent v has the unique identifier $\text{id}(v) \in I$ ($I = [0, \text{id}_{\max}]$, $\text{id}_{\max} \in O(n^c)$ for constant c). We say that G is undirected if it satisfies $(u, v) \in E \Leftrightarrow (v, u) \in E$. We define $n = |V|$ and $m = |E|$.

A *protocol* $P(Q, Y, I, R, T, O)$ consists of a finite set Q of states, a finite set Y of output symbols, a set of possible identifiers I, a range of random numbers $R \subset \mathbb{N}$, transition function

$T : (Q \times I) \times (Q \times I) \times R \to Q \times Q$, and output function $O : (Q \times I) \to Y$. When an interaction between two agents occurs, T determines the next states of the two agents based on the current states of the agents, identifiers of the two agents, and a random number $r \in R$ generated at each interaction. The *output of an agent* is determined by O: the output of agent v with state $q \in Q$ is $O(q, \mathrm{id}(v))$. We assume that the set of possible identifiers I is a given parameter and not subject to protocol design.

A *configuration* is a mapping $C : V \to Q$ that specifies the states of all the agents. We denote the set of all configurations of protocol P by $\mathcal{C}_{\mathrm{all}}(P)$. We say that configuration C changes to C' by interaction $e = (u, v)$ and integer $r \in R$, denoted by $C \overset{e,r}{\to} C'$, if we have $(C'(u), C'(v)) = T(C(u), \mathrm{id}(u), C(v), \mathrm{id}(v), r)$ and $C'(w) = C(w)$ for all $w \in V \setminus \{u, v\}$. A scheduler determines which interaction occurs at each time. In this chapter, we consider a uniformly random scheduler $\Gamma = \Gamma_0, \Gamma_1, \dots$: each $\Gamma_t \in E$ is a random variable such that $\Pr(\Gamma_t = (u, v)) = 1/m$ for any $t \ge 0$ and any $(u, v) \in E$. We also define the random number sequence as $\Lambda = R_1, R_2, \dots$: each number $R_t \in R$ is a random variable such that $\Pr(R_t = r) = 1/|R|$ for any $t \ge 0$ and $r \in R$. Given an initial configuration C_0, Γ, and Λ, the *execution* of protocol P is defined as $\Xi_P(C_0, \Gamma, \Lambda) = C_0, C_1, \dots$ such that $C_t \overset{\Gamma_t, R_t}{\to} C_{t+1}$ for all $t \ge 0$. We denote $\Xi_P(C_0, \Gamma, \Lambda)$ simply by $\Xi_P(C_0)$ when no misunderstanding can arise.

The leader election problem requires that every agent should output L or F which means "leader" or "follower" respectively. We say that a finite or infinite sequence of configurations $\xi = C_0, C_1, \dots$ preserves a unique leader, denoted by $\xi \in LE$, if there exists $v \in V$ such that $O(C_t(v), \mathrm{id}(v)) = L$ and $O(C_t(u), \mathrm{id}(u)) = F$ for any $t \ge 0$ and $u \in V \setminus \{v\}$. For $\xi = C_0, C_1, \dots,$ the holding time of the leader $HT(\xi, LE)$ is defined as the maximum $t \in \mathbb{N}$ that satisfies $(C_0, C_1, \dots, C_{t-1}) \in LE$. We define $HT(\xi, LE) = 0$ if $C_0 \notin LE$. We denote $\mathbf{E}[HT(\Xi_P(C), LE)]$ by $\mathrm{EHT}_P(C, LE)$. Intuitively, $\mathrm{EHT}_P(C, LE)$ is the expected number of interactions for which the population keeps the unique leader after protocol P starts from configuration C. For configuration sequence $\xi = C_0, C_1, \dots$ and a set of configurations $\mathcal{C}$, we define convergence time $CT(\xi, \mathcal{C})$ as the minimum $t \in \mathbb{N}$ that satisfies $C_t \in \mathcal{C}$. We define $CT(\xi, \mathcal{C}) = |\xi|$ if $C_t \notin \mathcal{C}$ for any $t \ge 0$, where $|\xi|$ is the length of ξ. We denote $\mathbf{E}[CT(\Xi_P(C), \mathcal{C})]$ by $\mathrm{ECT}_P(C, \mathcal{C})$. Intuitively, $\mathrm{ECT}_P(C, \mathcal{C})$ is the expected number of interactions by which the population reaches a configuration in $\mathcal{C}$ after P starts from C.

Definition 3. *Protocol $P(Q, Y, I, R, T, O)$ is an (α, β)-loosely-stabilizing leader election protocol if there exists set $\mathcal{S}$ of configurations satisfying two inequalities $\max_{C \in \mathcal{C}_{\mathrm{all}}(P)} \mathrm{ECT}_P(C, \mathcal{S}) \le \alpha$ and $\min_{C \in \mathcal{S}} \mathrm{EHT}_P(C, LE) \ge \beta$.*

3.2.1 Chernoff Bounds

In this section, we quote the three variants of Chernoff bounds [13] used in several proofs of this chapter.

Lemma 17 (from Eq. (4.2) in [13]). *The following inequality holds for any binomial random variable X:*

$$\Pr(X \ge 2\mathbf{E}[X]) \le e^{-\mathbf{E}[X]/3}.$$

Lemma 18 (from Eq. (4.5) in [13]). *The following inequality holds for any binomial random variable X:*

$$\Pr(X \le \mathbf{E}[X]/2) \le e^{-\mathbf{E}[X]/8}.$$

Lemma 19 (from Eq. (4.5) in [13]). *The following inequality holds for any binomial random variable X:*

$$\Pr(X \le \mathbf{E}[X]/4) \le e^{-9\mathbf{E}[X]/32}.$$

3.3 Leader Election with Identifiers

This section presents loosely-stabilizing leader election protocol P_{ID}, which works on arbitrary undirected graphs with unique identifiers of agents (Code 3.1). In the protocol description, we regard a state of agents as a collection of *variables* (e.g. `timer`), and denote a transition function as pseudo code that updates variables of initiator x and responder y. We denote the value of variable `var` of agent $v \in V$ by v.`var`. We also denote the value of `var` in state $q \in Q$ by q.`var`.

This protocol elects the agent with the minimum identifier, denoted by $v_{\min}$, as the leader. Each agent v tries to find the minimum identifier and stores it on v.`lid`. At interaction, two agents x and y compare their `lid` and store the smaller value on their `lid` (Lines 3 and 6), by which the smallest identifier $\mathrm{id}(v_{\min})$ eventually spreads to all the agents. Then, after some point, $v_{\min}$ is the unique leader because output function O makes only agents v satisfying $\mathrm{id}(v) = v$.`lid` output L and other agents output F.

However, in the initial configuration, some agents may have false identifiers (or the integers that are not identifiers of any agent in the population) on `lid`. A false identifier may spread to the population instead of $\mathrm{id}(v_{\min})$ if it is smaller than $\mathrm{id}(v_{\min})$. We define $\mathrm{ID} = \{\mathrm{id}(v) \mid v \in V\}$, which is the correct identifiers set (Note that $\mathrm{ID} \subseteq I$). Protocol P_{ID} removes false identifiers $i \notin \mathrm{ID}$ from `lid` of all the agents by the *timeout mechanism*. Specifically, if x.`lid` $\ne y$.`lid`, we take the timer value of the agent with smaller `lid`, decrease it by one, and substitute the decreased value into $x.$`lid` and $y.$`lid` (Lines 4 and 7). If $x.\texttt{lid} = y.\texttt{lid}$, we take the larger value of $x.$`timer` and $y.$`timer`, decrease it by one, and substitute the decreased value into $x.$`lid` and $y.$`lid` (Line 9). We call this event *larger value propagation.* If x or y is a leader, both timers are reset to $t_{\max}$ (Line 12). We call this event *timer reset.* When a timer becomes zero, agents x and y suspect that there exists no leader in the population. In this case, they elect the one with a smaller identifier as a leader by substituting $\min(\mathrm{id}(x), \mathrm{id}(y))$ into $x.$`lid` and $y.$`lid` (Line 14). We call this event *timeout.* Agents with false identifiers never experience timer reset; thus, their timers keep on decreasing. Hence, timeout eventually occurs and their `lid`s satisfy $\texttt{lid} \in \mathrm{ID}$. This mechanism rarely ruins the stability of the unique leader because agents with $\texttt{lid} \in \mathrm{ID}$ keep high value timers because of timer reset and lager value propagation.

Complexity Analysis The upper bound $t_{\max}$ of variable `timer` is the only parameter of P_{ID}, which affects the correctness and complexities of the protocol. We assume $t_{\max} \geq 8\delta \max(d,\ 2 + \log n)$ where δ is the maximum degree of the agents and d is the diameter of population G. (Note that δ is an even number because G is undirected.) We prove the following equations under this assumption:

$$\max\nolimits_{C \in \mathcal{C}_{\mathrm{all}}} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{S}_{\mathrm{id}}) = O(m\delta\tau \log n), \tag{3.1}$$

$$\min\nolimits_{C \in \mathcal{S}_{\mathrm{id}}} \mathrm{EHT}_{P_{\mathrm{ID}}}(C, LE) = \Omega(\tau e^{\tau}), \tag{3.2}$$

where $\tau = t_{\max}/(8\delta)$ and $\mathcal{S}_{\mathrm{id}}$ is the set of configurations in which $v.\texttt{lid} = \mathrm{id}(v_{\min})$ and $v.\texttt{timer} > t_{\max}/2$ hold for all $v \in V$ and $v_{\min}.\texttt{timer} = t_{\max}$ holds. When upper bounds N of n and Δ of δ are available and we assign $t_{\max} = 8N\Delta$, protocol P_{ID} is an $(O(m\Delta N \log n), \Omega(Ne^{N}))$-loosely-stabilizing leader election protocol.

First, we analyze the expected holding time. Let $C_0 \in \mathcal{S}_{\mathrm{id}}$ and $\Xi_{P_{\mathrm{ID}}}(C_0) = C_0, C_1, \ldots$. To prove (3.2), it suffices to show that both $C_0, \ldots, C_{2m\tau} \in LE$ and $C_{2m\tau} \in \mathcal{S}_{\mathrm{id}}$ hold with probability at least $p_{\mathrm{suc}} = 1 - O(ne^{-\tau})$. Then, we have $\min_{C_0 \in \mathcal{S}_{\mathrm{id}}} \mathrm{EHT}_{P_{\mathrm{ID}}}(C_0, LE) \geq 2m\tau/(1 - p_{\mathrm{suc}}) = \Omega(\tau e^{\tau})$.

Lemma 20. *The probability that every $v \in V$ joins only less than $t_{\max}/2$ interactions among $\Gamma_0, \ldots, \Gamma_{2m\tau-1}$ is at least $1 - ne^{-\tau}$.*

Proof . For any $v \in V$ and $t \geq 0$, v joins interaction Γ_t with probability at most δ/m. Thus, the number of interactions v joins during the $2m\tau$ interactions is bounded by binomial random

Code 3.1 Leader Election with Identifiers P_{ID}

Variables of each agent:

$\texttt{lid} \in I$, $\texttt{timer} \in [0, t_{\max}]$

Output function O:

if $v.\texttt{lid} = \mathrm{id}(v)$ holds, then the output of agent v is L; Otherwise, F.

Interaction between initiator x and responder y:

```
if $x.\texttt{lid} > \mathrm{id}(x)$ then $x.\texttt{lid} \leftarrow \mathrm{id}(x)$ endif
if $x.\texttt{lid} < y.\texttt{lid}$ then
    $y.\texttt{lid} \leftarrow x.\texttt{lid}$
    $x.\texttt{timer} \leftarrow y.\texttt{timer} \leftarrow \max(x.\texttt{timer} - 1,\ 0)$
else if $x.\texttt{lid} > y.\texttt{lid}$ then
    $x.\texttt{lid} \leftarrow y.\texttt{lid}$
    $x.\texttt{timer} \leftarrow y.\texttt{timer} \leftarrow \max(y.\texttt{timer} - 1,\ 0)$
else                                                  // $x.\texttt{lid} = y.\texttt{lid}$ at this time
    $x.\texttt{timer} \leftarrow y.\texttt{timer} \leftarrow \max(x.\texttt{timer} - 1,\ y.\texttt{timer} - 1,\ 0)$
end if
if $\mathrm{id}(x) = x.\texttt{lid}$ or $\mathrm{id}(y) = y.\texttt{lid}$ then          // a leader resets timers
    $x.\texttt{timer} \leftarrow y.\texttt{timer} \leftarrow t_{\max}$
else if $x.\texttt{timer} = 0$ then                   // a new leader is created at timeout
    $x.\texttt{lid} \leftarrow y.\texttt{lid} \leftarrow \min(\mathrm{id}(x),\ \mathrm{id}(y))$
    $x.\texttt{timer} \leftarrow y.\texttt{timer} \leftarrow t_{\max}$
end if
```

variable $X \sim B(2m\tau, \delta/m)$. Applying a variant of Chernoff bound (Lemma 17), we have

$$\begin{aligned}
\Pr(X \geq t_{\max}/2) &= \Pr(X \geq 2\mathbf{E}[X]) && \because t_{\max} = 8\delta\tau \\
&\leq e^{-\mathbf{E}[X]/3} \\
&= e^{-2\delta\tau/3} && \text{(By Chernoff Bound of Lemma 17)} \\
&\leq e^{-\tau}. && \because \delta \geq 2
\end{aligned}$$

Summing up the probabilities for all $v \in V$ gives the lemma. □

Lemma 21. *Let $C_0 \in \mathcal{L}_{\mathrm{lid}}$ and $\Xi_{P_{\mathrm{ID}}}(C_0) = C_0, C_1, \ldots$. Then, we have the following inequality:*

$$\Pr(\forall v \in V,\ C_{2m\tau}(v).\texttt{timer} > t_{\max}/2) \geq 1 - 2ne^{-\tau}.$$

Proof . It suffices to show $\Pr(C_{2m\tau}(v).\texttt{timer} > t_{\max}/2) \ge 1 - 2e^{-\tau}$ for any agent $v \in V$. We denote the shortest path from $v_{\min}$ to v by $(v_0, v_1, \dots, v_k)$ where $v_0 = v_{\min},\ v_k = v,\ 0 \le k \le d$ and $(v_{i-1}, v_i) \in E$ for all $i = 1, \dots, k$. For any $t \in [0, 2m\tau]$, we define $v_{\text{head}}(t)$ as v_l with maximum $l \in [1, k]$ such that there exist $t_1, t_2, \dots, t_l$ satisfying $0 \le t_1 < t_2 < \cdots < t_l < t$ and $\Gamma_{t_i} \in \{(v_{i-1}, v_i), (v_i, v_{i-1})\}$ for $i = 1, 2, \dots, l$. We define $v_{\text{head}}(t) = v_0$ if such l does not exist. Intuitively, $v_{\text{head}}(t)$ is the head of the agents in path $(v_0, v_1, \dots, v_k)$ to which a large timer value is propagated from $v_{\min}$. (Remember that $v_{\min}$ resets the timers to $t_{\max}$.) We define $J(t)$ as the number of integers $i \in [0, t]$ such that $v_{\text{head}}(i)$ joins interaction Γ_i. Intuitively, $J(t)$ is the number of interactions that the head agent joins among $\Gamma_0, \dots, \Gamma_t$. Obviously, we have $C_t(v_{\text{head}}(t)).\texttt{timer} \ge t_{\max} - J(t)$ for any $t \ge 0$.

In what follows, we prove $\Pr(v_{\text{head}}(2m\tau) = v) \ge 1 - e^{-\tau}$ and $\Pr(J(2m\tau) < t_{\max}/2) \ge 1 - e^{-\tau}$, which give $\Pr(C_{2m\tau}(v).\texttt{timer} > t_{\max}/2) \ge 1 - 2e^{-\tau}$. For any $i \in [1, k]$, a pair v_{i-1} and v_i interacts with probability $2/m$ at each interaction. Hence, we can say each interaction makes v_{head} forward with probability $2/m$. Therefore, by letting Z be a binomial random variable such that $Z \sim B(2m\tau, 2/m)$, we have

$$\begin{aligned}
\Pr(v_{\text{head}}(t) = v) &= 1 - \Pr(Z < k) \\
&\ge 1 - \Pr(Z < d) \\
&\ge 1 - \Pr\left(Z < \frac{1}{4} \cdot \mathbf{E}[Z]\right) && \because d \le \tau = \frac{1}{4} \cdot \mathbf{E}[Z] \\
&\ge 1 - e^{-9\mathbf{E}[Z]/32} && \text{(By Chernoff bound of Lemma 19)} \\
&> 1 - e^{-\tau}.
\end{aligned}$$

The probability that $v_{\text{head}}(t)$ joins interaction Γ_t is at most δ/m regardless of t. Hence, by letting Z' be a binomial random variable such that $Z' \sim B(2m\tau, \delta/m)$, we have

$$\begin{aligned}
\Pr(J(2m\tau) < t_{\max}/2) &> 1 - \Pr(Z' \ge t_{\max}/2) \\
&= 1 - \Pr(Z' \ge 2\mathbf{E}[Z']) \\
&> 1 - e^{-\mathbf{E}[Z']/3} && \text{(By Chernoff bound of Lemma 17)} \\
&= 1 - e^{-2\delta\tau/3} \\
&> 1 - e^{-\tau}. && \because \delta \ge 2
\end{aligned}$$

Thus, we have shown $\Pr(C_{2m\tau}(v).\texttt{timer} > t_{\max}/2) \ge 1 - 2e^{-\tau}$. □

Lemma 22. $\min_{C \in \mathcal{S}_{\text{id}}} \text{EHT}_{P_{\text{ID}}}(C, LE) = \Omega(\tau e^{\tau})$.

Proof . We have $C_0, \dots, C_{2m\tau} \in \mathit{LE}$ and $C_{2m\tau} \in \mathcal{S}_{\mathrm{id}}$ if $C_0 \in \mathcal{S}_{\mathrm{id}}$ holds, no timeout happens, and any agent interacts at most $t_{\max}/2$ times during $2m\tau$ interactions. Hence, probability p_{suc} discussed in the beginning of this section is at least $1 - 3ne^{-\tau}$ by Lemmas 20 and 21, which leads to the lemma. $\square$

Next, we analyze the expected convergence time. To prove (3.1), we define two sets of configurations: $\mathcal{C}_{\mathrm{lid}} = \{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{ID}}) \mid \forall v \in V, C(v).\mathtt{lid} \in \mathrm{ID}\}$ and $\mathcal{L}_{\mathrm{lid}} = \mathcal{C}_{\mathrm{lid}} \cap \{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{ID}}) \mid C(v_{\min}).\mathtt{lid} = \mathrm{id}(v_{\min}) \land C(v_{\min}).\mathtt{timer} = t_{\max}\}$.

Lemma 23. $\max_{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{ID}})} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{C}_{\mathrm{lid}}) = O(m\delta\tau \log n)$.

Proof . Let z be the maximum value of $v.\mathtt{timer}$ such that $v.\mathtt{lid} \notin \mathrm{ID}$. This z decreases by one every time all interactions of E occur. Thus, it takes at most $\frac{m}{m} + \frac{m}{m-1} + \dots \frac{m}{1} \le m(1 + \log m)$ expected steps to decrease z by one. Hence, $\max_{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{ID}})} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{C}_{\mathrm{lid}}) \le t_{\max} m(1+\log m) = O(m\delta\tau \log n)$. $\square$

Lemma 24. $\max_{C \in \mathcal{C}_{\mathrm{lid}}} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{L}_{\mathrm{lid}}) = O(m)$.

Proof . We have $v_{\min}.\mathtt{lid} = \mathrm{id}(v_{\min})$ and $v_{\min}.\mathtt{timer} = t_{\max}$ just after $v_{\min}$ interacts in any configuration of $\mathcal{C}_{\mathrm{lid}}$. This takes $O(m)$ expected interactions. $\square$

Lemma 25. $\max_{C \in \mathcal{L}_{\mathrm{lid}}} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{S}_{\mathrm{id}}) = O(m\tau)$.

Proof . Let $C_0 \in \mathcal{L}_{\mathrm{lid}}$ and $\Xi_{P_{\mathrm{ID}}}(C_0) = C_0, C_1, \dots$. Since $C_t \in \mathcal{L}_{\mathrm{lid}}$ holds for every $t \ge 0$, identifier $\mathrm{id}(v_{\min})$ is the smallest among $\mathtt{lid}$ of all the agents in any configuration $C_0, C_1, \dots$. Hence, once agent v satisfies $v.\mathtt{lid} = \mathrm{id}(v_{\min})$, then $v.\mathtt{lid} = \mathrm{id}(v_{\min})$ always holds until a timeout occurs at v. Lemma 21 has shown that, with probability at least $1 - 2ne^{-\tau}$, every agent v satisfies $v.\mathtt{lid} = \mathrm{id}(v_{\min})$ within $2m\tau$ interactions, and after that, keeps on satisfying $v.\mathtt{timer} > t_{\max}/2$ at least until $\Gamma_{2m\tau-1}$ finishes. Thus, the probability that $C_{2m\tau}(v).\mathtt{lid} = \mathrm{id}(v_{\min})$ and $C_{2m\tau}(v).\mathtt{timer} > t_{\max}/2$ hold for all $v \in V$ is at least $1 - 2ne^{-\tau}$. Note that $C_{2m\tau}(v_{\min}).\mathtt{timer} = t_{\max}$ holds with probability 1. Hence, we have

$$\max_{C \in \mathcal{L}_{\mathrm{lid}}} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{S}_{\mathrm{id}}) \le 2m\tau + 2ne^{-\tau} \cdot \max_{C \in \mathcal{L}_{\mathrm{lid}}} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{S}_{\mathrm{id}}).$$

Solving this inequality gives $\max_{C \in \mathcal{L}_{\mathrm{lid}}} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{S}_{\mathrm{id}}) \in O(m\tau)$. $\square$

The following lemma immediately follows from Lemmas 23, 24, and 25.

Lemma 26. $\max_{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{ID}})} \mathrm{ECT}_{P_{\mathrm{ID}}}(C, \mathcal{S}_{\mathrm{id}}) = O(m\delta\tau \log n)$.

Lemmas 22 and 26 gives the following theorem.

Theorem 3. *Protocol P_{ID} is a $(O(m\delta\tau \log n), \Omega(\tau e^{\tau}))$ loosely-stabilizing leader election protocol for arbitrary graphs when $t_{\max} \geq 8\delta \max(d, 2 + \log n)$.*

Therefore, given upper bound N and Δ of n and δ respectively, we get a $(O(m\Delta N \log n), \Omega(Ne^{N}))$ loosely-stabilizing leader election protocol for arbitrary graphs by assigning $t_{\max} = 8N\Delta$.

3.4 Leader Election with Random Numbers

This section presents loosely-stabilizing leader election protocol P_{RD}. It works on arbitrary undirected anonymous graphs with a random number generated at each interaction (Code 3.2). Random numbers are used in Line 11: When the protocol enters Line 11, the code is executed with probability $p = 1/|R|$. This is implemented as the code is executed only when a specific number is generated. For example, $p = 0.01$ if we assign $R = [0, 99]$ and treat 0 as a specific number.

Each agent has binary variable $\mathtt{DoA} \in \{\mathrm{DEAD}, \mathrm{ALIVE}\}$ and three timers $\mathtt{timer}_{\mathtt{L}}$, $\mathtt{timer}_{\mathtt{V}}$ and $\mathtt{timer}_{\mathtt{S}}$. The output function defines leaders based on $\mathtt{DoA}$: agent v is a leader if v is alive (or $v.\mathtt{DoA} = \mathrm{ALIVE}$), and a follower if v is dead (or $v.\mathtt{DoA} = \mathrm{DEAD}$). Protocol P_{RD} consists of a timeout mechanism (Lines 1-7) and a virus-war mechanism (Lines 8-14). By using $\mathtt{timer}_{\mathtt{L}}$, the timeout mechanism creates a leader when it is suspected that no leader exists. By using $\mathtt{timer}_{\mathtt{V}}$ and $\mathtt{timer}_{\mathtt{S}}$, the virus-war mechanism reduces the number of leaders.

The timeout mechanism is almost the same as P_{ID}. By the timer reset and the larger value propagation, timeout eventually occurs when no leader exists, and all agents keep high timer values with high probability when one ore more leaders exist. At timeout, a dead agent becomes a leader (Line 5).

In the virus-war mechanism, each leader tries to kill other leaders by viruses and become the unique leader. We say that agent v has a virus if $v.\mathtt{timer}_{\mathtt{V}} > 0$, and v wears a (head) shield if $v.\mathtt{timer}_{\mathtt{S}} > 0$. A leader creates a new virus with probability p when it interacts as an initiator (Line 11). When creating a virus, the agent wears a shield so as not to be killed by the new virus (Line 11). A virus spreads among agents by interactions (Line 8), and an agent is killed when it has a virus without a shield (Lines 13-14). A virus has TTL (time to live), which is memorized on $\mathtt{timer}_{\mathtt{V}}$ and decreased by one at each interaction of its owner (line 8). When $\mathtt{timer}_{\mathtt{V}}$ becomes zero, the virus vanishes and looses the ability to kill agents. A shield also has TTL, which is memorized on $\mathtt{timer}_{\mathtt{S}}$ and decreased by one at each interaction of its owner (Line 9). When $\mathtt{timer}_{\mathtt{S}}$ becomes zero, the shield vanishes and looses the ability to protect its owner from viruses.

The virus-war mechanism correctly works if p is sufficiently small and t_{shld} is sufficiently greater than t_{virus}. Consider the case multiple leaders exist. Since p is small, all viruses and

Code 3.2 Leader Election with Random Numbers P_{RD}

Variables of each agent:

$\mathtt{DoA} \in \{\mathrm{DEAD},\ \mathrm{ALIVE}\}$, $\mathtt{timer_L} \in [0, t_{\max}]$, $\mathtt{timer_V} \in [0, t_{\mathrm{virus}}]$, $\mathtt{timer_S} \in [0, t_{\mathrm{shld}}]$

Output function O:

if $v.\mathtt{DoA} = \mathrm{ALIVE}$ holds, then the output of agent v is L, otherwise F.

Interaction between initiator x and responder y:

```
1: $x.\mathtt{timer_L} \leftarrow y.\mathtt{timer_L} \leftarrow \max(x.\mathtt{timer_L} - 1,\ y.\mathtt{timer_L} - 1,\ 0)$
2: if $x.\mathtt{DoA} = \mathrm{ALIVE}$ or $y.\mathtt{DoA} = \mathrm{ALIVE}$ then
3:     $x.\mathtt{timer_L} \leftarrow y.\mathtt{timer_L} \leftarrow t_{\max}$          // a leader resets timer
4: else if $x.\mathtt{timer_L} = 0$ then          // a new leader is created at timeout
5:     $x.\mathtt{DoA} \leftarrow \mathrm{ALIVE}$
6:     $x.\mathtt{timer_L} \leftarrow y.\mathtt{timer_L} \leftarrow t_{\max}$
7: end if
8: $x.\mathtt{timer_V} \leftarrow y.\mathtt{timer_V} \leftarrow \max(x.\mathtt{timer_V} - 1,\ y.\mathtt{timer_V} - 1,\ 0)$
9: $x.\mathtt{timer_S} \leftarrow \max(0, x.\mathtt{timer_S} - 1)$
10: if $x.\mathtt{DoA} = \mathrm{ALIVE}$ then
11:     Execute $(x.\mathtt{timer_V} \leftarrow t_{\mathrm{virus}},\ x.\mathtt{timer_S} \leftarrow t_{\mathrm{shld}})$ with probability $p$
                // An alive initiator creates a new virus and a new shield with probability $p$.
12: end if
13: if $x.\mathtt{timer_V} > 0$ and $x.\mathtt{timer_S} = 0$ then $x.\mathtt{DoA} \leftarrow \mathrm{DEAD}$ endif
14: if $y.\mathtt{timer_V} > 0$ and $y.\mathtt{timer_S} = 0$ then $y.\mathtt{DoA} \leftarrow \mathrm{DEAD}$ endif
```

shields eventually vanish. After that, some agent eventually creates a new virus and shield. The created virus kills all other agents unless some of them also create a new virus and shield before the virus reaches them. Since p is sufficiently small, the probability of the latter is small. Thus, the unique leader is elected within a relatively short time. Even after that, the unique leader keeps on creating new viruses, each of which may kill the leader. However, the leader is not killed for an extremely long time: since $t_{\mathrm{shld}} \gg t_{\mathrm{virus}}$, the leader's shield rarely vanishes before all viruses vanish from the population.

Complexity Analysis We have four parameters in P_{RD}: three upper bounds $t_{\max}$, t_{virus}, and t_{shld} of the timers, and probability p. We assume $t_{\mathrm{virus}} = t_{\max}/2$, $t_{\max} \geq 8\delta \max(d,\ 2 + \log(13n\delta\lceil \log n \rceil))$, $t_{\mathrm{shld}} \geq 2\delta t_{\max}\lceil \log n \rceil$ and $p \leq (4mt_{\mathrm{shld}})^{-1}$. We prove the following equations under this assumption:

$$\max_{C \in \mathcal{C}_{\mathrm{all}}} \mathrm{ECT}_{P_{\mathrm{RD}}}(C, \mathcal{S}_{\mathrm{RD}}) = O(mp^{-1}), \tag{3.3}$$

$$\min_{C \in \mathcal{S}_{\mathrm{RD}}} \mathrm{EHT}_{P_{\mathrm{RD}}}(C, LE) = \Omega(\tau e^{\tau}), \tag{3.4}$$

where $\tau = t_{\max}/(8\delta)$ and $\mathcal{S}_{\mathrm{RD}}$ is the set of configurations we define later. When upper bounds N and Δ are available and we assign $t_{\max} = 8N\Delta$, $t_{\mathrm{shld}} = 2\Delta t_{\max}\lceil \log N \rceil$ and $p = (4N^2 t_{\mathrm{shld}})^{-1}$ (*i.e.*, $R = [0, 4N^2 t_{\mathrm{shld}} - 1]$), then P_{RD} is an $(O(m\Delta^2 N^3 \log N), \Omega(Ne^N))$-loosely-stabilizing leader election protocol.

Before proving equations (3.3) and (3.4), we define five sets of configurations:

$$\begin{aligned}
\mathcal{G}_{\mathrm{half}} &= \{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}}) \mid \exists v \in V,\ C(v).\mathtt{DoA} = \mathrm{ALIVE} \wedge C(v).\mathtt{timer_S} > t_{\mathrm{shld}}/2\},\\
\mathcal{V}_{\mathrm{clean}} &= \{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}}) \mid \forall v \in V,\ C(v).\mathtt{timer_V} = 0\},\\
\mathcal{L}_{\mathrm{half}} &= \{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}}) \mid \#_L(C) \ge 1\ \wedge\ \forall v \in V,\ C(v).\mathtt{timer_L} > t_{\max}/2\},\\
\mathcal{L}_{\mathrm{one}} &= \{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}}) \mid \#_L(C) = 1\},\\
\mathcal{S}_{\mathrm{RD}} &= (\mathcal{G}_{\mathrm{half}} \cup \mathcal{V}_{\mathrm{clean}}) \cap \mathcal{L}_{\mathrm{half}} \cap \mathcal{L}_{\mathrm{one}},
\end{aligned}$$

where $\#_L(C)$ denotes the number of leaders in configuration C. Note that $\mathcal{G}_{\mathrm{half}}$ requires that not all agents but at least one leader has $\mathtt{timer_S}$ more than $t_{\mathrm{shld}}/2$.

First, we analyze the expected holding time. Let $C_0 \in \mathcal{S}_{\mathrm{RD}}$ and $\Xi_{P_{\mathrm{RD}}}(C_0) = C_0, C_1, \dots$. To prove (3.4), it suffices to show that both $C_0, \dots, C_{8m\delta\tau\lceil \log n \rceil} \in LE$ and $C_{8m\delta\tau\lceil \log n \rceil} \in \mathcal{S}_{\mathrm{RD}}$ hold with probability no less than $p_{\mathrm{suc}} = 1 - O(n\delta \log n \cdot e^{-\tau})$. Then, $\min_{C_0 \in \mathcal{S}_{\mathrm{RD}}} \mathrm{EHT}_{P_{\mathrm{RD}}}(C_0, LE) \ge 8m\delta\tau\lceil \log n \rceil \tau/(1 - p_{\mathrm{suc}}) = \Omega(\tau e^{\tau})$.

We define two predicates PROP_i and HALF_i for any $i \ge 0$: $\mathrm{PROP}_i = 1$ if $C_{2m\tau(i+1)}(v).\mathtt{timer_L} > t_i - t_{\max}/2$ for all $v \in V$, otherwise $\mathrm{PROP}_i = 0$, where $t_i = \max_{v \in V} C_{2m\tau i}(v)$; $\mathrm{HALF}_i = 1$ if every agent joins only less than $t_{\max}/2$ interactions among $\Gamma_{2m\tau i}, \dots, \Gamma_{2m\tau(i+1)-1}$, otherwise $\mathrm{HALF}_i = 0$. Intuitively, $\mathrm{PROP}_i = 1$ means the maximum value of $\mathtt{timer_L}$ propagates to all the agents well during the $2m\tau$ interactions, and $\mathrm{HALF}_i = 1$ means every agent does not interact so much during the $2m\tau$ interactions.

Lemma 27. *Let $C_0 \in \mathcal{S}_{\mathrm{RD}}$ and $\Xi_{P_{\mathrm{RD}}}(C_0) = C_0, C_1, \dots$. Then, we have both $C_0, \dots, C_{8m\delta\tau\lceil \log n \rceil} \in LE$ and $C_{8m\delta\tau\lceil \log n \rceil} \in \mathcal{S}_{\mathrm{RD}}$ if the following conditions hold:*
(A) $\#_L(C_t) \ge 1$ for all $t = 0, \dots, 8m\delta\tau\lceil \log n \rceil$,
(B) $C_{8m\delta\tau\lceil \log n \rceil} \in \mathcal{G}_{\mathrm{half}} \cup \mathcal{V}_{\mathrm{clean}}$,
(C) $\mathrm{PROP}_i = 1$ for all $i = 0, \dots, 4\delta\lceil \log n \rceil - 1$, and
(D) $\mathrm{HALF}_i = 1$ for all $i = 0, \dots, 4\delta\lceil \log n \rceil - 1$.

Proof . We have $C_{2m\tau i}(v).\texttt{timer}_{\mathrm{L}} > t_{\max}/2$ for any $i \in [0, 4\delta\lceil\log n\rceil]$ from (A) and (C). Since no agent interacts more than $t_{\max}/2$ times among each $2m\tau$ interactions (i.e. (D)), timeout does not occur at any interaction $\Gamma_0, \dots, \Gamma_{8m\delta\tau\lceil\log n\rceil-1}$, by which we obtain $C_0, \dots, C_{8m\delta\tau\lceil\log n\rceil} \in LE$. We also obtain $C_{8m\delta\tau\lceil\log n\rceil} \in \mathcal{L}_{\mathrm{half}} \cap \mathcal{L}_{\mathrm{one}} \cap (\mathcal{G}_{\mathrm{half}} \cup \mathcal{V}_{\mathrm{clean}}) = \mathcal{S}_{\mathrm{RD}}$ from above discussion and (B). $\square$

Lemma 28. *The probability that every agent joins only less than $t_{\mathrm{shld}}/2$ interactions as an initiator among $\Gamma_0, \dots, \Gamma_{8m\delta\tau\lceil\log n\rceil-1}$ is at least $1 - ne^{-\delta\tau}$.*

Proof . For any $v \in V$ and $t \geq 0$, v joins interaction Γ_t as an initiator with probability at most $\delta/(2m)$ since v has at most $\delta/2$ outgoing edges. Thus, the number of interactions v joins as an initiator during the $8m\delta\tau\lceil\log n\rceil$ interactions is bounded by binomial random variable $X \sim B(8m\delta\tau\lceil\log n\rceil, \delta/(2m))$. We have

$$\begin{aligned}
\Pr(X \geq t_{\mathrm{shld}}/2) &\leq \Pr(X \geq 8\delta^2\tau\lceil\log n\rceil) \qquad \because t_{\mathrm{shld}} \geq 16\delta^2\tau\lceil\log n\rceil \\
&= \Pr(X \geq 2\mathbf{E}[X]) \\
&\leq e^{-\mathbf{E}[X]/3} \qquad \text{(By Chernoff Bound of Lemma 18)} \\
&= e^{-4\delta^2\tau\lceil\log n\rceil/3} \\
&= e^{-\delta\tau}.
\end{aligned}$$

Summing up these probabilities gives the lemma. $\square$

Lemma 29. *Let $C_0 \in \mathcal{S}_{\mathrm{RD}}$ and $\Xi_{P_{\mathrm{RD}}}(C_0) = C_0, C_1, \dots$.*
Then, we have $\Pr(\forall t \in [0, 8m\delta\tau\lceil\log n\rceil - 1],\ \#_L(C_t) \geq 1) \geq 1 - ne^{-\delta\tau}$.

Proof . By Lemma 28, it suffices to show that $\#_L(C_t) \geq 1$ holds for all $t \in [0, 8m\delta\tau\lceil\log n\rceil]$ when we assume every agent joins only less than $t_{\mathrm{shld}}/2$ interactions as an initiator among $\Gamma_0, \dots, \Gamma_{8m\delta\tau\lceil\log n\rceil-1}$. Since $C_0 \in \mathcal{S}_{\mathrm{RD}}$, we have $C_0 \in \mathcal{G}_{\mathrm{half}} \cup \mathcal{V}_{\mathrm{clean}}$. If $C_0 \in \mathcal{G}_{\mathrm{half}}$, there exists a leader v such that $C_0(v).\texttt{timer}_{\mathrm{S}} > t_{\mathrm{shld}}/2$. By the assumption, v decrease its $\texttt{timer}_{\mathrm{S}}$ by at most $t_{\mathrm{shld}}/2$; thus, v is never killed and remains a leader in $C_0, \dots, C_{8m\delta\tau\lceil\log n\rceil}$. If $C_0 \in \mathcal{V}_{\mathrm{clean}}$, no leader is killed before a new virus is created. Even if some leader u creates a new virus at interaction Γ_t ($0 \leq t < 8m\delta\tau\lceil\log n\rceil$), u wears a new shield at the same time. Hence, u remains a leader in $C_t, \dots, C_{8m\delta\tau\lceil\log n\rceil}$ by the assumption. $\square$

We define the first round time $\mathrm{RT}_\Gamma(1)$ as the minimum t satisfying $\forall e \in E,\ 0 \leq \exists t' \leq t,\ \Gamma_{t'} = e$. For any $i \geq 2$, we define the i-th round time $\mathrm{RT}_\Gamma(i)$ as the minimum t satisfying $\forall e \in E,\ \mathrm{RT}_\Gamma(i-1) < \exists t' \leq t,\ \Gamma_{t'} = e$. Lemma 31 bounds $\mathrm{RT}_\Gamma(i)$ from above with high probability. To prove the lemma, we firstly prove Lemma 30.

Lemma 30. *Let $v_1, v_2, \ldots, v_l$ be any l ($l < n$) agents in V. There exists at least $2l$ edges of E that are incident to at least one of the l agents.*

Proof . Since $l < n$, there exists agent $r \in V$ that differs from any $v_1, v_2, \ldots, v_l$. Since G is strongly connected, there exists a rooted spanning tree T on G where r is the root agent of T. Then, every v_i ($i \in [1,k]$) has two edges between v_i and the parent agent of v_i in T. (Remind that G is undirected, that is, $(u,v) \in E \Leftrightarrow (v,u) \in E$ for any $u, v \in V$.) These edges are mutually exclusive. Thus, we have $2l$ edges of E that are incident to at least one of the l agents. □

Lemma 31. $\Pr(\mathrm{RT}_\Gamma(i) < 2im\lceil \log n \rceil) \geq 1 - ne^{-i/4}$ *holds for any $i \geq 1$.*

Proof . Each round j ($j \geq 1$) finishes when every agent $v \in V$ interacts in round j. Consider the case that k ($k \geq 1$) agents have not yet interacted in round j and only $n-k$ agents have interacted in round j. We call the former uninvolved agents and the latter involved agents. If $k < n$, one of the k uninvolved agents joins the next interaction and becomes an involved agent with probability more than $2k/m$ by Lemma 30. If $k = n$, some uninvolved agent joins the next interaction with probability 1. Let $X_{j,k}$ ($j \geq 1,\ k \geq 1$) be the random variable that corresponds to the number of trials to the first success in which the success probability of each trial is $2k/m$. From the above discussion, we obtain

$$\begin{aligned}\Pr(\mathrm{RT}_\Gamma(i) \geq 2im\lceil \log n \rceil) &\leq \Pr\left(\sum_{j=1}^{i}\left(1 + \sum_{k=1}^{n-1} X_{j,k}\right) \geq 2im\lceil \log n \rceil\right) \\ &\leq \Pr\left(\sum_{k=1}^{n-1}\sum_{j=1}^{i} X_{j,k} \geq 2im\lceil \log n \rceil - i\right).\end{aligned} \tag{3.5}$$

For binomial random variable $Y_k \sim B(\lceil \frac{im}{k} \rceil, \frac{2k}{m})$, we have $\Pr(\sum_{j=1}^{i} X_{j,k} > \frac{im}{k}) \leq \Pr(\sum_{j=1}^{i} X_{j,k} \geq \lceil \frac{im}{k} \rceil) \leq \Pr(Y_k \leq i)$. Hence, we have

$$\begin{aligned}\Pr\left(\sum_{j=1}^{i} X_{j,k} > \frac{im}{k}\right) &\leq \Pr(Y_k \leq i) \\ &\leq \Pr\left(Y_k \leq \frac{1}{2} \cdot \mathbf{E}[Y_K]\right) \\ &\leq e^{-\mathbf{E}[Y_k]/8} \qquad \text{(By Chernoff Bound of Lemma 18)} \\ &\leq e^{-i/4}.\end{aligned} \tag{3.6}$$

From Inequalities (3.5) and (3.6), we have

$$\begin{aligned}\Pr(\mathrm{RT}_\Gamma(i) \geq 2im\lceil\log n\rceil) &\leq \Pr\left(\sum_{k=1}^{n-1}\sum_{j=1}^{i} X_{j,k} \geq 2im\lceil\log n\rceil - i\right)\\ &\leq \Pr\left(\sum_{k=1}^{n-1}\sum_{j=1}^{i} X_{j,k} > \sum_{k=1}^{n-1}\frac{im}{k}\right)\\ &\leq \sum_{k=1}^{n-1}\Pr\left(\sum_{j=1}^{i} X_{j,k} > \frac{im}{k}\right)\\ &\leq ne^{-i/4},\end{aligned}$$

where $\sum_{k=1}^{n-1}\frac{im}{k} \leq im(1+\log n) - i < 2im\lceil\log n\rceil - i$ is used for the second inequality. Thus, $\Pr(\mathrm{RT}_\Gamma(i) < 2im\lceil\log n\rceil) \geq 1 - ne^{-i/4}$ holds. □

Lemma 32. *Let* $C_0 \in \mathcal{S}_{\mathrm{RD}}$ *and* $\Xi_{P_{\mathrm{RD}}}(C_0) = C_0, C_1, \ldots$.
Then, we have $\Pr(C_{8m\delta\tau\lceil\log n\rceil} \in \mathcal{G}_{\mathrm{half}} \cup \mathcal{V}_{\mathrm{clean}}) \geq 1 - 2ne^{-\delta\tau}$.

Proof . Assume that $\mathrm{RT}_\Gamma(t_{\mathrm{virus}}) < 8m\delta\tau\lceil\log n\rceil$ holds and every agent joins only less than $t_{\mathrm{shld}}/2$ interactions as an initiator among $\Gamma_0, \ldots, \Gamma_{8m\delta\tau\lceil\log n\rceil-1}$. These assumptions lead to $C_{8m\delta\tau\lceil\log n\rceil} \in \mathcal{G}_{\mathrm{half}} \cup \mathcal{V}_{\mathrm{clean}}$ as follows. If a new virus is not created among $\Gamma_0, \ldots, \Gamma_{8m\delta\tau\lceil\log n\rceil-1}$, then all viruses in the initial configuration vanish during the period since each round decreases the maximum value of $\mathtt{timer}_\mathtt{V}$ by at least one. Thus, $C_{8m\delta\tau\lceil\log n\rceil} \in \mathcal{V}_{\mathrm{clean}}$ holds. If some agent v creates a new virus at Γ_t, then v wears a new shield at the same time. Thus, $C_{t+1}(v).\mathtt{timer}_\mathtt{S} = t_{\mathrm{shld}}$. Since v interacts as an initiator only less than $t_{\mathrm{shld}}/2$ times among $\Gamma_{t+1}, \ldots, \Gamma_{8m\delta\tau\lceil\log n\rceil-1}$, we have $C_{8m\delta\tau\lceil\log n\rceil}(v).\mathtt{timer}_\mathtt{S} > t_{\mathrm{shld}}/2$, which means $C_{8m\delta\tau\lceil\log n\rceil} \in \mathcal{G}_{\mathrm{half}}$. By $t_{\mathrm{virus}} = 4\delta\tau$ and Lemmas 28 and 31, the probability that the two assumptions hold is at least $1 - 2ne^{-\delta\tau}$. □

Lemma 33. $\Pr(\mathrm{PROP}_i = 1) \geq 1 - 2ne^{-\tau}$ *for any* $i \geq 0$.

Proof . The same argument as the proof of Lemma 21 gives the lemma. □

Lemma 34. $\Pr(\mathrm{HALF}_i = 1) \geq 1 - ne^{-\tau}$ *for any* $i \geq 0$.

Proof . Each interaction is independent. Thus, Lemma 20 gives the lemma. □

Lemma 35. $\min_{C \in \mathcal{S}_{\mathrm{RD}}} \mathrm{EHT}_{P_{\mathrm{RD}}}(C, LE) = \Omega(\tau e^{\tau})$.

Proof . Probability p_{suc}, discussed in the beginning of this section, is at least $1 - 3ne^{-\delta\tau} - 4\delta\lceil\log n\rceil \cdot 3ne^{-\tau} \geq 1 - 13n\delta\lceil\log n\rceil e^{-\tau}$ by Lemmas 27, 29, 32, 33 and 34, which leads to the lemma. □

Next, we analyze the expected convergence time. We define two sets of configurations: $\mathcal{N}_{\mathrm{oVG}} = \{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}}) \mid \forall v \in V,\ C(v).\mathtt{timer_V} = C(v).\mathtt{timer_S} = 0\}$ and $\mathcal{L} = \{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}}) \mid \#_L(C) \geq 1\}$. The goal of the rest of this section is to prove $\max_{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}})} \mathrm{ECT}_{P_{\mathrm{RD}}}(C, \mathcal{S}_{\mathrm{RD}}) = O(mp^{-1})$ (Lemma 41). To prove Lemma 41, we show Lemmas 37, 38 and 40. Lemma 37 (38, 40) gives the lower bound of the probability that the population enters from $\mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}})$ into $\mathcal{N}_{\mathrm{oVG}}$ (from $\mathcal{N}_{\mathrm{oVG}}$ into $\mathcal{N}_{\mathrm{oVG}} \cap \mathcal{L}$, from $\mathcal{N}_{\mathrm{oVG}} \cap \mathcal{L}$ into $\mathcal{S}_{\mathrm{RD}}$, respectively) within a certain number of interactions. We also show Lemmas 36 and 39 to prove Lemmas 37 and 40 respectively.

Lemma 36. *The probability that every $v \in V$ joins more than t_{shld} interactions as an initiator among $\Gamma_0, \ldots, \Gamma_{2mt_{\mathrm{shld}}}$ is at least $1 - ne^{-t_{\mathrm{shld}}/4}$.*

Proof . For any $v \in V$ and $t \geq 0$, v joins interaction Γ_t as an initiator with probability at least $1/m$. Thus, the number of interactions v joins during the $2mt_{\mathrm{shld}}$ interactions is bounded from below by binomial random variable $X \sim B(2mt_{\mathrm{shld}}, 1/m)$. Applying the Chernoff bound of Lemma 18, we have

$$\begin{aligned}\Pr(X \leq t_{\mathrm{shld}}) &= \Pr(X \leq \mathbf{E}[X]/2) \\ &\leq e^{-\mathbf{E}[X]/8} \qquad \text{(By Chernoff Bound of Lemma 18)} \\ &= e^{-t_{\mathrm{shld}}/4}.\end{aligned}$$

Summing up the probabilities for all $v \in V$ gives the lemma. □

Lemma 37. *Let $C_0 \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}})$ and $\Xi_{P_{\mathrm{RD}}}(C_0) = C_0, C_1, \ldots$. Then, we have* $\Pr(C_{2mt_{\mathrm{shld}}} \in \mathcal{N}_{\mathrm{oVG}}) \geq 1 - 2ne^{-\delta\tau} - 2mt_{\mathrm{shld}} \cdot p$.

Proof . First, we show that $C_{2mt_{\mathrm{shld}}} \in \mathcal{N}_{\mathrm{oVG}}$ holds when the following three conditions hold:

(A) every agent $v \in V$ joins more than t_{shld} interactions as an initiator among $\Gamma_0, \ldots, \Gamma_{2mt_{\mathrm{shld}}}$,

(B) $\mathrm{RT}(t_{\mathrm{virus}}) \leq 2mt_{\mathrm{shld}}$, and

(C) no new virus is created during $\Gamma_0, \ldots, \Gamma_{2mt_{\mathrm{shld}}}$.

Until a new virus is created, variable $v.\mathtt{timer_S}$ for each $v \in V$ is monotonically non-increasing and it decreases by one every time v interacts as an initiator. Hence, no agent wears a shield in configuration $C_{2mt_{\mathrm{shld}}}$ by (A) and (C). Until a new virus is created, the maximum value of all $v.\mathtt{timer_V}$ (i.e. $\max_{v \in V} v.\mathtt{timer_V}$) is monotonically non-increasing during $\Gamma_0, \ldots, \Gamma_{2mt_{\mathrm{shld}}}$ and it decreases at least by one in each round. Hence, no agent has a virus in configuration $C_{2mt_{\mathrm{shld}}}$ by (B) and (C). Thus, we have $C_{2mt_{\mathrm{shld}}} \in \mathcal{N}_{\mathrm{oVG}}$ when (A),(B) and (C) hold.

Next we give lower bounds on probability of (A),(B) and (C). The probability of (A) is at least $1 - ne^{-t_{\text{shld}}/4} > 1 - ne^{-\delta\tau}$ from Lemma 36. The probability of (B) is at least $1 - ne^{-t_{\text{virus}}/4} = 1 - ne^{-\delta\tau}$ from Lemma 31. At each interaction, a new virus is created with probability at most p (with probability exact p when a leader interacts as an initiator and with probability 0 otherwise). Hence, the probability of (C) is at least $1 - 2mt_{\text{shld}} \cdot p$. Thus, Conditions (A),(B) and (C) hold with probability at least $1 - 2ne^{-\delta\tau} - 2mt_{\text{shld}} \cdot p$. □

Lemma 38. *Let $C_0 \in \mathcal{N}_{\text{oVG}}$ and $\Xi_{P_{\text{RD}}}(C_0) = C_0, C_1, \ldots$. Then, we have* $\Pr(\exists i \in [0, 16m\delta\tau\lceil\log n\rceil],\ C_i \in \mathcal{N}_{\text{oVG}} \cap \mathcal{L}) \geq 1 - 2ne^{-\delta\tau}$.

Proof . The lemma trivially holds if C_0 has one or more leaders. Therefore, we consider the case C_0 does not have any leader (i.e. $C_0 \notin \mathcal{L}$). Since followers never create viruses or shields, there exists neither a virus nor a shield until a leader is created. Therefore, the population reaches a configuration of $\mathcal{N}_{\text{oVG}} \cap \mathcal{L}$ at the first timeout of execution $\Xi_{P_{\text{RD}}}(C_0) = C_0, C_1, \ldots$.

Thus, it suffices to show that a timeout occurs within $16m\delta\tau\lceil\log n\rceil$ interactions with probability at least $1 - 2ne^{-\delta\tau}$. During the period no leader exists, the maximum value of all $v.\texttt{timer}_{\texttt{L}}$ (i.e. $\max_{v\in V} v.\texttt{timer}_{\texttt{L}}$) is monotonically non-increasing and decreases at least by one in each round. This means a timeout occurs until $t_{\max}$ rounds finish. By Lemma 31, we have $\Pr(\mathrm{RT}(t_{\max}) < 16m\delta\tau\lceil\log n\rceil) \geq 1 - ne^{-t_{\max}/4} = 1 - ne^{-2\delta\tau}$. □

Lemma 39. *Let $C_0 \in \mathcal{C}_{\text{all}}(P_{\text{RD}})$ and $\Xi_{P_{\text{RD}}}(C_0) = C_0, C_1, \ldots$. Let t_{init} be the maximum value of all $v.\texttt{timer}_{\texttt{V}}$ in C_0 (i.e. $\max_{v\in V} C(v).\texttt{timer}_{\texttt{V}}$). Then, we have* $\Pr(\forall v \in V,\ C_{2m\tau}(v).\texttt{timer}_{\texttt{V}} > t_{\text{init}} - t_{\max}/2) > 1 - 2ne^{-\tau}$.

Proof . The same argument as the proof of Lemma 21 gives the lemma. □

Lemma 40. *Let $C_0 \in \mathcal{N}_{\text{oVG}} \cap \mathcal{L}$ and $\Xi_{P_{\text{RD}}}(C_0) = C_0, C_1, \ldots$. Then, we have* $\Pr(\exists i \in [0, \lceil 2mp^{-1}\rceil + 2m\tau],\ C_i \in \mathcal{S}_{\text{RD}}) \geq 1 - e^{-2} - 5ne^{-\tau} - 2m\tau \cdot p$.

Proof . Let t be the minimum integer (i.e. the first time) such that configuration C_t has a virus. During the period one or more leaders exist, each interaction makes a new virus with probability at least p/m. Hence, the probability of $t < \lceil 2mp^{-1}\rceil$ is at least $1 - (1 - p/m)^{\lceil 2mp^{-1}\rceil} > 1 - e^{-2}$.

Therefore, it suffices to show that $C_{t+2m\tau} \in \mathcal{S}_{\text{RD}}$ holds with probability at least $1 - 5ne^{-\tau} - 2m\tau \cdot p$. We denote the leader that creates a virus at interaction Γ_{t-1} by v. Note that, in configuration C_t, only v has a virus and a shield while the other agents do not have viruses or shields. Furthermore, the virus and the shield of v have the maximum TTL (t_{virus} and t_{shld} respectively in C_t. We have $C_{t+2m\tau} \in \mathcal{S}_{\text{RD}}$ if all the following conditions hold:

(A) every agent has a virus in $C_{t+2m\tau}$,

(B) every agent except for v does not wear a shield in $C_{t+2m\tau}$,

(C) agent v joins only less than $t_{\mathrm{shld}}/2$ interactions as an initiator during $\Gamma_t, \Gamma_{t+1}, \dots, \Gamma_{t+2m\tau-1}$, and

(D) every agent has $\mathtt{timer_L}$ larger than $t_{\max}/2$ in $C_{t+2m\tau}$.

By (A) and (B), all agents except for v are dead in $C_{t+2m\tau}$. By (C), v always has a shield larger than $t_{\mathrm{shld}}/2$ during the $2m\tau$ interactions, and hence, is alive (i.e. is a leader) in $C_{t+2m\tau}$. Therefore, $C_{t+2m\tau} \in \mathcal{L}_{\mathrm{one}} \cap \mathcal{G}_{\mathrm{half}}$ holds. Moreover, $C_{t+2m\tau} \in \mathcal{L}_{\mathrm{half}}$ holds by (D). Thus, we have $C_{t+2m\tau} \in \mathcal{L}_{\mathrm{one}} \cap \mathcal{G}_{\mathrm{half}} \cap \mathcal{L}_{\mathrm{half}} \subset \mathcal{S}_{\mathrm{RD}}$ when (A),(B),(C) and (D) hold.

Therefore, it suffices to show that all (A),(B),(C) and (D) hold with probability $1 - 5ne^{-\tau} - 2m\tau \cdot p$. Since $C_t(v).\mathtt{timer_V} = t_{\mathrm{virus}} = t_{\max}/2$, the probability of (A) is at least $1 - 2ne^{-\tau}$ by Lemma 39. The sufficient condition of (B) is that a new virus is not created during $\Gamma_t, \Gamma_{t+1}, \dots,$ $\Gamma_{t+2m\tau-1}$. The probability of this condition is at least $1 - 2m\tau \cdot p$. The probability of (C) is at least $1 - ne^{-\delta\tau} > 1 - ne^{-\tau}$ by Lemma 28. Finally, The probability of (D) is at least $1 - 2ne^{-\tau}$ by Lemma 33. Thus, all (A), (B), (C) and (D) hold with probability at least $1 - 5ne^{-\tau} - 2m\tau \cdot p$. $\square$

Lemma 41. $\max_{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}})} \mathrm{ECT}_{P_{\mathrm{RD}}}(C, \mathcal{S}_{\mathrm{RD}}) = O(mp^{-1})$.

Proof . By Lemmas 37, 38 and 40, starting from any configuration of $\mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}})$, the population reaches a configuration of $\mathcal{S}_{\mathrm{RD}}$ within $2mt_{\mathrm{shld}} + 16m\delta\tau\lceil\log n\rceil + \lceil 2mp^{-1}\rceil + 2m\tau$ interactions with probability at least $1 - 2ne^{-\delta\tau} - 2mt_{\mathrm{shld}} \cdot p - 2ne^{-\delta\tau} - e^{-2} - 5ne^{-\tau} - 2m\tau \cdot p$. The former expression is at most $\lceil (2m+1) \cdot p^{-1} \rceil$ and the latter expression is at least $1 - 3mt_{\mathrm{shld}} \cdot p - 6ne^{-\tau} - e^{-2} > 1 - 3/4 - 6e^{-2}/26 - e^{-2} > 0.08$. Hence, we have

$$\max_{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}})} \mathrm{ECT}_{P_{\mathrm{RD}}}(C, \mathcal{S}_{\mathrm{RD}}) \\ \leq \lceil (2m+1)p^{-1} \rceil + 0.92 \cdot \max_{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}})} \mathrm{ECT}_{P_{\mathrm{RD}}}(C, \mathcal{S}_{\mathrm{RD}}).$$

Solving this inequality gives $\max_{C \in \mathcal{C}_{\mathrm{all}}(P_{\mathrm{RD}})} \mathrm{ECT}_{P_{\mathrm{RD}}}(C, \mathcal{S}_{\mathrm{RD}}) = O(mp^{-1})$. $\square$

Lemmas 35 and 41 gives the following theorem.

Theorem 4. *Protocol P_{RD} is a $(O(mp^{-1}), \Omega(\tau e^{\tau}))$ loosely-stabilizing leader election protocol for arbitrary graphs when $t_{\max} \geq 8\delta \max(d, 2 + \log(13n\delta\lceil\log n\rceil))$, $t_{\mathrm{virus}} = t_{\max}/2$, $t_{\mathrm{shld}} \geq 2\delta t_{\max}\lceil\log n\rceil$ and $p \leq (4mt_{\mathrm{shld}})^{-1}$.*

Therefore, given upper bounds N of n and Δ of δ, we get a $(O(m\Delta^2 N^3 \log N), \Omega(Ne^N))$ loosely-stabilizing leader election protocol for arbitrary graphs by assigning $t_{\max} = 8N\Delta$, $t_{\text{virus}} = t_{\max}/2$, $t_{\text{shld}} = 2\Delta t_{\max} \lceil \log N \rceil$ and $p = (4N^2 t_{\text{shld}})^{-1}$.

3.5 Conclusion

The author has presented two loosely-stabilizing leader election protocols for arbitrary undirected graphs in the PP model: one works with agent-identifiers and the other works with random numbers. Both protocols keep a unique leader for an exponentially long expected time after reaching a loosely-safe configuration. The protocols use only upper bounds N of n and Δ of δ while any self-stabilizing leader election protocol needs the exact knowledge of n. The restriction of the protocols to *undirected* graph is only for simplicity of protocol description and complexity analysis. The proposed protocols also work on arbitrary *directed* graphs with slight modification: it is only necessary that a responder also executes some actions of an initiator (Line 1 of Protocol 1 and Lines 10-12 of Protocol 2). Both the two protocols use the timeout mechanism to detect the absence of a leader agent. This mechanism can be regarded as an implementation of oracle $\Omega?$ [12]. Although the oracle does not provide any guarantee about when it notify each agent the absence of a leader, the timeout mechanism does within a certain time with high probability, which leads to loosely-stabilizing solution.

The future work is to develop a loosely-stabilizing leader election protocol without agent-identifiers or random numbers for arbitrary graphs. The author will also tackle with loosely-stabilizing leader election for some classes of graphs (e.g. rings and trees). The author is also interested in the empirical evaluation of the holding time of loosely-stabilizing protocols. Since probabilistic evaluation of the holding time in this chapter is not tight, the actual holding time of the protocols should be much longer. By simulation experiments, the author will empirically evaluate the actual holding time (and convergence time) for various network sizes and graph topologies.

Chapter 4

Optimal Longcut Route Selection

4.1 Introduction

The current adoption rate of wireless mobile devices such as smart phones, tablet computers, and laptop computers is spectacular, and we see rapid spread of cloud computing services, which require the Internet connection inherently. Thus, users of such devices and services need Internet access not only when they stay at home or office, but also when they travel. However, at present, users cannot enjoy fast wireless communication everywhere. The cellular network has wide radio coverage, but it does not provide fast communication. On the other hand, Wireless LAN such as WiFi achieves a high transmission rate, but its radio coverage is narrow. Thus, wireless signal quality is highly dependent on the user's location, and hence, quality of communication during travel highly depends on the travel route that the users take. Therefore, it may be desirable for such users to select a "longcut route" to their destination that has larger travel time than the shortest route, but provides better quality of wireless communication during travel.

In this chapter, the author formulates the above situation as the "optimal longcut route problem". We consider that a user of a wireless mobile device needs to travel from starting location s to destination g within time Δ_t while using the device in an urban area. Wireless communication speeds differ at different locations. Therefore, the total amount of communication during travel differs depending on the route taken from s to g. The goal of this problem is to find the route with the maximum amount of communication subject to the constraint that the travel time is within Δ_t. We call this route an *optimal longcut route.* We define this problem as the following graph problem:

Optimal Longcut Route Selection（OLRS）
[Input]

- Directed graph $G = (V, E)$ where self-loops are allowed but multiple edges are not allowed.
- Starting node $s \in V$ and destination node $g \in V$.
- Travel time function $T : E \to \mathbb{N}^+$ where $T(e) = 1$ holds for any self-loop e.
- Communication amount function $P : E \to \mathbb{N}$.
- Time limit $\Delta_t \in \mathbb{N}^+$.

[Output]

s - g walk [1] $\omega = (v_0, e_1, v_1, \dots, e_l, v_l)$ $(v_0 = s,\ v_l = g)$ with the maximum amount of communication $P(\omega) = \sum_{i=1}^{l} P(e_i)$ subject to $T(\omega) = \sum_{i=1}^{l} T(e_i) \le \Delta_t$.

Self-loop (v, v) means that the user can stay at node $v \in V$. More specifically, we consider that the user stays at v for k time units if the user follows walk ω which has k self-loops (v, v).

Generally, an optimal route of an OLRS instance is not a shortest route from s to g. To achieve better quality of communication, it becomes a “longcut route” that makes a detour to the destination. The improvement of communication quality by longcut has already been evaluated by [21, 22]. The studies performed simulation experiments and evaluated the improvement in an ideal communication model [21] and in a more practical model with network simulator NS2 [22]. Both studies show that a longcut route with a small increase of travel time greatly improves the total amount of communication the user obtains. However, in these papers, we do not present solutions or algorithms for calculating the optimal long-cut route.

Delay Constraint Least Cost problem(DCLC) is a problem closely related to OLRS. DCLC is defined as follows.

Delay Constraint Least Cost(DCLC)
[Input]

- Simple graph $G = (V, E)$.
- Starting node $s \in V$, destination node $g \in V$.

[1] A walk is an alternating sequence of nodes and edges. A node (or an edge) may appear twice or more in a walk.

- Delay function $D : E \to \mathbb{N}^+$.
- Cost function $C : E \to \mathbb{N}$.
- Acceptable delay time $\Delta_d \in \mathbb{N}^+$.

[Output]

s - g walk $\omega = (v_0, e_1, v_1, \ldots, e_l, v_l)$ $(v_0 = s,\ v_l = g)$ with least cost $C(\omega) = \sum_{i=1}^{l} C(e_i)$ subject to acceptable delay time Δ_d.

The main difference between DCLC and OLRS is that we have to find the walk with the minimum cost for DCLC while we have to find the walk with the maximum gain (communication amount) for OLRS. It is known that DCLC is NP-hard [23]. Therefore, solutions for this problem are classified to two types: One finds an optimal solution in an exponential time (in the worst-case) [24], and the other finds an approximate solution in polynomial time [25, 26, 27, 28, 29]. Widyono [24] proposed an algorithm called CBF(Constrained Bellman-Ford), which computes an optimal solution with the branch and bound approach. This algorithm takes exponential time in the worst case, but in practice, it takes a relatively short time. Lorenz et al. [25] proposed a fully polynomial time approximation scheme (FPTAS) for DCLC: an $1 + \varepsilon$-approximation algorithm with $O(|V||E| \log |V| \log\log |V| + |V||E|/\varepsilon)$ time for any $\varepsilon > 0$. Most heuristic methods for this problem use aggregation of the two metrics, delay and cost. They generate a new metric $M = f(D, C)$ [26, 27, 28, 29] by combining delay D and cost C and reduce DCLC to the shortest path problem. Jüttner et al. [26] devised a fast algorithm that finds a nearly optimal solution for DCLC by selecting appropriate f dynamically with Lagrange relaxation. Feng et al. introduced a new method called non-linear Lagrange relaxation and presented an algorithm that finds a better solution than that of [26]. Neve et al. [28] and Guo et al. [29] presented algorithms which obtain solutions with excellent quality by storing multiple routes on each node $v \in V$.

Contribution of This Chapter In this chapter, we start proving that OLRS is NP-hard. Next, we propose two pseudo-polynomial time algorithms named OLRS$_1$ and OLRS$_2$ for this problem. The worst case time complexity of OLRS$_1$ and OLRS$_2$ are $\Theta(\Delta_t \cdot |E|)$ and $O(\Delta_t^2 \cdot |E| \log(\Delta_t \cdot |E|))$ respectively. Then, the author performs simulation experiments to evaluate the execution time of the proposed algorithms in practical settings representing urban areas. From the results, we observe that both algorithms solve the problem within a sufficiently short time even for large graphs. Also, we find that the execution time of OLRS$_2$ does not depend on time limit Δ_t, while that of OLRS$_1$ strictly depends Δ_t. Thus, OLRS$_2$ is faster than OLRS$_1$ in the experiments whereas OLRS$_1$ is asymptotically faster than OLRS$_2$ for the worst-case inputs.

4.2 Preliminaries

In this section, we introduce some expressions and notations. Some of them including those defined in Section 4.5.1 are listed in Table 4.1.

We denote the number of nodes and the number of edges of G by n and m respectively. The set of *incoming and outgoing neighboring nodes* of node v are defined as $N^+_{\text{in}}(v) = \{u \mid (u, v) \in E\}$ and $N^+_{\text{out}}(v) = \{u \mid (v, u) \in E\}$ respectively. Furthermore, we define $N^-_{\text{in}}(v) = N^+_{\text{in}}(v) \setminus \{v\}$ and $N^-_{\text{out}}(v) = N^+_{\text{out}}(v) \setminus \{v\}$.

A *walk* $\omega = (v_0, e_1, v_1, e_2, \ldots, e_l, v_l)$ is an alternating sequence of nodes and edges of G where $e_i = (v_{i-1}, v_i)$ for all $i = 1, \ldots, l$. Since G has no multiple edges, we sometimes use simplified representation $\omega = (v_0, v_1, \ldots, v_l)$, which omit the edges of ω. We denote the last node v_l by ω.end. For any node $v \in N^+_{\text{out}}(\omega.\text{end})$, we define $\omega + v$ as $(v_0, e_1, \ldots, v_l, (v_l, v), v)$. For any walk $\omega' = (u_0, d_1, \ldots, d_k, u_k)$ satisfying $u_0 = \omega.\text{end}\ (= v_l)$, we define $\omega+\omega'$ as $(v_0, e_1, \ldots, e_l, v_l, d_1, \ldots, d_k, u_k)$. We introduce *null walk* ϵ for convenience and define $\epsilon + \omega = \omega + \epsilon = \omega$ for any walk ω.

We define the *travel time* of walk $\omega = (v_0, e_1, \ldots, e_l, v_l)$ and *total amount of communication* of ω as $T(\omega) = \sum_{i=1}^{l} T(e_i)$ and $P(\omega) = \sum_{i=1}^{l} P(e_i)$ respectively. For any different two nodes u and v, we denote by $T_{\min}(u, v)$ the minimum travel time among all u - v walks. For convenience, we define $T_{\min}(v, v) = 0$ for any $v \in V$. The set of s - v walks ω satisfying $T(\omega) \le t$ is denoted by $S_{\text{walk}}(v, t)$.

Let v be a node in V and t be a time in $[0, \Delta_t]$. We define the maximum amount of communication from s to v within time t as $P_{\text{opt}}(v, t) = \max\{P(\omega) \mid \omega \in S_{\text{walk}}(v, t)\}$. We define $P_{\text{opt}}(v, t) = 0$ when $S_{\text{walk}}(v, t) = \emptyset$ holds. We call a walk ω satisfying $P(\omega) = P_{\text{opt}}(v, t)$ by an *optimal longcut route* from s to v at time t. We denote the set of such walks by $\text{OPT}(v, t)$. Giving input $(G, s, g, T, P, \Delta_t)$, problem OLRS requires us to find any one of $\text{OPT}(g, \Delta_t)$.

4.3 NP-hardness

In this section, we prove that OLRS is NP-hard. To prove the NP-hardness, we show a polynomial-time reduction from Unbounded Knapsack Problem (UKP), which is NP-hard [30], to OLRS. UKP is a variant of the knapsack problem where the number of each item packed to a knapsack is not restricted. The problem is formulated as follows.

Unbounded Knapsack Problem (UKP)
[Input]

Table 4.1: Expressions and notations we use in this chapter

$N^+_{\text{in}}(v)$	$\{u \mid (u,v) \in E\}$
$N^+_{\text{out}}(v)$	$\{u \mid (v,u) \in E\}$
$N^-_{\text{in}}(v)$	$N^+_{\text{in}}(v) \setminus \{v\}$
$N^-_{\text{out}}(v)$	$N^+_{\text{out}}(v) \setminus \{v\}$
ω.end	the last node of walk ω
$P(\omega)$	$\sum_{i=1}^{l} P(e_i)$ (ω is a walk $(v_0, e_1, \dots, e_l, v_l)$.)
$T(\omega)$	$\sum_{i=1}^{l} T(e_i)$ (ω is a walk $(v_0, e_1, \dots, e_l, v_l)$.)
$T_{\min}(u,v)$	$\min\{T(\omega) \mid \omega$ is a u - v walk$\}$
$S_{\text{walk}}(v,t)$	The set of s - v walks satisfying $T(\omega) \leq t$
$P_{\text{opt}}(v,t)$	$\max\{P(\omega) \mid \omega \in S_{\text{walk}}(v,t)\}$
$\text{OPT}(v,t)$	$\{\omega \in S_{\text{walk}}(v,t) \mid P(\omega) = P_{\text{opt}}(v,t)\}$
$P_{\max}(\omega)$	$\max_{0 \leq i \leq l} P(v_i, v_i)$ (ω is a nonstop walk $(v_0, e_1, \dots, e_l, v_l)$.)
$P_{\text{ex}}(\omega, t)$	$P(\omega) + P_{\max}(\omega) \cdot (t - T(\omega))$ (ω is a nonstop walk.)
$\omega >_i \omega'$	$(P_{\text{ex}}(\omega, i) = P_{\text{ex}}(\omega', i) \land P_{\max}(\omega) > P_{\max}(\omega')) \lor P_{\text{ex}}(\omega, i) > P_{\text{ex}}(\omega', i)$
$S_{\text{ns}}(v,t)$	the set of nonstop s - v walks ω satisfying $T(\omega) \leq t$
$\text{MX}_>(v,i)$	$\{\omega \in S_{\text{ns}}(v,i) \mid \forall \omega' \in S_{\text{ns}}(v,i),\ \neg(\omega' >_i \omega)\}$
$\text{CT}(v)$	$\{t \in [T_{\min}(s,v), \Delta_t] \mid \text{MX}_>(v, t-1) \cap \text{MX}_>(v,t) = \emptyset\}$

- A set of items $A = \{a_1, \dots, a_k\}$.
- Weight function $W : A \to \mathbb{N}^+$.
- Value function $P' : A \to \mathbb{N}$.
- Maximum weight $\Delta_w \in \mathbb{N}^+$.

[Output]

k tuple $(x_1, \dots, x_k) \in \mathbb{N}^k$ with the maximum value $P'(x_1, \dots, x_k) = \sum_{i=1}^{k} x_i P'(a_i)$ subject to $W(x_1, \dots, x_k) = \sum_{i=1}^{k} x_i W(a_i) \leq \Delta_w$.

Theorem 5. *OLRS is NP-hard.*

Proof . We present a polynomial time reduction from UKP to OLRS as follows.

Input transformation from a UKP instance $(A = (a_1, \ldots, a_k), W, P', \Delta_w)$ **to an OLRS instance** $(G(V, E), s, g, T, P, \Delta_t)$

- $V = \{v_1, \ldots, v_k, s, g\}$.
- $E = \bigcup_{i=1}^{k} \{(s, v_i), (v_i, s)\} \cup \{(s, g)\}$
- $T(s, v_i) = T(v_i, s) = W(a_i)$ and $T(s, g) = 1$
- $P(s, v_i) = P(v_i, s) = P'(a_i)$ and $P(s, g) = 0$.
- $\Delta_t = 2\Delta_w + 1$

Output translation from an OLRS solution, walk ω**, to a UKP solution** $(x_1, \ldots, x_k) \in \mathbb{N}^k$ Given walk ω, the solution of UKP is $(x_1, \ldots, x_k) = (\#(\omega, 1), \ldots, \#(\omega, k))$ where $\#(\omega, i)$ is the number of occurrences of v_i in ω. For example, we have $(x_1, x_2, x_3) = (2, 1, 3)$ when $\omega = (s, v_1, s, v_1, s, v_2, s, v_3, s, v_3, s, v_3, s, g)$.

It is trivial that both the transformation can be performed within polynomial time. Hence, it suffices to prove that the result $(x_1, \ldots, x_k)$ is the optimal solution of UKP with input instance $(A, W, P'\Delta_w)$.

Assume that $(x_1, \ldots, x_k)$ is not the optimal solution. Then, some k-tuple $(y_1, \ldots, y_k)$ exists such that $\sum_{i=1}^{k} y_i W(a_i) \leq \Delta_w$ and $\sum_{i=1}^{k} y_i P(a_i) > \sum_{i=1}^{k} x_i P(a_i)$. This contradicts the fact that ω is the optimal solution of OLRS with input instance $(G, s, g, T, P, \Delta_t)$ by the following reason. For walk ψ whose first and last nodes are the same, we define ψ^i as (i)$\psi^0 = \epsilon$ and (ii) $\psi^i = \psi^{i-1} + \psi$ $(i \geq 1)$. Let walk ω'_i be $(s, v_i, s)^{y_i}$ for any $i = 1, \ldots, k$. Then, $\omega' = \omega'_1 + \cdots + \omega'_k + (s, g)$ is a s - g walk and satisfies $T(\omega) = 2\sum_{i=1}^{k} y_i W(a_i) + 1 \leq 2\Delta_w + 1 = \Delta_t$, which means ω' is a feasible solution. However, $P(\omega') = 2\sum_{i=1}^{k} y_i P(a_i) > 2\sum_{i=1}^{k} x_i P(a_i) = P(\omega)$ holds, which contradicts the optimality of ω. Hence, $(x_1, \ldots, x_k)$ is optimal for UKP with input instance $(A, W, P'\Delta_w)$. □

4.4 Algorithm OLRS_1

In this section, we present OLRS_1 and show its time complexity. In the following, we describe as $\mathrm{OPT}_{\mathrm{just}}(v, t)$ the set of s - v walks ω with travel time exactly t and with maximum amount of communication. More specifically, $\mathrm{OPT}_{\mathrm{just}}(v, t) = \{\omega \in S_{\mathrm{just}}(v, t) \mid \forall\psi \in S_{\mathrm{just}}(v, t),\ P(\omega) \geq$

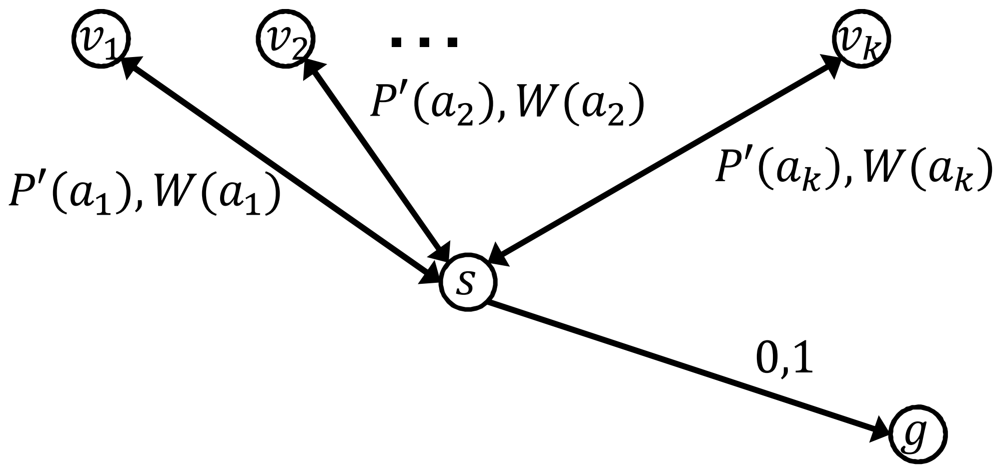


Figure 4.1: A problem instance of OLRS obtained by the transformation from UKP to OLRS. The first and the second elements of the label of each edge e represents $P(e)$ and $T(e)$ respectively.

$P(\psi)\}$ where $S_{\text{just}}(v,t) = S_{\text{walk}}(v,t) \setminus S_{\text{walk}}(v,t-1)$. In addition, we denote the communication amount of walks in $\mathrm{OPT}_{\text{just}}(v,t)$ as $P_{\text{just}}(v,t)$. We consider $P_{\text{just}}(v,t) = -\infty$ when $\mathrm{OPT}_{\text{just}}(v,t) = \emptyset$.

Algorithm OLRS_1 consists of $\Delta_t + 1$ steps from Step 0 to Step Δ_t (Code 4.1). At each Step i, the algorithm computes, for each node $v \in V$, any one walk of $\mathrm{OPT}_{\text{just}}(v,i)$ and stores the walk on variable $v.\mathrm{opt}(i)$. Step 0 exists for initialization. At Step 0, the algorithm set $v.\mathrm{opt}(i) = \mathrm{NULL}$ for any $v \in V$ and $i \in [0, \Delta_t]$, and then, stores initial walk (s) on $s.\mathrm{opt}(0)$ (Lines 1-2). At Step $i > 0$, the algorithm computes walk $u.\mathrm{opt}(i - T(u,v)) + v$ for every $u \in N^+_{\text{in}}(v)$, and then, stores walk ω with maximum $P(\omega)$ among the computed walks on $v.\mathrm{opt}(i)$. The correctness of this step is guaranteed by the following equation, which trivially holds for any positive integer $i \geq T_{\min}(s,v)$.

$$\begin{aligned} P_{\text{just}}(v,i) = \max\{&P_{\text{just}}(u, i - T(u,v)) + P(u,v) \\ &\mid u \in N^+_{\text{in}}(v),\ i \geq T(u,v)\} \end{aligned} \tag{4.1}$$

At the end of Step Δ_t, we have $P_{\text{opt}}(g, \Delta_t) = \max_{i \in [0,\Delta_t]} P(g.\mathrm{opt}(i))$. Therefore, we can obtain the optimal solution by selecting the walk with the maximum amount of communication among the walks stored on $g.\mathrm{opt}(i)$ for $i = 0, \ldots, \Delta_t$ (Line 11).

A walk is expressed by a linked list. Specifically, each variable $v.\mathrm{opt}(i)$ does not store the entire walk $\omega = (v_1, \ldots, v_l)$ $(v_1 = s,\ v_l = v)$ but only have the pointer to variable $v_{l-1}.\mathrm{opt}(i - T(v_{l-1}, v))$ and the value of $P(\omega)$. Thus, the comparing and the update of $v.\mathrm{opt}()$ at Lines 7-8 can be easily executed within $O(1)$ time.

Code 4.1 Optimal Longcut Route Selection OLRS_1

```
Step 0
 1: $v$.opt($i$) := NULL for every $v \in V$ and $i \in [0, \Delta_t]$
 2: $s$.opt(0) := ($s$)                         // ($s$) is a walk composed of only node $s$.
Step i ($1 \le i \le \Delta_t$)
 3: for each $v \in V$ do
 4:   for each $u \in N^+_{\mathrm{in}}(v)$ do
 5:     if $i < T(u, v)$ then continue the for-loop
 6:     if $u$.opt($i - T(u, v)$) = NULL then continue the for-loop
 7:     $\omega := u$.opt($i - T(u, v)) + v$
 8:     if $v$.opt($i$) = NULL or $P(\omega) > P(v$.opt($i$))
                                       then $v$.opt($i$) := $\omega$
 9:   end for
10: end for
11: if $i = \Delta_t$ then return $g$.opt($t$) such that $P(g$.opt($t$)) = $\max_{i \in [0,\Delta_t]} P(g$.opt($i$))
```

Theorem 6. *Algorithm* OLRS_1 *solves OLRS with time complexity of* $O(\Delta_t \cdot m)$.

Proof . The correctness of the algorithm is trivial from expression (4.1). Step 0 takes $O(\Delta_t \cdot n)$ time. Each Step $i > 0$ takes $O(m)$ time since one execution of the loop at Lines 4-9 takes $O(1)$ time, and the loop are executed at most $\sum_{v \in V} |N^+_{\mathrm{in}}(v)| = m$ times. Hence, the time complexity of OLRS_1 is $O(\Delta_t \cdot m)$. □

For the simulation of Section 4.6, we modify OLRS_1 so that it calculates $T_{\min}(s, v)$ for all $v \in V$ at Step 0, and the loop of Lines 3-10 is executed only for $v \in V$ satisfying $i \ge T_{\min}(s, v)$ at Step $i > 0$. This modification reduces the execution time of OLRS_1 to a certain extent.

4.5 Algorithm OLRS_2

Algorithm OLRS_1 spends $O(|N^+_{\mathrm{in}}(v)| \cdot \Delta_t)$ time for each $v \in V$ because it stores one of $\mathrm{OPT}(v, i)$ on v.opt(i) for every $i = 0, \ldots, \Delta_t$. A walk stored on v.opt(i) is utilized later to compute u.opt($i + T(v, u)$) for some $u \in N^+_{\mathrm{out}}(v)$. However, v.opt(i) are not necessary for all $i \in [0, \Delta_t]$. For example, consider the situation that the walks stored on v.opt(21), v.opt(22), $\ldots$, v.opt(50) are the same except for the number of self-loops. Then, we need not store all the walks using 30 variables. Instead, we can express this situation by only one walk that we obtain by removing all self-loops from the walks. Algorithm OLRS$_2$ uses this techniques to achieve time complexity that does not depend on Δ_t for some class of problem instances. In what follows, we define $P(v, v) = 0$ for convenience if node v does not have a self-loop.

4.5.1 Nonstop Walks

We introduce (maximal) *nonstop walks* as the key concept of OLRS$_2$. The set of *changing times* $\mathrm{CT}(v)$ for node v is also introduced, the size of which is utilized to bound the time complexity of the algorithm.

A walk is called *nonstop* if the walk has no self-loop. For nonstop walk $\omega = (v_0, v_1, \dots, v_l)$, we define *the stopping node* of ω as the node v_i with the maximum $P(v_i, v_i)$. If multiple nodes have the maximum $P(v_i, v_i)$, we adopt the node v_i with the minimum index i among the nodes. A walk $f(\omega, k)$ is the walk obtained from ω by inserting k self-loops at the stopping node v_i. For example, $f(\omega, 3)$ is $(v_1, v_2, v_2, v_2, v_2, v_3)$ when $\omega = (v_1, v_2, v_3)$ and the stopping node of ω is v_2. We define as $P_{\max}(\omega) = P(v_i, v_i)$, that is, $P_{\max}(\omega)$ is the communication amount of the stopping node of ω. If no node on ω has self-loop, we define $f(\omega, k) = \omega$ and $P_{\max}(\omega) = 0$. For any $t \in [T(\omega), \Delta_t]$, we define *the expanded communication amount* of ω as $P_{\mathrm{ex}}(\omega, t) = P(f(\omega, t - T(\omega)))$. Note that $P_{\mathrm{ex}}(\omega, t) = P(\omega) + P_{\max}(\omega) \cdot (t - T(\omega))$ holds.

We define binary relation $>_t$ for nonstop walks ω and ψ with travel times t or less as follows:

$$\begin{aligned}\omega >_t \psi \Leftrightarrow & \,(P_{\mathrm{ex}}(\omega, t) = P_{\mathrm{ex}}(\psi, t) \wedge P_{\max}(\omega) > P_{\max}(\psi)) \\ & \vee P_{\mathrm{ex}}(\omega, t) > P_{\mathrm{ex}}(\psi, t)\end{aligned}$$

For set X of nonstop walks with travel time t or less, we say that $\omega \in X$ is *maximal* in X about $>_t$ if $\psi >_t \omega$ does not hold for any $\psi \in X$. We define $\mathrm{MX}_>(v, t)$ as the set of maximal walks in $S_{\mathrm{ns}}(v, t)$ concerning $>_t$ where $S_{\mathrm{ns}}(v, t)$ is the set of all s - v nonstop walks with travel time t or less. Suppose that $S_{\mathrm{ns}}(v, \Delta_t) = \{\omega_a, \omega_b, \omega_c, \omega_d\}$ and the expanded communication amount of the four walks are those depicted in Fig. 4.2. Then, ω_a, ω_b and ω_c belongs to $\mathrm{MX}_>(v, t)$ for time $t \in [4, 6]$, $t \in [7, 9]$ and $t \in [10, 17]$ respectively, while ω_d is not maximal for any time. The goal of OLRS$_2$ is to find a walk $\omega \in \mathrm{MX}_>(g, \Delta_t)$, by which we get the optimal solution $f(\omega, \Delta_t - T(\omega)) \in \mathrm{OPT}(g, \Delta_t)$.

In Fig. 4.2, the maximal walk of node v changes at times 4, 7 and 10. We call such a time by a *changing time* of v, and denote the set of the changing times by $\mathrm{CT}(v)$. More specifically, we define as $\mathrm{CT}(v) = \{t \in [T_{\min}(s, v), \Delta_t] \mid \mathrm{MX}_>(v, t-1) \cap \mathrm{MX}_>(v, t) = \emptyset\}$. Note that $T_{\min}(s, v)$

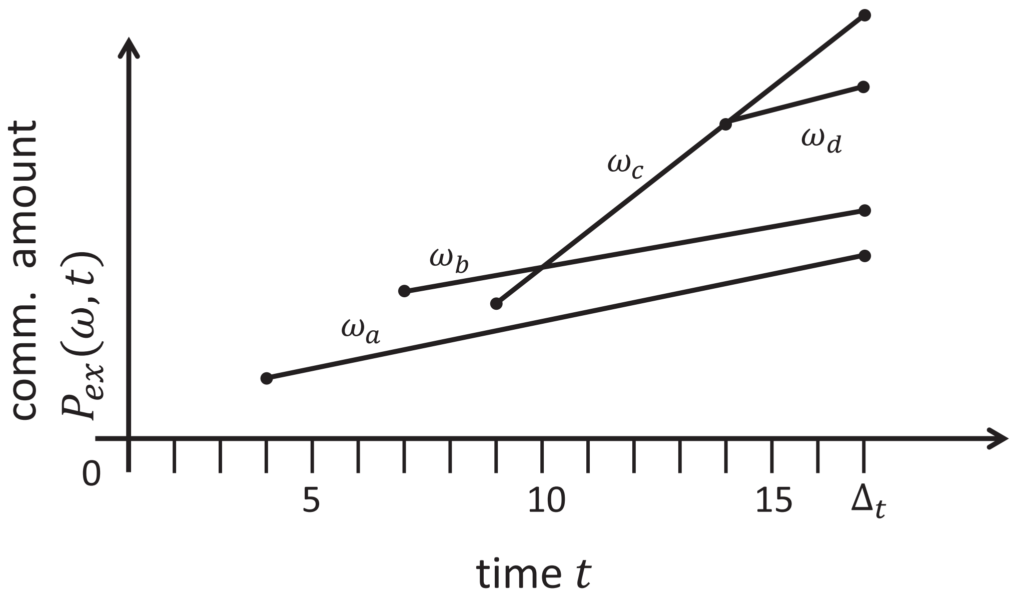


Figure 4.2: The expanded communication amount of s - v walks

always belongs to $\mathrm{CT}(v)$. In Section 4.5.2, we design OLRS_2 so that its time complexity is bounded by a polynomial function of $\max_{v \in V} |\mathrm{CT}(v)|$. Note that $\mathrm{CT}(v)$ is uniquely determined by a problem instance of OLRS and independent from algorithms.

4.5.2 Algorithm OLRS_2

Algorithm OLRS_2, shown in Code 4.2, finds walk $\omega \in \mathrm{MX}_>(g, \Delta_t)$ and outputs $f(\omega, \Delta_t - T(\omega))$ as the optimal solution of OLRS. To find a walk in $\mathrm{MX}_>(g, \Delta_t)$, the algorithm constructs nonstop walks in the increasing order of their travel times by utilizing heap H that stores a set of nonstop walks. First, the algorithm inserts the initial walk (s) to H (Line 4). Then, until H becomes empty, it repeats the following: extract a nonstop walk ω with the minimum travel time from H, expand ω to generate a nonstop walk $\omega + u$ for every $u \in N^-_{\mathrm{out}}(\omega.\mathrm{end})$, and insert it to H if its travel time is Δ_t or less (Lines 5-27). When H becomes empty, the algorithm just selects the maximal walk about $>_{\Delta_t}$ among the s - g walks it ever generates, which must be a walk of $\mathrm{MX}_>(g, \Delta_t)$. To reduce the execution time, the algorithm expands a nonstop walk ω only if $\omega \in \mathrm{MX}_>(v, t)$ for some v and t. Thus, the number of walk-expansions is bounded by $\sum_{v \in V}(|\mathrm{CT}(v)|)$ (Lemma 45 in Section 4.5.4).

A walk ω with the minimum travel time is extracted from H by invoking $\mathrm{extract}(H)$ (Line 5).

Code 4.2 Optimal Longcut Route Selection OLRS$_2$

```
{Initially, $H = \emptyset$ and $v$.max = NULL for every $v \in V$}
Create a new walk $(s)$ that consists of only one node $s$.
$(s)$.time := 0
insert($H$, $(s)$)                                        // insert the initial walk to $H$
while $H \neq \emptyset$ do
  $\omega$ := extract($H$)
  $i$ := $\omega$.time                                      // n ow at Period $i$
  $v$ := $\omega$.end
  if $v$.max = NULL or $\omega >_i v$.max then
    if $\omega$ is not marked as "already-expanded" then
      Mark $\omega$ as "already-expanded"
      for each $u \in N^-_{\text{out}}(v)$ s.t. $i + T(v, u) \leq \Delta_t$ do
        Create a new walk $\omega + u$
        $(\omega + u)$.time := $i + T(v, u)$
        insert($H, \omega + u$)
      end for
    end if
    $\omega_{\text{lost}}$ := $v$.max                               // t he old $v$.max is lost
    $v$.max := $\omega$                                    // $v$.max is updated
  else
    $\omega_{\text{lost}}$ := $\omega$                                         // $\omega$ is lost
  end if
  if $\omega_{\text{lost}} \neq$ NULL and $\exists j \in [i + 1, \Delta_t]$, $\omega_{\text{lost}} >_j v$.max then
    $\omega_{\text{lost}}$.time := $\min\{j \in [i + 1, \Delta_t] \mid \omega_{\text{lost}} >_j v.\text{max}\}$
    insert($H, \omega_{\text{lost}}$)
  end if
end while
return $f(g.\text{max}, \Delta_t - T(g.\text{max}))$
```

At this time, we do not use $T(\omega)$ as the travel time of ω. Instead, we use variable ω.time for every generated nonstop walk ω. The value of ω.time may change during execution, but $\omega.\text{time} \geq T(\omega)$ always holds. Let i be the minimum value of ω.time among all the walks in H. Heap H also guarantees that the maximal walk concerning $>_i$ in the set of walks in H is extracted. From the characteristic of heap H, $\omega.\text{time} \leq \psi.\text{time}$ holds if ω is extracted before ψ, that is, the travel times of the walks extracted from H are monotonically non-decreasing. In what follows, we call the period from the time a walk with travel time i is extracted at the first time until the time a walk with travel time $j > i$ is extracted at the first time "Period i". These period numbers can be skipped: Period 5 can be immediately followed by Period 13 (not Period 6).

Algorithm OLRS$_2$ uses variables v.max for every node $v \in V$. The initial values of them are "NULL". The goal of Period i is to find any one of $\text{MX}_>(v, i)$ and store it on v.max for every $v \in V$. During Period i, the algorithm extracts all the non-stop walks with travel time i from H in descending order of $>_i$. It handles each ω of the walks as follows: if $\omega >_i v.\text{max}$ holds where $v = \omega.\text{end}$, then store ω on v.max (Line 19) and expand ω unless the walk is already expanded before (Lines 10-17). Furthermore, letting ω_{lost} be the loser of the comparison at Line 9 , the algorithm re-inserts ω_{lost} to heap H if $\omega_{\text{lost}} >_j v.\text{max}$ holds for some $j > i$ (Line 23-26). This is because ω_{lost} may be one of $\text{MX}_>(v, t)$ for some t, and if so, should be stored on v.max at Period j. In the example of Fig. 4.2, ω_c is lost by ω_b at Period 9. After that, ω_c is re-inserted to H with update of $T(\omega_c) = 10$ because $\omega_c >_{10} \omega_b$ holds and hence, ω_c may be the unique walk of $\text{MX}_>(v, 10)$. Thus, the algorithm at Period i can update v.max for every $v \in V$. At the same time, it keeps at least one walk of $\text{MX}_>(v, j)$ for each $j > i$ on either heap H or v.max. As a result, $g.\text{max} \in \text{MX}_>(g, \Delta_t)$ is guaranteed at the end of Period Δ_t (Lemma 44).

A non-stop walk is expressed by a linked list. An object corresponding to walk ω has its last node ω.end and the pointer to the object of the walk from which ω is expanded. An object of ω also has the values of $P(\omega)$, $P_{\max}(\omega)$ and $T(\omega)$. When creating the initial walk (s) at Line 2, we set $P((s)) = 0$, $P_{\max}((s)) = P(s, s)$, and $T((s)) = 0$. When constructing a walk $\omega + v$ from walk ω at Line 13, we only set $(\omega + v)$.end to v, add the pointer from $\omega + v$ to ω, and calculate $P(\omega + v)$, $P_{\max}(\omega + v)$, and $T(\omega + v)$. This expansion can be done within $O(1)$ time because it does not need to make a copy of entire ω, and $P(\omega + v)$, $P_{\max}(\omega + v)$, and $T(\omega + v)$ are calculated by $P(\omega + v) = P(\omega) + P(w.\text{end}, v)$, $P_{\max}(\omega + v) = \max\{P_{\max}(\omega), P(v, v)\}$, and $T(\omega + v) = T(\omega) + T(\omega.\text{end}, v)$, respectively. Also, comparison $>_i$ of two non-stop walks can be done within $O(1)$ time since every non-stop walk ω store the value of $P(\omega)$, $P_{\max}(\omega)$ and $T(\omega)$. Thus, both creating a new walk and comparison $>_i$ of two non-stop walks can be done with $O(1)$ time.

4.5.3 Correctness

In the following, H_i denotes the set of non-stop walks existing in heap H at the end of Period i. Similarly, $\omega_{v,i}$ denotes the value of v.max at the end of Period i. In this subsection, we prove that $\omega_{v,i} \in \mathrm{MX}_>(v,i)$ holds for any $v \in V$ and $i \in [T_{\min}(s,v), \Delta_t]$, which guarantees correctness of OLRS$_2$.

Lemma 42. *Let v be a node and i be an integer not greater than Δ_t. If there exist ω and j $(j < i)$ such that $\omega \in H_j$ and $\omega \in \mathrm{MX}_>(v,i)$, then $\omega_{v,i} \in \mathrm{MX}_>(v,i)$ holds.*

Proof . We define predicate $P(k)$ as $P(k) \Leftrightarrow$ "some $\omega \in (H_k \cup \{\omega_{v,k}\})$ belongs to $\mathrm{MX}_>(v,i)$". We prove the lemma by showing $P(j) \Rightarrow (w_{v,i} \in \mathrm{MX}_>(v,i))$. Since $P(i-1)$ directly leads to $\omega_{v,i} \in \mathrm{MX}_>(v,i)$, it suffices to prove $P(k) \Rightarrow P(k+1)$ holds for any $k \in [j, i-2]$. Assume that $P(k)$ holds and let ω be a walk in $(H_k \cup \{\omega_{v,k}\})$ that belongs to $\mathrm{MX}_>(v,i)$.

Case 1. $\boldsymbol{\omega \in H_k \wedge (\omega.\mathrm{time} > k+1)}$ Since ω.time is more than $k+1$, ω is not extracted from H during Period $k+1$. Hence, ω also belongs to H_{k+1}.

Case 2. $\boldsymbol{(\omega = \omega_{v,k}) \vee (\omega.\mathrm{time} = k+1)}$ If $\omega_{v,k+1} \in \mathrm{MX}_>(v,i)$, then $P(k+1)$ clearly holds. Consider the case of $\omega_{v,k+1} \notin \mathrm{MX}_>(v,i)$. This means that ω is lost by $\omega_{v,k+1}$ during Period $k+1$. Then ω is re-inserted to heap H because $\omega >_i \omega_{v,k+1}$. Hence, we have $\omega \in H_{k+1}$, that leads to $P(k+1)$. □

Lemma 43. *Suppose that nonstop s - v walk $\omega = (v_0, e_1, \ldots, e_l, v_l)$ $(l \geq 1)$ belongs to $\mathrm{MX}_>(v,i)$ for some $i > 0$. Then, there exists integer $j \geq 0$ such that $\psi + v \in \mathrm{MX}_>(v,i)$ holds for any walk $\psi \in \mathrm{MX}_>(v_{l-1}, j)$.*

Proof . We prove the lemma by considering two cases $P_{\max}(\omega) = P(v,v)$ and $P_{\max}(\omega) > P(v,v)$. In what follows, let $\omega - v = (v_0, e_1, \ldots, e_{l-1}, v_{l-1})$.

Case 1. $\boldsymbol{P_{\max}(\omega) = P(v,v)}$ Let $j = T(\omega - v)$ for this case. For any walk $\psi \in \mathrm{MX}_>(v_{l-1}, j)$, we have

$$\begin{aligned}
P_{\mathrm{ex}}(\omega, i) &= P_{\mathrm{ex}}(\omega - v, j) + P(e_l) + P(v,v)(i - j - T(e_l)) \\
&\leq P_{\mathrm{ex}}(\psi, j) + P(e_l) + P(v,v)(i - j - T(e_l)) \\
&\leq P_{\mathrm{ex}}(\psi + v, i)
\end{aligned}$$

and $P_{\max}(\omega) \leq P_{\max}(\psi + v)$. Since ω is maximal concerning $>_i$, we obtain $\psi + v \in \mathrm{MX}_>(v,i)$.

Case 2. $\boldsymbol{P_{\max}(\omega) > P(v,v)}$ Let $j = i - T(e_l)$ for this case. For any walk $\psi \in \mathrm{MX}_>(v_{l-1}, j)$, we have

$$\begin{aligned} P_{\mathrm{ex}}(\omega, i) &= P_{\mathrm{ex}}(\omega - v, j) + P(e_l) \\ &\le P_{\mathrm{ex}}(\psi, j) + P(e_l) \\ &\le P_{\mathrm{ex}}(\psi + v, i) \ . \end{aligned} \tag{4.2}$$

The maximality of ω brings $P_{\mathrm{ex}}(\omega, i) = P_{\mathrm{ex}}(\psi + v, i)$. Hence, "$\le$"s in expression (4.2) become "="s, by which we obtain $P_{\mathrm{ex}}(\omega - v, j) = P_{\mathrm{ex}}(\psi, j)$. This leads to $P_{\max}(\omega - v) \le P_{\max}(\psi)$ since ψ belongs to $\mathrm{MX}_>(v_{l-1}, j)$. Hence we get $P_{\max}(\omega) \le P_{\max}(\psi + v)$. Thus, we have $P_{\mathrm{ex}}(\omega, i) = P_{\mathrm{ex}}(\psi + v, i)$ and $P_{\max}(\omega) \le P_{\max}(\psi + v)$, which brings $\psi + v \in \mathrm{MX}_>(v, i)$. □

Lemma 44. *Predicate* $\mathrm{MX}_>(v, t) \ne \emptyset \Rightarrow \omega_{v,i} \in \mathrm{MX}_>(v, i)$ *holds for any* $i \in [0, \Delta_t]$ *and* $v \in V$.

Proof . We prove the lemma by induction of i.

Initial Phase ($\boldsymbol{i = 0}$) The initial walk (s) is only the walk with travel time 0. Hence, we have $\omega_{s,0} = (s) \in \mathrm{MX}_>(s, 0)$. The predicate obviously holds for any node v other than s because of $\mathrm{MX}_>(v, 0) = \emptyset$.

Induction Phase Let v be a node such that $\mathrm{MX}_>(v, i) \ne \emptyset$. We prove $\omega_{v,i} \in \mathrm{MX}_>(v, i)$ holds under the inductive assumption that $\mathrm{MX}_>(u, j) \ne \emptyset \Rightarrow \omega_{u,j} \in \mathrm{MX}_>(u, j)$ holds for any $j \le i - 1$ and node $u \in V$. If $v = s$ and $\mathrm{MX}_>(v, i) = \{(s)\}$, then $\omega_{v,i}$ is clearly (s). Thus, we have $\omega_{v,i} \in \mathrm{MX}_>(v, i)$. Otherwise, at least one walk $\omega = (v_0, \dots, v_l) \in \mathrm{MX}_>(v, i)$ is not the initial walk (s), and hence, has two or more nodes. Then, from Lemma 43, there exists integer $k < i$ such that $\psi + v$ belongs to $\mathrm{MX}_>(v, i)$ for any $\psi \in \mathrm{MX}_>(v_{l-1}, k)$. On the other hand, $\omega_{v_{l-1},k} \in \mathrm{MX}_>(v_{l-1}, k)$ is obtained from the inductive assumption. Since $\omega_{v_{l-1},k}$ is expanded before the end of Period k, walk $\omega_{v_{l-1},k} + v \in \mathrm{MX}_>(v, i)$ is inserted to H before the end of Period $k < i$. By Lemma 42, we obtain $\omega_{v,i} \in \mathrm{MX}_>(v, i)$. □

Theorem 7. *Algorithm* OLRS_2 *solves OLRS, that is, it finds a walk in* $\mathrm{OPT}(g, \Delta_t)$.

Proof . By Lemma 44, $g.\max \in \mathrm{MX}_>(g, \Delta_t)$ holds at the end of Period Δ_t. Then, the output $f(g.\max, \Delta_t - T(g.\max))$ belongs to $\mathrm{OPT}(g, \Delta_t)$.

4.5.4 Time Complexity

We denote the loop of Lines 5-27 as Loop X, and the loop of Lines 12-16 as Loop Y. Let $H_{\max}$ be the maximum size of heap H during execution of OLRS_2. The time of one execution of Loop Y is $O(\log H_{\max})$. Similarly, the time of one execution of Loop X that excludes the execution time of Loop Y is $O(\log H_{\max})$. (Note that finding (the minimum) j at Lines 23-24 can be executed within $O(1)$ time given $P(\omega)$, $P_{\max}(\omega)$, $T(\omega)$, $P(v.\max)$,$P_{\max}(v.\max)$ and $T(v.\max)$.) Hence, letting A and B be the numbers of executions of Loop X and Loop Y respectively, the time complexity is $O((A+B)\log H_{\max})$. In the rest of this section, we show upper bounds of A, B and $H_{\max}$.

Lemma 45. *At most* $|\mathrm{CT}(v)|$ *nonstop* s - v *walks are expanded during execution of* OLRS$_2$*.*

Proof . A s - v walk is expanded only when $v.\max$ is updated. The update happens only at Period i such that $i \in \mathrm{CT}(v)$. Furthermore, two or more updates of $v.\max$ never happen at the same period because nonstop walks are extracted from heap H in descending order of $>_i$ at Period i. □

Corollary 4. *The number of executions of Loop Y is at most* $\sum_{v\in V}|\mathrm{CT}(v)|\cdot|N^-_{\mathrm{out}}(v)|$.

Proof . Loop Y (lines 11-14) are executed only when a walk is expanded. When a s - v walk is expanded, Loop Y is executed at most $|N^-_{\mathrm{out}}(v)|$ times. □

Corollary 5. *The number of different* s - v *walks inserted to heap* H *is* $\sum_{u\in N^-_{\mathrm{in}}(v)}|\mathrm{CT}(u)|$ *if* $v \neq s$*. It is* $1+\sum_{u\in N^-_{\mathrm{in}}(v)}|\mathrm{CT}(u)|$ *if* $v = s$*.*

Proof . A new s - v walk other than the initial walk (s) is inserted to H only when a s - u walk is expanded for $u \in N^-_{\mathrm{in}}(v)$. Therefore, at most $\sum_{u\in N^-_{\mathrm{in}}(v)}|\mathrm{CT}(u)|$ different s - v walks are inserted to H for node $v \neq s$. In addition to these walks, the initial walk (s) is inserted to H in the case of $v = s$. □

Corollary 6. $H_{\max} \le \sum_{v\in V}|\mathrm{CT}(v)|\cdot|N^-_{\mathrm{out}}(v)|$

Proof . Clearly, $H_{\max}$ is bounded by the total number of different generated walks. By Corollary 5, we have $H_{\max} \le \sum_{v\in V}\sum_{u\in N^-_{\mathrm{in}}(v)}|\mathrm{CT}(u)| = \sum_{v\in V}|\mathrm{CT}(v)|\cdot|N^-_{\mathrm{out}}(v)|$. (Note that walk (s) is extracted at Period 0 and never included in H at the time H has the maximum number of walks.) □

Lemma 46. *Any* s - v *walk is inserted to heap* H *at most* $|\mathrm{CT}(v)|$ *times.*

Proof . Let ω be a nonstop s - v walk. We will prove that the number of re-insertions of ω is at most $|\mathrm{CT}(v)|-1$. Suppose that ω is lost by $\omega_{v,i}$ at Period i, and the algorithm updates ω.time to j $(i < j \le \Delta_t)$ and re-inserts ω to H. (Note that ω is never lost by walk other than $\omega_{v,i}$ at Period i.) Then, $\omega >_k \omega_{v,i}$ holds for any $k \ge j$ (Fig. 4.3). This means that ω is never lost twice

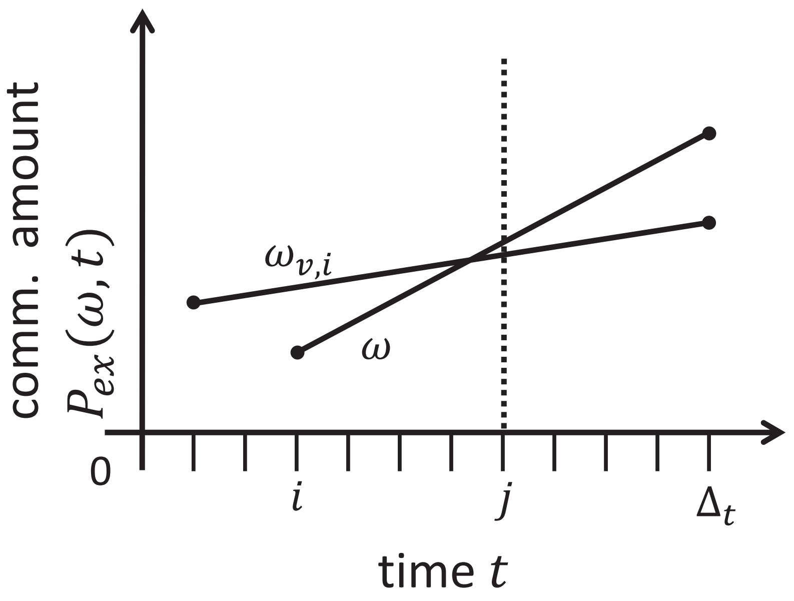


Figure 4.3: $\omega >_k \omega_{v,i}$ holds for any $k \geq j$.

or more by the same walk. Thus, the number of re-insertions of ω is bounded by the number of walks that ω is lost by. If $\omega \neq \omega_{v,\Delta_t}$, the number of re-insertions of ω is at most $|\mathrm{CT}(v)| - 1$ since ω is discarded and never re-inserted when ω is lost by ω_{v,Δ_t}. If $\omega = \omega_{v,\Delta_t}$, it is also bounded by $|\mathrm{CT}(v)| - 1$ because $|\{\omega_{v,i} \mid T_{\min}(s,v) \leq i \leq \Delta_t\} \setminus \{\omega\}| = |\mathrm{CT}(v)| - 1$. □

We obtain the following corollary by Corollary 5 and Lemma 46.

Corollary 7. *The total number of times nonstop walks are inserted to heap H is at most* $|\mathrm{CT}(s)| + \sum_{(u,v)\in E} |\mathrm{CT}(u)| \cdot |\mathrm{CT}(v)|$.

The number of executions of Loop X equals to the number of extractions of walks from H, which also equals to the number of insertions of walks to H. Therefore, we have $A \leq |\mathrm{CT}(s)| + \sum_{(u,v)\in E} |\mathrm{CT}(u)| \cdot |\mathrm{CT}(v)|$. We also see both B and $H_{\max}$ are at most $\sum_{(u,v)\in E} |\mathrm{CT}(u)|$ by Corollaries 4 and 6. Hence, the time complexity of OLRS$_2$ is at most $(A+B)\log H_{\max} = O((\sum_{(u,v)\in E} |\mathrm{CT}(u)||\mathrm{CT}(v)|) \cdot (\log \sum_{(u,v)\in E} |\mathrm{CT}(u)|))$.

Theorem 8. *Algorithm* OLRS$_2$ *solves OLRS with time complexity of* $O((\sum_{(u,v)\in E} |\mathrm{CT}(u)||\mathrm{CT}(v)|) \cdot (\log \sum_{(u,v)\in E} |\mathrm{CT}(u)|))$.

Let us express as $\mathrm{CT}_{\max} = max_{v\in V} |\mathrm{CT}(v)|$. Then, the time complexity of the algorithm can be expressed as $O(m{\mathrm{CT}_{\max}}^2 \log m\mathrm{CT}_{\max}) \subseteq O(m\Delta_t^2 \cdot \log m\Delta_t)$.

4.6 Simulation Results

In this section, we show the simulation results of execution times of two proposed algorithms. The simulation is executed for graphs modeling urban areas with WiFi access points.

4.6.1 Simulation 1

The execution time of OLRS_1 strictly depends on time limit Δ_t, while that of OLRS_2 depends on $\mathrm{CT}_{\max}$ and does not necessarily depend on Δ_t. To verify this, in Simulation 1, we evaluate the execution times of the two algorithms with changing the value of Δ_t.

We give a 30×30 square-grid $G = (V, E)$ as the input graph (Fig. 4.4). The coordinates of the starting node s and the goal node g is $(15, 1)$ and $(15, 30)$ respectively. We select five nodes randomly from V as access points in G. We set the communication amount $P(v, v)$ of self-loops at node $v \in V$ so that $P(v, v)$ is inversely proportional to the biquadrate of distance d [meter] between v and the closest access point. Specifically, letting the distance between every pair of neighboring nodes be 20 meter, we set $P(v, v)$ as follows:

$$P(v, v) = \max\left(50, 20\log_2\left(1 + \frac{1.2 \times 10^7}{(d^2 + 3^2)^2}\right)\right) \text{ [Mbps]}$$

This expression simply models the communication speed where every access point is located at three meters above the ground and the communication protocol is 802.11g. We also define the communication amount of all edges $e \in E$ other than self-loops as $P(e) = T(e) \cdot (P(u, u) + P(v, v))/2$. The travel time $T(e)$ is uniformly chosen from integers of $[1, x]$. The upper limit x is variable: we set x to $1, 2, 2^2, \ldots, 2^9$. The time limit Δ_t is set to twice of the shortest time $T_{\min}(s, g)$. Since Δ_t is almost proportional to x, we can observe how the execution time of the two algorithms depend on Δ_t by changing the value of x.

Fig. 4.5 shows the average execution time of the two algorithms. The average is evaluated from one hundred executions for each $x = 1, 2, \ldots, 2^9$. We randomly select five access points and the travel times of edges for each execution. The execution time of OLRS_1 increases linearly with respect to x. On the other hand, the execution time of OLRS_2 is almost stationary for $x \geq 4$ whereas it is slightly increasing for $0 \leq x \leq 4$. This confirms the hypothesis that the execution time of OLRS_2 does not necessarily depend on Δ_t while the execution time of OLRS_1 strictly depends on Δ_t.

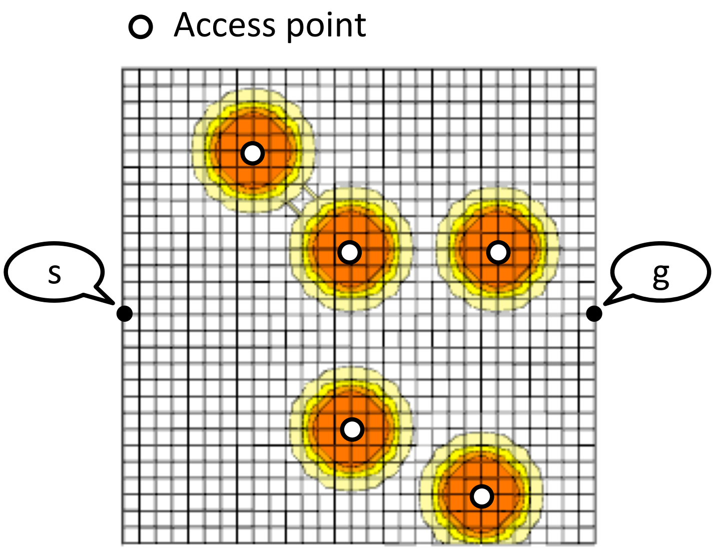


Figure 4.4: The input graph with five access points.

4.6.2 Simulation 2

In Simulation 2, we evaluate the execution times of the two proposed algorithms with changing the size of the input graph to verify the scalability of the algorithms.

We give a $30 \times y$ square-grid $G = (V, E)$ as the input graph where the coordinates of the starting node s and the goal node g is $(15, 1)$ and $(15, y)$ respectively. The number of columns y is variable: we set y to $30, 60, 90, \ldots, 300$. The time limit Δ_t is set to twice of the shortest time $T_{\min}(s, g)$. We select $y/6$ nodes randomly from V as access points. The communication amount of self-loops $P(v, v)$ and other edges $P(e)$ are defined in the same way as Simulation 1. We define the travel time of all edges e as $T(e) = 1$.

Fig. 4.5 shows the average execution time of the two algorithms. The average is evaluated from one hundred executions for each $y = 30, 60, 90, \ldots, 300$. We randomly select y/6 access points for each execution. The both algorithms solves OLRS within practical time: the execution times of OLRS$_1$ and OLRS$_2$ are approximately 7.0 seconds and 0.46 seconds respectively even for $y = 300$ (the number of nodes is 9000). As in Simulation 1, OLRS$_2$ finishes its execution earlier than OLRS$_1$ for all $y = 30, 60, 90, \ldots, 300$.

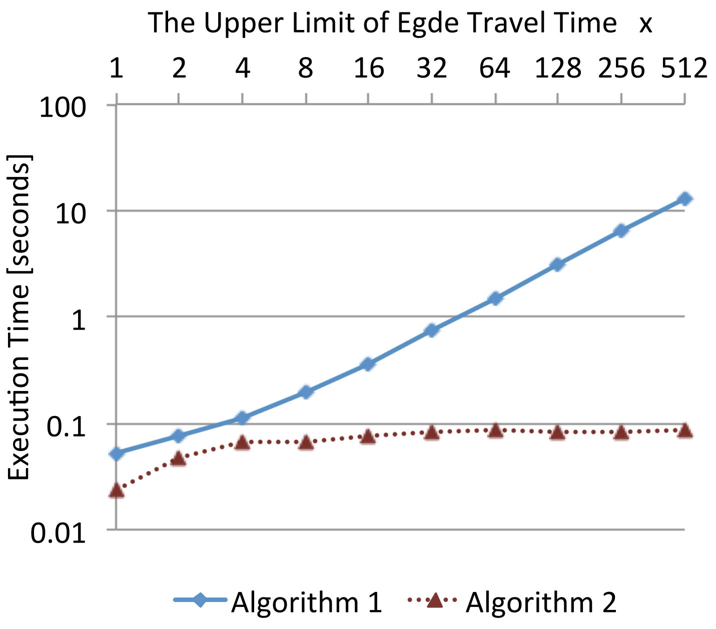


Figure 4.5: The average execution time in Simulation 1.

4.7 Discussion about Objective Function

In this chapter, we focus on the maximization of the total amount of communication while traveling. This goal fits the needs of the mobile users who require the large amount of communication (e.g. downloading a huge file such as a video file or updating a large number of applications of smart phones). Another candidate of the goal is the minimization of disconnection time while traveling. This goal reflects more natural needs of the mobile users who require stable communication (e.g. surfing on the Internet or using IP telephone service). However, in this chapter, we focus on the maximization of the total amount of communication for the following two reasons.

- The problem of minimizing the disconnection time is reduced to DCLC described in Subsection 1.1 by assigning the cost $C(e)$ of each edge $e = (v, u)$ to the disconnection time while the user moves from node v to u. ($0 \leq C(e) \leq D(e)$). Therefore, we can use many exiting DCLC algorithms to minimize disconnection time of a route. On the other hand, maximizing the total amount of communication (OLRS) has never been studied to the best

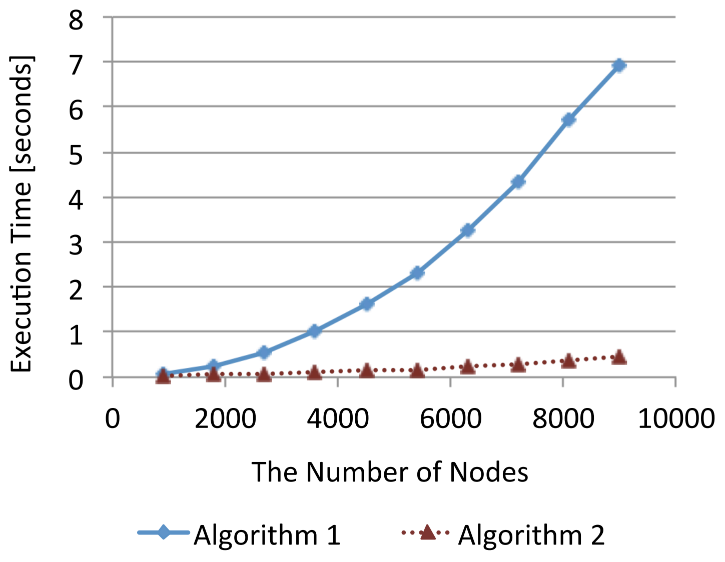


Figure 4.6: The average execution time in Simulation 2

of the author's knowledge. Hence, tackling with OLRS is of theoretical importance.

- Thanks to the progress of cellular networks, there is low possibility in urban area that the mobile users get disconnected from the Internet. Hence, in such an area, selection of a route does not make much difference on disconnection time. On the other hand, selection of a route makes a large difference on the total amount of communication since communication speed highly depends on the user's location. Hence, maximizing the total amount of communication is of practical importance.

4.8 Conclusion

In this chapter, the author introduced a new optimization problem OLRS (Optimal Longcut Route Selection). The author proved that the problem is NP-hard. The author also presented two pseudo-polynomial algorithms named OLRS_1 and OLRS_2. Their time complexities are $O(m\Delta_t)$ and $O(m\text{CT}_{\max}{}^2 \log m\text{CT}_{\max}) \subseteq O(m{\Delta_t}^2 \log m\Delta_t)$, respectively. Simulations proved that both algorithms solves OLRS within practical time for graphs modeling urban area with WiFi access points. In particular, for those graphs, OLRS_2 is always faster than OLRS_1, and its execution time in independent from the size of the time limit Δ_t.

Chapter 5

Performance Evaluation for Cloud Computing Systems by Audit Measurements

5.1 Introduction

Commercial use of cloud computing service [31] grows significantly thanks to progress of virtualization technology and broad band networks. Especially, IaaS (Infrastructure as a Service) cloud services, which provide their users with virtual machines (VMs) through networks, are widely in use all over the world. Users of IaaS cloud service can create, manage, and operate VMs in the cloud computing systems through networks. Users can create and delete VMs within a few minutes and pay usage fee based on utilization time. Therefore, users can manage the number of the used VMs according to the loads of their system, so that they can reduce the cost.

There are many providers that provide IaaS cloud services in the world and users select one or more providers among them which best meet their requirements. The most important selection criterion are the price and VMs' performance. The most natural way for users is to select the most inexpensive service satisfying desired performance or to select the service showing the best cost-performance ratio. Therefore, though most cloud services are best-effort services and do not provide performance guarantees [32], it is fairly important for users to evaluate the VMs' performance of each cloud service (e.g. average performance and stability of performance).

However, there are no ways today to estimate the VMs' performance of cloud services exactly.

Most cloud service providers publish their VMs' specification such as the number of CPU (Central Processing Unit) cores and the amount of memory, but do not publish their VMs' performance in detail such as CPU processing speed and memory transfer speed. The VMs of the same specification may show far different performance according to different cloud service providers, hence users cannot evaluate real performance of VMs from their specification. In fact, the author conducted multiple benchmark tests for low, middle, and high specification VMs of five providers (total fifteen services) described in Section 5.4.2, and showed that many VMs of the same specification show large difference (more than double) in average and standard deviation (representing unstability) of performance. Furthermore, it is known that even two VMs of the same service (i.e., the same specification and the same providers) may show different performance according to the hardware type of the host machine assigned to VMs when they are booted [33, 34].

Existing studies [33, 35, 34, 36] observed that the performances of VMs in cloud computing systems show very different characteristics from those of physical computers. First, variability of performance is significantly large. The variation coefficients[1] of the benchmark scores of VMs in cloud computing systems in terms of CPU performance, memory performance, disk read/write performance, and communication performance is tens of times or hundreds of times as large as those of physical machines [33]. In addition, performance distribution of VMs in cloud computing systems sometimes changes in medium- to long-term. In one instance, Figure 5.1 shows that the time series data of the score of disk-read benchmark for some cloud service [36]. One can see that the variability of performance is significantly large and the performance distribution changes temporally.

Therefore, it is insufficient to execute benchmarks several times in order to evaluate VMs' performance of cloud services. Owing to large variability, a small number of measurements do not suffice to evaluate VMs' performances exactly. Owing to temporal changes of performance distribution, measurement results at a specific time may give few information about the future performance distribution. Thus, it is necessary to obtain the time series data of VMs' performances continuously.

The goal of this chapter is to devise the mechanism by which users can obtain such time series data of each cloud service. There are some simple mechanisms for it but they have practical issues. Consider the mechanism where each user continuously and periodically measures VMs' performance of all the cloud services he or she is interested in. In this mechanism, each user has to pay vast fee for VMs usage to execute benchmarks, hence this is not a practical mechanism.

[1] A variation coefficient of samples is quotient of the standard deviation and the average of samples. It is typically used to compare variabilities of sets of samples with different scales.

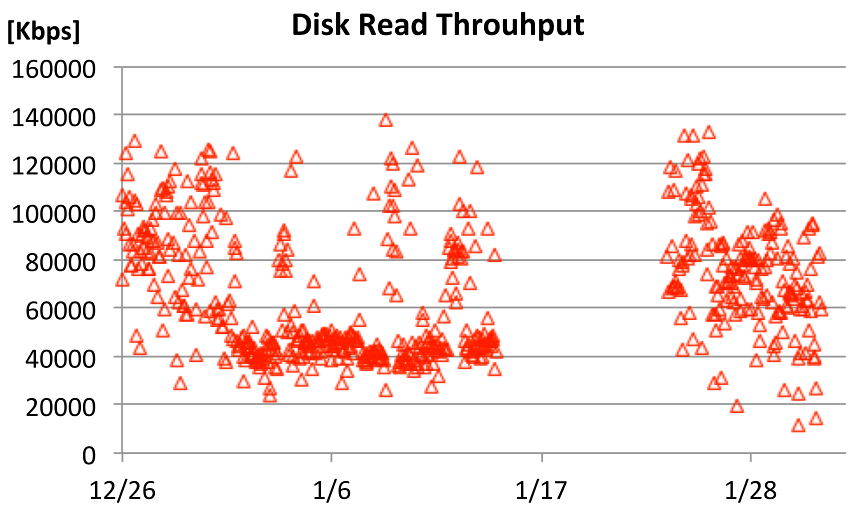


Figure 5.1: The disk-read performance of the low specification VM in Cloud A

Again, consider the mechanism where each cloud service provider measures the performance of its VMs and publishes the result of the measurement continuously. In this case, the cost of measurement is relatively low, but the trustworthiness of the published data is not guaranteed because some providers may exaggerate its performance in its published data.

Contribution of This Chapter In this chapter, the author presents the method by which each user can get trustworthy information about VMs' performance of each cloud service with low cost. This method consists of publication of VMs' performance by cloud service providers and auditing measurement by users. The measurement cost of this method is low because auditing measurements by users are intermittent. The trustworthiness of the published information is guaranteed statistically: the users conduct hypothesis test (e.g. t-test and Mood test) to verify whether there is no significant difference between the published data and the performance data obtained by auditing measurements. If some provider fabricates the published data of VMs' performance to a certain extent, the users can detect the fabrication since the fabrication makes significant difference between two data. One may concern that the users may detect significant difference between published data and auditing data even if the providers does not make any fabrication. However, the probability of such false detection is low and the users can adjust the probability as we shall see in Section 5.3.3.

5.2 Preliminaries

This section presents the model of cloud services and users. Three example methods are also introduced, and merits and demerits of the methods are analyzed.

5.2.1 Model

This section presents the model of cloud services and users. There exist multiple *cloud computing systems* and many *users* in the cloud computing market (Figure 5.2). Cloud computing systems provide users with *cloud computing services*. In the following, cloud computing systems and cloud computing services are called just clouds and services, respectively. A cloud has one or more services where the specifications of VMs and the usage fees per hour differ depending on the services. We denote all the services in the market by $s_1, s_2, \ldots, s_n$. A user is interested in one or more services, and wants to evaluate the VMs' performance of the services. We denote by $u(s_i)$ the number of users who are interested in service s_i. In the example of Figure 5.2, we have $u(s_4) = 3$. A cloud also wants to make its potential users evaluate the VMs' performance of its services correctly. However, some clouds make some sort of fabrication so that their services are overestimated by users. These clouds are called *dishonest clouds*.

Each user tries to obtain the *performance data* of all the services he or she is interested in. Performance data of a service is the set of time series data (an example is shown in Figure 5.1) and one graph of the set corresponds to the scores of a certain benchmark. Each benchmark measures the VMs' performance such as CPU performance, memory performance, disk read/write performance, or LAN bandwidth. It is assumed that the common benchmark set is used for all the services $s_1, s_2, \ldots, s_n$. Benchmarks are executed on VMs, hence benchmark execution for a service requires some cost. We denote by $C(s_i)$ the measurement cost needed to construct a year's performance data of service s_i[2].

5.2.2 Simple Methods

This subsection presents three naive methods to obtain the performance data of services, which are compared with the proposed method presented in Section 5.3. The merits and demerits of

[2] We assume the measurement cost $C(s_i)$ is the same, whether the benchmark set is executed on service s_i by a user or its provider. It is because the amount of resources (VMs) used for the measurements is the same in both the cases.

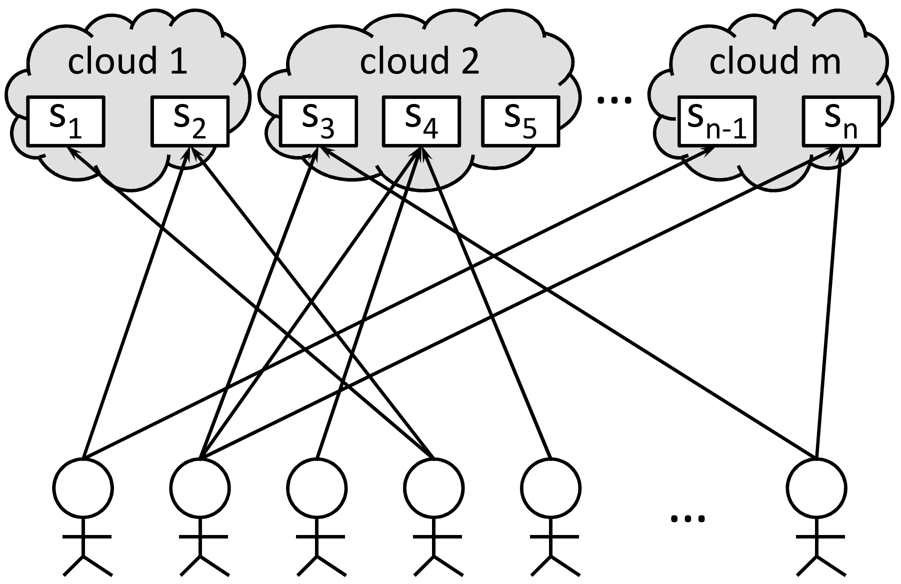


Figure 5.2: The model of cloud services and users

the three methods are also discussed.

The Cloud-publish Method

In the *cloud-publish method*, each cloud constructs the performance data of all its services and publish them for all the users. Each cloud periodically initiates VMs of its services, executes the benchmark set on the VMs, and updates the published performance data of the services.

In this method, the performance of each service is measured by only one organization (i.e., the cloud that provides the service). Therefore, the total cost required to measure service s_i in the whole market is just $C(s_i)$ per a year. However, the trustworthiness of published performance data is not guaranteed since the cloud that creates and publishes the performance data may be dishonest. Then, the dishonest cloud may fabricate the performance data and publish them so that its services are overestimated by users.

The Evaluation Organization Method

In the *evaluation organization method*, a third party called *evaluation organization* constructs the performance data of all the services in the market and publishes them. To construct the performance data of each service s_i, the evaluation organization initiates a VM of service s_i as a user of s_i, and executes the benchmark set on the VM.

The trustworthiness of the published data obtained by this method is higher than that of the cloud-publish method, but still insufficient. First, it is possible for a dishonest cloud to assign rich resources (CPUs and memory) specially only for VMs used by evaluation organizations. Then, the performance data published by the evaluation organization are unreliable and the services of the dishonest cloud are overestimated by users. Even if the evaluation organizations hide their role to clouds, it may be possible for the clouds to identify the account used by the organization. For example, consider the case a dishonest cloud provides service s_i that assigns several model of CPUs to its VMs. The dishonest cloud can memorize the models of assigned CPUs for the VMs of each account at every hour. Then, the cloud may identify the account of the evaluation organization by comparing the assigned CPUs for VMs of each account at every hour and the benchmark scores at every hour in the performance data published by the evaluation organization. Secondly, it is hard to elect a completely neutral evaluation organization. Users cannot deny possibility of the collusive relationship between a cloud and an evaluation organization. Thus, the additional method to deny the possibility is required. If there are many evaluation organizations in the market, then the risk of the collusive relationship is reduced since the published performance data of unrightful evaluation organization show different distribution from those of honest evaluation organizations. The total cost required to obtain the performance data of service s_i is $aC(s_i)$ per a year in the whole market where a is the number of evaluation organizations.

The User-Evaluation Method

In the user-evaluation method, each user directly creates the performance data of all the service he or she is interested in. The user periodically initiates VMs of the services, executes the benchmark set on the VMs, and updates the performance data of the services.

In this method, the trustworthiness of performance data is sufficiently high. Since users directly measure the VMs' performance of services, the users can create trustworthy performance data even for services of dishonest clouds. However, the cost of this method is higher than the above two methods. Since all the users who are interested in service s_i measure the performance of s_i independently, the total cost required to obtain the performance of s_i is $u(s_i) \cdot C(s_i)$ per a year. For example, when $u(s_i) = 1000$, the cost is thousand times as high as that of the cloud-publish method, which is something unrealistic.

5.3 The Proposed Method

The three simple methods presented in Section 5.2.2 have the demerit of cost or trustworthiness of performance data. The trustworthiness of performance data is not guaranteed in the cloud-publish method and the evaluation organization method, while the total cost required to obtain performance data is high in the user-evaluation method. In this section, the author proposes a novel method by which users can obtain trustworthy performance data of services with significantly lower cost than the user-evaluation method. In the proposed method, clouds publish the performance data of their services as in the cloud-publish method. On the other hand, users infrequently conduct snap inspections to verify the correctness of the published performance data statistically.

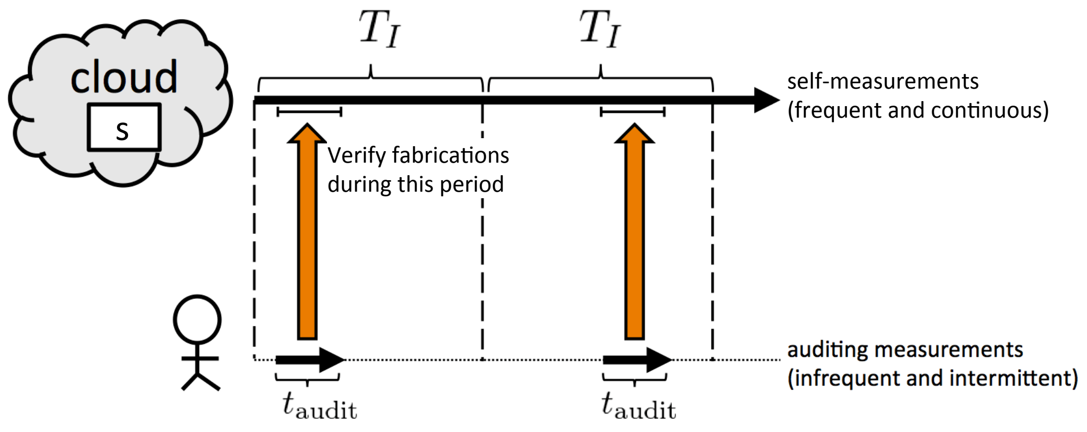
First, the proposed method is presented in Section 5.3.1. Next, the fabrications assumed by this method are shown in Section5.3.2. Finally, the hypothesis tests used in this method are explained in Section 5.3.3.

5.3.1 Method Description

The overview of the proposed method is illustrated in Figure 5.3. In the proposed method, each cloud publishes the performance data of its services as in the cloud-publish method. In the following, these published performance data are called just *published data.* The trustworthiness of this published data is not guaranteed as is pointed out in Section 5.2.2 since dishonest cloud may fabricate the published data. To complement this demerit, the proposed method introduces the mechanism to detect such fabrication.

In this method, the users who are interested in service s measure the VMs' performance of s at irregular intervals. We refer this measurement as *audit measurement* and the performance data obtained by this measurement as *audited data.* The audit measurement is conducted intermittently. Specifically, the users conduct audit measurements only in consecutive t_{audit} days at every T_I days, and does not measure the performance in other $T_I - t_{\text{audit}}$ days. For example, the users conduct audit measurements in consecutive seven days at every 365 days (i.e. every year). Consecutive t_{audit} days are randomly selected among T_I days. We call these t_{audit} days *auditing period.*

The users verify the correctness of s's published data utilizing the audited data of s. Specifically, the users conducts hypothesis tests to detect significant difference between the published data in the auditing period and audited data in terms of average, standard deviation, and distribution profile of the score of any benchmark. If significant difference is detected, the users



cloud
s
T_I
T_I
self-measurements (frequent and continuous)
Verify fabrications during this period
auditing measurements (infrequent and intermittent)
t_{audit}
t_{audit}

Figure 5.3: The proposed method

recognize that the cloud that provides service s is dishonest.

5.3.2 Assumed Fabrication

The proposed method detects fabrications by testing for significant difference between some statistics of published data and audited data. What kind of fabrication we assume determines what statistics we should conduct tests for. This section gives the assumed fabrication in this chapter.

Generally, users are interested in benchmark scores of VMs in terms of average value and variability. Therefore, it is a natural assumption that dishonest clouds make fabrication to increase the average value of benchmark scores or reduce the variability of benchmark scores. Thus, we assume these two fabrications in the following. These fabrications are modeled in this section as *the uniformly-increasing fabrication* and *the variability-reducing fabrication.*

The Uniformly-increasing Fabrication This fabrication increases all the benchmark scores of some period $[t_1, t_2]$ uniformly by some constant value. Specifically, for time series data $D_i(t_1, t_2) = (d_{i,t_1}, d_{i,t_1+1}, \ldots, d_{i,t_2})$ of benchmark i, we define the fabricated data $D'_i(t_1, t_2) = (d'_{i,t_1}, d'_{i,t_1+1}, \ldots, d'_{i,t_2})$ as follows:

$$d'_{i,t} = d_{i,t} + \Delta_1 \overline{D_i(t_1, t_2)},$$

where $\overline{D_i(t_1, t_2)} = \frac{1}{t_2 - t_1 + 1} \sum_{j \in [t_1, t_2]} d_{i,j}$ and $\Delta_1 > 0$. Owing to this fabrication, the average of D_i becomes $1 + \Delta_1$ times as large as the original average. We call Δ_1 the *fabrication level* of the uniformly-increasing fabrication. Meanwhile, the standard deviation of $D_i$which represents the variability are not changed by this fabrication.

The Variability-reducing Fabrication This fabrication reduces the variability (or standard deviation) of the benchmarked scores of some period $[t_1, t_2]$. Specifically, for time series data $D_i(t_1, t_2) = (d_{i,t_1}, d_{i,t_1+1}, \ldots, d_{i,t_2})$ of benchmark i, we define the fabricated data $D'_i(t_1, t_2) = (d'_{i,t_1}, d'_{i,t_1+1}, \ldots, d'_{i,t_2})$ as follows:

$$d''_{i,t} = (1 - \Delta_2)(d_{i,t} - \overline{D_i(t_1, t_2)}) + \overline{D_i(t_1, t_2)}.$$

where $0 < \Delta_2 \leq 1$. Owing to this fabrication, the standard deviation of D_i becomes $1 - \Delta_2$ times as large as the original variability. We call Δ_2 the *fabrication level* of the variability-reducing fabrication. Meanwhile, the average of D_i are not changed by this fabrication.

5.3.3 Hypothesis Tests

The proposed method uses hypothesis tests to verify whether there is significant difference of the average or variability between the published data and the audited data. Generally, hypothesis test outputs a real number called p-value given the two data as an input. This p-value represents the probability that the difference of the statistic (e.g. average and variance) between the two data occurs if the difference occurs by chance (i.e. not on purpose). When p-value is smaller than the given threshold α, then we have the conclusion that the two data has significant difference about the statistic.

Threshold α gives the probability of false detection. In other words, even if no fabrication are made on the published data, the hypothesis test detects significant difference between the published data and the audited data with probability α. On the other hand, we set α to a higher value, we can detect smaller level of fabrication. The author shows later (in Section 5.4.2) that we can detect small level of fabrication made on published data even when we set α to sufficiently small value (0.001).

There are many hypothesis tests established in the statistics field and these tests can be applied to detect fabrication of published data. For example, Welch's t-test, MannWhitney U test, and Wilcoxon signed-rank test can be used to detect significant difference on average. Concerning detection of difference in variability (or standard variance), F-test, Levene test, Ansari-Bradley test, Mood test, and so on can be used. We use MannWhitney U test for the uniformly-increasing fabrication and Mood test for the variability-reducing fabrication. (The reason of adopting these tests will be described in Section 5.4.3.) In what follows, we call MannWhitney U test just U-test.

Table 5.1: Comparison between the proposed method and the three simple methods ($f = t_{\mathrm{audit}}/T_I$)

	cloud–publish	evaluation organization	user–evaluation	the porposed method
trustworthiness	×	△	◎ not subject to fabrications	○ able to detect fabrications
cost	◎ $C(s_i)$	○ $a \cdot C(s_i)$	× $u(s_i) \cdot C(s_i)$	○ $C(s_i) + f \cdot u(s_i) \cdot C(s_i)$

5.4 Evaluation of the Proposed Method

Table 5.1 shows the summary of comparison between the proposed method and the three simple methods in Section 5.2.2. The proposed method has higher trustworthiness of performance data than the cloud-publish method and the evaluation organization method, and takes significantly lower cost than the user-evaluation method.

This section shows the reason for these comparison results. In Section 5.4.1, the author shows that the cost of the proposed method is significantly smaller than the user-evaluation method. In Section 5.4.2, the author shows the performance data obtained in the proposed method are trustworthy. Specifically, it is verified that users can detect fabrication of published data even when the level of the fabrication is sufficiently small. Since this verification strongly depends on the distribution of VMs' performance, the author measures the actual VMs' performance of commercial cloud services to verify trustworthiness of the proposed method adequately.

5.4.1 Measurement Costs

The annual cost of the proposed method required to obtain performance data of service s_i is $C(s_i) + f \cdot u(s_i)C(s_i)$ where $f = t_{\mathrm{audit}}/T_I$. The first and the second terms correspond to the costs of the self-measurements by the cloud of service s_i and the auditing measurements by the users who are interested in s_i, respectively. The cost of the proposed method is significantly smaller than that of the user-evaluation method when f is sufficiently small, that is, the frequency of audit measurement is small. For example, when $u(s_i) = 500$ and $f = 7/365$ holds (auditing measurements are conducted seven days per year), the annual cost of the proposed method is $C(s_i) + f \cdot u(s_i)C(s_i) = 10.6C(s_i)$, which is only 2.1 percent of that of the user-evaluation method ($500C(s_i)$).

Table 5.2: The detail specification of VMs used for the measurement experiment

	Cloud A			Cloud B			Cloud C			Cloud D			Cloud E		
spec. level	low	middle	high	low	middle	high	low	middle	high	low	middle	high	low	middle	high
CPU	2vCPU	4vCPU	8vCPU	2vCPU	4vCPU	4vCPU	2vCPU	4vCPU	8vCPU	2vCPU	4vCPU	8vCPU	2vCPU	4vCPU	8vCPU
clock frequency	1vCPU: 1.6Ghz			1vCPU: 2Ghz			1vCPU: 1.6Ghz			1vCPU: 1.0−1.2Ghz			1vCPU: 1.6Ghz		
memory	4GB	8GB	16GB	4GB	8GB	16GB	4GB	8GB	16GB	3.75GB	7.5GB	15GB	3.5GB	7GB	14GB

5.4.2 Trustworthiness of Performance Data

The trustworthiness of the performance data in the proposed method is guaranteed by the auditing mechanism to detect fabrication made on published data. However, it is hard to detect too small fabrication since the detection is based on statistical analysis (hypothesis test). On the other hand, we can say that the performance data is trustworthy if users can detect sufficiently small level of fabrication in practical applications.

The goal of this section is to verify the trustworthiness of the proposed method by observing how small fabrication can be detected by the proposed method. This verification strongly depends on the distribution of VMs' performance. Therefore, the author measures the actual VMs' performance of five commercial cloud services, and evaluates trustworthiness of the proposed method analyzing the measured data.

Measurement Experiment

The author measures the performance of services of five commercial cloud, Cloud A, Cloud B, Cloud C, Cloud D, and Cloud E. This experiment measures the performance of three services of low, middle, and high specification for each cloud. The detail specification of the services' VM is shown in Table 5.2. The specification of the same level (low, middle and high) of services are almost the same among all the clouds except that clock frequency is relatively low in Cloud D and the high level VMs of Cloud B have only four virtual CPU cores. (the high level VMs of the other clouds have eight virtual CPU cores.) This experiments use two VMs for each specification (low, middle and high) of each cloud. One is used to create the published data and the other is used to create the audited data. Hence, we use six VMs to be benchmarked for each cloud.

This experiment uses the same benchmark set as Shad et al. [33], which is as follows.

- CPU performance：ubench(CPU) [37]
- Memory performance：ubench(memory)
- Disk read throughput：bonnie++(seq. input) [38]

- Disk write throughput : bonnie++(seq. output)
- LAN bandwidth : iperf [39]

Benchmark iperf needs a destination host. Therefore, an additional VM is set for each cloud, and the bandwidth between the additional VM and each of the six benchmarked VMs is measured by iperf.

The measurements period is thirty six days from December 26th 2012 to January 30th 2013. In the period, the benchmark set is executed on each VM every hour (total 864 executions for each VM). Specifically, the experiment repeats the following processes every hour.

1. Create six VMs on each cloud,
2. Execute all the benchmark on each VM and upload the result scores to our storage,
3. Delete the six VMs.

The repetition of creation and deletion of VMs are necessary to obtain exact performance data of the services. (Physical resources are assigned when VMs are created.)

Verification Methodology

In this verification, we regard the two performance data of each service, which are obtained from the measurement experiment in Section 5.4.2, as published data and audited data. The author makes the uniformly-increasing fabrication and the variability-reducing fabrication on the published data, and conduct hypothesis tests (U-test and Mood test) to detect significant difference between these two data. Then, we observe the fabrication level required to decrease p-value less than threshold α. In the following, we refer to this fabrication level as *detectable fabrication level*. We observes the detectable fabrication level in case of $\alpha = 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001$. As mentioned above, U-test and Mood test is used to detect the uniformly-increasing fabrication and the variability-reducing fabrication, respectively.

The detectable fabrication level depends on the number of benchmarked scores in the audited data, and this number depends on the audited period. In this verification, we adopt seven days as audited period. Specifically, we divide the experiment period of thirty six days into the five disjoint periods of seven days each, and regard the data in each period as a audited data (Table 5.3). Note that the number of benchmark scores of each audited data is $7 \times 24 = 168$.

Table 5.3: The audited periods

Period 1	Period 2	Period 3	Period 4	Perio 5
12/27 – 1/2	1/3 – 1/9	1/10 – 1/16	1/17 – 1/23	1/24 – 1/30

Verification Results

This section shows the detectable fabrication level of each service (VMs of low, middle and high specification of each cloud) and each type of performance (CPU performance, memory performance, disk read/write throughput, and LAN bandwidth). The detectable fabrication level shown in this section is the average of those of the five audited periods shown in Table 5.3.

Before showing the results on this verification, the author explains some troubles that occurred in the measurement experiment. In Cloud A, the performance data between 16:00 January 14th and 14:00 January 23rd are not obtained because VMs could not be created in this period owing to failure of the cloud computing system. Hence, the detectable fabrication level in Cloud A is shown as the average of three audited periods Period 1, Period 2, and Period 5. Furthermore, in Cloud C and Cloud E, the creation and deletion of VMs failed with high probability so that the author cannot get sufficient number of benchmark scores for these two clouds. Hence, we exclude Cloud C and Cloud E from the objects of this verification.

Figure 5.4 represents the detectable fabrication levels for the service of low specification in Cloud A when published data is modified by the uniformly-increasing fabrication. For all types of performance, the smaller threshold brings the bigger detectable fabrication level. However, these increase are gradual: the detectable fabrication levels increase linearly according to exponential decrease of threshold α. For example, the detectable fabrication level of LAN bandwidth increases one percent every time threshold α is reduced to one-tenth. This tendency is commonly observed for all services (all specification of all clouds) and all types of performance. Furthermore, this tendency is also common for the case of the variability-reducing fabrication.

In the following, we focus on the detectable fabrication levels in case of $\alpha = 0.001$. However, it does not loose generality because the detectable fabrication levels in case of other thresholds have the same characteristics.

Figure 5.5 represents the detectable fabrication levels in case of the uniformly-increasing fabrication. Even though we set a strict threshold $\alpha = 0.001$, fabrications of less than ten percent level can be detected in most cases. However, the detectable fabrication levels of disk write throughput

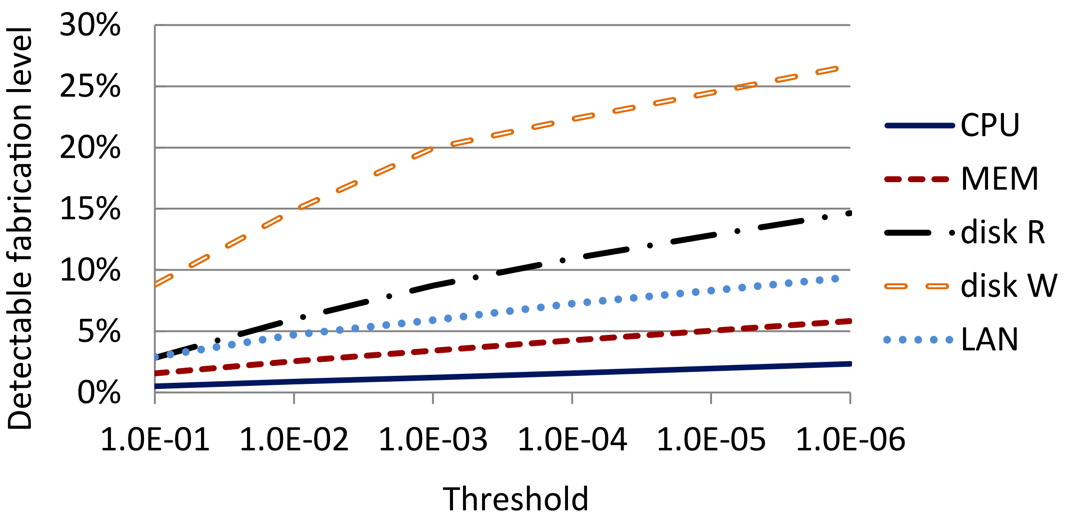


Figure 5.4: Detectable fabrication levels for the uniformly increasing fabrication (low specification, Cloud A)

and memory performance is relatively high, and exceeds 10 percent in some cases. This is because the variabilities of these benchmark scores are high. Generally, the powers of hypothesis tests to detect significant difference of averages decreases when variabilities of samples are large. In fact, in every case (every specification of every clouds, every benchmarks), large variation co-efficients (represent the degree of variability) bring large detectable fabrication levels (See Figure 5.6).

Extreme smallness of detectable fabrication levels for CPU performance and memory performance in Cloud D is worthy of special mention. This is because the performance distribution has multiple bands[3] in these cases, and the variability of each band is extremely small (See the case of memory performance of the middle specification shown in Figure 5.7). U-test focuses on the order of each benchmark score in the published data and the audited data among all the benchmark scores of both data. It calculates p-value by comparing the sum of the orders of all benchmark scores in the published data and those of the audited data. In the above cases, since the variability of each band is small, even slight fabrication such as 0.16 percent level (in case of Cloud D, low specification, and CPU performance) changes the orders of the benchmark scores heavily, which drastically decreases p-value of U-test. Hence detectable fabrication levels in these

[3] As many studies [33, 34, 36] pointed out, performance distribution of commercial cloud computing services may have multiple bands partly because multiple types of hardware can be assigned to VMs.

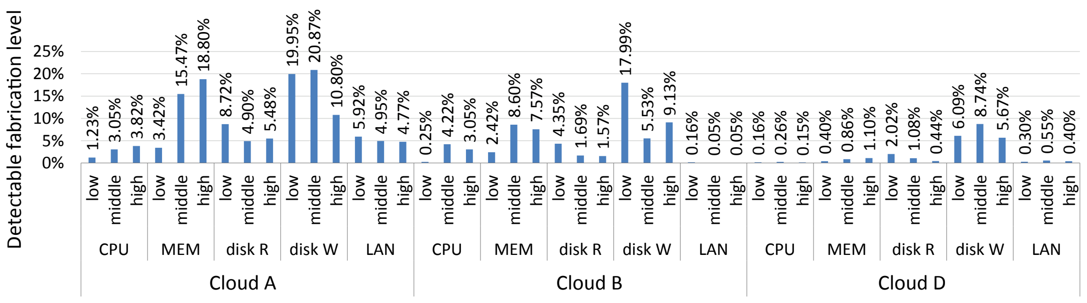


Figure 5.5: Detectable fabrication levels for the uniformly-increasing fabrication ($\alpha = 0.001$)

cases are extremely small.

Figure 5.8 represents the detectable fabrication levels in case of the variability-reducing fabrication. It appears that detectable fabrication levels are higher than those of the cases of the uniformly-increasing fabrication. This is just because detecting significant difference of variabilities is difficult. In fact, the detectable fabrication level is 23.90 percent on average even when the published data and the audited data follow Gaussian distribution perfectly.[4] On the other hand, detectable fabrication levels are small in some cases including CPU performance in Cloud D. This is because the performance distribution of these cases have multiple bands. As in U-test, Mood test focuses on the orders of each benchmark score to calculate p-value. Since slight fabrication changes the orders of benchmark scores heavily in case that multiple bands exist in performance distribution, the detectable fabrication levels for such cases are small.

5.4.3 Discussion

The verification of Section 5.4.2 observes how small fabrication the proposed method detects. Even when the published data is modified by the uniformly-increasing fabrication (or by the variability-reducing fabrication), fabrications of less than ten percent level (or less than thirty percent level, respectively) can be detected in most cases. In case of the uniformly-increasing fabrication, much smaller fabrication can be detected for performance data of CPU performance and LAN bandwidth where variability of benchmark scores is small. In case of both the uniformly-increasing fabrication and the variability-reducing fabrication, much smaller fabrication can be detected if the performance distribution has multiple bands.

[4] The author randomly creates thousand pairs of data that follows Gaussian distribution, and calculates detectable fabrication levels for the pairs in the same way as the verification of this section.

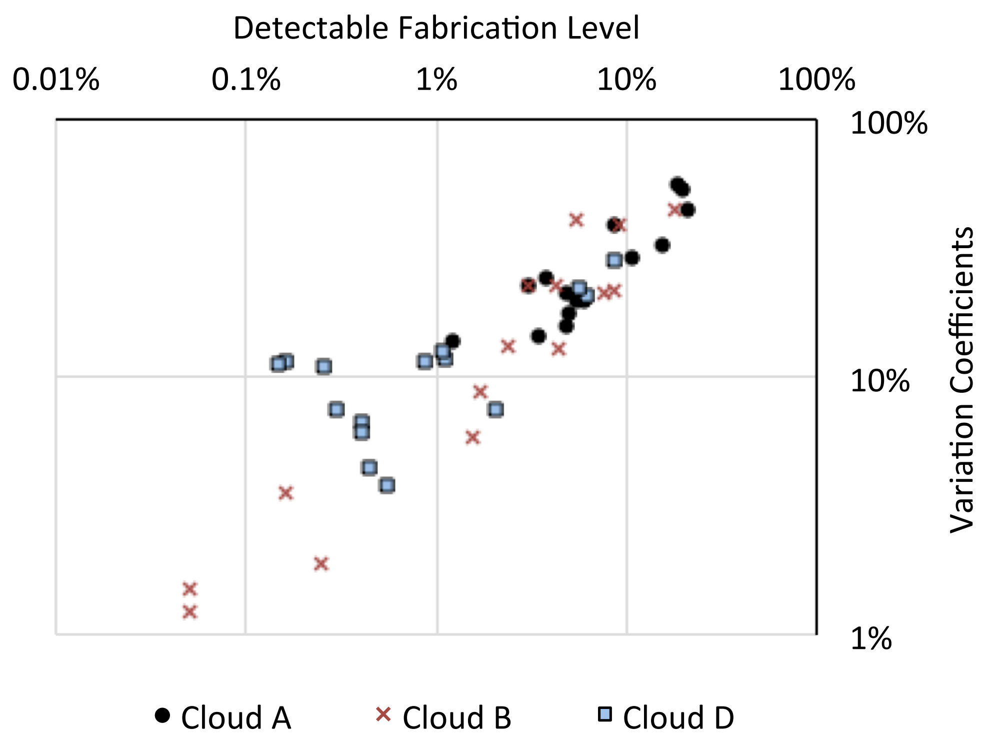


Figure 5.6: The correlation chart of detectable fabrication level ($\alpha = 0.001$) and variation coefficient

In what follows, we briefly look at using other tests to detect fabrication made for the performance of cloud-computing services. Welch's t-test and Wilcoxon signed-rank test can be used to detect the uniformly-increasing fabrication. However, Welch's t-test is unsuitable for the proposed method since it premises Gaussian distribution: the power of the test is weak when the performance distribution deviates from Gaussian distribution, especially when the performance distribution has multiple bands. Wilcoxon signed-rank test can be used when sample values of the two tested data have correspondence relation. We can consider that two benchmark scores measured at the same time have correspondence relation for the published data and the audited data. However, in case that the performance distribution has multiple bands, the correspondence does not fit and the power of the test becomes low. F-test, Levene test, and Ansari-Bradley test

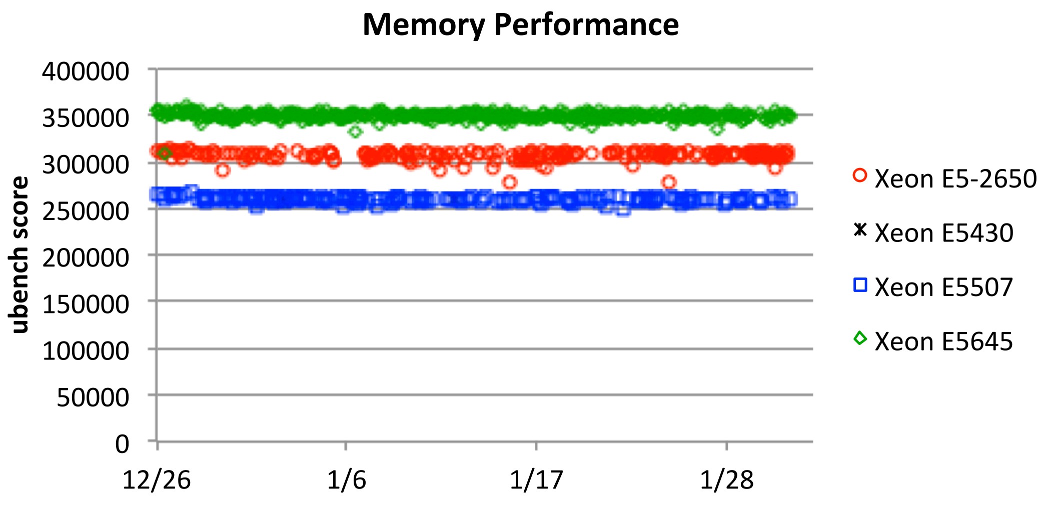


Figure 5.7: Memory performance of the middle specification VM in Cloud D

can be used to detect the variability-reducing fabrication. However, F-test is unsuitable since it cannot calculate p-value when the performance distribution deviates from Gaussian distribution. Although Levene test has robustness for deviation from Gaussian distribution to a certain extent, we confirm that it does not calculate exact p-value when the performance distribution has multiple bands by a computer simulation. Ansari-Bradley test does not have a problem to detect the variability-reducing fabrication. Deviation from Gaussian distribution or multiple bands in performance distribution does not cause any problem for this test. However, in the verification in Section 5.4.2, the author confirmed that this test can attain a lightly higher detectable fabrication level than Mood test.

5.5 Practical Examples of VMs' Performance Evaluation

There are some practical examples of VMs' performance evaluation of cloud services. There are two sites, CloudHarmony [40] and CloudClimae [41], which measure multiple cloud services and publish the performance data obtained by the measurement. (We refer this kind of evaluation method as *the evaluation organization method* in Section 5.2.2.) CloudHarmony executes many benchmarks on the VM of more than tens of cloud services and memorize the score of the

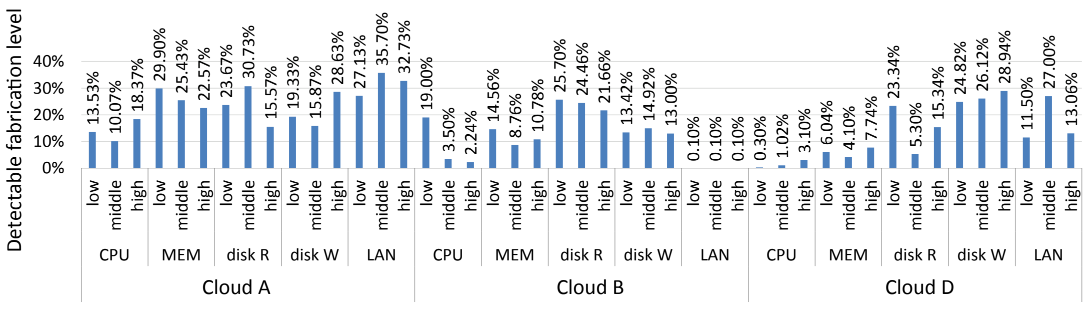


Figure 5.8: Detectable fabrication level for the variability-reducing fabrications ($\alpha = 0.001$)

benchmarks in its system. Users of this site request benchmark scores by selecting cloud service providers, VMs' specifications (the specifications of services), and benchmarks (e.g. 7zip and Unixbench), then the site offers the corresponding score memorized in the system. The shortage point of this site is infrequency of benchmarking. The same benchmark for the same service is executed by this system only a few times a year. Hence, the users cannot obtain the VMs' performance of each service since the variability of VMs' performance of cloud services is large and the performance distributions vary temporally. CloudClimte publishes denser performance information of each cloud which is measured every five minutes. However, the number of cloud services and the number of benchmarks is small, hence the benefits of its users are limited.

5.6 Conclusion

In this chapter, the author presented the method by which users economically obtain trustworthy performance data of cloud computing services. In this method, cloud service providers publish the performance data of their services, and users infrequently conduct snap inspections to verify the preciseness of the published performance data statistically. If users conduct a snap inspection for seven days every year, the cost of the proposed method is one-fiftieth of that of the user evaluation method. On the other hand, when the published performance data is fabricated, users can detect the fabrication with high probability. From verification based on the performance distribution of commercial cloud computing services, we confirmed that a snap inspection of 168 executions of the benchmark set detects sufficiently small fabrication. To conclude, the proposed method enables users to obtain trustworthy performance data of cloud computing services economically.

Chapter 6

Conclusion

This thesis introduced the author's studies to provide stable services in unstable distributed systems by realizing fault-tolerance, communication performance, and computational performance.

Chapter 2 presented a novel concept of loose-stabilization, which brings distributed systems high fault tolerance. Loose-stabilization is a variant of self-stabilization, which relaxes the closure requirement of the original self-stabilization without impairing its fault-tolerance at least in practical perspective. Since the closure requirement is relaxed, loose-stabilization can be applied to more models and more problems of distributed systems. For example, the author presents a loosely-stabilizing algorithm that solves leader election problem on complete graphs in the population protocol model (the PP model). This algorithm exemplifies the benefit of loose-stabilization since its is impossible to devise self-stabilizing algorithm for this problem unless the exact number of nodes is given.

Chapter 3 developed the study of Chapter 2 to present loosely-stabilizing algorithms that solve the leader election problem on *arbitrary graphs* in the PP model. Unlike complete graphs where all nodes can directly communicate with each other, each node has to detect the existence or absence of a distant leader node in leader election on arbitrary graphs. The author presented two loosely-stabilizing algorithms that create a new leader when no leader exists and keeps the new leader for exponentially long time by utilizing the unique identifiers of nodes or random numbers. These algorithms also exemplify the benefit of loose-stabilization since it is impossible to devise a self-stabilizing algorithm for this problem even if node-identifiers or random numbers is available.

In Chapter 4, the author focused on communication quality of mobile users in urban areas. The communication quality (wireless communication bandwidth) of mobile networks differs depending on the location the user exists. The author tackled with a new problem to find the route that maximizes the total amount of communication by which users can move from the given starting point to the given destination point within the given limit time. First, the author proved that this problem is NP-hard. Then, the author presented two pseudo-polynomial algorithms that solve this problem. One of the two is better in terms of the worst-case execution time, and the other one is better in terms of the average execution time. We observed from the simulation experiments that both the two algorithms compute the optimal solution within practical execution time even for a sufficiently large input (i.e. the topology of urban area) consisting of thousands of nodes.

In Chapter 5, the author developed the method by which users of cloud computing services obtain trustworthy data of VMs' performance of those services. It is known that the VMs' performances of commercial cloud computing services often deviate from their specifications. Then, we need frequent and continuous performance measurements to recognize the VMs' performance of those services. If these measurements are conducted by the users (the user-evaluation method) then the costs of users are impractically high, and if these measurements are conducted and the obtained performance data are published by cloud computing providers (the cloud-publish method) then we face no guarantee for the trustworthiness of the published data. The author proposed the method where the providers continuously measure and publish the VMs' performance of their services while the users infrequently conduct snap inspections to guarantee the trustworthiness of the published data. The costs of the proposed method is significantly lower than the user-evaluation method. Furthermore, the users can obtain sufficiently trustworthy performance data by the proposed method. In fact, the author observed from the performance distributions of commercial cloud computing services that the snap inspections of the users detect fabrications of less than several percent level in most cases.

From now on, the author is going to propel the study of loose-stabilization introduced in Chapters 2 an 3. Although the proposed algorithms in Chapter 3 work on arbitrary graphs, they need node identifiers or random numbers. One of the author's future work is to remove these requirements and devise a loosely-stabilizing leader election algorithm that works on arbitrary graphs without node-identifiers and random numbers. While Chapters 2 and 3 focused on the leader election in the PP model, the author is also going to apply the concept of loose-stabilization to other models and other problems.

Acknowledgements

The author has been fortunate to receive assistance from many people. He would especially like to express his gratitude to his supervisor Professor Toshimitsu Masuzawa for his guidance and encouragement. The author has also received precious advice from Professors of the Graduate School of Information Science and Technology, Osaka University. Among them, the author would like to extend his gratitude to Professor Katsuro Inoue, Professor Kenichi Hagihara and Associate Professor Hirotsugu Kakugawa for their valuable comments on this thesis.

The author would like to thank Dr. Tutomu Murase, Dr. Gen Motoyoshi, Professor Junichi Murayama, Mr. Kunio Hato, Assistant Professor Yukiko Yamauchi, Assistant Professor Junya Nakamura and Mr. Daisuke Baba for their useful comments on his work. The author also thanks to Mrs. Fusae Nishioka and Mrs. Hisako Suzuki for their kind support. He is also grateful to the staffs and students of Algorithm Engineering Laboratory, the Graduate School of Information Science and Technology, Osaka University. In particular, he thanks to Assistant Professor Fukuhito Ooshita for their time and kindness. He also thanks to Mr. Atsushi Takada, Mr. Ryo Nakai, Mr. Shinji Kawai and Mr. Yonghwan Kim for their friendly supports and discussions in the laboratory. The author expresses his gratitude to all members of his workplace of Network Security Project in NTT Secure Platform Laboratories for their kind understanding for his academic research.

Finally, the author wishes to thank his wife Yuki Sudo and his son Yutaro Sudo for their support and kindness during his life at the university.

Bibliography

[1] E. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Communications of the ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[2] A. Israeli and M. Jalfon, "Token management schemes and random walks yield self-stabilizing mutual exclusion," in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pp. 119–131, ACM New York, NY, USA, 1990.

[3] J. Lin, T. Huang, C. Yang, and N. Mou, "Quasi-self-stabilization of a distributed system assuming read/write atomicity," *Computers and Mathematics with Applications*, vol. 57, no. 2, pp. 184–194, 2009.

[4] M. Gouda, "The Theory of Weak Stabilization," in *Proceedings of the 5th International Workshop on Self-Stabilizing Systems*, pp. 114–123, Springer, 2001.

[5] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta, "Computation in networks of passively mobile finite-state sensors," *Distributed Computing*, vol. 18, no. 4, pp. 235–253, 2006.

[6] D. Angluin, J. Aspnes, and D. Eisenstat, "fast computation by population protocols with a leader," in *DISC*, pp. 61–75, 2006.

[7] D. Angluin, J. Aspnes, M. J. Fischer, and H. Jiang, "Self-stabilizing population protocols," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 3, no. 4, p. 13, 2008.

[8] S. Dolev, *Self-stabilization.* MIT Press, 2000.

[9] S. Devismes, S. Tixeuil, and M. Yamashita, "Weak vs. self vs. probabilistic stabilization," in *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS 2008)*, pp. 681–688, 2008.

[10] D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta, "Stably computable properties of network graphs," in *DCOSS*, pp. 63–74, 2005.

[11] S. Cai, T. Izumi, and K. Wada, "How to prove impossibility under global fairness: On space complexity of self-stabilizing leader election on a population protocol model," *Theory of Computing Systems*, vol. 50, no. 3, pp. 433–445, 2012.

[12] M. J. Fischer and H. Jiang, "Self-stabilizing leader election in networks of finite-state anonymous agents," in *OPODIS*, pp. 395–409, 2006.

[13] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[14] O. Michail, I. Chatzigiannakis, and P. G. Spirakis, "Mediated population protocols," *Theoretical Computer Science*, vol. 412, no. 22, pp. 2434–2450, 2011.

[15] J. Beauquier, J. Burman, L. Rosaz, and B. Rozoy, "Non-deterministic population protocols," in *OPODIS*, pp. 61–75, Springer, 2012.

[16] R. Mizoguchi, H. Ono, S. Kijima, and M. Yamashita, "On space complexity of self-stabilizing leader election in mediated population protocol," *Distributed Computing*, vol. 25, no. 6, pp. 451–460, 2012.

[17] X. Xu, Y. Yamauchi, S. Kijima, and M. Yamashita, "Space complexity of self-stabilizing leader election in population protocol based on k-interaction," in *SSS*, pp. 86–97, Springer, 2013.

[18] J. Beauquier, P. Blanchard, and J. Burman, "Self-stabilizing leader election in population protocols over arbitrary communication graphs," in *OPODIS*, pp. 38–52, Springer, 2013.

[19] D. Canepa and M. G. Potop-Butucaru, "Stabilizing leader election in population protocols," 2007. http://hal.inria.fr/inria-00166632.

[20] R. Guerraoui and E. Ruppert, "Even small birds are unique: Population protocols with identifiers," *Rapport de Recherche CSE-2007-04, Department of Computer Science and Engineering, York University, York, ON, Canada*, 2007.

[21] Y. Sudo, G. Motoyoshi, T. Murase, and T. Masuzawa, "Optimal longcut route selection for wireless mobile users," in *IEICE Technical Report(RCS2009-269)*, p. 71, 2010.

[22] G. Motoyoshi, Y. Sudo, T. Murase, and T. Masuzawa, “Advantages of optimal longcut route for wireless mobile users,” in *Communications (ICC), 2011 IEEE International Conference on*, pp. 1–6, IEEE, 2011.

[23] M. R. Garey and D. S. Johnson, *”Computers and Intractability. A Guide to the Theory of NP-completeness. A Series of Books in the Mathematical Sciences”*. WH Freeman and Company, San Francisco, Calif, 1979.

[24] R. Widyono, “The design and evaluation of routing algorithms for real-time channels,” tech. rep., International Computer Science Institute, University of California at Berkley, 1994.

[25] D. H. Lorenz, A. Orda, D. Raz, and Y. Shavitt, “Efficient qos partition and routing of unicast and multicast,” *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1336–1347, 2006.

[26] A. Jüttner, B. Szviatovszki, I. Mécs, and Z. Rajkó, “Lagrange relaxation based method for the qos routing problem,” in *INFOCOM*, pp. 859–868, 2001.

[27] G. Feng, C. Douligeris, K. Makki, and N. Pissinou, “Performance evaluation of delay-constrained least-cost qos routing algorithms based on linear and nonlinear lagrange relaxation,” in *IEEE International Conference on Communications (ICC 2002)*, vol. 4, pp. 2273–2278, IEEE, 2002.

[28] H. de Neve and P. van Mieghem, “A multiple quality of service routing algorithm for pnni,” in *ATM Workshop Proceedings, 1998 IEEE*, pp. 324–328, IEEE, 2002.

[29] L. Guo and I. Matta, “Search space reduction in qos routing,” *Computer Networks*, vol. 41, no. 1, pp. 73–88, 2003.

[30] G.S.Lueker, “Two np-complete problems in nonnegative integer programming,” Tech. Rep. 178, Computer Science Laboratory, Princeton University, 1975.

[31] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[32] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, “Early observations on the performance of windows azure,” in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 367–376, ACM, 2010.

[33] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: observing, analyzing, and reducing variance," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 460–471, 2010.

[34] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, "What are you paying for? performance benchmarking for infrastructure-as-a-service offerings," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 484–491, IEEE, 2011.

[35] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pp. 104–113, IEEE, 2011.

[36] Y. Sudo and K. Hato, "Peformance analysis of public clouds," in *IEICE Technical Report IN2013-60*, vol. 113, pp. 13–18, Sep. 2013.

[37] "Ubench 0.32." `http://www.phystech.com/download/ubench.html`.

[38] "Bonnie++ now at 1.03e." `http://www.coker.com.au/bonnie++`.

[39] "iperf." `https://code.google.com/p/iperf/`.

[40] "Cloudharmony." `http://cloudharmony.com/benchmarks`.

[41] "Cloudclimate." `http://www.cloudclimate.com`.