A Study on Algorithms for Efficient Multi-Point Communication in Autonomous Distributed Networks

（自律分散ネットワークにおける効率的多地点間通信のためのアルゴリズムの研究）

by
Seiichiro Tani
谷 誠一郎

A thesis submitted in partial fulfillment of the
requirements for the degree of

Doctor of Information Science and Technology
in
Computer Science

in the

Graduate School
of the
University of Tokyo

January 27, 2006

Abstract

Data communication has increasingly become essential in people's daily life since the IP network was invented, and the most major type of recent data communication is that among multiple sites, called *multi-point communication*, such as WWW, live broadcast and TV conferences, rather than simple one-to-one communication. One reason for this increasing popularity would be that the IP network makes it possible to easily develop network applications that are attractive to even non-expert people, especially those based on multi-point communication. Another reason would be that the IP network can work and be deployed in an autonomous distributed manner.

Therefore, it is of great importance to develop the technologies that improve the efficiency of multi-point communication but do not spoil distributed autonomy of the network.

The thesis studies efficient multi-point communication in autonomous distributed networks from various points of view: from practice to theory and from the present to the future.

The present multi-point communication can be roughly classified into two types, depending on application behaviors: one is called *file-transfer type* multi-point communication, which non-simultaneously transfers stored data to many receivers (e.g., WWW), and the other is called *streaming type* multi-point communication, which simultaneously transfers non-stored/stored data to many receivers (e.g., live broadcast).

To increase the efficiency of the file-transfer type multi-point communication, *network caching* is the most popular strategy. The thesis discusses the file caching problem, a core problem concerning network caching, and gives competitive on-line algorithms for the problem. A standard way of efficient streaming type multi-point communication is *multicast*. The thesis presents some issues concerning multicast, and proposes new multicast protocols to overcome the issues.

To assess the future multi-point communication, the thesis discusses quantum multi-point communication. In particular, the leader election problem is considered in an anonymous quantum network, and exact algorithms to solve the problem are given.

Acknowledgments

Throughout the thesis, I would like to express my sincere appreciation to Professor Hiroshi Imai of the University of Tokyo for invaluable suggestions and continuous encouragement.

The research concerning file-transfer and streaming type multi-point communications was done at NTT Network Innovation Laboratories with many collaborators. Among them, I would like to express my special gratitude to Dr. Toshiaki Miyazaki (at present, who is a professor of the University of Aizu) for a lot of discussions, encouragement and support. I would also like to thank Dr. Takeru Inoue for his significant contribution to the improvement of Flexcast implementation, and Drs. Shin'ichi Minato (at present, of Hokkaido University) and Noriyuki Takahashi for valuable comments and technical advice.

The research concerning quantum multi-point communication was done at NTT Communication Science Laboratories and JST ERATO Quantum Computation and Information (QCI) Project. I was able to significantly improve my preliminary results in collaboration with Drs. Hirotada Kobayashi and Keiji Matsumoto of National Institute of Informatics. I would like to express my appreciation for their great help. I would also like to thank Dr. Yasuhito Kawano and the other members of the quantum computation group of NTT for a lot of useful comments. Furthermore, I like to thank all members of JST ERATO QCI Project for discussions and support.

I feel indebted to Dr. Shigeru Yamashita of Nara Institute of Science and Technology for his advice and encouragement. Further, I would like to deeply appreciate the kindly support and encouragement for writing this thesis by NTT, especially Drs. Shigeru Katagiri, Takehiro Moriya and Kiyoshi Shirayanagi.

Finally, I would like to thank my wife and children for great encouragement and tolerance, from the bottom of my heart.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Data communication has increasingly become essential in people's daily life since the IP network [163, 164, 161, 162, 145] was invented, and the most major type of recent data communication is that among multiple sites, called *multi-point communication*, such as WWW, live broadcast and TV conferences, rather than simple one-to-one communication. One reason for this increasing popularity would be that the IP network makes it possible to easily develop network applications that are attractive to even non-expert people, especially those based on multi-point communication as previously described. Another reason would be that the IP network supports distributed autonomy in terms of both behavior and deployment. More precisely, every element of a network can behave themselves depending almost only on local information, i.e., without complete knowledge of the entire network; this enables even huge-sized networks to work well. Further, whenever we connect a new network appliance (such as a router) to an existing network, we have only to update local configurations of the network; this makes it possible to expand the network at any site almost independently of the other sites. In general, however, it is difficult to strictly control communication traffic in such an autonomous distributed network. Therefore, it is of great importance to develop the technologies that improve the efficiency of multi-point communication without spoiling distributed autonomy of the network.

In tackling multi-point communication as well as other research areas, theory plays important roles of providing a solid foundation for practical technologies and assessing the potential of a new technology. On the other hand, in autonomous distributed networks, multi-point communication is supposed to be performed among different kinds of systems that were built based on individual philosophy; this often requires quite practical viewpoints beyond technical advantages. Thus, if we consider the efficiency of multi-point communication in autonomous distributed networks, we must understand it from a variety of viewpoints including theoretical and practical ones.

The purposes of the thesis include theoretical and practical contributions to improve the efficiency of major types of multi-point communication on the present network, and to assess a new type of multi-point communication that would be a crucial component in the future network.

In the present network, end hosts are often distinguished from the network they are connected to. When end hosts want to do some task in collaboration with one another, communication efficiency could be improved by optimize the application-layer protocol they use, which specifies the content of the data they exchange and the ordering of the data; such optimization strongly depends on the

application in question. Even after optimizing application-layer protocols, some redundant traffic may often go through networks. In what follows, we consider general approaches to remove such redundancy by clever network management.

With regard to multi-point communication on the present network, we can classify it into three types in terms of the numbers of senders and receivers: one-to-many, many-to-one, and many-to-many. For instance, one-to-many communication includes WWW and live broadcast; many-to-one communication, sensor-networks, i.e., systems that gather the information obtained by distributed sensors via networks; many-to-many communication, TV conferences and file exchange. Among these three types, many-to-many communication often consists of one-to-many or many-to-one communications; improving the latter two communications is essential. Many-to-one communication, however, intrinsically involves little redundant traffic; it is hard to greatly improve the efficiency. Thus, we focus on the efficiency of one-to-many communication.

We further classify one-to-many communication into two types, depending on application behaviors: one is called *file-transfer type* communication, which non-simultaneously transfers stored data to many receivers (e.g., WWW), and the other is called *streaming type* communication, which simultaneously transfers non-stored/stored data to many receivers (e.g., live broadcast).

To increase the efficiency of the file-transfer type communication, *network caching* is the most popular strategy. Network caching saves the redundant traffic that goes through particular sites at different time; some copies of previously transferred files are stored at certain sites on the network and, when a receiver requests one of the files, a copy of the file is transferred to him from one of the sites having the copy instead of the owner of the file. Since the site having the copy is usually closer to the receiver, network caching can reduce network traffic; it also decreases the latency required to transfer files and the load of the owner. Although there are some other strategies such as deploying multiple mirror sites, which is effective especially for particular popular file owners, network caching is effective even in general situations; it is widely used in practical environments [1].

The nodes having the network caching function, called *cache servers*, are placed on the sites where congestion is likely to occur. In a standard way of use, every cache server performs caching independently of other cache servers' behavior, and installing a new cache server has no influence on others' configurations; cache servers work in a fully autonomous and distributed way (even so, network caching is still considerably effective). In this case, the performance of each individual cache server has a great impact on the efficiency of entire communication. One of the most influential factors on cache server performance is how to decide which file should be stored, since all files cannot be stored due to the limited resources (memory, storage, CPU, etc.) of cache servers. A good decision may save transferring a file that is requested again from a remote site. This issue

is formulated as the *file caching problem* [63, 200, 112, 19]. The difficulty of the problem is that we need to decide which files should be stored without knowing future requests although files are stored in order to service future requests (in general, such problems are called *on-line problems*, and algorithms for on-line problems are called *on-line algorithms*). Three cost models were proposed depending on the correlation between file size and file cost, i.e., the cost incurred to obtain a file from other sites: the *fault model* [112], the *bit model* [112] and the *general model* [19]. For these models, a major research direction has been to find algorithms that incur low cost to retrieve mishit files, i.e., achieve high hit rate, while developing fast algorithms is another important direction.

The file caching problem is a generalized form of the *paging problem* associated with microprocessor caches, which was first studied comprehensively in [38] and has been extensively studied for more than three decades [39, 93, 176, 91, 142, 200, 49, 114, 125]. The difference between the problems is that the former deals with objects (i.e., files) of various size while the latter handles objects (i.e., pages) of uniform size; the file caching problem is more complicated than the paging problem. A standard way of analyzing on-line problems is *competitive analysis* (see [47]); the measure used in the analysis is called a *competitive ratio*, which quantifies how badly an on-line algorithm behaves itself in the worst case, relative to an optimal off-line algorithm, i.e., an optimal algorithm that knows all requests in advance. If the competitive ratio of an on-line algorithm is bounded by a certain constant that is independent of the number of requests to files, the algorithm is said to be *competitive*; the goal of competitive analysis approach is to develop competitive on-line algorithms that run in reasonable time.

The streaming type multi-point communication has its unique characteristics: real-time streaming data such as voice and moving pictures has to be simultaneously delivered to many receivers. Thus, we cannot store the data at some sites on the network to improve communication efficiency. If, however, an owner of streaming data directly delivers it to every receiver via one-to-one communication, it obviously consumes too much bandwidth and burdens the owner with heavy load. A standard way of efficient streaming type multi-point communication is *multicast*: it simultaneously delivers streaming data via a rooted tree constructed on the network, in which the root and leaves are the owner and receivers, respectively, such that, when every internal node of the tree receives data from his parent, he duplicates the data to send it to every child. Thus, the owner has only to deliver the data to a small number of children; the traffic of the entire network is considerably reduced.

A huge number of multicast protocols have been proposed for more than two decades [83, 157]. Early researches targeted to multicast on a local network. Multicast routing protocols over an internetwork was first introduced in [78]. Since then, many multicast routing protocols over an internetwork have been proposed

and standardized [79, 80, 88, 31, 30, 150, 189, 89, 62, 187, 172]. These protocols work at the network layer and are called *IP multicast*. Although IP multicast protocols have been supported by a lot of application softwares and routers, there are still some difficulties in using IP multicast over an internetwork [84]. To use IP-multicast, we need special IP addresses, called *IP-multicast addresses*, to distinguish multicast groups. However, it is difficult to preserve the uniqueness of IP-multicast addresses in a distributed manner, since IP-multicast addresses are basically independent of the structure of networks, and group members may change frequently. This makes it difficult to deploy multicast service on large networks. Furthermore, all routers need to be able to route IP-multicast packets, i.e., packets whose destination addresses are IP multicast addresses; this prevents IP multicast from being autonomously deployed at each individual site on existing one-to-one communication (unicast) networks, since two sites cannot communicate via IP multicast if there are some local networks between them that do not support IP multicast. As a result, IP-multicast is used only in closed or experimental networks such as MBone [87].

To overcome these difficulties, there are several attempts. Source-Specific Multicast (SSM) [100] restricts IP-multicast communication to one-to-many so that it can simplify group address allocation and data distribution. Nevertheless, SSM does not allow multicast service to be autonomously deployed at each individual local site, since all routers still need to route IP-multicast packets. A completely different approach was proposed, called *IP-unicast-based multicast* [177, 72, 192, 50]: all packets used in that new approach are IP-unicast packets, i.e., the packets that are used in ordinary one-to-one (unicast) communication. This new technology is thought as a promising tool for the progressive deployment of multicast service in an autonomous and distributed manner, since it is possible for legacy routers, i.e., the routers that can only have the function of ordinary unicast routing, to be located on the paths between receivers and streaming data owners. The group address administration issue can also be solved by restricting multicast communication to one-to-many as SSM. However, no protocol has been established yet as a standard one of IP-unicast-based multicast.

We have considered so far two major strategies to improve the efficiency of (one-to-many) multi-point communication on the present network. The same lines are obviously effective even in the future network, as long as the mechanism of the future network is essentially the same as that of the present network. To provide for the future multi-point communication, however, it is also quite important to assess new types of communication, especially those based on completely different principles, from the viewpoint of efficient multi-point communication. Quantum communication [153] is one of such new-type communications, and is regarded as a possible crucial component of the future network: quantum communication is based on the quantum physics instead of classical physics.

Recently, it has been revealed that quantum communication has quite unique characteristics even in the case of one-to-one communication. The most significant result would be quantum key distribution protocols [44, 86, 43] for sharing a random bit string among two parties as a secret key in a cryptographic context. The protocols were proved by many researchers (e.g. [141, 135, 175, 179, 180, 95]) to be secure against eavesdroppers even with unlimited computational power under various circumstances, whereas no existing classical protocols can achieve such security. This implies that quantum communication can significantly improve the security of secret-key cryptosystems. Meanwhile, the polynomial time quantum factoring algorithm in [174] has been threatening the security of the RSA cryptosystem, which is widely used in secure multi-point communication such as electronic commerce, since the computational hardness of integer factoring provides a theoretical foundation for the security of the RSA cryptosystem. Recently, it has been shown that quantum communication can make a positive contribution to secure multi-point communication [35, 73, 24, 25, 148, 42].

For distributed computing in which no cheating parties are assumed, quantum one-to-one communication also has a great advantage in communication efficiency over the classical one in some cases. It was proved that certain functions can be computed in the quantum setting with much lower communication complexity (i.e., the number of communicated quantum/classical bits) than in the classical setting. When input is restricted under a certain condition, there is an exponential gap between the quantum and classical communication complexities [58, 166]. With no restriction over input, the known biggest gap is quadratic when constant error probability is allowed [58, 10, 167].

As for the efficiency of multi-point quantum communication, there are not many studies [60, 57, 199, 120, 34]. For a three-party case, there is an exponential gap between the quantum and classical communication complexities, when two parties are allowed to send only one message depending on their inputs to the third party who is supposed to compute a predefined function of the inputs [57, 199, 34]. In [60], a logarithmic factor separation (in the number of parties) was shown in the completely different communication model where parties have entangled qubits before starting computation instead of quantum communication links.

1.2 Overview of the Thesis

Section 2 discusses network caching: on-line algorithms for the file caching problem. Among the three cost models, the fault model would be useful when it were expensive to establish each connection to a file owner. From the viewpoint of traffic reduction, however, the bit and the general models seem to be reasonable; we focuses on these two cost models. Subsection 2.1 formulates the file caching problem, defines the bit and the general models and introduces the competitive

analysis in both deterministic and randomized settings. Subsection 2.1 also discusses the computational hardness of the file caching problem when all requests are known in advance; we give a proof of $\mathcal{NP}$-hardness of the file caching problem in the bit model (and thus in the general model), although this fact was already referred to in [19] without proof.

For the bit model, the most famous algorithm of the paging problem, Least-Recently-Used (LRU) [38], is also the most popular strategy of the file caching problem in a practical sense, and has the optimal competitive ratio; however, LRU does not fully utilize the information on the size of each file, while its practical performance is not so bad and it runs very fast, i.e., in $O(1)$ steps, per file insertion to and file eviction from storage, when implemented on software by using doubly-linked lists. In fact, in the bit model, a lot of heuristics considering file size have been proposed and shown to be more effective than LRU in typical cases by using trace-driven simulations [160], while a randomized competitive algorithm in [112] is expected to work better than LRU in terms of competitiveness (but it seems to be too theoretical). Unfortunately, these heuristics have no performance guarantee. We thus relate heuristics to the theoretical framework of competitiveness in Subsection 2.2: we give a simple but sufficient condition for deterministic online algorithms to be competitive, and develop a general framework based on the condition to easily construct a competitive algorithm from a heuristics. More precisely, the constructed algorithm adaptively switches from the heuristics to LRU and vice versa according to the characteristics of input requests. By applying the framework to a known heuristics that always evicts the largest file, called SIZE, we construct a competitive algorithm "Competitive_SIZE." The results of event-driven simulations show that Competitive_SIZE is much better than LRU when small files are requested with high probability; in other cases, Competitive_SIZE is comparable to or only slightly worse than LRU. Furthermore, we show that Competitive_SIZE is better than either LRU and SIZE for a real web proxy log.

For the general model, an optimal deterministic competitive algorithm, called Greedy-Dual-Size (GDS), was proposed in [63, 200]. This algorithm also achieves high hit rate in experiments as shown in [63] and is adopted as an optional file caching strategy in Squid [1], the most major WWW proxy server software. However, GDS has the worst time complexity of $O(k)$ per file eviction, for storage size k. We thus give a fast randomized algorithm that is expected to be as competitive as GDS in Subsection 2.3; the randomized algorithm runs in $O(\log k)$ time in the worst case and is expected to run in $O(2^{\log^* k})$ time per file eviction or insertion. Notice that $2^{\log^* k}$ is a much more slowly increasing function than k. To confirm its practicality, we conducted trace driven simulations by using real web proxy logs. Experimental results show that our algorithm works only slightly worse than GDS when file cost is proportional to file size, and performs as well as GDS when the cost is the latency for file retrieval, i.e., almost independent of file

size.

Subsection 2.4 summarizes Section 2, and Subsection 2.5 gives a brief survey of the studies on the paging problem (including the file caching problem as a generalization of the paging problem).

Section 3 discusses multicast and proposes a new IP-unicast-based multicast protocol, called *Flexcast*. Subsection 3.1 describes the network model we assume. Subsection 3.2 presents the basic protocol of Flexcast, and illustrates its behavior by using some examples; it is shown that Flexcast dynamically constructs a multicast tree by sharing maximal common links among paths from receivers to a sender (i.e., an owner of streaming data), and maintains it against the change of unicast routes, the moving of receivers and senders, and the frequent joining and leaving of receivers, with minimum rearrangement of the tree. These dynamic construction and maintenance are realized by a hierarchical keep-alive mechanism in a highly autonomous and distributed fashion.

Subsection 3.3 describes a security issue of IP-unicast-based multicast protocols including the basic protocol of Flexcast, and gives some modifications to the basic protocol to make Flexcast more tolerant against the issue. More precisely, most IP-unicast-based multicast protocols allow the protocol packets from end hosts (receivers or senders) to change the states of network nodes, i.e., to consume computational resources of the nodes. This fact may cause a serious issue if malicious users try to collapse the multicast system by using the packets.

Furthermore, we provide an optional function for Flexcast from a practical point of view. As is often the case with new protocols, it is not easy to have people install application softwares that process the new protocols. In the case of multicast, there are many commercial/free application softwares that handle IP multicast; they have already been installed on user's computers in many cases. Hence it would be useful to provide a system that enables the users of IP multicast application softwares to use Flexcast. Subsection 3.4 presents Flexcast-IP-multicast bidirectional translators, called *Flexcast gateways*.

Subsection 3.6 describes experiments on stream delivery using Flexcast among three widely dispersed locations in Japan and U.S.. The experimental results include the latency required to receive the first streaming data packet, and the influence of both communication delay jitter and packet-loss rate on the Flexcast operation, and show the robustness and stability of the Flexcast protocol in the absence of unusual congestion on any communication links.

Subsection 3.7 summarizes Section 3 and gives some remarks on the design of IP-unicast-based multicast protocols. Subsection 3.8 briefly describes a survey of multicast protocols.

Section 4 theoretically considers multi-point communication on the future network: quantum multi-point communication. The multi-point communication described in the previous two sections is concerned with a network supporting dis-

tributed autonomy. To observe the ability of such multi-point communication in the quantum setting, we consider the *leader election problem* in an *anonymous* network. More precisely, the goal of this problem is to elect a unique leader from among distributed parties under the assumption that every party is completely identical to one another. Due to this assumption, every party has to make the most of quantum mechanics in a highly autonomous and distributed fashion to solve the problem.

It is well-known that no classical algorithm can solve the leader election problem in an anonymous network in bounded time without error (i.e., exactly) even if the number of parties is given to every party. We give two quantum distributed algorithms that, when the parties are connected by quantum communication links and the number of parties is given to every party, can exactly solve the problem for any network topology in polynomial rounds and polynomial communication/time complexity with respect to the number of parties. Furthermore, they can be easily modified so that they work well even when each party knows only the upper bound of the number of parties in advance. This implies that the exact number of parties can be computed without error in bounded time when its upper bound is given, whereas no classical zero-error algorithm exists in such cases for any topology that has a cycle as its subgraph [118].

Subsection 4.1 defines the network model and the leader election problem in an anonymous network.

Subsection 4.2 gives the first algorithm; it runs with lower time and communication complexity than the second algorithm. However, the quantum communication is the dominant factor in the total communication complexity. Since sending a qubit would cost more than sending a classical bit, it would be reasonable to reduce the quantum communication complexity.

The second algorithm in Subsection 4.3 incurs higher time complexity, but demands the quantum communication of fewer qubits, than the first one. Furthermore, the second algorithm requires fewer rounds, including only one round of quantum communication, than the first one, where a "round" represents each turn of simultaneous message exchange. This one-round quantum communication allows us to easily modify the second algorithm so that it can work even if the underlying graph is directed (more rigorously, strongly-connected); the modified algorithm is also shown in the subsection. We summarize the complexities of the first and second algorithms in Table 1.

To reduce the amount of quantum communication, our second algorithm makes use of a classical technique, called *view*, which was introduced in [195, 196]. However, a naïve application of view incurs exponential classical time/communication complexity. To keep the complexity moderate, a new technique of *folded view* is introduced in Subsection 4.4, with which the algorithm still runs in time/communication polynomial with respect to the number of par-

	Time	Quantum C. C.	Total C. C.	Rounds
Alg. I	$O(n^4)$	$O(n^4)$	$O(n^4)$	$O(n^2)$
Alg. II	$O(n^6(\log n)^2)$	$O(n^2)$	$O(n^6(\log n)^2)$	$O(n \log n)$

Table 1: Complexity of the two algorithms for the number n of parties. "C.C." denotes "communication complexity." "Total C.C." includes both of the quantum and classical communication complexity.

ties. Subsection 4.5 summarizes Section 4. Subsection 4.6 gives a brief survey of quantum distributed computing.

Finally, Section 5 concludes the thesis.

2 File-Transfer-Type Multi-Point Communication – Network File Caching –

This section discusses an efficient way of the file-transfer type multi-point communication: solving the file caching problem.

2.1 Preliminaries

2.1.1 The model

A reasonable cost assumption for the file caching problem is that a cache server services a request at no cost if it already has a copy of the requested file (called a *hit*); otherwise (on a miss) it incurs a *fault* and needs to pay some cost to retrieve a copy of the file. Thus, the total cost for a request sequence is defined as the sum of the retrieval costs of all fault files. The simplest cost model, called the *fault model* [112], is that all files have uniform cost even if they are significantly different in size. This model would be useful when it were expensive to establish each connection to a file owner. In order to reduce traffic in networks, however, the next two models seem to be reasonable: the *bit model* [112] identifies file size with retrieval cost while the *general model* [19] assumes that retrieval cost is arbitrary and independent of file size. The bit model is intuitively understandable, since, when a file is being transferred, the traffic on a communication link would be almost linear in the size of the file. Meanwhile, there is often the case where retrieved files are quite different in the distance they travel. In particular, when file size does not greatly differ over all files, retrieval cost would largely depend on the distance to travel, which could be assessed by the latency required to get the file. In this case, the files downloaded from remote sites would be expensive. The general model can handle this case. For these models, a major research direction has been to find algorithms that incurs low retrieval cost, i.e., achieves high hit rate, while developing fast algorithms is another important direction.

For the general model, a formal definition of the file caching problem is as follows.

Definition 1 *The input is a request sequence to files:* $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\sigma|}\}$, *where* σ_i *is the file identified by the* i*th request. Each file* σ_i *has its size and cost, denoted by* $\mathsf{size}(\sigma_i)$ *and* $\mathsf{cost}(\sigma_i)$*, respectively, where, without loss of generality, the size and cost of every file are assumed to be positive integers. Let* C_i *be the set of files stored in the cache immediately before the* i*th request* σ_i *is serviced.*

Suppose that the storage size of cache is some fixed positive integer k*. The goal is to minimize* $\sum_{\sigma_i \in \sigma} f(i)\mathsf{cost}(\sigma_i)$*, where* $f(i) = 1$ *if a fault occurs for* σ_i*, i.e.,* $\sigma_i \notin C_i$*, and* $f(i) = 0$ *otherwise, under the condition that*

- $C_{i+1} \subseteq C_i \cup \{\sigma_i\}$ *and* $\sigma_i \in C_{i+1}$,
- $\sum_{j \in C_i} \mathsf{size}(j) \leq k$ *for any* i.

In particular, $\mathsf{size}(i) := \mathsf{cost}(i)$ for any file i in the bit model.

The off-line version of the file caching problem is $\mathcal{NP}$-hard even for the bit model as referred to in [19, 112] without proof; we will give a proof of this fact. It follows that there is a significant difference in complexity between the off-line versions of the file caching problem and the paging problem, since it is well-known that the paging problem has a linear-time optimal off-line algorithm [38]. For the fault model, it is unknown whether the problem is in $\mathcal{P}$.

Remark 1 *In our definition, each requested file must be inserted into the cache when it is retrieved, following the usual setting of the paging problem. It is also a possible setting to allow obtained files to be discarded without storing them [111].*

Remark 2 *The actual cost of retrieving a file may vary with time in many applications. This issue is not considered here, nor is the cache consistency issue, i.e., the file is updated at the server while stored in the cache.*

2.1.2 $\mathcal{NP}$-hardness of the file caching problem in the off-line setting

An instance of the file caching problem for request sequence σ can be mapped into a graph representation. In the following, we give the mapping, and prove the $\mathcal{NP}$-hardness of the resulting problem.

Let $|\sigma|$ be the number of the requests in σ. A directed graph $G_\sigma = (V_\sigma, E_\sigma)$ has a set $V_\sigma = (u_1, u_2, \ldots, u_{|\sigma|})$ of vertices and a set $E_\sigma \subseteq V_\sigma \times V_\sigma$ of edges such that for each i and j $(i < j)$ $(u_i, u_j) \in E_\sigma$ if $j = \min\{j | \sigma_i = \sigma_j, i < j\}$. Each edge $e_{i,j} = (u_i, u_j) \in E_\sigma$ has size, $\mathsf{size}(e_{i,j})$, and cost, $\mathsf{cost}(e_{i,j})$, corresponding to the size and cost of σ_i $(= \sigma_j)$. The edge $e_{i,j}$ is interpreted as storing σ_i from time i to time j $(> i)$ and hitting the request at time j. Then, the file caching problem is translated to selecting a subset E' of E_σ that maximizes the total cost saving $\sum_{e \in E'} \mathsf{cost}(e)$ under the condition that at each time the sum of the size of the stored files is at most k and that each σ_i is stored at time i. This is because we can assume without loss of generality that optimal solutions evict σ_i at time $(i+1)$ if σ_i will be evicted before the next request to the same file as σ_i. Now we formalize the problem into which the paging problem is mapped.

Problem 2 *(Graph representation of the file caching problem) For* $G_\sigma = (V_\sigma, E_\sigma)$ *and a constant* k*, find* $E' \subseteq E_\sigma$ *that maximizes* $\sum_{e \in E'} \mathsf{cost}(e)$ *under the condition that for each* i $(1 \leq i \leq |\sigma|)$, $\sum_{e=(j_1,j_2) \in E', j_1 < i < j_2,} \mathsf{size}(e) \leq k - \mathsf{size}(e_i)$*, where* e_i *is one of the existing edge(s) connecting to* u_i*.*

We note that G_σ is an extension of the graph representation for the paging problem introduced in [186].

Theorem 1 *Problem 2 is $\mathcal{NP}$-hard even if the cost of each file equals its size.*

Proof. Let the number of different files, N, be $|V_\sigma|/2$ ($= |\sigma|/2$). Consider request sequence σ such that for all $i(\leq N)$, $\sigma_i = \sigma_{2N+1-i}$, $\sigma_i \neq \sigma_j$ $(j \neq 2N+1-i)$ and $\mathsf{size}(\sigma_i) \leq \mathsf{size}(\sigma_{i+1})$ $(i = 1, \ldots, N-1)$. Construct $G_\sigma = (V_\sigma, E_\sigma)$, where $V_\sigma = \{u_1, \ldots, u_{2N}\}$ and $E_\sigma = \{(u_i, u_{2N+1-i}) \mid (i = 1, 2, \ldots, N)\}$. For each edge $e_i = (u_i, u_{2N+1-i})$, we define a variable $\lambda_i \in \{0, 1\}$, which is 1 if e_i is chosen as a member of E' and 0 otherwise. The problem can then be expressed as 0-1 integer programming; maximize $\sum_{i=1}^{N} \lambda_i \mathsf{cost}(e_i)$, subject to the next $N-1$ inequalities: for $i = 1, \ldots, N-1$,

$$\lambda_1 \mathsf{size}(e_1) + \cdots + \lambda_i \mathsf{size}(e_i) \leq k - \mathsf{size}(e_{i+1}).$$

From these inequalities, we can drive

$$\lambda_1 \mathsf{size}(e_1) + \cdots + \lambda_{i-1} \mathsf{size}(e_{i-1}) \leq k - \mathsf{size}(e_{i+1}) - \lambda_i \mathsf{size}(e_i) \leq k - \mathsf{size}(e_i)$$

by using $\mathsf{size}(e_{i+1}) \geq \mathsf{size}(e_i)$. Thus, the above $N-1$ inequalities can be simplified as

$$\lambda_1 \mathsf{size}(e_1) + \cdots + \lambda_{N-1} \mathsf{size}(e_{N-1}) \leq k - \mathsf{size}(e_N).$$

This is the KNAPSACK problem of $N-1$ items since λ_N is always 1. When $\mathsf{cost}(e_i) = \mathsf{size}(e_i)$ for all i, the problem is SUBSETSUM, which is still $\mathcal{NP}$-hard [144]. Thus, SUBSETSUM is polynomial-time reducible to Problem 2. Hence, the lemma holds. □

2.1.3 Competitive analysis

The competitive analysis [47] gives a performance measure of on-line algorithms: it evaluates an on-line algorithm by determining the worst-case ratio of the cost incurred by the algorithm to that incurred by an optimal off-line algorithm (or an adversary). More formally, let $ALG(I, k)$ be the cost incurred by an on-line algorithm ALG having the amount k of resources for an instance I of a minimization problem. Similarly, let $ADV(I, h)$ be the cost incurred by the adversary ADV having the amount h $(\leq k)$ of resources for I. If there are constants c and α such that, for all I,

$$ALG(I, k) \leq c \cdot ADV(I, h) + \alpha,$$

ALG with amount k of resources is said to be c-competitive against ADV with amount h of resources. The infimum of such c is called the *competitive ratio* of ALG with amount k of resources against an adversary with amount k of resources.

If ALG is a randomized algorithm, ALG is c-competitive if there exist constants c and α such that for all I,

$$\mathbf{E}[ALG(I,k) - c \cdot ADV(I,h)] \leq \alpha,$$

where $\mathbf{E}[\,]$ is the expectation taken over the random choices made by ALG. In the case of the file caching problem, I is a request sequence σ to files, and k and h are the memory spaces that ALG and ADV, respectively, can use for storing files.

Actually, we can define three types of adversaries when we analyze randomized on-line algorithms; the competitive ratio of ALG can be defined against each of the adversaries. The adversaries ADV are first classified into *oblivious* and *adaptive* adversaries, and, for the latter one, a further distinction is made with respect to adversary cost: *adaptive-offline* and *adaptive-online* adversaries. An *oblivious* adversary must choose the entire request sequence in advance, without any knowledge of the actions taken by ALG. In contrast, at each time, an adaptive adversary knows all the actions taken by ALG for servicing the requests thus far; the adaptive adversary may choose the next request based on this knowledge. An adaptive adversary is called *adaptive-offline* if its adversary cost is the optimal off-line cost on the request sequence that is created on-line by the adversary, while it is called *adaptive-online* if it must service each request it creates before ALG services the request. Note that, for deterministic on-line algorithms, these three types of adversaries are identical. If we denote by $OPT(I,h)$ the cost incurred by the optimal off-line algorithm with amount h of resources against instance I, we can reformulate the competitive ratio c as follows: for an oblivious adversary ADV,

$$\mathbf{E}[ALG(I,k)] - c \cdot OPT(I,h) \leq \alpha,$$

since $ADV(I,h) = OPT(I,h)$ and I is independent of the random choices of ALG, and for an adaptive off-line adversary ADV,

$$\mathbf{E}[ALG(I,k) - c \cdot OPT(I,h)] \leq \alpha,$$

since $ADV(I,h) = OPT(I,h)$ and I is a random variable depending on the random choices of ALG. However, such a reformulation is unknown for an adaptive-online adversary.

By the definition, adaptive-offline adversaries are obviously stronger than adaptive-online adversaries; adaptive-online adversaries are stronger than oblivious adversaries since the former can simulate the latter. In other words, if we let $C_{\mathrm{obl}}(ALG,k,h)$, $C_{\mathrm{adon}}(ALG,k,h)$ and $C_{\mathrm{adof}}(ALG,k,h)$ be the competitive ratios of ALG with amount k of resources against oblivious, adaptive-online, and adaptive-offline adversaries, respectively, with amount h of resource, the following holds:

$$C_{\mathrm{obl}}(ALG,k,h) \leq C_{\mathrm{adon}}(ALG,k,h) \leq C_{\mathrm{adof}}(ALG,k,h).$$

This implies the following. When we prove upper bounds on randomized competitive ratios, the stronger the adversary type, the stronger the result is; when we prove lower bounds, the weaker the adversary, the stronger the result. Surprisingly, it is known that there is no advantage to using randomization against adaptive-offline algorithms.

Proposition 2 ([40]) *If there is a c-competitive randomized on-line algorithm against adaptive-offline adversaries, then there exists a c-competitive deterministic on-line algorithm.*

Thus, our concern is oblivious and adaptive-online adversaries; we often try to prove upper bounds against adaptive-online adversaries and lower bounds against oblivious adversaries.

2.2 Algorithms for the bit model

We give a simple but sufficient condition for deterministic on-line algorithms to be competitive in the bit model. We first prove that any on-line algorithm with cache of size k satisfying the condition is $\frac{k}{\max\{k-h+1,s\}}$-competitive against adversary with cache of size h, where s is the size of the smallest file. We then give a general framework based on the condition to make a non-competitive algorithm "competitive" by mixing it with LRU. Thus, the framework enables us to make performance-guaranteed algorithms from non-competitive algorithms that perform well in situations that could be expected to be observed in real request sequences. The framework has a parameter c to change freely the extent to which the non-competitive algorithm dominates LRU. In terms of the competitive analysis, any algorithm made by the framework can be proved to be $\frac{ck}{\max\{k-h+1,s\}}$-competitive for a given positive integer c.

As an example, we make a competitive algorithm, called Competitive_SIZE, by applying the framework to a heuristics SIZE. The results of event-driven simulations show that Competitive_SIZE is much better than LRU in the case where small files are requested with high probability. In other cases, Competitive_SIZE is comparable to or only slightly worse than LRU. Furthermore, we show that Competitive_SIZE is better than either LRU and SIZE for a real web proxy log.

2.2.1 A sufficient condition for competitiveness

We first divide request sequence σ into phases from the first request in the following way. Let phase 0 be empty. For $i \geq 1$, phase i is the maximal subsequence in which the sum of the sizes of the different files requested is at most k (i.e., cache size), and the next request of the last request of phase $(i-1)$ is the first request

of phase i. Such a phase is called a k-phase. Note that the way of dividing σ into k-phases is uniquely determined and independent of the behavior of algorithms.

Now we give a simple condition on algorithms in terms of the k-phase.

Definition 3 (k-phase conservative condition) *If an algorithm incurs at most k bits in each k-phase, the algorithm is said to be k-phase conservative.*

We note that, in the bit model, Young's optimal algorithm [202] holds the above condition. In what follows, we denote any algorithm satisfying the condition by a k-phase conservative algorithm.

Before giving an upper bound of the competitive ratio of the k-phase conservative algorithm, we define several symbols. Let ϵ_k^- be the difference between k and the maximum possible value less than or equal to k made by summing the sizes of different files. ϵ_h^- is the same value as ϵ_k^- for h. Finally, let ϵ_k^+ be the minimum possible value larger than k made by summing the sizes of different files. Clearly, $0 < \epsilon_k^+ \leq S, 0 \leq \epsilon_k^- < S$ and $0 \leq \epsilon_h^- < S$, where S is the largest file size.

Theorem 3 *Let ALG be any k-phase conservative algorithm with cache of size k, and ADV be an optimal off-line algorithm with cache of size h ($\leq k$). ALG is $\frac{k-\epsilon_k^-}{\max\{k-h+\epsilon_k^+ +\epsilon_h^-, s\}}$-competitive, where s is the size of the smallest file.*

Proof. Clearly, ALG incurs cost of at most $(k - \epsilon_k^-)$ bits in each k-phase.

For arbitrary i, let the q_ith request be the first request of the ith k-phase, and B_i be the sum of the sizes of the different files requested in the ith k-phase. Consider the subsequence ρ from the $(q_i + 1)$st request to the q_{i+1}st request.

In the case where there is no request for the same file as σ_{q_i} in ρ, the sum of the sizes of the different files requested in ρ is $B_i - \mathsf{size}(\sigma_{q_i}) + \mathsf{size}(\sigma_{q_{i+1}})$ bits. Immediately after serving σ_{q_i}, the sum of the sizes of the files except σ_{q_i} in the ADV's cache is at most $h - \mathsf{size}(\sigma_{q_i}) - \epsilon_h^-$ bits. Thus, ADV incurs faults of at least

$$(B_i - \mathsf{size}(\sigma_{q_i}) + \mathsf{size}(\sigma_{q_{i+1}})) - (h - \mathsf{size}(\sigma_{q_i}) - \epsilon_h^-) = k - h + \epsilon + \epsilon_h^-$$

bits in ρ, where $B_i + \mathsf{size}(\sigma_{q_{i+1}})$ is denoted by $k + \epsilon$. By the definition of ϵ_k^+, we have $0 < \epsilon_k^+ \leq \epsilon$.

In the case where there is at least one request in ρ for the same file as σ_{q_i}, the sum of the sizes of the different files requested in ρ is $B_i + \mathsf{size}(\sigma_{q_{i+1}})$ bits. Immediately after serving σ_{q_i}, the sum of the sizes of the files including σ_{q_i} in the ADV's cache is at most $h - \epsilon_h^-$ bits. Thus, ADV incurs faults of at least

$$(B_i + \mathsf{size}(\sigma_{q_{i+1}})) - (h - \epsilon_h^-) = k - h + \epsilon + \epsilon_h^-$$

bits in ρ.

Therefore, ADV incurs faults of at least $k-h+\epsilon+\epsilon_h^-$ bits in ρ for both cases. Since $k-h+\epsilon+\epsilon_h^- \geq k-h+\epsilon_k^+ + \epsilon_h^- > 0$, ADV incurs a fault for at least one request, i.e., at least $\max\{k-h+\epsilon_k^+ + \epsilon_h^-, s\}$ bits in ρ.

If α is a constant determined by the last k-phase that may not be completed, we have, any σ,

$$ALG(\sigma) \leq \frac{k-\epsilon_k^-}{\max\{k-h+\epsilon_k^+ + \epsilon_h^-, s\}} ADV(\sigma) + \alpha.$$

□

Since the size is a positive integer, $k-h+\epsilon_k^+ + \epsilon_h^- \geq k-h+1$. Thus, the following holds.

Corollary 4 *Let ALG be any k-phase conservative algorithm with cache of size k, and ADV be an optimal off-line algorithm with cache of size h ($\leq k$). For any σ with the smallest file size s, ALG is $\frac{k}{\max\{k-h+1,s\}}$-competitive.*

Remark 3 *Consider the case where all files have size s. In this case, the problem is the paging problem where ALG has cache of size $\lfloor k/s \rfloor$ and ADV has cache of size $\lfloor h/s \rfloor$. Since the best achievable competitive ratio in the paging problem is $\frac{k}{k-h+1}$ [176] for any deterministic on-line algorithm with cache of size k and an optimal off-line algorithm with cache of size h, it follows that the competitive ratio of ALG is at least $\frac{\lfloor k/s \rfloor}{\lfloor k/s \rfloor - \lfloor h/s \rfloor + 1}$.*

Competitiveness for other cost models

For the general model, it is not difficult to see that the k-phase conservative algorithm is not competitive. For the fault model, the k-phase conservative algorithm is competitive. This is because the number of faults the k-phase conservative algorithm incurs in a k-phase is at most $\lfloor k/s \rfloor$ by the definition of the algorithm.

Theorem 5 *Let ALG be an any k-phase conservative algorithm with cache of size k, and ADV be an optimal off-line algorithm with cache of size h ($\leq k$). ALG is*

$$\begin{cases} \lfloor k/s \rfloor \text{-competitive} & \text{if } \lfloor k/S \rfloor \leq \lfloor h/s \rfloor \\ \frac{\lfloor k/s \rfloor}{\lfloor k/S \rfloor - \lfloor h/s \rfloor + 1} \text{-competitive} & \text{if } \lfloor k/S \rfloor > \lfloor h/s \rfloor \end{cases},$$

where s and S are the sizes of the smallest and largest file, respectively.

Proof. While serving the ith k-phase, ALG incurs at most $\lfloor k/s \rfloor$.

We define the q_ith request and ρ in the same way as the proof of Theorem 3. Assume that, during the ith k-phase, x different files are requested. Let y be the

number of the files including σ_{q_i} that will be requested in ρ, among the files that ADV have in its cache immediately before serving ρ.

In the case where there is no request in ρ for the same file as σ_{q_i}, ρ requests x different files. Thus, ADV incurs $x-(y-1)$ faults. In the other case, ρ requests $(x+1)$ different files including σ_{q_i}. ADV incurs $(x+1-y)$ faults. In both cases, ADV incurs $(x-y+1)$ faults. Thus the ratio of the number of faults of ALG to that of ADV is at most $\frac{\lfloor k/s \rfloor}{x-y+1}$.

Since $x \geq \lfloor k/S \rfloor$ and $y \leq \lfloor h/s \rfloor$, if $\lfloor k/S \rfloor > \lfloor h/s \rfloor$, the ratio attains maximum value $\frac{\lfloor k/s \rfloor}{\lfloor k/S \rfloor - \lfloor h/s \rfloor + 1}$ when $x = \lfloor k/S \rfloor$, $y = \lfloor h/s \rfloor$. If $\lfloor k/S \rfloor \leq \lfloor h/s \rfloor$, the ratio attains maximum value $\lfloor k/s \rfloor$ when $x = y$ since $y \leq x$ by definition. □

When $s = S = 1$, the upper bound is $\frac{k}{k-h+1}$, which is the competitive ratio of optimal deterministic algorithms for the paging problem [176].

2.2.2 A framework to generate competitive algorithms from heuristics and its application

LRU is a k-phase conservative algorithm and widely used in network caches since it is robust in that it performs well for various input distributions. However, there are numerous heuristics that are better than LRU under certain distributions of request sequences that would often appear in real environments, although much worse under others.

Based on the k-phase conservative condition, we give a framework that yields competitive algorithms by adaptively switching LRU and heuristics.

Framework of a competitive algorithm

Consider an algorithm, ALG, such that priority is given to each file by a function, say ψ. For a given parameter c, Figure 1 gives a $\frac{ck}{\max\{k-h+1,s\}}$-competitive algorithm, called Competitive_ALG, by adaptively switching ALG and LRU.

Competitive_ALG behaves like ALG at first but switches to LRU if ALG frequently incurs faults for the same files. Thus Competitive_ALG mimics ALG in the case where ALG is better, while it mirrors LRU otherwise. This adaptability has a significant advantage over the mixture of LRU and heuristics when the input distribution continuously changes, which is often the case with real environments.

To analyze Competitive_ALG, we generalize the k-phase conservative condition. An algorithm is said to be c-relaxed k-phase conservative, if it incurs at most $c \cdot k$ bits in each k-phase. By Lemma 7 below and a proof similar to that of Theorem 3, we have the following theorem.

Theorem 6 *Competitive_ALG is* $\frac{c \cdot k}{\max\{k-h+1,s\}}$ *-competitive in the bit model.*

Framework

1. Evict files in non-decreasing order of values given by ψ until a marked file is found or sufficient space is created. In the latter case, go to 3.
2. Evict files in LRU order from the remaining files including the marked file until sufficient space is created.
3. Store p, and mark it if the current fault is the cth one for p in the current k-phase.

Figure 1: Framework to generate competitive algorithms from heuristics. Variable p is the currently requested file for which the algorithm incurs a fault.

Lemma 7 *Competitive_ALG is c-relaxed k-phase conservative.*

Proof. Let p be a file for which Competitive_ALG incurred c faults in the ith k-phase up until the current request. Since p was marked in the third step of the above framework, p is not evicted in the first step. Assume that p is evicted in the second step when file r $(\neq p)$ is requested in the ith k-phase. Since the second step evicts files according to LRU, all files in the cache immediately after evicting p and before inserting r were requested at least once after the previous request to p. Thus these files were requested at least once in the ith k-phase. Let ΔC be the set of the files. It follows that at least $(\sum_{q\in\Delta C} \mathsf{size}(q)+\mathsf{size}(p)+\mathsf{size}(r))$ bits are requested in the ith k-phase. Since p is evicted to store r, $\sum_{q\in\Delta C} \mathsf{size}(q)+\mathsf{size}(p)+\mathsf{size}(r) > k$. This contradicts the definition of the k-phase. Therefore, p cannot be evicted in the ith k-phase. It follows that Competitive_ALG incurs at most c faults for each file in the ith k-phase. □

An implementation of Competitive_ALG

Competitive_ALG can be implemented by using two kinds of tables: a *fault-count table* and a *cache table*. The latter is usually used to store attributes of the files in cache, like size and the last request time, in order to maintain the files. The fault-count table holds the number of faults (fault-count) for each file requested in the current k-phase.

Consider a currently requested file p. When Competitive_ALG incurs a fault for p, it stores p, and marks p if the fault-count for p equals some specified value c. File marking is performed by setting "mark bit" to 1 for the file in the cache

```
For given parameter c, total priority order ψ,
cache size k, and currently requested file p,
1 procedure Competitive_ALG(file p)
2 begin
3   h=MaintainFT(p);/* get the fault count of p */
4   if(p is not in CT)
5     set lru_flag=0;
6     while(k- used_space_of_CT < size(p))
7       if(lru_flag==0 AND the lowest priority file r on ψ is not marked)
8         evict r;
9         set used_space_of_CT=used_space_of_CT - size(r);
10      else
11        set lru_flag=1;
12        evict least recently used file r';
13        set used_space_of_CT=used_space_of_CT - size(r');
14  if(h ≥ c)
15    insert p into CT with a mark;
16  else insert p into CT without a mark;
17 else return p to the request sender;
18end;
```

Figure 2: Competitive_ALG algorithm. CT and used_space_of_CT are the cache table and the sum of the size of files registered in CT, respectively. Subroutine MaintainFT is described in Figure 3.

table. Figure 2 describes more precisely the behavior of Competitive_ALG, which calls procedure MaintainFT in Figure 3 as a subroutine. The first, second and final steps of the framework of Competitive_ALG correspond to lines 7 to 9, 10 to 13 and 14 to 16, respectively. Procedure MaintainFT maintains the fault-count table. The table is initialized to be empty when entering a new k-phase (in lines 3 to 4). When inserting a file into the fault-count table, the fault-count of the file is set to 0 if the file is in the cache, and it is set to 1 otherwise (in lines 5 to 6).

2.2.3 Empirical results

One of the most popular heuristics is SIZE [13]. SIZE introduces file size as a parameter which determines the priority value of each file, based on the observation that there are relatively many requests to small files in real web requests. However, SIZE is not competitive, which can be easily shown by considering continuous requests to large files after requesting small files. As a reasonable example

```
1 procedure MaintainFT(file r)
2 begin
3   if(used_space_of_FT + size(r) > k)
4     flush FT;
5   if(r is in CT but not in FT)
6     insert r into FT with fault_count=0;
7   else if(r is in neither CT nor FT)
8     insert r into FT with fault_count=1;
9   else if(r is in FT but not in CT)
10    Set (fault_count of r) = (fault_count of r) +1;
11  return (fault_count of r);
12end;
```

Figure 3: Subroutine to maintain the fault-count table, which is used in Competitive_ALG in Figure 2. CT and FT are the cache table and the fault-count table, respectively, and used_space_of_FT denotes the sum of the size of files registered in FT.

of the use of the framework, we choose SIZE as ALG and set $\psi(\sigma_i) = \frac{1}{\mathsf{size}(\sigma_i)}$ for each file σ_i. The resulting algorithm is called Competitive_SIZE.

For several algorithms including Competitive_SIZE, we conducted both event-driven and trace-driven simulations.

Event-driven simulations

Let N be the number of distinct files to be requested, and let s and S be the maximum and minimum size, respectively, of files. We set $N = 500$, $s = 1$ and $S = 500$. The length of each request sequence is set to 10,000.

We used three kinds of request sequences generated in the following way. Let $p(i)$ and $\mathsf{size}(i)$ be the request probability and the size, respectively, of the ith file. We set $p(i)$ according to a Zipf distribution, in which $p(i)$ is proportional to $1/\pi(i)^\alpha$ $(0 < \alpha \leq 1)$, where α is called the Zipf parameter and π is a permutation vector. Several studies [74, 53] report that WWW traffic follows the Zipf distribution, but they differ on the value of α. Here, we set α at 1.0 and 0.8. The request sequences generated correspond to three kinds of correlation between size and request probability of files. Table 2 shows the request sequences, where (1) implies that smaller files are requested with higher probability, (2) implies that larger files are requested with higher probability, and (3) implies that there is no correlation between size and request probability of files.

The simulations examined Competitive_SIZEs with $c = 2$ and 3, LRU,

Table 2: Three kinds of request sequences, where $H_N = \sum_{j=1}^{N} 1/j$ (the Nth Harmonic number $\approx \ln N$).

	$p(i)$	$\mathsf{size}(i)$
(1)	$\frac{1}{iH_N}$	$s + \lfloor (i-1)\frac{S-s}{N-1} \rfloor$
(2)	$\frac{1}{iH_N}$	$s + \lfloor (N-i)\frac{S-s}{N-1} \rfloor$
(3)	$\frac{1}{iH_N}$	uniformly randomly chosen between s and S

SIZE [13], LOG(SIZE) [13], an SLRU [16]. The last three algorithms are defined as follows.

- SIZE evicts files in non-increasing order of file size.
- LOG(SIZE) divides the set of files in the cache by $\lceil \log_2(\text{file size}) \rceil$ into classes. LOG(SIZE) evicts files that belong to the class of the largest files in LRU order until sufficient space is created or all files in the class are evicted. In the latter case, LOG(SIZE) repeats the above operation for the class of the largest files among the remaining ones.
- SLRU evicts files in non-increasing order of the product of their size and the number of requests since the last request to the files.

Figures 4, 5 and 6 show the simulation results for request sequences, (1), (2), and (3), respectively, in the case of $\alpha = 1.0$, while Figures 7, 8 and 9 show the results for request sequences, (1), (2), and (3), respectively, in the case of $\alpha = 0.8$. The horizontal axes represent the cache size as a percentage of the storage capacity required for storing all files requested. The vertical axes represent weighted hit rate (WHR), which is the percentage of the sum of the size of hit files to that of the size of all files requested, and is also called the "byte hit rate."

From Figures 4 to 9, SIZE and LOG(SIZE) are the best two algorithms for (1), but extremely poor for the others. On the other hand, LRU is robust in that it has relatively high WHR in all cases, while it is worse than SIZE for (1).

More precisely, in the case of (1), Figures 4 and 7 show that the WHR of Competitive_SIZE with $c = 3$ is very near to that of the best algorithm for (1), SIZE. Competitive_SIZE with $c = 2$ is also much better than LRU, though it is slightly worse than SLRU when cache size is large. In the case of (2), Figures 5 and 8 show that Competitive_SIZEs with $c = 2$ and 3 are comparable to or only slightly worse than LRU unless cache size is small. SLRU is worse than LRU for all cache sizes. In the case of (3), Figures 6 and 9 show that Competitive_SIZEs with $c = 2$ and 3, LRU and SLRU achieve almost the same performance.

The summary of the above observation is that Competitive_SIZE is much better than LRU in the case of (1), while, in the cases of (2) and (3), the former is only slightly worse than LRU if the cache size is not too small. In the latter cases, the WHR of Competitive_SIZE approaches from below to that of LRU as cache size grows. This is for the reason below. Competitive_SIZE acts as SIZE at the beginning of each k-phase and may turn into LRU based on the fault-counts of stored files. Since the request probability of each file does not change through the simulation and the fault-count is initialized only when the algorithm enters the next k-phase, the behavior of Competitive_SIZE in the case of (3) becomes closer to that of LRU as k increases. On the other hand, SLRU and LOG(SIZE) do not have the characteristics like this, since they do not adaptively choose LRU or SIZE, but mix SIZE with LRU.

The observation in terms of the value of c is that as c grows the WHR of Competitive_SIZE in the best case, i.e., (1), increases at the expense of the WHR in the other cases, while as c decreases the worst-case WHR improves. This explains the fact that c expresses the extent to which the SIZE dominates LRU. Thus, we can tune Competitive_SIZE freely by choosing the value of c. For example, we should choose small c for robustness, i.e., improving the worst-case WHR, and large c for high WHR in the base case.

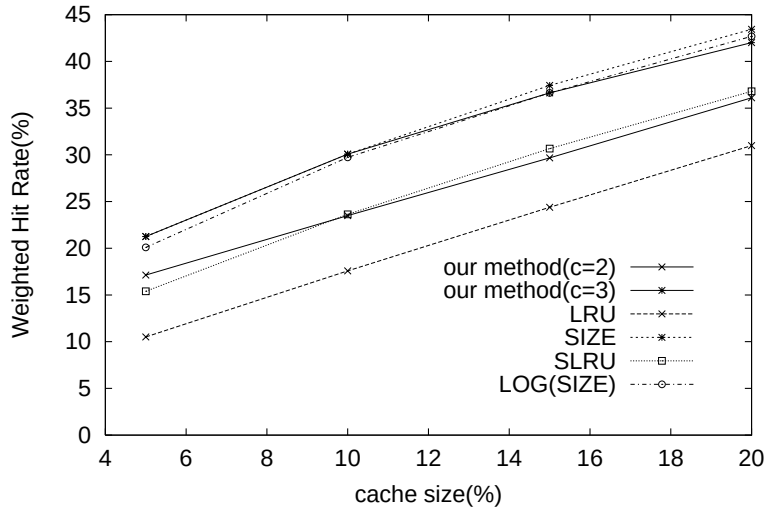

Figure 4: Case (1) of $\alpha = 1.0$: Smaller files are requested with higher probability.

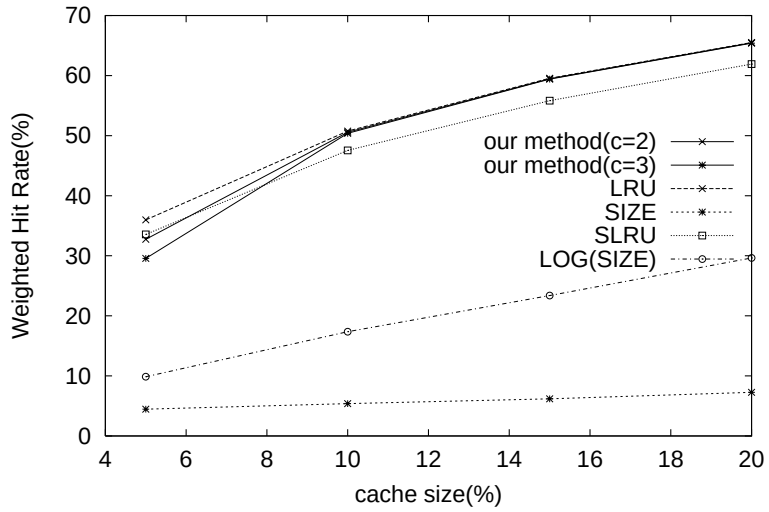

Figure 5: Case (2) of $\alpha = 1.0$: Larger files are requested with higher probability.

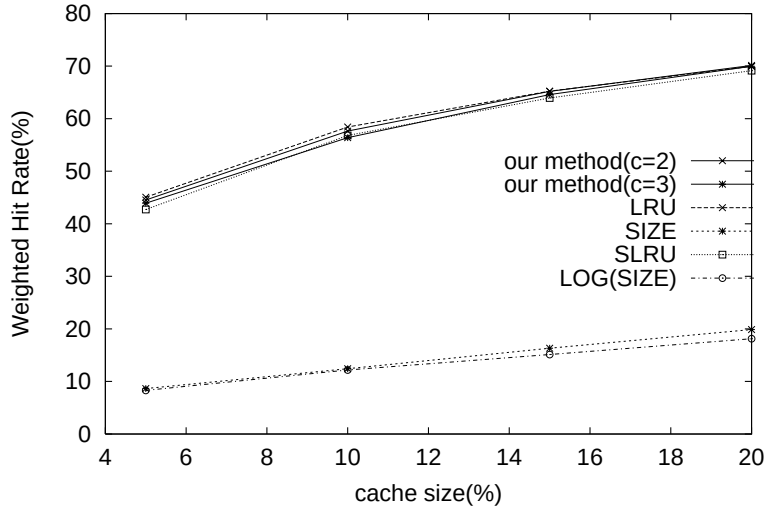

Figure 6: Case (3) of $\alpha = 1.0$: There is no correlation between size and request probability of files.

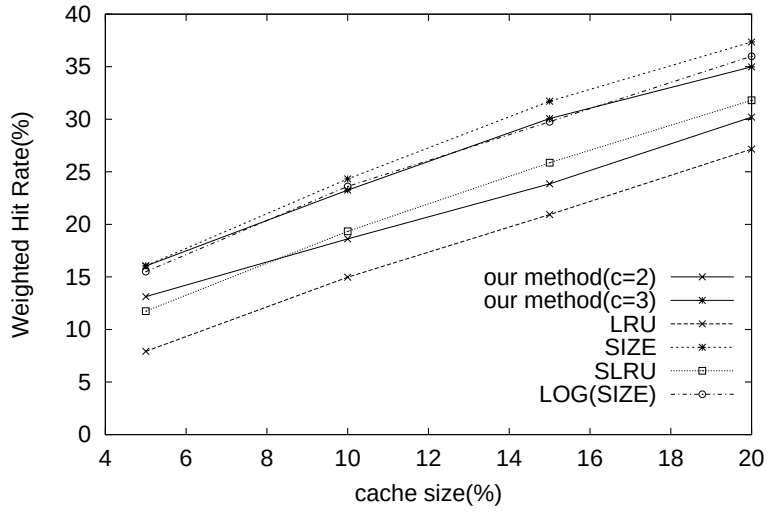

Figure 7: Case (1) of $\alpha = 0.8$: Smaller files are requested with higher probability.

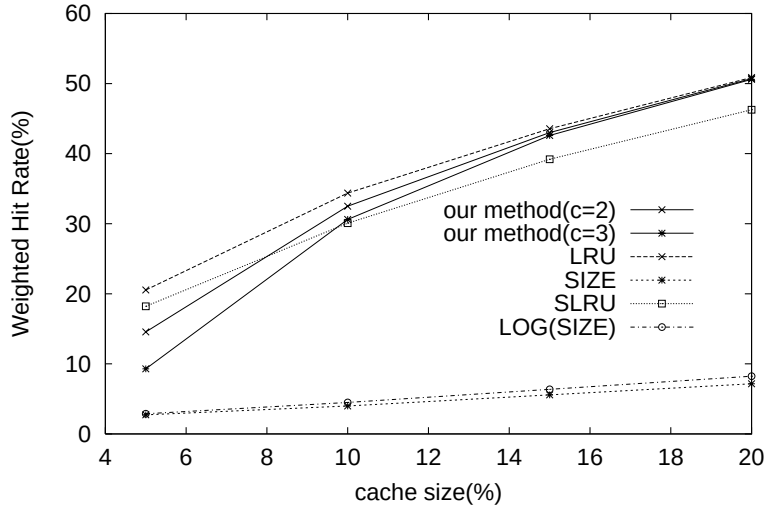

Figure 8: Case (2) of $\alpha = 0.8$: Larger files are requested with higher probability.

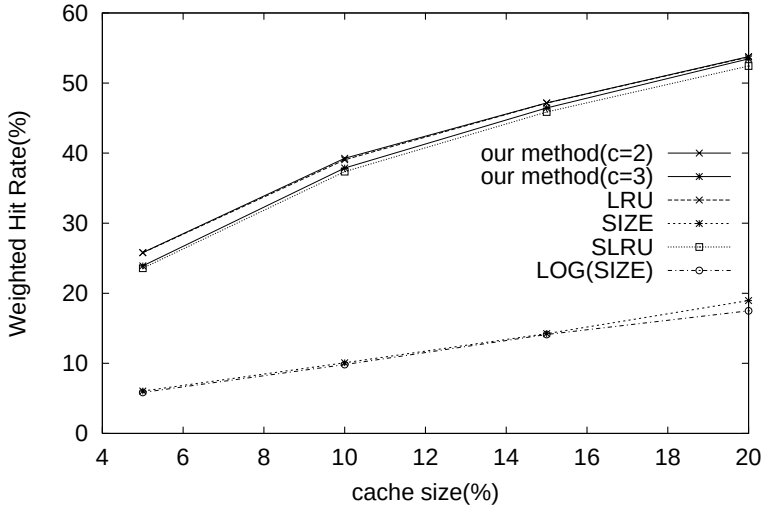

Figure 9: Case (3) of $\alpha = 0.8$: There is no correlation between size and request probability of files.

Trace-driven simulations

We also did trace-driven simulations for 100,000 requests from a proxy cache log of NLANR [156]. The results are shown in Figure 10. In the simulations, we used Competitive_SIZE with $c = 2$, LRU, SIZE, LOG(SIZE), and SLRU. In order to indicate the upper bound of WHR, we also conducted a simulation for a cache with unlimited storage and denote its result by INF. Horizontal and vertical axes plot the same parameters as for the above event-driven simulations. Figure 10 shows that Competitive_SIZE is best for almost all cache sizes and better than LRU for all cache sizes.

The interesting point is that Competitive_SIZE is better than either SIZE or LRU at least in the simulations. This implies that small files are requested with high probability in some request subsequences but not in others. Equivalently, SIZE is better than LRU for the some sequences, while SIZE is worse for the entire sequence as shown in Figure 10. Competitive_SIZE, roughly speaking, chooses SIZE when small files are frequently requested, and chooses LRU otherwise by using the fault-count table. This adaptability of Competitive_SIZE works well in the simulations, although we have to set c to an appropriate value.

These simulation results indicate that we should take into account local distributions of requests and their transitions. For example, when we fit a request sequence from logs to a Zipf distribution, the length of the request sequence can have a great impact on the resulting distribution. Observing a long request se-

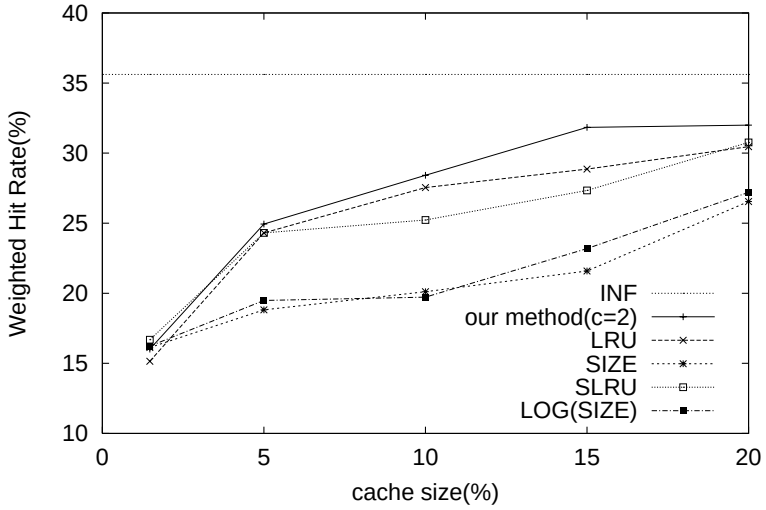

Figure 10: Simulation results using a proxy cache log of NLANR [156] of 100,000 requests.

quence results in suppressing the locality of requests. Excessively short request sequences are not sufficient to allow analysis. Therefore, it is important to model local distributions of requests and their transitions.

2.3 Algorithms for the general model

We will give a $\frac{k}{k-h+1}$-competitive randomized algorithm against an adaptive-online adversary with cache of size h that runs in $O(\log k)$ time in the worst case and is expected to take only $O(2^{\log^* k})$ time in the worst case and $O(\log^* k)$ in the amortized case[1], per file eviction or insertion. Thus, our algorithm is expected to be as competitive as Greedy-Dual-Size (GDS), which is an optimal deterministic algorithm, and is expected to take much shorter time than GDS in terms of both worst and amortized case.

To confirm practicality, trace-driven simulations were conducted for request sequences extracted from four proxy cache logs: two from NLANR [156] and the others from DEC [70]. Performance was measured in terms of the number of hits (hit rate), the sum of the size of hit files (byte hit rate), and the sum of the latency that would be required to get hit files (reduced latency). The results show that our

[1]We use the same terminology as in Ref. [140] i.e. $\log^* n$ is the smallest integer k such that k applications of the binary logarithm function applied to n, i.e., $\log(\log(\ldots\log(n)))$, is at most 1. Function $\log^* n$ increases very slowly. For $n \leq 65536$, $\log^* n \leq 4$; for $n \leq 2^{65536}$, $\log^* n \leq 5$. Actually, $2^{\log^* n} = o(\log n)$, proof is given later.

algorithm attains only slightly worse byte hit rates and sufficiently large reduced latency in comparison with GDS, while hit rate is worse than GDS. This implies that our algorithm mishits only files that are little valuable in terms of byte hit-rate and reduced latency.

In addition, we will show algorithms with cache of size k that are k-competitive against an adaptive-online adversary with cache of size k but not $\frac{k}{k-h+1}$-competitive against an oblivious (and thus, an adaptive-online) adversary with cache of size $h(\leq k)$: there are online algorithms that seem to have the same competitiveness against an adaptive-online adversary with cache of size k, but differ in competitiveness against an adaptive-online adversary with cache of size h. This suggests that competitive analysis against an adaptive-online adversary with cache of size h $(\leq k)$ can give more precise results than is seen against an adaptive-online adversary with cache of size k.

2.3.1 The algorithm and its analysis

Our algorithm is as follows.

Algorithm 4 (RECIPROCAL-SIZE(RS)) *If there is not enough space to store a requested file, repeatedly evict a randomly selected file, say p, from the cache until sufficient space is created, where the probability that p is selected is*

$$\frac{\mathsf{size}(p)/\mathsf{cost}(p)}{\sum_{x \in C_{RS}} \mathsf{size}(x)/\mathsf{cost}(x)},$$

and C_{RS} denotes the set of the files that RECIPROCAL-SIZE has in its cache.

Before proving the competitiveness of RS against an adaptive-online adversary, we review the standard potential-function-based technique, that is used in our proof.

Suppose that the number of request is n (that may depend on adaptive-online adversary ADV and the random choices of ALG). For $i = 1, \ldots n$, let Φ_{2i-1} and Φ_{2i} be the value of function Φ after an adaptive-online adversary (ADV) and an online algorithm (ALG), respectively, perform a series of elementary operations to service the ith request. Here, we assume that ALG services the ith request after ADV services the request. For an arbitrary request sequence given in an online problem, denote the cost of the ith operation of ADV and ALG by ADV_i and ALG_i, respectively.

Here, we assume the next two inequalities:

$$\Phi_{2i-1} - \Phi_{2(i-1)} \leq \alpha \cdot ADV_i$$

and

$$\Phi_{2i} - \Phi_{2i-1} \leq -ALG_i.$$

By summing these inequalities over the request sequence,

$$(\Phi_n - \Phi_0) \leq \sum_{i=1}^{n} (\alpha ADV_i - ALG_i),$$

where Φ_0 is the initial value of Φ. If the left value is bounded below, we have

$$\sum_{i=1}^{n} ALG_i \leq \alpha \sum_{i=1}^{n} ADV_i + constant,$$

i.e., ALG is α-competitive against an adaptive-online adversary.

Lemma 8 *RS with cache of size k is $\frac{k}{k-h+1}$-competitive against an adaptive-online adversary with cache of size h ($\leq k$).*

Proof. We first consider the actions of RS at the time when it misses requested file q. If there is enough space to store q in its cache when RS retrieves q by downloading q, RS puts q into its cache and no eviction occurs. Unless there is enough space, it evicts one or more files in order to create a space to store q.

Thus, the only actions of RS when it incurs a fault are file retrieval, and/or one or more file evictions followed by file retrieval. This is also true of an adaptive-online adversary (ADV). Thus we can consider file retrieval and file eviction as the two elementary operations for RS and ADV. In what follows, we will prove that, for each elementary operation performed by ADV, potential function Φ increases by at most $\frac{k}{k-h+1}$ times the value of the operation cost, and that for each elementary operation by RS, Φ decreases by at least the value of the operation cost. Note that retrieving file p costs $\mathsf{cost}(p)$, while any file can be evicted for free. This completes the proof if Φ is bounded from below.

The potential function we use is

$$\Phi = \frac{k}{k-h+1} \sum_{y \in C_{RS} \setminus C_A} \mathsf{cost}(y) - \sum_{y \in C_{RS}} \mathsf{cost}(y) + \frac{k}{k-h+1} \sum_{y \in C_A} \mathsf{cost}(y),$$

where C_{RS} and C_A are the set of files stored in the cache of, respectively, RS and ADV, and "$\setminus$" denotes set difference. For each request, RS services the request after ADV does.

Without loss of generality, we assume that both C_{RS} and C_A are initially empty, i.e., $\Phi = 0$. After having processed all requests, Φ is bounded from below if file cost is constant for any file, since Φ does not depend on the number of requests. Thus we will show the following facts at each request: (1) when ADV retrieves file q, Φ increases by at most $\frac{k}{k-h+1} \cdot \mathsf{cost}(q)$, (2) when RS retrieves file q, Φ is expected to decrease by at least $\mathsf{cost}(q)$, (3) at any other time Φ does not increase.

Eviction/retrieval by ADV: When ADV evicts file p, ADV incurs no cost. If p is in C_{RS}, p enters $C_{RS} \setminus C_A$ by evicting p from C_A; otherwise $C_{RS} \setminus C_A$ does not change. Thus, the first term of Φ increases by at most $\frac{k}{k-h+1} \cdot \mathsf{cost}(p)$. Clearly, the second term does not change. The third term decreases by $\frac{k}{k-h+1} \cdot \mathsf{cost}(p)$. Hence Φ does not increase.

When ADV retrieves file q, ADV incurs $\mathsf{cost}(q)$. If q is in C_{RS}, q leaves $C_{RS} \setminus C_A$ since q was not in C_A before retrieving q; otherwise $C_{RS} \setminus C_A$ does not change. Thus, the first term does not increase. Clearly, the second term does not change. The third term increases by $\frac{k}{k-h+1} \cdot \mathsf{cost}(q)$. Hence Φ increases by at most $\frac{k}{k-h+1} \cdot \mathsf{cost}(q)$.

Eviction/retrieval by RS: When RS evicts file p, RS incurs no cost. If p was in $C_{RS} \setminus C_A$ before evicting p, p leaves $C_{RS} \setminus C_A$ and the first term decreases by $\frac{k}{k-h+1} \cdot \mathsf{cost}(p)$. If p was not in $C_{RS} \setminus C_A$, $C_{RS} \setminus C_A$ does not change, i.e., the first term of Φ does not change. Since p is selected at probability $\frac{\mathsf{size}(p)/\mathsf{cost}(p)}{\sum_{x \in C_{RS}} \mathsf{size}(x)/\mathsf{cost}(x)}$, the expected decrease of the first term is

$$\frac{k}{k-h+1} \sum_{y \in C_{RS} \setminus C_A} \frac{\mathsf{size}(y)/\mathsf{cost}(y)}{\sum_{x \in C_{RS}} \mathsf{size}(x)/\mathsf{cost}(x)} \mathsf{cost}(y) = \frac{\frac{k}{k-h+1}\mathsf{size}(C_{RS} \setminus C_A)}{\sum_{x \in C_{RS}} \mathsf{size}(x)/\mathsf{cost}(x)},$$

where $\mathsf{size}(C_{RS} \setminus C_A)$ is the sum of the sizes of all files in $C_{RS} \setminus C_A$ (here and in the following analysis of expected decrease or increase, C_{RS} is the file set before evicting p). This notation of the size of a file set will be used hereafter. The second term of Φ increases by $\mathsf{cost}(p)$ with probability $\frac{\mathsf{size}(p)/\mathsf{cost}(p)}{\sum_{x \in C_{RS}} \mathsf{size}(x)/\mathsf{cost}(x)}$. Thus the second term is expected to increase by

$$\sum_{y \in C_{RS}} \frac{\mathsf{size}(y)/\mathsf{cost}(y)}{\sum_{x \in C_{RS}} \mathsf{size}(x)/\mathsf{cost}(x)} \mathsf{cost}(y) = \frac{\mathsf{size}(C_{RS})}{\sum_{x \in C_{RS}} \mathsf{size}(x)/\mathsf{cost}(x)}.$$

Clearly, the third term does not change. Thus, the expected decrease of Φ is

$$\frac{\frac{k}{k-h+1}\mathsf{size}(C_{RS} \setminus C_A) - \mathsf{size}(C_{RS})}{\sum_{x \in C_{RS}} \mathsf{size}(x)/\mathsf{cost}(x)}.$$

Since $\mathsf{size}(C_{RS})$ is at most k, $\frac{k}{k-h+1}\mathsf{size}(C_{RS} \setminus C_A) - \mathsf{size}(C_{RS})$ is not negative by Claim 9; i.e., Φ does not increase.

When RS retrieves file q, it incurs $\mathsf{cost}(q)$. Since q has already been retrieved by ADV, i.e., q is already in C_A, $C_{RS} \setminus C_A$ does not change. Hence, the first term of Φ does not change. The second term decreases by $\mathsf{cost}(q)$, since q enters C_{RS}. The third term does not change. Thus, Φ decreases by exactly $\mathsf{cost}(q)$.

This completes the proof. □

Claim 9 *Inequality* $\mathsf{size}(C_{RS} \setminus C_A) \geq k - h + 1$ *holds immediately before RS evicts a file to service a request.*

Proof. Let q be the file to be requested. Since ADV has serviced the request for q before q is requested to RS, ADV has q in its cache. Thus q is in $C_A \setminus C_{RS}$. This leads to $\mathsf{size}(C_A \setminus C_{RS}) \geq \mathsf{size}(q)$. Since $h \geq \mathsf{size}(C_A \setminus C_{RS}) + \mathsf{size}(C_{RS} \cap C_A)$, $h - \mathsf{size}(C_{RS} \cap C_A) \geq \mathsf{size}(q)$.

Let $\mathsf{size}(empty_{RS})$ be the size of unused space in RS's cache, i.e., $k - \mathsf{size}(C_{RS})$. It follows that

$$\begin{aligned} \mathsf{size}(empty_{RS}) + \mathsf{size}(C_{RS} \setminus C_A) &= k - \mathsf{size}(C_{RS}) + \mathsf{size}(C_{RS} \setminus C_A) \\ &= k - \mathsf{size}(C_{RS} \cap C_A) \\ &\geq k - h + \mathsf{size}(q). \end{aligned}$$

Since RS evicts a file in order to retrieve q, $\mathsf{size}(empty_{RS})$ is less than $\mathsf{size}(q)$. Hence $\mathsf{size}(C_{RS} \setminus C_A) \geq k - h + \mathsf{size}(q) - \mathsf{size}(empty_{RS}) \geq k - h + 1$, since file size is integer. □

Remark 4 *RS is a generalization of the RECIPROCAL algorithm [149] for the weighted paging problem: each file is identical in size but file cost is arbitrary. RECIPROCAL evicts file* p *with probability* $\frac{1/\mathsf{cost}(p)}{\sum_{x \in C_R} 1/\mathsf{cost}(x)}$ *where* C_R *is the set of files stored in the cache. In other words, RS is obtained by replacing* $\mathsf{cost}(p)$ *of RECIPROCAL's eviction probability with* $\mathsf{cost}(p)/\mathsf{size}(p)$*, which is the same approach used in generalizing the Greedy-Dual algorithm [200] to obtain Greedy-Dual-Size [63, 202].*

When RS is applied to the paging problem, a special case of the file caching problem, RECIPROCAL is called the RANDOM algorithm. Since RANDOM's competitive ratio is at least $\frac{k}{k-h+1}$ [165] against an oblivious adversary, Lemma 8 leads to the next theorem.

Theorem 10 *The competitive ratio of RS with cache of size* k *is* $\frac{k}{k-h+1}$ *against an oblivious/adaptive-online adversary with cache of size* h*.*

Implementing RS requires the dynamic generation of random variates on discrete distributions, such that, when given elements $1, 2, \ldots, N$ and their respective weights $w_1, w_2, \ldots, w_N \geq 0$, we generate integer j ($1 \leq j \leq N$) with probability $w_j / \sum_{1 \leq i \leq N} w_i$. Furthermore, we often need to update the set of the elements.

Matias, Vitter and Ni [140] gave an efficient randomized algorithm for generating random variates. An implementation of RS using their algorithm realizes the following time complexity.

Theorem 11 *RS is expected to take $O(2^{\log^* k})$ time in the worst case and $O(\log^* k)$ time in the amortized case per file insertion or eviction.*

Proof. Suppose that RS has N files in the cache, and that the files are numbered from 1 to N and have weights $w_1, w_2, \ldots, w_N \geq 0$, respectively. We note that the files need not be numbered serially, if the numbers have a one-to-one correspondence to the files.

When RS evicts a file, RS does the following process. RS first generates integer j ($1 \leq j \leq N$) with probability $w_j / \sum_{1 \leq i \leq N} w_i$ by using the algorithms in [140], and evicts the file whose number is j. Then, RS updates the data structure used by the algorithm to reflect the file eviction. Similarly, RS updates the data structure when RS inserts a file.

The algorithm is expected to take $O(\log^* k)$ time to generate a random variate in the worst case, and $O(2^{\log^* k})$ expected worst-case time and $O(\log^* k)$ expected amortized time to update the data structure according to insertion and eviction. □

Actually, $2^{\log^* n}$ is quite small as shown in the next proposition.

Proposition 12 $2^{\log^* n} = o(\log n)$

Proof. Define a function $F(k)$ such that $F(1) = 2$ and $F(k+1) = 2^{F(k)}$ for positive integer k, i.e.,

$$F(k) = \left. 2^{2^{\cdot^{\cdot^{\cdot^2}}}} \right\} k.$$

In what follows, we will prove $\lim_{n \to +\infty} \frac{2^{\log^* n}}{\log n}$=0, leading to $2^{\log^* n} = o(\log n)$.

For positive integer n such that $F(m) < n \leq F(m+1)$, $\log^* n = m + 1$ and $\log n > F(m-1)$ from the definition of $\log^* n$. Thus,

$$\frac{2^{\log^* n}}{\log n} < \frac{2^{m+1}}{F(m-1)} = 2 \cdot 2^{m - F(m-2)}.$$

Since $F(m) > 2^m$ ($m \geq 3$) can be easily shown by induction, $m - F(m-2) < m - 2^{m-2}$ for $m \geq 3$. This results in

$$\lim_{m \to +\infty} \{m - F(m-2)\} = -\infty,$$

i.e.,

$$\lim_{m \to +\infty} 2 \cdot 2^{m - F(m-2)} = 0.$$

Since $F(m)$ is a monotone increasing function,

$$\lim_{n \to +\infty} \frac{2^{\log^* n}}{\log n} = \lim_{m \to +\infty} 2 \cdot 2^{m - F(m-2)} = 0.$$

□

The binary-tree-based scheme developed by Wong and Easton [193] takes $O(\log k)$ time in the worst case to generate a radom variate and update the set of the elements. Thus, RS takes $O(\log k)$ time in the worst case per file insertion or eviction, by the same argument as the above.

2.3.2 Comparison with RECIPROCAL

RECIPROCAL [165] takes the file cost into account but not file size. Intuitively, RECIPROCAL seems to be much worse than RS in addressing the file caching problem. However, it is as competitive as RS against an adaptive-online adversary with cache of size k from Theorem 10, the next lemma, and the fact that the competitive ratio of RANDOM is at least k [165] against an oblivious adversary.

Lemma 13 *RECIPROCAL with cache of size k is k-competitive against an adaptive-online adversary with cache of size k.*

Proof. As in Lemma 8, for each file eviction and retrieval of an adaptive-online adversary (ADV) and RECIPROCAL, we consider the increase or decrease of the potential function,

$$\Phi = k \sum_{y \in C_R \setminus C_A} \mathsf{cost}(y) - \sum_{y \in C_R} \mathsf{cost}(y) + k \sum_{y \in C_A} \mathsf{cost}(y),$$

where C_R and C_A are the sets of files stored in the cache of, respectively, RECIPROCAL and ADV. For each request, RECIPROCAL services the request after ADV does.

Eviction/retrieval by ADV: When ADV evicts file p, ADV incurs no cost. The first term of Φ increases by at most $k \cdot \mathsf{cost}(p)$, since p may enter $C_R \setminus C_A$. The second term does not change. The third term decreases by $k \cdot \mathsf{cost}(p)$. Hence Φ does not increase.

When ADV retrieves file q, ADV incurs $\mathsf{cost}(q)$. The first term does not increase. The second term does not change, and the third term increases by $k \cdot \mathsf{cost}(q)$. Hence Φ increases by at most $k \cdot \mathsf{cost}(q)$.

Eviction/retrieval by RECIPROCAL: When RECIPROCAL evicts file p, it incurs no cost. If p was not in $C_R \setminus C_A$, the first term of Φ does not change; otherwise the first term decreases by $k \cdot \mathsf{cost}(p)$. Thus the expected decrease of the first term is

$$k \sum_{y \in C_R \setminus C_A} \frac{1/\mathsf{cost}(y)}{\sum_{x \in C_R} 1/\mathsf{cost}(x)} \mathsf{cost}(y) = \frac{k|C_R \setminus C_A|}{\sum_{x \in C_R} 1/\mathsf{cost}(x)},$$

where $|C_R \setminus C_A|$ represents the number of files in $C_R \setminus C_A$. The second term of Φ is expected to increase by

$$\sum_{y \in C_R} \frac{1/\mathsf{cost}(y)}{\sum_{x \in C_R} 1/\mathsf{cost}(x)} \mathsf{cost}(y) = \frac{|C_R|}{\sum_{x \in C_R} 1/\mathsf{cost}(x)},$$

where $|C_R|$ is the number of files stored in RECIPROCAL's cache. The third term does not change. Thus, the expected decrease in Φ is

$$\frac{k|C_R \setminus C_A| - |C_R|}{\sum_{x \in C_R} 1/\mathsf{cost}(x)}.$$

Claim 9 is still true of RECIPROCAL, since it does not depend on the eviction policy of RS. Thus $\mathsf{size}(C_R \setminus C_A) \geq k - k + 1 = 1$. This implies $|C_R \setminus C_A| \geq 1$. Thus $k|C_R \setminus C_A| - |C_R|$ is not negative, i.e., Φ does not increase, since $|C_R|$ is at most k.

When RECIPROCAL retrieves file q, it incurs $\mathsf{cost}(q)$. Since q is already in C_A, the first term of Φ does not change. The second term decreases by $\mathsf{cost}(q)$. The third term does not change. Hence Φ decreases by exactly $\mathsf{cost}(q)$. □

The next lemma implies that the competitive ratio of RECIPROCAL with cache of size k is more than that of RS with cache of size k, against an oblivious/adaptive-online adversary with cache of size h. The proof is similar to, but much more complicated than, that introduced in the lower bound theorem of the RANDOM algorithm for the paging problem [165]. The proof uses the next proposition.

Proposition 14 ([149]) *Let X be a random variable giving the "waiting time" for success in a sequence of Bernoulli trials with success probability p. For any positive integer t, define the "truncated" random variable, X_t, such that $X_t = X$ if $X \leq t$ and $X_t = t$ otherwise. It follows that the expected value of X_t is $\frac{1}{p}(1 - (1-p)^t)$.*

Lemma 15 *For the file caching problem, the competitive ratio of RECIPROCAL with cache of size k is more than $\frac{k}{k-h+1}$ against an oblivious (and thus, an adaptive-online) adversary with cache of size h $(k > h > 10)$.*

Proof. Let ϵ be given and the cache of RECIPROCAL be initially empty. We consider the following request sequence:

$$\sigma = (b_1, a_2, \ldots, a_{h-1})^m, (b_2, a_2, \ldots, a_{h-1})^m, (b_3, a_2, \ldots, a_{h-1})^m \cdots,$$

where m is an integer whose value is determined later, and $\mathsf{size}(b_i) = 2$, $\mathsf{size}(a_j) = 1$ and $\mathsf{cost}(b_i) = \mathsf{cost}(a_j) = 1$ for each i $(i = 1, 2, \ldots)$ and j $(j = 2, 3, \ldots, h-1)$.

For each $i = 1, 2, \ldots$, the subsequence $(b_i, a_2, \ldots, a_{h-1})^m$ of σ, called the ith block, is composed of m repetitions of the segment $b_i, a_2, \ldots, a_{h-1}$. RECIPROCAL incurs the first fault while servicing the $\left(\left\lfloor \frac{k-h+2}{2} \right\rfloor + 1\right)$st block since the sum of the sizes of the files requested until the ith block is $2i + h - 2$, and $2 \times \left\lfloor \frac{k-h+2}{2} \right\rfloor + h - 2 \leq k < 2\left(\left\lfloor \frac{k-h+2}{2} \right\rfloor + 1\right) + h - 2$. In what follows, we consider the ith block such that $i \geq \left\lfloor \frac{k-h+2}{2} \right\rfloor + 1$. Clearly, the oblivious adversary incurs exactly one fault for b_i during the ith blocks for each $i \geq 2$, since the sum of the sizes of files requested in each block is h.

Immediately before servicing the ith block, RECIPROCAL keeps at most $h - 2$ files of those requested in the block since a_j's are common in all blocks but b_i is only in the ith block. Hereafter we consider RECIPROCAL's cache when servicing segment $b_i, a_2, \ldots, a_{h-1}$ of the ith block. We say that RECIPROCAL succeeds if, on its fault, RECIPROCAL has exactly $h-2$ files of $\{b_i, a_2, \ldots, a_{h-1}\}$ and it evicts (if necessary) a file other than $b_i, a_2, \ldots, a_{h-1}$. RECIPROCAL evicts at most one file when it succeeds since it evicts certain $b_{i'}$ $(i' \neq i)$ if it does, which creates a space of size 2. For each segment repetition until after success, RECIPROCAL incurs at least one fault, since RECIPROCAL does not have b_i before servicing the ith block. Thus we will evaluate the lower bound of the expected number of segment repetitions until success within the ith block. This lower bound directly represents that of the competitive ratio since the oblivious adversary incurs just one fault during each block (except the first one). In what follows, we consider the lower bound for the two cases where $k - h + 1$ is either even or odd.

(A) the case where $k - h + 1$ is even:
(A-I) When RECIPROCAL incurs a fault for b_i and keeps files $a_2, \ldots, a_{h-1}$ in its cache, RECIPROCAL also keeps $\lfloor \frac{k-(h-2)}{2} \rfloor = \frac{k-h+1}{2}$ $b_{i'}$'s such that $i' \neq i$, since $k - (h - 2)$ is odd. To store b_i and succeed, RECIPROCAL evicts a certain $b_{i'}$ $(i' \neq i)$, since the unoccupied space is of size 1. Thus, the probability of a success is at most

$$\frac{\frac{k-h+1}{2}}{\frac{k-h+1}{2} + h - 2} = \frac{k-h+1}{k+h-3},$$

since each file cost equals 1 and RECIPROCAL evicts a file with equi-probability.
(A-II) When RECIPROCAL incurs a fault for certain a_l and keeps files $b_i, a_2, \ldots, a_{l-1}, a_{l+1}, \ldots, a_{h-1}$ in its cache, RECIPROCAL also holds $\lfloor \frac{k-(h-3)-2}{2} \rfloor = \frac{k-h+1}{2}$ b_i's. Thus, the probability of a success is at most

$$\frac{\frac{k-h+1}{2}}{\frac{k-h+1}{2} + h - 2} = \frac{k-h+1}{k+h-3}.$$

From (A-I) and (A-II), the expected number of faults incurred by RECIPROCAL during the ith block is at least

$$\frac{k+h-3}{k-h+1}\left(1-\left(1-\frac{k-h+1}{k+h-3}\right)^m\right),$$

by using Proposition 14.

For any given ϵ, there exists sufficiently large m such that the online-to-offline cost ratio with respect to σ is larger than $\frac{k+h-3}{k-h+1}-\epsilon$, since $\frac{k-h+1}{k+h-3}<1$ $(k\geq h>2)$. Clearly, $\frac{k+h-3}{k-h+1}>\frac{k}{k-h+1}$ under $h>3$.

(B) the case where $k-h+1$ is odd:

(B-I) When RECIPROCAL incurs a fault for b_i and keeps files $a_2,\ldots,a_{h-1}$ in its cache, RECIPROCAL also keeps $\lfloor\frac{k-(h-2)}{2}\rfloor=\frac{k-h+2}{2}$ b_i's. Note that $k-(h-2)$ is even. Thus, the probability of success is at most

$$\frac{\frac{k-h+2}{2}}{\frac{k-h+2}{2}+h-2}=\frac{k-h+2}{k+h-2}.$$

(B-II) When RECIPROCAL incurs a fault for a certain a_l and keeps files $b_i,a_2,\ldots,a_{l-1},a_{l+1},\ldots,a_{h-1}$ in its cache, RECIPROCAL also keeps $\lfloor\frac{k-(h-3)-2}{2}\rfloor=\frac{k-h}{2}$ b_i's. This implies that there is free space of size 1 in RECIPROCAL's cache. Since size$(a_l)=1$, no file is evicted when inserting a_l. In this case, RECIPROCAL succeeds with probability 1. However, this success is thanks to the fact that RECIPROCAL keeps $b_i,a_2,\ldots,a_{l-1},a_{l+1},\ldots,a_{h-1}$. In other words, RECIPROCAL "implicitly succeeded" at the last fault.

More formally, we say that RECIPROCAL "implicitly succeeds" if on its fault, RECIPROCAL has exactly $h-3$ files of $\{b_i,a_2,\ldots,a_{h-1}\}$ and after inserting the new file RECIPROCAL has exactly b_i and $h-3$ files of $\{a_2,\ldots,a_{h-1}\}$ by evicting (if necessary) files other than $b_i,a_2,\ldots,a_{h-1}$. It follows that RECIPROCAL will succeed during the next $h-1$ requests after it implicitly succeeds during the ith block. The number of repetitions of the segment until an implicit success is equal to or smaller by 1 than that of the repetitions of the segment until success, since RECIPROCAL must implicitly succeed before success. Thus the expected number of faults is at least equal to the number of repetitions of the segment until an implicit success. We can regard the probability of an implicit success as that of success in the case where the adversary's cache size is $h-1$. Hence the probability of an implicit success is at most

$$\frac{k-(h-1)+1}{k+(h-1)-3}=\frac{k-h+2}{k+h-4},$$

since $k-(h-1)+1$ is even.

When file g is requested:
1 `if` is not in the cache `then`
2 `until` there is space for g in the cache:
3 For each file f in the cache,
decrease $\mathsf{credit}(f)$ by $\Delta \cdot \mathsf{size}(f)$,
where $\Delta := \min_{f \in cache} \mathsf{credit}(f)/\mathsf{size}(f)$.
4 Evict from the cache any file f
such that $\mathsf{credit}(f) = 0$.
5 Bring g into the cache
and set $\mathsf{credit}(g) \leftarrow \mathsf{cost}(g)/\mathsf{size}(g) = 0$.
6 `else` Reset $\mathsf{credit}(g)$ to any value
between its current value and $\mathsf{cost}(g)$.

Figure 11: Greedy-Dual-Size algorithm [202].

From (B-I), (B-II), $\frac{k-h+2}{k+h-2} < \frac{k-h+2}{k+h-4}$ and Proposition 14, the expected number of faults incurred by RECIPROCAL during the ith block is at least

$$\frac{k+h-4}{k-h+2}\left(1-\left(1-\frac{k-h+2}{k+h-4}\right)^m\right).$$

For any given ϵ, there exists a sufficiently large m such that the online-to-offline cost ratio with respect to σ is larger than $\frac{k+h-4}{k-h+2} - \epsilon$. Further, we have $\frac{k+h-4}{k-h+2} > \frac{k}{k-h+1}$ if $k > h > 10$, since

$$\frac{k+h-4}{k-h+2} - \frac{k}{k-h+1} = \frac{(k-h)(h-5)-4}{(k-h+2)(k-h+1)} > 0 \ (k > h > 10).$$

By (A) and (B), the proof is completed. □

Greedy-Dual-Size(GDS) [202] and its original version, Greedy-Dual [200] have a similar relation. Greedy-Dual-Size maintains a variable "credit" for each file as shown in Figure 11. Greedy-Dual-Size evicts a file if it has a credit of 0 when Greedy-Dual-Size needs to create space. If there is no such file, it decreases the credit $\mathsf{credit}(f)$ of each file f by $\Delta \cdot \mathsf{size}(f)$ where Δ is the minimum value of credit-size ratio among all cached files. When retrieving and storing a file, Greedy-Dual-Size sets its credit as its cost. At the time of a hit, Greedy-Dual-Size can optionally raise the credit of the hit file to at most its cost in line 6. Whether raising the credit or not, Greedy-Dual-Size with cache size k has competitive ratio $\frac{k}{k-h+1}$ [202] against an adversary with cache size h. Greedy-Dual can be obtained

by setting size of every file to unit size in Figure 11. The next theorem says that Greedy-Dual's competitive ratio is larger than Greedy-Dual-Size when file size is arbitrary.

Theorem 16 *For the file caching problem, Greedy-Dual is k-competitive against an adversary with cache of size k but not $\frac{k}{k-h+1}$-competitive against an adversary with cache of size h $(\leq k)$.*

The proof is similar to those of Lemmas 8 and 15.

Thus, Young's result [202], i.e., $\frac{k}{k-h+1}$-competitiveness of GDS, is meaningful while Cao and Irani [63] proved before Young that GDS is k-competitive against an adversary with cache of size k.

2.3.3 Empirical results

We conducted trace-driven simulations for four proxy cache logs: two from NLANR [156] and the others from DEC [70]. We call the four logs, "Log I", "Log II", "Log III" and "Log IV". The number of requests in the logs ranges from 260,000 to 830,000. Performance was measured in terms of hit rate, byte hit rate, and reduced latency.

We tested the following algorithms:

- RECIPROCAL-SIZE (RS)
- RECIPROCAL (R) [165]
- FIFO version of Greedy-Dual-Size (GDS) [202]
- LRU version of Greedy-Dual-Size (GDS(LRU)) [63, 202]
- Greedy-Dual (GD) [200]
- Least Recently Used (LRU).

GDS is obtained by not raising the credit in line 6 of Figure 11. GDS(LRU) always raises $\mathsf{credit}(g)$ to $\mathsf{cost}(g)$ in line 6. Intuitively, GDS gradually decreases the value of each file in the cache regardless of whether the file is requested frequently. GDS is similar to FIFO in this sense. GDS(LRU) also gradually decreases the value, however, when a file is hit, GDS(LRU) raises its value to its initial value. Hence, GDS is similar to LRU.

These algorithms can be characterized by the three types of information they consider: request frequency, file size, and file cost. Request frequency is the information on how frequently each file has been requested while file size and cost depend only on the characteristics of each file. Table 3 summarizes the information used by each algorithm.

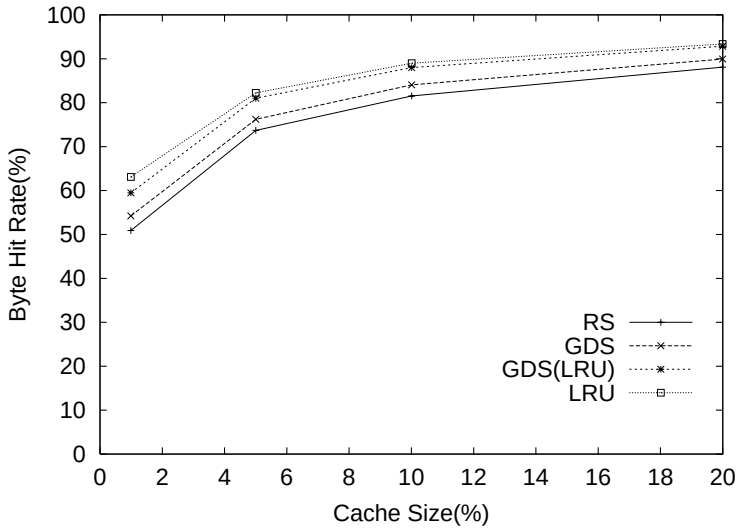

Figure 12: Byte hit rate for Log I.

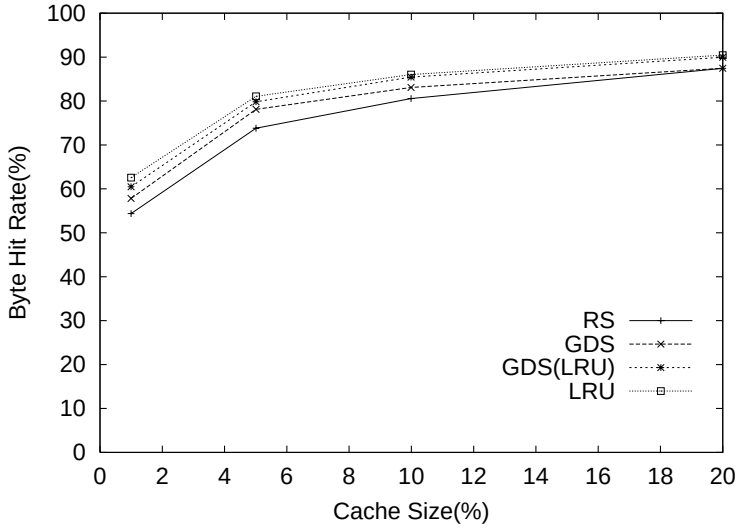

Figure 13: Byte hit rate for Log II.

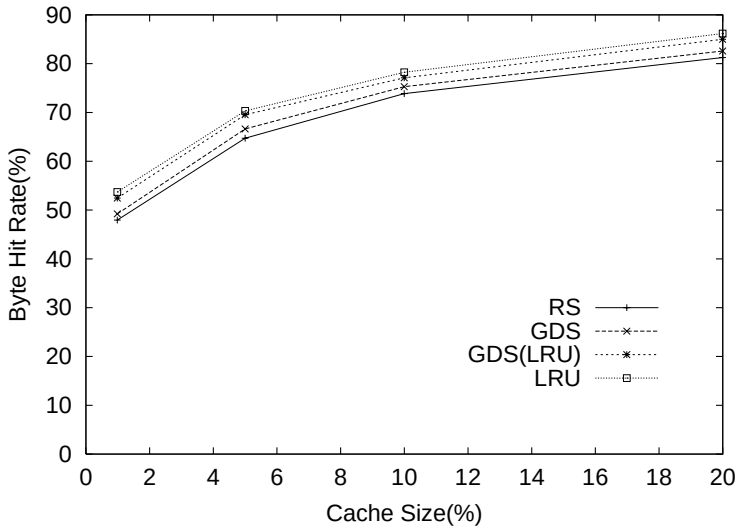

Figure 14: Byte hit rate for Log III.

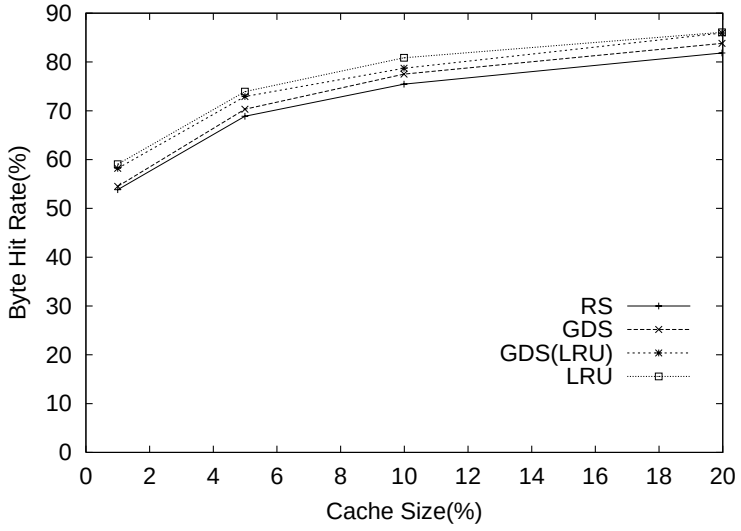

Figure 15: Byte hit rate for Log IV.

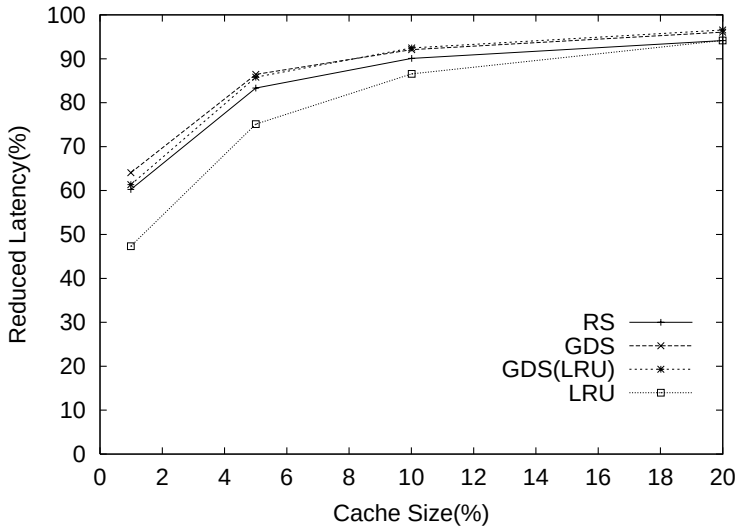

Figure 16: Reduced latency for Log I.

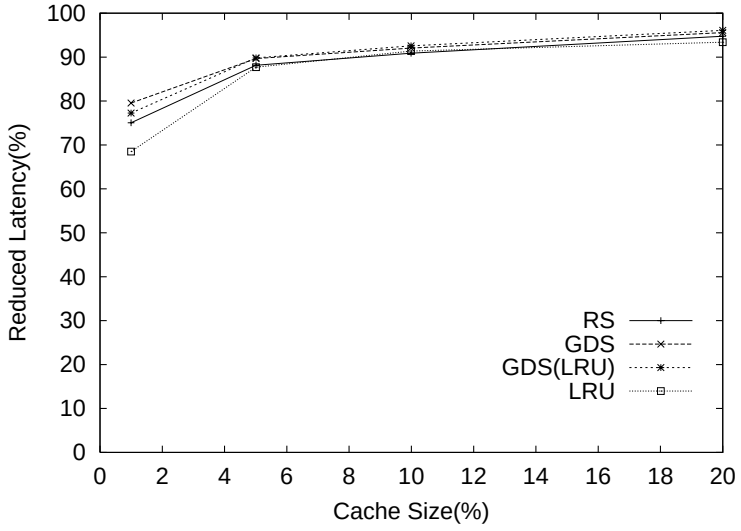

Figure 17: Reduced latency for Log II.

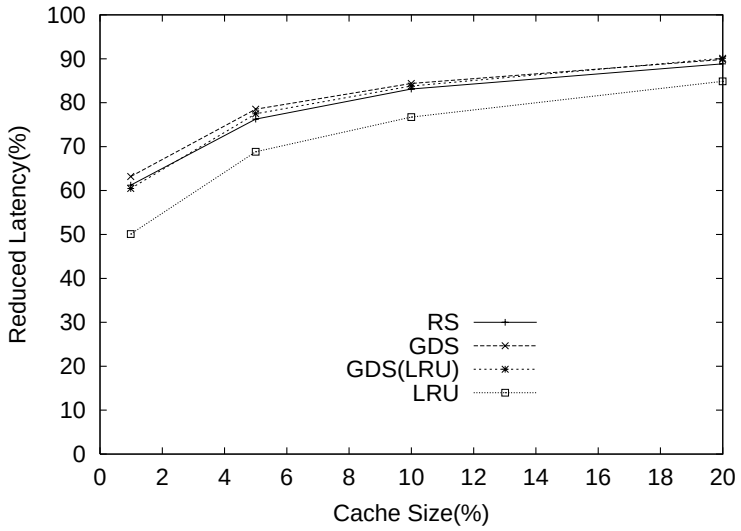

Figure 18: Reduced latency for Log III.

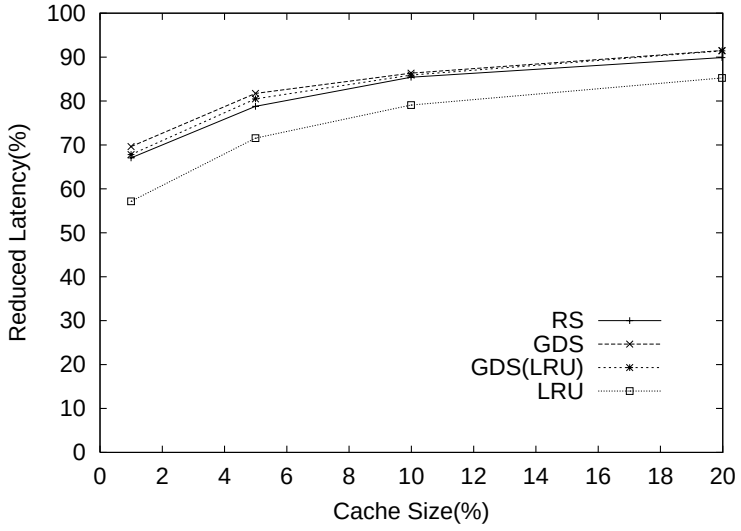

Figure 19: Reduced latency for Log IV.

Table 3: Characterization of algorithms by three types of information. If an algorithm takes a certain type of information into account, "◯" is placed in the corresponding square; otherwise "×."

	RS	R	GDS	GDS(LRU)	GD	LRU
Request Frequency	×	×	×	◯	◯	◯
File Size	◯	×	◯	◯	×	×
File Cost	◯	◯	◯	◯	◯	×

In the simulations, algorithms R and GD performed much worse than RS, GDS and GDS(LRU). These results would indicate that the competitive ratios of R and GD against an adversary with cache of size h are larger than those of the others. In the following, we show the results for RS, GDS, GDS(LRU) and LRU.

The first simulation involved the bit model, i.e., the cost model in which file cost equals file size. Figures 12, 13, 14 and 15 are the results for Log I, II, III and IV, respectively. The horizontal axes represent the percentage of the cache size that the algorithms have against the sum of the size of the different files in the corresponding request sequence. The vertical axes plot the percentage of the total saved traffic (i.e., the sum of the size of hit files) of the algorithms against that of a cache that has infinite space.

The second simulation involved the cost model in which file cost is taken as latency (i.e., elapsed time from when the cache accepts an HTTP request until it responds) as extracted from the logs. Figures 16, 17, 18 and 19 are the results for Log I, II, III and IV, respectively. The horizontal axes have the same meaning as the first simulation. The vertical axes plot the percentage of the total reduced latency of the algorithms against that of a cache that has infinite space.

The first simulation results show that LRU is the best, and GDS(LRU) is comparable. RS and GDS is only slightly worse than LRU. From the second simulation, GDS(LRU), GDS and RS are the best. LRU performance worsens as cache size decreases. The results indicate that RS performs very well in terms of reduced latency and attains sufficiently good byte hit rate.

LRU attains the best byte hit rate. In addition, LRU is not bad at reducing latency, except for the case of small cache size. From a worst-cast analysis point of view, however, LRU is not competitive in the general model.

For all simulations, GDS performs only slightly worse than GDS(LRU). This may indicate that each request is little correlated to other requests for the same file as is usual in second-level caches. GDS(LRU) and LRU have the possibility of performing better for other logs that have much correlation between requests.

Even so, RS is a good candidate for caches requiring high-speed processing, since a reasonable situation requiring high-speed caches is second-level caching in the large networks like those of network carriers.

2.4 Summary

In the bit model, LRU is competitive and can be performed in $O(1)$ time per file eviction by using doubly-linked lists, but does not take file size into account. On the other hand, there are many heuristics considering the difference in size of files that work well for the input distributions that are typical in practical environments.

We gave a simple but sufficient condition such that any algorithm subject to the condition is $\frac{k}{\max\{k-h+1,s\}}$-competitive, where s is the size of the smallest file. Based on this condition, we developed a general framework to easily generate competitive algorithms from non-competitive heuristics; the generated algorithms adaptively switch from the heuristics to LRU and vice versa according to input requests. As an application of the framework, we constructed a competitive algorithm, called Competitive_SIZE, from SIZE heuristics. The results of our event-driven simulations show that Competitive_SIZE is much better than LRU when smaller files are requested with higher probability. In other cases, Competitive_SIZE is comparable to or only slightly worse than LRU. Furthermore, we showed that Competitive_SIZE is better than either LRU or SIZE for a real web proxy log. This is because Competitive_SIZE chooses the most appropriate strategy among SIZE and LRU, by implicitly analyzing local distributions of the given request sequence. However, in general, the criterion for choosing strategies would be quite sensitive. It would depend on how frequently local distributions change, as well as storage space of cache and the heuristics that was made competitive. In this sense, we have to tune the generated algorithm through simulations to the real environments in which the algorithm is used.

In the general model, the Greedy-Dual-Size (GDS) is an optimal deterministic competitive algorithm and is shown to achieve excellent performance in experiments in terms of the amount of cost paid on faults. GDS with cache of size k, however, takes as many as $O(k)$ steps per file eviction in the worst case. This would be disadvantage especially in file caching services to many hosts, which requires a quick response to each request.

We gave a fast randomized algorithm that runs in $O(\log k)$ time in the worst case and is expected to run in only $O(2^{\log^* k})$ time per file eviction and file insertion, while the algorithm is expected to be $\frac{k}{k-h+1}$-competitive against an oblivious/adaptive-online adversary , i.e., as competitive as GDS on average. We conducted trace driven simulations to confirm the practicality of our algorithm. Experimental results show that our algorithm performs very well in terms of reduced latency and attains a sufficiently good byte hit-rate.

For the future work, we have several problems.

In the bit model, we have already known optimal deterministic on-line algorithms such as LRU and an optimal randomized on-line algorithm (see the brief survey in the next subsection). An interesting problem is to know whether these algorithms are still optimal or not if we introduce some restriction on input request sequences to competitive analysis, such as access graphs and the Markov model introduced in the analysis of the paging problem [47].

In the general model, we have devised a fast randomized algorithm that is expected to be as competitive as GDS. However, we do not know the competitive ratio of optimal randomized on-line algorithms (against an oblivious adversary). It would be an interesting open problem to know the optimal randomized competitive ratio, since we do not know any randomized online algorithm that achieves the competitive ratio less than k against an oblivious adversary even in a special case, i.e., the weighted caching problem, in which every file size equals 1. Another interesting direction is to develop a framework to generate a competitive algorithm from a non-competitive heuristics, as we have done in the bit model.

For both of the bit and general models, it is $\mathcal{NP}$-hard to solve the file caching problem in an off-line manner. It is a well-known open problem to settle the hardness of the off-line file caching problem in the fault model.

2.5 A brief survey of the caching problem

The file caching problem is a generalized form of the paging problem associated with microprocessor caches, which has been extensively researched for decades. The difference between these problems is that the former deals with objects (i.e., files) of various size while the latter handles objects (i.e., pages) of uniform size.

Sleator and Tarjan [176] first analyzed on-line paging algorithms by comparing their performance on any sequence of requests to that of an optimal off-line algorithm (i.e., an *adversary*): they applied the competitive analysis. They showed that the least-recently-used (LRU) and many other deterministic algorithms with cache of size k could be worse than an adversary with cache of size h by a factor of $\frac{k}{k-h+1}$, but not more, and that no deterministic on-line algorithm could achieve a factor less than $\frac{k}{k-h+1}$. Karlin, Manasse, Rudolph and Sleator [123] introduced the term "competitive" and proposed a k-competitive deterministic paging strategy, called flush-when-full (FWF) against an adversary with cache of size k.

Randomized paging was first studied by Fiat, Karp, Luby, McGeoch, Sleator and Young [91]. They showed that their Randomized Marking Algorithm achieves a competitive ratio no worse than $2H_k$ (against an oblivious adversary with cache of size k), where H_k represents the kth Harmonic number. In addition, they proved that no randomized paging algorithm can have a competitive ratio less than H_k. McGeoch and Sleator [142] gave an optimal randomized algorithm that achieves a

competitive ratio of H_k against an oblivious adversary. Later, Achlioptas, Chrobak and Noga [14] proposed a much simpler optimal randomized algorithm achieving a competitive ratio of H_k and also showed that the Randomized Marking Algorithm [91] has the exact competitive ratio of $2H_k - 1$.

From a point of view of web caching, Gopalan, Karloff, Mehta, Mihail and Vishnoi [94] studied the paging problem assuming that pages have expiration times beyond which they are no longer valid. They showed that LRU with minimal adaptations is asymptotically optimal.

The above Sleator-Tarjan's way of analysis is, however, too coarse to make a distinction between LRU and FIFO, whereas in practice LRU is almost always superior to FIFO. To address this fact, Borodin, Irani, Raghavan and Schieber [49] developed a graph-theoretic model of a program's locality reference, called the *access graph*, and proved that the competitive ratio of LRU is at most twice that of FIFO for any access graph, and LRU and FIFO are far from optimal competitiveness for some graphs. They also gave an deterministic algorithm FAR that achieves a competitive ratio within a factor of $O(\log k)$ of that of an optimal on-line algorithm for any k and access graph. Actually, FAR was proved to be strongly competitive by Irani, Karlin and Phillips [114], i.e., it achieves a competitive ratio within a constant factor of that of an optimal on-line algorithm, for any k and access graph. Fiat and Karlin [90] gave a strongly competitive randomized algorithm for any access graph against an oblivious adversary. They also gave another strongly competitive deterministic algorithm against adversaries that can use multiple pointers into the undirected access graph to generate the request sequence; this models the reasonable situation where programs often use multiple data structures. Chrobak and Noga [66] proved that LRU has the competitive ratio that is less than or equal to that of FIFO under any access graph, which had been conjectured by Borodin et al.. The access graph model assumes a prior knowledge of the underlying access graph. There are models that require weaker assumptions. Fiat and Rosen [92] introduced the dynamic access graph and examined LRU and several heuristics by simulations. Karlin, Phillips and Raghavan [125] proposed the paging problem under the assumption that the sequence of pages accessed is generated by a Markov chain, and provided a mathematical basis for the poor performance in simulations of certain paging algorithms such as the random placement algorithm, which randomly chooses a file to be evicted, and the frequency count algorithm, which evicts the file that is least frequently accessed. Koutsoupias and Papadimitriou [130] proposed a more general model, called the *diffuse adversary model*, where a request sequence is generated by a probability distribution that is chosen from a known class of distributions. Young [200] defined a new notion of loose competitiveness against an adversary with cache of size h ($\leq k$), which ignores input sequences giving a high competitive ratio for only a few values of h. Lund, Phillips and Reingold [137] discusses caching

strategies in the context of virtual circuit management, in which any particular host may have only a fixed number k of virtual circuits open and it must choose a virtual circuit to close, before opening a circuit to the destination of the packet that has arrived. When the interarrival time of the packets exchanged between each pair of hosts has a fixed and independent distribution, their first algorithm has the competitive ratio that is at most 5 times that of the best on-line algorithm. They also discusses a more general model in which, for each pair p, q of pages in the cache, on-line algorithms can only determine the probability that p will next be accessed before q; for that model, they gave a randomized on-line algorithm with the competitive ratio that is at most four times that of the best on-line algorithm. Recently, Albers, Favrholdt and Giel [21] proposed a simple model for the paging with locality of reference based on Denning's working set concept [81, 82].

A generalized problem of the paging problem is called the weighted caching problem, in which the size of each file is still uniform but the retrieval cost of each file is arbitrary. Chrobak, Karloff, Payne and Vishwanathan [65] showed that a simple algorithm (called "Balance" algorithm) is k-competitive against an adversary with cache of size k. They also gave an optimal off-line polynomial time algorithm for the problem. Subsequently, Young [200] proved that his Greedy-Dual algorithm with cache of size k is $\frac{k}{k-h+1}$-competitive against an adversary with cache of size h, which implies that Greedy-Dual is optimal. The Greedy-Dual algorithm generalizes many well-known paging and weighted paging strategies, including least-recently-used (LRU), first-in-first-out (FIFO), flush-when-full (FWF), and the Balance algorithm. Raghavan and Snir [165] gave a randomized algorithm (called "RECIPROCAL"[2]) that has competitive ratio $\frac{k}{k-h+1}$ against an oblivious/adaptive-online adversary with cache of size h. Although this algorithm does not defeat the deterministic algorithm in terms of competitiveness, it does not make use of any information other than which pages are stored, i.e., memoryless. They also discussed the tradeoff between memory and randomness. This simplicity realized by randomness is effective when available resource is limited or time complexity per request is critical. In the special case where pages have one of two fixed weights, Irani [113] developed an $O(\log k)$-competitive algorithm against an oblivious adversary with cache of size k by nesting the Randomized Marking Algorithm. However, no randomized on-line algorithms with competitive ratio less than k against an adversary with cache of size k are known in the general case.

The file caching problem is further generalized form of the weighted caching problem such that the size of each file (page), in addition to retrieval cost (or file cost), is arbitrary. Irani [112] investigated the two special cases where file size is arbitrary while file cost is uniform (the fault model), and where file cost equals

[2] In Ref. [165], this algorithm is called HARMONIC, but Ref. [149] calls it RECIPROCAL.

file size (the bit model), and proved that LRU is optimally competitive in both models. Although the paging problem is inherently on-line, on-line algorithms for the problem often attempt to imitate the behavior of an optimal off-line algorithm with partial information on the request sequence; LRU can be regarded as an on-line version of the optimal offline polynomial-time algorithm shown in [38]. However, the off-line file caching problem is $\mathcal{NP}$-hard in the bit model, and hence in the general model, as proved in this section (it is unknown whether the problem for the fault model is in $\mathcal{P}$). Irani [112] thus presented polynomial time off-line algorithms that achieve approximation ratio of $O(\log K)$ for the fault and bit models, where $K = k/D_{\min}$ and $D_{\min}$ is the size of the smallest file among the files that could be requested. She also gave $O(\log^2 k)$-competitive randomized on-line algorithms in both models against oblivious adversaries.

As for the general model (arbitrary file sizes and costs), Cao and Irani [63] proposed the a k-competitive algorithm, called Greedy-Dual-Size, against an adversary with cache of size k by generalizing Greedy-Dual algorithm [200]. Young [202] gave the result that Greedy-Dual-Size[3] is $\frac{k}{k-h+1}$-competitive against an adversary with cache of size h. Albers, Arora and Khanna [19] developed a polynomial-time off-line approximation algorithm with cache size k for the general model. The algorithm achieves constant factor approximation ratio against the optimal off-line algorithm with cache of size $k + O(S)$, for the largest file size S among the sizes of the files that could be requested. They also gave another polynomial-time off-line approximation algorithm with cache size k for the general model with approximation ratio $O(\log(k + C))$ against the optimal off-line algorithm with cache of size k, for the largest file cost C among the costs of the files that could be requested. They also presented a randomized on-line algorithm with cache of size $(1 + c)k$ for the bit model that achieves competitive ratio $O(\ln(1+1/c))$ against an oblivious adversary with cache of size k. Bar-Noy, Bar-Yehuda, Freund, Naor and Schieber [33] presented a 4-approximation off-line algorithm for the general model that does not use extra space.

There are also a lot of empirical studies of the file caching problem. In the bit model, by using trace-driven simulations, Abrams, Standridge, Abdulla, Williams and Fox [13] showed that the SIZE algorithm, which simply evicts files in non-increasing order of their size, is often better for web requests than the most popular algorithm, LRU. However, SIZE is highly dependent on the correlation between file size and request frequency. To overcome this non-robustness, they also proposed LOG(SIZE) and LRU-MIN as mixtures of SIZE and LRU. Subsequently, Aggarwal, Wolf and Yu [16] developed an algorithm called SLRU, which can also be regarded as a mixture of SIZE and LRU, by applying an approximation algorithm for the KNAPSACK problem [144] to the set of stored files and the

[3] Actually, this algorithm is called "LANDLORD" in [202]

currently requested every time a fault occurs. Unfortunately, the above mixtures have no performance guarantee.

For the general model, Greedy-Dual-Size (GDS) also achieves excellent performance in simulations [63], and is adopted as an optional caching strategy of a major Web proxy server software, Squid [1]. Cao and Irani [63] showed an implementation of GDS that incurs $O(\log k)$ time per insertion or deletion in the worst case. Their implementation, however, uses an "inflation" offset variable in order to prevent the k subtractions that could occur when a file is evicted. If the value of the variable is not allowed to inflate infinitely (for theoretical or practical reasons), k subtractions occur when the value of the variable attains its limit. This pushes up the worst-case time complexity to $O(k)$, although amortized time per insertion or eviction may still be $O(\log k)$.

Podlipnig and Boszörömenyi [160] comprehensively surveyed numerous heuristics for the file caching problem by categorizing them in terms of kinds of parameters the heuristics are based on.

3 Streaming-Type Multi-Point Communication – IP-Unicast-Based Multicast (Flexcast) –

This section discusses an efficient way of streaming type multi-point communication: *IP-unicast-based multicast*.

3.1 The model

The Internet protocols span the complete range of the layers of the OSI (Open Systems Interconnection) basic reference model defined by ISO (the International Organization for Standardization).

The Internet Protocol (IP) [163] is the primary network-layer (Layer 3) protocol of the Internet that involves addressing information and some control information that enables packets to be routed. IP provides connection-less and best-effort delivery of datagrams through an internetwork, and realizes fragmentation and reassembly of datagrams to support data links with different maximum-transmission unit (MTU) sizes.

The Transmission Control Protocol (TCP) [164] and the User Datagram Protocol (UDP) [161] are the two central protocols of the transport layer (Layer 4), and work on the IP environment. TCP is a connection-oriented protocol and provides reliable transmission of data and flow control for each connection. UDP is a connection-less protocol and provides no reliability and flow control. This simplicity of UDP imposes less load on end hosts than TCP; UDP is useful in the situations where the reliability mechanisms and/or flow control of TCP are not appropriate to a higher-layer protocol.

In the following, our protocol is assumed to be realized on UDP (while it can also be realized on TCP): the packets used in our protocol are all UDP datagrams, and the source and destination addresses of the packets are those written in the IP headers while the source and destination ports of the packets are those written in the UDP headers. To avoid confusion, we use “sender” to indicate the source of a multicast tree.

3.2 Flexcast protocol

We introduce here a new IP-unicast-based multicast protocol, called “Flexcast,” which dynamically constructs a multicast tree for every pair (S, G) of a sender S and a group identifier G that is unique within S, by sharing common segments among unicast (reverse) paths from receivers to S. The communication realized by the tree still appears to be one-to-one for each sender and receiver. Our tree construction is based on a hierarchical keep-alive mechanism, i.e., join packets are periodically sent destined to sender S. The mechanism dynamically reconstructs

the tree so that it can prevent particular Flexcast nodes from being overloaded and can optimize the tree against changes in IP-unicast routing paths (here we denote by a *Flexcast node*, a node that can process the Flexcast packets and also work as an ordinary router). Furthermore, this dynamic reconstruction naturally supports the mobility of senders and receivers.

We will present the basic operation of Flexcast and then describe its functions of load balancing and dynamic reconstruction. Next we consider how to implement the keep-alive mechanism used in the protocol.

3.2.1 Basic operation

Receivers send unicast packets, called *join packets*, containing the paired information (S, G) of sender address S and group identifier G that is unique within sender S, destined to a certain fixed port of S; the port should be reserved for Flexcast in advance. In the following, pair (S, G) is called a "channel." Legacy nodes (i.e., non-Flexcast nodes that route unicast packets in an ordinary way) on the unicast routing path between Flexcast nodes simply forward the Flexcast packets (i.e., join packets and delivery packets, introduced later) according to their unicast routing tables, since the Flexcast packets are just unicast packets. Suppose that a join packet for (S, G) arrives at a Flexcast node which has no entry for (S, G) in its Flexcast routing table, called *delivery table*. This table determines to which children the multicast data is to be delivered. The Flexcast node recognizes the join packet by its destination port, and registers its source address as a child on the tree for channel (S, G) in the delivery table. The node then changes the source address of the packet to itself and forwards it according to its unicast routing table. Actually, the join packets to be forwarded should be aggregated in an appropriate way so that the Flexcast node may not send join packets too often (if the node forwards every received join packet, sender S should eventually receive too many join packets in a short period). A heuristics to aggregate join packets will be described later. This joining operation propagates from the receiver through intermediate Flexcast nodes until the join packet reaches sender S or a Flexcast node that already has some entries for (S, G) in its delivery table. This operation is performed by a hierarchical "keep-alive mechanism." Each receiver periodically sends join packets as long as it wants to receive the multicast data. The parent Flexcast node of the receiver continues to send the multicast data while the join packets from the receiver continuously arrive within some predefined interval. In other words, if a receiver stops sending join packets, the entry of the receiver will soon expire in the delivery table of its parent, and the parent will then stop delivering the multicast data to the receiver. This keep-alive mechanism works between every pair of a child and its parent. In this way, the keep-alive mechanism starts from receivers and passes through each parent and child pair. Multicast data is

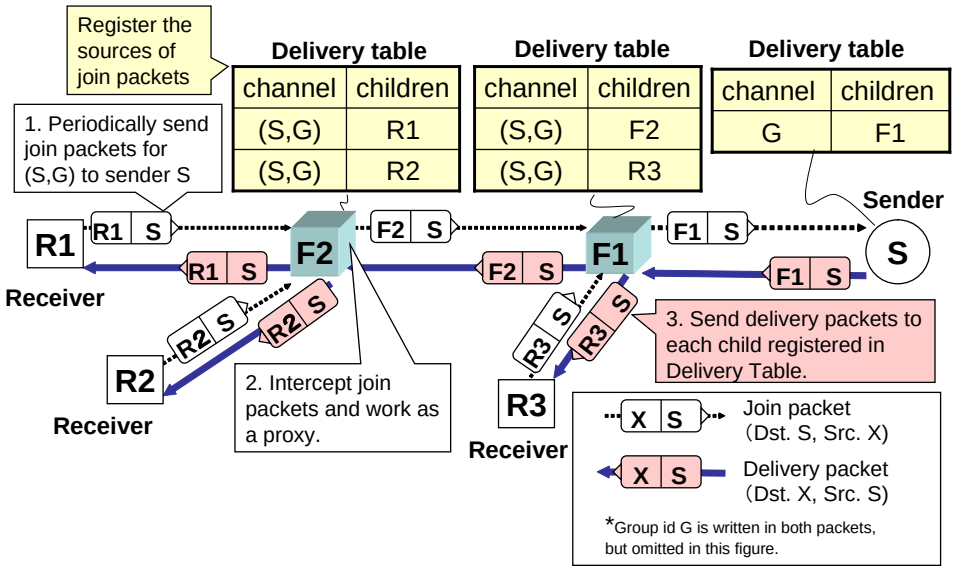

Figure 20: Basic operation of the Flexcast protocol.

carried by another kind of unicast packets, called *delivery packets*, each of which contains channel (S, G) and is destined to a fixed port; the port should be assigned for Flexcast. Every Flexcast node forwards every received delivery packet to each child according to its delivery table. For efficiency, we can optionally introduce *prune packets*, that are sent by receivers or Flexcast nodes to inform their parents that they want to leave the multicast tree. In what follows, we do not assume prune packets for simplicity.

An example of the above operation is shown in Figure 20. Receivers $R1$, $R2$, and $R3$ periodically send join packets for channel (S, G) destined to sender S. Group id G is contained in both join packets and delivery packets, but we omit G for simplicity in the figure. Suppose that Flexcast node $F2$ lies on both of the unicast routing paths from $R1$ and $R2$ to S. Flexcast node $F2$ then intercepts the join packets and registers their source addresses, i.e., $R1$ and $R2$, in its delivery table, and forwards the join packets after setting their source addresses to $F2$, i.e., $F2$ works as a proxy. If Flexcast node $F1$ lies on both of the unicast routing paths from $F2$ and $R3$ to S, $F1$ picks up the join packets from $F2$ and $R3$ and registers $F2$ and $R3$ in its delivery table, and forwards the join packets that identify $F1$ as their source addresses. When S receives the join packets, it registers $F1$ as a child and transmits delivery packets to $F1$. Flexcast node $F1$ copies the delivery packets and sends them to $F2$ and $R3$ by referring to its delivery table. In the same way, $F2$ sends the delivery packets to $R1$ and $R2$. Figures 21 and 22 precisely describe the behavior of a Flexcast node when it receives a join packet and a

Join Packet Processing Algorithm

1. Wait until receiving a join packet.
   Let J be the received join packet.
   Assume that J is for channel (S, G), and has source address s and destination address S.

2. Register tuple (S, G, s, t) in the delivery table, where t is the time when the join packet J was received (if there is already $(S, G, s, *)$ in the delivery table, just overwrite (S, G, s, t)).

3. if $(T_{\rm now} - T_{\rm last}(S, G)) \geq D$,
   set $T_{\rm last}(S, G)$ to $T_{\rm now}$ and
   send a join packet for (S, G) with source address F and destination address S,
   where $T_{\rm now}$ is the current time and $T_{\rm last}(S, G)$ is the time when F sent the last join packet for (S, G).

Figure 21: The algorithm that processes join packets at Flexcast node F, where D is a predefined interval.

delivery packet, respectively. The algorithm for processing join packets at sender S is just performing steps 1 and 2 in Figure 21. Sender S transmits delivery packets in the same way as step 2 of Figure 22.

From the above, it is clear that delivery traces the reverse of the unicast routing path from every receiver to sender S. Intuitively, the reverse paths between receivers and sender S are agglomerated as much as possible by the Flexcast nodes on the paths. Note that the Flexcast protocol can work even if the reverse paths differ from the unicast routing paths (i.e., the forward paths) from sender S to the receivers. While forward-paths generally yield higher stream delivery quality than reverse-paths, forward-path-based tree construction often results in complicated, non-adaptable or unstable protocols. Since IP networks are being optimized to support peer-to-peer communication, it would be reasonable in many cases to assume that there is little difference in quality between forward- and reverse-paths. Thus the Flexcast protocol uses the reverse-path-based tree; this yields scalability in terms of the number of nodes and significant adaptability to IP-route changes.

3.2.2 Load balancing

Consider the tree in Figure 23. The tree consists of sender S (the topmost node), three Flexcast nodes (numbered $F8$ to $F10$) and six receivers (numbered $R1$ to

Delivery Packet Processing Algorithm

1. Wait until receiving a delivery packet.
   Let K be the received delivery packet.
   Assume that K is for channel (S, G), and has source address S and destination address F.

2. For every entry (S, G, s, t) in the delivery table, forward delivery packet K after changing its destination address to s if $(T_{\text{now}} - t) \leq X$, where T_{now} is the current time; otherwise remove entry (S, G, s, t) from the delivery table.

Figure 22: The algorithm that processes delivery packets at Flexcast node F, where X is the expiration interval determined by an appropriate heuristics (one of such heuristics is described in 3.2.4).

$R6$). A new receiver $R7$ now starts to send join packets with destination S. Here we assume that Flexcast nodes $F9$ and $F10$ can accommodate at most three and four children, respectively, due to the limit of their computational power. Hence $F9$ cannot register $R7$ in the delivery table, even if the first join packet from $R7$ arrives. $F9$ then routes the join packets from R_7 toward sender S without any change. When the first join packet from $R7$ arrives at $F10$, the source address $R7$ of the join packet is registered in the delivery table of $F10$. The resulting tree is shown in Figure 24. This is a natural extension of the basic protocol and it accommodates as many receivers as possible without over-burdening particular Flexcast nodes, at the cost of additional bandwidth. It is stressed that, when the path from $F10$ to $R7$ is congested, this optional function may affect the quality of the stream received by $R4$, $R5$ and $R6$.

3.2.3 Dynamic reconstruction of multicast trees

In Figure 24, we assume that $R5$ has stopped sending join packets. This causes $R5$ to expire at $F9$. Flexcast node $F9$ can now accommodate one more child. Thus, when the first join packet from $R7$ arrives at $F9$ after the expiration of $R5$, $R7$ is registered in the delivery table of $F9$ as shown in Figure 25. Note that $R7$ periodically sends join packets to stay alive. The join packets from $R7$ are no longer forwarded by $F9$ toward S. This leads to the expiration of $R7$ at $F10$. In this way, the keep-alive mechanism dynamically reconstructs the tree so as to minimize traffic.

This dynamic reconstruction is effective especially when receivers are mobile

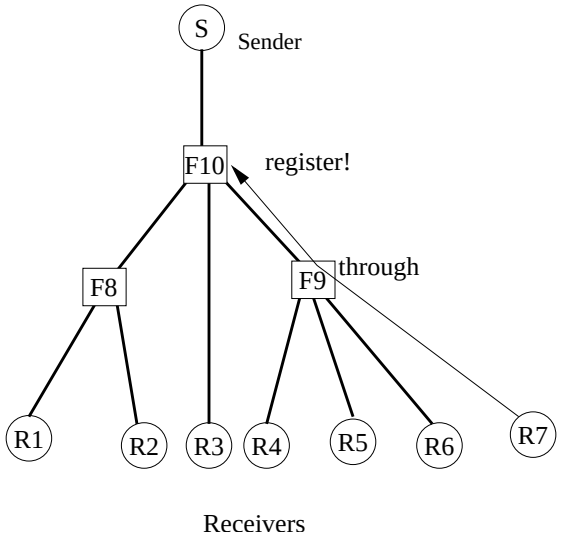

Figure 23: Receiver R7 joins when Flexcast node F9 cannot accommodate any more children.

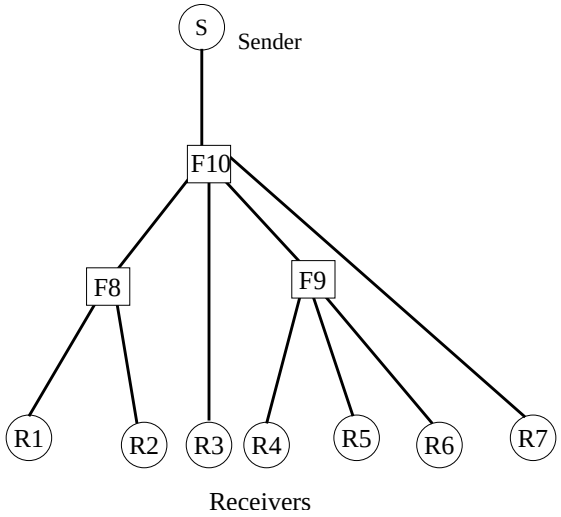

Figure 24: The multicast tree after receiver R7 joined.

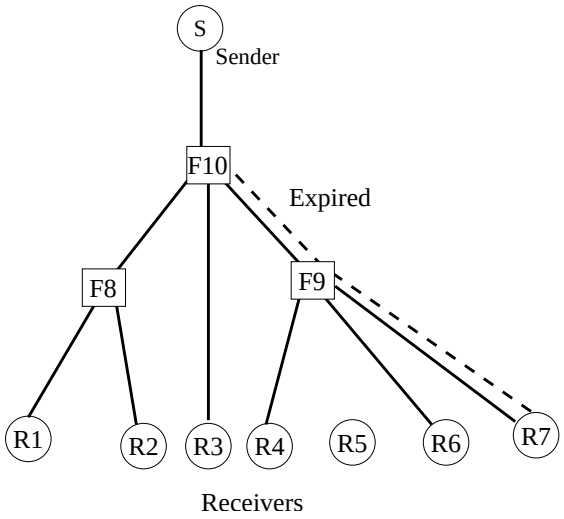

Figure 25: The multicast tree after receiver R5 has stopped sending join packets and expired.

hosts. In Figure 26, $R3$ moves and its neighbor changes from $F10$ to $F8$. Receiver $R3$ is then registered at $F8$ and expires at $F10$.

Another reconstruction is triggered by the movement of sender S as is possible when S is a mobile host such as a palm-top computer with a small video camera. If the network has a platform to support unicast to mobile hosts, say, the mobile-IP framework [159], the keep-alive mechanism reconstructs the tree as follows. As in Figure 27, sender S moves and its neighboring Flexcast node changes from $F10$ to $F11$. The join packets from receivers are routed for the new location of S by the mobile-IP framework, since the join packets are unicast packets with destination S. Here we assume that $F11$ comes to lie on the paths from $R3$, $F8$ and $F9$ to S, and that $F10$ is not on any of the paths yet. As a result, $R3$, $F8$ and $F9$ are registered at $F11$, and $F11$ sends toward S join packets with source address $F11$. Flexcast node $F10$ may also be registered at $F11$ when the join packets with source $F10$ destined to S arrive, if $F11$ is on the path from $F10$ to S. $R3$, $F8$ and $F9$ will expire at $F10$ after a moment since the join packets from them do not reach $F10$ any more. Flexcast node $F10$ will then stop sending join packets destined to S and expire at $F11$.

During tree reconstruction, multiple packets carrying the same data may arrive until the old parent-child relationships disappear. In Figure 25, $F9$ and $F10$ may send redundant packets to $R7$. In Figure 27, $F10$ and $F11$ may send the redundant packets to $R3$, $F8$ and $F9$. However, these packets can be discarded at receivers or Flexcast nodes by checking the time-stamp and/or sequence number assigned by sender S according to, say, RTP (Real-Time Transport Protocol) [171].

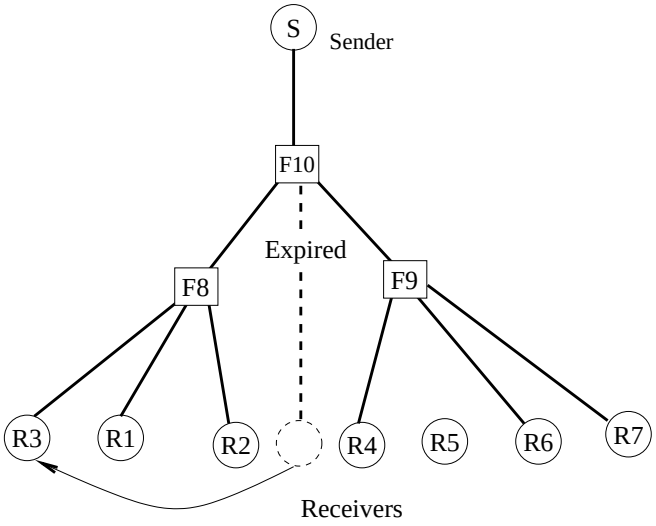

Figure 26: Multicast tree reconstruction when receiver R3 moves.

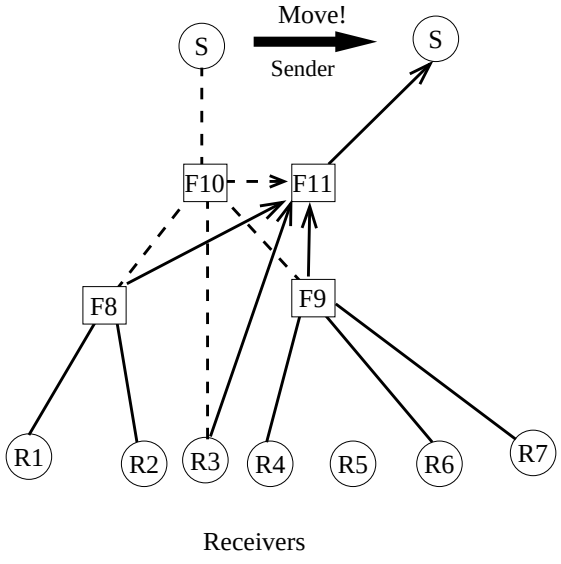

Figure 27: Multicast tree reconstruction when the sender moves.

3.2.4 Expiration time

The keep-alive mechanism must not allow any node to expire at its parent node when the node has at least one child. A simple way of accomplishing this is to use a timer at each Flexcast node in order to ensure that the node periodically sends join packets toward the sender as long as it has at least one child. However, this may so badly load resources that it may delay other tasks like data packet delivery.

Another way of implementing the keep-alive mechanism is to allow each Flexcast node to transmit a join packet toward the sender only when the node receives a join packet from a child. If, however, each Flexcast node sends a join packet every time it receives a join packet, the sender may receive more join packets than it can process.

We now consider when a Flexcast node should send a join packet and how to set the expiration time interval. Assume that all receivers send join packets periodically at interval D. Each Flexcast node has variable T_{last}, which is the time when the node sent the last join packet. When the node receives a join packet, it compares the current time T_{now} with T_{last}. If $(T_{\text{now}} - T_{\text{last}}) \geq D$, it sends a join packet destined to the sender.

Before analyzing the above algorithm, we define the height of each Flexcast node as the maximum length (i.e., the number of communication links) of the unicast routing path from any descendant receiver of the node. Let $D_s(h)$ and $D_r(h)$ be the maximum intervals between consecutive join packets being sent and received, respectively, at a Flexcast node of height h.

If we assume that there is no jitter of link and processing delays, i.e., $D_r(h) = D_s(h-1)$, we can prove by induction that

$$D_r(h) \leq h \times D,$$

as follows. Suppose that a Flexcast node of height $h-1$ receives join packets from all children at once just before time $D_r(h-1)$ has passed since the last time the node sent a join packet. At this time the node does not send any join packet since $(T_{now} - T_{last}) < D$. Obviously, the node will not receive any join packet before time D passes from now. This case maximizes $D_s(h-1)$. Thus, we have $D_r(h) = D_s(h-1) \leq D + D_r(h-1)$. Clearly, $D_r(1) = D$.

If we cannot ignore the jitter of link delay and processing delay, we can evaluate $D_r(h)$ by slightly modifying the above. Let ϵ_1 and ϵ_2 be the maximum differences between the maximum and minimum values of link delay and processing delay, respectively. $D_r(h) \leq D_s(h-1) + \epsilon_1$ and $D_s(h) \leq D + D_r(h) + \epsilon_2$. Thus, we have

$$D_r(h) \leq h \times (D + \epsilon_1 + \epsilon_2),$$

since $D_r(1) \leq D + \epsilon_1 + \epsilon_2$.

In a reliable network in which packets are seldom lost, a straightforward way of setting the expiration time interval is to set it to $D_r(h^*)$ for all nodes, where h^* is the height of the sender. However, $D_r(h^*)$ is too large for nodes whose height is less than h^*. This may result in consuming bandwidth wastefully, since a Flexcast node does not stop sending delivery packets to its children until they expire. A solution to this problem is to make each node dynamically set the expiration time for every child of height i to $D_s(i)$. The node can know the height of its children by using join packets as follows. Receivers write height 0 in the join packets they send. Flexcast nodes choose the maximum value among the heights written in join packets sent by their children and write the value plus 1 in the join packet they send. Clearly, the height of a child is equal to the value written in the join packets sent by the child. This dynamic way of setting the expiration time interval ensures that the keep-alive-based tree construction is scalable. The above procedure can also be applied to computing other information such as the number of receivers and packet loss rate [183].

3.3 Security issue and modifications to the Flexcast protocol

The protocol we described assumes that receivers know the correct sender address. If a receiver sends join packets destined to an incorrect sender address, the entries made in the delivery tables at Flexcast nodes are never used, but are kept until the receiver stops sending the join packets. This situation could occur when a user types an incorrect address in the sender address field of his receiver application software.

An easy way to overcome this problem is to introduce a new packet by which the receiver confirms the existence of the sender: the receiver transmits the packet destined to the sender, and, only when receiving an acknowledgment packet from the sender or the nearest Flexcast node that is already a member of the multicast tree, starts to send join packets destined to the sender. However, if we assume malicious receivers, they can still easily consume the resource of Flexcast nodes by deliberately transmitting join packets destined to incorrect addresses so that their source addresses can be registered in the delivery tables of the nodes. In what follows, we will propose two modifications to the Flexcast protocol. The key idea of these modifications is to allow the state (i.e., the content of the delivery table) of any Flexcast node to be changed only when both a packet from a receiver and a packet from the sender arrive at the node within a prespecified interval. This makes it difficult for every end host to change the state of the Flexcast node without collaboration.

The modified protocols still have the same advantages as the basic protocol, since they use only IP-unicast packets and adopt the hierarchical keep-alive mechanism. Furthermore, they can work even if reverse-paths are different from

forward-paths.

3.3.1 First Modification

The outline of the modification is as follows. Initially, no Flexcast nodes are permitted to register the sources of join packets. For each channel, permission for registration is given by special packets, called *permission packets*, which are sent by sender S (or the Flexcast nodes F which already have the permission), when, from the information carried by a received join packet, S (or F) detects that there is at least one Flexcast node that does not have the permission on the path from the source of the join packet to S (or F).

Concretely, every join packet includes a list of the Flexcast nodes through which the join packet passed and that do not have the permission. We call this list a *host list*. The host list will be updated each time the join packet passes through a Flexcast node that does not have the permission. Consider a join packet for channel (S, G) that arrives at a Flexcast node (or the sender) F. If F already has at least one entry for (S, G) in the delivery table and the host list is empty, the source is registered in the delivery table. This situation implies that there are no Flexcast nodes that do not have the permission on the path from the source of the join packet to F. If F is not the sender, it forwards the join packet toward the sender after setting the source address to F. If F has an entry for (S, G) in the delivery table, but the host list is NOT empty, there is at least one Flexcast node F' that does not have the permission on the path from the source to F. In this case, F sends a permission packet to the last host included in the host list carried by the received join packet; the permission packet conveys the host list obtained by removing the last host from the host list. The permission packet gives F' permission to register the source of join packets for (S, G). The permission packet is routed according to the host list it conveys, and gives the permission to all hosts included in the host list. It is stressed that, the packet gives the permission to all the Flexcast nodes that do not have the permission on the path from the source of the join packet to F, even if unicast routing paths are asymmetric. If F' receives a join packet for (S, G) after obtaining the permission, F' processes the packet in the same way as the basic Flexcast protocol: F' registers the source of the packet, and forwards the join packet toward sender S after setting the source to F'. Figures 28 and 29 give the precise descriptions of the algorithms that process join packets and permission packets, respectively. Delivery packets are processed in the same way as the basic Flexcast protocol.

It is stressed that every permission packet affects no Flexcast nodes except those included in the host list conveyed by the join packet that triggers the permission packet. It follows that malicious receivers cannot affect any Flexcast nodes by sending join packets to randomly chosen addresses, and also, malicious senders

Join Packet Processing Algorithm of the First Modification

1. Wait until receiving a join packet. Let J be the received packet.
   Assume that J is for channel (S, G), and has source address s, destination address S, and host list Q.
2. If F is already permitted to make entries for (S, G) in the delivery table and Q is empty, perform the next steps and go back to step 1.
   (a) Register (S, G, s, t) in the delivery table, where t is the time when join packet J was received (if there is already $(S, G, s, *)$ in the delivery table, just overwrite (S, G, s, t)).
   (b) If $(T_{\rm now} - T_{\rm last}(S, G)) \geq D$, set $T_{\rm last}(S, G)$ to $T_{\rm now}$ and send a join packet for (S, G) having source address F, destination address S, and an empty host list,
   where $T_{\rm now}$ is the current time, $T_{\rm last}(S, G)$ is the time when F sent the last join packet for (S, G).
3. If F is already permitted to make entries for (S, G) in the delivery table and Q is NOT empty, perform the next steps and go back to step 1.
   (a) Let F' be the last host in Q and remove the host from Q.
   (b) Transmit a permission packet for (S, G) that has source address F, destination address F', and host list Q.
4. Append F to the end of Q.
   Send a join packet for (S, G) having source address s, destination address S, and host list Q.

Figure 28: The algorithm that processes join packets of the first modification to the Flexcast protocol at Flexcast node F.

cannot affect any Flexcast nodes by using permission packets if they do not know exact addresses of the Flexcast nodes. The disadvantage of this solution is that the sizes of join packets and permission packets are proportional to the length of the path from a receiver to the sender or the nearest Flexcast node having the permission. This may increase the load of Flexcast nodes when they process join packets or permission packets, while this modification can work even if the host list is long since a standard implementation automatically divides a large UDP datagram into small IP packets.

By using Figure 30, we describe the behavior of the protocol in the situation where there are some Flexcast nodes having no permission on the path from receiver $R1$ to sender S. Suppose that receiver $R1$ starts to periodically send join packets that have the "empty" host lists (1. in the figure). The join packet sent by $R1$ goes through $F2$ and $F1$ and arrives at S. Sender S then sends a permission

Permission Packet Processing Algorithm of the First Modification

1. Wait until receiving a permission packet. Let P be the received permission packet.
   Assume that P is for (S, G), and has source address s, destination address F, and host list Q.
2. Give permission for (S, G) to F; the permission expires when time X has passed since the last join packet arrived.
3. If Q is empty, discard P and go back to step 1; othewise perform the next steps.
   (a) For the last host F' in Q, remove F' from Q.
   (b) Forward a permission packet for (S, G) that has destination F' and host list Q (the source address of the packet may optionally be changed from s to F).

Figure 29: The algorithm that processes permission packets of the first modification to the Flexcast protocol at Flexcast node F. X is the expiration interval determined by appropriate heuristics (one of such heuristics is described in 3.2.4).

packet for (S, G) to the last host in the host list conveyed by the join packet (2. in the figure). The permission packet carries the host list obtained by removing the last host from the host list extracted from the join packet. When this permission packet arrives at $F1$, it gives the permission to $F1$ (3. in the figure). Then $F1$ transmits a permission packet to the last host in the host list carried by the received permission packet. The permission packet sent by $F1$ includes the host list obtained by removing the last host from the host list extracted from the received permission packet, i.e., an "empty" host list in this case. When the permission packet reaches $F2$, it gives the permission to $F2$. If $F2$ receives a join packet for (S, G) after obtaining the permission, $F2$ registers the source of the packet (4. in the figure) and forwards the join packet destined to sender S after setting the source to $F2$. It is stressed that the host list carried by this packet is empty, which enables the source of the join packet to be registered at $F1$. These operations construct a multicast tree, and finally $R1$ receives delivery packets.

Figure 31 shows the situation where a new receiver $R2$ is joining the multicast tree that has already been constrcuted. $R2$ periodically sends join packets for (S, G) destined to S with an "empty" host list (7. in the figure). When $F2$ receives

the join packet, $F2$ concludes that there are no Flexcast nodes that do not have the permission on the path from the source of the join packet. Hence, $F2$ registers the source and starts to send delivery packets to $R2$.

3.3.2 Second Modification

Another modification is secure as much as the first modification, and uses only join packets and delivery packets. The length of join packets and delivery packets is independent of the topology of networks. A unique feature is that delivery packets as well as join packets contribute to constructing a multicast tree. As the first modification, initially, no Flexcast nodes are permitted to register the sources of join packets. For each channel (S, G), the permission for registration is given by delivery packets, which are sent by sender S (or Flexcast node F which already has the permission). When a join packet arrives at a Flexcast node F' that does not have the permission, the packet is forwarded after the source is set to F'. When the join packet finally arrives at Flexcast node F that has the permission, the source of the join packet is registered in the delivery table. Everytime a delivery packet reaches this node from its parent, a copy of the packet is sent to each child by referring to the delivery table. When the delivery packet arrives at F', the packet gives the permission to F' since F' does not have the permission yet, and the packet is then discarded. When a join packet arrives at F' after that, its source address is registered in the delivery table. By repeating this procedure, a multicast tree is constructed. More precisely, Figures 32 and 33 describe the algorithms that process join packets and delivery packets, respectively, at Flexcast node F.

Figure 34 shows an example of the behavior of the second modification. Suppose that receiver $R1$ sends a join packet for (S, G) destined to sender S (1. in the figure). The join packet is routed along $F2$ and $F1$, and reaches S (2. in the figure), since $F2$ and $F1$ do not have the permission for (S, G). It is stressed that, before $F1$ and $F2$ transmit the join packet, $F1$ and $F2$ change the source address of the packet to themselves, respectively. When the join packet reaches S, the source $F1$ of the packet is registered in the delivery table of S. Sender S then sends delivery packets for (S, G) to $F1$ (3. in the figure). When the delivery packets arrive at $F1$, it gives $F1$ the permission for (S, G) (4. in the figure). If a join packet arrives at $F1$ after $F1$ was given the permission, the source $F2$ of the join packet is registered in the delivery table at $F1$ (7. in the figure), which enables $F1$ to send delivery packets to $F2$ (8. in the figure). The delivery packets will give the permission for (S, G) to $F2$. In this way, receiver $R1$ can finally receive delivery packets. Figure 35 describes the situation where a join packet for channel (S, G) reaches a Flexcast node that already has the permission for (S, G). Suppose that receiver $R2$ sends a join packet for (S, G). When $F3$ receives the packet, $F3$ changes the source of the packet to itself and forwards it toward S (12.

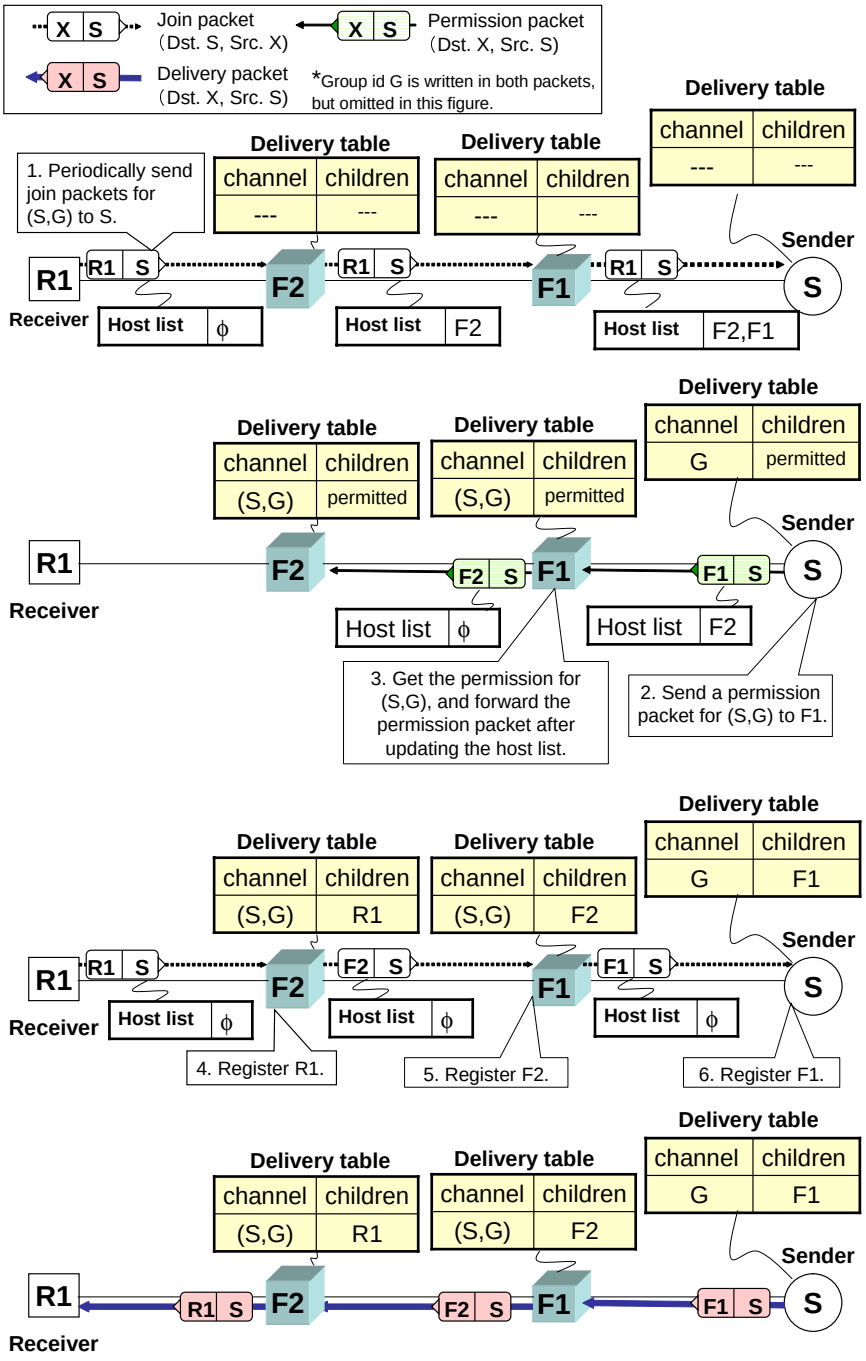

Figure 30: An example of the behavior of the first modification to the Flexcast protocol.

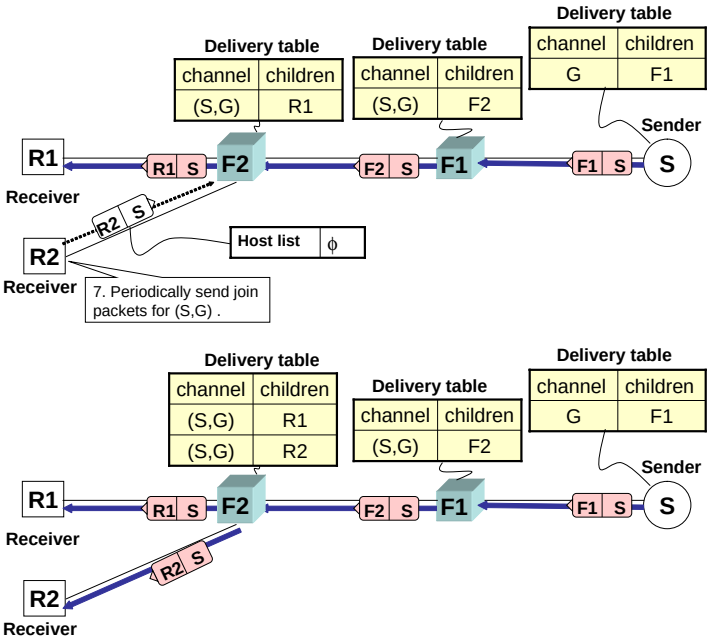

Figure 31: Another example of the behavior of the first modification.

in the figure). When $F2$ receives the packet, it registers its source (since it already has the permission) and starts to transmit delivery packets for (S, G) to $F3$, which will give the permission to $F3$ (13. in the figure). $F3$ then registers the source of a join packet that reaches $F3$ after $F3$ gets the permission (14. in the figure). Finally, $F3$ sends delivery packets to $R2$.

The only disadvantage of this protocol is that receiver R has to send at least m join packets in the worst case before receiving delivery packets, where m is the length of the path from R to sender S. This may cause a large delay. If $m = 20$ and the periodicity of sending join packets is one second, it takes at least 20 seconds for a receiver to receive the first delivery packet. This problem is solved by keeping the periodicity small until receiving the first delivery packet. This initial small periodicity can be variable depending on the length of the path from a receiver to the nearest Flexcast node having the permission, though the length has to be measured by an appropriate protocol.

The protocol uses only two type of packets: join and delivery packets. Hence the delay taken for receivers to start to receive delivery packets depends on the time interval between consecutive delivery packets. In many cases, this causes no problem, since the interval is usually sufficiently small. However, it is possible to prepare special packets to give the permission to Flexcast nodes.

Join Packet Processing Algorithm of the Second Modification

1. Wait until receiving a join packet. Let J be the received join packet. Assume that J is for channel (S, G) and has source address s and destination address S.

2. If there is at least one entry for (S, G) or F is already permitted to register an entry of (S, G) in the delivery table, perform the next steps.

   (a) Register (S, G, s, t) in the delivery table, where t is the time when the join packet J was received (if there is already $(S, G, s, *)$ in the delivery table, just overwrite (S, G, s, t)).

   (b) if $(T_{\text{now}} - T_{\text{last}}(S, G)) \geq D$,
   set $T_{\text{last}}(S, G)$ to T_{now} and
   send a join packet for (S, G) having source address F and destination address S,
   where T_{now} is the current time, and $T_{\text{last}}(S, G)$ is the time when F sent the last join packet for (S, G).

   Otherwise send a join packet for (S, G) having source address F and destination address S.

Figure 32: The algorithm that processes join packets of the second modification to the Flexcast protocol at Flexcast node F.

3.4 Tunneling function

Senders and receivers need to be able to run the Flexcast protocol, while there are many commercial/free application softwares for IP multicast senders/receivers. Hence it would be useful to provide a system that enables the users of IP multicast application softwares to enjoy the benefit of Flexcast. In what follows, we describe a Flexcast-IP-multicast bidirectional translator, called a Flexcast gateway. An example of Flexcast gateway use is shown in Figure 36, where S and R are Flexcast gateways. They are on the segments connected to IP multicast sender X and IP multicast receivers, respectively. Let Z be one of such IP multicast receivers. From the viewpoint of Flexcast, sender-side gateway S and receiver-side gate R are a Flexcast sender and a Flexcast receiver, respectively. For IP multicast senders/receivers, S and R provide a tunnel through which IP multicast packets from X pass to receiver Z. The tunnel is a tree-shaped one connecting S and multiple R's, since the tunnel itself is a multicast tree of Flexcast. This achieves

Delivery Packet Processing Algorithm of the Second Modification

1. Wait until receiving a delivery packet. Let K be the received delivery packet.
   Assume that K is for channel (S, G) and has source address S and destination address F.

2. If there is at least one entry of the form $(S, G, *, *)$ in the delivery table, perform the next step.

   - For every entry (S, G, s, t) for channel (S, G) in the delivery table, forward delivery packet K after changing its destination address to s if $(T_{\text{now}} - t) \leq X$, where T_{now} is the current time, otherwise remove entry (S, G, s, t) from the delivery table.

   Otherwise, give F the permission to register entries for (S, G) in the delivery table and discard K; the permission expires when time X has passed since the last join packet arrived.

Figure 33: The algorithm that processes delivery packets of the second modification to the Flexcast protocol at Flexcast node F, where X is the expiration interval determined by appropriate heuristics (one of such heuristics is described in 3.2.4).

remarkable efficiency, compared with ordinary unicast tunnels between IP multicast routers; the tunnels are usually configured manually for every pair of IP multicast routers. Here we assume that IP multicast senders and receivers speak the version 3 of Internet Group Management Protocol (IGMPv3) [62] (for IPv6, we can assume the version 2 of Multicast Listener Discovery (MLD) [188], since it has also a similar mechanism to IGMPv3). In what follows, we will describe the behavior of the "Flexcast tunneling."

Receiver-side Flexcast gateway R works as an ordianry IP multicast router on a local segment: R issues IGMP queries to periodically check if there are IP multicast receivers which want to receive IP multicast data. If R detects an IGMP membership report, it extracts sender address S and group id G (IP multicast address) from the report, and starts to send join packets of Flexcast for channel (S, G) destined to sender-side gateway S. When S receives join packets requesting channel (S, G), it picks up IP multicast packets for G that are passing through the local segment connected to S, encapsulates them in delivery packets of Flexcast, and sends the delivery packets to the children of S according to the Flexcast protocol. When R receives delivery packets for (S, G), it extracts the IP multi-

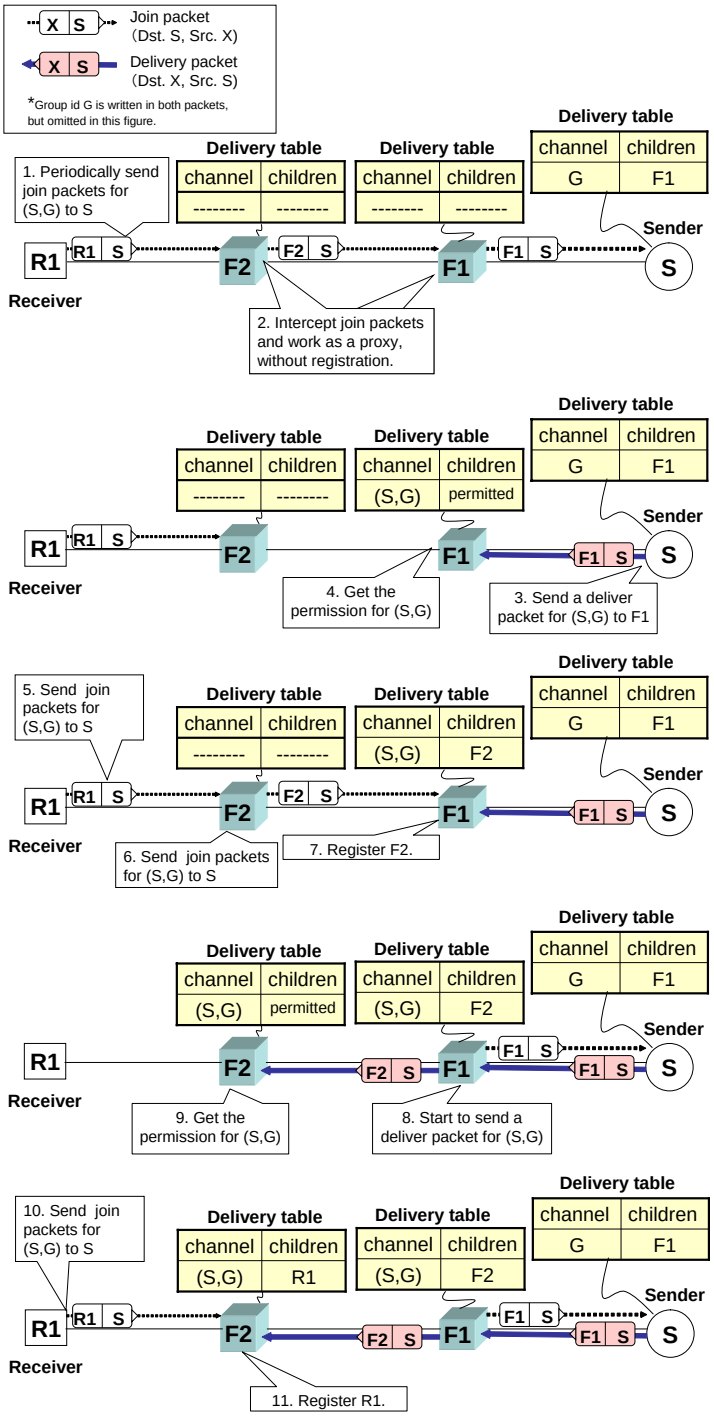

Figure 34: An example of the behavior of the second modification to the Flexcast protocol.

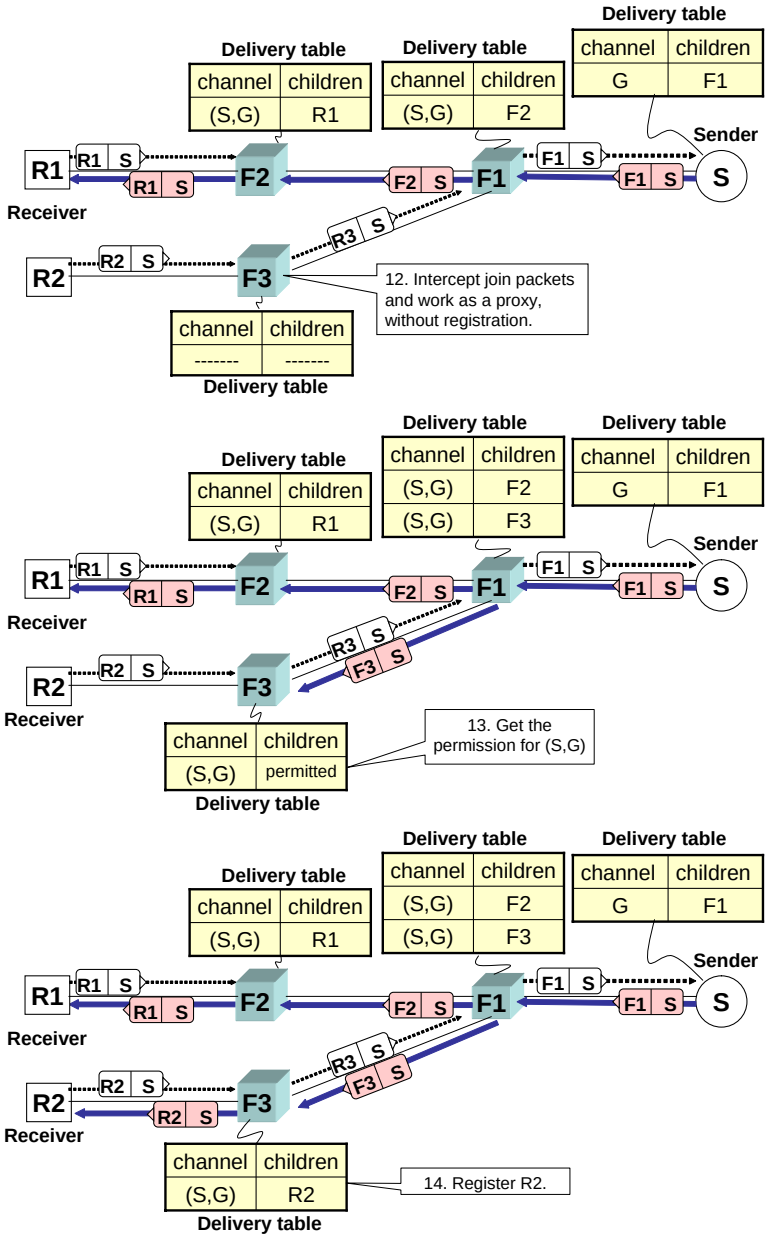

Figure 35: Another example of the behavior of the second modification to the Flexcast protocol.

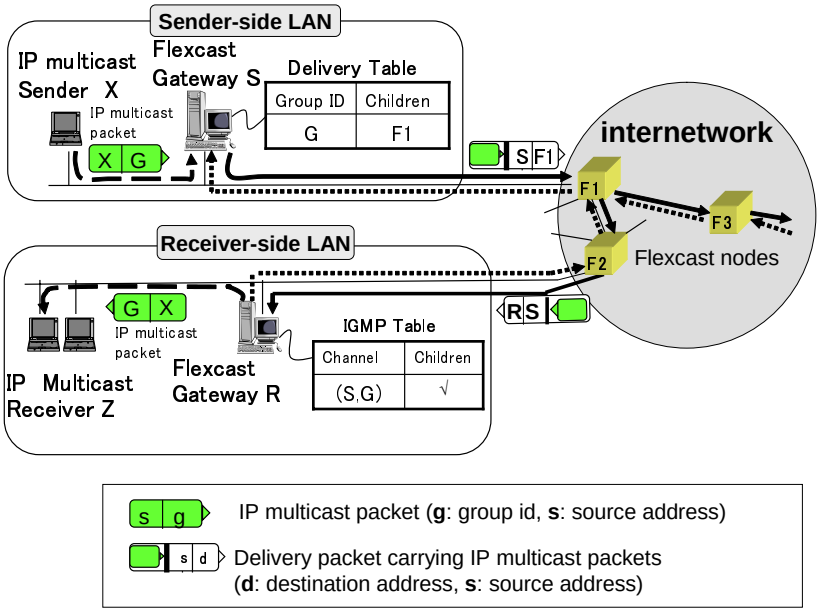

Figure 36: Flexcast tunneling of IP-multicast packets.

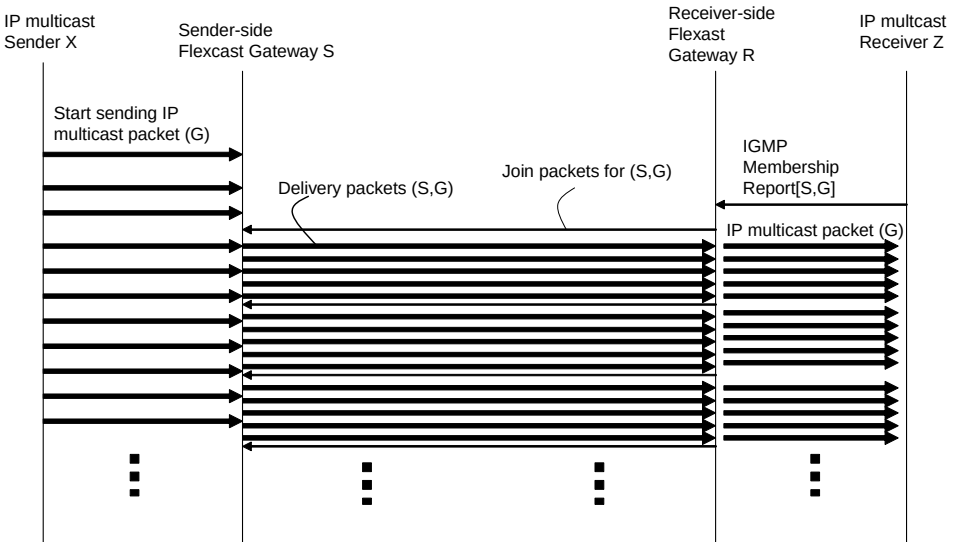

Figure 37: Protocol sequence of the Flexcast tunneling.

cast packets from the delivery packets and releases the IP multicast packets on the local segment it is connected to. Finally, IP multicast receiver Z can get the IP multicast packets. Figure 37 describes the outline of the protocol sequence of the Flexcast tunneling. It follows that the "Flexcast tunnels" are automatically created and removed under the control of IP multicast receivers.

When using the version 2 of Internet Group Management Protocol (IGMPv2) [89], gateway R cannot extract sender address S, since IGMPv2 membership reports contain no sender address. In this case, we need a system to resolve the sender address from the group id carried by the reports, which requires a special system to guarantee the global uniqueness of every group id. This issue reveals the limitation of the current IP multicast system based on IGMPv2.

3.5 Implementation of Flexcast nodes

Inoue et al. [109] proposed a scalable and fault-tolerant way of implementing a Flexcast node at low cost. Their implementation is based on the technology of cluster computing consisting of ordinary personal computers (PCs), which has been developed for a high performance search engine and GRID computing. They implemented a Flexcast node as a cluster of Linux PCs and examined them in experiments. The experimental results showed that forwarding performance and fault-tolerance are significantly enhanced by cluster computing. In what follows, we review their technology.

In a general cluster-based system, users' requests are distributed to each node in the cluster (*or* cluster node) by a front-end node, called a load-balancer, in a way of balancing the load of the cluster nodes. After cluster nodes process the requests, the answers to the request are relayed by the load-balancer to the users that issued them. When the load of distributing the requests is much heavier than that of just forwarding the answers, this approach is effective. However, if the answer causes quite heavy traffic such as moving picture stream, even just forwarding the answer might burden the load-balancer with heavy load. To decrease the load imposed on the load-balancer, a novel approach called the direct sender return was developed: the answers bypass the load-balancer and are directly returned to the users. Inoue et al. modified this approach for their cluster-based Flexcast node: while join packets are distributed by the load-balancer, delivery packets are directly delivered from/to the cluster nodes without passing through the load-balancer. The same technique can be applied to Flexcast gatways.

They implemented a Flexcast node on a set of PCs connected to a real router in use, as illustrated in Figure 38, where the router is called the *mother router*. Each of the PCs works as either a load-balancer or a cluster node. Hereafter, we call a cluster node *a splitter*, since the cluster node splits stream in this case.

Basically, the functions of a splitter are the same as those of the Flexcast node

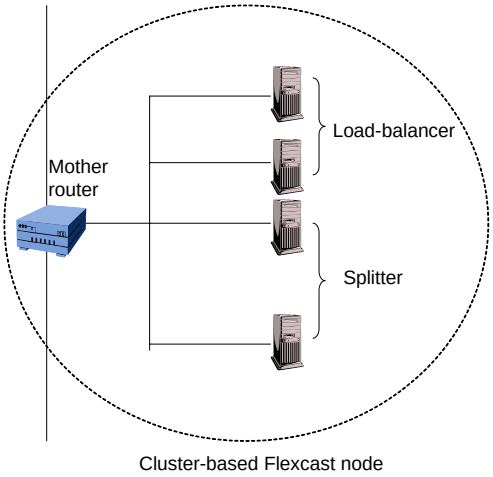

Figure 38: An implementation of a cluster-based Flexcast node [109].

except that it periodically sends *keepalive* packets to the load-balancer. The load-balancer maintains a *node table*, which includes active splitters and the channels assigned to each splitter. A splitter that stopped sending keepalive packets expires, and is removed from the node table. For high fault tolerancy, multiple load-balancers can be installed in a single cluster together with a redundancy protocol like Virtual Router Redundancy Protocol (VRRP) [98]. In what follows, we briefly introduce the behavior of the system when a single load-balancer is installed.

The mother router is configured to examine the port number of UDP packets passing through it, and intercepts the join packets of Flexcast. The intercepted join packet is forwarded to the load-balancer. The load-balancer consults the node table and finds the splitter assigned to the requested channel. If no splitter has been assigned to the channel, the load-balancer uses an arbitrary algorithm to select a splitter for the channel. Finally, the load-balancer forwards the packet to the assigned splitter without changing the source address of the packet. The splitter then registers the source address in its delivery table, and creates a new join packet to be sent destined to the sender, if necessary. In this way, join packets are distributed among the splitters in the cluster. Delivery packets from the parent Flexcast node are directly delivered to the appropriate splitters without passing through the load-balancer.

3.6 Experiments over the Internet

Experiments on stream delivery using the basic version of the Flexcast protocol were conducted among three widely dispersed locations with senders and receivers located at each site. We observed the response time of Flexcast, and the influence of communication delay jitter and the packet-loss rate, on the Flexcast operations. The results show that, in the absence of unusual congestion on any communication links, the Flexcast parameters such as the expiration time interval, offer sufficient margins, confirming the robustness and stability of the Flexcast protocol.

3.6.1 Environment

We conducted streaming experiments [181, 107, 108, 71] over Internet2 [110] and NTT's[4] experimental fiber-network connecting Japan and the U.S., called GEMnet. Since GEMnet uses ATM (Asynchronous Transfer Mode), the available bandwidth of one direction is independent of that of the other direction. Senders and receivers were located in NTT Yokosuka R&D Center (hereafter called Yokosuka), the University of Southern California (USC), Los Angeles, and the University of Illinois (UIC), Chicago. Yokosuka was connected to GEMnet; USC and UIC were connected to Internet2. Since GEMnet and Internet2 had a bidirectional connection point in Sunnyvale, U.S., USC and UIC could communicate with Yokosuka via Internet2 and GEMnet. GEMnet had a bottleneck link between Japan and the U.S.; the link had a constant bit rate (CBR) of 17 Mbps in each direction. This link was also the bottleneck of the entire network used in the experiments, since Internet2 offered 10-Gbps links while the local networks of USC and UIC had 1-Gbps links. The main purpose of the experiments was to verify that the Flexcast protocol could handle widely separated locations. The distances involved are: Yokosuka to Chicago: 12,000 km, Yokosuka to Los Angeles: 9,000 km, and Chicago to Los Angeles: 4,000 km. Figure 39 schematically shows the networks used. Each site had one or two Flexcast gateways on every local segment at the site. In this experiment, we did not place any Flexcast nodes between the gateways; the gateways directly communicated with each other. Each gateway had IP-multicast senders and receivers on the local segment connected to the gateway. Since there were no Flexcast nodes between the gateways, two streams of the same contents passed through GEMnet when a sender in Yokosuka serviced receivers in Los Angeles and Chicago. This was not a problem because the experiment was intended to verify the correct operation of the Flexcast protocol over a very-wide-area network, not to measure its scalability in terms of the number of receivers. We observed traffic at the three sites and checked the quality of moving

[4]Nippon Telegraph and Telephone Corporation (http://www.ntt.co.jp/).

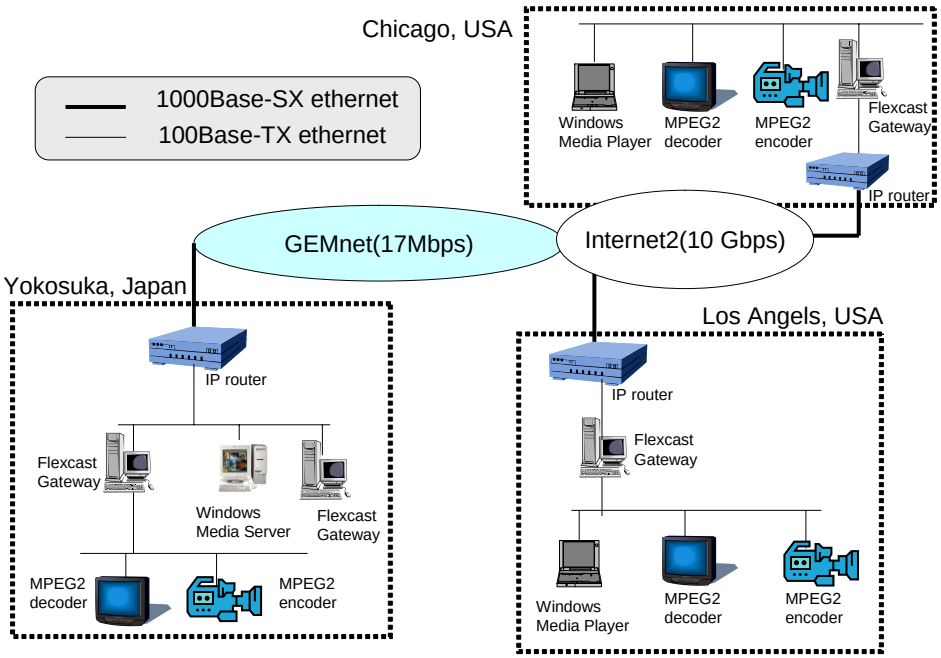

Figure 39: Schematic view of the network.

	CODEC type	Average bandwidth
System 1	MPEG2	7 Mbps
System 2	MS Windows Media ver. 7	625 Kbps

Table 4: Commercial IP-multicast sender/receiver systems used in the experiments.

pictures delivered by the Flexcast protocol.

3.6.2 Parameters and metrics

We used software-based Flexcast gateways implemented on personal computers (PCs) and two types of commercial IP-multicast sender/receiver systems as shown in Table 4. The average bandwidth of an MPEG2 stream was 7 Mbps and that of MS Windows Media version 7 was 625 Kbps. We measured the following metrics:

- *Response time*: the interval between the time when a receiver-side gateway sent the first join packet and the time when the gateway received the first delivery packet,

- *Join-packet sending-interval*: the time interval between two consecutive join packets sent by a sender, which was configured to be 1.0 second,

- *Join-packet arrival-interval*: the time interval between two consecutive join packets as received by a sender-side gateway,

- *Packet-loss rate*: the percentage of packets lost while they were passing through the networks.

Response time includes the round trip time (RTT) between sender-side and receiver-side gateways, and the delay Δ at the sender-side gateway, where RTT is the sum of communication delay from a receiver-side gateway to a sender-side gateway and that from the latter to the former, and Δ includes the waiting-time until the sender-side gateway receives IP-multicast packets and the processing time taken to encapsulate the IP-multicast packets in delivery packets and transmit the delivery packets.

We measured the response time and join packet sending-/arrival- interval by using TCPDUMP [2], while we observed the packet-loss rate by serially numbering packets at sender-side gateways and checking the packet-number of each packet received at receiver-side gateways.

3.6.3 Scenarios

The experiments were conducted using the following two scenarios:

Scenario A (Figure 40): MPEG2 live streams were simultaneously delivered to Chicago and Yokosuka from Los Angeles. Only 7.5-Mbps traffic was supposed to flow through the Japan-U.S. bottleneck link, which had the bandwidth of 17 Mbps in each direction. This scenario assessed a congestion-free situation.

Scenario B (Figure 41): Three sets of contents were simultaneously delivered: MPEG2 live streams were delivered to Chicago and Los Angeles from Yokosuka, another MPEG2 live stream to Yokosuka from Los Angeles, and Windows Media live streams to Chicago and Los Angeles from Yokosuka. Thus, over 16 Mbps of traffic had to pass from Japan to the U.S., while 7.5 Mbps of traffic was supposed to pass from the U.S. to Japan. This scenario corresponded to a congested situation.

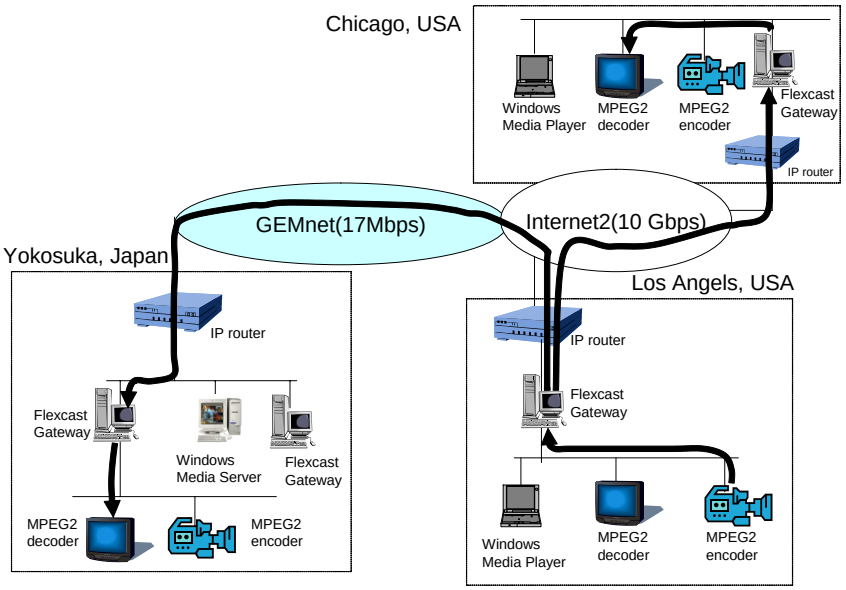

Figure 40: Scenario A

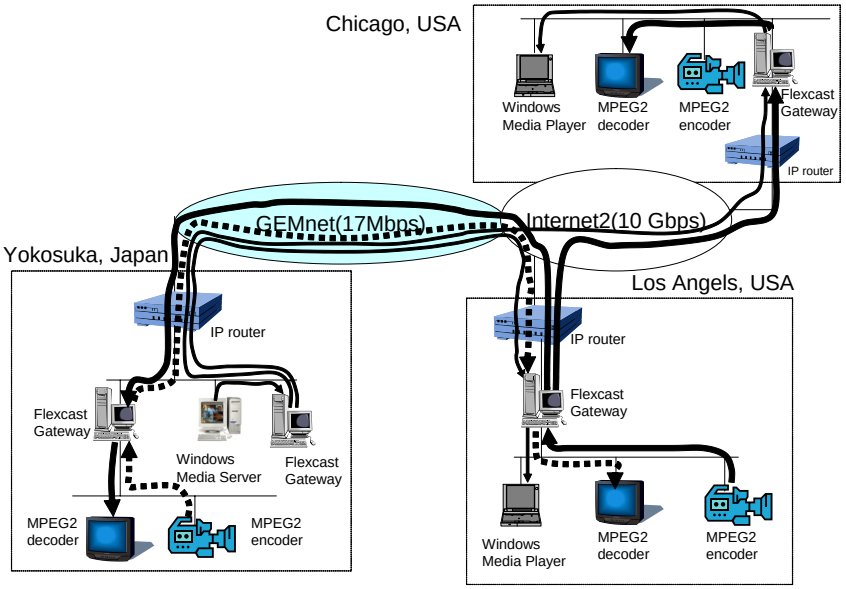

Figure 41: Scenario B

Response time (Av.) [ms]	136
Join-packet sending-interval (av.) [s]	1.00
Join-packet sending-interval (dev.) [s]	2.15E-03
Join-packet arrival-interval (av.) [sec]	1.00
Join-packet arrival-interval (dev.) [s]	2.12E-03
Packet-loss rate (Yokosuka → Los Angeles) %	0
Packet-loss rate (Los Angeles → Yokosuka) %	5

Table 5: Measured data of MPEG2 stream delivered by Flexcast from Los Angels to Yokosuka in scenario A.

3.6.4 Data analysis (scenario A)

The measured data is summarized in Table 5. The standard deviation of the join-packet arrival-interval was very small (below 10 milliseconds); the join-packet arrival-interval was almost the same as the join-packet sending-interval. This confirms the good stability of the Flexcast protocol, since the protocol allows the intervals of consecutive join packets to be up to 2 seconds in the current setting. The packet-loss rates in the directions from Yokosuka to Los Angeles and from Los Angeles to Yokosuka were 0% and 5%, respectively. Since join packets were transmitted from Yokosuka to Los Angeles, the observed packet loss rate implies that no join packets were lost. The packet-loss rate of 5% may be due to the burstiness of MPEG2, whose peak rate exceeded the bandwidth of the bottleneck link.

The round-trip time (RTT) between Los Angeles and Yokosuka was 128 milliseconds. This RTT is included in the response time of 136 milliseconds. The delay at the Yokosuka gateway would be $136 - 128 = 8$ milliseconds; this includes the waiting-time of IP-multicast packets and the processing time.

3.6.5 Data analysis (scenario B)

The traffic rate through the bottleneck link exceeded 16 Mbps in the Japan-to-U.S. direction. Since streaming traffic is very bursty in general, this situation was critical even though the bandwidth of GEMnet was 17 Mbps (CBR). We investigated the influence of the congestion of the bottleneck link on both join and delivery packets.

We observed the join packets from Yokosuka, since the congestion occurred in the Japan-to-U.S. direction. The measured data is summarized in Table 6. The standard deviation of the join-packet arrival-interval is extremely large compared to that of the join-packet sending-interval. This is due to the communication delay jitter and the high packet-loss rate (14%) caused by the congestion. Figure 42 plots

Response time (av.) [ms]	143
Join-packet sending-interval (av.) [s]	1.00
Join-packet sending-interval (dev.)[s]	2.69E-03
Join-packet arrival-interval (av.) [s]	1.12
Join-packet arrival-interval (dev.)[s]	0.138
Packet-loss rate (Yokosuka → Los Angeles)(%)	14
Packet-loss rate (Los Angeles → Yokosuka)(%)	5.2

Table 6: Measured data of MPEG2 stream delivered by Flexcast from Los Angeles to Yokosuka in scenario B.

the join-packet sending/arriving intervals. The largest and second-largest values of the join-packet arrival-interval were almost integral, i.e., around 3 (second) and 2 (second), respectively. This means that some join packets may have been lost, since the join-packet sending-interval was set to 1 second.

3.7 Summary

IP-multicast protocols have been standardized and there are many application softwares and routers that conform to IP multicast. Nevertheless, there are still some difficulties in using IP multicast over an internetwork. These difficulties result essentially from using IP-multicast addresses that are quite different from unicast addresses. Flexcast is an IP-unicast-based multicast that works across the network and application layers, and uses only IP-unicast packets, which enables progressive deployment over existing IP networks and solves the issue of the uniqueness of multicast group addresses.

To construct a high-quality multicast tree, a straightforward idea is to agglomerate the maximal common segments of the forward-paths (i.e., the unicast routing paths from the multicast sender to receivers). However, this forces the sender to transmit a packet to every receiver to assess the maximal common segments each time the receiver joins the multicast tree, since each node has only a part of the information on unicast routing, and furthermore, such information may dynamically change. It would follow that the sender is burdened with heavy load if many receivers simultaneously join, and that multicast protocols are likely to be complicated. Therefore, sharing the maximal common segments of the reverse-paths (i.e., the unicast routing paths from receivers to the sender) would be a reasonable strategy, if there is little difference in quality between forward-paths and reverse-paths.

Flexcast constructs a multicast tree by sharing the maximal common segments of reverse-paths and maintains it by a hierarchical keep-alive mechanism. Flexcast

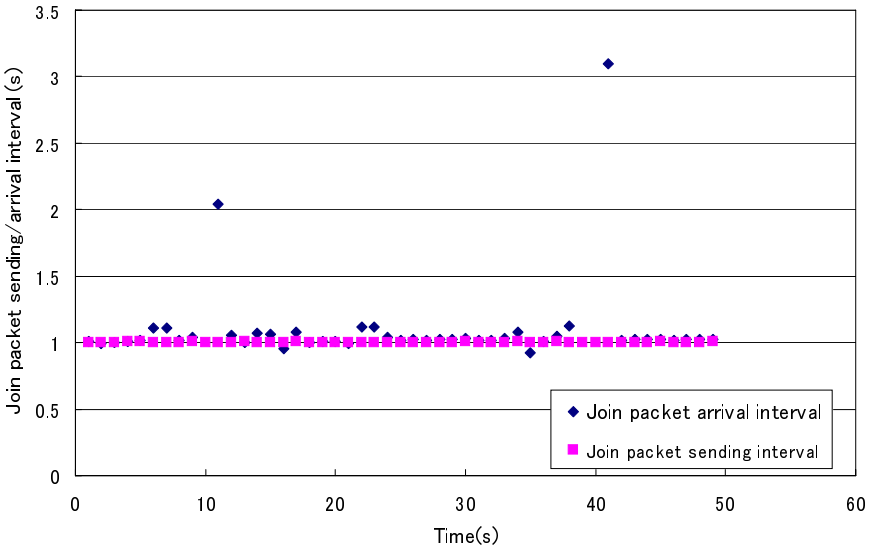

Figure 42: Join-packet sending/arrival-interval of scenario B.

protocol provides significant adaptability against the change of unicast routes, the moving of receivers and senders, and the frequent joining and leaving of receivers, with minimum rearrangement of the tree.

Many IP-unicast-based multicast protocols including the basic protocol of Flexcast allow the packets from end-hosts (receivers or senders) to change the states of network nodes. This fact may cause a serious security issue if we assume malicious users that try to collapse the multicast system by using the packets. Against this issue, we have proposed two modifications to the Flexcast protocol.

The key idea is that the states (i.e., the contents of the delivery table) of the Flexcast nodes can be changed only when both a packet from a receiver and a packet from the corresponding sender arrive at the node. This makes it difficult for each end-hosts to change the states of the Flexcast nodes without collaboration

In the Flexcast protocol and its modifications, any Flexcast node needs to change the destination of every delivery packet even if the node has only one child. This could impose a heavier load on the node than just unicast-routing the delivery packet. A modification to the Flexcast protocol was proposed in [106] to allow a Flexcast node to just unicast-route delivery packets only when the node has one child by using a similar idea introduced in [177, 72].

Streaming experiments were conducted to assess the robustness and stability of the Flexcast protocol when used to deliver streaming data to widely dispersed locations. We placed streaming servers and receivers in three locations: Yokosuka in Japan and Chicago and Los Angeles in the U.S.. The results show that the jitter

of communication delay was much smaller than expected and that the current configuration of the Flexcast parameters offer sufficient margin if no communication links experience unusual congestion. Good picture quality and quick response time were also confirmed in the experiments.

Some other experiments were also conducted. It was reported in [184] that streaming data of 1 Mbps was successfully delivered to 1,000 receiver-PC's. It was also demonstrated in [116, 178, 147] that Flexcast worked well on an MPLS Network and/or for particular broad-band applications.

Basically, Flexcast has no transmission-error correction mechanisms, but can be combined with various error correction technologies, including Automatic Repeat reQuest (ARQ) and Forward Error Correction (FEC) [157]. ARQ corrects errors by retransmitting the packets which have some errors or dropped. To realize reliable multicast protocols using ARQ, a variety of heuristics have been developed to avoid concentration of retransmission request on a sender (e.g., [129, 158]). However, ARQ may cause large delay due to the retransmission; large delay spoils the value of real-time streaming and forces us to consume large memory space for buffering all packets following dropped packets. On the other hand, FEC only takes time to process error correcting codes. Even in the case of large round trip time, FEC is quite effective for real-time streaming if the number of error/dropped packets is so moderate that the error-correcting codes can restore them.

Since ARQ and FEC each have own advantage, the combination of ARQ and FEC has been also studied as a promising direction (e.g., [154]). To examine the compatibility of Flexcast and FEC, we delivered the stream encoded by the code introduced in [61] via Flexcast and confirmed their nice collaboration.

For the future work, we mention a couple of issues concerning protocol design.

We have shown the load balancing mechanism of Flexcast. However, this may cause the congestion of particular communication links. Thus, it is important to improve the mechanism so that it can decide where to split the streaming data according to the conditions of both communication links and nodes. Of course, it is a possible option to control admission to the multicast tree.

We have proposed a way of automatically setting expiration time in the delivery table of every Flexcast node, which sets expiration time to the interval that is linear to the distance from the node to the farthest child. In a stable network, however, this method seems to be too pessimistic. It would be interesting to devise a more appropriate way of setting expiration time in a stable network, say, by efficiently assessing the conditions of communication links.

3.8 A brief survey of multicast

Constructing a multicast tree is strongly related to the Steiner tree problem [144], which is $\mathcal{NP}$-complete even if all edge weights are identical. There are many approximation algorithms for the problem (we can find an excellent review in [99]). In the case of multicast, however, the set of the nodes that are supposed to form the tree cannot be assumed to be known in advance, since nodes can dynamically join and leave the multicast group. Waxman [190] first considered this situation, and Imase and Waxman [105] formulated it as an online problem with a request sequence of adding or removing nodes: the *dynamic Steiner tree problem*. They gave an approximation algorithm whose approximation ratio[5] is within a factor of two against an optimal algorithm if no nodes are removed from the tree. They also proved that, if both adding and removing nodes are allowed and if the tree cannot be rearranged, no upper bound on the approximation ratio exists. Several algorithms [37, 17, 97] have been proposed in order to accommodate both adding and removing nodes while rearranging the tree is allowed within some limited cost. They assumed, however, that every node v knows the cost of the shortest path between v and every other node v'. This assumption would be unreasonable in an internetwork.

Multicast routing protocols over an internetwork was first introduced by Deering [78]. He proposed extensions to two common internetwork unicast routing algorithms, distance-vector routing and link-state routing, to support low-delay datagram multicasting: Distance Vector Multicast Routing Protocol (DVMRP) [189] and Multicast Open Shortest Path First (MOSPF) [150]. These multicast routing mechanisms were, however, intended for use within regions where the members of a group were densely distributed. When group members are distributed sparsely across a wide area, the mechanisms are quite inefficient. Thyagarajan and Deering [187] proposed a hierarchical version of DVMRP to improve its efficiency. Ballardie, Francis and Crowcroft [31, 30, 29] proposed the Core-Based-Tree (CBT) protocol that uses a single multicast tree for each group, even if there are multiple senders in the group. Shields and Garcia-Luna-Aceves [172] observed the basic problem of CBT with multiple cores and provided a modification to the basic protocol of CBT. The total amount of control traffic of CBT could be modest even in the case where group members are sparsely distributed, however, CBT can result in traffic concentration on certain links and large delay between a pair of group members. Deering, Estrin, Farinacci, Jacobson, Liu and Wei [80, 88] developed a multicast routing architecture, called Protocol Independent Multicast Sparse Mode (PIM-SM), that can efficiently es-

[5]The supremum, over all instances, of $\max_{0<i\leq K} \frac{A(S_i)}{OPT(S_i)}$ for a request sequence of length K, where S_i is the set of clients after the ith request, $A(S_i)$ is the cost of the multicast tree for S_i constructed by on-line algorithm A, and $OPT(S_i)$ is the cost of a minimum Steiner tree for S_i.

tablish multicast trees even when receivers are sparsely distributed. The above protocols work in the network layer and are called *IP multicast*, for which we can find more comprehensive surveys in [83, 168].

Many IP multicast protocols have been standardized, however there are still some difficulties in using IP multicast over an internetwork. To use IP-multicast, we need special IP addresses (or Class D addresses), called *IP-multicast addresses*, to specify the multicast groups. However, it is difficult to administrate IP-multicast addresses in a distributed manner to guarantee the uniqueness of each of the addresses, since IP-multicast addresses are basically independent of the structure of networks and group members may change frequently. Furthermore, all routers need to be able to route IP-multicast packets, which prevents the progressive deployment of the multicast service in existing unicast networks. As a result, IP-multicast is used only in closed or experimental networks such as MBone [87].

Holbrook and Cheriton developed the Source-Specific Multicast protocol (SSM) [100], which restricts the multicast communication to one-to-many so that it can simplify group address allocation and data distribution. More concretely, each multicast group (or channel) is represented by a pair of a multicast sender address and a group identifier that needs to be unique only within the sender. Even with this restriction, the protocol still covers most of the current multicast applications. Nevertheless, SSM does not allow the progressive deployment of the multicast service, since all routers still need to route IP-multicast packets. There is some work in progress specific to multicast tunneling, i.e., transferring IP multicast packets from/to remote multicast-capable networks by encapsulating the packets in unicast packets. Basically, however, the endpoints of tunnels have to be manually configured, or at best, one can find the endpoints by issuing a query to the servers administrating the gateways (i.e., end-points) of multicast-capable networks [133].

IP-unicast-based multicast is a technology that is promising as a tool for the progressive deployment of multicast service. It is possible for legacy routers to be located on the paths between receivers and senders, since they use just unicast packets. There are two types of IP-unicast-based multicast technologies: "*application layer multicast*" works in the application layer while the other works across the network and application layers. In application layer multicast [67, 121, 32], internal nodes of a multicast tree are usually located at the edges of the network or the receivers themselves. Each internal node terminates the streaming data, duplicates the data and forwards it to the next internal nodes or receivers. These processes are executed in the application layer. However, receivers and nodes require a special mechanism to know their children and parents, since a path between the sender and a receiver in the multicast tree is not necessarily the unicast routing path between them. This would also cause redundant routes and make the protocol not scalable.

There are several technologies that work across the network and application layers [177, 192, 72, 50], which all restrict multicast communication to one-to-many. The advantage of this kind of multicast protocols is to use the unicast routing in order to construct multicast trees. It follows that, basically, they can construct shortest-path trees based on the unicast routing; they also enjoy other benefits such as fault-tolerant routing that the unicast routing protocols provide. Stoica, Eugene and Zhang proposed a protocol called REUNITE [177], which recursively constructs a multicast tree for each channel defined as a pair of a sender address and a group identifier that needs to be unique only within the sender. REUNITE uses the unicast routing path from the sender to the receiver that first joined as the trunk of the multicast tree. Thus, all paths from the sender to receivers may not be optimal, and the leaving of the first receiver may cause drastic reconstruction of the tree, leading to instability. Furthermore, asymmetric routing leads REUNITE to unneeded packet duplications on certain links (an example is given in [72]). Costa, Fdida and Duarte [72] developed the Hop-By-Hop protocol (HBH) by modifying REUNITE to avoid the instability of REUNITE. The multicast tree of HBH seems to be the best in quality even in asymmetric networks, since HBH constructs forward-path-based shortest path trees: the path from the sender to every receiver in the tree is the unicast path from the sender to the receiver. To realize this optimality of the tree, however, the sender needs to update the tree each time a new receiver joins the tree. Furthermore, it is possible in a certain situation that a receiver cannot receive the streaming data even if it is sending join messages, as indicated in [50]. Wen, Griffioen and Calvert [192] gave a protocol that can construct forward-path-based shortest-path trees by issuing packets for probing the network. The probing packets are also used when nodes attempt to move the branch points in a multicast tree. However, the sender and receivers need to exchange many probing packets to gather information on network topology and to negotiate which branch point should be moved. While some heuristics to reduce the probing packets were given, the scalability of the protocol would be limited to some extent.

For the multicast of a small number of receivers, there are protocols that have a completely different idea from that of the above protocols. Xcast [46, 51] lists all the receivers' addresses in the header of packets sent by senders. Each Xcast-aware router along the way parses the header, partitions the destinations with respect to next hops, and forwards a copy of the packet with an appropriate Xcast header to each next hop. Boudani and Cousin [50] proposed a similar approach, but they separated control packets and data packets for efficiency.

4 Quantum Multi-Point Communication – Leader Election in Anonymous Networks –

This section discusses quantum multi-point communication by considering the leader election problem in the anonymous setting.

4.1 Preliminaries

4.1.1 Quantum Computation

Here we briefly introduce quantum computation (for more detailed introduction, see [153, 126]). Suppose that there is a system consisting of q components, each of which has two states, say 0 or 1. If the system is subject to classical physics, q bits are sufficient to describe a state of the system. However, in a quantum system, 2^q complex numbers are required to describe a state of the system. To be more precise, each state of the quantum system is a unit-length vector of the 2^q-dimensional Hilbert space. For any basis $\{|B_0\rangle, \ldots, |B_{2^q-1}\rangle\}$ of the space, every state can be represented by

$$\sum_{i=0}^{2^q-1} \alpha_i |B_i\rangle,$$

where complex number α_i, called *amplitude*, holds $\sum_i |\alpha_i|^2 = 1$. Each component of the quantum system is called a *qubit*, which is a unit of quantum information and computation. There is a simple basis in which every basis state $|B_i\rangle$ corresponds to one of 2^q possible classical positions in the space: $(i_0, i_2, \ldots, i_{q-1}) \in \{0,1\}^q$. We often denote $|B_i\rangle$ by

$$|i_0\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_{q-1}\rangle,$$

$$|i_0\rangle|i_2\rangle \cdots |i_{q-1}\rangle,$$

or

$$|i_0 i_2 \cdots i_{q-1}\rangle,$$

where $\otimes$ denotes the tensor product. This basis is called the *computational basis*.

If the quantum system is measured with respect to any basis $\{B_0, \ldots, B_{2^q-1}\}$, the probability of observing basis state $|B_i\rangle$ is $|\alpha_i|^2$. As a result of measurement, the state is projected on the observed basis state. Measurement can also be performed on a part of the system or some of the qubits forming the system. Let $\sum_{i=0}^{2^q-1} \alpha_i |B_i\rangle$ be $|\phi_0\rangle|0\rangle + |\phi_1\rangle|1\rangle$. If we measure the last qubit with respect to basis $(|0\rangle, |1\rangle)$, we obtain $|0\rangle$ with probability $\langle\phi_0|\phi_0\rangle$ and $|1\rangle$ with probability $\langle\phi_1|\phi_1\rangle$, where $\langle\phi|\psi\rangle$ is the inner product of vectors $|\phi\rangle$ and $|\psi\rangle$, leaving post-measurement states are $|\phi_0\rangle/\sqrt{\langle\phi_0|\phi_0\rangle}$ and $|\phi_1\rangle/\sqrt{\langle\phi_1|\phi_1\rangle}$, respectively. In the

same way, we measure the system with respect to other bases. In particular, we call

$$\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}, \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)$$

the Hadamard basis of the 2-dimensional Hilbert space, denoted by $(|+\rangle, |-\rangle)$. If $\sum_{i=0}^{2^n-1} \alpha_i |B_i\rangle$ is expressed as $|\phi_+\rangle|+\rangle + |\phi_-\rangle|-\rangle$, by measuring the last qubit with respect to the Hadamard basis, we observe $|+\rangle$ with probability $\langle\phi_+|\phi_+\rangle$ and $|-\rangle$ with probability $\langle\phi_-|\phi_-\rangle$, leaving post-measurement states are $|\phi_+\rangle/\sqrt{\langle\phi_+|\phi_+\rangle}$ and $|\phi_-\rangle/\sqrt{\langle\phi_-|\phi_-\rangle}$, respectively.

In order to perform computation over the quantum system, we must be able to change the state of the system. The laws of quantum mechanics permit only unitary transformations over the Hilbert space. These transformations are represented by unitary matrices, where a unitary matrix is one whose conjugate transpose is equal to its inverse.

4.1.2 The distributed network model

A *distributed system* (or *network*) is composed of multiple parties and bidirectional classical communication links connecting parties. In a quantum distributed system, every party can perform quantum computation and communication, and each adjacent pair of parties has a bidirectional quantum communication link between them. When the parties and links are regarded as nodes and edges, respectively, the topology of the distributed system is expressed by an undirected connected graph, denoted by $G = G(V, E)$. In what follows, we may identify each party/link with its corresponding node/edge in the underlying graph for the system, if it is not confusing. Every party has *ports* corresponding one-to-one to communication links incident to the party. Every port of party l has a unique label i, $(1 \leq i \leq d_l)$, where d_l is the number of parties adjacent to l. More formally, G has a *port numbering*, which is a set σ of functions $\{\sigma[v] \mid v \in V\}$ such that, for each node v of degree d_v, $\sigma[v]$ is a bijection from the set of edges incident to v to $\{1, 2, \ldots, d_v\}$. It is stressed that each function $\sigma[v]$ may be defined independently of the others. Just for ease of explanation, we assume that port i corresponds to the link connected to the ith adjacent party of party l. In our model, each party knows the number of his ports and the party can appropriately choose one of his ports whenever he transmits or receives a message.

Initially, every party has local information, such as his local state, and global information, such as the number of nodes in the system (or his upper bound). Every party runs the same algorithm, which has local and global information as its arguments. If all parties have the same local and global information except the number of ports the parties have, the system is said to be *anonymous*. This is

essentially equivalent to the situation in which every party has the same identifier since we can regard the local/global information of the party as his identifier.

Traditionally, the distributed system is either synchronous or asynchronous. In the synchronous case, message passing is performed synchronously. The unit interval of synchronization is called a *round*, which is the combination of the following two steps [138], where two functions that generate messages and change local states are defined by the algorithm invoked by each party: (1) each party changes the local state as a function of the current local state and the incoming messages, and removes the messages from the ports; (2) each party generates messages and decides ports through which the messages should be sent as another function of the current local state, and sends the messages via the ports. If message passing is performed synchronously, a distributed system is called *synchronous*.

4.1.3 Leader election problem in anonymous networks

The leader election problem is formally defined as follows.

Definition 5 (Leader Election Problem (LE_n)) *Suppose that there is an n-party distributed system with the underlying graph G, and that each party $i \in \{1, 2, \ldots, n\}$ in the system has a variable x_i initialized to some constant c_i. For a certain integer $k \in \{1, 2, \ldots, n\}$, set x_k to 1 and $x_i = 0$ for $i \neq k$.*

When each party i has his own unique identifier, i.e., $c_i \in \{1, 2, \ldots, n\}$ such that $c_i \neq c_j$ for $i \neq j$, LE_n can be deterministically solved in $\Theta(n)$ rounds in the synchronous case and $\Theta(n \log n)$ rounds in the asynchronous case [138].

When $c_i = c_j$ for all i and j ($i \neq j$), we say that the parties are anonymous and that distributed system (network) consisting of anonymous parties is said to be *anonymous*. The leader election problem in an anonymous network was first investigated by Angluin [27]. Her model is based on Milne and Milner's model [146], in which any adjacent two parties can toss coins, and among the two parties a leader can exactly be elected, where "exactly" means "in bounded time and without error." In her model, she showed that there are anonymous networks for which no algorithms exist that exactly solve the problem, and gave a necessary and sufficient condition in terms of graph covering to exactly solve the leader election problem. In the anonymous network model defined in subsection 4.1.2, which does not assume coin-tossing between any adjacent parties, Yamashita and Kameda [195] proved that, if the "symmetricity" (defined in [195]) of the network topology is more than one, LE_n cannot be solved exactly (more rigorously speaking, there are some port numberings for which LE_n cannot be solved exactly) by any classical algorithm even if all parties know the topology of the network (and thus the number of parties). Symmetricity more than one implies that, a certain

port numbering satisfies that, for any party i, there is an automorphism on the underlying graph with the port numbering that exchanges i and another party i'. This condition holds for a broad class of graphs, including regular graphs. Formally, the symmetricity is defined later by using "views".

Since it is quite hard to exactly solve the problem in the classical setting, many probabilistic algorithms were proposed. Itai and Rodeh [117, 118] gave a zero-error algorithm for a synchronous/asynchronous unidirectional ring of size n that is expected to take $O(n)$/$O(n \log n)$ rounds by $O(n)$/$O(n \log n)$ bit communication.

When every party knows only the upper bound of the number of the parties that is at most twice the exact number, Itai and Rodeh [118] showed that the problem can be solved with zero error. In general, however, they showed that there is no zero-error classical algorithm for a ring. This impossibility result can be extended to a general topology having cycles. They also proved that there is a bounded-error algorithm for a ring, given an upper bound of the number of parties.

Schieber and Snir [169] gave bounded-error algorithms for any topology even when no information on the number of parties is available, although every party cannot detect the termination of the algorithms (such algorithms are called *message termination algorithms*). Subsequently, Afek and Matias [15] described more efficient bounded-error message termination algorithms.

Yamashita and Kameda [196] examined the case in which every party has an identity number that is not necessarily distinct in four network models that differ in every party's ability to distinguish the ports through which he sends or receives messages.

4.2 Quantum leader election algorithm I

For simplicity, we assume that the network is synchronous and each party knows the number n of parties prior to algorithm invocation. It is easy to generalize our algorithm to the asynchronous case and to the case where only the upper bound N of the number of parties is given, as will be discussed at the end of this subsection.

Initially, all parties are eligible to become the unique leader. The key to solving the leader election problem in an anonymous network is to break symmetry, i.e., to have just a single party possess a certain state corresponding to the leader.

First we introduce the concept of *consistent* and *inconsistent* strings. Suppose that each party l has a c-bit string x_l: the n parties share cn-bit string $x = x_1 x_2 \cdots x_n$. For convenience, we may consider that each x_l expresses an integer, and identify string x_l with the integer it expresses. Given a set $S \subseteq \{1, \ldots, n\}$, string x is said to be *consistent* over S if x_l has the same value for all l in S. Otherwise, x is said to be *inconsistent* over S. We also say that a cn-qubit pure state $|\psi\rangle = \sum_x \alpha_x |x\rangle$ shared by the n parties is *consistent (incon-*

sistent) over S if $\alpha_x \neq 0$ only for x's that are consistent (inconsistent) over S. We do not consider here a state in which both consistent and inconsistent strings are superposed, since we do not use such a state in our algorithms. For a positive integer m, we denote the pure state that is of the form of $(|0^m\rangle + |1^m\rangle)/\sqrt{2}$, by the *$m$-cat state*. When we apply the operator of the form

$$\sum_j \alpha_j |\eta_j\rangle \mapsto \sum_j \alpha_j |\eta_j\rangle \otimes |\eta_j\rangle$$

for some orthonormal basis $\{|\eta_j\rangle\}$, we just say "copy" if it is not confusing.

4.2.1 The algorithm

The algorithm repeats one procedure exactly $(n-1)$ times, each of which is called a *phase*. In each phase, the number of eligible parties either decreases or remains the same, but never increases or becomes zero. After $(n-1)$ phases, the number of eligible parties becomes one with certainty.

Each phase has a parameter denoted by k, whose value is $(n-i+1)$ in phase i. In each phase i, let $S_i \subseteq \{1,\ldots,n\}$ be the set of all ls such that party l is still eligible. First, each eligible party prepares the state $(|0\rangle + |1\rangle)/\sqrt{2}$ in register $\mathbf{R}_0$, while each ineligible party prepares the state $|0\rangle$ in $\mathbf{R}_0$. Next every party calls Subroutine A, followed by partial measurement. This transforms the system state, i.e., the state in all parties' $\mathbf{R}_0$s into either $(|0^{|S_i|}\rangle + |1^{|S_i|}\rangle) \otimes |0^{n-|S_i|}\rangle/\sqrt{2}$ or a state that is inconsistent over S_i, where the first $|S_i|$ qubits represent the qubits in eligible parties' $\mathbf{R}_0$s. In the former case, each eligible party calls Subroutine B, which uses a new ancilla qubit in register $\mathbf{R}_1$. If k equals $|S_i|$, Subroutine B always succeeds in transforming the $|S_i|$-cat state in eligible parties' $\mathbf{R}_0$s into a $2|S_i|$-qubit state that is inconsistent over S_i by using the $|S_i|$ ancilla qubits. Now, each eligible party l measures his qubits in $\mathbf{R}_0$ and $\mathbf{R}_1$ in the computational basis to obtain (a binary expression of) some two-bit integer z_l. Parties then compute the maximum value of z_l over all eligible parties l, by calling Subroutine C. Finally, parties with the maximum value remain eligible, while the other parties become ineligible. More precisely, each party l having d_l adjacent parties performs Algorithm I described in Figure 43 with parameters "eligible," n, and d_l. The party who obtains the output "eligible" is the unique leader. Precise descriptions of Subroutines A, B, and C are to be found in 4.2.2, 4.2.3, and 4.2.4, respectively.

4.2.2 Subroutine A

Subroutine A is essentially for the purpose of checking the consistency over S_i of each string that is superposed in a quantum state shared by parties. We use a commute operator "$\circ$" over a set $\mathcal{S} = \{0, 1, *, \times\}$ whose operations are summarized in Table 7. Intuitively, "0" and "1" represent the possible values all eligible

Algorithm I

Input: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$, integers n, d

Output: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$

1. Prepare one-qubit quantum registers $\mathbf{R}_0$, $\mathbf{R}_1$, and $\mathbf{S}$.
2. For $k := n$ down to 2, do the following:
   - 2.1 If $\mathbf{status} =$ "eligible," prepare the states $(|0\rangle + |1\rangle)/\sqrt{2}$ and $|$"consistent"$\rangle$ in $\mathbf{R}_0$ and $\mathbf{S}$, otherwise prepare the states $|0\rangle$ and $|$"consistent"$\rangle$ in $\mathbf{R}_0$ and $\mathbf{S}$.
   - 2.2 Perform Subroutine A with $\mathbf{R}_0$, $\mathbf{S}$, $\mathbf{status}$, n, and d.
   - 2.3 Measure the qubit in $\mathbf{S}$ in the $\{|$"consistent"$\rangle, |$"inconsistent"$\rangle\}$ basis. If this results in $|$"consistent"$\rangle$ and $\mathbf{status} =$ "eligible," prepare the state $|0\rangle$ in $\mathbf{R}_1$ and perform Subroutine B with $\mathbf{R}_0$, $\mathbf{R}_1$, and k.
   - 2.4 If $\mathbf{status} =$ "eligible," measure the qubits in $\mathbf{R}_0$ and $\mathbf{R}_1$ in the $\{|0\rangle, |1\rangle\}$ basis to obtain a nonnegative integer z expressed by the two bits; otherwise let $z := -1$.
   - 2.5 Perform Subroutine C with z, n, and d to know the maximum value $z_{\max}$ of z over all parties. If $z \neq z_{\max}$, let $\mathbf{status} :=$ "ineligible."
3. Output $\mathbf{status}$.

Figure 43: Quantum leader election algorithm I.

parties will have when the string finally turns out to be consistent; "$*$" represents "don't care," which means that the corresponding party has no information on the values possessed by eligible parties; and "$\times$" represents "inconsistent," which means that the corresponding party already knows that the string is inconsistent. Although Subroutine A is essentially the same as the algorithm in [131], we give the precise description of Subroutine A in Figure 44 for completeness.

As one can see from the description of Algorithm I, the content of $\mathbf{S}$ is "consistent" whenever Subroutine A is called. Therefore, after every party finishes Subroutine A, the state shared by parties in their $\mathbf{R}_0$s is decomposed into a consistent state for which each party has the content "consistent" in his $\mathbf{S}$, and an inconsistent state for which each party has the content "inconsistent" in his $\mathbf{S}$. Steps 4 and 5 are performed to disentangle work quantum registers $\mathbf{X}_i^{(t)}$s.

The next two lemmas are for correctness and complexity.

Lemma 17 *Suppose that n parties share n-qubit state $|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle$ in n one-qubit registers $\mathbf{R}_0$s, where α_is are any complex numbers such that*

$\sum_{i=0}^{2^n-1} \alpha_i \alpha_i^* = 1$. *If each party runs Subroutine A with the following objects as input: (1)* $\mathbf{R}_0$ *and another one-qubit quantum register* $\mathbf{S}$*, which is initialized to* $|0\rangle$*, (2) a classical variable* $\mathbf{status} \in \{\mathsf{eligible}, \mathsf{ineligible}\}$*, (3)* n *and the number* d *of neighbors of the party, Subroutine A then outputs* $\mathbf{R}_0$ *and* $\mathbf{S}$ *such that the qubits in all parties'* $\mathbf{R}_0$*s and* $\mathbf{S}$*s are in the state* $\sum_{i=0}^{2^n-1}(\alpha_i|i\rangle \otimes |s_i\rangle^{\otimes n})$*, where* s_i *is "*$\mathsf{consistent}$*" if* $|i\rangle$ *is consistent over* S*, i.e., the set of indices of parties whose* $\mathbf{status}$ *is "*$\mathsf{eligible}$*," and "*$\mathsf{inconsistent}$*" otherwise.*

Proof. Subroutine A just superposes an application of a reversible classical algorithm to each basis state. Furthermore, no interference occurs since the contents of $\mathbf{R_0}$s are never changed during the execution of the subroutine. Thus, it is sufficient to prove the correctness of Subroutine A when the content of $\mathbf{R}_0$ is a classical bit. It is stressed that all ancilla qubits used as work space can be disentangled by inverting every communication and computation.

Suppose that we are given one-bit *classical* registers $\mathbf{R_0}$ and $\mathbf{S}$, *classical* variable $\mathbf{status}$, and integers n and d. For any party l and a positive integer t, the content of $\mathbf{X}_{0,l}^{(t+1)}$ is set to $x_{0,l}^{(t)} \circ x_{1,l}^{(t)} \circ \cdots \circ x_{d,l}^{(t)}$ in step 2.3, where $\mathbf{X}_{i,l}^{(t)}$ is $\mathbf{X}_i^{(t)}$ of party l, and $x_{i,l}^{(t)}$ is the content of $\mathbf{X}_{i,l}^{(t)}$. For any l, by expanding this recurrence relation, the content of $\mathbf{X}_{0,l}^{(n)}$ can be expressed in the form of $x_{0,l_1}^{(1)} \circ \cdots \circ x_{0,l_m}^{(1)}$ for some $m \leq (n-1)^{(n-1)}$. Since the diameter of the underlying graph is at most $n-1$, there is at least one $x_{0,l'}^{(1)}$ in $x_{0,l_1}^{(1)}, \ldots, x_{0,l_m}^{(1)}$ for each l'. Thus $x_{0,l_1}^{(1)} \circ \cdots \circ x_{0,l_m}^{(1)}$ is equal to $x_{0,1}^{(1)} \circ x_{0,2}^{(1)} \circ \cdots \circ x_{0,n}^{(1)}$, since $\circ$ is commutative and associative, and $x \circ x = x$ for any $x \in \{0, 1, *, \times\}$.

Therefore, we can derive the following facts: (1) if and only if there are both 0 and 1 in the contents of $\mathbf{R}_0$s of all parties, Subroutine A outputs $\mathbf{S} =$'$\times$', which will be translated into "$\mathsf{inconsistent}$"; (2) if and only if there are either 0's or 1's but not both in the contents of $\mathbf{R}_0$s (which possibly include '$*$'), Subroutine A outputs $\mathbf{S} =$'0' or '1', respectively, which are both translated into "$\mathsf{consistent}$." □

Lemma 18 *Let* $|E|$ *and* D *be the number of edges and the maximum degree of the underlying graph, respectively. Subroutine A takes* $\Theta(n)$ *rounds and* $\Theta(Dn)$ *time. The total communication complexity over all parties is* $\Theta(|E|n)$.

Proof. Since step 3 takes constant time and steps 4 and 5 are just the inversions of steps 2 and 1, respectively. We thus consider steps 1 and 2. Step 1 takes at most $\Theta(Dn)$ time. For each t, steps 2.1 and 2.1 take $\Theta(D)$ time, and step 2.3 can compute $x_0^{(t)} \circ x_1^{(t)} \circ \cdots \circ x_d^{(t)}$ in $O(D)$ time by performing each $\circ$ one-by-one from left to right. Hence step 2 takes $\Theta(Dn)$ time.

As for the number of rounds and communication complexity, it is sufficient to consider just step 2, since only steps 2 and 4 involve communication and step 4 is

Table 7: The definition of commute operator "$\circ$."

x	y	$x \circ y$	x	y	$x \circ y$	x	y	$x \circ y$	x	y	$x \circ y$
0	0	0	1	0	$\times$	$*$	0	0	$\times$	0	$\times$
0	1	$\times$	1	1	1	$*$	1	1	$\times$	1	$\times$
0	$*$	0	1	$*$	1	$*$	$*$	$*$	$\times$	$*$	$\times$
0	$\times$	$\times$	1	$\times$	$\times$	$*$	$\times$	$\times$	$\times$	$\times$	$\times$

the inversion of step 2. It is easy to see that the number of rounds is $\Theta(n)$. With regard to communication complexity, every party sends two qubit via each link for each iteration in step 2. Hence every party needs to send $\Theta(nD)$ qubits in step 2. By summing up the number of qubits sent over all parties, the communication complexity is $\Theta(|E|n)$. □

4.2.3 Subroutine B

Suppose that, among n parties, k parties are still eligible and share the k-cat state $(|0^k\rangle + |1^k\rangle)/\sqrt{2}$ in their $\mathbf{R}_0$'s. Subroutine B has the purpose of transforming the k-cat state to an inconsistent state with certainty by using k fresh ancilla qubits that are initialized to $|0\rangle$, when k is given. Figure 45 gives the precise description of Subroutine B, where $\{U_k\}$ and $\{V_k\}$ are two families of unitary operators,

$$U_k = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{-i\frac{\pi}{k}} \\ -e^{i\frac{\pi}{k}} & 1 \end{pmatrix},$$

$$V_k = \frac{1}{\sqrt{R_k + 1}} \begin{pmatrix} 1/\sqrt{2} & 0 & \sqrt{R_k} & e^{i\frac{\pi}{k}}/\sqrt{2} \\ 1/\sqrt{2} & 0 & -\sqrt{R_k}e^{-i\frac{\pi}{k}} & e^{-i\frac{\pi}{k}}/\sqrt{2} \\ \sqrt{R_k} & 0 & \frac{e^{-i\frac{\pi}{2k}}I_k}{i\sqrt{2}R_{2k}} & -\sqrt{R_k} \\ 0 & \sqrt{R_k + 1} & 0 & 0 \end{pmatrix},$$

where R_k and I_k are the real and imaginary parts of $e^{i\frac{\pi}{k}}$, respectively.

The point is that the amplitudes of the states $|00\rangle^{\otimes k}$, $|01\rangle^{\otimes k}$, $|10\rangle^{\otimes k}$, and $|11\rangle^{\otimes k}$ shared by k eligible parties in their registers $\mathbf{R}_0$ and $\mathbf{R}_1$ are simultaneously zero after every eligible party applies Subroutine B with parameter k, if the qubits in $\mathbf{R}_0$s of all eligible parties form the k-cat state. The next two lemmas describe this rigorously.

Instead of U_k, we give a proof for a more general case.

Subroutine A

Input: one-qubit quantum registers $\mathbf{R}_0$ and $\mathbf{S}$, a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$, integers n, d

Output: one-qubit quantum registers $\mathbf{R}_0$ and $\mathbf{S}$

1. Prepare two-qubit quantum registers
$$\mathbf{X}_0^{(1)}, \ldots, \mathbf{X}_d^{(1)}, \ldots, \mathbf{X}_0^{(n-1)}, \ldots, \mathbf{X}_d^{(n-1)}, \mathbf{X}_0^{(n)}.$$
If $\mathbf{status} =$ "eligible," copy the content of $\mathbf{R}_0$ to $\mathbf{X}_0^{(1)}$; otherwise set the content of $\mathbf{X}_0^{(1)}$ to "$*$."
2. For $t := 1$ to $n-1$, do the following:
   2.1 Copy the content of $\mathbf{X}_0^{(t)}$ to each of $\mathbf{X}_1^{(t)}, \ldots, \mathbf{X}_d^{(t)}$.
   2.2 Exchange the qubit in $\mathbf{X}_i^{(t)}$ with the party connected via port i for $1 \leq i \leq d$ (i.e., the original qubit in $\mathbf{X}_i^{(t)}$ is sent via port i, and the qubit received via that port is newly set in $\mathbf{X}_i^{(t)}$).
   2.3 Set the content of $\mathbf{X}_0^{(t+1)}$ to $x_0^{(t)} \circ x_1^{(t)} \circ \cdots \circ x_d^{(t)}$, where $x_i^{(t)}$ denotes the content of $\mathbf{X}_i^{(t)}$ for $0 \leq i \leq d$.
3. If the content of $\mathbf{X}_0^{(n)}$ is "$\times$," turn the content of $\mathbf{S}$ over (i.e., if the content of $\mathbf{S}$ is "consistent," it is flipped to "inconsistent").
4. Invert every computation and communication in Step 2.
5. Invert every computation in Step 1.
6. Output quantum registers $\mathbf{R}_0$ and $\mathbf{S}$.

Figure 44: Subroutine A.

Lemma 19 *Suppose that, for any even integer $k(\geq 2)$, k parties each have one of the k-cat-state qubits. After every party performs*

$$U_k(\psi, t) = \frac{1}{\sqrt{2}} \begin{pmatrix} e^{i\psi} & e^{i(\psi - \frac{2t+1}{k}\pi)} \\ -e^{-i(\psi - \frac{2t+1}{k}\pi)} & e^{-i\psi} \end{pmatrix}$$

to his qubit, the resulting k-qubit state is inconsistent over the set of the indices of the k parties, where ψ and t are any fixed real and integer, respectively.

Proof. $U_k(\psi, t)$ is unitary since $U_k(\psi, t)U_k(\psi, t)^\dagger = U_k(\psi, t)^\dagger U_k(\psi, t) = I$, where $U_k(\psi, t)^\dagger$ is the adjoint of $U_k(\psi, t)$, and I is the two-dimensional identity operator. It is sufficient to prove that the amplitudes of states $|00\ldots0\rangle$ and $|11\ldots1\rangle$ are both zero after every party applies $U_k(\psi, t)$ to his k-cat-state qubit.

Subroutine B

Input: one-qubit quantum registers $\mathbf{R}_0, \mathbf{R}_1$, an integer k

Output: one-qubit quantum registers $\mathbf{R}_0, \mathbf{R}_1$

1. If k is even, apply U_k to the qubit in $\mathbf{R}_0$; otherwise copy the content in $\mathbf{R}_0$ to that in $\mathbf{R}_1$, and then apply V_k to the qubits in $\mathbf{R}_0$ and $\mathbf{R}_1$.
2. Output quantum registers $\mathbf{R}_0$ and $\mathbf{R}_1$.

Figure 45: Subroutine B.

After every party applying $U_k(\psi, t)$, the amplitude of state $|00\ldots0\rangle$ is

$$\frac{1}{\sqrt{2}}\left(\left(\frac{e^{i\psi}}{\sqrt{2}}\right)^k + \left(\frac{e^{i(\psi-\frac{2t+1}{k}\pi)}}{\sqrt{2}}\right)^k\right) = 0.$$

The amplitude of state $|11\ldots1\rangle$ is

$$\frac{1}{\sqrt{2}}\left(\left(\frac{-e^{-i(\psi-\frac{2t+1}{k}\pi)}}{\sqrt{2}}\right)^k + \left(\frac{e^{-i\psi}}{\sqrt{2}}\right)^k\right) = 0,$$

since k is even. □

Corollary 20 *Suppose that, for any even integer $k(\geq 2)$, k parties each have one of the k-cat-state qubits. After every party performs $U_k \otimes I$ to the qubit and a fresh ancilla qubit, the resulting $2k$-qubit state is inconsistent over S, where S is the set of the indices of the k parties.*

Proof. By setting both ψ and t to 0 in Lemma 19, the proof is completed. □

As for V_k, we can prove similarly.

Lemma 21 *Suppose that, for any odd integer k (≥ 3), k parties each have two of $2k$-cat-state qubits. After every party performs V_k to his two qubits, the resulting $2k$-qubit state is inconsistent over S, where S is the set of indices of the k parties.*

Proof. The matrix of V_k is well-defined since the denominator in any element of V_k is positive due to $R_k + 1 > 0$ and $R_{2k} > 0$ for $k \geq 3$. We can verify that V_k is unitary by some calculation.

To complete the proof, we will show that the amplitudes of states $|00\ldots00\rangle$, $|01\ldots01\rangle$, $|10\ldots10\rangle$, and $|11\ldots11\rangle$ are all zero after every party applies V_k to

his two qubits, since these states imply that all parties observe the same two-bit value by measuring their two qubits. Here we assume the ordering of qubits in which party l has the $(2l-1)$st and $2l$th qubits for $l = 1, 2, \ldots, k$.

After every party applies V_k, the amplitude of state $|00\ldots00\rangle$ is

$$\frac{1}{\sqrt{2}}\left\{\left(\frac{1}{\sqrt{2(R_k+1)}}\right)^k + \left(\frac{e^{i\frac{\pi}{k}}}{\sqrt{2(R_k+1)}}\right)^k\right\} = 0.$$

In the same way, the amplitudes of states $|01\ldots01\rangle$ and $|10\ldots10\rangle$ are

$$\frac{1}{\sqrt{2}}\left\{\left(\frac{1}{\sqrt{2(R_k+1)}}\right)^k + \left(\frac{e^{-i\frac{\pi}{k}}}{\sqrt{2(R_k+1)}}\right)^k\right\} = 0,$$

$$\frac{1}{\sqrt{2}}\left\{\left(\frac{R_k}{\sqrt{R_k+1}}\right)^k + \left(-\frac{R_k}{\sqrt{R_k+1}}\right)^k\right\} = 0,$$

respectively, since k is odd. The amplitude of state $|11\ldots11\rangle$ is obviously 0. □

From the above two lemmas, the correctness of Subroutine B is immediate.

Lemma 22 *Suppose that k parties each have a qubit in one-qubit register $\mathbf{R}_0$ whose content forms a k-cat state together with the contents of the $(k-1)$ qubits of the other parties, and prepare a fresh ancilla qubit initialized to $|0\rangle$ in another one-qubit register $\mathbf{R}_1$. After running Subroutine B with $\mathbf{R}_0$, $\mathbf{R}_1$ and k, the qubits in $\mathbf{R}_0$s and $\mathbf{R}_0$s form an inconsistent state over S, where S is the set of the indices of the k parties.*

The next lemma is obvious.

Lemma 23 *Subroutine B takes $O(1)$ time and needs no communication.*

4.2.4 Subroutine C

Subroutine C is a classical algorithm that computes the maximum value over the values of all parties. It is very similar to Subroutine A. In fact, Subroutines A and C can be merged into one subroutine, although we will explain them separately for simplicity. Figure 46 gives the precise description of Subroutine C.

Lemma 24 *Suppose that each party l has integer z_l and d_l neighbors in an n-party distributed system. If every party l runs Subroutine C with $z = z_l$, n and $d = d_l$ as input, Subroutine C outputs the maximum value $z_{\max}$ among all z_ls.*

Subroutine C

Input: integers z, n, d

Output: an integer $z_{\max}$

1. Let $z_{\max} := z$.
2. For $t := 1$ to $n - 1$, do the following:
   2.1 Let $y_0 := z_{\max}$.
   2.2 Send y_0 via every port i for $1 \leq i \leq d$.
   Set y_i to the value received via port i for $1 \leq i \leq d$.
   2.3 Let $z_{\max} := \max_{0 \leq i \leq d} y_i$.
3. Output $z_{\max}$.

Figure 46: Subroutine C.

Proof. We will prove by induction the next claim: after repeating steps 2.1 to 2.3 t times, $z_{\max}$ of party l is the maximum among z_js of all parties j who can be reached from party l via a path of length at most t. When $t = 1$, the claim obviously holds. Assume that the claim holds for $t = m$. After the next iteration of steps 2.1 to 2.3, $z_{\max}$ is updated to the maximum value among y_0's of party l and his neighbors. Since y_0 is the $z_{\max}$ of the previous iteration, the claim holds for $t = m + 1$ due to the assumption. Since any graph has diameter at most $n - 1$, Subroutine C outputs the maximum value $z_{\max}$ among all z_is. □

In a quite similar way of the proof of Lemma 18, we have the next lemma.

Lemma 25 *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph, respectively. Subroutine C takes $\Theta(n)$ rounds and $\Theta(Dn)$ time. The total communication complexity over all parties is $\Theta(|E|n)$.*

4.2.5 Complexity analysis and generalization

Theorem 26 *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph, respectively. Given a classical variable* $\mathbf{status}$ *initialized to* "eligible" *and the number n of parties, Algorithm I exactly elects a unique leader in $\Theta(n^2)$ rounds and $\Theta(Dn^2)$ time. Each party connected with d parties requires $\Theta(dn^2)$-qubit communication, and the total communication complexity over all parties is $\Theta(|E|n^2)$.*

Proof. Let S_i be the set of the indices of parties with $\mathbf{status}$ = "eligible" (i.e., eligible parties) just before phase i. From Lemmas 17 and 22, we can see that,

in each phase i, Algorithm I generates an inconsistent state over S_i, if $k = |S_i|$. Algorithm I then decreases the number of the eligible parties in step 2.5 by at least one, which is implied by Lemma 24. If k is not equal to $|S_i|$, the number of the eligible parties are decreased or unchanged. We can thus prove that k is always at least $|S_i|$ in any phase i by induction, since $k = |S_1|$ before entering phase 1 and k is decreased by 1 in every phase. It is stressed that there is always at least one eligible party, since the eligible parties having $z = z_{\max}$ at step 2.5 remain eligible. It follows that, after step 2, the number of eligible parties is exactly 1. This proves the correctness of Algorithm I.

As for complexity, Subroutines A, B and C are dominant in step 2. Due to Lemmas 18, 23 and 25; the total communication complexity is $\Theta(|E|n) \times n = \Theta(|E|n^2)$ (each party with d neighbors requires $\Theta(dn^2)$ communication complexity); the time complexity is $\Theta(Dn) \times n = \Theta(Dn^2)$; the number of rounds required is $\Theta(n) \times n = \Theta(n^2)$. □

If each party knows only the upper bound N of the number of parties in advance, each party has only to perform Algorithm I with N instead of n. The correctness in this case is obvious from the proof of Theorem 26. The complexity is described simply by replacing every n by N in Theorem 26.

Corollary 27 *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph, respectively. Given a classical variable* status *initialized to* "eligible" *and the number N of parties, Algorithm I exactly elects a unique leader in $\Theta(N^2)$ rounds and $\Theta(DN^2)$ time. Each party connected with d parties requires $\Theta(dN^2)$-qubit communication, and the total communication complexity over all parties is $\Theta(|E|N^2)$.*

Furthermore, Algorithm I is easily modified so that it works well even in the asynchronous settings. Note that all parties receive messages via each port at each round. In the modified version, each party postpones performing the operations of the $(i+1)$st round until he finishes receiving all messages that are supposed to be received at the ith round. If all communication links work in the first-in-first-out manner, it is easy to recognize the messages sent at the ith round for any i. Otherwise, we tag every message, which increases the communication and time complexity by multiplicative factor $O(\log n)$, in order to know at which round every received message was sent. This modification enables us to simulate synchronous behavior in asynchronous networks.

4.2.6 Combination with the quantum amplitude amplification

Another exact algorithm can be obtained by combining Subroutines A and C with the exact quantum amplitude amplification [182]; the algorithm has the same complexity as Algorithm I up to a multiplicative constant factor.

We first quote the exact quantum amplitude amplification theorem.

Proposition 28 ([64, 52]) *Let $\mathcal{A}$ be any quantum algorithm that uses no measurements to search a truth assignment for any Boolean function χ. Given the success probability $a \geq 0.25$ of $\mathcal{A}$, $Q(\mathcal{A}, \chi, \phi, \psi)\mathcal{A}|0\rangle$ gives a correct assignment with certainty by setting ψ and ϕ $(0 \leq \psi, \phi < 2\pi)$ to some appropriate values depending on a, where $Q(\mathcal{A}, \chi, \phi, \psi) = -\mathcal{A}F_0(\phi)\mathcal{A}^{-1}F_\chi(\psi)$ such that: $F_\chi(\psi)$ transforms $|x\rangle$ into $e^{i\psi}|x\rangle$ if $\chi(x) = 1$ and $|x\rangle$ if $\chi(x) = 0$, and $F_0(\phi)$ transforms $|x\rangle$ into $e^{i\phi}|x\rangle$ if $x = 0 \cdots 0$ and $|x\rangle$ otherwise.*

The structure of the algorithm is same as that of Algorithm I: the algorithm consists of $(n-1)$ phases. Let S_i be the set of the indices of eligible parties in phase i. Each phase i prepares an uniform superposition of $|S_i|$-bit strings x's, where $x = x_1 \cdots x_{|S_i|}$ and x_l is a bit possessed by party l for $l \in S_i$. The phase then amplifies the amplitude of any basis state inconsistent over S_i so that we can get an inconsistent string over S_i with certainty by measurement. This is possible due to Proposition 28 if every party knows the number $|S_i|$ of eligible parties, since he can exactly compute the probability of getting an inconsistent string by measuring the uniform superposition, which is clearly $1 - \frac{2}{2^{|S_i|}} > 0.25$, and set ψ and ϕ to appropriate values $\psi_{|S_i|}$ and $\phi_{|S_i|}$, respectively.

Actually, every party does not know $|S_i|$. However, we can detour to avoid this issue by using parameter $k = n - i + 1$ instead of $|S_i|$ for each phase i to set ψ and ϕ to ψ_k and ϕ_k, due to the same argument as used in Algorithm I: for any phase i, k is always at least $|S_i|$ (the argument will be described later in a more concrete way). When amplifying the amplitude, $F_\chi(\psi)$ and $F_0(\phi)$ need to be realized in a distributed and anonymous way, i.e., in a way that every party performs the same operation. This can be done as follows. For $F_\chi(\psi)$, every party multiplies the amplitude of any inconsistent basis state by the factor of $e^{i\frac{1}{n}\psi}$, which multiplies it as a whole by the factor of $e^{i\psi}$. To distinguish inconsistent basis states from consistent basis states, every party performs Subroutine A. For $F_0(\phi)$, every party multiplies the amplitude of the all-zero basis state $|00\cdots0\rangle$ by the factor of $e^{i\frac{1}{n}\phi}$, which multiplies it as a whole by the factor of $e^{i\phi}$. To distinguish the all-zero basis state from the other basis states, every party performs a slight modification to Subroutine A, called Subroutine Z (described later).

Once eligible parties share an inconsistent string after measurement, the number of eligible parties can be reduced with certainty by allowing only those parties remain eligible who have the maximum one-bit value among all one-bit values in the string.

As described above, every party uses k instead of $|S_i|$ to set ψ and ϕ to some appropriate values. Hence, the eligible parties may not share an inconsistent state, since k does not necessarily represent $|S_i|$. In this case, the above operations cannot reduce the eligible parties with certainty: the operations may not change

Algorithm Using the Exact Amplitude Amplification

Input: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$, integers n, d

Output: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$

1. For $k := n$ down to 2, do the following:
   - 1.1 Prepare one-qubit quantum registers $\mathbf{R}_0$.
     Call Subroutine EAA with $\mathbf{status}$, n, d, k and $\mathbf{R}_0$ to share an inconsistent state over S_{k-n+1}.
   - 1.3 If $\mathbf{status} =$ "eligible," measure the qubit in $\mathbf{R}_0$ in the $\{|0\rangle, |1\rangle\}$ basis to obtain one-bit value z; otherwise let $z := -1$.
   - 1.4 Perform Subroutine C with z, n, and d to know the maximum value $z_{\max}$ of z over all parties.
     If $z \neq z_{\max}$, let $\mathbf{status} :=$ "ineligible."
2. Output $\mathbf{status}$.

Figure 47: Quantum leader election algorithm using the exact amplitude amplification.

Table 8: The definition of commute operator "$\diamond$."

x	y	$x \diamond y$	x	y	$x \diamond y$	x	y	$x \diamond y$	x	y	$x \diamond y$
0	0	0	1	0	$\times$	$*$	0	0	$\times$	0	$\times$
0	1	$\times$	1	1	$\times$	$*$	1	$\times$	$\times$	1	$\times$
0	$*$	0	1	$*$	$\times$	$*$	$*$	$*$	$\times$	$*$	$\times$
0	$\times$	$\times$	1	$\times$	$\times$	$*$	$\times$	$\times$	$\times$	$\times$	$\times$

S_i. If k equals $|S_i|$ by chance, $|S_i|$ is reduced by at least one (but not to zero), although parties cannot recognize this case. It is clear from this observation that k is always at least $|S_i|$ in each phase i, since k is $n = |S_1|$ before entering the first phase and is decreased by 1 after each phase finishes. It follows that exactly one leader is elected after the last phase.

More precisely, each party l performs the algorithm shown in Figure 47 with parameters "eligible," n, and the number d_l of neighbors of party l. The party who obtains the output "eligible" is the unique leader.

Subroutine Z works in almost the same way as Subroutine A except that it uses commute operator $\diamond$ over set $\mathcal{S} = \{0, 1, *, \times\}$ instead of $\circ$ and, "allzero" and "nonallzero" instead of "eligible" and "ineligible," respectively. Operator $\diamond$ is defined in Table 8.

Theorem 29 *Let $|E|$ and D be the number of edges and the maximum degree of*

Subroutine EAA

Input: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$, integers n, d, k, one-qubit register $\mathbf{R}_0$

Output: one-qubit register $\mathbf{R}_0$

1. If $\mathbf{status} =$ "eligible," prepare the states $(|0\rangle + |1\rangle)/\sqrt{2}$ and $|$"consistent"$\rangle$ in one-qubit registers $\mathbf{R}_0$ and $\mathbf{S}$; otherwise prepare the states $|0\rangle$ and $|$"consistent"$\rangle$ in $\mathbf{R}_0$ and $\mathbf{S}$.
2. To realize $F_\chi(\psi_k)$, perform the next operations.
   - 2.1 Perform Subroutine A with $\mathbf{R}_0$, $\mathbf{S}$, $\mathbf{status}$, n, and d.
   - 2.2 Transform $|r\rangle|s\rangle$ into $e^{i\frac{1}{n}\psi_k}|r\rangle|s\rangle$ if s is "inconsistent," and $|r\rangle|s\rangle$ otherwise, where r and s are the contents of $\mathbf{R}_0$ and $\mathbf{S}$, respectively.
   - 2.3 Invert every step of 2.1 to disentangle $\mathbf{S}$.
3. If $\mathbf{status}$="eligible," apply the Hadamard operator H to the qubit in $\mathbf{R}_0$.
4. To realize $F_0(\phi_k)$, perform the next operations.
   - 4.1 Perform Subroutine Z with $\mathbf{R}_0$, $\mathbf{S}$, $\mathbf{status}$, n, and d, which sets the content of $\mathbf{S}$ to "allzero" if the contents of $\mathbf{R}_0$s are all 0's, and to "nonallzero" otherwise.
   - 4.2 Transform $|r\rangle|s\rangle$ into $e^{i\frac{1}{n}\phi_k}|r\rangle|s\rangle$ if s is "allzero," and $|r\rangle|s\rangle$ otherwise, where r and s are the contents of $\mathbf{R}_0$ and $\mathbf{S}$, respectively.
   - 4.3 Invert every step of 4.1 to disentangle $\mathbf{S}$.
5. If $\mathbf{status}$="eligible," apply the Hadamard operator H to the qubits in $\mathbf{R}_0$.
6. Output $\mathbf{R}_0$.

Figure 48: Subroutine EAA.

the underlying graph, respectively. Given a classical variable $\mathbf{status}$ *initialized to* "eligible" *and the number* n *of parties, the above algorithm exactly elects a unique leader in* $\Theta(n^2)$ *rounds and* $\Theta(Dn^2)$ *time. Each party connected with* d *parties requires* $\Theta(dn^2)$*-qubit communication, and the total communication complexity over all parties is* $\Theta(|E|n^2)$.

Proof. The correctness is obvious from Lemmas 17 and 24, and Proposition 28. As for complexity, Subroutines A, Z and C are clearly dominant. From Lemmas 18 and 25, the theorem follows. □

Remark 5 *The amplitude of desirable states can be amplified to one also by decreasing the success probability of* $\mathcal{A}$ *[52]. Consider preparing* $\alpha|0\rangle+\beta|1\rangle$ *instead of* $(|0\rangle + |1\rangle)/\sqrt{2}$ *in step 1 of Subroutine EAA for some complex numbers* α *and*

β *such that* $|\alpha|^2 + |\beta|^2 = 1$. *Suppose that* $S_{n-k+1} = k$ *(since it is sufficient). In any phase* i*, the probability of getting an inconsistent string by measuring before amplitude amplification, is* $1 - (|\alpha^k|^2 + |\beta^k|^2)$. *Thus, if we set* α *and* β *such that* $1 - (|\alpha^k|^2 + |\beta^k|^2) = 1/4$*, i.e.,* $|\alpha^k|^2 + |\beta^k|^2 = 3/4$*, the amplitude of the inconsistent states can be amplified to exactly one by applying* $Q(\mathcal{A}, \chi, \pi, \pi)$ *to* $\mathcal{A}|0\rangle$. *Such* α *and* β *exist, since* $|\alpha^k|^2 + |\beta^k|^2 = 1 > 3/4$ *if* $\alpha = 0$ *and* $\beta = 1$ *and* $|\alpha^k|^2 + |\beta^k|^2 = 2/2^k < 3/4$ *if* $\alpha = \beta = \frac{1}{\sqrt{2}}$.

4.3 Quantum leader election algorithm II

To reduce the amount of quantum communication, our second algorithm makes use of a classical technique, called *view*, which was introduced by Yamashita and Kameda [195, 196]. However, a naïve application of view incurs exponential classical time/communication complexity. To keep the complexity moderate, we introduce the new technique of *folded view*, with which the algorithm still runs in time/communication polynomial with respect to the number of parties.

4.3.1 View and folded view

First, we briefly review the classical technique, *view*. Let $G = G(V, E)$ be the underlying network topology and let $n = |V|$. Suppose that each party corresponding to node $v \in V$, *or* simply party v, has a value $x_v \in U$ for a finite subset U of the set of integers, and a mapping $X \colon V \to U$ is defined by $X(v) = x_v$. We identify the label of node in G with the value given by X. For each v and port numbering σ, the *view* $T_{G,\sigma,X}(v)$ is a labeled, rooted tree with infinite depth defined recursively as follows: (1) $T_{G,\sigma,X}(v)$ has the root u with label $X(v)$, corresponding to v, (2) for each vertex v_j adjacent to v in G, $T_{G,\sigma,X}(v)$ has vertex u_j labeled with $X(v_j)$, and an edge from root u to u_j with label $\mathsf{label}((v, v_j))$, where $\mathsf{label}((v, v_j)) = (\sigma[v](v, v_j), \sigma[v_j](v, v_j))$, and (3) u_j is the root of $T_{G,\sigma,X}(v_j)$. It should be stressed that v, v_j, u, and u_j are not identifiers of parties and are introduced just for definition. For simplicity, we often use $T_X(v)$ instead of $T_{G,\sigma,X}(v)$, because we usually discuss views of some fixed network with some fixed port numbering. The *view of depth* h *with respect to* v, denoted by $T_X^h(v)$, is the subtree of $T_X(v)$ of depth h with the same root as $T_X(v)$.

If two views $T_X(v)$ and $T_X(v')$ for $v, v' \in V$ are isomorphic (including edge labels and node labels, but ignoring local names of vertices such as u_i), their relation is denoted by $T_X(v) \equiv T_X(v')$. With this relation, V is divided into equivalence classes: v and v' are in the same class if and only if $T_X(v) \equiv T_X(v')$. In [195, 196], it was proved that all classes have the same cardinality for fixed G, σ and X; the cardinality is denoted by $c_{G,\sigma,X}$, or simply c_X (the maximum value of $c_{G,\sigma,X}$ over all port numbering σ is called *symmetricity* $\gamma(G, X)$ and used

to give the necessary and sufficient condition to exactly solve LE_n in anonymous classical networks). We denote the set of non-isomorphic views by $\Gamma_{G,\sigma,X}$, i.e., $\Gamma_{G,\sigma,X} = \{T_{G,\sigma,X}(v) \mid v \in V\}$, and the set of non-isomorphic views of depth h by $\Gamma^h_{G,\sigma,X}$, i.e., $\Gamma^h_{G,\sigma,X} = \{T^h_{G,\sigma,X}(v) \mid v \in V\}$. For simplicity, we may use Γ_X and Γ^h_X instead of $\Gamma_{G,\sigma,X}$ and $\Gamma^h_{G,\sigma,X}$, respectively. We can see that $c_X = n/|\Gamma_X|$, since the number of views isomorphic to $T_X \in \Gamma_X$ is constant over all T_X. For any subset S of U, let $\Gamma_X(S)$ be the maximal subset of Γ_X such that any view $T_X \in \Gamma_X(S)$ has the root labeled with a value in S. Then the number $c_X(S)$ of parties having values in S is $c_X|\Gamma_X(S)| = n|\Gamma_X(S)|/|\Gamma_X|$. When S is a singleton set $\{s\}$, we may use $\Gamma_X(s)$ and $c_X(s)$ instead of $\Gamma_X(\{s\})$ and $c_X(\{s\})$.

To compute $c_X(S)$, every party v constructs $T^{2(n-1)}_X(v)$, and then computes $|\Gamma_X|$ and $|\Gamma_X(S)|$. To construct $T^h_X(v)$, in the first round, every party v constructs $T^0_X(v)$, i.e., the root of $T^h_X(v)$. If every party v_j adjacent to v has $T^{i-1}_X(v_j)$ in the ith round, v can construct $T^i_X(v)$ in the $(i+1)$st round by exchanging a copy of $T^{i-1}_X(v)$ for a copy of $T^{i-1}_X(v_j)$ for each j. By induction, in the $(h+1)$st round, each party v can construct $T^h_X(v)$. It is clear that, for each $v' \in V$, at least one node in $T^{n-1}_X(v)$ corresponds to v', since there is at least one path of length of at most $(n-1)$ between any pair of parties. Thus party v computes $|\Gamma_X|$ and $|\Gamma_X(S)|$ by checking the equivalence of every pair of views that have their roots in $T^{n-1}_X(v)$. The view equivalence can be checked in finite steps, since $T_X(v) \equiv T_X(v')$ if and only if $T^{n-1}_X(v) \equiv T^{n-1}_X(v')$ for $v, v' \in V$ [155]. This implies that $|\Gamma_X|$ and $|\Gamma_X(S)|$ can be computed from $T^{2(n-1)}_X(v)$.

Note that the size of $T^h_X(v)$ is exponential in h, which results in exponential time/communication complexity in n when we construct it if $h = 2(n-1)$. To reduce the time/communication complexity to something bounded by a polynomial, we introduce the new technique called *folded view* by generalizing Ordered Binary Decision Diagrams (OBDD) [54]. A *folded view (f-view) of depth* h is a vertex- and edge-labeled directed acyclic multigraph obtained by merging nodes at the same level in $T^h_X(v)$ into one node if the subtrees rooted at them are isomorphic. An f-view is said to be minimal and denoted by $\widetilde{T}^h_X(v)$ if it obtained by maximally merging nodes of view $T^h_X(v)$ under the above condition. For simplicity, we may call a minimal f-view just an f-view only in this subsection. The number of nodes in each level of an f-view is obviously bounded by n, and thus the total number of nodes in an f-view of depth h is at most hn. Actually, an f-view of depth h can be recursively constructed in a similar manner to view construction without unfolding intermediate f-views. The details will be described in 4.4.

Theorem 30 *If each party has a label of a constant-bit value, every f-view of depth h is constructed in $O(D^2h^2n(\log n)^2)$ time for each party and $O(h)$ rounds with $O(D|E|h^2n\log D)$ bits of classical communication. Once $\widetilde{T}^{2(n-1)}_X(v)$ is constructed, each party can compute $|\Gamma_X|$ and $|\Gamma_X(S)|$ without communication in*

$O(Dn^5 \log n)$ time, where S is any subset of range U of X, and $|E|$ and D are the number of edges and the maximum degree, respectively, of the underlying graph.

4.3.2 The algorithm

As in the previous subsection, we assume that the network is synchronous and each party knows the number n of parties prior to algorithm invocation. Again our algorithm is easily generalized to the asynchronous case. It is also possible to modify our algorithm to work well even if only the upper bound N of the number of parties is given, which is discussed in 4.3.4.

The algorithm consists of two stages, which we call Stages 1 and 2 hereafter. Stage 1 aims to have the n parties share a certain type of entanglement, and thus, this stage requires the parties to exchange quantum messages. In Stage 1, each party performs Subroutine Q $s = \lceil \log n \rceil$ times in parallel to share s pure quantum states $|\phi^{(1)}\rangle, \ldots, |\phi^{(s)}\rangle$ of n qubits. Here, each $|\phi^{(i)}\rangle$ is of the form $(|x^{(i)}\rangle + |\overline{x^{(i)}}\rangle)/\sqrt{2}$ for an n-bit string $x^{(i)}$ and its bitwise negation $\overline{x^{(i)}}$, and the lth qubit of each $|\phi^{(i)}\rangle$ is possessed by the lth party. It is stressed that only one round of quantum communication is necessary in Stage 1.

In Stage 2, the algorithm decides a unique leader among the n parties only by local quantum operations and classical communications with the help of the shared entanglement prepared in Stage 1. This stage consists of at most s phases, each of which reduces the number of eligible parties by at least half. Let $S_i \subseteq \{1, \ldots, n\}$ be the set of all ls such that party l is still eligible just before entering phase i. First every party runs Subroutine $\tilde{\text{A}}$ to decide if state $|\phi^{(i)}\rangle$ is consistent or inconsistent over S_i. Here we assume consistent/inconsistent strings/states as defined in the previous subsection. If state $|\phi^{(i)}\rangle$ is consistent, every party performs Subroutine $\tilde{\text{B}}$, which first transforms $|\phi^{(i)}\rangle$ into the $|S_i|$-cat state $(|0^{|S_i|}\rangle + |1^{|S_i|}\rangle)/\sqrt{2}$ shared only by eligible parties and then calls Subroutine B described in the previous subsection to obtain an inconsistent state over S_i. Each party l then measures his qubits to obtain a label and performs Subroutine $\tilde{\text{C}}$ to find the minority among all labels. The number of eligible parties is then reduced by at least half via minority voting with respect to the labels.

More precisely, each party l having d_l adjacent parties performs Algorithm II described in Figure 49 with parameters "eligible," n, and d_l. The party who obtains output "eligible" is the unique leader.

Subroutine Q:

Subroutine Q is mainly for the purpose of sharing a cat-like quantum state $|\phi\rangle = (|x\rangle + |\overline{x}\rangle)/\sqrt{2}$ for an n-bit random string x. It also outputs a classical string, which is used in Stage 2 for each party to obtain the information on $|\phi\rangle$

Algorithm II

Input: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$, integers n, d

Output: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$

Stage 1:

Let $s := \lceil \log n \rceil$ and prepare one-qubit quantum registers $\mathbf{R}_0^{(1)}, \ldots, \mathbf{R}_0^{(s)}$ and $\mathbf{R}_1^{(1)}, \ldots, \mathbf{R}_1^{(s)}$, each of which is initialized to the $|0\rangle$ state.

Perform s attempts of Subroutine Q in parallel, each with $\mathbf{R}_0^{(i)}$ and d for $1 \leq i \leq s$, to obtain d-bit string $y^{(i)}$ and to share $|\phi^{(i)}\rangle = (|x^{(i)}\rangle + |\overline{x^{(i)}}\rangle)/\sqrt{2}$ of n qubits.

Stage 2:

Let $k := n$.

For $i := 1$ to s, repeat the following:

1. Perform Subroutine $\tilde{\mathrm{A}}$ with $\mathbf{status}$, n, d, and $y^{(i)}$ to obtain its output $\mathbf{consistency}$.
2. If $\mathbf{consistency} =$ "consistent," perform Subroutine $\tilde{\mathrm{B}}$ with $\mathbf{R}_0^{(i)}$, $\mathbf{R}_1^{(i)}$, $\mathbf{status}$, k, n, and d.
3. If $\mathbf{status} =$ "eligible," measure the qubits in $\mathbf{R}_0^{(i)}$ and $\mathbf{R}_1^{(i)}$ in the $\{|0\rangle, |1\rangle\}$ basis to obtain a nonnegative integer z $(0 \leq z \leq 3)$; otherwise set $z := -1$.
   Perform Subroutine $\tilde{\mathrm{C}}$ with $\mathbf{status}$, z, n, and d to compute nonnegative integers z_{minor} and $c_{z_{\mathrm{minor}}}$.
4. If $z \neq z_{\mathrm{minor}}$, let $\mathbf{status} :=$ "ineligible."
   Let $k := c_{z_{\mathrm{minor}}}$.
5. If $k = 1$, terminate and output $\mathbf{status}$.

Figure 49: Quantum leader election algorithm II.

via just classical communication. This subroutine can be performed in parallel, and thus Stage 1 involves only one round of quantum communication. First each party prepares the state $(|0\rangle + |1\rangle)/\sqrt{2}$ in a quantum register and computes the XOR of the contents of his own and each adjacent party's registers. The party then measures the qubits whose contents are the results of the XORs. This results in the state of the form $(|x\rangle + |\overline{x}\rangle)/\sqrt{2}$. Figure 50 gives the precise description of Subroutine Q.

The next two lemmas are for correctness and complexity.

Lemma 31 *For an n-party distributed system, suppose that every party l calls Subroutine Q with a one-qubit register whose content is initialized to $|0\rangle$ and the number d_l of his neighbors as input $\mathbf{R}_0$ and d, respectively. After performing Sub-*

Subroutine Q

Input: a one-qubit quantum register $\mathbf{R}_0$, an integer d

Output: a one-qubit quantum register $\mathbf{R}_0$, a binary string y of length d

1. Prepare $2d$ one-qubit quantum registers $\mathbf{R}'_1, \ldots, \mathbf{R}'_d$ and $\mathbf{S}_1, \ldots, \mathbf{S}_d$, each of which is initialized to the $|0\rangle$ state.
2. Generate the $(d+1)$-cat state $(|0^{d+1}\rangle + |1^{d+1}\rangle)/\sqrt{2}$ in registers $\mathbf{R}_0$, $\mathbf{R}'_1, \ldots, \mathbf{R}'_d$.
3. Exchange the qubit in $\mathbf{R}'_i$ with the party connected via port i for $1 \leq i \leq d$ (i.e., the original qubit in $\mathbf{R}'_i$ is sent via port i, and the qubit received via that port is newly set in $\mathbf{R}'_i$).
4. Set the content of $\mathbf{S}_i$ to $x_0 \oplus x_i$, for $1 \leq i \leq d$, where x_0 and x_i denote the contents of $\mathbf{R}_0$ and $\mathbf{R}'_i$, respectively.
5. Measure the qubit in $\mathbf{S}_i$ in the $\{|0\rangle, |1\rangle\}$ basis to obtain a bit y_i, for $1 \leq i \leq d$.
   Set $y := y_1 \cdots y_d$.
6. Apply CNOT controlled by the content of $\mathbf{R}_0$ and targeted to the content of each $\mathbf{R}'_i$ for $i = 1, 2, \ldots, d$ to disentangle $\mathbf{R}'_i$s.
7. Output $\mathbf{R}_0$ and y.

Figure 50: Subroutine Q.

routine Q, all parties share $(|x\rangle + |\overline{x}\rangle)/\sqrt{2}$ with certainty, where x is a randomly chosen n-bit string.

Proof. After step 2 of Subroutine Q, the system state, i.e., the state in $\mathbf{R}_0$'s, $\mathbf{R}'_1$'s, $\ldots$, $\mathbf{R}'_d$'s and $\mathbf{S}_1$'s, $\ldots$, $\mathbf{S}_d$'s of all parties, is the tensor product of the states of all parties as described by formula (1). Notice that the state in $\mathbf{R}_0$'s and $\mathbf{R}'_1$'s, $\ldots$, $\mathbf{R}'_d$'s of all parties is the uniform superposition of some basis states in an orthonormal basis of $2^{\sum_{l=1}^{n}(d_l+1)}$-dimensional Hilbert space: the basis states correspond one-to-one to n-bit integers a and each of them is of the form $|a_1\rangle^{\otimes(d_1+1)} \otimes \cdots \otimes |a_n\rangle^{\otimes(d_n+1)}$, where a_l is the lth bit of the binary expression of a and a_l is the content of $\mathbf{R}_0$ of party l. If we focus on the lth party's part of the basis state corresponding to a, step 3 transforms $|a_l\rangle^{\otimes(d_l+1)}$ to $|a_l\rangle \left(\bigotimes_{j=1}^{d_l} |a_{l_j}\rangle\right)$, where party l is connected to party l_j via port j. More precisely, step 3 transforms the system state into the state as described in formula (2). After step 4, we have the state of formula (3). Then every party l measures the last d_l registers $\mathbf{S}_i$'s at

step 5.

$$\bigotimes_{l=1}^{n}|0\rangle|0^{d_l}\rangle|0^{d_l}\rangle \rightarrow \bigotimes_{l=1}^{n}\frac{|0^{d_l+1}\rangle+|1^{d_l+1}\rangle}{\sqrt{2}}|0^{d_l}\rangle \quad (1)$$

$$\rightarrow \frac{1}{\sqrt{2^n}}\sum_{a=0}^{2^n-1}\bigotimes_{l=1}^{n}\left\{|a_l\rangle\left(\bigotimes_{j=1}^{d_l}|a_{l_j}\rangle\right)|0^{d_l}\rangle\right\} \quad (2)$$

$$\rightarrow \frac{1}{\sqrt{2^n}}\sum_{a=0}^{2^n-1}\bigotimes_{l=1}^{n}\left\{|a_l\rangle\left(\bigotimes_{j=1}^{d_l}|a_{l_j}\rangle\right)\left(\bigotimes_{j=1}^{d_l}|a_l\oplus a_{l_j}\rangle\right)\right\} \quad (3)$$

Claim 32 *Suppose that every party l obtained measurement results $y(l) = y_1(l)y_2(l)\cdots y_{d_l}(l)$ of d_l bits where $y_j(l)\in\{0,1\}$. There are exactly two binary strings $a = a_1a_2\cdots a_n$ that satisfy equations $a_l\oplus a_{l_j} = y_j(l)(l=1,\ldots,n, j=1,\ldots,d_l)$. If the binary strings are A and $\overline{A}$, then $\overline{A}$ is the bit-wise negation of A.*

Proof. We call binary strings a "solutions" of the equations. By the definition, there is at least one solution. If A is such a string, obviously its bit-wise negation $\overline{A}$ is also a solution by the fact that $a_i\oplus a_j = \overline{a_i}\oplus\overline{a_j}$ for $1\le i,j\le n$. We will prove that there is the unique solution such that $a_1 = 0$. It follows that there is the unique solution such that $a_1 = 1$ since the bitwise negation of a solution is also a solution. This completes the proof.

Let $\{V_0, V_1, \ldots, V_p\}$ be the partition of the set V of the indices of parties such that $V_0 = \{1\}$ and $V_i = \mathsf{Adj}(\bigcup_{m=0}^{i-1}V_m)\setminus\bigcup_{m=0}^{i-1}V_m$, where p is the maximum length of the shortest path from party 1 to party l over all l, and $\mathsf{Adj}(V')$ for a set $V'\subseteq V$ is the set of neighbors of the parties in V'.

Equations $a_l\oplus a_{l_j} = y_j(l)$ are equivalent to $a_{l_j} = y_j(l)\oplus a_l$ $(l=1,\ldots,n, j=1,\ldots,d_l)$. Assume that $a_1 = 0$. For all l in V_1, a_l is uniquely determined by the equations. Similarly, if a_l is fixed for all l in $\bigcup_{m=0}^{i-1}V_m$, a_l is uniquely determined for all l in V_i. Since the underlying graph of the distributed system is connected, a_l is uniquely determined for all l. □

From the above claim, we get the superposition of two basis states corresponding to A and its bit-wise negation $\overline{A}$ after step 5 as described by formula (4), where A_l is the lth bit of A. Step 6 transforms the state into that represented by formula (5), in which registers $\mathbf{R}_i'$'s of all parties are disentangled because of $|A_l\oplus A_{l_j}\rangle = |\overline{A_l}\oplus\overline{A_{l_j}}\rangle$. Thus, $\mathbf{R}_0$'s is in the state of $(|x\rangle+|\overline{x}\rangle)/\sqrt{2}$.

$$\frac{1}{\sqrt{2}}\bigotimes_{l=1}^{n}\left(|A_l\rangle\bigotimes_{j=1}^{d_l}|A_{l_j}\rangle\right)+\frac{1}{\sqrt{2}}\bigotimes_{l=1}^{n}\left(|\overline{A_l}\rangle\bigotimes_{j=1}^{d_l}|\overline{A_{l_j}}\rangle\right) \quad (4)$$

$$\rightarrow \frac{1}{\sqrt{2}}\bigotimes_{l=1}^{n}\left(|A_l\rangle\bigotimes_{j=1}^{d_l}|A_l\oplus A_{l_j}\rangle\right)+\frac{1}{\sqrt{2}}\bigotimes_{l=1}^{n}\left(|\overline{A_l}\rangle\bigotimes_{j=1}^{d_l}|\overline{A_l}\oplus\overline{A_{l_j}}\rangle\right) \quad (5)$$

□

Lemma 33 *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph of an n-party distributed system. Subroutine Q takes $O(D)$ time for each party and one round, and requires $2|E|$-qubit communication.*

Proof. Each party l performs just one-round communication of d_l qubits. The local computations can be done in time linear in d_l. □

Subroutine Ã:

Suppose that, after Subroutine Q, n-qubit state $|\phi\rangle = (|x\rangle + |\overline{x}\rangle)/\sqrt{2}$ is shared by the n parties such that the lth party has the lth qubit. Let x_l be the lth bit of x, and let X and $\overline{X}$ be mappings defined by $X(v) = x_l$ and $\overline{X}(v) = \overline{x_l}$ for each l, respectively, where $v \in V$ represents the node corresponding to the lth party in the underlying graph $G = G(V, E)$ of the network topology. For any v in V, let $W[v] : V \to \{0, 1\} \times \{\text{“eligible”}, \text{“ineligible”}\}$ be the mapping defined as $(Y[v], Z)$, where $Y[v]$ is X if $X(v) = 0$ and $\overline{X}$ otherwise, and $Z : V \to \{\text{“eligible”}, \text{“ineligible”}\}$ maps $v \in V$ to the value of **status** possessed by the party corresponding to v. We denote $(\overline{Y[v]}, Z)$ by $\overline{W[v]}$.

Subroutine Ã checks the consistency of $|\phi\rangle$, but in quite a different way from Subroutine A. Every party l constructs the folded view $\widetilde{T}^{n-1}_{W[v]}(v)$ by using the output y of Subroutine Q. The folded view is constructed by the f-view construction algorithm in Figure 59 with slight modification; the modification is required since mapping $W[v]$ is not necessarily common over all parties v. The construction is still only by classical communication. By checking if the nodes for eligible parties in the folded view have the same labels, Subroutine Ã can decide whether $|\phi\rangle$ is consistent or not over the set of the indices of eligible parties. Figure 51 gives the precise description of Subroutine Ã. The next lemmas present the correctness and complexity of Subroutine Ã.

Lemma 34 *Suppose that the n parties share n-qubit cat-like state $(|x\rangle+|\overline{x}\rangle)/\sqrt{2}$, where x is n-bit string $X(v_1)X(v_2)\cdots X(v_n)$ for $v_i \in V$ and $\overline{x}$ is the bitwise negation of x. Let S be the set of the indices of the parties among the n parties whose variable* **status** *is “*eligible*,” and let $v \in V$ be the corresponding node of party l. If every party l runs Subroutine Ã with the following objects as input:*

- *a classical variable* **status** $\in \{\text{“eligible”}, \text{“ineligible”}\}$
- *n and the number d_l of the neighbors of party l,*

Subroutine $\tilde{\mathrm{A}}$

Input: a classical variable $\mathbf{status} \in \{\text{“eligible”}, \text{“ineligible”}\}$, integers n, d, a binary string y of length d

Output: a classical variable $\mathbf{consistency} \in \{\mathsf{consistent}, \mathsf{inconsistent}\}$

1. Set $\widetilde{T}^0_{W[v]}(v)$ to a node labeled with $(0, \mathbf{status})$,
   where $W[v] : V \to \{0,1\} \times \{\text{“eligible”}, \text{“ineligible”}\}$ be the mapping defined as $(Y[v], Z)$, $Y[v]$ is X if $X(v) = 0$ and $\overline{X}$ otherwise, and Z is the underlying mapping naturally induced by the values of $\mathbf{status}$.
2. For $i := 1$ to $(n-1)$, do the following:
   2.1 Send $\widetilde{T}^{i-1}_{W[v]}(v)$ and receive $\widetilde{T}^{i-1}_{W[v_j]}(v_j)$ via port j, for $1 \le j \le d$, where node v_j corresponds to the party connected via port j.
   2.2 If the jth bit y_j of y is 1, transform $\widetilde{T}^{i-1}_{W[v_j]}(v_j)$ into $\widetilde{T}^{i-1}_{\overline{W[v_j]}}(v_j)$ by negating the first element of every node label for $1 \le j \le d$,
   where $\overline{W[v_j]}$ represents $(\overline{Y[v_j]}, Z)$.
   2.3 Set the root of $\widetilde{T}^i_{W[v_j]}(v)$ to the node labeled with $(0, \mathbf{status})$.
   Set the jth child of the root of $\widetilde{T}^i_{W[v]}(v)$ to $\widetilde{T}^{i-1}_{\overline{W[v_j]}}(v_j)$, for $1 \le j \le d$.
   For every level of $\widetilde{T}^i_{W[v]}(v)$, merge nodes at that level into one node if the views rooted at them are isomorphic.
3. If both label $(0, \text{“eligible”})$ and label $(1, \text{“eligible”})$ are found among the node labels in $\widetilde{T}^{n-1}_{W[v]}(v)$, let $\mathbf{consistency} :=$ “inconsistent”; otherwise let $\mathbf{consistency} :=$ “consistent.”
4. Output $\mathbf{consistency}$.

Figure 51: Subroutine $\tilde{\mathrm{A}}$.

- *a binary string $y = y_1 \cdots y_{d_l}$ of length d_l such that $y_j = X(v) \oplus X(v_j)$ for $j = 1, \ldots, d_l$ where v_j is the jth adjacent node of v,*

Subroutine $\tilde{A}$ outputs a classical valuable $\mathbf{consistency}$*, which has value* “consistent” *if $(|x\rangle + |\overline{x}\rangle)/\sqrt{2}$ is consistent over S, and* “inconsistent” *otherwise.*

Proof. It will be proved later that steps 1 and 2 construct an f-view of depth $(n-1)$ for mapping either (X, Z) or $(\overline{X}, Z)$. Since the f-view is made by merging those nodes at the same depth which are the roots of isomorphic views, the f-view contains at least one node that has the same label as $(X(v), Z(v))$ or $(\overline{X}(v), Z(v))$ for any $v \in V$. Once the f-view is constructed, every party can know whether X is constant over all $l \in S$ or not in step 3 by checking the labels including “eligible.”

Notice that no party needs to know for which mapping of (X, Z) or $(\overline{X}, Z)$ it has constructed the f-view.

In what follows, we prove that steps 1 and 2 construct an f-view for mapping either X or $\overline{X}$. The proof is by induction on depth i of the f-view. Clearly, step 1 generates $\widetilde{T}^0_{W[v]}(v)$. Assume that every party l' has constructed $\widetilde{T}^{i-1}_{W[v']}(v')$ where node v' represents party l'. In order to construct $\widetilde{T}^i_{W[v]}(v)$, party l needs $\widetilde{T}^{i-1}_{W[v]}(v_j)$ for every node v_j adjacent to v. Although $W[v]$ is not always identical to $W[v_j]$, we can transform $\widetilde{T}^{i-1}_{W[v_j]}(v_j)$ to $\widetilde{T}^{i-1}_{W[v]}(v_j)$. Since y_j is equal to $X(v) \oplus X(v_j) = \overline{X}(v) \oplus \overline{X}(v_j)$, each of X and $\overline{X}$ gives the same value for v and v_j if and only if $y_j = 0$. This fact and $Y[v](v) = Y[v_j](v_j) = 0$ imply that $Y[v]$ is identical to $Y[v_j]$ if and only if $y_j = 0$. It follows that, if $y_j = 0$, $\widetilde{T}^{i-1}_{W[v_j]}(v_j)$ is isomorphic to $\widetilde{T}^{i-1}_{W[v]}(v_j)$, and otherwise $\widetilde{T}^{i-1}_{\overline{W[v_j]}}(v_j)$ is isomorphic to $\widetilde{T}^{i-1}_{W[v]}(v_j)$. In the latter case, the party corresponding to v negates the first elements of all node labels in $\widetilde{T}^{i-1}_{W[v_j]}(v_j)$ to obtain $\widetilde{T}^{i-1}_{\overline{W[v_j]}}(v_j)$. Thus step 3 can construct $\widetilde{T}^i_{Y[v]}(v)$. This completes the proof. □

Lemma 35 *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph of an n-party distributed system. Subroutine $\tilde{A}$ takes $O(D^2n^3(\log n)^2)$ time for each party, $O(n)$ rounds and requires classical communication of $O(D|E|n^3 \log D)$ bits.*

Proof. Steps 1 and 2 is basically the f-view construction algorithm in Figure 59 except step 2.2; this step takes $O(Dn^2)$ time since an f-view of depth $O(n)$ has $O(Dn^2)$ edges. Thus, steps 1 and 2 take $O(D^2n^3(\log n)^2)$ time for each party, $O(n)$ rounds and exchanges $O(D|E|n^3 \log D)$ bits by Theorem 30. Step 3 takes $O(Dn^2)$ time. □

Subroutine $\tilde{\mathbf{B}}$:

Suppose that $|\phi\rangle = (|x\rangle + |\overline{x}\rangle)/\sqrt{2}$ shared by the n parties is consistent over the set S of the indices of eligible parties. Subroutine $\tilde{\mathrm{B}}$ is for the purpose of transforming $|\phi\rangle$ into an inconsistent state over S. Let k be $|S|$. First every ineligible party measures its qubit in the $\{|+\rangle, |-\rangle\}$ basis, where $|+\rangle$ and $|-\rangle$ denote $(|0\rangle + |1\rangle)/\sqrt{2}$ and $(|0\rangle - |1\rangle)/\sqrt{2}$, respectively. As a result, the state shared by the eligible parties is either $\pm(|0^k\rangle + |1^k\rangle)/\sqrt{2}$ or $\pm(|0^k\rangle - |1^k\rangle)/\sqrt{2}$. The state $\pm(|0^k\rangle - |1^k\rangle)/\sqrt{2}$ is shared if and only if the number of ineligible parties that measured $|-\rangle$ is odd, as proved in Lemma 38. In this case, every eligible party applies unitary operator W_k to its qubit to transform the state into

Subroutine $\tilde{\mathrm{B}}$

Input: one-qubit quantum registers $\mathbf{R}_0, \mathbf{R}_1$, a classical variable $\mathbf{status} \in \{\text{“eligible”}, \text{“ineligible”}\}$, integers k, n, d

Output: one-qubit quantum registers $\mathbf{R}_0, \mathbf{R}_1$

1. Let $w := 0$.
2. If $\mathbf{status} =$ “ineligible,” measure the qubit in $\mathbf{R}_0$ in the $\{|+\rangle, |-\rangle\}$ basis. If this results in $|-\rangle$, let $w := 1$.
3. Construct f-view $\widetilde{T}_W^{(2n-1)}(v)$ to count the number p of parties with $w = 1$, where W is the underlying mapping naturally induced by the w values of all parties.
4. If p is odd and $\mathbf{status} =$ “eligible,” apply W_k to the qubit in $\mathbf{R}_0$.
5. If $\mathbf{status} =$ “eligible,” perform Subroutine B with $\mathbf{R}_0$, $\mathbf{R}_1$ and k.
6. Output quantum registers $\mathbf{R}_0$ and $\mathbf{R}_1$.

Figure 52: Subroutine $\tilde{\mathrm{B}}$.

$\pm(|0^k\rangle + |1^k\rangle)/\sqrt{2}$, where the family $\{W_k\}$ of unitary operators is defined by

$$W_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{k}} \end{pmatrix}.$$

Again let v denote the node corresponding to the party that invokes the subroutine. Figure 52 gives the precise description of Subroutine $\tilde{\mathrm{B}}$. The correctness and complexity of the subroutine are described in Lemmas 36 and 37, respectively.

Lemma 36 *Suppose that the n parties share n-qubit cat-like state $|\phi\rangle = (|x\rangle + |\overline{x}\rangle)/\sqrt{2}$, where x is any n-bit string that is consistent over S, and $\overline{x}$ is the bitwise negation of x. If each party l runs Subroutine $\tilde{B}$ with the following objects as input:*

- *one-qubit register $\mathbf{R}_0$, which stores one of n qubits in state $|\phi\rangle$,*
- *one-qubit register $\mathbf{R}_1$, which is initialized to $|0\rangle$,*
- *a classical variable $\mathbf{status}$, the value of which is “eligible” if l is in S and “ineligible” otherwise,*
- *integers k, n, and the number d_l of neighbors of party l,*

Subroutine $\tilde{B}$ outputs two one-qubit registers $\mathbf{R}_0, \mathbf{R}_1$ such that, given k is equal to $|S|$, the qubits in the registers satisfy the conditions:

- *the $2k$ qubits possessed by all parties l' for $l' \in S$ are in an inconsistent state over S,*
- *the $2(n-k)$ qubits possessed by all parties l' for $l' \notin S$ are in a classical state (as a result of measurement).*

Proof. Lemma 38 guarantees that, after step 2, the eligible parties (i.e., the parties who have $\mathbf{status}$ = "eligible") share $(|0^k\rangle + |1^k\rangle)/\sqrt{2}$ ($(|0^k\rangle - |1^k\rangle)/\sqrt{2}$) if the number of those parties is even (respectively, odd) who have measured $|-\rangle$. When the eligible parties share $(|0^k\rangle - |1^k\rangle)/\sqrt{2}$, step 4 is performed to transform $(|0^k\rangle - |1^k\rangle)/\sqrt{2}$ into $(|0^k\rangle + |1^k\rangle)/\sqrt{2}$. Due to Lemma 22, only the eligible parties share an inconsistent state over S after step 5. This completes the proof. □

Lemma 37 *Let $|E|$ and D be the number of edges of the underlying graph of an n-party distributed system. Subroutine $\tilde{B}$ takes $O(Dn^5 \log n)$ time for each party, $O(n)$ rounds and requires $O(D|E|n^3 \log D)$-bit communication.*

Proof. Since Subroutine B takes $O(1)$ time and does no communication, step 3 is dominant. The proof is completed by Theorem 30. □

Lemma 38 *Let S be an arbitrary subset of $\{1, 2, \ldots n\}$ parties such that $|S| = k$. Suppose that n parties share n-qubit cat-like state $(|x\rangle + |\overline{x}\rangle)/\sqrt{2}$, where x is any n-bit string that is consistent over S, and $\overline{x}$ is the bitwise negation of x. If every party l for $l \notin S$ measures his qubit with respect to the Hadamard basis $(|+\rangle, |-\rangle)$, the resulting state is $(|0^k\rangle + |1^k\rangle)/\sqrt{2}$ ($(|0^k\rangle - |1^k\rangle)/\sqrt{2}$) when the number of those parties is even (respectively, odd) who have measured $|-\rangle$.*

Proof. For an m-bit integer $z = z_1 z_2 \cdots z_m$ where $z_i \in \{0, 1\}$, let $z_{[i,m]}$ be the substring $z_i z_{i+1} \cdots z_m$ of z for $1 \leq i \leq m$. Thus, $(|x\rangle + |\overline{x}\rangle)/\sqrt{2} = (|x_{[1,n]}\rangle + |\overline{x_{[1,n]}}\rangle)/\sqrt{2}$. For simplicity, we also use $z_{[m]}$ instead of $z_{[1,m]}$, and z_m instead of $z_{[m,m]}$.

For any integer m $(2 \leq m \leq n)$, we denote m-qubit states $(|x_{[m]}\rangle + |\overline{x_{[m]}}\rangle)/\sqrt{2}$ and $(|x_{[m]}\rangle - |\overline{x_{[m]}}\rangle)/\sqrt{2}$ by $C^+(m)$ and $C^-(m)$, respectively. For any integer m $(1 \leq m \leq n)$, let $HW(m)$ be the Hamming weight of the binary expression of m (i.e., the number of bits of value 1 in the binary expression of m), and let $|\widetilde{z}_{[m]}\rangle$ be $H^{\otimes m}|z_{[m]}\rangle$ for the 2-dimensional Hadamard operator H. For example,

$$\frac{|0_{[1]}\rangle - |1_{[1]}\rangle}{\sqrt{2}} \otimes \frac{|0_{[1]}\rangle + |1_{[1]}\rangle}{\sqrt{2}} = |\widetilde{1}_{[1]}\rangle \otimes |\widetilde{0}_{[1]}\rangle = |\widetilde{2}_{[2]}\rangle.$$

We will prove the next equation by induction on k in decreasing order: for any k $(0 \leq k \leq n-1)$

$$\begin{aligned} C^+(n) &= \frac{1}{2^{(n-k)/2}} \sum_{i_{[n-k]} \in E_{n-k}} (-1)^{HW(x_{[k+1,n]} \wedge i_{[n-k]})} C^+(k) |\widetilde{i}_{[n-k]}\rangle \\ &\quad + \frac{1}{2^{(n-k)/2}} \sum_{j_{[n-k]} \in O_{n-k}} (-1)^{HW(x_{[k+1,n]} \wedge j_{[n-k]})} C^-(k) |\widetilde{j}_{[n-k]}\rangle, \quad (6) \end{aligned}$$

where $\wedge$ is the bitwise AND operation, E_m is the maximal subset of $\{0, 1, \ldots, 2^m - 1\}$ such that, for any $i \in E_m$, $HW(i) = 0 \pmod 2$, O_m is defined in the same way as E_m, except for $HW(i) = 1 \pmod 2$ instead of $HW(i) = 0 \pmod 2$. The equation implies that, if there is an even number of parties which have measured $|-\rangle$, the resulting state is $C^+(k)$, and otherwise the state is $C^-(k)$, up to global phases. This completes the proof.

For any integer m $(m \geq 3)$, we can see by some calculations that

$$C^+(m) = \{C^+(m-1)|\widetilde{0}_{[1]}\rangle + (-1)^{x_m} C^-(m-1)|\widetilde{1}_{[1]}\rangle\}/\sqrt{2}, \quad (7)$$

$$C^-(m) = \{C^-(m-1)|\widetilde{0}_{[1]}\rangle + (-1)^{x_m} C^+(m-1)|\widetilde{1}_{[1]}\rangle\}/\sqrt{2}. \quad (8)$$

Eq. (6) holds for $k = n-1$ by setting m in eq. (7) to n. Assume that eq. (6) holds for k. By eqs. (7) and (8) for $m = k$, eq. (6) can be written as

$$\begin{aligned} C^+(n) &= \frac{1}{2^{(n-k)/2}} \frac{1}{\sqrt{2}} \left\{ C^+(k-1)|\widetilde{0}_{[1]}\rangle + (-1)^{x_k} C^-(k-1)|\widetilde{1}_{[1]}\rangle \right\} \\ &\quad \times \sum_{i_{[n-k]} \in E_{n-k}} (-1)^{HW(x_{[k+1,n]} \wedge i_{[n-k]})} |\widetilde{i}_{[n-k]}\rangle \\ &\quad + \frac{1}{2^{(n-k)/2}} \frac{1}{\sqrt{2}} \left\{ C^-(k-1)|\widetilde{0}_{[1]}\rangle + (-1)^{x_k} C^+(k-1)|\widetilde{1}_{[1]}\rangle \right\} \\ &\quad \times \sum_{j_{[n-k]} \in O_{n-k}} (-1)^{HW(x_{[k+1,n]} \wedge j_{[n-k]})} |\widetilde{j}_{[n-k]}\rangle. \end{aligned}$$

Equivalently,

$$\begin{aligned} C^+(n) &= \frac{1}{2^{(n-k+1)/2}} \sum_{i_{[n-k+1]} \in E_{n-k+1}} C^+(k-1) |\widetilde{i}_{[n-k+1]}\rangle \\ &\quad + \frac{1}{2^{(n-k+1)/2}} \sum_{j_{[n-k+1]} \in O_{n-k+1}} C^-(k-1) |\widetilde{j}_{[n-k+1]}\rangle. \end{aligned}$$

In the above, we use the next basic relations:

$$E_{n-k+1} = \left(\bigcup_{i_{[n-k]} \in E_{n-k}} \{0_{[1]} i_{[n-k]}\} \right) \cup \left(\bigcup_{j_{[n-k]} \in O_{n-k}} \{1_{[1]} j_{[n-k]}\} \right),$$

Subroutine $\tilde{\mathrm{C}}$

Input: integers $z \in \{-1, 0, 1, 2, 3\}$, n, d

Output: integers z_{minor}, $c_{z_{\mathrm{minor}}}$

1. Construct f-view $\widetilde{T}_Z^{(2n-1)}(v)$, where Z is the underlying mapping naturally induced by the z values of all parties.
2. For $i := 0$ to 3, count the number c_i of parties having a value $z = i$ using $\widetilde{T}_Z^{(2n-1)}(v)$.
   If $c_i = 0$, let $c_i := n$.
3. Let $z_{\mathrm{minor}} \in \{m \mid c_m = \min_{0 \le i \le 3} c_i\}$.
4. Output z_{minor} and $c_{z_{\mathrm{minor}}}$.

Figure 53: Subroutine $\tilde{\mathrm{C}}$

$$O_{n-k+1} = \left(\bigcup_{i_{[n-k]} \in E_{n-k}} \{1_{[1]} i_{[n-k]}\} \right) \cup \left(\bigcup_{j_{[n-k]} \in O_{n-k}} \{0_{[1]} j_{[n-k]}\} \right),$$

$$HW(x_{[k+1,n]} \wedge i_{[n-k]}) = HW(x_{[k,n]} \wedge 0_{[1]} i_{[n-k]}),$$

$$HW(x_{[k+1,n]} \wedge i_{[n-k]}) + x_k = HW(x_{[k,n]} \wedge 1_{[1]} i_{[n-k]}).$$

Hence, eq. (6) holds for any k $(< n)$, which the lemma follows. □

Subroutine $\tilde{\mathrm{C}}$:

Suppose that each party l has value z_l. Subroutine C is a classical algorithm that computes value z_{minor} such that the number of parties with value z_{minor} is non-zero and the smallest among all possible non-negative z_l values. It is stressed that the number of parties with value z_{minor} is at most half of the number of parties having non-negative z_l values, and that the parties having non-negative z_l values are eligible as described in Figure 49. Figure 53 gives the precise description of Subroutine $\tilde{\mathrm{C}}$.

The next two lemmas give the correctness and complexity of Subroutine $\tilde{\mathrm{C}}$.

Lemma 39 *Suppose that each party l among n parties has an integer $z_l \in \{-1, 0, 1, 2, 3\}$. If every party l runs Subroutine $\tilde{C}$ with z_l, n and the number d_l of neighbors as input, Subroutine $\tilde{C}$ outputs $z_{\mathrm{minor}} \in \{z_1, \ldots, z_n\} \setminus \{-1\}$, and $c_{z_{\mathrm{minor}}}$ such that the number $c_{z_{\mathrm{minor}}}$ of parties having z_{minor} is not more than that of parties having any other z_l (ties are broken arbitrarily).*

Proof. The first line of step 2 in Figure 53 counts the number c_i of parties having i as z for each $i \in \{0, 1, 2, 3\}$ by using f-view. Since $c_i = 0$ implies $z_{\text{minor}} \neq i$, c_i is set to n so that i cannot be selected as z_{minor} in step 3. Thus, z_{minor} is selected among $\{z_1, \ldots, z_n\} \setminus \{-1\}$. □

Lemma 40 *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph of an n-party distributed system. Subroutine $\tilde{C}$ takes $O(Dn^5 \log n)$ time for each party, $O(n)$ rounds and requires $O(D|E|n^3 \log D)$-bit communication.*

Proof. Steps 1 and 2 are dominant. The proof is completed by Theorem 30. □

4.3.3 Complexity analysis

Theorem 41 *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph, respectively. Given the number n of parties, Algorithm II exactly elects a unique leader in $O(Dn^5(\log n)^2)$ time and $O(n \log n)$ rounds of which only the first round requires quantum communication. The total communication complexity over all parties is $O(D|E|n^3(\log D) \log n)$ which includes the communication of only $O(|E| \log n)$ qubits.*

Proof. Lemma 31 guarantees that Stage 1 works correctly. We will prove that steps 1 to 5 of Stage 2 decrease the number of eligible parties by at least half, without eliminating all eligible parties, if there are at least two eligible parties. This directly leads to the correctness of Algorithm II, since $s := \lceil \log n \rceil$.

The proof is by induction on phase number i. At the beginning of the first phase, k represents obviously the number of eligible parties. Next we prove that if k is equal to the number of eligible parties immediately before entering phase i, steps 1 to 5 decrease the number of eligible parties by at least half without eliminating all such parties, and set k to the number of the updated eligible parties. By Lemmas 34 and 36 and the assumption that k is the number of eligible parties, only eligible parties share an inconsistent state with certainty after steps 1 and 2. Thus, it is impossible that all eligible parties get the same value by measurement at step 3. Subroutine $\tilde{\text{C}}$ correctly computes z_{minor} and the number $c_{z_{\text{minor}}}$ as proved in Lemma 39. Hence, step 4 reduces eligible parties by at least half with certainty and sets k to the number of the updated eligible parties.

In what follows, we will analyze the complexity of Algorithm II. By Lemma 33, Stage 1 takes $O(D)$ time for each party and one round, and requires $O(|E| \log n)$-qubit communication. Stage 2 iterates Subroutines $\tilde{\text{A}}$, $\tilde{\text{B}}$ and $\tilde{\text{C}}$ at most $O(\log n)$ times. By Lemmas 35, 37 and 40, Subroutines $\tilde{\text{A}}$, $\tilde{\text{B}}$ and $\tilde{\text{C}}$ take $O(n)$ rounds, $O(Dn^5 \log n)$ time and require $O(D|E|n^3 \log D)$ classical bit

communication for each iteration. Hence, Stage 2 takes $O(n \log n)$ rounds and $O(Dn^5(\log n)^2)$ time, and requires $O(D|E|n^3(\log D)\log n)$ classical bit communication. This completes the proof. □

Remark 6 *In fact, we can improve the number of rounds and the communication complexity in the second stage (and, thus, those of Algorithm II) at the cost of a constant multiplicative factor to the time complexity per phase when n holds a certain condition described later; we modify Subroutine $\tilde{C}$ as follows: by using constructed f-views $\widetilde{T}_Z^{2(n-1)}(v)$, parties select a certain common equivalence class of $(n-1)$-depth views rooted at nodes having $i \in \{0,1,2,3\}$, and then every party sets* **status** *to "*ineligible*," if his view does not belong to the equivalence class, and sets k to the number of the updated eligible parties, i.e., the cardinality of the equivalence class. To carry out these operations on f-view $\widetilde{T}_Z^{2(n-1)}(v)$, every party first selects in $O(Dn^5 \log n)$ time an f-view that can be obtained from a subtree with root having label $i \in \{0,1,2,3\}$ of depth $(n-1)$ of $T_Z^{2(n-1)}(v)$, i.e., a subgraph of $\widetilde{T}_Z^{2(n-1)}(v)$, instead of an equivalence class. Every party then checks if the f-view and $\widetilde{T}_Z^{(n-1)}(v)$ can be obtained from isomorphic views; this can be done by simultaneously traversing both f-views as shown in Figure 62 in time $O(Dn^2 \log n)$ (by Lemma 56 with constant L). The cardinality of the equivalence class can be computed by using the view counting algorithm in Figure 60; this takes $O(Dn^5 \log n)$ time due to Lemma 58. Thus, the modification to Subroutine $\tilde{C}$ increases the time complexity by a constant multiplicative factor.*

The f-view of some representative view in the equivalence class can be found without unfolding f-view $\widetilde{T}_Z^{2(n-1)}(v)$ in $O(n^5 D \log n)$ time as follows. First, every party computes the set of f-views obtained from non-isomorphic subtrees of depth $(n-1)$ (strictly, the set W of the roots of such f-views) in $\widetilde{T}_Z^{2(n-1)}(v)$ by using the view counting algorithm in Figure 60; this takes $O(n^5 D \log n)$ time from Lemma 58. Every party then minimizes each of the f-views by using the f-view minimization algorithm in Figure 57; this takes $|W| \times O(n^2 D(\log n)^2) = O(n^3 D(\log n)^2)$ by setting $|V^f| = O(n^2)$ and $L = O(1)$ in Lemma 53, since $|W| = O(n)$ if there are n parties. Finally, every party encodes each of the minimized f-views in any simple binary encoding scheme that maps any two f-views to the same binary string if and only if the f-views are isomorphic, and sorts the encoded f-views; the encoding takes $|W| \times O(Dn^2 \log D)$ and the sorting takes $O(|W| \log |W|) \times O(Dn^2 \log D) = O(Dn^3(\log D)\log n)$, since each of the f-views is of size $O(Dn^2)$ and has node labels of $O(1)$ bits and edge labels of $O(\log D)$ bits. Finally every party selects the first f-view among the sorted f-views. Since the minimal f-view for any fixed view is unique up to isomorphism by Corollary 51 and any two non-isomorphic views cannot be transformed into isomorphic f-views, the f-views selected by all parties are those obtained from

isomorphic views. Thus, the algorithm works well.

Let k be the number of eligible parties and let q be the cardinality of the equivalence class. q is a divisor of k since all the equivalence classes have the same cardinality. Thus, if n is expressed as $p_1^{\alpha_1} p_2^{\alpha_2} \dots p_m^{\alpha_m}$ for prime numbers p_is, Algorithm II takes at most $\alpha = \sum_{j=1}^{m} \alpha_j$ phases in stage 2. In particular, when $\alpha = O(1)$ (e.g. prime numbers), the algorithm takes only constant phases. Notice that when $n = 2^m$, the necessary number of phases is still $m = \log n$.

4.3.4 Generalization of the algorithm

In the case where only the upper bound N of the number of parties is given, we cannot apply Algorithm II as it is, since Algorithm II depends strongly on counting the exact number of eligible parties and this requires the exact number of parties.

We modify Algorithm II so that it outputs $\mathbf{status} =$ "error" and halts (1) if steps 1 to 5 of Stage 2 are iterated over $\log n$ times, or (2) if it is found that non-integer values are being stored into the variables whose values should be integers. Notice that we can easily see that this modified Algorithm II can run (though it may halt with output "error") even when it is given the wrong number of parties as input, unless the above condition (2) becomes true during execution. Let the modified Algorithm II be $\mathsf{LE}(\mathbf{status}, n, d)$.

The basic idea is to run $\mathsf{LE}(\text{"eligible"}, m, d)$ $(2 \leq m \leq N)$ in parallel. Here we assume that every party has one processor, and all local computations are performed sequentially. Instead, message passing is done in parallel, i.e., at each round the messages of $\mathsf{LE}(\text{"eligible"}, m, d)$ $(2 \leq m \leq N)$ are packed into one message and sent to adjacent parties. Although this parallelism cannot reduce time/communication complexity, it can reduce the necessary number of rounds. Let M be the largest $m \in \{2, 3, \dots, N\}$ such that $\mathsf{LE}(\text{"eligible"}, m, d)$ terminates with output "eligible" or "ineligible." The next lemma implies that M is equal to the hidden number of parties, i.e., n, and thus $\mathsf{LE}(\text{"eligible"}, M, d)$ elects the unique leader. Figure 54 describes this generalized algorithm. We call it the generalized Algorithm II.

Lemma 42 *For any number m larger than the number n of parties, if every party l runs $\mathsf{LE}(\text{"eligible"}, m, d_l)$, it always outputs "error," where d_l is the number of neighbors of party l.*

Proof. It is sufficient to prove that k is never equal to 1 in the modified Algorithm II, since step 5 of Stage 2 outputs $\mathbf{status}$ only when $k = 1$. k is set to $c_{z_{\text{minor}}}$ at step 4 of Stage 2 and $c_{z_{\text{minor}}}$ is computed at step 3 of Subroutine $\tilde{\mathrm{C}}$. If we prove that, for any i, $c_i > 1$ at step 2 of Subroutine $\tilde{\mathrm{C}}$, the lemma follows. Subroutine $\tilde{\mathrm{C}}$ computes

$$c_i = m \frac{|\Gamma^{2(m-1)}(i)|}{|\Gamma^{2(m-1)}|}.$$

Generalized Algorithm II

Input: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$, integers N, d

Output: a classical variable $\mathbf{status} \in \{$"eligible", "ineligible"$\}$

1. Run in parallel LE("eligible", m, d) for $m = 2, 3, \ldots, N$.
2. Output $\mathbf{status}$ returned by LE("eligible", M, d), where M is the largest value of $m \in \{2, 3, \ldots, N\}$ such that LE("eligible", m, d) outputs $\mathbf{status} \in \{$"eligible", "ineligible"$\}$.

Figure 54: Generalized Algorithm II.

If i is in $\{0, 1, 2, 3\} \setminus \{z_1, \ldots, z_n\}$ where z_l is the z value of party l, c_i is 0 and then set to n; otherwise

$$c_i \geq m/n > 1,$$

since $|\Gamma^{2(m-1)}(i)| \geq 1$ and $|\Gamma^{2(m-1)}| \leq n$.

□

Theorem 43 *Let $|E|$ and D be the number of edges and the maximum degree of the underlying graph, respectively. Given the upper bound N of the number of parties, the generalized Algorithm II exactly elects a unique leader in $O(DN^6(\log N)^2)$ time and $O(N \log N)$ rounds of which only the first round requires quantum communication. The total communication complexity over all parties is $O(D|E|N^4(\log D) \log N)$ which includes the communication of only $O(|E|N \log N)$ qubits.*

Proof. From Lemma 42, the correctness is obvious. As for the complexity, we can obtain the complexity of the modified Algorithm II, i.e., LE("eligible", m, d), simply by replacing n with m in the complexity of the (original) Algorithm II. Since the generalized Algorithm II runs LE("eligible", m, d) for $m = 2, \ldots, N$ in parallel, the number of rounds required is the same as the maximum of that of the modified Algorithm II over $m = 2, \ldots N$. The time/communication complexity is $O(\sum_{m=2}^{N} C(m)) = O(N \cdot C(N))$, where $C(m)$ is that of Algorithm II when m is used instead of n, described in Theorem 41. □

In fact, it is possible to reduce the time/communication complexity at the expense of the number of rounds. Suppose that every party l runs LE("eligible", m, d_l) sequentially in the decreasing order of m starting at N. When LE("eligible", m, d_l) outputs $\mathbf{status}$ that is either "eligible" or "ineligible," the algorithm halts. From Lemma 42, it is clear that the algorithm halts when

$m = n$, which saves time and communication that are supposed to be taken by $\mathsf{LE}(\text{“eligible”}, m, d_l)$ for $m < n$.

4.3.5 Application to network topologies of directed graphs

Algorithm II can be easily modified so that it can be applied to the network topologies whose underlying graph is directed and strongly-connected.

We slightly modify the network model as follows. In a quantum distributed system, every party can perform quantum computation and communication, and each pair of parties has at most one *uni*directional quantum communication link in each direction between them. For each pair of parties, there is at least one directed path between them for each direction. When the parties and links are regarded as nodes and edges, respectively, the topology of the distributed system is expressed by a strongly-connected graph, denoted by $G = G(V, E)$. Every party has two kinds of *ports*: *in-ports* and *out-ports*; they correspond one-to-one to incoming and outgoing communication links, respectively, incident to the party. Every port of party l has a unique label i, $(1 \leq i \leq d_l)$, where d_l is the number of parties adjacent to l and $d_l = d_l^I + d_l^O$ for the number d_l^I (d_l^O) of in-ports (resp. out-ports) of party l. For $G(V, E)$, the port numbering σ is defined in the same as in the case of the undirected graph model. Just for ease of explanation, we assume that in-port i of party l corresponds to the incoming communication link connected to the ith party among all adjacent parties that have an outgoing communication link destined to party l; out-port i of party l is also assumed in a similar way.

The view for the strongly-connected underlying graph can be naturally defined. For each v and port numbering σ, the *view* $T_{G,\sigma,X}(v)$ is a labeled, rooted tree with infinite depth defined recursively as follows: (1) $T_{G,\sigma,X}(v)$ has the root w with label $X(v)$, corresponding to v, (2) for the source v_j of every directed edge coming into v in G, $T_{G,\sigma,X}(v)$ has vertex w_j labeled with $X(v_j)$, and an edge from root w to w_j with label $\mathsf{label}((v, v_j))$ given by $\mathsf{label}((v, v_j)) = (\sigma[v](v, v_j), \sigma[v_j](v, v_j))$, and (3) w_j is the root of $T_{G,\sigma,X}(v_j)$. $T_X^h(v)$ is defined in the same way as in the case of the undirected graph model. The above definition also gives the way of constructing $T_{G,\sigma,X}^h(v)$. It is stressed that every party corresponds to at least one node of the view if the underlying graph is strongly-connected. It can be proved in almost the same way as [195, 196] that the equivalence classes with respect to the isomorphism of views have the same cardinality for fixed $G = G(V, E), \sigma$ and X; $c_{G,\sigma,X}(S)$ can be computed from a view.

Lemma 44 *For the distributed system whose underlying graph $G = G(V, E)$ is strongly-connected, the number of views isomorphic to view T is constant over all T for fixed σ and X.*

Proof. Let $T(v)$ and $T(v')$ be any two non-isomorphic views for fixed σ and X, and let $\{T(v_1), \ldots, T(v_m)\}$ be the set of all views isomorphic to $T(v)$, where $v \in \{v_i \mid i = 1, \ldots, m\} \subseteq V$. Since the underlying graph is strongly-connected, there is that subtree of $T(v_1)$ which is isomorphic to $T(v')$. Let s be the sequence of edge and node labels from the root of $T(v_1)$ to the root u_1 of the subtree. Since all views in $\{T(v_1), \ldots, T(v_m)\}$ are isomorphic to one another, there is the node u_i that can be reached from the root of $T(v_i)$ along s for each $i = 1, \ldots m$. Clearly, every u_i is the root of a tree isomorphic to $T(v')$. Since v_i and v_j correspond to different parties if $i \neq j$, so do u_i and u_j if $i \neq j$. This implies that the number of views isomorphic to $T(v')$ is not less than that of views isomorphic to $T(v)$. By changing $T(v)$ for $T(v')$, we can see that the number of views isomorphic to $T(v)$ is not less than that of views isomorphic to $T(v')$. This completes the proof. □

Since f-view depends only on the outgoing edges of every node, f-view also works for any strongly-connected underlying graph.

From the above, it is not difficult to see that Subroutines Ã, B̃ and C̃ can work (with only slight modification), since they use only classical communication. In what follows, we describe a modification, called Subroutine Q', to Subroutine Q. This leads to the correctness of the modification to Algorithm II for any strongly-connected underlying graph.

Subroutine Q' just restricts Subroutine Q so that every party can send qubits only via out-ports and receive qubits only via in-ports. Figure 55 gives a precise description of Subroutine Q'; the subroutine requires those two integers d^I and d^O together with $\mathbf{R}_0$ as input, which are supposed to be d_l^I and d_l^I, respectively. Thus, Algorithm II needs to be slightly modified so that it can handle d^I and d^O in stead of d.

We can prove the next lemma in a similar way to Lemma 31.

Lemma 45 *For an n-party distributed system, suppose that every party l calls Subroutine Q' with a one-qubit register whose content is initialized to $|0\rangle$ and d_l^I and d_l^O as input $\mathbf{R}_0$, d^I and d^O, respectively. After performing Subroutine Q', all parties share $(|x\rangle + |\overline{x}\rangle)/\sqrt{2}$ with certainty, where x is a random n-bit string.*

Proof (Sketch). After step 2, the state in $\mathbf{R}_0$'s and $\mathbf{R}_1''$'s, $\ldots$, $\mathbf{R}_{d^I}''$'s of all parties is a uniform superposition of some basis states in an orthonormal basis of $2^{\sum_{l=1}^n (d_l^O+1)}$-dimensional Hilbert space: the basis states correspond one-to-one to n-bit integers a and each of the basis states is of the form $|a_1\rangle^{\otimes(d_1^O+1)} \otimes \cdots \otimes |a_n\rangle^{\otimes(d_n^O+1)}$, where a_l is the lth bit of the binary expression of a. If we focus on the lth party's part of the basis state corresponding to a, step 3 transforms $|a_l\rangle^{\otimes(d_l^O+1)}$ into $|a_l\rangle \left(\bigotimes_{j=1}^{d_l^I} |a_{l_j}\rangle\right)$, where we assume that party l is connected to party l_j via in-port j. Notice that the total number of qubits over all parties is preserved, since $\sum_{l=1}^n d_l^I = \sum_{l=1}^n d_l^O$. More precisely, steps 1 to 4 transform the system state as

Subroutine Q'

Input: a one-qubit quantum register $\mathbf{R}_0$, integers d^I and d^O

Output: a one-qubit quantum register $\mathbf{R}_0$, a binary string y of length d^I

1. Prepare d^O one-qubit quantum registers $\mathbf{R}'_1, \ldots, \mathbf{R}'_{d^O}$ and $2d^I$ one-qubit quantum registers $\mathbf{R}''_1, \ldots, \mathbf{R}''_{d^I}$, $\mathbf{S}_1, \ldots, \mathbf{S}_{d^I}$, each of which is initialized to the $|0\rangle$ state.
2. Generate the (d^O+1)-cat state $(|0^{d^O+1}\rangle + |1^{d^O+1}\rangle)/\sqrt{2}$ in registers $\mathbf{R}_0$, $\mathbf{R}'_1, \ldots, \mathbf{R}'_{d^O}$.
3. Send the qubit in $\mathbf{R}'_i$ to the party connected via out-port i for $1 \leq i \leq d^O$. Receive the qubit from the party connected via in-port i and store it into one-qubit register $\mathbf{R}''_i$ for $1 \leq i \leq d^I$.
4. Set the content of $\mathbf{S}_i$ to $x_0 \oplus x_i$, for $1 \leq i \leq d^I$, where x_0 and x_i denote the contents of $\mathbf{R}_0$ and $\mathbf{R}''_i$, respectively.
5. Measure the qubit in $\mathbf{S}_i$ in the $\{|0\rangle, |1\rangle\}$ basis to obtain a bit y_i, for $1 \leq i \leq d^I$.
   Set $y := y_1 \cdots y_{d^I}$.
6. Apply CNOT controlled by the content of $\mathbf{R}_0$ and targeted to the content of each $\mathbf{R}''_i$ for $i = 1, 2, \ldots, d^I$ to disentagle $\mathbf{R}''_i$s.
7. Output $\mathbf{R}_0$ and y.

Figure 55: Subroutine Q'

follows:

$$\begin{aligned}
\bigotimes_{l=1}^{n} |0\rangle|0^{d_l^O}\rangle|0^{d_l^I}\rangle &\rightarrow \bigotimes_{l=1}^{n} \frac{|0^{d_l^O+1}\rangle + |1^{d_l^O+1}\rangle}{\sqrt{2}}|0^{d_l^I}\rangle \\
&\rightarrow \frac{1}{\sqrt{2^n}} \sum_{a=0}^{2^n-1} \bigotimes_{l=1}^{n} \left\{ |a_l\rangle \left(\bigotimes_{j=1}^{d_l^I} |a_{l_j}\rangle \right) |0^{d_l^I}\rangle \right\} \\
&\rightarrow \frac{1}{\sqrt{2^n}} \sum_{a=0}^{2^n-1} \bigotimes_{l=1}^{n} \left\{ |a_l\rangle \left(\bigotimes_{j=1}^{d_l^I} |a_{l_j}\rangle \right) \left(\bigotimes_{j=1}^{d_l^I} |a_l \oplus a_{l_j}\rangle \right) \right\}.
\end{aligned}$$

After every party measures registers $\mathbf{S}_i$'s at step 5, the state transformation can be described as follows, due to a similar argument to Claim 32 (using the strong

connectivity of the underlying graph):

$$\frac{1}{\sqrt{2}}\bigotimes_{l=1}^{n}\left(|A_l\rangle\bigotimes_{j=1}^{d_l^I}|A_{l_j}\rangle\right)+\frac{1}{\sqrt{2}}\bigotimes_{l=1}^{n}\left(|\overline{A_l}\rangle\bigotimes_{j=1}^{d_l^I}|\overline{A_{l_j}}\rangle\right)$$
$$\rightarrow\ \frac{1}{\sqrt{2}}\bigotimes_{l=1}^{n}\left(|A_l\rangle\bigotimes_{j=1}^{d_l^I}|A_l\oplus A_{l_j}\rangle\right)+\frac{1}{\sqrt{2}}\bigotimes_{l=1}^{n}\left(|\overline{A_l}\rangle\bigotimes_{j=1}^{d_l^I}|\overline{A_l}\oplus\overline{A_{l_j}}\rangle\right).$$

Finally, the qubits in $\mathbf{R}_0$'s are in the state $(|x\rangle+|\overline{x}\rangle)/\sqrt{2}$.

□

4.3.6 A special case: n is a power of two

We introduce an algorithm in [182] that is restricted to the case wherein the number of parties is a power of two. The algorithm makes the most of the nice combination of the view and operator U_k defined in Subroutine B; it takes at most only $6n$ rounds for any topology, while the total communication complexity is $O(n^6 \log n)$ over which the quantum communication is dominant.

The idea is as follows. The algorithm generates an inconsistent state $|\phi\rangle$ shared by all parties such that $|\phi\rangle$ is a superposition of only the n-bit strings of odd Hamming weights (the Hamming weight is defined for any binary string x as the number of bits in x of value 1). Suppose that every party l obtains a single-bit value y_l by measuring his qubit. The number of the parties who got 1 is odd; the number is relatively prime to n, since n is a power of two. In this case, Proposition 48 says that, if every party l uses his view of depth $n-1$ as its identifier, a unique leader can be elected deterministically. Thus, only a single run of the above process is sufficient to elect a unique leader, which takes just linear rounds in n.

First, every party prepares $(|0\rangle+|1\rangle)/\sqrt{2}$ and $|0\rangle$ in one-qubit registers $\mathbf{R}_0$ and $\mathbf{S}$, respectively. They then call Subroutine HW to set the content of $\mathbf{S}$ to the Hamming weight $\pmod 2$ of the contents in all $\mathbf{R}_0$s. Subroutine HW first computes a superposition of the views of depth $2(n-1)$, regarding the contents of $\mathbf{R}_0$'s as node labels. This takes at most $2n$ rounds to construct the view. Subroutine HW then computes $c_X(1) = n|\Gamma_X^{(n-1)}(1)|/|\Gamma_X^{(n-1)}| \pmod 2$ (i.e. the Hamming weight) from the view with respect to each basis state, where X is the mapping naturally induced by each basis state. As in Subroutine A, Subroutine HW finally disentangles all work qubits used while constructing the view by inverting all computation and communication that was performed to construct the view; this takes another $2n$ rounds. Next every party measures the qubit in $\mathbf{S}$ in the $\{|0\rangle,|1\rangle\}$ basis and stores the result into variable y. If $y=0$ (1), the resulting state is a uniform superposition of only the n-bit strings that have the Hamming weights of

Algorithm for the case where n is a power of two

Input: classical variable $\mathbf{status}$ = "eligible", integers n, d

Output: classical variable $\mathbf{status} \in$ {"eligible", "ineligible"}

1. Prepare $(|0\rangle+|1\rangle)/\sqrt{2}$ and $|0\rangle$ in one-qubit registers $\mathbf{R}_0$ and $\mathbf{S}$, respectively.
2. Call Subroutine HW with $\mathbf{R}_0$, $\mathbf{S}$, n and d, to set to $\mathbf{S}$ the Hamming weight $(\mathrm{mod}\ 2)$ of the contents of all parties' $\mathbf{R}_0$s.
   Measure the qubit in $\mathbf{S}$ in the $\{|0\rangle, |1\rangle\}$ basis and store the result into variable y.
3. If $y = 0$, apply $U_n \cdot H$ to the qubit in $\mathbf{R}_0$. Measure the qubit in $\mathbf{R}_0$ in the $\{|0\rangle, |1\rangle\}$ basis and store the result into variable z.
4. Call Subroutine L with $\mathbf{status}$, z, n and d.
5. Output $\mathbf{status}$.

Figure 56: Linear-round algorithm for the case where n is a power of two.

even (resp. odd) values. When $y = 0$, every party applies the Hadamard operator H and U_n (in this order) to the qubit in $\mathbf{R}_0$ to transform the superposition into a superposition of only the strings that have odd Hamming weights due to Lemmas 46 and 47, where

$$U_n = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{-i\frac{\pi}{n}} \\ -e^{i\frac{\pi}{n}} & 1 \end{pmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

By measuring the qubit in $\mathbf{R}_0$, every party gets a single-bit value z. Finally, every party calls Subroutine L, which elects a leader by classically constructing the view of depth $2(n-1)$ in $2n$ rounds by regarding z values as node labels. Lemma 48 implies that, when n is a power of two, no two parties have an isomorphic view if the number of parties having 1 is odd. By regarding the view of every party as his identifier, parties can elect a unique leader from among them. Therefore the total number of rounds required is at most $6n$ rounds.

More precisely, each party l performs the algorithm shown in Figure 56 with parameters "eligible," n, the number d_l of the neighbors of party l. The party who obtains output "eligible" is the unique leader.

If we use f-views instead of views, the time/communication complexity can be polynomially bounded in the number of parties.

The next lemmas prove that U_nH transform a uniform superposition of all n-bit strings having even Hamming weight into a superposition of only the n-bit strings having odd Hamming weight. More precisely, the resulting state after applying H is the n-qubit cat-state; U_n transforms the cat-state into the state that we want.

Lemma 46 *For any positive integer n, suppose that n parties share a uniform superposition of all n-bit strings of even Hamming weight. If every party performs the Hadamard operator H to its qubit that forms the superposition, the state is transformed to the n-qubit cat state, i.e.,*

$$(H)^{\otimes n}\frac{1}{\sqrt{2^{n-1}}}\sum_{\substack{0\le j\le 2^n-1\\ \mathrm{HW}(j)=0 \pmod 2}}|j\rangle=\frac{|0^n\rangle+|1^n\rangle}{\sqrt{2}},$$

where $\mathrm{HW}(j)$ denotes the Hamming weight of (the binary expression of) j.

Proof. Let $|\psi_n\rangle$ be

$$(H^\dagger)^{\otimes n}\frac{|0^n\rangle+|1^n\rangle}{\sqrt{2}}=H^{\otimes n}\frac{|0^n\rangle+|1^n\rangle}{\sqrt{2}}.$$

It is sufficient to prove that $|\psi_n\rangle$ is a uniform superposition of all n-bit strings that have even Hamming weight.

Fix an n-bit string j whose Hamming weight is $2m$ for any nonnegative integer m ($\le \lfloor n/2\rfloor$). The amplitude of $|j\rangle$ in $|\psi_n\rangle$ is

$$\frac{1}{\sqrt{2}}\left\{\left(\frac{1}{\sqrt{2}}\right)^{n-2m}\left(\frac{1}{\sqrt{2}}\right)^{2m}+\left(\frac{1}{\sqrt{2}}\right)^{n-2m}\left(\frac{-1}{\sqrt{2}}\right)^{2m}\right\}=\frac{1}{\sqrt{2^{n-1}}}.$$

Fix an n-bit string j whose Hamming weight is $2m-1$ for any positive integer m ($\le \lfloor (n+1)/2\rfloor$). The amplitude of $|j\rangle$ in $|\psi_n\rangle$ is

$$\frac{1}{\sqrt{2}}\left\{\left(\frac{1}{\sqrt{2}}\right)^{n-(2m-1)}\left(\frac{1}{\sqrt{2}}\right)^{(2m-1)}+\left(\frac{1}{\sqrt{2}}\right)^{n-(2m-1)}\left(\frac{-1}{\sqrt{2}}\right)^{(2m-1)}\right\}=0.$$

This completes the proof. □

Lemma 47 *Suppose that $|\phi_n\rangle=(U_n)^{\otimes n}((|0^n\rangle+|1^n\rangle)/\sqrt{2})$. Then $|\phi_n\rangle$ is a superposition of only the n-bit strings that have odd Hamming weight.*

Proof. Fix an n-bit string j whose the Hamming weight is $2m$ for any nonnegative integer m ($\le \lfloor n/2\rfloor$). The amplitude of $|j\rangle$ in $|\phi_n\rangle$ is as follows:

$$\frac{1}{\sqrt{2}}\left\{\left(\frac{1}{\sqrt{2}}\right)^{n-2m}\left(\frac{-e^{i\frac{\pi}{n}}}{\sqrt{2}}\right)^{2m}+\left(\frac{e^{-i\frac{\pi}{n}}}{\sqrt{2}}\right)^{n-2m}\left(\frac{1}{\sqrt{2}}\right)^{2m}\right\}=0.$$

□

Proposition 48 *Suppose that n is a power of two. For any n-party distributed system with underlying graph $G = (V, E)$, if X is a mapping such that $X : V \to \{0, 1\}$ and $c_X(1) = 1 \pmod 2$, no two views $T^{(n-1)}_{X,\sigma,G}(v)$'s are isomorphic to each other.*

Proof. The cardinality of each equivalence class with respect to view isomorphism is constant for fixed G, X and port numbering σ. Let this cardinality be c_X. Cardinality c_X is thus a divisor of n. The roots of the views in any equivalence class are mapped by X to the same value. Thus, the number of parties of the same label (defined by X) needs to be a multiple of c_X. Hence, cardinality c_X is a common divisor of n and $c_X(1)$. When n is a power of two and $c_X(1) = 1 \pmod 2$, the common divisor is uniquely determined to be 1. Since $T_{X,\sigma,G}(v)$ is isomorphic to $T_{X,\sigma,G}(v')$ if and only if $T^{(n-1)}_{X,\sigma,G}(v)$ is isomorphic to $T^{(n-1)}_{X,\sigma,G}(v')$ for any $v, v' \in V$, the proof is completed. □

4.4 Folded view and its algorithms

View size is exponential against its depth since a view is a tree. Therefore, exponential communication bits are needed if the implementation simply exchanges intermediate views. Hence we introduce a technique to compress views by sharing isomorphic subtrees of the views. We call a compressed view *a folded view* (or an *f-view*). The key observation is that there are at most n isomorphic subtrees in a view when the number of parties is n. This technique reduces not only communication complexity but also local computation time by *folding* all intermediate views and constructing larger f-views *without unfolding* intermediate f-views.

4.4.1 Terminology

The folded view has all information possessed by the corresponding view. To describe such information, we introduce a new notion, "*path set*," which is equivalent to a view in the sense that any view can be reconstructed from the corresponding path set, and vice versa.

A path set, $P_{G,\sigma,X}(v)$, is defined for view $T_{G,\sigma,X}(v)$. Let u_{root} be the root of $T_{G,\sigma,X}(v)$. Suppose that every edge of a view is directed and their sources are the ends that are nearer to u_{root}. $P_{G,\sigma,X}(v)$ is the set of directed labeled paths starting at u_{root} with infinite length in $T_{G,\sigma,X}(v)$. More formally, let $s(p) = (\mathsf{label}(u_0), \mathsf{label}(e_0), \mathsf{label}(u_1), \cdots)$ be the sequence of labels of those nodes and edges which form an infinite-length directed path $p = (u_0, e_0, u_1, \cdots)$ starting at $u_0 (= u_{\mathrm{root}})$, where u_i is a node, e_i is the directed edge from u_i to u_{i+1} in $T_{G,\sigma,X}(v)$, and $\mathsf{label}(u_i)$ and $\mathsf{label}(e_i)$ are the labels of u_i and e_i, respectively. It is stressed that u_i and e_i are not identifiers of a node and an edge, and are just used

for definition. $P_{G,\sigma,X}(v)$ is the set of $s(p)$ for all p in $T_{G,\sigma,X}(v)$. For $T^h_{G,\sigma,X}(v)$, we naturally define $P^h_{G,\sigma,X}(v)$, i.e., the set of all sequences of labels of those nodes and edges which form directed paths of length h starting at $u_{\rm root}$ in $T^h_{G,\sigma,X}(v)$. In the following, we simply call a sequence in a path set, a *path*, and identify the common length of the paths in a path set with the *length of the path set*.

By the above definition, $P^h_{G,\sigma,X}(v)$ is easily obtained by traversing view $T^h_{G,\sigma,X}(v)$. On the other hand, given $P^h_{G,\sigma,X}(v)$, we can construct the view rooted at $u_{\rm root}$ by sharing the maximal common prefix of any pair of paths in $P^h_{G,\sigma,X}(v)$. In this sense, $P^h_{G,\sigma,X}(v)$ has all information possessed by view $T^h_{G,\sigma,X}(v)$. Let u^j be any node at depth j in $T^h_{G,\sigma,X}(v)$, and suppose that u^j corresponds to node v_{u^j} of G. Since view is defined recursively, we can define the path set $P^{h'}_{G,\sigma,X}(v_{u^j})$ for the h'-depth subtree rooted at u^j, as the set of h'-length directed paths starting at u^j for $h' \leq h - j$. To avoid complicated notations, we may use $P^{h'}_{G,\sigma,X}(u^j)$ instead of $P^{h'}_{G,\sigma,X}(v_{u^j})$. We call $P^{h'}_{G,\sigma,X}(u^j)$ *the path set of length h' defined for u^j*. In particular, when h' is the length from u^j to a leaf, i.e., $h-j$, we call $P^{h-j}_{G,\sigma,X}(u^j)$ *the path set defined for u^j*. For any node u of a view, we use $\mathsf{depth}(u)$ to represent the depth of u, i.e., the length of the path from the root to u, in the view that u belongs to.

4.4.2 Folded view

For any node u in a view and its corresponding party l, if an outgoing edge e of u corresponds to the communication link incident to party l via port i, we call edge e "*i-edge of u*" and denote the destination of e by $\mathsf{Adj}_i(u)$. Now we define an operation, called the "merging operation," which folds a view.

Definition 6 (Merging operation) *For any pair of nodes u and u' at the same depth in a view, the merging operation eliminates one of nodes u and u', and redirects all incoming edges of this node to the remaining one, if u and u' satisfy the following conditions: (1) u and u' have the same label, and (2) when u and u' have outgoing edges (i.e., neither u nor u' is a leaf), u and u' have the same number of outgoing edges, and the i-edges of u and u' have the same label and are directed to the same node for all i.*

Obviously, the merging operation never eliminates the root of a view. Further, the merging operation does not change the length of the directed path from the root to each (remaining) node. Thus, we define the depth of each node u that remains after applying the merging operation as the length of the path from the root to u and denote it again by $\mathsf{depth}(u)$. We call the directed acyclic graph obtained by applying the merging operation to a view, a *folded view (f-view)*, and define the size of an f-view as the number of nodes in the f-view. Since views of finite depth are sufficient for our use, we only consider f-views that are obtained by applying

the merging operation to views of finite depth hereafter. For any view $T^h_{G,\sigma,X}(v)$, its minimal f-view is uniquely determined up to isomorphism as will be proved later and is denoted by $\widetilde{T}^h_{G,\sigma,X}(v)$. We can extend the definition of the path set to f-views: the path set of length h defined for node u in an f-view is the set of all directed labeled paths of length h from u in the f-view. For any node u in an f-view, we often use d_u to represent the number of the outgoing edges of u when describing algorithms later. Since the merging operation does not change the number and labels of the outgoing edges of any remaining node, every node u in an f-view has the same set of outgoing edges as the node corresponding to u in the underlying graph of the distributed system.

The next lemma is essential.

Lemma 49 *For any (f-)view, the merging operation does not change the path set defined for every node of the (f-)view if the node exists after the operation. Thus, the path set of any f-view obtained from a view by applying the merging operation is identical to the path set of the view.*

Proof. Let u' be the node that will be merged into u (i.e., u' will be eliminated). By the definition of the merging operation, the set of the maximal length paths starting at u is identical to the set of those starting at u'. Thus, by eliminating u' and redirecting all incoming edges of u' to u, the path set defined for every node does not change. □

We can characterize f-views by using "path sets." Informally, for every distinct path set P defined for a node at any depth j in a view, any f-view obtained from the view has at least one node which defines P, at depth j.

Before giving a formal characterization of f-views in the next lemma, we need to define some notations. Suppose that u^j be any node at depth j in $T^h_{G,\sigma,X}(v)$. We define $\mathcal{P}^j_{G,\sigma,X}(v)$ as the set of path sets $P^{h-j}_{G,\sigma,X}(u^j)$'s defined for all u^j's. For any path set P, let $P|_x$ be the set obtained by cutting off the first node and edge from all those paths in P which have x as the first edge label.

Lemma 50 *$\acute{T}^h_{G,\sigma,X}(v)$ is an f-view of $T^h_{G,\sigma,X}(v)$ if and only if $\acute{T}^h_{G,\sigma,X}(v)$ is a labeled connected directed acyclic graph $G^f(V^f, E^f)$ such that V^f is the union of disjoint sets V^f_j $(j = 0, \ldots, h)$ of nodes with $|V^f_0| = 1$, and E^f is the union of disjoint sets $E^f_j \subseteq V^f_j \times V^f_{j+1}$ $(j = 0, \ldots, h-1)$ of directed edges, for which every node in V^f_j $(j = 1, \ldots, h)$ can be reached via a directed path from $u_r \in V^f_0$, and there is a mapping ψ from V^f_j onto $\mathcal{P}^j_{G,\sigma,X}(v)$ for each $j = 0, \ldots, h$ such that,*

- *C1 each node $u \in V^f$ has the label that is identical to the common label of the first nodes of paths in $\psi(u)$,*

C2 for each $u \in V^f$, there is a bijective mapping from the set of outgoing edges of u to the set of the first edge labels of paths in $\psi(u)$, such that any outgoing edge (u, u') has label x to which (u, u') is mapped and $\psi(u')$ is equal to $\psi(u)|_x$.

Proof. ($\Rightarrow$) We will prove that, for any f-view obtained by applying the merging operation to $T^h_{G,\sigma,X}(v)$, there exists ψ that satisfies C1 and C2. From Lemma 49, the merging operation does not change the path set defined for any node (if it exists after the operation). It follows that the path set defined for any node at depth j in the f-view belongs to $\mathcal{P}^j_{G,\sigma,X}(v)$. Conversely, for every path set P in $\mathcal{P}^j_{G,\sigma,X}(v)$, there is at least one node at depth j in the f-view such that the path set defined for the node is P, since the merging operation just merges two nodes defining the same path set. Let ψ be the mapping that maps every node u of the f-view to the path set defined for u. From the above argument, ψ is a mapping from V^f_j onto $\mathcal{P}^j_{G,\sigma,X}(v)$ and meets C1. To show that ψ meets C2, we use simple induction on the sequence of the merging operation. By the definition, $T^h_{G,\sigma,X}(v)$ meets C2. Suppose that one application of the merging operation transformed an f-view to a smaller f-view, and that ψ meets C2 for the f-view before the operation. If we define ψ' for the smaller f-view as the mapping obtained by restricting ψ to the node set of the smaller f-view, ψ' meets C2 by the definition of the merging operation.

($\Leftarrow$) We will prove that any graph G^f for which there is ψ satisfying C1 and C2 can be obtained by applying the merging operation to $T^h_{G,\sigma,X}(v)$. We can easily show by induction that the set P of all maximal-length labeled directed paths starting at $u_0 \in V^f_0$ is identical to the path set $P^h_{G,\sigma,X}(v)$ of $T^h_{G,\sigma,X}(v)$ from the definition of G^f. We will give an inversion of the merging operation that does not change P when it is applied to G^f, and show that we can obtain the view that defines $P^h_{G,\sigma,X}(v)$ by maximally applying the inversion. This view is isomorphic to $T^h_{G,\sigma,X}(v)$, since the view is uniquely determined for a fixed path set. It follows that G^f can be obtained from $T^h_{G,\sigma,X}(v)$ by reversing the sequence of the inversion.

The inverse operation of the merging operation is defined as follows: if some node $u^j \in V^f_j$ $(1 \leq j \leq h)$ has multiple incoming edges, say, $e_1, \ldots e_t \in E_{j-1}$, the inverse operation makes a copy u' of u^j together with its outgoing edges and redirects e_2,$\ldots e_t$ to u' (e_1 is still directed to the original node u^j). Let $G^{f'}$ be the resulting graph. The sets of maximal-length paths from $u_0 \in V^f_0$ and $u'_0 \in V^{f'}_0$ are obviously identical to each other. Consider a mapping ψ' such that ψ' is identical to ψ of G^f for all nodes except u', and $\psi'(u')$ is equal to $\psi(u)$. Then ψ' is a mapping from $V^{f'}_j$ onto $\mathcal{P}^j_{G,\sigma,X}(v)$ and meets C1 and C2. Thus, the inverse operation can be applied repeatedly without changing the set of maximal-length paths from $u_0 \in V^f_0$ until there are no nodes that have multiple incoming edges. It follows that G^f is transformed into a view that defines $P^h_{G,\sigma,X}(v)$ by maximally

applying the operation. □

From this lemma, we obtain the next corollary.

Corollary 51 *Any minimal f-view $\widetilde{T}^h_{G,\sigma,X}(v)$ is unique up to isomorphism and has exactly $|\mathcal{P}^j_{G,\sigma,X}(v)|$ nodes at depth j $(0 \leq j \leq h)$. The minimal f-view of depth h for any n-party distributed system has $O(hn)$ nodes and $O(hDn)$ edges, where D is the maximum degree over all nodes of the underlying graph.*

Proof. When ψ in Lemma 50 is a bijective mapping from V^f_j to $\mathcal{P}^j_{G,\sigma,X}(v)$ for all j, the f-view is minimal. Thus, the f-view has $|\mathcal{P}^j_{G,\sigma,X}(v)|$ nodes at depth j $(0 \leq j \leq h)$.

Let $\widetilde{T}^h_{X,a}(v)$ and $\widetilde{T}^h_{X,b}(v)$ be any two minimal f-views of $T^h_{G,\sigma,X}(v)$, and let ψ_a and ψ_b be their corresponding bijective mappings defined in Lemma 50, respectively. If we define $\phi = \psi_b^{-1}\psi_a$ for the inverse mapping ψ_b^{-1} of ψ_b, ϕ is a bijective mapping from the node set of $\widetilde{T}^h_{X,a}(v)$ to that of $\widetilde{T}^h_{X,b}(v)$. Since any node u_a at depth j of $\widetilde{T}^h_{X,a}(v)$ is mapped by ψ_a to some path set P in $\mathcal{P}^j_{G,\sigma,X}(v)$, which is mapped to some node u_b at depth j by ψ_b^{-1}. Obviously, u_a and u_b have the same degree and have the same label as the first node of paths in P. Let u'_a be any node incident to a directed edge with label x emanating from u_a. Node u'_a is mapped to $\psi_a(u)|_x$, which is mapped to node u'_b incident to the directed edge with label x emanating from u_b. Thus, ϕ is an isomorphism from $\widetilde{T}^h_{X,a}(v)$ to $\widetilde{T}^h_{X,b}(v)$.

For the second part, if there are n parties, it is obvious that $|\mathcal{P}^j_{G,\sigma,X}(v)| \leq n$ for any j $(0 \leq j \leq h)$. Since each node has at most D outgoing edges, the lemma follows. □

4.4.3 Folded-view minimization

The key idea of the minimization algorithm is to maximally applying the merging operation to the (f-)view to be minimized. This idea works well, because of the next lemma.

Lemma 52 *Let $\acute{T}^h_{G,\sigma,X}(v)$ be an f-view for view $T^h_{G,\sigma,X}(v)$. $\acute{T}^h_{G,\sigma,X}(v)$ is isomorphic to the minimal f-view $\widetilde{T}^h_{G,\sigma,X}(v)$ if and only if no merging operation is applicable to $\acute{T}^h_{G,\sigma,X}(v)$.*

Proof. Obviously, no more merging operation can be applied to the minimal f-view. We will prove the other direction in the following. Suppose that there is a non-minimal f-view $\acute{T}^h$ expressing $P^h_{G,\sigma,X}(v)$, to which no more merging operation can be applied. $\acute{T}^h$ must have more than $|\mathcal{P}^j_{G,\sigma,X}(v)|$ nodes at depth j for some j. Let j' be the largest such j. From Lemma 49, for any node $u^{j'}$ at depth j'

in $\acute{T}^h$, the path set defined for $u^{j'}$ is in $\mathcal{P}^{j'}_{G,\sigma,X}(v)$ if $\acute{T}^h$ is an f-view of $T^h_{G,\sigma,X}(v)$. Thus, $\acute{T}^h$ must have at least one pair of nodes at depth j' such that the path sets defined for the two nodes are identical. Therefore, the outgoing edges of the two nodes with the same edge label are directed to the same node, since no two nodes at depth $j'+1$ have the same path set. Thus, the merging operation can still be applied to the node pair. This is contradiction. □

The algorithm

The minimization algorithm applies the merging operation to every node in the (f-)view in a bottom-up manner, i.e., in the decreasing order of the depth of nodes. Clearly, this ensures that no application of the merging operation at any depth j creates that new node pair at depth larger than j to which the merging operation is applicable. Thus, no more merging operations can be applied when the algorithm halts. It follows that the algorithm outputs the minimal f-view by Lemma 52.

In order to apply the merging operation, we need to be able to decide if two edges are directed to the same node, which implies that we need to identify each node. We thus assign a unique identifier, denoted by $\mathsf{id}(u)$, to each node u in the (f-)view. In order to efficiently check condition (2) of the merging operation, we also construct the data structure for each node that includes the label of the node, and the labels and destination node id of all outgoing edges: the data structure, called *key*, for node u is of the form $(\mathsf{label}(u), \mathsf{ekey}(u))$, where

- $\mathsf{label}(u)$ is the label of u,
- $\mathsf{ekey}(u)$ is a linked list of pairs $(x_1, y_1) \cdots (x_{d_u}, y_{d_u})$ of the label x_i and destination y_i, respectively, of the i-edge of u for all i, where d_u is the number of outgoing edges of u.

We prepare another linked list V^f_j of all nodes at depth j for each j to make sure that the merging operation is applied to the set of all nodes at depth j before moving on to depth $j-1$. Data structures $\mathsf{id}(u)$ and $\mathsf{ekey}(u)$ for every node u and V^f_j for every depth j can be constructed by one traversal of the input f-view in a breadth-first manner.

We then perform the merging operation in a bottom-up manner. For each depth j from h to 1, the next operations are performed. (Notice that the algorithm ends at $j = 1$, since at depth 0 is only the root, which is never removed.) First we sort the nodes u's in V^f_j by regarding keys, i.e., $(\mathsf{label}(u), \mathsf{ekey}(u))$ as a binary string, which makes all nodes having the same key adjacent to each other in V^f_j. For each maximal subsequence of those nodes in V^f_j which have the same pair $(\mathsf{label}(u), \mathsf{ekey}(u))$, we will remove all nodes but the first node in the subsequence

and redirect all incoming edges of the removed nodes to the first node. Here we introduce variables $\mathbf{primary}$ and $\mathbf{primarykey}$ to store the first node u and its key $(\mathsf{label}(u), \mathsf{ekey}(u))$, respectively, of the subsequence currently processed. Concretely, we perform the next operation to every node u in V_j^f in the sorted order: if $(\mathsf{label}(u), \mathsf{ekey}(u))$ is equal to $\mathbf{primarykey}$ (i.e., conditions (1) and (2) of the merging operation are met), then remove u from V_j^f, redirect all incoming edges of u to $\mathbf{primary}$, remove u and all its outgoing edges from $\acute{T}^h$, and set $\mathsf{id}(u)$ to $\mathsf{id}(\mathbf{primary})$ so as to reflect the merger to ekey; otherwise, set $\mathbf{primary}$ to u and $\mathbf{primarykey}$ to $(\mathsf{label}(u), \mathsf{ekey}(u))$. More precisely, the minimization algorithm described in Figure 57 is invoked with an f-view T^h and its depth h (actually, h can be computed from T^h, but we give h as input to simplify the algorithm). The minimization algorithm calls Subroutine Traversal (I) shown in Figure 58 to compute ekey and V_j^f, where `CONTINUE` starts the next turn of the inner-most loop where it lies with the updated index; `DEQUEUE`(Q) removes an element from FIFO queue Q and returns the element; `ENQUEUE`(Q, q) appends an element q to FIFO queue Q; `CON`(L, l) appends an element l to the end of list L.

The minimization algorithm is used as a subroutine when constructing the minimal f-view from scratch as described later.

The next lemma states the time complexity of the minimization algorithm.

Lemma 53 *If the input of the minimization algorithm is an f-view with node set V^f for an n-party distributed system, and any node label is represented by an $O(\log L)$-bit value for some positive integer L, the time complexity of the algorithm is $O(|V^f|(\log|V^f|)(\log L + D\log(D|V^f|)))$, where D is the maximum degree of the underlying graph.*

Proof. We first consider Subroutine Traversal (I) in Figure 58. It is obvious that steps 1 and 2 take constant time. Notice that, by a standard implementation of `DEQUEUE`, `ENQUEUE` and `CON`, each call of them takes constant time. Step 3 traverses the input (f-)view in a bread-first manner. Hence, the time required for step 3 is proportional to the number of edges, which is at most $D|V^f|$. Steps 3.2.2 and 3.2.3 take $O(\log|V^f|)$. It follows that step 3 takes $O(D|V^f|\log|V^f|)$ time.

Now we consider the minimization algorithm in Figure 57. In step 2, steps 2.2 and 2.3 are dominant. In step 2.2, sorting all elements in V_j^f for all j needs $O(|V^f|\log|V^f|)$ comparisons and takes $O(\log L + D\log(D|V^f|))$ time for each comparison, since $\mathsf{label}(u)$ and $\mathsf{ekey}(u)$ have $O(\log L + D\log(D|V^f|))$ bits for any u ($\mathsf{ekey}(u)$ has at most D pairs of an edge label and a node id, which are $\lceil\log D\rceil$ bits and $\lceil\log|V^f|\rceil$ bits, respectively). Thus step 2.2 takes $O(|V^f|(\log|V^f|)(\log L + D\log(D|V^f|))$ time.

Step 2.3 repeats steps 2.3.1 and 2.3.2 at most $|V^f|$ times, since steps 2.3.1 and 2.3.2 are performed once for each node in $\acute{T}^h$ except the root. Clearly,

F-View Minimization Algorithm

Input: a(n) (f-)view $\acute{T}^h$ of depth h, and a positive integer h

Output: minimal f-view $\widetilde{T}^h$

1. Call Subroutine Traversal (I) with $\acute{T}^h$,
   to compute ekey and V_j^f $(j = 1, \ldots, h)$ by breadth-first traversal of $\acute{T}^h$.
2. For $j := h$ down to 1, do the following steps.
   - **2.1** Set $\mathbf{primarykey} := \phi$.
   - **2.2** Sort all elements u in V_j^f by the value obtained by regarding $(\mathsf{label}(u), \mathsf{ekey}(u))$ as a binary string.
   - **2.3** While V_j^f is not empty, perform the following steps.
     - **2.3.1** Remove the first element of V_j^f and set u to the element.
     - **2.3.2** If $(\mathsf{label}(u), \mathsf{ekey}(u)) = \mathbf{primarykey}$,
       redirect all incoming edges of u to $\mathbf{primary}$
       remove u and all its outgoing edges (if they exist) from $\acute{T}^h$
       set $\mathsf{id}(u) := \mathsf{id}(\mathbf{primary})$ to reflect this merger to ekey;
       otherwise,
       set $\mathbf{primary} := u$ and $\mathbf{primarykey} := (\mathsf{label}(u), \mathsf{ekey}(u))$.
3. Output the resulting graph $\widetilde{T}^h$.

Figure 57: F-view minimization algorithm.

each run of step 2.3.1 takes constant time. In each run of step 2.3.2, (a) it takes $O(\log L + D \log(D|V^f|))$ time to compare $(\mathsf{label}(u), \mathsf{ekey}(u))$ with $\mathbf{primarykey}$, (b) it takes $O(d_u^I)$ time to redirect all incoming edges of u to $\mathbf{primary}$ where d_u^I is the number of the incoming edges of u, and (c) it takes $O(d_u)$ time to remove u and all its outgoing edges (if they exist) from $\acute{T}^h$. More precisely, we implicitly assume an appropriate data structure to represent $\acute{T}^h$: each u has two linked lists of incoming edges and outgoing edges such that each edge is registered in the incoming-edge list of his destination and the outgoing-edge list of his source, and the two entries of the edge lists are linked to each other. Although the lists need to be maintained when (b) and (c) are performed, we can easily see that the operation including this maintenance can be done with the above time complexity. Finally, it takes constant time to set $\mathsf{id}(u)$, $\mathbf{primary}$ and $\mathbf{primarykey}$ to new values. The total time required for step 2.3 is proportional to

$$
\begin{aligned}
& O\left(|V^f|(\log L + D \log(D|V^f|)) + \sum_{u \in V^f} d_u^I + \sum_{u \in V^f} d_u \right) \\
= \;& O\left(|V^f|(\log L + D \log(D|V^f|)) + D|V^f| + D|V^f| \right),
\end{aligned}
$$

Subroutine Traversal (I)

Input: a(n) (f-)view $\acute{T}^h$

Output: ekey and V_j^f $(j = 1, \ldots, h)$

1. Perform $\texttt{ENQUEUE}(Q, u_r)$, where u_r is the root of $\acute{T}^h$ and Q is an empty first-in-first-out queue.
   Set $\mathsf{depth}(u_r) := 0$ and $\mathbf{size} := 1$.
2. Set $\mathsf{id}(u_r) := \mathbf{size}$ and then set $\mathbf{size} := \mathbf{size} + 1$.
3. While Q is not empty, repeat the following steps.
   3.1 Set $u := \texttt{DEQUEUE}(Q)$ and then set $\mathsf{ekey}(u) := \phi$.
   If u is a leaf, `CONTINUE`.
   3.2 For $i := 1$ to d_u, where d_u is the degree of u,
   3.2.1 Set $u_i := \mathsf{Adj}_i(u)$.
   If u_i has already been traversed, `CONTINUE`.
   3.2.2 Set $\mathsf{id}(u_i) := \mathbf{size}$ and then set $\mathbf{size} := \mathbf{size} + 1$.
   3.2.3 Set $\mathsf{depth}(u_i) := \mathsf{depth}(u) + 1$.
   Perform $\texttt{CON}(V^f_{\mathsf{depth}(u_i)}, u_i)$ and $\texttt{ENQUEUE}(Q, u_i)$.
   3.2.4 Perform $\texttt{CON}(\mathsf{ekey}(u), (\mathsf{label}((u, u_i)), \mathsf{id}(u_i)))$.
4. Output ekey and V_j^f $(j = 1, \ldots, h)$.

Figure 58: Subroutine Traversal (I).

since no edge can be redirected or removed more than once in step 2.3.2. Hence, step 2.3 takes $O(|V^f|(\log L + D\log(D|V^f|))$ time.

By summing these up, the total time complexity is

$$O(|V^f|(\log|V^f|)(\log L + D\log(D|V^f|)).$$

□

4.4.4 Minimal folded-view construction

We now describe the entire algorithm that constructs a minimal f-view of depth h from scratch by using the f-view minimization algorithm as a subroutine. This construction algorithm is almost the same as the original view construction algorithm: parties exchange minimal intermediate f-views instead of intermediate views, and they construct an f-view $\acute{T}^j_{G,\sigma,X}(v)$ of depth j by connecting received minimal f-views $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$ of depth $j-1$ to the root without unfolding them, and then apply the f-view minimization algorithm to $\acute{T}^j_{G,\sigma,X}(v)$. It is stressed that

F-View Construction Algorithm

Input: integers h, d and x

Output: minimal f-view $\widetilde{T}^h_{G,\sigma,X}$, where X is the underlying mapping naturally induced by the x values of all parties.

1. Generate $\widetilde{T}^0_{G,\sigma,X}(v)$, which consists of only one node with label x.
2. For $j := 1$ to h, perform the following steps.
   - **2.1** Send a copy of $\widetilde{T}^{j-1}_{G,\sigma,X}(v)$ to every adjacent party.
   - **2.2** Receive the minimal f-view $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$ via port i for $1 \leq i \leq d$, where v_i is the node corresponding to the party connected via port i.
   - **2.3** Construct an f-view $\acute{T}^j_{G,\sigma,X}(v)$ from $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$'s as follows.
     - 2.3.1 Set the root u_{root} of $\acute{T}^j_{G,\sigma,X}(v)$ to $\widetilde{T}^0_{G,\sigma,X}(v)$,
     - 2.3.2 Set the ith child u_i of u_{root} to the root of $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$,
     - 2.3.3 Label the edge from u_{root} to u_i with (i, i'), where i' is the port from which $\widetilde{T}^{j-1}_{G,\sigma,X}(v_i)$ was sent, i.e., $i' := \sigma[v_i](v, v_i)$.
   - **2.4** Call the f-view minimization algorithm with $\acute{T}^j_{G,\sigma,X}(v)$ and j. Set $\widetilde{T}^j_{G,\sigma,X}(v)$ to the output of the algorithm.
3. Output $\widetilde{T}^h_{G,\sigma,X}(v)$.

Figure 59: F-view construction algorithm.

$\acute{T}^j_{G,\sigma,X}(v)$ is an f-view, since $\acute{T}^j_{G,\sigma,X}(v)$ can be constructed from view $T^j_{G,\sigma,X}(v)$ by applying the merging operation to every subtree rooted at depth 1. Thus, the minimization algorithm can be applied to $\acute{T}^j_{G,\sigma,X}(v)$. More precisely, each party l having d_l adjacent parties and x_l as his label performs the f-view construction algorithm described in Figure 59 with h, d_l and x_l, in which we assume that v is the node corresponding to party l in the underlying graph.

Lemma 54 *For any distributed system of n parties labeled with $O(\log L)$-bit values, the f-view construction algorithm constructs the minimal f-view of depth $h(= O(n))$ in $O(Dh^2 n(\log n)(\log Ln^D))$ time for each party and $O(|E|h^2 n \log(LD^D))$ communication complexity, where $|E|$ and D are the number of edges and the maximum degree of the underlying graph.*

Proof. $\widetilde{T}^{j-1}_{G,\sigma,X}(v)$ has at most $j \cdot n$ nodes. Thus, $\widetilde{T}^{j-1}_{G,\sigma,X}(v)$ can be expressed by $O(jn \log L + jDn \log D) = O(jn \log(LD^D))$ bits. It follows that steps 2.1 and 2.2 take $O(jDn \log(LD^D))$ time. Since any party has at most D neighbors,

$\acute{T}^j_{G,\sigma,X}(v)$ has at most $j \cdot D \cdot n + 1$ nodes. From Lemma 53, step 2.4 in Figure 59 takes

$$O(jDn \log(jDn)(\log L + D \log(D \cdot jDn))) = O(jDn(\log n)(\log L + D \log n))$$

time for each j, since $j = O(n)$. Thus the total time complexity is

$$O\left(\sum_{j=1}^{h} jDn(\log n)(\log L + D \log n)\right) = O(Dh^2 n(\log n)(\log Ln^D)).$$

A node label is represented by $O(\log L)$ bits. An edge label is a pair of port numbers, which are $O(\log D)$-bit values. Therefore the minimal f-view of depth j can be expressed by $O(j \cdot n \cdot \log L + jDn \cdot \log D) = O(j \cdot n \log(LD^D))$ bits. Hence, the sum of the bits exchanged by all parties is $O(j \cdot |E| n \log(LD^D))$ for each j. It follows that the total communication complexity to construct an f-view of depth h is

$$O\left(\sum_{j=1}^{h} (j|E|n \log(LD^D))\right) = O\left(|E|h^2 n \log(LD^D)\right).$$

□

Remark 7 *Kranakis and Krizanc [131] gave a simple but efficient view construction algorithm with only slightly worse communication complexity. Their algorithm, however, requires $O(n^2)$ rounds to construct an $O(n)$-depth view for an n-party distributed system, while ours needs only $O(n)$ rounds.*

4.4.5 Counting the number of parties having specified values

After constructing $\widetilde{T}_X^{2(n-1)}(v)$, we often need to compute $|\Gamma_X^{(n-1)}|$ and/or $|\Gamma_X^{(n-1)}(S)|$ for a subset S of the range of X to obtain $c_X = n/|\Gamma_X^{(n-1)}|$ or $c_X(S) = n|\Gamma_X^{(n-1)}(S)|/|\Gamma_X^{(n-1)}|$. We will give an algorithm that computes $|\Gamma_X^{(n-1)}(S)|$ from given minimal f-view $\widetilde{T}_X^{2(n-1)}(v)$, S, and n. By setting S to the range of X, we can also obtain $|\Gamma_X^{(n-1)}|$. Hereafter, we use "*a sub-f-view rooted at u*" to indicate the subgraph of an f-view induced by the set of nodes which includes node u and all nodes that can be reached from u via directed edges.

The algorithm

The algorithm computes the maximal set W of those nodes of depth at most $n-1$ which define distinct path sets of length $n-1$. Then $|\Gamma_X^{(n-1)}(S)|$ is computed

by counting the number of those nodes in W which are labeled with values in S. The algorithm traverses input f-view $\widetilde{T}_X^{2(n-1)}(v)$ in a breadth first manner, in which each visit to a new node invokes a subroutine that traverses sub-f-views of $\widetilde{T}_X^{2(n-1)}(v)$ in a breadth-first manner.

To help the breadth-first traversals and simplify the description, the algorithm first calls Subroutine Traversal (II) to prepare the next objects by a breadth-first traversal of $\widetilde{T}_X^{2(n-1)}(v)$: (1) the size, **size**, of $\widetilde{T}_X^{2(n-1)}(v)$, (2) function

$$\mathsf{depth} : V^f \to \{0, 1, \ldots, 2(n-1)\}$$

that maps any node in node set V^f of $\widetilde{T}_X^{2(n-1)}(v)$ to their depth, (3) the bijective mapping

$$\mathsf{id} : V^f \to \{1, \ldots, |V^f|\}$$

such that $\mathsf{id}(u_1) < \mathsf{id}(u_2)$ if $\mathsf{depth}(u_1) < \mathsf{depth}(u_2)$ for any $u_1, u_2 \in V^f$, and (4) the inverse mapping id^{-1} of id.

To compute W, the algorithm first sets W to $\{u_r\}$, where u_r is the root of $\widetilde{T}_X^{2(n-1)}(v)$, and repeats the following operations for each i (≥ 2) in an increasing order until the depth of the ith node ($= \mathsf{id}^{-1}(i)$) is larger than $n-1$: for each node u in W, the algorithm calls Subroutine P (described later) to test if the sub-f-view rooted at $\mathsf{id}^{-1}(i)$ has the same path set of length $n-1$ as that rooted at u, and sets $W := W \cup \{\mathsf{id}^{-1}(i)\}$ if the test is false (i.e., the two sub-f-views do not have the same path set of length $n-1$). At the ith repetition, W is clearly the subset of $\{\mathsf{id}^{-1}(j) \mid j < i\}$ such that no pair of sub-f-views rooted at nodes in W has a common path set of length $n-1$.

The entire algorithm is precisely described in Figure 60. Subroutine Traversal (II) is precisely shown in Figure 61, where `CONTINUE` starts the new turn of the inner-most loop where it lies with the updated index; `BREAK` quits the inner-most loop and moves on to the next operation; `DEQUEUE`(Q) removes an element from FIFO queue Q and returns the element; `ENQUEUE`(Q, q) appends q to Q. These operations are assumed to be implemented in a standard way.

Subroutine P is based on the next lemma.

Lemma 55 *Suppose that $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ are any two sub-f-views of depth $(n-1)$ of a minimal f-view $\widetilde{T}_X^{2(n-1)}(v)$, such that $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ are rooted at nodes u_r and w_r of $\widetilde{T}_X^{2(n-1)}(v)$, respectively, and $\mathsf{depth}(u_r) \leq \mathsf{depth}(w_r) \leq (n-1)$. Let V_a and V_b be the vertex sets of $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$, respectively, and let E_a and E_b be the edge sets of $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$, respectively.*

$\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ have a common path set of length $(n-1)$, if and only if there is a unique homomorphism ϕ from V_a onto V_b such that,

C1: *for each $u \in V_a$, $\phi(u)$ has the same label as u,*

View Counting Algorithm

Input: minimal f-view $\widetilde{T}_X^{2(n-1)}(v)$, subset S of the range of X, and integer n
Output: $|\Gamma_X^{(n-1)}(S)|$

1. Call Subroutine Traversal (II) with $\widetilde{T}_X^{2(n-1)}(v)$ to compute the size, $\mathbf{size}$, of $\widetilde{T}_X^{2(n-1)}(v)$, depth, id and id^{-1}.
2. Let W be $\{u_r\}$, where u_r is the root of $\widetilde{T}_X^{2(n-1)}(v)$.
3. For $i := 2$ to $\mathbf{size}$, perform the next operations.
   3.1 If $\mathsf{depth}(\mathsf{id}^{-1}(i)) > n-1$, `BREAK`.
   3.2 For each $u \in W$,
   call Subroutine P with n, depth, id, and two sub-f-views rooted at $\mathsf{id}^{-1}(i)$ and u, to test if the two sub-f-views have the same path set of length $n-1$,
   set $W := W \cup \{u\}$ if the result is "No."
4. Count the number n_S of those nodes in W which are labeled with some value in S.
5. Output n_S.

Figure 60: View counting algorithm.

C2: *for each $u \in V_a$, there is an edge-label-preserving bijective mapping from the set of outgoing edges of u to the set of outgoing edges of $\phi(u)$ such that any outgoing edge (u, u') of u is mapped to $(\phi(u), \phi(u'))$.*

Proof. ($\Rightarrow$) Let $\phi' : V_a \to V_b$ be the mapping defined algorithmically as follows. We first set $\phi'(u_r) := w_r$. We then repeat the next operations for each j from 0 to $(n-1)-1$: for every node $u \in V_a$ of depth $(j + \mathsf{depth}(u_r))$ and every edge $(u, u') \in E_a$, set $\phi'(u') := w' \in V_a$ if (u, u') and $(\phi'(u), w') \in E_b$ have the same label. Notice that $\phi'(u)$ has been already fixed, since the above operations proceed toward leaves in a breadth-first manner.

We will prove that, if $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ have a common path set of length $(n-1)$, ϕ' is well-defined, meets C1 and C2, and also an onto-mapping. Finally, we prove the uniqueness of ϕ satisfying C1 and C2.

Suppose that $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ have a common path set of length $(n-1)$. We prove that ϕ' is well-defined and meets C1 and C2 by induction with respect to depth.

Clearly, ϕ' is well-defined for u_r, and meets C1 for u_r. Furthermore, u_r and w_r define the same path set of length $2(n-1) - \mathsf{depth}(w_r)$, since two views of

Subroutine Traversal (II)

Input: minimal f-view $\widetilde{T}_X^{2(n-1)}(v)$.

Output: variable $\mathbf{size}$ of $\widetilde{T}_X^{2(n-1)}(v)$, and functions depth, id and id^{-1}.

1. $\texttt{ENQUEUE}(Q, u_{\mathrm{root}})$, where Q is an empty first-in-first-out queue, and u_{root} is the root of $\widetilde{T}_X^{2(n-1)}(v)$.
   Set $\mathbf{size} := 1$.
2. Set $\mathsf{id}(u_{\mathrm{root}}) := \mathbf{size}$ and $\mathsf{id}^{-1}(\mathbf{size}) := u_{\mathrm{root}}$.
   Set $\mathsf{depth}(u_{\mathrm{root}}) := 0$.
3. Set $\mathbf{size} := \mathbf{size} + 1$.
4. While Q is not empty, repeat the following steps.
   4.1 Set $u := \texttt{DEQUEUE}(Q)$.
   4.2 If u is a leaf, `CONTINUE`.
   4.3 For $i := 1$ to d_u, do the following.
      4.3.1 Set $u_i := \mathsf{Adj}_i(u)$.
      4.3.2 If u_i has already been traversed, `CONTINUE`.
      4.3.3 Set $\mathsf{id}(u_i) := \mathbf{size}$ and $\mathsf{id}^{-1}(\mathbf{size}) := u_i$.
      Set $\mathsf{depth}(u_i) := \mathsf{depth}(u) + 1$.
      4.3.4 $\texttt{ENQUEUE}(Q, u_i)$.
      4.3.5 Set $\mathbf{size} := \mathbf{size} + 1$.
5. Output $\mathbf{size}$, depth, id and id^{-1}.

Figure 61: Subroutine Traversal (II).

infinite depth are isomorphic if and only if the two views are isomorphic up to depth $(n-1)$. Thus, ϕ' is well-defined for every node incident to an outgoing edge of u_r (i.e., every node at depth $(1 + \mathsf{depth}(u_r))$, and meets C2 for u_r.

For $j \geq 1$, we assume that for any $u \in V_a$ at depth not more than $(j + \mathsf{depth}(u_r))$, ϕ' is well-defined, meets C1, and u and $w = \phi'(u)$ define the same path set of length $(2(n-1) - \mathsf{depth}(w))$. Further, we assume that ϕ' meets C2 for any $u \in V_a$ at depth not more than $((j-1) + \mathsf{depth}(u_r))$. For any fixed $u \in V_a$ at depth $(j + \mathsf{depth}(u_r))$ and every edge $(u, u') \in E_a$, there is a node w' that satisfies the condition of the above algorithmic definition (i.e., (u, u') and $(\phi'(u), w')$ have the same label), since u and $w = \phi'(u)$ define the same path set of length $(2(n-1) - \mathsf{depth}(w))$. If we set $\phi'(u') := w'$, ϕ' meets C1 for u' and C2 for u, and u' and w' have the same path set of length $(2(n-1) - \mathsf{depth}(w'))$. To show that ϕ' is well-defined for any node at depth $((j+1) + \mathsf{depth}(u_r))$, we have

to prove that, for any node u_1, u_2 at depth $j + \mathsf{depth}(u_r)$, there are no two edges $(u_1, u'), (u_2, u') \in E_a$ that induce two distinct images of u' by ϕ'. We assume that such two edges exist and let w'_1 and w'_2 be the distinct images of u'. This implies that path set $P_X^{2(n-1)-\mathsf{depth}(w'_1)}(w'_1)$ is identical to $P_X^{2(n-1)-\mathsf{depth}(w'_2)}(w'_2)$, since both of them are identical to the path set of length $2(n-1) - \mathsf{depth}(w'_1)$ $(= 2(n-1) - \mathsf{depth}(w'_2))$ defined for u'. This contradicts Corollary 51, since w'_1 and w'_2 are nodes at the same depth of minimal f-view $\widetilde{T}_X^{2(n-1)}(v)$.

If we assume that ϕ' is not an onto-mapping, there is at least one node in $V_b \setminus \{\phi'(u) \mid u \in V_a\}$. Let w be the node of the smallest depth in $V_b \setminus \{\phi'(u) \mid u \in V_a\}$. Thus, the source of any incoming edge of w is in $\{\phi'(u) \mid u \in V_a\}$. Since the source satisfies C2, w needs to be in $\{\phi'(u) \mid u \in V_a\}$. This is contradiction.

Finally, we prove the uniqueness of ϕ. We assume that there are two different homomorphisms satisfying C1 and C2. Let ϕ_1 and ϕ_2 be the two homomorphisms. Then there is at least one node u in V_a such that $\phi_1(u) = w_1 \neq w_2 = \phi_2(u)$. Since any $(n-1)$-length directed path emanates from w_r in $\acute{T}_{X,b}^{(n-1)}$, $\phi_1(u_r) = \phi_2(u_r) = w_r$. For any directed path from u_r to u, ϕ_1 and ϕ_2 define a path from w_r to w_1 and a path from w_r to w_2, respectively. By C1 and C2, these two paths are both isomorphic to the path from u_r to u. Since there is at most one such path in $\acute{T}_{X,b}^{(n-1)}$ by the definition of f-views, w_1 must be identical to w_2. This is contradiction.

$(\Leftarrow)$ If ϕ meets C1 and C2, any directed edge (u, u') in $\acute{T}_{X,a}^{(n-1)}$ is mapped to a directed edge $(\phi(u), \phi(u'))$ in $\acute{T}_{X,b}^{(n-1)}$ of the same edge and node labels. It follows that any directed path in $\acute{T}_{X,a}^{(n-1)}$ is mapped to an isomorphic directed path in $\acute{T}_{X,b}^{(n-1)}$. Thus, any $(n-1)$-length directed path from u_r in $\acute{T}_{X,a}^{(n-1)}$ has to be mapped to some isomorphic $(n-1)$-length directed path in $\acute{T}_{X,b}^{(n-1)}$. Therefore, the path set of $\acute{T}_{X,a}^{(n-1)}$ is a subset of that of $\acute{T}_{X,b}^{(n-1)}$.

Conversely, fix an $(n-1)$-length directed path p starting at w_r. Since any $(n-1)$-length directed path in $\acute{T}_{X,a}^{(n-1)}$ is mapped to some $(n-1)$-length directed path in $\acute{T}_{X,b}^{(n-1)}$, u_r is only the preimage of w_r by ϕ. If u is a preimage of the jth node on p with respect to ϕ, there is only one preimage of the $(j+1)$st node on p among nodes incident to the outgoing edges of u due to C2. By induction, a unique path is determined as the preimage of p. Thus, the path set of $\acute{T}_{X,b}^{(n-1)}$ is a subset of that of $\acute{T}_{X,a}^{(n-1)}$.

If there are at least two homomorphisms that meet C1 and C2, $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ have a common path set from the above argument. It follows that there is a uniqe homorphism that meets C1 and C2 by the $(\Rightarrow)$ part. This is contradiction. □

Lemma 55 implies that, if we can construct ϕ' (defined in the proof) that meets

Subroutine P

Input: an integer n;
functions depth that gives the depth of nodes in $\widetilde{T}_X^{2(n-1)}(v)$, and id;
two sub-f-views, $\acute{T}_{X,a}^{(n-1)}$ rooted at u_r and $\acute{T}_{X,b}^{(n-1)}$ rooted at w_r, of a minimal f-view $\widetilde{T}_X^{2(n-1)}(v)$ such that $\mathsf{depth}(u_r) \leq \mathsf{depth}(w_r) \leq n-1$

Output: "Yes" or "No"

1. Perform $\texttt{ENQUEUE}(Q, u_r)$, where Q is an empty first-in-first-out queue.
2. Set $\phi(u_r) := w_r$.
3. While Q is not empty, repeat the following steps.
   3.1 Set $u := \texttt{DEQUEUE}(Q)$.
   3.2 If $\mathsf{depth}(u) > (n-1) + \mathsf{depth}(u_r)$, go to step 4.
   3.3 If $\mathsf{label}(u) \neq \mathsf{label}(\phi(u))$, go to step 5.
   3.4 If $\mathsf{depth}(u) = (n-1) + \mathsf{depth}(u_r)$, CONTINUE.
   3.5 If $d_u \neq d_{\phi(u)}$, go to step 5;
   otherwise, for $i := 1$ to d_u, perform the next steps.
      3.5.1 Set $u_i := \mathsf{Adj}_i(u)$ and $w_i := \mathsf{Adj}_i(\phi(u))$.
      3.5.2 If $\mathsf{label}((u, u_i)) \neq \mathsf{label}((\phi(u), w_i))$, go to step 5.
      3.5.3 If u_i has already been traversed and $\mathsf{id}(\phi(u_i)) \neq \mathsf{id}(w_i)$, go to step 5; otherwise set $\phi(u_i) := w_i$.
      3.5.4 $\texttt{ENQUEUE}(Q, u_i)$.
4. Halt and output "Yes."
5. Halt and output "No."

Figure 62: Subroutine P.

C1 and C2, $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ have a common path set of length $(n-1)$. Conversely, if we cannot construct ϕ', there is no mapping ϕ that satisfies C1 and C2; $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ do not have a common path set of length $(n-1)$.

As described in the proof, ϕ' can be constructed by simultaneously traversing $\acute{T}_{X,a}^{(n-1)}$ and $\acute{T}_{X,b}^{(n-1)}$ in a breadth-first manner. Figure 62 gives a precise description of Subroutine P, where ENQUEUE, DEQUEUE and CONTINUE are defined in the same way as in the case of Subroutine Traversal (II), and are assumed to be implemented in a standard way.

Lemma 56 *Suppose that minimal f-view $\widetilde{T}_X^{2(n-1)}(v)$ is a view of a distributed system of n parties having $O(\log L)$-bit values. Given two sub-f-views $\acute{T}_{X,a}^{(n-1)}$ and*

$\acute{T}^{(n-1)}_{X,b}$ of depth $(n-1)$ of a minimal f-view $\widetilde{T}^{2(n-1)}_X(v)$, Subroutine P outputs "Yes" if and only if $\acute{T}^{(n-1)}_{X,a}$ and $\acute{T}^{(n-1)}_{X,b}$ have a common path set of length $(n-1)$. The time complexity is $O(n^2 \log(n^D L))$, where D is the maximum degree over all nodes of the underlying graph of the distributed system.

Proof. Subroutine P constructs $\phi = \phi'$ (defined in the proof of Lemma 55). Subroutine P outputs "Yes" only when the subroutine visits some node at depth $n + \mathsf{depth}(u_r)$ or Q is empty. The former case implies that the subroutine has already visited all nodes in $\acute{T}^{(n-1)}_{X,a}$, since the subroutine visits nodes by breadth-first traversal. The latter case occurs only when $(n-1) + \mathsf{depth}(u_r)$ is $2(n-1)$ (i.e., the bottom level of $\acute{T}^{(n-1)}_{X,a}$ is identical to that of $\widetilde{T}^{2(n-1)}_X(v)$), and this implies again that the subroutine has already visited all nodes in $\acute{T}^{(n-1)}_{X,a}$. It follows that, when Subroutine P outputs "Yes," ϕ meets C1 of Lemma 55 (due to step 3.3) and C2 (due to the first part of step 3.5 and step 3.5.2). Thus, $\acute{T}^{(n-1)}_{X,a}$ and $\acute{T}^{(n-1)}_{X,b}$ have a common path set of length $(n-1)$ by Lemma 55. Conversely, if $\acute{T}^{(n-1)}_{X,a}$ and $\acute{T}^{(n-1)}_{X,b}$ have a common path set of length $(n-1)$, the subroutine outputs "Yes," by the *only-if* part in the proof of Lemma 55. This proves the correctness.

Let V_a and E_a be the edge set and node set, respectively, of $\acute{T}^{(n-1)}_{X,a}$. Step 3 is obviously dominant in terms of time complexity. Step 3.1 takes just constant time for each evaluation. Steps 3.2 and 3.4 take $O(\log n)$ time for each u; they take $O(|V_a| \log n)$ time in total. Step 3.3 takes $O(\log L)$ time for each u since node labels are $O(\log L)$-bit values; it takes $O(|V_a| \log L)$ time in total. The first line of step 3.5 takes $O(d_u)$ time for each u; it takes $O(|E_a|)$ time in total. Steps 3.5.1 and 3.5.4 take constant time. Each execution of step 3.5.2 takes $O(\log D)$ time, since edge labels are $O(\log D)$-bit values. Each excecution of step 3.5.3 takes $O(\log n)$ time, since $\widetilde{T}^{(n-1)}_X$ has $O(n^2)$ nodes. Since every edge is visited exactly once, step 3.5 takes $O(|E_a| \log n)$ time in total. The time complexity of step 3 is thus $O(|V_a| \log(nL) + |E_a| \log n)$; this is $O(n^2 \log(Ln^D))$ since $|V_a| = O(n^2)$ and $|E_a| = O(n^2 D)$. □

The correctness and complexity of the view counting algorithm is described in the next lemmas.

Lemma 57 *Given a minimal f-view $\widetilde{T}^{2(n-1)}_X(v)$, a subset S of the range of X, and the number n of parties, the algorithm correctly outputs $|\Gamma^{(n-1)}_X(S)|$.*

Proof. Let $\widetilde{\mathcal{C}}$ be the collection of distinct path sets of length $(n-1)$ defined for all nodes u^j at depth j over all $j \leq n-1$ in f-view $\widetilde{T}^{2(n-1)}_X(v)$, and let $\mathcal{C}$ be $\bigcup_{j \leq n-1} \bigcup_{u_j} P^{(n-1)}_X(u^j)$ where u^j denotes a node at depth j in $T^{2(n-1)}_X(v)$. Suppose that n'_S is the number of those path sets in $\widetilde{\mathcal{C}}$ of which the first node is labeled with

some value in S. Since $\widetilde{\mathcal{C}}$ is identical to $\mathcal{C}$ by Corollary 50, $n'_S = |\Gamma_X^{(n-1)}(S)|$. By Lemma 56, n'_S is equal to the number n_S of those nodes in W which are labeled with values in S. This completes the proof. □

Lemma 58 *For a given distributed system of n parties each of which has a value of $O(\log L)$ bits, $|\Gamma_X^{(n-1)}|$ and $|\Gamma_X^{(n-1)}(S)|$ for any subset S of the range of X can be computed from $\widetilde{T}_X^{2(n-1)}(v)$ in $O(n^5 \log(n^D L))$ time for each party, where D is the maximum degree over all nodes of the underlying graph.*

Proof. The dominant part of Subroutine Traversal (II) is step 4 in Figure 61, which performs a simple bread-first traversal of $\widetilde{T}_X^{2(n-1)}(v)$. The traversal takes $O(\log |V^f|)$ time for each edge. Thus, the time complexity of step 1 in Figure 60 is $O(n^2 D \log |V^f|)$.

Next we analyze step 3 of the view counting algorithm in Figure 60. We can see that (1) $|W|$ is at most n since there are n parties in the system, and (2) there are $O(n^2)$ nodes whose depth is at most $n-1$ in $\widetilde{T}_X^{2(n-1)}(v)$ since there are at most n nodes at each depth. Hence, Subroutine P is called for each of $O(n^3)$ pairs of sub-f-views. Since one call of Subroutine P takes $O(n^2 \log(n^D L))$ time by Lemma 56, step 3.2 thus takes $O(n^5 \log(n^D L))$ time; this is also sufficient to perform the other operations.

The total time complexity is thus $O(n^5 \log(n^D L))$. □

4.5 Summary

It is well-known that LE_n in the anonymous setting cannot be solved classically in a deterministic sense for a certain broad class of network topologies such as regular graphs, even if all parties know the exact number of parties. We have proposed two quantum algorithms that exactly solve the problem when each party knows the number of parties. The two algorithms each have their own characteristics.

The first algorithm runs in $O(n^2)$ rounds for any topology of anonymous networks. The communication complexity of this algorithm is $O(n^4)$, but it requires quantum communication of $O(n^4)$ qubits. The local computation requires $O(n^3)$ steps for each party.

The second algorithm has communication complexity $O(n^5(\log n)^2)$, but involves quantum communication of just $O(n^2 \log n)$ qubits of one round. Furthermore the second algorithm runs in $O(n \log n)$ rounds. As for local computation time, the algorithm requires $O(n^6 \log n)$ time for each party. To reduce the amount of quantum communication, the second algorithm makes use of a classical technique, called *view*. However, a naïve application of view incurs exponential classical time/communication complexity. To reduce time/communication complexity,

we introduced the view compression technique, which enables views to be constructed in polynomial time and communication. The technique can be used to deterministically check if the unique leader is selected or not in polynomial time and communication. As another application of this techinique, we can construct zero-error probabilistic algorithms requiring expected polynomial time and communication for any topology by combining the technique and usual coin flipping.

Furthermore, our leader election algorithms can solve the problem exactly even when each party knows only the upper bound of the number of parties. In this setting for any topology with cycles, no zero-error probabilistic algorithms can solve the problem. From anther point of view, our algorithms can exactly solve the problem when the underlying graph is not connected: for a given n-distributed system that consists of multiple connected components, elect a unique leader in each component for a given n.

As a generalization in terms of the underlying graph, the second algorithm can easily be modified so that it can be applied even when the underlying graph is directed (and strongly connected). In contrast, such modification is not easy for the first algorithm, since, at some points, the first algorithm needs to invert the computation and communication that were performed in the previous steps.

Our algorithms use unitary gates depending on the number of parties that are eligible for a leader during their execution. Thus, the algorithms require an elementary gate set of size $O(n)$ for the number n of parties. From a practical point of view, however, it would be desirable to perform leader election for any n by using a constant-sized set of elementary unitary gates. It is open whether the leader election problem in an anonymous network can exactly be solved in the quantum setting by using a constant-sized set of elementary gates.

Improving the complexity of solving the problem would also be interesting. In general, however, it is difficult to optimize both communication complexity and round complexity (i.e., the number of rounds required). A reasonable direction is to clarify the tradeoff between them. As for communication complexity, it is also a natural open question what kind of tradeoff between quantum and classical communication complexity exists.

It is also open whether the problem can be solved by a processor terminating algorithm (i.e., the algorithm that terminates when every party goes into a halting state) in the quantum setting even without knowing the upper bound of the number of parties. In this situation, there are just message terminating algorithms with bounded error in the classical setting.

4.6 A brief survey of quantum distributed computing

Quantum bits (or qubits) are the units of quantum information. Holevo [101] proved that qubits cannot be used to compress classical messages better than with

classical bits: an n-bit classical message needs n qubits to be stored or sent via a communication channel. If two parties share prior entanglement, one party can transfer an n-bit classical message to the other by $n/2$-qubit communication via *superdense coding*, which was found by Bennet and Wiesner [45], but cannot compress the message more. Interestingly, however, quantum bits can significantly reduce the amount of communication when we perform certain distributed computational tasks.

The quantum communication complexity model first proposed by Yao [198] is one of the extensively studied areas in quantum distributed computing as a quantum counterpart of the classical communication complexity model, which was introduced by Abelson [12] and Yao [197]. While there are a lot of variations of the model with respect to the number of parties, the quantumness of links and prior entanglement, the standard setting is that two parties which are connected via a bidirectional quantum communication link, get inputs $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, respectively, and have to output $f_n(x,y) : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, by communication and local computation. In what follows, we denote the ith bit of x (y) by x_i (resp. y_i).

For exact computation, Buhrman, Cleve and Wigderson [58] showed an exponential gap between quantum and classical models for the equality function with the promise that the Hamming distance between x and y is either 0 or $n/2$, while the bounded-error classical communication complexity for this function is just $O(1)$. However, for any total function f_n, Kremer [132] showed that the quantum communication complexity in the exact setting has a lower bound $(\log rank(M_{f_n}))/2$, and Buhrman and de Wolf [59] proved that this bound is true even in the case where parties share an unlimited number of prior EPR pairs, i.e., $(|00\rangle + |11\rangle)/\sqrt{2}$, where communication matrix M_{f_n} is the $2^n \times 2^n$ matrix whose (x,y) element is b if $f_n(x,y) = b$, and $rank(M_{f_n})$ is the rank of M_{f_n} (this is quite similar to a lower bound $\log rank(M_{f_n})$ in the classical case [143]). This implies that, for any function f_n of $rank(M_{f_n}) = n$, quantum communication can reduce communication complexity by at most half. De Wolf [76] gave the tight lower bound $n+1$ of exactly computing the equality function and the disjoint function (defined later) by proving a general lower bound of non-deterministic quantum communication complexity.

For the zero-error setting, Buhrman, Cleve, de Wolf and Zalka [55] showed a total relation problem with n-bit input given to each party, for which, there exists a zero-error $O(n^{2/3}\log n)$-qubit quantum protocol, whereas any zero-error classical protocol needs $\Omega(n)$ bits of communication. Subsequently, Klauck [127] gave a polynomial gap for a total Boolean function f_n with n-bit input given to each party: there is a quantum zero-error protocol for f_n with $O(n^{10/11+\epsilon})$ qubits of communication for any $\epsilon > 0$, whereas every classical zero-error protocol for f needs $\Omega(n/\log n)$ bits of communication.

In the case of bounded-error computation, Buhrman, Cleve and Wigderson [58] proved by using Grover search [96] that, the disjoint function DISJ$=\bigwedge_{i=1}^{n}\overline{x_i \wedge y_i}$, which is known to be in co $\mathcal{NP}^{cc}$-complete [28], has the quantum communication complexity of $O(\sqrt{n}\log n)$, while the classical communication complexity is known to be $\Omega(n)$ [122]. This was improved to $O(\sqrt{n}c^{\log^\star n})$ for a constant c by Høyer and de Wolf [102], and $O(\sqrt{n})$ by Aaronson and Ambainis [10], which matches the lower bound $\Omega(\sqrt{n})$ given by Razborov [167]. Raz [166] obtained an exponential gap in the bounded-error setting between quantum and public-coin classical models for a partial function P (in which the set of possible inputs for each player is infinite, but output is a Boolean value). Subsequently, Bar-Yossef, Jayram and Kerenidis [34] gave an exponential gap in the case restricted to one-way bounded-error communication (allowing public coins only in the classical setting) for a relation problem. Ambainis, Achulamn, Ta-Shma, Vazirani and Wigderson [26] gave an exponential separation in the bounded error setting for the sampling model: two parties get no inputs, and each have to output $(x, f_n(x, y))$ and $(y, f_n(x, y))$ for x and y chosen according to some prespecified distribution. Tani, Nakanishi and Yamashita [185] extended the public-to-private classical coin conversion technique [151] to the quantum case in order to give a quantum protocol that computes a total Boolean function more efficiently than in the classical case. On the other hand, there is a function for which the quantum model has as high communication complexity as the classical one has: for function IP$=\bigoplus_{i=1}^{n}(x_i \wedge y_i)$, Kremer [132] and Yao proved by applying the discrepancy method that the bounded-error quantum communication complexity is $\Omega(n)$, and Cleve, van Dam, Nielsen and Tapp [69] proved that the complexity is $\Omega(n)$ even with an unlimited number of prior EPR-pairs.

Non-deterministic quantum communication complexity was considered in de Wolf [76]. He proved that non-deterministic quantum communication complexity of function f_n is equal to the logarithm of the rank of a non-deterministic version of communication matrix M_{f_n}. He also gave total functions for which the non-deterministic quantum communication complexity is exponential smaller than the classical one. His definition of non-determinism is somewhat different from the standard definition and is shown to be strictly stronger than the latter [76]: non-deterministic communication complexity is the worst case communication complexity of the optimal protocol that can compute a function with some positive probability.

Klauck, Nayak, Ta-Shma and Zuckerman [128] considered the tradeoff between the number of rounds and the number of qubits that have to be exchanged, and showed that there is a problem such that its $(k+1)$-round classical communication complexity is exponentially smaller than its k-round quantum communication complexity in the bounded-error setting. They also obtained an $\Omega(n^{1/k})$ lower bound for k-round bounded-error protocols for DISJ, which generalizes an $\Omega(n)$

lower bound of one-round protocols for DISJ in [59, 127]. Further, they gave an exponential separation between k and $k+1$ round bounded-error protocols for the pointer jumping function when k is small (the pointer jumping function has been studied a lot in the past to show rounds versus communication tradeoffs in classical communication complexity). Subsequently, Jain, Radhakrishnan and Sen [119] gave, for all k, an exponential separation between k and $k+1$ round bounded-error protocols for (a slightly different version of) the pointer jumping function. Later, Jain, Radhakrishnan and Sen [120] proved a lower bound of a multi-party version of DISJ and gave as a corollary $\Omega(n/k^2)$ lower bound for two-party k-round protocols for DISJ, which is better than Razborov's bound $O(\sqrt{n})$ when $k = o(n^{\frac{1}{4}})$.

The simultaneous message passing (SMP) model is a three-party one-way communication model, where two party are each given input x and y and send one message to the third party called a referee, who receives the messages from the two parties and computes a function $f(x, y)$ and outputs the result. Buhrman, Cleve, Watrous, de Wolf [57] proved an exponential gap for the equality function of n bits in the SMP model: the bounded-error quantum communication complexity is $O(\log n)$, while the classical one (without public coins) is $\Theta(\sqrt{n})$ [22, 152]. Yao [199] generalized this result: for any n-bit function for which there is a classical bounded-error public-coin protocol of $O(1)$-bit communication, there is a quantum bounded-error protocol of $O(\log n)$-qubit communication. (Note that the equality function can be computed in $O(1)$ communication complexity in the SMP model.) Bar-Yossef, Jayram and Kerenidis [34] gave an exponential separation even when classical protocols are allowed to use public coins for a relation problem instead of Boolean functions.

Quantum entanglement is one of the most remarkable notion of quantum computation and communication. Entangled particles exhibit characteristic effects, "nonlocal effects", which Einstein, Podolsky and Rosen [85] first alluded to. Unfortunately, this does not mean that entanglement contributes to transferring information to a physically separated place, as described previously. On the other hand, it is well-known that public coins can significantly reduce the communication required to compute certain functions such as the equality function. Motivated by this fact, entanglement has been studied in the communication complexity scenario, where two parties connected by a classical communication link share prior entanglement. Cleve and Buhrman [68] proved that, three parties with prior entanglement can exactly compute a certain promise Boolean function by two bits of communication, whereas they require three bits of communication without prior entanglement. Buhrman and de Wolf [59] proved a general lower bound $\log rank(M_{f_n})$ for any Boolean function f_n. Buhrman, Cleve and van Dam [56] proved that two parties with prior entanglement can produce a certain correlation by two bits of communication with higher probability than those who share a

random string instead of quantum entanglement.

For multiparty communication, Buhrman, van Dam, Høyer and Tapp [60] gave a separation by a logarithmic factor in the number of parties.

Quantum distributed computing has also been extensively studied in cryptographic context (e.g., [35, 73, 24, 25, 148, 42]), where parties have to do some computational tasks without being affected by *cheating parties* as little as possible. This research field has its own characteristics in the sense that algorithms or protocols are evaluated by security measures as well as efficiency measures. This paper does not give further information on this field.

5 Conclusion

The thesis has made theoretical and practical contributions to improve the efficiency of major types of multi-point communication on the present network, and to assess a new type of multi-point communication that would be a crucial component in the future network.

With regard to multi-point communication on the present network, one-to-many communication is an essential building block whose improvement in communication efficiency has a great impact on various multi-point communication. We have first discussed two major types of one-to-many multi-point communication: file-transfer type multi-point communication and streaming type multi-point communication.

To improve the efficiency of the file-transfer type multi-point communication, the most popular strategy is *network caching*. We described on-line algorithms that solve one of the most important problems of network caching: the file caching problem. For the bit model, our purpose was to develop performance-guaranteed on-line algorithms that can achieve high hit-rate with the help of heuristics, since LRU does not fully utilize the information on file size (although it is competitive and achieve relatively high-hit rate in practical environments). We have given a simple but sufficient condition of competitiveness of deterministic on-line algorithms and developed a general framework by using the condition to easily construct a competitive algorithm from a non-competitive heuristics. The constructed algorithm adaptively switches from the heuristics to LRU and vice versa according to the characteristics of input requests. As an application of the framework, we constructed a competitive algorithm, called Competitive_SIZE, from SIZE heuristics. We confirmed its performance by conducting event-driven simulations and trace-driven simulations using a real web proxy log. As a future work, it would be interesting to introduce a modification to competitive analysis so that it can handle the distribution of input request sequences, such as access graphs and the Markov chain model, which were introduced to precisely analyze the paging problem.

For the general model, an optimal deterministic competitive algorithm, Greedy-Dual-Size(GDS), is known to achieve excellent performance even in experiments. GDS with storage size k, however, takes as many as $O(k)$ steps per file eviction in the worst case. We have given a fast randomized on-line algorithm that runs in $O(\log k)$ time and is expected to work in only $O(2^{\log^* k})$ time per file eviction or insertion, while the algorithm is expected to be $\frac{k}{k-h+1}$-competitive, i.e., as competitive as GDS. We conducted trace driven simulations by using real web proxy logs to confirm its practicality. Experimental results show that our algorithm performs as well as GDS when file cost is set to communication latency required to get a file. As a future work, it is an interesting open problem to settle the competitive ratio of optimal randomized on-line algorithms. Another interest-

ing direction is to develop a framework to construct a competitive algorithm from a non-competitive heuristics, as we have done in the bit model.

For both of the bit and general models, it is $\mathcal{NP}$-hard to solve the file caching problem in the off-line setting. It is a well-known open problem to settle the hardness in the fault model.

As for the streaming type multi-point communication, we have proposed an IP-unicast-based multicast, called Flexcast, that works across the network and application layers; it enables progressive deployment in an autonomous and distributed fashion over existing IP networks that support only unicast, and solves the issue of preserving the uniqueness of multicast group addresses. Flexcast constructs a multicast tree by sharing the maximal common segments of any pair of reverse-paths and maintains it by a hierarchical keep-alive mechanism. Flexcast protocol provides significant adaptability against the change of unicast routes, the moving of receivers and senders, and the frequent joining and leaving of receivers, with minimum rearrangement of the tree. We have also discussed the issues relating to many IP-unicast-based multicast protocols, including the basic protocol of Flexcast, and proposed two modifications to the Flexcast protocol against the issues.

Streaming experiments were conducted to confirm the robustness and stability of the Flexcast protocol when used to deliver streaming data to widely dispersed locations; streaming servers and receivers were placed at three locations: Yokosuka in Japan, and Chicago and Los Angeles in the U.S..

We have a couple of future works concerning protocol design.

We have shown a node-load balancing mechanism of Flexcast. However, this may cause traffic congestion on particular communication links. Thus, it is important to improve the mechanism so that it can decide where to split streaming data according to the conditions of communication links as well as those of nodes. Obviously, a possible option is to control admission to the multicast tree. This is exactly an on-line problem similar to the file caching problem, since every node can decide to which receivers it should send streaming data whenever it gets requests from a new node but cannot accommodate more. Another future work is concerned with reasonably setting expiration time in the delivery table at every Flexcast node. We have proposed a way of automatically setting the expiration time: our method sets the expiration time to a certain interval that is linear to the distance from the node to the farthest descendant. In a stable network, however, this method could be too pessimistic. It would be interesting to devise a more appropriate way of setting the expiration time in a stable network, say, by a probabilistic method.

We have mainly considered one-to-many communication, since many-to-many communication often consists of one-to-many or many-to-one communications, and many-to-one communication intrinsically involves little redundant

traffic. If, however, we take into account specific tasks that end hosts want to do, we may improve the efficiency of even many-to-one communication by performing some computation depending on the tasks at network nodes, as commented in Section 3. This approach seems to be effective if we can select a small number of typical applications, and can manage to avoid imposing heavy load on network nodes.

We have next considered multi-point communication on the future network based on quantum mechanics. To this end, we considered the leader election problem in an anonymous network; every party needs to work in a highly autonomous and distributed fashion to solve the problem, since parties are assumed to be completely identical to one another. It is well-known that the problem cannot be solved classically in a deterministic sense for a certain broad class of network topologies such as regular graphs, even if all parties know the exact number of parties. We have proposed two quantum algorithms that exactly solve the problem for any topology in time/communication/round complexity polynomially bounded in the number of parties, when each party knows the number of parties in advance. The two algorithms each have their own characteristics in terms of the above complexity measures. Furthermore, our leader election algorithms can exactly solve the problem even when each party knows only the upper bound of the number of parties in advance. In this setting of any classical networks with cycles, there are no zero-error probabilistic algorithms.

Theoretically, our quantum results imply the separation in computability between quantum and classical distributed environments. From a practical point of view, however, it is hard to exactly solve the problem for a large number of parties due to the limited precision of physical devices. In this sense, our quantum results are in a position to motivate us to develop efficient quantum protocols in practical environments.

There are some interesting open problems. One is whether the leader election problem in an anonymous network can exactly be solved in the quantum setting by using a constant-sized set of elementary gates, since this leads to computing by using physical devices of limited precision. Improving the complexity of solving the problem would also be interesting. In general, however, it is difficult to optimize both communication complexity and round complexity (i.e., the number of rounds required). A reasonable direction is to clarify the tradeoff between them. As for communication complexity, it is also a natural open question what kind of tradeoff between quantum and classical communication complexities exists. It is also open whether the problem can be solved by a processor terminating algorithm (i.e., every party goes into a halting state whenever the algorithm stops) in the quantum setting even without knowing the upper bound of the number of parties. In this situation, there are just message terminating algorithms with bounded error in the classical setting.

Our quantum results naturally raise the question: can quantum communication make a contribution to efficient data transfer? Obviously, this is impossible when we focus on one-way communication between two parties due to the Holevo's theorem. However, we do not know the answer to the question in more complicated situations relating to multi-point communication. Network caching and multicast both essentially duplicate data on the network to make communication efficient. A negative thing is that it is impossible to duplicate (or clone) quantum superposition while superposition is the most significant aspect of quantum communication and computation. It would be essential to overcome this contradiction.

References

[1] Squid web proxy cache. http://www.squid-cache.org/.

[2] Tcpdump public repository. http://www.tcpdump.org/.

[3] *Proceedings of the Thirty-Fifth Annual IEEE Symposium on Foundations of Computer Science*, 1994.

[4] *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, 1996.

[5] *Proceedings of the First NASA International Conference Quantum Computing and Quantum Communications*, volume 1509 of *Lecture Notes in Computer Science*. Springer, 1998.

[6] *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, 2001.

[7] *Proceedings of the Forty-Third Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[8] *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, 2002.

[9] *Proceedings of the Forty-Fourth Annual IEEE Symposium on Foundations of Computer Science*, 2003.

[10] Scott Aaronson and Andris Ambainis. Quantum search of spatial regions. In *Proceedings of the Forty-Fourth Annual IEEE Symposium on Foundations of Computer Science* [9], pages 200–209.

[11] Harold Abelson. Lower bounds on information transfer in distributed computations. In *Proceedings of the Nineteenth Annual IEEE Symposium on Foundations of Computer Science*, pages 151–158, 1978. (Journal version [12]).

[12] Harold Abelson. Lower bounds on information transfer in distributed computations. *Journal of the ACM*, 27(2):384–392, 1980. (Conference version [11]).

[13] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies: Limitations and potentials. In *Proceedings of the Fourth International WWW conference*, 1995.

[14] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.

[15] Yehuda Afek and Yossi Matias. Elections in anonymous networks. *Information and Computation*, 113(2):312–330, 1994.

[16] Charu Aggarwal, Joel L. Wolf, and Philip S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, 1999.

[17] Ehud Aharoni and Reuven Cohen. Restricted dynamic steiner trees for scalable multicast in datagram networks. In *Proceedings of the Sixteenth Annual IEEE Conference on Computer Communications* [104], pages 876–883. (Journal version [18]).

[18] Ehud Aharoni and Reuven Cohen. Restricted dynamic steiner trees for scalable multicast in datagram networks. *IEEE/ACM Transactions on Networking*, 6(3):286–297, 1998. (Conference version [17]).

[19] Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. Page replacement for general caching problems. In *Proceedings of the Tenth Annual ACM/SIAM Symposium on Discrete Algorithms (SODA '99)*, pages 31–40, 1999.

[20] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing* [8], pages 258–267.

[21] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *Journal of Computer and System Sciences*, 70(2):145–175, 2005. (Conference version [20]).

[22] Andris Ambainis. Communication complexity in a 3-computer model. *Algorithmica*, 16(3):298–301, 1996.

[23] Andris Ambainis. A new protocol and lower bounds for quantum coin flipping. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing* [6], pages 134–142. (Journal version [24]).

[24] Andris Ambainis. A new protocol and lower bounds for quantum coin flipping. *Journal of Computer and System Sciences*, 68(2):398–416, 2004. (Conference version [23]).

[25] Andris Ambainis, Harry M. Buhrman, Yevgeniy Dodis, and Hein Röhrig. Multiparty quantum coin flipping. In *Proceedings of the Nineteenth Annual IEEE Conference on Computational Complexity*, pages 250–259, 2004.

[26] Andris Ambainis, Leonard J. Schulman, Amnon Ta-Shma, Umesh V. Vazirani, and Avi Wigderson. The quantum communication complexity of sampling. *SIAM Journal on Computing*, 32(6):1570–1585, 2003.

[27] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 82–93, 1980.

[28] László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity. In *Proceedings of the Twenty-Seventh Annual IEEE Symposium on Foundations of Computer Science*, pages 337–347, 1986.

[29] A. Ballardie. Core Based Trees (CBT) Multicast Routing Architecture. RFC 2201, IETF, September 1997.

[30] A. Ballardie. Core Based Trees (CBT version 2) Multicast Routing – Protocol Specification –. RFC 2189, IETF, September 1997.

[31] Tony Ballardie Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (CBT). In *Proceedings of the ACM SIGCOMM '93 Conference on Communications Architectures, Protocols and Applications*, pages 85–95, 1993.

[32] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 205–217, 2002.

[33] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

[34] Ziv Bar-Yossef, Thathachar S. Jayram, and Iordanis Kerenidis. Exponential separation of quantum and classical one-way communication complexity. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 128–137, 2004.

[35] Howard Barnum, Claude Crépeau, Daniel Gottesman, Adam D. Smith, and Alain Tapp. Authentication of quantum messages. In *Proceedings of the Forty-Third Annual IEEE Symposium on Foundations of Computer Science* [7], pages 449–458.

[36] Fred Bauer and Anujan Varma. ARIES: A rearrangeable inexpensive edge-based on-line steiner algorithm. In *Proceedings of the Fifteenth Annual IEEE Conference on Computer Communications* [103], pages 361–368. (Journal version [37]).

[37] Fred Bauer and Anujan Varma. ARIES: A rearrangeable inexpensive edge-based on-line steiner algorithm. *IEEE Journal on Selected Areas in Communications*, 15(3):382–397, 1997. (Conference version [36]).

[38] L. A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.

[39] L. A. Belady, R. A. Nelson, and G. S. Shedler. An anomaly in space-time characteristics of certain programs running in a paging machine. *Communications of the ACM*, 12(6):349–353, 1969.

[40] Shai Ben-David, Allan Borodin, Richard M. Karp, and Avi Wigderson Gábor Tardos. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994. (Conference version [41]).

[41] Shai Ben-David, Allan Borodin, Richard M. Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in online algorithms (extended abstract). In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 379–386, 1990. (Journal version [40]).

[42] Michael Ben-Or and Avinatan Hassidim. Fast quantum Byzantine agreement. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, pages 481–485, 2005.

[43] Charles H. Bennett. Quantum cryptography using any two nonorthogonal states. *Physical Review Letters*, 68(21):3121–3124, 1992.

[44] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, 1984.

[45] Charles H. Bennett and Stephen J. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Physical Review Letters*, 69(20):2881–2884, 1992.

[46] Rick Boivie, Nancy Feldman, Yuji Imai, Wim Livens, Dirk Ooms, and Olivier Paridaens. Explicit multicast (Xcast) basic specification. IETF Internet Draft, June 2004. work in progress.

[47] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[48] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference (preliminary version). In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 249–259, 1991. (Journal version [49]).

[49] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *Journal of Computer and System Sciences*, 50(2):244–258, 1995. (Conference version [48]).

[50] Ali Boudani and Bernard Cousin. SEM: A new small group multicast routing protocol. In *Proceedings of the Tenth IEEE International Conference on Telecommunications*, pages 450–455, 2003.

[51] Ali Boudani, Alexandre Guitton, and Bernard Cousin. GXcast: Generalized explicit multicast routing protocol. IETF Internet Draft, March 2004. work in progress.

[52] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series*, pages 53–74. 2002.

[53] Lee Breslau, Pei Cao, Graham Phillips Li Fan, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the Eighteenth Annual Conference on Computer Communications (INFOCOM '99)*, pages 126–134. IEEE Computer and Communications Societies, 1999.

[54] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[55] Harry M. Buhrman, Richard E. Cleve, Ronald de Wolf, and Christof Zalka. Bounds for small-error and zero-error quantum algorithms. In *Proceedings of the Fortyth Annual IEEE Symposium on Foundations of Computer Science*, pages 358–368, 1999.

[56] Harry M. Buhrman, Richard E. Cleve, and Willem van Dam. Quantum entanglement and communication complexity. *SIAM Journal on Computing*, 30(6):1829–1841, 2000.

[57] Harry M. Buhrman, Richard E. Cleve, John H. Watrous, and Ronald de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001.

[58] Harry M. Buhrman, Richard E. Cleve, and Avi Wigderson. Quantum vs. classical communication and computation. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 63–68, 1998.

[59] Harry M. Buhrman and Ronald de Wolf. Communication complexity lower bounds by polynomials. In *Proceedings of the Sixteenth Annual IEEE Conference on Computational Complexity*, pages 120–130, 2001.

[60] Harry M. Buhrman, Willem van Dam, Peter Høyer, and Alain Tapp. Multiparty quantum communication complexity. *Physical Review A*, 60(4):2737–2741, 1999.

[61] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication,*, pages 56–67, 1998.

[62] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, IETF, October 2002.

[63] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 193–206, 1997.

[64] Dong Pyo Chi and Jinsoo Kim. Quantum database search by a single query. In *Proceedings of the First NASA International Conference Quantum Computing and Quantum Communications* [5], pages 148–151.

[65] Marek Chrobak, Howard J. Karloff, Thomas. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[66] Marek Chrobak and John Noga. LRU is better than FIFO. *Algorithmica*, 23(2):180–185, 1999.

[67] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proceedings of International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS 2000)*, pages 1–12. ACM, 2000.

[68] Richard E. Cleve and Harry M. Buhrman. Substituting quantum entanglement for communication. *Physical Review A*, 56(2):1201–1204, 1997.

[69] Richard E. Cleve, Wim van Dam, Michael Nielsen, and Alain Tapp. Quantum entaglement and the communication complexity of the inner product function. In *Proceedings of the First NASA International Conference Quantum Computing and Quantum Communications* [5], pages 61–74.

[70] Digital Equipment Corporation. Dec web proxy traces. ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html.

[71] NTT Corporation. http://www.ntt.co.jp/news/news02/0211/021113.html (in Japanese).

[72] Lui's Henrique Maciel Kosmalski Costa, Serge Fdida, and Otto Carlos Muniz Bandeira Duarte. Hop by hop multicast routing protocol. In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 249–259, 2001.

[73] Claude Crépeau, Daniel Gottesman, and Adam D. Smith. Secure multiparty quantum computation. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing* [8], pages 643–652.

[74] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of www client-based traces. Technical Report BU-CS-95-010, Boston University, 1995.

[75] Ronald de Wolf. Characterization of non-deterministic quantum query and quantum communication complexity. In *Proceedings of the Fifteenth Annual IEEE Conference on Computational Complexity*, pages 271–278, 2000. (Journal version [76]).

[76] Ronald de Wolf. Nondeterministic quantum query and communication complexity. *SIAM Journal on Computing*, 32(3):681–699, 2003. (Conference version [75] and, partly, [102]).

[77] Stephen E. Deering. Multicast routing in internetworks and extended LANs. In *Proceedings of the ACM Symposium on Communications Architectures and Protocols (SIGCOMM '88)*, pages 55–64, 1988. (Journal version [78]).

[78] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, 1990. (Conference version [77]).

[79] Stephen E. Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. An architecture for wide-area multicast routing. In *Proceedings of the ACM SIGCOMM '94 Conference on Communications Architectures, Protocols and Applications*, pages 126–135, 1994. (Journal version [80]).

[80] Stephen E. Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, 1996. (Conference version [79]).

[81] Peter J. Denning. The working set model for program behaviour. *Communications of the ACM*, 11(5):323–333, 1968.

[82] Peter J. Denning. Working sets past and present. *IEEE Transactions on Software engineering*, 6(1):64–84, 1980.

[83] Christophe Diot, Walid Dabbous, and Jon Crowcroft. Multipoint communication: A survey of protocols, functions, and mechanisms. *IEEE Journal on Selected Areas in Communications*, 15(3):277–290, 1997.

[84] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, 2000.

[85] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete. *Physical Review*, 47:777–780, 1935.

[86] Artur K. Ekert. Quantum cryptography based on Bell's theorem. *Physical Review Letters*, 67(6):661–663, 1991.

[87] Hans Eriksson. MBONE: The Multicast Backbone. *Communications of ACM*, 37(8):54–60, 1994.

[88] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. RFC 2362, IETF, June 1998.

[89] W. Fenner. Internet Group Management Protocol, Version 2. RFC 2236, IETF, November 1997.

[90] Amos Fiat and Anna R. Karlin. Randomized and multipointer paging with locality of reference. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 626–634, 1995.

[91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[92] Amos Fiat and Ziv Rosen. Experimental studies of access graph based heuristics: Beating the LRU standard? In *Proceedings of the Eighth Annual ACM/SIAM Symposium on Discrete Algorithms (SODA '97)*, pages 63–72, 1997.

[93] Peter A. Franaszek and Terry J. Wagner. Some distribution-free aspects of paging algorithm performance. *Journal of the ACM*, 21(1):31–39, 1974.

[94] Parikshit Gopalan, Howard J. Karloff, Aranyak Mehta, Milena Mihail, and Nisheeth Vishnoi. Caching with expiration times. In *Proceedings of the Thirteenth Annual ACM/SIAM Symposium on Discrete Algorithms (SODA '02)*, pages 540–547, 2002.

[95] Daniel Gottesman, Hoi-Kwong Lo, Norbert Lütkenhaus, and John Preskill. Security of quantum key distribution with imperfect devices. *Quantum Information and Computation*, 4(5):325–360.

[96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* [4], pages 212–219.

[97] Anupam Gupta. Steiner points in tree metrics don't (really) help. In *Proceedings of the Twelfth Annual ACM/SIAM Symposium on Discrete Algorithms (SODA '01)*, pages 220–227, 2001.

[98] R. Hinden and Ed. Virtual Router Redundancy Protocol (VRRP). RFC 3768, IETF, April 2004.

[99] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard problems*. PWS Publishing Company, 1997.

[100] Hugh W. Holbrook and David R. Cheriton. IP Multicast Channels: EXPRESS support for large-scale single-source application. In *Proceedings of the ACM SIGCOMM '99 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 65–78, 1999.

[101] Alexander S. Holevo. Bounds for the quantity of information transmitted by a quantum communication channel. *Problemy Peredachi Informatsii*, 9(3):3–11, 1973. English Translation in *Problems of Information Transmission*, volume 9, pages 177–183, 1973.

[102] Peter Høyer and Ronald de Wolf. Improved quantum communication complexity bounds for disjointness and equality. In *Proceedings of the Nineteenth Symposium on Theoretical Aspects of Computer Science (STACS '02)*, volume 2285 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2002.

[103] IEEE Computer and Communications Societies. *Proceedings of the Fifteenth Annual IEEE Conference on Computer Communications*, 1996.

[104] IEEE Computer and Communications Societies. *Proceedings of the Sixteenth Annual IEEE Conference on Computer Communications*, 1997.

[105] Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

[106] Takeru Inoue. A study on recursive unicast approach for large scale multicasting — Flexcast (in Japanese). Technical Report NS2003-213, IEICE, 2003.

[107] Takeru Inoue, Seiichiro Tani, Katsuhiro Ishimaru, Shin'ichi Minato, and Toshiaki Miyazaki. Wide-area multicasting based on Flexcast: Toward the ubiquitous network. In *Proceedings of the Fifth Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT'03)*, pages 301–306. IEICE Communication Society, 2003.

[108] Takeru Inoue, Seiichiro Tani, Shin'ichi Minato, Hirokazu Takahashi, Satoshi Kotabe, and Toshiaki Miyazaki. An approach to inter-multicasting using Flexcast and report of inter-pacific experiments (in Japanese). Technical Report NS2003-37, IEICE, 2003.

[109] Takeru Inoue, Seiichiro Tani, Hirokazu Takahashi, Shin'ichi Minato, Toshiaki Miyazaki, and Kan Toyoshima. Design and implementation of multicast routers based on cluster computing. In *Proceedings of International Conference on Parallel and Distributed Systems (ICPADS'05)*, pages 328–334. IEEE, 2005.

[110] Internet2. http://www.internet2.org/.

[111] Sandy Irani. Page replacement with multi-size pages and applications to web caching. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 701–710, 1997. (Journal version [112]).

[112] Sandy Irani. Page replacement with multi-size pages and applications to web caching. *Algorithmica*, 33(3):384–409, 2002. (Conference version [111]).

[113] Sandy Irani. Randomized weighted caching with two page weights. *Algorithmica*, 32(4):624–640, 2002.

[114] Sandy Irani, Anna Karlin, and Steven Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, 25(3):477–497, 1996. (Conference version [115]).

[115] Sandy Irani, Anna R. Karlin, and Steven Phillips. Strongly competitive algorithms for paging with locality of reference. In *Proceedings of the Third Annual ACM/SIAM Symposium on Discrete Algorithms*, pages 228–236, 1992. (Journal version [114]).

[116] Katsuhiro Ishimaru, Takao Ogura, Takeru Inoue, Shin'ichi Minato, Toshiaki Miyazaki, and Tomonori Aoyama. Development of simultaneous wide-area multipoint transmission platform of HD streams over JGN (in Japanese). Technical Report IN2003-21, IEICE, 2003.

[117] Alon Itai and Michael Rodeh. Symmetry breaking in distributive networks. In *Proceedings of the Twenty-Second Annual IEEE Symposium on Foundations of Computer Science*, pages 150–158, 1981. (Journal version [118]).

[118] Alon Itai and Michael Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990. (Conference version [117]).

[119] Rahul Jain, Jaikumar Radhakrishnan, and Pranab Sen. Privacy and interaction in quantum communication complexity and a theorem about the relative entropy of quantum states. In *Proceedings of the Forty-Third Annual IEEE Symposium on Foundations of Computer Science* [7], pages 429–438.

[120] Rahul Jain, Jaikumar Radhakrishnan, and Pranab Sen. A lower bound for the bounded round quantum communication complexity of set disjointness. In *Proceedings of the Forty-Fourth Annual IEEE Symposium on Foundations of Computer Science* [9], pages 220–229.

[121] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James O'Toole Jr. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI 2000)*, pages 197–212. USENIX Association, 2000.

[122] Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

[123] Anna R. Karlin, Larry Rudolph Mark S. Manasse, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[124] Anna R. Karlin, Steven J. Phillips, and Prabhakar Raghavan. Markov paging (extended abstract). In *Proceedings of the Thirty-Third Annual IEEE Symposium on Foundations of Computer Science*, pages 208–217, 1992. (Journal version [125]).

[125] Anna R. Karlin, Steven J. Phillips, and Prabhakar Raghavan. Markov paging. *SIAM Journal on Computing*, 30(3):906–922, 2000. (Conference version [124]).

[126] Alexei Yu. Kitaev, Alexander H. Shen, and Mikhail N. Vyalyi. *Classical and Quantum Computation*, volume 47 of *Graduate Studies in Mathematics*. AMS, 2002.

[127] Hartmut Klauck. On quantum and probabilistic communication: Las Vegas and one-way protocols. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 644–651, 2000.

[128] Hartmut Klauck, Ashwin Nayak, Amnon Ta-Shma, and David Zuckerman. Interaction in quantum communication and the complexity of set disjointness. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing* [6], pages 124–133.

[129] Alex Koifman and Stephen Zabele. RAMP: A reliable adaptive multicast protocol. In *Proceedings of the Fifteenth Annual IEEE Conference on Computer Communications* [103], pages 1442–1451.

[130] Elias Koutsoupias and Christos H. Papadimitriou. Beyond competitive analysis. In *Proceedings of the Thirty-Fifth Annual IEEE Symposium on Foundations of Computer Science* [3], pages 394–400.

[131] Evangelos Kranakis, Danny Krizanc, and Jacob van den Berg. *Information and Computation*, 114(2):214–236, 1994.

[132] Ilan Kremer. Quantum communication. Master's thesis, Computer Science Department, The Hebrew University, 1995.

[133] Pieter Liefooghe and Marnix Goossens. An architecture for seamless access to multicast content. In *Proceedings of the Twenty-Fifth Conference on Local Computer Networks*, pages 488–494. IEEE Computer Society, 2000.

[134] John C. Lin and Sanjoy Paul. RMTP: A reliable multicast transport protocol. In *Proceedings of the Fifteenth Annual IEEE Conference on Computer Communications* [103]. (Journal version [158]).

[135] Hoi-Kwong Lo and Hoi Fung Chau. Unconditional security of quantum key distribution over arbitrarily long distances. *Science*, 283:2050–2056, 1999.

[136] Carsten Lund, Steven Phillips, and Nick Reingold. IP over connection-oriented networks and distributional paging. In *Proceedings of the Thirty-Fifth Annual IEEE Symposium on Foundations of Computer Science* [3], pages 424–434. (Journal version [137]).

[137] Carsten Lund, Steven Phillips, and Nick Reingold. Paging against a distributional and IP networking. *Journal of Computer and System Sciences*, 58:222–231, 1999. (Conference version [136]).

[138] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufman Publishers, 1996.

[139] Yossi Matias, Jeffrey Scott Vitter, and Wen-Chun Ni. Dynamic generation of discrete random variates. In *Proceedings of the Fourth Annual ACM/SIAM Symposium on Discrete Algorithms (SODA '93)*, pages 361–370, 1993. (Journal version [140]).

[140] Yossi Matias, Jeffrey Scott Vitter, and Wen-Chun Ni. Dynamic generation of discrete random variates. *Theory of Computing Systems*, 36(4):329–358, 2003. (Conference version [139]).

[141] Dominic Mayers. Unconditional security in quantum cryptography. *Journal of the ACM*, 48(3):351–406, 2001.

[142] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[143] Kurt Mehlhorn and Erik Meineche Schmidt. Las Vegas is better than determinism in VLSI and distributed computing. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 330–337, 1982.

[144] Michael R. Garey and David S. Johnson. *COMPUTERS AND INTRACTABILITY — A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 2 edition, 1979.

[145] Philip Miller. *TCP/IP Explained*. Digital Press, 1997.

[146] George Milne and Robin Milner. Concurrent processes and their syntax. *Journal of the ACM*, 26(2):302–321, 1979.

[147] Shin'ichi Minato, Hirokazu Takahashi, Takeru Inoue, Hiroshi Tohjo, and Kan Toyoshima. A framework of programmable multicast applications based on Flexcast and JavaApplet (in Japanese). Technical Report IN2005-49, IEICE, 2005.

[148] Carlos Mochon. Quantum weak coin-flipping with bias of 0.192. In *Proceedings of the Forty-Fifth Annual IEEE Symposium on Foundations of Computer Science*, pages 2–11, 2004.

[149] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[150] J. Moy. Multicast Extensions to OSPF. RFC 1584, IETF, March 1994.

[151] Ilan Newman. Private vs. common random bits in communication complexity. *Information Processing Letters*, 39(2):67–71, 1991.

[152] Ilan Newman and Mario Szegedy. Public vs. private coin flips in one round communication games (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* [4], pages 561–570.

[153] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[154] Jörg Nonnenmacher, Ernst Biersack, and Don Towsley. Parity-based loss recovery for reliable multicast transmission. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 289–300, 1997.

[155] Nancy Norris. Universal covers of graphs: Isomoriphism to depth n-1 implies isomoriphism to all depths. *Discrete Applied Mathematics*, 56(1):61–74, 1995.

[156] National Lab of Applied Network Research. Sanitized access log. ftp://ircache.nlanr.net/Traces/.

[157] Sanjoy Paul. *MULTICASTING ON THE INTERNET AND ITS APPLICATIONS*. Kluwer Academic Publishers, 1998.

[158] Sanjoy Paul, Krishan K. Sabnani, John C.-H Lin, and Supratik Bhattacharyya. Reliable multicast transport protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407–421, 1997. (Conference version [134]).

[159] C. Perkins. IP Mobility Support for IPv4. RFC 3344, IETF, August 2002.

[160] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys*, 35(4):374–398, 2003.

[161] J. Postel. User Datagram Protocol. RFC 0768, IETF, August 1980.

[162] J. Postel. Internet Control Message Protocol. RFC 0792, IETF, September 1981.

[163] J. Postel. Internet Protocol. RFC 0791, IETF, September 1981.

[164] J. Postel. Transmission Control Protocol. RFC 0793, IETF, September 1981.

[165] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages and Programming(ICALP'89)*, volume 372 of *Lecture Notes in Computer Science*, pages 687–703. Springer, 1989. Revised version available as IBM Journal of Research and Development, 38(6):683-708, 1994.

[166] Ran Raz. Exponential separation of quantum and classical communication complexity. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 358–367, 1999.

[167] Alexander A. Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya Mathematics*, 67(1):145–159, 2003.

[168] Laxman H. Sahasrabuddhe and Biswanath Mukherjee. Multicast routing algorithms and protocols: A tutorial. *IEEE Network*, 14(1):90, 2000.

[169] Baruch Schieber and Marc Snir. Calling names on nameless networks. *Information and Computation*, 113(1):80–101, 1994. (Conference version [170]).

[170] Baruch Schieber and Marc Snir. Calling names on nameless networks. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing (PODC '89)*, pages 80–101. ACM, 1994. (Journal version [169]).

[171] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, IETF, July 2003.

[172] Clay Shields and J. J. Garcia-Luna-Aceves. The ordered core based tree protocol. In *Proceedings of the Sixteenth Annual IEEE Conference on Computer Communications* [104], pages 884–891.

[173] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the Thirty-Fifth Annual IEEE Symposium on Foundations of Computer Science* [3], pages 124–134. (Journal version [174]).

[174] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. (Conference version [173]).

[175] Peter W. Shor and John Preskill. Simple proof of security of the BB84 quantum key distribution protocol. *Physical Review Letters*, 85(2):441–444, 2000.

[176] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[177] Ion Stoica, T. S. Eugene Ng, and Hui Zhang. REUNITE: A recursive unicast approach to multicast. In *Proceedings of the Nineteenth Conference on Computer Communications (INFOCOM'00)*, pages 1644–1653. IEEE Computer and Communication Societies, 2000.

[178] Hirokazu Takahashi, Takeru Inoue, Hiroshi Tohjo, Kan Toyoshima, and Shin'ichi Minato. A study on IP distance education system using Flexcast (in Japanese). Technical Report IN2005-50.

[179] Kiyoshi Tamaki, Masato Koashi, and Nobuyuki Imoto. Security of the Bennett 1992 quantum-key distribution protocol against individual attack over a realistic channel. *Physical Review A*, 67(3):032310, 2003.

[180] Kiyoshi Tamaki and Norbert Lütkenhaus. Unconditional security of the Bennett 1992 quantum key-distribution protocol over a lossy and noisy channel. *Physical Review A*, 69(3):032316, 2004.

[181] Seiichiro Tani, Takeru Inoue, Hirokazu Takahashi Shin'ichi Minato, Satoru Kotabe, and Toshiaki Miyazaki. Global multi-point streaming experiments based on the Flexcast protocol. *NTT Technical Review*, 1(5):24–30, 2003.

[182] Seiichiro Tani, Hirotada Kobayashi, and Keiji Matsumoto. Quantum leader election via exact amplitude amplification. In *Proceedings of ERATO conference on Quantum Information Science*, pages 11–12, 2005.

[183] Seiichiro Tani, Toshiaki Miyazaki, and Noriyuki Takahashi. Adaptive stream multicast based on IP unicast and dynamic commercial attachment mechanism: An active network implementation. In *Proceedings of IFIP-TC6 Third International Working Conference on Active Networks (IWAN '01)*, volume 2207 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2001.

[184] Seiichiro Tani, Toshiaki Miyazaki, Noriyuki Takahashi, and Shin'ichi Minato. Flexcast: Self-organizing multicast technology (in Japanese). *NTT R&D*, 52(3):213–222, 2003.

[185] Seiichiro Tani, Masaki Nakanishi, and Shigeru Yamashita. A quantum protocol for the list-nonequality function. In *Proceedings of the Eleventh Quantum Information Technology Symposium (QIT11)*, number QIT2004-51, pages 21–25. IEICE.

[186] Olivier Temam. An algorithm for optimally exploiting spatial and temporal locality in upper memory levels. *IEEE Transactions on Computers*, 48(2):150–158, 1999.

[187] Ajit S. Thyagarajan and Stephen E. Deering. Hierarchical distance-vector multicast routing for the MBone. In *Proceedings of the ACM SIGCOMM 1995 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 60–66, 1995.

[188] R. Vida and L. Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810, IETF, June 2004.

[189] D. Waitzman, C. Partridge, and S.E. Deering. Distance Vector Multicast Routing Protocol. RFC 1075, IETF, November 1988.

[190] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[191] Su Wen, Jim Griffioen, and Kenneth L. Calvert. Building multicast services from unicast forwarding and ephemeral state. In *Proceedings of the Fourth*

International Conference on Open Architectures and Network Programming (OPENARCH '01). IEEE Communication Society, 2001. (Journal version [192]).

[192] Su Wen, Jim Griffioen, and Kenneth L. Calvert. Building multicast services from unicast forwarding and ephemeral state. *Computer Networks*, 38(3):327–345, 2002. (Conference version [191]).

[193] Chak-Kuen Wong and Malcolm C. Easton. An efficient method for weighted sampling without replacement. *SIAM Journal on Computing*, 9(1):111–113, 1980.

[194] Masafumi Yamashita and Tsunehiko Kameda. Computing on an anonymous network. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing (PODC '88)*, pages 117–130, 1988. (Journal version [195]).

[195] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks: Part I – characterizing the solvable cases. *IEEE Transactions on Parallel Distributed Systems*, 7(1):69–89, 1996. (Conference version [194]).

[196] Masafumi Yamashita and Tsunehiko Kameda. Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Transactions on Parallel and Distributed Systems*, 10(9):878–887, 1999.

[197] Andrew Chi-Chih Yao. Some complexity questions related to distributed computing. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.

[198] Andrew Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of the Thirty-Fourth Annual IEEE Symposium on Foundations of Computer Science*, pages 352–361, 1993.

[199] Andrew Chi-Chih Yao. On the power of quantum fingerprinting. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 77–81, 2003.

[200] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.

[201] Neal E. Young. On-line file caching. In *Proceedings of the Ninth Annual ACM/SIAM Symposium on Discrete Algorithms (SODA '98)*, pages 82–86, 1998. (Journal version [202]).

[202] Neal E. Young. On-line file caching. *Algorithmica*, 33:371–383, 2002. (Conference version [201]).

List of Publications by the Author

Journal

1. Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. The complexity of the optimal variable ordering problems of a shared binary decision diagram. *IEICE Transactions on Information and Systems*, E79-D(4):271–281, 1996.
2. Seiichiro Tani, Mitsuo Teramoto, Tomoo Fukazawa, and Kazuyoshi Matsuhiro. Efficient path selection for delay testing based on path clustering. *Journal of Electronic Testing (JETTA)*, 15(1/2):75–85, 1999.
3. Seiichiro Tani and Toshiaki Miyazaki. A design framework for online algorithms solving the object replacement problem. *IEICE Transactions on Information and Systems*, E84-D(9):1135–1143, 2001.
4. Shin'ya Ishihara, Toshiaki Miyazaki, Atsushi Takahara, and Seiichiro Tani. An approach to adaptive network. *IEICE Transactions on Information and Systems*, E85-D(5):839–846, 2002.
5. Seiichiro Tani and Toshiaki Miyazaki. A randomized online algorithm for the file caching problem. *IEICE Transactions on Information and Systems*, E86-D(4):686–697, 2003.
6. Takeru Inoue, Seiichiro Tani, Hirokazu Takahashi, Shin'ichi Minato, Toshiaki Minato, and Kan Toyoshima. Design and implementation of the incrementally deployable multicast system based on Flexcast (in Japanese). *IEICE Transactions on Information and Systems*, J88-D1(2):272–291, 2004.

Conference

1. Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *Proceedings of the Fourth International Symposium on Algorithms and Computation (ISAAC'93)*, volume 762 of *Lecture Notes in Computer Science*, pages 389–398. Springer-Verlag, 1993.
2. Seiichiro Tani and Hiroshi Imai. A reordering operation for an ordered binary decision diagram and an extended framework for combinatorics of graphs. In *Proceedings of the Fifth International Symposium on Algorithms and Computation (ISAAC'94)*, volume 834 of *Lecture Notes in Computer Science*, pages 575–583. Springer-Verlag, 1994.
3. Kazuyoshi Hayase, Kunihiko Sadakane, and Seiichiro Tani. Output-size sensitiveness of OBDD construction through maximal independent set problem. In *Proceedings of the First Annual International Conference on Computing and Combinatorics, (COCOON '95)*, volume 959 of *Lecture Notes in Computer Science*, pages 229–234. Springer, 1995.
4. Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computing the Tutte polynomial of a graph of moderate size. In *Proceedings of the Sixth International Symposium on Algorithms and Computation (ISAAC '95)*, volume 1004 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 1995.
5. Seiichiro Tani, Mitsuo Teramoto, Tomoo Fukazawa, and Kazuyoshi Matsuhiro. Efficient path selection for delay testing based on partial path evaluation. In *Proceedings of the Sixteenth IEEE VLSI Test Symposium (VTS '98)*, pages 188–193, 1998.
6. Atsushi Takahara, Seiichiro Tani, Shin'ya Ishihara, Toshiaki Miyazaki, Mitsuo Teramoto, and Tomoo Fukazawa. Virtual BUS: An easy-to-use environment for distributed resources. In *Proceedings of the Twenty-Fourth Conference on Local Computer Networks*, pages 62–70. IEEE Computer Society, 1999.
7. Shin'ya Ishihara, Seiichiro Tani, and Atsushi Takahara. Virtual BUS: A simple implementation of an effortless networking system based on PVM. In *Proceedings of the Sixth European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (PVM/MPI '99)*, volume 1697 of *Lecture Notes in Computer Science*, pages 461–468. Springer-Verlag, 1999.
8. Toshiaki Miyazaki, Atsushi Takahara, Shin'ya Ishihara, Seiichiro Tani, Takahiro Murooka, Tomoo Fukazawa, Mitsuo Teramoto, and Kazuyoshi

Matsuhiro. Virtual Bus: A network technology for setting up distributed resources in your own computer. In *Proceedings of the Fourteenth International Parallel & Distributed Processing Symposium (IPDPS'00)*, pages 535–540. IEEE Computer Society, 2000.

9. Seiichiro Tani, Toshiaki Miyazaki, and Noriyuki Takahashi. Adaptive stream multicast based on IP unicast and dynamic commercial attachment mechanism: An active network implementation. In *Proceedings of IFIP-TC6 Third International Working Conference on Active Networks (IWAN '01)*, volume 2207 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2001.
10. Satoru Ohta, Seiichiro Tani, and Toshiaki Miyazaki. Multicast as a traffic variance smoother for IP streaming service. In *Proceedings of Networks 2002*, pages 105–110, 2002.
11. Takeru Inoue, Seiichiro Tani, Katsuhiro Ishimaru, Shin'ichi Minato, and Toshiaki Miyazaki. Wide-area multicasting based on Flexcast: Toward the ubiquitous network. In *Proceedings of the Fifth Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT'03)*, pages 301–306. IEICE Communication Society, 2003.
12. Takeru Inoue, Seiichiro Tani, Hirokazu Takahashi, Shin'ichi Minato, Toshiaki Miyazaki, and Kan Toyoshima. Design and implementation of multicast routers based on cluster computing. In *Proceedings of International Conference on Parallel and Distributed Systems (ICPADS'05)*, pages 328–334. IEEE, 2005.
13. Seiichiro Tani, Hirotada Kobayashi, and Keiji Matsumoto. Exact quantum algorithms for the leader election problem. In *Proceedings of the Twenty-Second Symposium on Theoretical Aspects of Computer Science (STACS '05)*, volume 3404 of *Lecture Notes in Computer Science*, pages 581–592. Springer, 2005.
14. Seiichiro Tani, Hirotada Kobayashi, and Keiji Matsumoto. Quantum leader election via exact amplitude amplification. In *Proceedings of ERATO conference on Quantum Information Science*, pages 11–12, 2005.
15. Seiichiro Tani. An application of entanglement to leader election in anonymous networks (invited paper). In *Proceedings of COE-Kakenhi Workshop on Quantum Information Theory and Quantum Statistical Inference*, pages 59–62, 2005.
16. Seiichiro Tani, Masaki Nakanishi, and Shigeru Yamashita. Quantum communication complexity for the distinctness function on a ring. In *Proceedings of Workshop on Theory of Quantum Computation, Communication, and Cryptograpy*, 2006.

Technical Reports

1. Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams (in Japanese). Technical Report COMP92-100, IEICE, March 1993.

2. Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. KUIS Technical Report KUIS-93-0009, Kyoto University, July 1993.

3. Seiichiro Tani, Kiyoharu Hamaguchi, and Shuzo Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. Technical Report TR93-6, University of Tokyo, December 1993.

4. Hiroshi Imai and Seiichiro Tani. Ordered binary decision diagrams, gaussian elimination, and graph theory. IPSJ SIG Notes 94-AL-41, IPSJ, 1994.

5. Hiroshi Imai, Seiichiro Tani, and Kyoko Sekine. Ordered binary decision diagrams, graph theory and computational geometry. Technical Report TR95-05, University of Tokyo, Tokyo, July 1995.

6. Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computation of the Tutte polynomial and BDD. Technical Report 1995-07, IEICE, 1995.

7. Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computing the Tutte polynomial of a graph and the Jones polynomial of an alternating link of moderate size. Technical Report TR95-06, University of Tokyo, Tokyo, July 1995.

8. Seiichiro Tani, Mitsuo Teramoto, Tomoo Fukazawa, and Kazuyoshi Matsuhiro. Path selection based on real delay estimation
for path delay fault testing. Technical Report VLD97-83, IEICE, 1997.

9. Seiichiro Tani, Shin'ya Ishihara, and Atsushi Takahara. A resource-organizing method in networks (in Japanese). Technical Report IN99–122, IEICE, 2000.

10. Seiichiro Tani. A competitive algorithm for network caching and its design methodology (in Japanese). Technical Report COMP2000-10, IEICE, 2000.

11. Toshiaki Miyazaki, Seiichiro Tani, and Noriyuki Takahashi. A stream-data multicast protocol using IP unicast address (in Japanese). Technical Report IN2001-9, IEICE, 2001.

12. Satoru Ohta, Seiichiro Tani, and Toshiaki Miyazaki. Traffic smoothing effect by an adaptive multicast technique (in Japanese). Technical Report IN2002-19, IEICE, 2002.

13. Seiichiro Tani, Toshiaki Miyazaki, Noriyuki Takahashi, and Shin ichi Minato. Flexcast: Self-organizing multicast technology (in Japanese). *NTT R&D*, 52(3):213–222, 2003.

14. Takeru Inoue, Seiichiro Tani, Shin'ichi Minato, Hirokazu Takahashi, Satoshi Kotabe, and Toshiaki Miyazaki. An approach to inter-multicasting using Flexcast and report of inter-pacific experiments (in Japanese). Technical Report NS2003-37, IEICE, 2003.

15. Seiichiro Tani, Takeru Inoue, Hirokazu Takahashi Shin'ichi Minato, Satoru Kotabe, and Toshiaki Miyazaki. Global multi-point streaming experiments based on the Flexcast protocol. *NTT Technical Review*, 1(5):24–30, 2003.

16. Seiichiro Tani and Yasuhito Kawano. A quantum distributed algorithm for the leader election problem with prior shared cat-states. Technical Report QIT2003-60, IEICE, 2003.

17. Seiichiro Tani, Masaki Nakanishi, and Shigeru Yamashita. A quantum protocol for the list-nonequality function. Technical Report QIT2004-51, IEICE, 2005.

18. Takeru Inoue, Seiichiro Tani, Hirokazu Takahashi, Shin-ichi Minato, Toshiaki Miyazaki, and Kan Toyoshima. On design and development of multicast system based on Flexcast (in Japanese). Technical Report NS2004-50, IEICE, 2004.