アドホック無線ネットワークにおける通信アルゴリズムと ネットワークアーキテクチャに関するグラフ理論的研究

Graph Theoretic Study on Communication Algorithms and Network Architectures in Ad Hoc Radio Networks

A Dissertation
Submitted to the Department of Computer Science and Engineering,
Graduate School of Engineering, Nagoya Institute of Technology
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Engineering

Jiro UCHIDA

2008

Preface

In this dissertation, radio broadcasting and cluster-based architecture for ad hoc radio networks are studied.

A radio network is a collection of transmitter-receiver devices (referred to as nodes), and can be represented by a directional graph $G = (V, E)$ in which V denotes a set of nodes and when node v is in the transmission range of node u, there exists a directed edge $(u, v) \in E$. In our model each node performs transmission or reception at one round. Each node does not have a collision detection capability and knows only own ID.

Acknowledged Radio Broadcasting(ARB) means transmitting a message from one special node called source to all other nodes and informing the source about its completion. It is known that no ARB algorithm exists in the model without collision detection. In this dissertation, we show that if $n \geq 2$, where n is the number of nodes in the network, we can construct ARB algorithms in $O(n)$ rounds for bidirectional graphs and in $O(n^{4/3} \log^{10/3} n)$ rounds for strongly connected graphs and construct Acknowledged Radio Gossiping(ARG) algorithms in $O(n \log^3 n)$ rounds for bidirectional graphs and in $O(n^{4/3} \log^{10/3} n)$ rounds for strongly connected graphs without collision detection.

About the lower bounds of deterministic radio broadcasting, the lower bound of $\Omega(n)$ for bidirectional graphs when each node knows only own ID. In order to perform a fast radio broadcasting, we propose a dynamic cluster-based architecture with two atomic operations *join* and *leave*. Before joining to the network, each node knows only self ID and even the number of nodes n in the network is unknown. During join and leave operations each node maintains the neighbors' information. Clustering breaks the network into physical proximity clusters which are simpler to manage by the nodes called as *cluster head*. The subsequent *backbone* construction formed from cluster heads provides the communication between the clusters. Furthermore, the backbone and the clusters are combined into a spanning tree of G which we call *cluster-based network* of G, denoted as $CNet(G)$. Let $|BT(G)|$ be the number of nodes of backbone in our clustering. A $CNet(G)$ has several novel properties: (1) the backbone consists of at most $2p_G - 1$ nodes, where p_G is the cardinality of the minimum clique partition of G, and so when G is a dense graph, $p_G \ll n$; (2) a broadcasting on G can be executed via the backbone in $O(|BT(G)|)$ time (it needs $\Omega(n)$ time on a flat network [2]); and (3) if G is a unit disk graph, $|BT(G)| \leq 10 \times |MDS|$.

Acknowledgements

The author would like to express his sincere thanks to supervisor Professor Koichi Wada. He has given innumerable advice through the whole work.

The author is also deeply indebted the other members of the doctoral committee. The author would like to note in particular Professor Naohisa Takahashi and Associate Professor Nobuhiro Inuzuka.

The author is also deeply indebted Professor Wei Chen and Associate Professor Yoshiaki Katayama.

Along the way the author had substantial help from colleagues. The author would like to thank to Mr. Naoki Inaba who supported me and achieved the results contributing for this dissertation.

Contents

Chapter 1

Introduction

1.1 Research Backgrounds

1.1.1 About Radio Networks

Recently, a wireless data communication is very active because of the spread of cellular phones and an increase in Internet access from mobile terminals such as PDA and laptop. Most of a existing wireless communication assumes the network infrastructure. It might be difficult to establish a wireless network with the infrastructure by the reason such as the high installation cost and geographical limitations.

There is increasing interest in multi-hop wireless ad-hoc networks composed of a large number of autonomous nodes communicating via radio without any additional infrastructure. They are constrained as for the critical resources in some cases. A typical example is given by wireless sensor networks, where sensor nodes equipped with transmitter-receiver devices are usually irreplaceable, and become unusable after energy depletion or other failures.

Many applications for ad-hoc networks are considered. At first, it is considered for the situation in which infrastructure is not available, such as in the war, disaster and so on. In the situation in which the infrastructure is destroyed or cannot be provided, the network can be constructed only with bringing the mobile node into there. Now its potential value is evaluated, and applications except for military affairs are investigated all over the world. For example, ecology observation of the seabird which can live only at a specific place.

In ad-hoc networks, after the nodes of an ad-hoc network are deployed physically, a *flat* network topology is formed in which a link exists between two nodes if they are in each others communication range.

In one of the traditional network models, an ad-hoc radio network is a collection of transmitter-receiver devices (referred to as nodes), and can be represented by a directional graph $G = (V, E)$ in which V denotes a set of nodes and when node v is in the transmission range of node u, there exists a directed edge $(u, v) \in E$. Each node takes a synchronization per round and performs transmission or reception at one round. In such an above network model, there are many researches about the completion times of communication tasks and its solvabilities theoretically.

Communication such as broadcast, multicast, and data gathering are required on ad-hoc networks. One of the fundamental tasks in network communication is radio broadcasting (RB). Its goal is to transmit a message from one node of the network, called the source, to all other nodes. The message which is disseminated is called source message. Remote nodes get the source message via intermediate nodes, along directed paths in the network. In an acknowledged radio broadcasting (ARB) the goal is not only to achieve RB but also to inform the source about the completion of RB. This may be essential, e.g., when the source has several messages to disseminate, none of the nodes should receive the next message until all nodes get the previous one [7]. Another task is radio gossiping (RG) which broadcasts the message of each node to all other nodes. We also consider the task acknowledged radio gossiping (ARG) which achieves RG and informs every node about the completion of RG.

In such a flat network topology after the nodes are deployed physically, there is no established structure to facilitate efficient communication. Since an ad-hoc network changes its geographical topology easily when physical conditions change, we need network reconfiguration operations such as *nodes getting out* of and *nodes joining into* an existing network.

Clustering is seen as the step to provide the flat ad-hoc network with a *hierarchical* architecture which minimizes communication overhead and accommodates network organization. The basic idea is that of breaking the network into physical proximity clusters which are simpler to manage by the nodes called as *cluster head*. The subsequent *backbone* construction formed from cluster heads provides the communication between the clusters. Clustering and the subsequent hierarchical architecture of a flat ad-hoc network have been investigated in many literature, where many proposed approaches select cluster heads through finding small dominating set or a large independent set of the ad-hoc network [3,6,10,16,21,22]. Although communication tasks can be done more efficient on cluster-based network, extra network maintenance expense is needed. A good network architecture should be not only communication efficient but also reconfiguration efficient.

1.1.2 Related Researches

About (Acknowledged) Radio Broadcasting

In the network model in Subsection 1.1.1, communication tasks and its completion times have been investigated in many literatures. The network model is theoretical, and so are the results.

The standard collision-free communication procedure for ad-hoc radio networks is called *Round Robin* [11]. Round Robin contains n rounds. In the i-th round the node with identifier i transmits its whole knowledge to all its out-neighbors. In every round at most one node acts as a transmitter, hence collisions are avoided. Round Robin is used as a subroutine in many RB and RG algorithms. An RG completes in $O(n^2)$ rounds, where n is the number of nodes.

There are two situations for communication procedures in radio networks: one is that nodes have full knowledge of the network (such as the topology of the network, the number of the nodes in the network, IDs of the neighbors etc.), the other is that nodes are ignorant of the network information. Various algorithms are studied in radio networks, e.g. the centralized algorithms with the mechanism in which all nodes are concentrated and managed, and the distributed algorithms without such a mechanism; the deterministic algorithms whose process become settled uniquely, and randomized algorithms which are not so [1,5,7–9,11,12,14,15,17,20].

Under the assumption that the nodes have full knowledge of the network, in [1] the authors proved the existence of a family of n-node networks of radius 2, for which any broadcast requires $\Omega(\log^2 n)$ time, while in [15] an optimal deterministic algorithm which produces a broadcast scheme of time $O(D + \log^2 n)$ is given, for any n-node network of diameter D.

Hereafter, we assume that the nodes have neither the knowledge of the network nor the knowledge of their neighborhood.

For randomized algorithms, the lower bound of $\Omega(D \cdot \log(n/D))$ for bidirectional graphs is shown by Kushilevitz and Mansour [17], and the lower bound of $\Omega(\log^2 n)$ for constant diameter networks is obtained by Alon *et al.* [1].

For deterministic distributed algorithms, on the model without collision detection, Chlebus *et al.* have presented an optimal linear-time broadcasting protocol for bidirectional ad-hoc radio networks [7]. Also, on the model with collision detection, they presented an $O(r \cdot ecc)$-time RB algorithm for arbitrary graphs, an $O(n)$-time ARB algorithm for bidirectional graphs, and an $O(n \cdot ecc)$-time ARB algorithm for strongly connected graphs, where *ecc* is the maximum distance from the source. It is not easier to solve a problem for arbitrary directional graphs than for bidirectional ones. Note that on the model without collision detection there does not exist any algorithm for ARB, even for bidirectional graphs [7]. The best $O(n^{4/3} \log^{10/3} n)$ time gossiping algorithm for strongly connected graphs is shown in [12].

About the lower bounds of deterministic RB, the lower bound of $\Omega(n)$ for bidirectional graphs [14] and the lower bound of $\Omega(n \log n)$ for arbitrary graphs [5] are shown.

Table 1.1 shows the results we discussed above. All these results are obtained from deterministic algorithms under the same radio network model.

About Clustering

Distributed clustering for a flat ad-hoc network topology G has been investigated in many literatures. Most of the proposed protocols end up generating cluster heads and forming a corresponding backbone. Usually, cluster heads form a *dominating set (DS)* or an *independent set (IS)* of G [3, 10, 16, 21, 22]. A set is a *DS* of G if any node of G is either a node of *DS* or is neighbor of a node of *DS*. A set is an *IS* of G if no two nodes of

Problem	Collision detection	Graphs	Computation time
RB	without	bidirectional	$O(n)$ [7]
			$\Omega(n)$ [14]
		arbitrary	$O(n \log^2 ecc)$ [9]
			$\Omega(n \log n)$ [5]
	with	bidirectional	$O(r + ecc)$ [20]
		strongly connected	$O(n \log^2 ecc)$ [9]
		arbitrary	$O(r \cdot ecc)$ [7]
RG	without	strongly connected	$O(n^{\frac{4}{3}} \log^{\frac{10}{3}} n)$ [12]
ARB	without	bidirectional	algorithm does not exist [7]
		bidirectional ($n \geq 2$)	$O(n)^*$
		strongly connected ($n \geq 2$)	$O(n^{\frac{4}{3}} \log^{\frac{10}{3}} n)^*$
	with	bidirectional	$O(n)$ [7]
			$O(r + ecc)$ [20]
		strongly connected	$O(n \cdot ecc)$ [7]
ARG	without	bidirectional ($n \geq 2$)	$O(n \log^3 n)^*$
		strongly connected ($n \geq 2$)	$O(n^{\frac{4}{3}} \log^{\frac{10}{3}} n)^*$

(n:number of nodes, ecc:largest distance from the source, r:length of the source message, *:our result)

Table 1.1: Previous results and ours* (Deterministic and Distributed)

the set are the neighbors in G. In most of these distributed algorithms, the nodes need two hops knowledge, i.e., the knowledge of the neighbors and the neighbors' neighbors which need $\Omega(n)$ time to get. It is known that finding a minimum *DS* (*MDS*) of G is an *NP*-complete problem. Therefore, finding a clustering with the minimum number of clusters is also an *NP*-complete problem. Sometimes, an ad-hoc network can be modeled by a *unit disk graph*, where an edge exists between two nodes iff the Euclidean distance of two nodes is at most one. A cluster structure can be formed by selecting a *DS* or an *IS* and then connect them to a backbone in $O(n)$ - $O(n^2)$ time depending on if the construction of clusters and a backbone is explicitly specified [3, 21, 22]. In [16], a randomized algorithm is presented to compute an asymptotically optimal *MDS* (i.e., it finds a set of cluster heads but not form a cluster structure) in polylogarithmic time. The comparative performance evaluation of these above algorithms for clustering and backbone formation is shown in [4].

1.2 Contributions

In this dissertation, we consider the ARB, the ARG, and the clustering algorithms on the model of ad-hoc radio networks without collision detection.

First, we show the contribution for ARB and ARG.

In the network model in Subsection 1.1.1, communication tasks and its completion times has been investigated in many literatures. The network model is theoretical, and so are the results.

As we mentioned on the model without collision detection, there does not exist any ARB algorithm even for bidirectional graphs [7], which is proved by using a special case: when the source does not receive any message about the completion of the RB, the source can not distinguish between the situations that the network has only the source node (thus the source does not receive any message) and that at least two in-neighbors of the source transmit some messages (thus collision occurs).

If we assume that each node knows the number of nodes or its in-neighbors in the network, RB algorithms can be easily modified to ARB ones. It is interesting to know some weakest conditions needed for performing an ARB. In this dissertation, we show that if the network contains at least two nodes, we can construct ARB algorithms for bidirectional graphs and strongly connected graphs under the assumption that the network has no collision detection and each node knows only its ID.

The computation time of our ARB algorithm for bidirectional graphs is the same as the existing best RB algorithm which uses $O(n)$ rounds. The computation time of our ARB algorithm for strongly connected graphs is $O(6n + \sum_{i=1}^{\lceil \log n \rceil}\{2 \cdot RB(2^i) + RG(2^i)\})$, where $RB(n)$ and $RG(n)$ is the number of rounds which an RB and an RG requires for n-node strongly connected graphs, respectively. It becomes $O(n^{4/3} \log^{10/3} n)$ when using the $O(n^{4/3} \log^{10/3} n)$-time gossiping algorithm from [12].

In addition, we consider acknowledged radio gossiping (ARG) algorithms. We show that our ARB algorithms can be extended to ARG algorithms for both of bidirectional graphs and strongly connected graphs. Our ARB algorithm for bidirectional graphs needs a leader, and we use the source node to be the leader in the algorithm. In ARG, since no source node is given, we need to elect a leader for ARG when we extend the ARB algorithm to an ARG algorithm. For strongly connected graphs our ARB algorithm does not need a leader, therefore , in this case, the ARB algorithm can be extended to an ARG algorithm directly. The computation time of the extended ARG algorithms is $O(n + \sum_{i=1}^{\lceil \log n \rceil}\{LE(2^i)\})$ for bidirectional graphs and $O(6n + \sum_{i=1}^{\lceil \log n \rceil}\{RB(2^i) + 2 \cdot RG(2^i)\})$ for strongly connected graphs, respectively, where $LE(n)$ denotes the number of the rounds needed to elect a leader for n-node bidirectional graphs. The computation times of ARG algorithms become $O(n \log^3 n)$ and $O(n^{4/3} \log^{10/3} n)$, respectively, by using the $O(n \log^3 n)$-time leader election algorithm from [8] and the $O(n^{4/3} \log^{10/3} n)$-time gossiping algorithm from [12]. Our results for RB and ARB are also summarized in Table 1.1.

The second contribution of this dissertation is clustering and its maintenance algorithms.

Distributed clustering for a flat ad-hoc network topology G has been investigated in many literatures, and most of the proposed protocols end up generating cluster heads and forming a corresponding backbone. Although many efforts have been made for establishment of a hierarchical clustering on an ad-hoc network, the research for the maintenance of the cluster organization under the similar scenario is seldom seen. This dissertation puts emphasis on the maintenance. In this dissertation, we consider an ad-hoc network in which the network topology dynamically changes. We propose a novel cluster architecture on which two operations *join* and *leave* are defined for maintaining the cluster organization.

First, we show a basic cluster architecture. In our clustering, the nodes of a flat network G are grouped into disjoint clusters, and the backbone is a tree consisting of cluster heads and gateway nodes (if any, gateway nodes are used to connect heads). Furthermore, the backbone and the clusters are combined into a spanning tree of G which we call *cluster-based network* of G, denoted as $CNet(G)$. Let n be the number of nodes in G and $|BT(G)|$ be the number of nodes of backbone in our clustering. A $CNet(G)$ has several novel properties: (1) the backbone consists of at most $2p_G - 1$ nodes, where p_G is the cardinality of the minimum clique partition of G, and so when G is a dense graph, $p_G \ll n$; (2) a broadcasting on G can be executed via the backbone in $O(|BT(G)|)$ time (it needs $\Omega(n)$ time on a flat network [2]); and (3) if G is a unit disk graph, $|BT(G)| \leq 10 \times |MDS|$.

In our clustering, when a $CNet(G)$ is established, each node keeps one hop knowledge (i.e., each node knows its neighbors in $CNet(G)$ and G, respectively). We will show that a $CNet(G)$ can be established either in a static way or in a dynamic way. The static way refers to the process of gathering all topological information somewhere and the problem is solved there. On the other hand the dynamic way refers to solve the problem locally without gathering all information. Using these methods a $CNet(G)$ can be established in $O(n)$ time or in expected $O(|E|)$ time, respectively. The operations join and leave maintain the cluster architecture for G with the same properties when a node gets out of or joins into G.

Moreover, we propose another cluster architecture on which a join operation that merge two or more networks $(G_1, G_2, \ldots, G_m)$ by a joining node and a leave operation that separates a network into two or more ones by a leaving node can be performed efficiently and the size of its backbone is small. The join operation can be performed in $O(q + \max_{1 \leq i \leq m} \{|BT(G_i)|\})$ expected time on unit disk graphs, where q is the number of neighbors of joining node in G, and $|BT(G)|$ is the size of backbone tree of G[1]. And the

[1]For the join operation for general graphs, it takes a little bit more time (see section 5.4.2).

leave operation can be performed in $O(|T|)$ time on general graphs, where T is a subtree of the cluster-based network $CNet(G)$ with the leaving node as the root. Simulation shows a better result that the size of the backbone decreases by 10 percent compared with our first proposed basic architecture. Moreover, we remove the assumption that each node works in a synchronous round.

The rest of the dissertation is organized as follows: We give definitions for some technical terms in Chapter 2. Chapter 3 deals with the communication model of the networks. Chapter 4 describes ARB and ARG algorithm which is the first contribution of this dissertation. The proposed cluster-based architectures and its maintenance algorithms is presented in Chapter 5. Finally, we conclude the dissertation and show our future works in Chapter 6.

Chapter 2

Preliminaries

2.1 Graph Theoretic Definitions

We give definitions for some technical terms which will be used throughout the dissertation. Let $G = (V, E)$ be a directed graph.

- *Bi-directional graph*: A graph G is a bi-directional graph if there is an edge from node u to node v, then there is an edge from v to u.
- *Induced subgraph*: For a graph $G = (V, E)$, the graph $H = (U, F)$, where $U \subseteq V$ and F is the set of all edges in E with both ends in U, is called the subgraph of G induced by U, denoted by $G[U]$.
- *Strongly connected component*: A directed graph, in which there exists at least one path from u to v for any two distinct nodes u and v, is said to be strongly connected. A strongly connected component $C = G[U]$ of a directed graph G is a subgraph of G induced by U which satisfies that no node of G can be added to U such that C is strongly connected.
- *Tree*: For a graph $G = (V, E)$, let $G' = (V, E')$ be the graph obtained by replacing all edges in E with undirected ones. G is a tree if G' is a connected graph without cycle.
- *In-neighbors*: Node u is an in-neighbor of node v, if there is a directed edge from node u to node v.

- *Independent set*: An independent set of G is a set $U \subseteq V$ in which no pair of nodes is adjacent in G.
- *Maximal independent set (MIS)*: An independent set I of nodes in a graph G such that no more nodes can be added and it still be an independent set.
- *Dominating set*: A set $D(\subseteq V)$ of nodes is a dominating set of G if any node in G is either in D or the neighbor of a node in D.
- *Unit disk graph*: In a unit disk graph $G = (V, E)$, there is an edge $(u, v) \in E$ iff the Euclidean distance between u and v is at most 1.

Since a bi-directional graph can be treated as an undirected graph, in this dissertation, our networks are represented by undirected graphs, and the figure of a graph and technical terms also follow it (e.g. strongly connected component $\rightarrow$ connected component).

Chapter 3

Model of Radio Networks

In this dissertation, we consider the radio networks without a collision detection. We describe the model of radio networks:

- A priori knowledge of every node is limited to its own ID.
- Each node knows whether itself is a source or not in broadcasting.
- Nodes in the radio network work per round synchronized by a global clock.
- In every round, each node acts either as a transmitter or as a receiver.
- A node acting as a receiver in a given round gets a message iff exactly one of its in-neighbors transmits in this round.
- If more than one in-neighbor transmit simultaneously in a given round, collision occurs and none of the messages is received in this round.
- A node cannot notice the occurrence of a collision (i.e. without collision detection).

For simplicity we assume that each node is labeled with distinct integers between 1 and n in an n-node network. But note that all the arguments hold if the labels are distinct integers between 1 and $Z = O(n)$, and we do not use the property that the labels are in $\{1, 2, \ldots, n\}$.

For our clustering maintenance algorithms, we will remove the assumption that a round is synchronized in section 5.6.

Chapter 4

ARB, ARG

4.1 ARB and ARG in Bidirectional Graphs

In this section, we describe ARB and ARG algorithms for bidirectional graphs where the number of nodes in the network is at least 2. First, we describe the overview of our algorithms, secondly we show an ARB algorithm and then modify it to an ARG algorithm.

4.1.1 Overview of Our Algorithm

We organize the algorithms into phases. Each node judges whether ARB or ARG has ended in each phase, and if the task has not ended, proceeds to the next phase. The main idea of our algorithms is that, in k-th phase, there will have 2^k nodes will confirm their in-neighbors. In the k-th phase, first the in-neighbors of any node v whose IDs are no more than 2^k send their own IDs, thus the node v can recognize its in-neighbors' IDs that are no more than 2^k. Then in the same phase the node whose ID is the minimum one among the in-neighbors with IDs no more than 2^k, and nodes whose IDs are more than 2^k send their IDs simultaneously. If the node v receives the minimum ID (i.e. collision does not occur), it recognizes that it knows all of the in-neighbor in this phase. It is easy to perform the ARB if every node knows all of its in-neighbors. If the node v does not receive the minimum ID (i.e. collision occurs), v recognizes that it does not know all of the in-neighbors and the algorithm performs the next phase.

4.1.2 Algorithm bi-ARB

We show an ARB algorithm named bi-ARB for bidirectional graphs in an n-node radio network, where $n \geq 2$.

Algorithm bi-ARB works phase by phase, numbered by consecutive positive integers. Phase k lasts $9 \cdot 2^{k-1}$ rounds divided into four stages. Stage A consists of 2^{k-1} rounds, Stage B consists of 2^k rounds, Stage C consists of 2^k rounds, and Stage D consists of 2^{k+1} rounds. We denote the ID of node v as ID(v). We define the following notations.

- L_k : the set of nodes with IDs in $\{1, \ldots, 2^k\}$.
- G_k : the connected component containing the source of the network induced by L_k. $G_k = \phi$ if the ID of the source node is larger than 2^k.
- N_v^k : the set of IDs smaller than or equal to 2^k from the in-neighbors of node v.
- $\min(N_v^k)$: the minimum ID in N_v^k. If $N_v^k = \phi$, $\min(N_v^k) = \perp$.
- Q_v : the set of v's out-neighbors in G_k which were not yet visited by the token (mentioned in the algorithm). Q_v is initialized to N_v^k.

Note that in bidirectional graphs the in-neighbors of each node v are the same as the out-neighbors of v.

Informally we show the algorithm of phase k. Stage A is a Round Robin which intends to let each node v know its in-neighbors (and out-neighbors) whose IDs are at most 2^k (N_v^k). In Stage B each node v in L_k sends $\min(N_v^k)$, which will be the only node in in-neighbors of v that can transmit to v in the next stage C. Stage C is used to judge whether the node v of G_k knows all of its in-neighbors or not. In Stage C the node whose ID is $\min(N_v^k)$ and nodes not in L_k send their IDs, then according to whether receiving $\min(N_v^k)$ or not every node v in G_k recognizes whether it knows all its in-neighbors or not. In Stage D the source node in G_k broadcasts the source message to every node of G_k. In this stage the source node also collects the information that whether each node in G_k knows all its in-neighbors. Thereby the source node can confirm the completion of RB. We modify and use a broadcasting technique shown in [7] in this stage. The technique is also used in the next chapter, and we call it as procedure *Eulerian*.

Eulerian(H) performs a broadcast on a bi-directional graph H, where each node of H knows all its neighbors' IDs in H. Let N_v be the set of IDs of v's neighbors. In every round, at most one node acts as a transmitter and all other nodes act as receivers. A message called token containing the source message starts from the source node, visits every node and turns to the source node. At the beginning, the token is in the source node. It then visits each node in H from the source node in depth-first order, and the movement of the token constructs a spanning tree T of H and an Eulerian cycle C on this tree. Every node v maintains a list Q_v containing the set of its neighbors in H which were not yet visited by the token. Q_v is initialized to N_v. Whenever v gets the message `visited` from some neighbor w, or whenever v gets the token from w, node v removes w from the list Q_v. When v gets the token then:

If $Q_v = \emptyset$ then

- v sends the message `<ID(v),visited>`;
- v sends the token to the node w from which it got it for the first time.

If $Q_v \neq \emptyset$ then

- v sends the message `<ID(v),visited>`;
- v sends the token to the node w with the smallest ID in the list Q_v.

In both cases both messages are concatenated and are sent in a single round. The procedure finishes when the list Q_s of the source node s is empty and s receives the token. This concludes the description of procedure *Eulerian*(H).

The following lemma holds for *Eulerian*.

Lemma 4.1. [7] *Let $H = (V, E)$ be a connected bi-directional graph. If each node of H knows all its neighbors' IDs in H, procedure Eulerian(H) completes broadcasting for H in $O(|V|)$ rounds.*

Now we formally show our algorithm bi-ARB.

bi-ARB Phase 0 consists of one round, the node with ID 1 (if it possibly exists) acts as a transmitter and sends its ID in this phase. The other nodes act as receivers.

Hereafter, we explain phase $k > 0$, of bi-ARB.

Stage A. The rounds in Stage A of phase k are numbered by integers $2^{k-1}+1, \ldots, 2^{k-1}+2^{k-1}$. In round number i of Stage A only the node v with ID i acts as a transmitter and sends a message ID(v).
Stage B. The rounds of this stage are numbered by integers $1, \ldots, 2^k$. In round i of Stage B only the node v with ID i acts as a transmitter and sends a message $\min(N_v^k)$. If $\min(N_v^k) = \perp$, the node v sends no message. The node w that receives $\min(N_v^k)$ stores it if ID(w)=$\min(N_v^k)$.
Stage C. The rounds in Stage C of phase k are numbered by integers $1, \ldots, 2^k$. In round i of Stage C, the node v with ID i acts as a receiver. The node with ID=$\min(N_v^k)$ acts as a transmitter and sends its ID and all the nodes whose IDs are larger than 2^k (not only in-neighbors of v) also send their own IDs in the round.

Every node v not receiving $\min(N_v^k)$ in the round ID(v), is set to the state ***warned*** which means that v does not know all its in-neighbors, or in other words, v has the in-neighbors whose IDs are larger than 2^k.
Stage D. The rounds in Stage D of phase k are numbered by integers $1, \ldots, 2^{k+1}$. The source initiates Stage D if its ID is less than or equal to 2^k. Otherwise all nodes do nothing in these 2^{k+1} rounds. We use a message called token. At the beginning of this stage every node $v \in G_k$ knows its out-neighbor N_v^k in G_k and maintains a list Q_v containing the set of its out-neighbors in G_k which were not yet visited by the token.

When a ***warned*** node sends the token to an out-neighbor, it appends a ***warning*** message to the token, and the out-neighbor getting the token becomes ***warned***.

When node v gets the token, it acts as follows:

step 1. Node v sends the message `<ID(v),visited>`. If a node u receives the message, it removes v from the list Q_u.

step 2. Node v sends the token `<source message, ID(w), (warning)>` to the following node w:

- (i) If $Q_v = \phi$, w is the node from which v got the message in step 1 for the first time.
- (ii) If $Q_v \neq \phi$, w is the node with the smallest ID in the list Q_v.

the messages are concatenated and are sent in a single round. Node w which gets the token repeats the procedure of step 1 and step 2.

If, at the end of phase k, the source is ***warned***, it knows that the RB has not been completed, and proceeds to the next phase. Otherwise the algorithm terminates.

In Appendix A we give the pseudocode of bi-ARB that each node executes.

Correctness of Algorithm bi-ARB

Lemma 4.2. *The following invariants are maintained after phase k of bi-ARB, for any positive integer k.*

- *Every node v knows the N_v^k, the set of IDs at most 2^k from the in-neighbors(and out-neighbors) of v.*
- *Every node in G_k knows the source message, if G_k contains the source node.*

Proof. In phase $k = 0$, G_k contains only the source node if its ID equals 1, and $N_v^k = \phi$. Therefore, Lemma 1 holds obviously in this case.

Assume that the invariants hold after phase $k-1$, $k \geq 1$. We show that the invariants are maintained after phase k.

In Stage A of phase k, the nodes whose IDs are between 2^{k-1} and 2^k transmit their IDs. In every round, exactly one node acts as a transmitter and the other nodes act as receivers, hence collisions are avoided. Any node v has already known N_v^{k-1} after phase $k-1$ from the assumption, v learns $N_v^k - N_v^{k-1}$ the remaining neighbors in G_k during phase k.

In Stage D if G_k contains the source node, the token is patrolled from the source node to all nodes in G_k. At the beginning of Stage D the token is in the source node. It visits each node of G_k from the source node in depth-first order. When node v got the token, it sends the token with the source message and its ID to its out-neighbors which have not received the token yet, following the Eulerian cycle C_k of a spanning tree of G_k as follows: Q_v is the set of out-neighbors of v in G_k which were not yet visited by the token. The node v that receives the token has to send the message `<ID(v), visited>` to its in-neighbors, node w that receives the message removes v from the list Q_w. If v has the neighbors which are not visited by the token, it passes the token to the one with the smallest ID. Else, v returns the token to the node from which it got the token for the

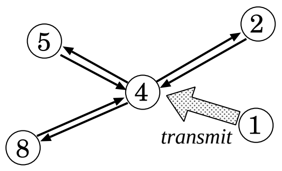

Figure 4.1: knowing all in-neighbors (k=3)

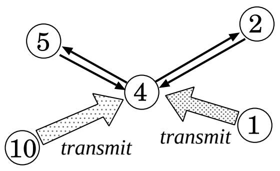

Figure 4.2: otherwise (k=3)

first time. In Stage A every node v in G_k knows its out-neighbors in G_k, so the token patrols every node in G_k and returns to the source finally. $\square$

Theorem 4.3. *Algorithm bi-ARB performs an ARB in time $O(n)$, for any n-node bidirectional graph with $n \geq 2$.*

Proof. Let l be such that $2^{l-1} < n \leq 2^l$. It is sufficient to show that

(1) After phase l all nodes of the network get the source message.

(2) At the end of phase l the source is not ***warned***.

In order to prove (1) consider phase l. Since G_l is the entire network, (1) follows from Lemma 4.2. The number of the rounds needed for this algorithm is at most $\sum_{i=1}^{l} 9{\cdot}2^{i-1} \leq 9 \cdot 2^l \leq 18n$.

We prove (2). At the end of Stage A each node v knows N_v^k. It sends $\min(N_v^k)$ in round i=ID(v) of Stage B. The node w receiving $\min(N_v^k)$ memorizes the number of the round if ID(w)=$\min(N_v^k)$, otherwise ignores the message. Thereby in round i of Stage C only the node with ID i can act as transmitter.

A node in G_k recognizes whether it knows all its in-neighbors in Stage C. In round i of this stage for the node v with ID i, the node with ID=$\min(N_v^k)$ and the nodes with IDs larger than 2^k send their own IDs. Therefore the node v having in-neighbors with ID larger than 2^k cannot receive $\min(N_v^k)$ in round ID(v) due to a collision. Then v recognizes that it does not know all in-neighbors, and becomes ***warned***. If v knows all in-neighbors, it can receive $\min(N_v^k)$ and will not become ***warned***. Figure 4.1 shows the case where a node knows all in-neighbors, and Figure 4.2 shows the other case in round 4 of phase 3, where the number of node represents its ID.

Consider phase l. Since there is no node whose ID is larger than 2^l, each node v can receive $\min(N_v^k)$ in the round ID(v) in Stage C. Therefore no node becomes ***warned*** in Stage D. Hence the source node is not ***warned*** at the end of phase l. $\square$

Message size. Let S be the maximum length of the message transmitted each time and let r be the length of the source message. In Stage A,B and C each node transmits at most one ID respectively, thus $S = O(\log n)$. In Stage D each node transmits message `<ID(v), visited>` and the token `<source message, ID(w), (warning)>`, thus $S = O(r + \log n)$. Hence the maximum message size is at most $O(r + \log n)$ for algorithm bi-ARB.

4.1.3 Algorithm bi-ARG

The ARG algorithm bi-ARG for bidirectional graphs is obtained by changing a part of bi-ARB.

Algorithm bi-ARG works in phases, numbered by consecutive positive integers similar to bi-ARB. Each phase consists of four stages A,B,C and D. Stage A,B,C are the same as these of algorithm bi-ARB but Stage D is different. It needs a leader election procedure and an extra token patrolling. Recall that in bi-ARB, the source node is used to be the starting point of the token patrolling. Furthermore, each node knows whether itself is a source or not. But the source node does not exist for ARG. We have to elect one leader for each connected component induced by L_k so that the token patrolling can be performed in each component. We use a leader election procedure. The leader of each connected component acts as initiator and makes the token patrol twice in its connected component in Stage D. In the first patrol the leader of each connected component collects the messages which each node has and ***warning*** messages from the nodes to the leader (the same as that in bi-ARB), then in the second patrol it disseminates the messages which were collected in the first patrol to all the nodes in the component. Thereby any node knows whether RG have completed or not.

In order to use an leader election algorithm, each node must know the completion time of the algorithm, since the leader election procedure must finish in each phase of bi-ARG.

For example, we can use the algorithm FIND MAX shown in [8] as a leader election procedure. The algorithm FIND MAX elects a leader by calculating the maximum ID on a strongly connected graph under the assumption that each node knows the upper bound of IDs of nodes in the network. Moreover, if each node knows (the upper bound of) the number of nodes n in the network, it can compute the completion time of FIND MAX, which is $cn\log^3 n$ for some known constant c. Algorithm FIND MAX finds $id_{max} = \max_v \mathrm{ID}(v)$, using binary search. At each step, all nodes know that the id_{max} (the node having this ID is elected as a leader) among all nodes is between a and b, where $a \leq b$. Initially $a = 0$ and $b = n$. If $a = b$, then $id_{max} = a$, and the computation of id_{max} is complete. For $a < b$, we proceed as follows. Let $c = \lceil (a+b)/2 \rceil$. Each node v for which $c \leq \mathrm{ID}(v) \leq b$ sends the message $[c, b]$ to all other nodes. Since all these nodes send the same message, we consider an extension of the RB that broadcasts from several source nodes with the same messages to all reachable nodes, and use the algorithm that performs such an extended RB with completion time $RB(n)$. Then, after $RB(n)$ steps, either all nodes will receive the message $[c, b]$, in which case they know that the maximum is between c and b, or (informing by silence) all nodes will not receive anything for time $RB(n)$, in which case they know the maximum is between a and $c - 1$. Depending on the outcome, either all nodes update their $[a, b]$ intervals (containing maximum) to $[\lceil (a+b)/2 \rceil, b]$ or all of them update it to $[a, \lceil (a+b)/2 - 1 \rceil]$.

In each phase we use this algorithm to elect a leader for each connected component. In phase k, the upper bound of IDs and that of the number of nodes in the connected components induced by L_k is known to be 2^k.

Theorem 4.4. *Algorithm bi-ARG performs an ARG in time $O(n + \sum_{i=1}^{\lceil \log n \rceil} \{LE(2^i)\})$, for any bidirectional graph with $n \geq 2$, where $LE(k)$ denotes the number of the rounds of any leader election algorithm for k-node bidirectional graphs in which each node knows the completion time.*

Proof. Algorithm bi-ARG works in phases and stages similar to bi-ARB. Stage A,B,C are the same as these of algorithm bi-ARB. In Stage D, it needs a leader election procedure and an extra token patrolling. From the proof of Theorem 4.3, all the process except for the leader election can be done in time $O(n)$. For each phase i, the leader election algorithm takes at most $LE(2^i)$ rounds. Therefore, algorithm bi-ARG performs an ARG in time $O(n + \sum_{i=1}^{\lceil \log n \rceil} \{LE(2^i)\})$. □

We obtain the following corollary from Theorem 4.4 using the $O(n\log^3 n)$-time leader election algorithm FIND MAX.

Corollary 4.5. *Algorithm bi-ARG performs ARG in time $O(n \log^3 n)$, for any bidirectional graph with $n \geq 2$.*

Proof. From Theorem 4.4 and the $O(n \log^3 n)$-time leader election algorithm FIND MAX, the completion time of bi-ARG is

$$\begin{aligned} n + \sum_{i=1}^{\lceil \log n \rceil} 2^i \log^3 2^i &\leq n + \sum_{i=1}^{\lceil \log n \rceil} 2^i \cdot i^3 \\ &\leq n + 2(2^{\lceil \log n \rceil} - 1) \cdot (\log n + 1)^3 \\ &\leq n + 4n \cdot (\log n + 1)^3. \end{aligned}$$

Hence, algorithm bi-ARG performs ARG in time $O(n \log^3 n)$. □

Our algorithm bi-ARG is improvable if more efficient leader election algorithms can be designed for bidirectional graphs under the condition that each node knows the maximum of IDs and n.

Message size. Let S be the maximum length of the message transmitted each time and let r be the length of the message each node has. In Stage A,B and C, $S = O(\log n)$ which are the same as that of bi-ARB. In Stage D first $S = O(\log n)$ for the leader election procedure FIND MAX [8]. Next each node adds its own message to the token, $S = O(rn + \log n)$. Hence the maximum message size is at most $O(rn + \log n)$ for algorithm bi-ARG.

4.2 ARB and ARG in Strongly Connected Graphs

4.2.1 Algorithm st-ARB

The ARB algorithm st-ARB for strongly connected graphs is obtained by changing a part of bi-ARB.

Algorithm st-ARB works in phases, numbered by consecutive positive integers. Every phase starts in the round following the end of the previous phase. Phase $k(> 0)$ lasts $3 \cdot 2^{k-1} + 2 \cdot RB(2^k) + RG(2^k)$ rounds divided into four stages. Stage A consists of 2^{k-1}

rounds, Stage B consists of $RG(2^k)$ rounds, Stage C consists of 2^k rounds, and Stage D consists of $2 \cdot RB(2^k)$ rounds.

Here we show the outline of this algorithm in phase k. Stage A and C of st-ARB are the same as those of bi-ARB, and the purpose of Stage B and D also does not change. Although in bidirectional graphs a node v can transmit $\min(N_v^k)$ to its in-neighbor w whose ID=$\min(N_v^k)$ because the in-neighbors of v is also its out-neighbors, node v cannot do that in strongly connected graphs since w may not be an out-neighbor of v. To do this, v must gossip on the subgraph induced by L_k in Stage B. In Stage D each node other than the source node in L_k transmits the ***warning*** message and the source node broadcasts the source message. Thereby the source node can confirm the completion of RB.

In st-ARB we use the RB and RG in the subgraph induced by L_k (not necessarily strongly connected). In order to apply the RB algorithm for strongly connected graphs to our algorithm, it is sufficient to perform the task for all reachable nodes. About RG algorithm, it is not necessary to perform the task for all reachable nodes. Any algorithm of RB and RG can be applied to our algorithm if each node knows the completion time. We consider an extension of the RB that broadcasts from several source nodes with the same messages to all reachable nodes, and use the algorithm that performs such an extended RB in Stage D. Since the algorithm does not depend on the information of the source node, it can perform an RB in the situation such that several source nodes exist.

st-ARB Phase 0 consists of one round, the node with ID 1 acts as transmitter and sends its ID in this phase. The other nodes act as receivers.

Hereafter, we explain phase $k(> 0)$ of st-ARB. Stage A and C is the same as that of bi-ARB. Every node that is not transmitter is receiver in the explanation.

Stage A. Rounds in Stage A of phase k are numbered by integers $2^{k-1}+1, \dots, 2^{k-1}+2^{k-1}$. In round number i of Stage A the only node v with ID i acts as a transmitter and sends a message ID(v).

Stage B. Stage B consists of $RG(2^k)$ rounds. In Stage B each node v in L_k acts as a transmitter, gossiping the message `<ID(v), min(N`$_v^k$`)>`. If $\min(N_v^k) = \lambda$, the node v sends no message.

Stage C. Rounds in Stage C of phase k are numbered by integers $1, \dots, 2^k$. In round number i of Stage C the node v with ID i acts as a receiver. The node with ID $\min(N_v^k)$ and the nodes whose IDs are larger than 2^k act as transmitter, sending their own IDs.

Every node v not receiving $\min(N_v^k)$ in the round ID(v), is set to the state ***warned***.
Stage D. Stage D consists of $2 \cdot RB(2^k)$ rounds. First, each node sends a ***warning*** message if it is ***warned***. Next, if the source does not receive the ***warning*** message, it knows that there is no node in L_k whose in-neighbors with ID$> 2^k$ and then broadcasts the source message, otherwise it knows that there still exist nodes in L_k whose in-neighbors with ID$> 2^k$ and then it becomes ***warned***, and shifts to the next phase.

Correctness of Algorithm st-ARB

Lemma 4.6. *If there are* ***warned*** *nodes in the strongly connected graph after phase k of st-ARG then there is a path from at least one* ***warned*** *node to the source node that contains only nodes whose IDs are not larger than 2^k.*

Proof. Let v be some ***warned*** node. In the original graph there is a path from v to the source. If there are nodes with ID$> 2^k$ in this path, let the out-neighbor of the last of them in the path be v'. The path from v' to the source proves the lemma. □

Theorem 4.7. *Algorithm st-ARB performs ARB in time $O(6n + \sum_{i=1}^{\lceil \log n \rceil}\{2 \cdot RB(2^i) + RG(2^i)\})$, in any strongly connected graphs with n nodes, where $n \geq 2$ and $RB(k)$ and $RG(k)$ denotes the number of the rounds of any extended RB and RG algorithm for k-node strongly connected graphs in which each node knows the completion time, respectively.*

Proof. Let l be such that $2^{l-1} < n \leq 2^l$. It is enough to show that

(1) After phase l all nodes of the network get the source message.

(2) At the end of phase l the source node does not ***warned***.

In order to prove (1) consider phase l. Since L_l is the entire network, each node considers the upper bound of the number of nodes is 2^l and does broadcasting, then every node gets the source message. The completion time of this algorithm is at most

$$\sum_{i=1}^{l}\{3 \cdot 2^{i-1} + 2 \cdot RB(2^i) + RG(2^i)\} \leq 6n + \sum_{i=1}^{\lceil \log n \rceil}\{2 \cdot RB(2^i) + RG(2^i)\}$$

We prove (2). Since Stage A is the same as that of bi-ARB for phase k, any node v knows N_v^k in the stage.

In Stage B each node v in L_k gossips `<ID(v), min(`$\mathtt{N_v^k}$`)>`. If the gossiping are performed correctly, in Stage C only one node in N_v^k can act as transmitter. If L_k does not contain all nodes of the graph, the induced subgraph by L_k is not necessarily strongly connected and the gossiping of all messages is not secured. But L_l contains all node in the graph, all messages are gossiped correctly.

Stage C is also the same as that of bi-ARB, each node v recognizes whether it knows all its in-neighbors. Similar to bi-ARB the node v having in-neighbors with ID larger than 2^k cannot receive $\min(N_v^k)$ in round $\mathrm{ID}(v)$. The node v which could not receive $\min(N_v^k)$ recognizes that it does not know all in-neighbor, and becomes ***warned***. If there is no node with ID$> 2^k$ in the graph, all messages are gossiped in Stage B. It means that v can receive $\min(N_v^k)$ in Stage C and does not become ***warned***.

In Stage D each node confirms whether it receives the ***warning*** message or not, and the source node sends the source message. From Lemma 4.6 if there exists at least one ***warned*** node, its ***warning*** message reaches the source node. Then the source node knows that there exist the nodes in the graph with ID$> 2^k$. Consider phase l, since there is no node in the graph with ID$> 2^l$, each message of any node is gossiped to all nodes in Stage B correctly. Therefore any node does not become ***warned*** in Stage C. Hence, the source node confirms the completion of RB and is not ***warned*** at the end of phase l since L_l is the entire network and there is no ***warned*** node in the graph. □

We obtain the following corollary from Theorem 4.7 using the $O(n\log^2 n)$-time broadcasting algorithm from [8] and the $O(n^{4/3}\log^{10/3} n)$-time gossiping algorithm from [12]. The broadcasting Algorithm DOBROADCAST from [8] which can perform the extended RB is as follows: All sets considered in this algorithm are subsets of $\{1,\ldots,n\}$. We say that a set S *hits* a set X iff $|S\cap X| = 1$, and that S *avoids* X iff $S\cap X = \emptyset$. Given a positive integer w, a family $\overline{S}$ of sets is called a w-*selector* if it satisfies the following property: "For any two disjoint sets X, Y with $w/2 \le |X| \le w$ and $|Y| \le w$ there exists a set in $\overline{S}$ which hits X and avoids Y". The algorithm is specified as a sequence of *transmission sets*. At each round t, the nodes that transmit the message are those that belong to the t-th transmission set and have already received the message. For each $j = 0,\ldots,\log n$ let

$\overline{S}_j = (S_{j,0}, S_{j,1}, \ldots, S_{j,m_j-1})$ be a 2^j-selector with $m_j = O(2^j \log n)$ sets. The algorithm DOBROADCAST consists of stages, with each stage having $\log n + 1 = O(\log n)$ steps. The transmission set at the j-th round of stage s is $S_{j,\ s \bmod m_j}$. Since each node does not use the information whether it is the source or not and does not depend on the message it received in the previous round, RB can be done on condition that several source nodes have the same message. Each node can compute the completion time of each algorithm under the assumption that it knows the upper bound of IDs of nodes in the network.

Corollary 4.8. *Algorithm st-ARB performs ARB in time $O(n^{4/3} \log^{10/3} n)$, for any strongly connected graphs with $n \geq 2$.*

Message size. Let S be the maximum length of the message transmitted each time and let r be the length of the source message. In Stage A and C each node transmits at most one ID, thus $S = O(\log n)$. In Stage B each node v gossips ID(v) and $\min(N_v^k)$, thus $S = O(n \log n)$. In Stage D each node transmits a ***warning*** message, the source node transmits the source message, thus $S = O(r)$. Hence the maximum message size is at most $O(r + n \log n)$ for algorithm st-ARB.

4.2.2 Algorithm st-ARG

The ARG algorithm st-ARG for strongly connected graphs is obtained by changing a part of st-ARB.

Algorithm st-ARG works in phases, numbered by consecutive positive integers as well as st-ARB. Stages A,B and C are the same as those of st-ARB. We perform ARG by changing Stage D. Stage D consists of $RB(2^k) + RG(2^k)$ rounds. First step where each node confirms whether it receives the ***warning*** message or not is the same as that of Stage D of st-ARB. If a node does not receive ***warning*** message, it knows that there is no node with ID$> 2^k$ and gossips its own message, otherwise it knows that there still exist nodes with ID$> 2^k$ and becomes ***warned***, then shifts to the next phase.

Theorem 4.9. *Algorithm st-ARG performs ARG in time $O(6n + \sum_{i=1}^{\lceil \log n \rceil} \{RB(2^i) + 2 \cdot RG(2^i)\})$, for any strongly connected graph with n nodes, where $n \geq 2$ and $RB(k)$ and $RG(k)$ denotes the number of the rounds of any RB and RG algorithm for k-node strongly connected graphs in which each node knows the completion time, respectively.*

We obtain the following corollary from Theorem 4.9 using the $O(n\log^2 n)$-time broadcasting algorithm from [8] and the $O(n^{4/3}\log^{10/3} n)$-time gossiping algorithm from [12] as well as Corollary 4.8.

Corollary 4.10. *Algorithm st-ARG performs ARG in time $O(n^{4/3}\log^{10/3} n)$, for any strongly connected graph with n nodes, where $n \geq 2$.*

Message size. Let S be the maximum length of the message transmitted each time and let r be the length of the message each node has. In Stage A,B and C, $S = O(n\log n)$ is the same as that of st-ARB. In Stage D each node v broadcasts a ***warning*** message and gossips its own message, thus $S = O(rn)$. Hence the maximum message size is at most $O(rn + n\log n)$ for algorithm st-ARG.

Chapter 5

Cluster-based Architecture

As shown in the previous chapter, the upper bound and the lower bound for deterministic distributed broadcasting are $O(n)$ rounds and $\Omega(n)$ rounds for bidirectional graphs. And we propose the $O(n)$ rounds ARB algorithm when $n \geq 2$ in Chapter 4. In order to provide the flat ad-hoc network with a hierarchical architecture and perform broadcasting fast, clustering is considered. Distributed clustering for a flat ad-hoc network topology G has been investigated in many literatures. Although many efforts have been made for establishment of a hierarchical clustering on an ad-hoc network, the research for the maintenance of the cluster organization is seldom seen. In this chapter, we consider an ad-hoc network in which the network topology dynamically changes, and propose a novel cluster-based architecture on which two operations join and leave are defined for maintaining the cluster organization. On the architecture faster broadcasting can be archived.

5.1 Basic Architecture

In this section, we define a basic architecture which is used our proposed architectures.

Let $G = (V, E)$ be a connected bi-directional graph. *A cluster* of G is a star subgraph of G, where one node, called a *head*, has an edge to each other node called a *member*. No edge exists between any two members in the cluster. *A clustering* of G is to partition G into node disjoint clusters. The union of the clusters is produced by the clustering of G and denoted as $C(G) = (V, E_C)$, where E_C is the set of edges between the heads and their members. In order to minimize the number of clusters, our clustering does not allow two heads to be neighbors with each other. In other words, the set of the heads in our clustering is a maximal independent set in G. Our clustering makes any two heads are joined by one special member called *gateway node* which is in an intersection of neighbors in G of two heads.

Clustering provides a hierarchical organization to a flat graph G. *A backbone* of G is a connected subgraph of G formed by only heads and gateway nodes, where a gateway node is connected with two or more heads. Since heads can not be neighbors with each other, any edge in a backbone must be formed between a head and a gateway node. Since G is a connected bi-directional graph, a backbone must exist. *A backbone tree* of G, denoted as $BT(G) = (V_{BT}, E_{BT})$, is defined to be a spanning tree of a backbone of G (see Figure 5.1).

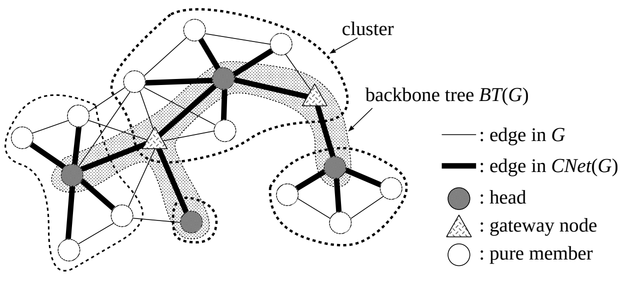

Figure 5.1: G, $BT(G)$ and $CNet(G)$

Backbone tree $BT(G)$ can be considered as a communication highway on G. To see this, let u and v be two members, and h_u and h_v be their heads, respectively. If u wants to send a message to v, u first sends the message to its head h_u. h_u then sends the message to h_v via $BT(G)$, and finally h_v sends the message to its member v. A transmission between a head and its members is called *local transmission*, and a transmission between heads is called *backbone transmission.*

Now we use the backbone tree to connect the clusters for forming a structured network on G. *A cluster-based network* of $G = (V, E)$ is a rooted tree $CNet(G) = (V, E_{BT} \cup E_C)$ with one head as a root. The edges of E_{BT} come from the backbone tree and the edges of E_C come from all the clusters (see Figure 5.1). Since gateway nodes are also members, we also call the members which are not gateway nodes as *pure members*. In *CNet*(G), pure members are connected only with their heads.

In the following sections, we will show how a graph G can be organized and maintain to a cluster-based network *CNet*(G). Before we discuss the algorithms, we first define a data structure for *CNet*(G) clearly.

A *CNet*(G) has two level structures: a set of clusters, and a backbone tree which is used to connect the clusters. Each node v in G maintains the information described below:

- *v.stat*: v's status, head, gateway or member.
- *v.prt*, *v.chd*: the ID of v's parent and the set of IDs of v's children on *CNet*(G), respectively. For a root r, $r.prt = \perp$, and for each node m who has no child, $m.chd = \emptyset$.
- *v.oneigh*: the set of IDs of v's neighbors on G except *v.prt* and *v.chd*.
- *v.rootID*: the ID of the root of *CNet*(G) to which v belongs.

Each node maintains its neighbor's status and ID as pairs. Hereafter we use *v.neigh* as the neighbors of v on G and *v.bneigh* as the neighbors of v on $BT(G)$, respectively (these can be derived from *v.prt*, *v.chd* and *v.oneigh*). We call above information as *total 1-hop data*. When the information are maintained for each node in G, it is called that G *is organized* with total 1-hop data.

We define two operations *join* and *leave* on a *CNet*(G).

- join: Given disjoint graphs $G_1, G_2, \ldots, G_m$, *CNet*(G_1), *CNet*$(G_2), \ldots,$ *CNet*(G_m) and a joining node u which connects these graphs, *CNet*(G_1), *CNet*$(G_2), \ldots,$ *CNet*(G_m) are reconfigured to one *CNet*(G).
- leave: Given a graph G, *CNet*(G) and a leaving node which divides G into disjoint graphs G_1, $G_2, \ldots, G_m$, *CNet*(G) is reconfigured to *CNet*(G_1), *CNet*(G_2), $\ldots$, *CNet*(G_m).

Fig.5.2 shows the operations join and leave.

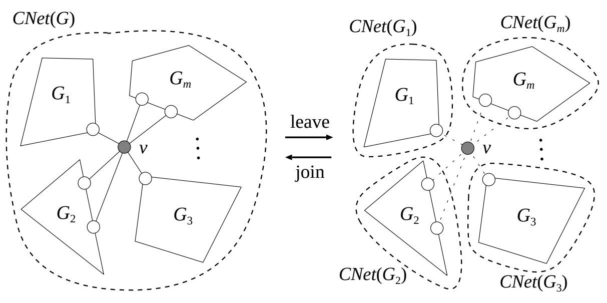

Figure 5.2: Operations join and leave

In what follows, we also call the *join* which makes two or more graphs to one connected graph as *merge*, and call the *leave* which separates a graph to two or more disjoint graphs as *separation*.

Since any two heads are not adjacent with each other on G and any two gateway nodes are not adjacent with each other on $BT(G)$, if a joining node v connects a head and a gateway node in different $CNet(G)$ and $CNet(G')$, v cannot be head nor gateway node and the merge are not done efficiently. So, we consider better architectures in Section 5.4 where the two operation can be performed efficiently, moreover, the size of its backbone tree is smaller than this architecture.

5.2 Properties of the Basic Architecture

The basic architecture in Section 5.1 has the following properties.

Lemma 5.1. *Let G be a connected bidirectional graph and $BT(G)$ be a backbone tree of G. If $BT(G)$ has p heads, then the number of nodes in $BT(G)$ is at most $2p-1$ nodes.*

Proof. We define a red-connected-by-blue tree to be a tree which contains only red and blue nodes. Each blue node connects two or more red nodes, and no edge exists between any two red nodes. From the definition a blue node can not be a leaf.

Obviously, $BT(G)$ is a red-connected-by-blue if we consider heads to be red and gateway nodes to be blue. We prove the conclusion that if a red-connected-by-blue tree has p red nodes, then it has at most $2p-1$ nodes totally.

Let T be a red-connected-by-blue tree with p red nodes. If $p = 1$, T contains only one node which is red. Therefore, the conclusion holds when $p = 1$. Assuming that the conclusion holds when $p = k$, we prove the conclusion holds for the case $p = k + 1$.

Let u be a leaf of T. Obviously, u is red and is adjacent to some blue node g in T. If g connects more than two red nodes, then remove u from T. Otherwise, g connects two red nodes in which one is u, and in this case we remove both u and g from T. No matter which situation the resulting tree T' is a red-connected-by-blue tree with k red nodes. From the induction assumption, $|T'| \leq 2k - 1$, therefore $|T| \leq 2k - 1 + 2 = 2(k + 1) - 1$. □

The following lemmas use the fact that heads are not adjacent to each other on G.

Lemma 5.2. *Let G be a connected bidirectional graph and p_G be the cardinality of minimum clique partition of G. The number of heads in $CNet(G)$ is at most p_G.*

Proof. From the definition of the clustering of G, there is only one head in one cluster, and the heads are not connected with each other. On the other hand, any two nodes are connected with each other in a complete graph. Therefore, there exists at most one head in any complete subgraph of G. Hence, the number of clusters in $CNet(G)$ is at most p_G. □

Lemma 5.3. *Let $G = (V, E)$ be a unit disk graph, and $MDS(G)$ be the minimum dominating set of G. The number of heads in $CNet_C(G)$ is not larger than $5 \times |MDS(G)|$.*

Proof. For a minimum dominating set $MDS(G)$, we show that each node $x \in MDS(G)$ has at most 5 heads as its neighbors in the $CNet(G)$.

If $x \in MDS(G)$ is a head, then it does not have any other head as its neighbor. Otherwise, we prove that x can have at most 5 heads as its neighbors. Let's consider the nodes of G as the points on the plane.

Assume that x has heads u and v as its neighbors.

In triangle uxv, (u, x) and (x, v) are the edges in G. Therefore, $|ux| \leq 1$ and $|vx| \leq 1$. On the other hand, since u and v are heads, there is no edge between them in G. Therefore, $|uv| > 1$. Hence, the angle between (u, x) and (x, v) is larger than $60°$. It means that x has at most 5 heads as its neighbors. □

Lemma 5.4. *Let $G = (V, E)$ be a unit disk graph, and $BT(G)$ be a backbone tree of G. The maximum degree of $BT(G)$ is constant.*

Proof. Gateway nodes are adjacent to only heads on $BT(G)$. Therefore, the degree of gateway nodes on $BT(G)$ is less than 6 from the proof of Lemma 5.3.

Heads are adjacent to only gateway nodes and gateway nodes connect two or more heads on $BT(G)$. So, the degree of any head h on $BT(G)$ is less than or equal to the number of heads within two hops from h. Since the number of heads within two hops is less than 20 [13], the maximum degree of $BT(G)$ is at most 19. □

5.3 Broadcasting on A Cluster-based Architecture

In this section, we present our broadcasting algorithm using $BT(G)$.

We show our broadcasting algorithm BroadcastALG in *CNet*(G), where s is the source node with a message M and needs to be informed to the rest of the nodes in a given network G.

A broadcasting in *CNet*(G) can be completed by performing *Eulerian* on $BT(G)$. Algorithm 1 shows our algorithm BroadcastALG.

Algorithm 1 BroadcastALG

```
1: if source node $s$ is a pure member then
2:     $s$ sends source message $M$ to $s.prt$;
3:     $s.prt$ calls procedure $Eulerian(BT(G))$;
4: else
5:     $s$ calls procedure $Eulerian(BT(G))$;
6: end if
```

Theorem 5.5. *Given G and $BT(G) = (V_{BT}, E_{BT})$, a broadcasting on G can be done in $O(|V_{BT}|)$ rounds.*

Proof. Each node in $BT(G)$ knows its neighbors in $BT(G)$. If a source node is in $BT(G)$, a broadcasting on $BT(G)$ is completed in $O(|V_{BT}|)$ rounds from Lemma 4.1. Since every cluster head in *CNet*(G) is in $BT(G)$ and participates in relaying the source message, all the members in *CNet*(G) receive the source message from their cluster heads when

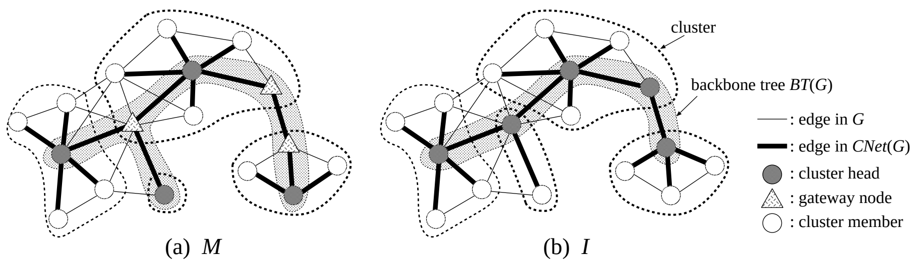

Figure 5.3: Architectures $\mathcal{M}$, and $\mathcal{I}$

they relay the source message on $BT(G)$. If the source node is not in $BT(G)$, it takes one more round to send message to its cluster head. Hence, a broadcasting from any node can be done in $O(|V_{BT}|)$ rounds. □

The time of above broadcasting algorithm is based on $BT(G)$, so the smaller the size of $BT(G)$, the faster a broadcast can be done.

5.4 Better Architectures

In this section, we present the main proposed architecture $\mathcal{I}$ and its algorithms that enables merge/separation efficiently. And then we improve $\mathcal{I}$ to have a smaller backbone tree. Before the description of $\mathcal{I}$, we introduce other derivative architecture [18] where merge/separation can be done efficiently.

Architecture $\mathcal{M}$ in [18]: On architecture $\mathcal{M}$, the backbone tree consists of heads and gateways, where the set of heads is an independent set of G as well as the basic architecture in Section 5.1. Gateways are allowed to be adjacent to each other on the backbone tree (see Fig.5.3). By allowing gateway nodes to be adjacent on backbone tree, merge/separation operations can be performed on $\mathcal{M}$, but the size of the backbone tree is larger than that of the basic architecture.

5.4.1 Architecture $\mathcal{I}$

On the basic architecture in Section 5.1, the merge of two or more cluster-based networks caused by joining of a node and separation of a cluster-based network into two or more ones caused by a leaving node are not done efficiently since the condition of a backbone tree is strict (that is, heads are not adjacent on G and gateway nodes are not adjacent on $BT(G)$). If a head in one cluster-based network is connected to a gateway of the other cluster-based network via a joining node, in a simple way, all of the nodes in a cluster-based network need to call join operations to reconstruct a new one. We propose architecture $\mathcal{I}$ that enables merge/separation efficiently and is constructed by only heads and members.

Let $CNet_{\mathcal{I}}(G)$ be the cluster-based network of G with architecture $\mathcal{I}$, and $BT_{\mathcal{I}}(G)$ be its backbone tree. $CNet_{\mathcal{I}}(G)$ and $BT_{\mathcal{I}}(G)$ have the same properties $CNet(G)$ and $BT(G)$ have. In addition, on architecture $\mathcal{I}$, a set of heads is not independent set of G and a backbone tree is constructed only by heads (see Fig.5.3).

5.4.2 Algorithms

Join operation:

First we show the join algorithm for $\mathcal{I}$.

Let *new* be a node who wants to join a network $G = (V, E)$, where G consists of disjoint subgraphs $G_1, G_2, \ldots, G_m$. Let $G' = (V \cup \{new\}, E \cup E_{new})$ in this subsection, where $E_{new} = \{(u, new) | u$ is in the transmitting range of the node *new*, $u \in V\}$. Let $q = |E_{new}|$. We simply use "neighbors" as "neighbors in G'" in this subsection.

What should be performed by the join operation is to decide the status of *new*, construct a backbone tree, and update the information which the neighbors of *new* in G' have.

First, in order to decide the status of *new* and whether a merge is caused, *new* needs to know the status of its neighbors and their rootIDs. In the case that *new* receives only one rootID from its neighbors, no merge is necessary. In this case, if there exist heads in the neighbors of *new* in G', *new* selects one to be it's head and itself becomes a member. Else there are no heads in its neighbors, *new* becomes a head and sets one neighboring member to be a head. Based on the decided status of *new*, the neighbors of *new* update their information. This process is shown in Procedure $\mathcal{I}$-status. In any case, the process affects only 2-hop neighbors of *new*.

In the case that *new* receives two or more rootIDs, we merge $G_1, G_2, \ldots, G_m$. If every node in G_i simply calls $\mathcal{I}$-status one after another to construct $CNet_{\mathcal{I}}(G)$, the size of the backbone tree does not become larger (will prove later) but the join operation takes $O(n)$ rounds and is not efficient, where n is the number of nodes in G. Here, we control the order of $\mathcal{I}$-status called by each node, and let each node b in $BT_{\mathcal{I}}(G_i)$ $(1 \le i \le m)$ call $\mathcal{I}$-status first. If b becomes a head, b's children can become its member without calling $\mathcal{I}$-status. In this way, we can reduce the completion time of join.

Now we show the detail of our algorithm for merging $CNet_{\mathcal{I}}(G_i)$ $(1 \le i \le m)$, called Procedure $\mathcal{I}$-merge. Let R be $CNet_{\mathcal{I}}(G_1)$ with rootID r and the joining node *new*. In $\mathcal{I}$-merge, first *new* determines its rootID. Then, each node b in $BT_{\mathcal{I}}(G_i)$ $(2 \le i \le m)$, change their status by $\mathcal{I}$-status(b, R) for R, and move into R one by one. R grows as nodes move into it, and we also denote the grown graph as R. After b in $BT_{\mathcal{I}}(G_i)$ call $\mathcal{I}$-status(b, R), (i) if b become heads, their member nodes in $CNet_{\mathcal{I}}(G_i)$ can remain as the children of them, (ii) otherwise, when b does not become a head, not all of the b's children can remain as its members. But, if there is a child m which is not adjacent to any head in R, all other children of b can remain b's members by $\mathcal{I}$-status(m, R) and changing b and m into heads. Therefore, we need to find such a child m which is not adjacent to a head in R by the procedure called deliver-member. The node b asks each child c one by one whether it is adjacent to a head in R. If c is adjacent to a head, c changes its parent to the adjacent head. Else c calls $\mathcal{I}$-status(c, R) as b is its parent (i.e., they become heads), and this procedure ends. If all of the children of b are adjacent to heads in R, b remains member.

In order to determine the status of *new*, *new* needs to know all of its neighbors. To do so, it is sufficient that the neighbors of *new* transmit their own IDs and status one by one. It can be done by numbering the neighbors of *new* from 1 to q and transmitting their information in order of the numbers.

Lemma 5.6. *The neighbors of new can be numbered from 1 to q in $O(q)$ expected rounds, where q is the number of neighbors of new in G'.*

Proof. By simulating the Initialization Protocol [19] on a complete graph in $O(1)$ rounds, it is possible to number the nodes from 1 to q on a star graph with *new* as a center node. (See APPENDIX B about how to simulate the protocol). □

Now, we present our join algorithm $\mathcal{I}$-join and subroutines used in $\mathcal{I}$-join, Procedure $\mathcal{I}$-status, $\mathcal{I}$-merge and deliver-member to organize the architecture $\mathcal{I}$, respectively.

Algorithm 2 $\mathcal{I}$-join(new, G)

```
1: The joining node $new$ sends $AddMe$ message;
2: The nodes receiving $AddMe$ message are numbered from 1 to $q$, and send their IDs
   and status to $new$ one by one;
3: if $new$ does not receives two or more rootIDs then
4:    $\mathcal{I}$-status($new, G$);
5: else
6:    $\mathcal{I}$-merge($new, G$);
7: end if
```

Procedure 3 $\mathcal{I}$-status(new, G)

```
1: if there are heads in neighbors of $new$ then
2:    $new$ sends I'mMember message to the neighboring head $h$ with minimum ID;
3:    $new.rootID := h.rootID$, $new.stat :=$ member,
      $new.prt := h$, $h.chd := h.chd \cup \{new\}$;
4: else {There are only members in neighbors of $new$}
5:    $new$ sends BeHead message to a neighboring member $m$ with minimum ID;
6:    $new.rootID := m.rootID$,
      $new.stat :=$ head, $m.stat :=$ head,
      $new.prt := m$, $m.chd := m.chd \cup \{new\}$;
7:    $m$ sends ChgHead message to its neighbors;
8:    Neighbors of $m$ change $m$'status into head in their information;
9: end if
```

Lemma 5.7. *Let $CNet_{\mathcal{I}}(G)$ be a cluster-based network of G. When G is organized with total 1-hop data, after an execution of $\mathcal{I}$-join for a node new, G' is organized with total 1-hop data.*

Proof. We show that the information of each node is maintained correctly and it satisfies the property of $BT_{\mathcal{I}}(G')$ and $CNet_{\mathcal{I}}(G')$ as follows: a set of heads is a dominating set of G, $BT_{\mathcal{I}}(G')$ is connected and any edge in $BT_{\mathcal{I}}(G')$ is formed between heads, and $CNet_{\mathcal{I}}(G')$ is rooted spanning tree of G'.

Procedure 4 $\mathcal{I}$-merge(new, G)

```
1: % Let $r$ be the minimum rootID in neighbors of $new$;
2: % Let $R = G[\{v|v \in CNet_{\mathcal{I}}(G_i)$ with rootID $r\} \cup \{new\} \cup \{v|v$ has called $\mathcal{I}$-status$\}]$;
3: % Let $t$ be a node with a token during $Eulerian$;
4: $new$ performs $\mathcal{I}$-status($new, R$) to the nodes with rootID $r$;
5: for each rootID $i$ of $new$'s neighbors do
6:   if there is no head in $new$'s neighbors with rootID $i$ then
7:     $new$ sends $DoJoin$ message to a neighboring member $u$ with rootID $i$ and minimum ID;
8:   else
9:     $new$ sends $DoJoin$ message to a neighboring head $u$ with rootID $i$ and minimum ID;
10:  end if
11:  $u$ call $\mathcal{I}$-status($new, R$);
12:  $u$ calls $Eulerian$ on the backbone tree with rootID $i$ which works as follows in each round:
13:    $t$ joins into $R$ by $\mathcal{I}$-status($new, R$);
14:    if $t$ becomes a member then
15:      deliver-member($t, G$);
16:    end if
17:    Nodes in $t.chd$ set their parent into $t$;
18: end for
```

A new node *new* will become a head or a member through $\mathcal{I}$-join. If there are heads in G as neighbors of *new*, *new* becomes a member of one of the heads h and $new.prt = h$, $h.chd = h.chd \cup \{new\}$. Else if there are no neighboring heads, *new* becomes a head. In this case, one of the members m in the neighbors of *new* becomes a head and a parent for head *new* and $m.chd = m.chd \cup \{new\}$. Whenever *new* becomes a head, it is adjacent to a head, and the head is neighbor of a head (which is parent of it) in $CNet_{\mathcal{I}}(G)$. So, the backbone tree $BT_{\mathcal{I}}(G')$ of G' is connected and any edge in $BT_{\mathcal{I}}(G')$ is formed by heads.

In each case, since only one edge and one node are added to existing cluster-based network $CNet_{\mathcal{I}}(G)$ which is spanning tree of G, the constructed new cluster-based network $CNet_{\mathcal{I}}(G')$ is also a spanning tree of G'. And *new* becomes a head or member as a child of a head, so a set of heads is a dominating set of G'. Finally, since all neighbors of *new* send their IDs and status, *new.oneigh* and their status are updated correctly. □

Theorem 5.8. *For disjoint graphs $G_1, G_2, \ldots, G_m$ and $CNet_{\mathcal{I}}(G_1)$, $CNet_{\mathcal{I}}(G_2)$, $\ldots$,*

Procedure 5 deliver-member(v, G)

```
1: $v.mlist := v.chd - v.bneigh$;
2: $v.chd := \emptyset$;
3: $v.stat :=$ member;
4: while $v.mlist \neq \emptyset$ and $v.stat =$ member do
5:    $v$ sends $ChkM$ message to a member $m \in v.mlist$;
6:    if there is no head in $m.neigh$ then
7:       $m$ calls $\mathcal{I}$-status as $v$ is its parent for $R$;
8:    else
9:       $m$ sends $GetNM$ message to one head $h \in m.neigh$;
10:      $h$ sends its neighboring members' IDs $NM$ to $v$ via $m$;
11:      $v$ sends $CoveredM$ message with $CM := v.mlist \cap NM$, and nodes in $CM$ set
         their parent to $h$;
12:      $m$ sends $CM$ to $h$, and $h.chd := h.chd \cup CM$;
13:      $v.mlist := v.mlist - CM$;
14:   end if
15: end while
16: $v.chd := v.mlist$;
17: Nodes in $v.chd$ set their parent into $v$;
```

$CNet_{\mathcal{I}}(G_m)$, when these graphs are organized with total 1-hop data, the operation join can be done in $O(q)$ expected rounds provided that no merge occurs or in $O(q + \max_{1\le i\le m} \{\sum_{v\in BT_{\mathcal{I}}(G_i)} \min(|N_m(v)|, |N_h^2(v)|)\}$ expected rounds when merge occurs, and G' is organized with total 1-hop data, where q is the number of neighbors of new in G', $N_m(v)$ is neighboring members of v, and $N_h^2(v)$ is heads in 2-hop distance of v.

Proof. In the join algorithm $\mathcal{I}$-join, the neighbors of *new* in G' can be numbered from 1 to q in expected $O(q)$ rounds by Lemma 5.6. After numbering of the neighbors, they send their IDs and status. It takes expected $O(q)$ rounds. If merge is caused, moreover, an *Eulerian* for each $CNet_{\mathcal{I}}(G_i)$ and *deliver-member* are performed. The *Eulerian*s in `FOR` loop (line 5) of *$\mathcal{I}$-merge* can be performed in parallel for each $CNet_{\mathcal{I}}(G_i)$ and *deliver-member* requires $\min(|N_m(v)|, |N_h^2(v)|)$ time. So *$\mathcal{I}$-merge* can be done in $O(\max_{1\le i\le m} \{\sum_{v\in BT_{\mathcal{I}}(G_i)} \min(|N_m(v)|, |N_h^2(v)|)\})$. □

Corollary 5.9. *For disjoint unit disk graphs G_1, G_2, …, G_m and $CNet_{\mathcal{I}}(G_1)$, $CNet_{\mathcal{I}}(G_2)$, …, $CNet_{\mathcal{I}}(G_m)$, when these graphs are organized with total 1-hop data, the operation join can be done in $O(q)$ expected rounds provided that no merge occurs or in*

$O(q + \max_{1 \le i \le m} \{|BT_{\mathcal{I}}(G_i)|\})$ expected rounds when merge occurs, and G' is organized with total 1-hop data, where q is the number of neighbors of new in G'.

Proof. The degree of the backbone tree is constant for an unit disk graph from Lemma 5.4 and the proof of Lemma 5.13 (described later). Hence this corollary can be derived easily from Theorem 5.8. □

Leave operation:

Next, we show our leave algorithm $\mathcal{I}$-leave. Let lev be a node who wants to leave from G and G' be the graph after lev leaves, that is, $G' = G[V - \{lev\}]$ in this subsection, where G' consists of disjoint graphs $G_1, G_2, \ldots, G_m$ (see Fig.5.4). We will show that we can judge whether G' is connected (i.e., $m = 1$) in $O(|T|)$ rounds, where $T = (V(T), E(T))$ is a subtree of $CNet_{\mathcal{I}}(G)$ with the leaving node lev as the root.

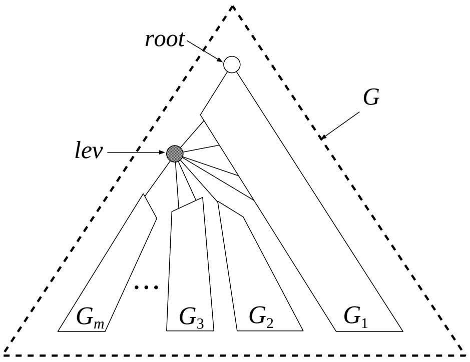

Figure 5.4: G and subgraphs separated by lev

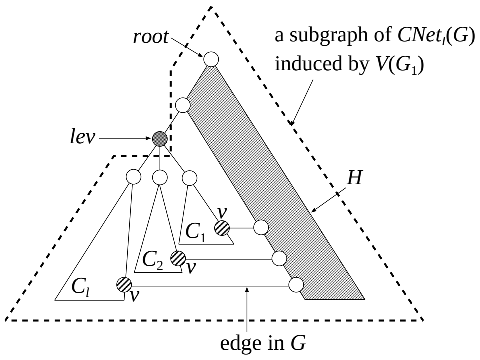

Figure 5.5: G_1 and its connected components

Our leave algorithm is executed when lev wishes to leave from the network. If lev is a member, it sends *I'mLeaving* message and simply leaves from the network. Otherwise, the leave algorithm works as follows: First, we consider the case where lev is not the root of $CNet_{\mathcal{I}}(G)$. The case where lev is the root is described later. If lev is a head, $CNet_{\mathcal{I}}(G)$ is divided into two subtrees. One is the tree T with lev as the root (not including the root in $CNet_{\mathcal{I}}(G)$), and one is the tree H with the root of $CNet_{\mathcal{I}}(G)$ as the root. The algorithm $\mathcal{I}$-leave removes *lev* from T, adds other nodes of T to H if they connect with H. Then the remaining nodes in $G_1, G_2, \ldots, G_m$ will be reconfigured to $CNet_{\mathcal{I}}(G_1)$, $CNet_{\mathcal{I}}(G_2), \ldots,$ $CNet_{\mathcal{I}}(G_m)$ by repeating $\mathcal{I}$-join one by one.

Let $C_i (i = 1, 2, \ldots, l)$ be the connected components of $G[V(T) - \{lev\}]$ (see Fig.5.5). H always changes and grows larger each time a node in T is added to H.

The edges in G between T and H are used in order to add the nodes of T to H. We make all of the nodes in T that have path to H join into H by $\mathcal{I}$-join. First, *lev* calls $Eulerian(T)$ to wake up each node of T. Whenever the waken node $v \in C_i$ has an edge connected with H, v moves to H by $\mathcal{I}$-join, then v calls $Eulerian(C_i)$ and each node in C_i moves to H following v by $\mathcal{I}$-join one by one. Each node already knows its neighbors in G and their status, therefore this $\mathcal{I}$-join can be performed deterministically in $O(1)$ rounds (just call of $\mathcal{I}$-status).

The above process will be repeated until all of *lev*'s children have received a token or moved to H.

After the process is finished, *lev* checks its children whether they move to H. If all of the children move to H, G is a connected graph after *lev* leaves, then the leave operation is completed. Else G is disconnected. Node *lev* sends message to its each child in G_j which is not joined to H. The child becomes a new root of $CNet_{\mathcal{I}}(G_j)$ and all nodes in G_j repeat $\mathcal{I}$-join to construct $CNet_{\mathcal{I}}(G_j)$.

Here we describe about an exception, when *lev* is a root of $CNet_{\mathcal{I}}(G)$. If $lev.bneigh \neq \emptyset$, electing a head which is 2-hop neighbor of *lev* and setting it to a new root of $CNet_{\mathcal{I}}(G)$, which is shown in *change-root*, our algorithm in the general case can be used. Otherwise *lev* selects one node in *lev.neigh* becomes the new root of $CNet_{\mathcal{I}}(G')$. The new root calls $Eulerian(G[V - \{lev\}])$ and a cluster-based network is constructed sequentially by repeating $\mathcal{I}$-join for the node with token.

Our leave algorithm $\mathcal{I}$-leave is described in Algorithm $\mathcal{I}$-leave and Procedure subroutines of $\mathcal{I}$-leave.

The following lemma can be easily derived.

Lemma 5.10. *Let $CNet_{\mathcal{I}}(G)$ be a cluster-based network of G. When G is organized with total 1-hop data, after an execution of $\mathcal{I}$-leave for a node lev, G' is organized with total 1-hop data.*

Algorithm 6 $\mathcal{I}$-leave

```
% Let $T = (V(T), E(T))$ be a subtree of $CNet_{\mathcal{I}}(G)$ with root $lev$;
% Let $C_i (i = 1, 2, ...)$ be the connected components of $G[V(T) - \{lev\}]$;
% Let $H = G[V - V(T) \cup \{v | v \in V(T), v$ has called $\mathcal{I}$-join$\}]$;
% Let $t$ be a node with a token in $Eulerian$;
  for each $v \in G$ do
    $v.link := v.neigh$;
  end for
  $lev$ sends I'mLeaving message;
  if $lev.stat$ = member then
    nodes that received I'mLeaving delete $lev$ from neighbor list in $G$;
  else
    if $lev$ is a root of $CNet_{\mathcal{I}}(G)$ then
      if $lev.bneigh \neq \emptyset$ then
        $lev$ sends chkchild message;
        Each node $v \in lev.bneigh$ sends $v.child$ to $lev$ one by one;
      end if
      if There is a head 2-hop away from $lev$ then
        change-root;
      else
        exception; exit;
      end if
    end if
    if $lev$ is a head and $|(lev.prt).bneigh| = 2$ then
      $(lev.prt).chd := (lev.prt).chd - lev$;
      deliver-member(lev.prt);
      Nodes in $(lev.prt).chd$ set their parent into $lev.prt$;
    end if
    % Let $T' := T$;
    $lev$ calls $Eulerian(T)$ which works as follows in each round:
       $v.link := v.link - \{t\}$ for each node $v$ who receives messages from $t$;
    while there is a node in $lev.chd$ who has not received a token and not joined to $H$ do
      $lev$ calls $Eulerian(T')$, and it works as follows in each round:
         if $t.link \neq \emptyset$ then the procedure finishes;
      $t \in C_j$ calls $Eulerian(C_j)$, and it works as follows in each round:
         $t$ joins into $H$ by $\mathcal{I}$-join except line 1,2, and each neighbor $v$ of $t$ adds $t$ to $v.link$;
      $t$ sends the token back to $lev$ by $Eulerian(T')$;
      nodes who have joined to $H$ are removed from $T'$;
    end while
    while there is a $lev$'s child $v \in C_i$ who has not determined its status do {separation}
      $lev$ sends a message to $v$;
      $v$ makes $CNet_{\mathcal{I}}(C_i)$ with root $v$ by $Eulerian(C_i)$ performing $\mathcal{I}$-join one by one;
    end while
  end if
```

Procedure 7 subroutines of $\mathcal{I}$-leave

```
exception
1: $lev$ sends a message to one of its neighbors $r$, and $r$ becomes the root and has a
   token ($t := r$);
2: $r$ sends $I'mRoot$ message and $v.link := v.link \cup \{r\}$ for each neighbor $v$ of $r$;
3: % Let $G'' := G[\{r\}]$;
4: $t$ calls $Eulerian(G[G(V)-\{lev\}])$ which works as follows in each round:
5:     if $t$ has not joined then
6:         $t$ joins into $G''$ according to the status of nodes in $t.link$ by $\mathcal{I}$-join;
7:         Each neighbor $v'$ of $t$ adds $t$ to $v'.link$;
8:     end if

change-root
1: $lev$ sends a message to a head $h$ in $lev.bneigh$;
2: $lev.prt := h$, $h.chd := h.chd \cup \{lev\}$;
3: $h$ sends a message to a head $h'$ in $h.bneigh$;
4: $h.prt := h'$, $h'.chd := h'.chd \cup \{h\}$;
5: $h'.prt := \perp$ and $h'$ becomes a root;
```

Proof. When lev is a member, only node lev and the edge between lev and $lev.prt$ are removed from $CNet_{\mathcal{I}}(G)$. Therefore, $BT_{\mathcal{I}}(G')$=$BT_{\mathcal{I}}(G)$, the status of each node is not changed, and $CNet_{\mathcal{I}}(G')$ is also rooted spanning tree. The information of each node is maintained by removing lev from it.

Now we consider the case that lev is a gateway node or a head. Let $T = (V(T), E(T))$ be a subtree of $CNet_{\mathcal{I}}(G)$ with root lev and $H = (V(H), E(H)) = G[V - V(T) \cup \{v|v \in V(T), v \text{ has called } \mathcal{I}\text{-join}\}]$. Since an edge between $V(T)$ and $V(H)$ is only (lev, $lev.prt$) in $CNet_{\mathcal{I}}(G)$, just $lev.prt$ updates its information about $stat$, prt, chd, and each neighbor v of lev removes lev from $v.oneigh$. G is organized from the assumption, so H is also organized. *Eulerian* procedure calls determines the order of $\mathcal{I}$-join for each node in $V(T)$, a node v, s.t. $v \in u.neigh$, $u \in V(H)$, joins to H by $\mathcal{I}$-join. So, from Lemma 5.7, if H is organized, $G[V(H) \cup v]$ is organized. Since this is performed repeatedly for all $v \in V(T)$, $H = G[V - \{lev\}]$ is organized after an execution of $\mathcal{I}$-join for a node lev.

When lev is the root of $CNet_{\mathcal{I}}(G)$, the root is changed and $\mathcal{I}$-join is performed in order similarly. □

Theorem 5.11. *Let $CNet_{\mathcal{I}}(G)$ be a cluster-based network of G and T be the subtree of $CNet_{\mathcal{I}}(G)$ with the leaving node lev as root. When G is organized with total 1-hop data,* *leave of lev can be done in $O(|T|)$ rounds, and G' is organized with total 1-hop data.*

Proof. lev calls $Eulerian(T)$ so that each node knows its neighbors in T. Next, $Eulerian(G[V - V(H) - \{lev\}])$ is called to find the edge (u, v), $u \notin V(H)$, $v \in V(H)$, and to make each node in T join to H. These calls of $Eulerian$ takes $O(|T|)$ rounds.

When lev is a root in $CNet_{\mathcal{I}}(G)$ and $lev.bneigh = \emptyset$, lev calls $Eulerian(T)$ once and each node joins in some round during the procedure. It takes $O(|T|)$ rounds. Otherwise, if lev is a root in $CNet_{\mathcal{I}}(G)$, replacing the role of lev by other head can be done in $O(1)$ rounds. Then each node in T joins to H as already mentioned above.

So, a leave operation can be done in $O(|T|)$ rounds. □

5.4.3 Properties of Architecture $\mathcal{I}$

On $\mathcal{I}$, a join/leave operation which enables merge/separation for ad-hoc network(s) can be performed with a backbone tree which has the same size as $\mathcal{C}$. Moreover, the nodes of $\mathcal{I}$ has only two kinds of status as member or head and the backbone tree is simplified.

The architecture $\mathcal{I}$ constructed with above two operations has the following property.

Lemma 5.12. *The number of clusters in $CNet_{\mathcal{I}}(G)$ is equal to $|BT_{\mathcal{I}}(G)|$.*

Proof. Since $BT_{\mathcal{I}}(G)$ consists only of heads on $\mathcal{I}$, the number of clusters in $CNet_{\mathcal{I}}(G)$ is $|BT_{\mathcal{I}}(G)|$. □

Lemma 5.13. *Let G be a connected bidirectional graph and p_G be the cardinality of minimum clique partition of G. $|BT_{\mathcal{I}}(G)|$ is at most $2p_G - 1$ on $\mathcal{I}$ which is constructed by $\mathcal{I}$-join/leave.*

Proof. We consider the process of forming $BT_{\mathcal{I}}(G)$ by $\mathcal{I}$-join. For convenience sake, we divide the heads in $BT_{\mathcal{I}}(G)$ into H_1 and H_2, and H_1 contains one node in the initial state (H_1 and H_2 represent not only the status but the set of nodes whose status are H_1 and H_2 unless confusion is caused, respectively). This is not real status and can be used only for explanation. Then we show that the following properties hold:

(i) $|H_2| < |H_1|$,

(ii) nodes in H_1 are not adjacent each other.

In the initial state, (i) and (ii) hold since there is only one node which is H_1.

First, we consider $\mathcal{I}$-join in the case that merge does not happen. From Procedure $\mathcal{I}$-status, we show the following virtual $\mathcal{I}$-status: (a) if there are neighbors of *new* in H_1 and H_2, *new* becomes a member; (b) else change a status of one of them into H_2, and *new* becomes H_1. Thus, nodes in H_1 are not adjacent each other. $|H_1|$ and $|H_2|$ increase in the same number respectively, so $|H_2| < |H_1|$. When merge of cluster-based networks occurs, we just control the order of joining of each nodes. Therefore above argument in the case that merge does not happen can hold.

Next, we consider $\mathcal{I}$-leave. When *lev* is a member, H_1 and H_2 do not change. When *lev* is H_1, the parent, which is H_2, of *lev* becomes a member, or one H_1 node appears except *lev* by *deliver-member*. In the case that the parent becomes a member, $|H_1|$ and $|H_2|$ are decrease by one, so (i) and (ii) hold. Another case that an H_1 node appears except *lev*, since *lev* leaves and a new H_1 node appears, $|H_1|$ and $|H_2|$ are unchanged and (i) is satisfied. Then a new H_1 node is not adjacent to other head except for the parent, which is H_2. Therefore (ii) is also satisfied. We can apply the same argument for the proof when *lev* is H_2.

Hence, (i) and (ii) are always satisfied. Since $|H_1|$ is at most the cardinality of minimum clique partition of G from Lemma 5.2, $|BT_{\mathcal{I}}(G)| < 2p_G$. □

Lemma 5.14. *Let $G = (V, E)$ be a unit disk graph, and $MDS(G)$ be the minimum dominating set of G. $|BT_{\mathcal{I}}(G)|$ is not larger than $10 \times |MDS(G)|$.*

Proof. This lemma can be derived easily from Lemma 5.3 and 5.13. □

The properties shown in lemmas 5.13, 5.14 are not derived from the definition of structure of $\mathcal{I}$, but derived from the algorithm (Algorithm $\mathcal{I}$-join and $\mathcal{I}$-leave).

5.4.4 Improvement of $\mathcal{I}$

Here we show some improvement in order to reduce the size of backbone tree on $\mathcal{I}$ while preserving the completion time of the join/leave operations.

On the original architecture $\mathcal{I}$, *new* always joins as a head, however, it can be member if its parent changes the status to head even if there is no head in the neighbors of *new*.

In this case, the size of backbone tree can be reduced. To make theoretical analysis easy, we divide status member into status member1 and status member2. The nodes in status member1 are not adjacent to each other in G. Then the size of a backbone tree can be analyzed in a similar way as the proof of Lemma 5.13. A member1 node is candidate of H_1 in the proof which becomes a head next time. Thereby we can obtain the same property about the size of a backbone tree as Lemma 5.13. Moreover $|BT_{\mathcal{I}}(G)|$ can be decreased by the number of members in status member1. We will give a simulation result for the size reduction caused by the improved architecture.

Structure: We divide members into two classes, member1 and member2, on the improved architecture. Other structure is the same as original architecture $\mathcal{I}$. Here member1 nodes are not adjacent each other.

Algorithms: First we show the join algorithm for improved $\mathcal{I}$.

The joining node *new* becomes a head in $\mathcal{I}$-status if there is no head in its neighbor. Then in the improved algorithm, *new* always becomes a member and we divide the member into member1 and member2 so that member1 nodes are not adjacent each other.

We extract only the part relative to a determination of status in Procedure $\mathcal{I}$-status(improved). Most part of the other algorithm and procedure of improved $\mathcal{I}$ are not changed.

Procedure 8 $\mathcal{I}$-status(improved)

```
1: if there are heads in neighbors of $new$ then
2:     % Let $h$ be a head with minimum ID in neighbors of $new$;
3:     $new.stat$ := member2, $new.prt := h$, $h.chd := h.chd \cup \{new\}$;
4: else if there are member1s in neighbors of $new$ then
5:     % Let $m_1$ be a member1 with minimum ID in neighbors of $new$;
6:     $new.stat$ := member2, $m_1.stat$ := head;
7:     $new.prt := m_1$, $m_1.chd := m_1.chd \cup \{new\}$;
8: else {only member2}
9:     % Let $m_2$ be a member2 with minimum ID in neighbors of $new$;
10:    $new.stat$ := member1, $m_2.stat$ := head;
11:    $new.prt := m_2$, $m_2.chd := m_2.chd \cup \{new\}$;
12: end if
```

Our leave algorithm is basically the same as Algorithm $\mathcal{I}$-leave. Only in the case that the leaving node *lev* is a root or member1, the algorithm differs slightly. If *lev* is member1, *lev* changes its status into head, and performs $\mathcal{I}$-leave. Else if *lev* is a root, it searches a head which is away from *lev* by 2-hop. If such a head does not exist, it searches a member1 which is away from *lev* by 2-hop, changes its status into head, and performs $\mathcal{I}$-leave. Else it simply performs $\mathcal{I}$-leave.

The completion times and properties of these algorithms remain the same.

Improved architecture $\mathcal{I}$ which is constructed with above two operations has the same property as $\mathcal{I}$ we showed before. We also show that the size of a backbone tree on our architecture is reduced by a simulation in the next section.

5.5 Simulation

We compare the sizes of backbone trees for the three architectures $\mathcal{M}$, $\mathcal{I}$ and improved $\mathcal{I}$ and its algorithms by simulation. The setting of the simulation is as follows: Each node is treated as a point without volume; The field where nodes are deployed is infinite plane; Nodes are added to the field from the initial state with one node until the number of nodes reach n; Each node is set randomly within a range where existing nodes can transmit.

Under above setting, we measure the size of backbone tree from $n = 1000$ to 8000 by 1000 nodes (Fig.5.6). Each plot point represents a average value for 100 trials.

The size of backbone tree of $\mathcal{I}$ is less than that of the other architectures within the limit of $n = 1000, \ldots, 8000$, and it is expected that $\mathcal{I}$ is also superior to the others for $n \gg 8000$ from Fig.5.6.

We can consider that the size of backbone tree of $\mathcal{I}$ is less than that of $\mathcal{M}$ for the reason that the number of nodes which become members increases by constructing backbone tree with only heads, i.e. the nodes that are able to have children. And as another reason, it is also considered that improved $\mathcal{I}$ can greatly decrease the number of heads which exist in a part of the outer of the region where nodes are deployed actually owing to member1.

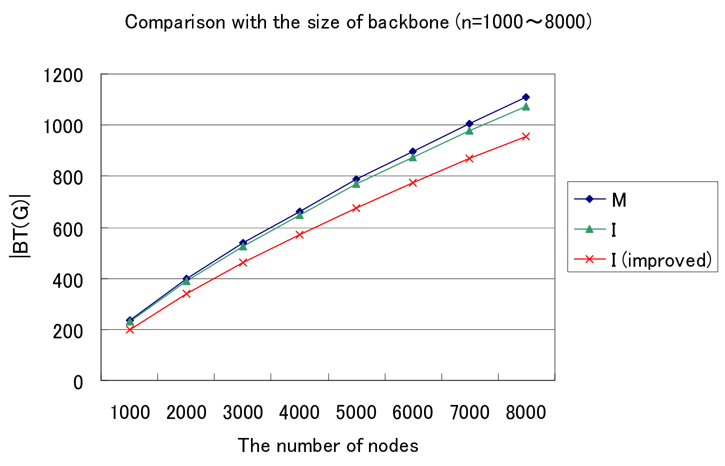

Figure 5.6: Comparison with the size of backbone tree

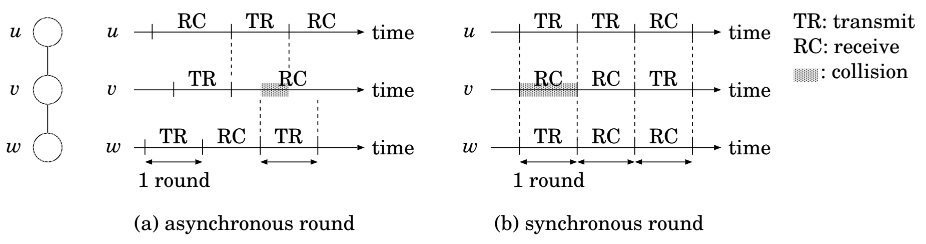

Figure 5.7: Synchronous and asynchronous rounds

5.6 Conversion of Synchronization into Asynchronization

In this section, we describe how our algorithms can work in asynchronous rounds. The algorithms proposed in this dissertation look like synchronized ones; however, essentially they have the properties that asynchronized algorithms have, in other words, they can run in an asynchronized fashion by introducing a delay D.

Now we consider the following asynchronous rounds:

- Nodes repeat transmissions and receptions.
- Each node transmits in fixed intervals, called *rounds*, otherwise acts as receiver.
- Node u acting as a receiver gets a message which neighbor v sends in a given round r iff no neighbor of u except for v transmits in a round which is overlapped with r. When more than one neighbor transmits simultaneously in rounds which are overlapped with r, collision occurs at u and none of the messages is received in r (see an example of three nodes in Fig.5.7(a)).

Algorithms in synchronous system do not work correctly in asynchronous system directly. Two or more messages that are transmitted in different rounds in synchronous system may cause a collision in asynchronous system even if these messages do not cause a collision in synchronous system. We should prevent them from colliding.

We let the interval of rounds for every node be fixed. However, we allow each node starts the interval at any time. When a node act as transmitter, it can keep sending a message during the interval. When two or more intervals of transmitting nodes overlap, collision occurs. We allow a delay of receiving a message. We take a delay D into account and it is much shorter than the interval of rounds, say, less than the half of the interval.

Now we consider the case that node v sends a massage and neighbor(s) of v sends back a reply to v. Let t_0 be the time when v has finished sending a message, then the time when v finishes receiving the replies of its neighbors is at most $t_0 + 2D + I$, where I is the interval of one round. Since $D < \frac{1}{2}I$, $t_0 + 2D + I < t_0 + 2I$.

If two or more replies are sent to v, a collision occur at v during $[t_0 \dots t_0 + 2I)$ because it takes two or more rounds to receive these messages, then v can not receive the replies. Else no collision occurs.

Therefore, in order to simulate a round in synchronous system, "node v enters receiving state and waits for replies during two rounds after it sends a message, then v does its next action". By doing so, in asynchronous system, it is possible for a node which sends message and waits for its replies (if any) to simulate the same situation of collision/non-collision in synchronous system. If a transmitting node v receives a message in the following two rounds in asynchronous system, v must receive a message in synchronous system, and if collision occurs at v for these two rounds in asynchronous system, it must occur at v in synchronous system.

Now we mention the application to our proposed algorithms: (i) for *broadcast*, it works for above asynchronous system since at most one node repeats "if receives token, then sends it to a neighbor" by *Eulerian* in each round and there is no collision in synchronous system. (ii) for *move-in*, at first the neighbors of *new* are numbered. In the algorithm *new* and its neighbors send message in turn after waiting replies, so the above discussion can be applied. After numbering of neighbors, no collision occurs since only *new* and its parent send messages in two rounds. (iii) for *move-out*, the number of transmitting nodes is at most one and it repeats "if receives token, then sends it to a neighbor" as well as broadcast algorithm. So it also works in asynchronous system.

For above discussion, each node takes only one more round by waiting replies for each round. Therefore, the order of completion times of our algorithms does not change due to the conversion into asynchronous system.

Chapter 6

Conclusions and Future Works

In this dissertation, we consider the ARB, the ARG, and the clustering algorithms on the model of ad hoc radio networks without collision detection.

We show that we can construct deterministic and distributed ARB algorithms for bidirectional graphs in time $O(n)$, and for strongly connected graphs in time $O(6n + \sum_{i=1}^{\lceil \log n \rceil}\{2 \cdot RB(2^i) + RG(2^i)\})$, where n is the number of the nodes in the graphs and $n \geq 2$. We also show that our each ARB algorithm can be extended to ARG algorithm which completes ARG in time $O(n + \sum_{i=1}^{\lceil \log n \rceil}\{LE(2^i)\})$ for bidirectional graphs and in time $O(6n + \sum_{i=1}^{\lceil \log n \rceil}\{RB(2^i) + 2 \cdot RG(2^i)\})$ for strongly connected graphs.

Our algorithms can be improved if we can find more efficient leader election algorithms for bidirectional graphs and if ARB can be achieved without using RG for strongly connected graphs. The leader election algorithm FIND MAX uses broadcast algorithm as its subroutine, and we use gossiping algorithm to collect the *warned* information of all nodes for strongly connected graphs. So, we would like to find out if leader election may be done faster than using broadcast algorithm, and gathering may be done faster than gossip.

We also have proposed a novel cluster-based architecture $CNet(G)$ for dynamic ad hoc radio networks G, in which broadcasting can be done in $O(p_G)$ rounds, where p_G is the cardinality of the minimum clique partition of G, and so when G is a dense graph, $p_G \ll n$. In order to support dynamic changes of the architecture we have used two operations join and leave, and proposed some algorithms for them.

In future work, we will concentrate on the following aspects. First, we plan to improve the time complexity for a join and a leave operations. Second, we plan to deal with data gathering and routing problems on this architecture. Third, we plan to propose new architectures with better properties than that of the architecture $CNet(G)$ in this dissertation.

Moreover, for the sake of better theoretical model of an ad-hoc network, we must consider the fault-tolerance and self-stability. Dealing with fault is necessary because of the instability of node itself and the communication via radio. We recognize that the achievement of the fault tolerance is an important point as our development in the future. The self-stabilization is considered as a promising paradigm about that. The most important point for the self-stabilization is to get rid of the assumption that the node joins one by one from an initial state (one node), which is our present model. It is necessary to consider clustering from an arbitrary situation.

Finally, we should consider not only the problems on this network communication model but also the validity of the model itself in order to bring it close to reality, and this is also our future work.

Appendix A

Algorithm bi-ARB for Each Node

Here we show the pseudocode of our algorithm bi-ARB in Figure A.1. Each node v executes the pseudocode, where receive(R) is the procedure which tries to receive a message, denoted as R. It returns "true" if received a message, or "false" if not. Since the goal of ARB is to achieve RB and inform the source about the completion of RB, only the source node terminates the pseudocode in Figure A.1. But, we can easily modify the pseudocode for each node to terminate it by repeating one more phase. It is enough that the source node informs every node about the completion of RB in an additional phase.

```
var $N_v^k$      : set of integers   **init** $\emptyset$;
    $Q_v$        : set of integers   **init** $\emptyset$;
    $Min_v$      : set of integers   **init** $\emptyset$;
    $v.id$       : integer           **init** ID of node $v$ itself;
    $first$      : integer           **init** $-1$;
    $i, k$       : integer;

begin
  { Phase 0: }
  if $v.id$=1 then send <ID(v)>
  else if receive($R$) then $N_v^k$ := {sender's ID of $R$};
```

```
$k := 1$
{ Phase 1, . . . : }
**repeat**
    { Stage A: }
    **for** $i := 1$ to $2^{k-1}$ **do**
        **if** $v.id = 2^{k-1} + i$ **then** send <ID(v)>
        **else if** receive($R$) **then**
            $N_v^k := N_v^k \cup \{$sender's ID of $R\}$;
    { Stage B: }
    **for** $i := 1$ to $2^k$ **do**
        **if** $v.id = i$ **then** send a message to $\min(N_v^k)$
        **else if** receive($R$) **then**
            **if** $R$ is the message to $v$ **then**
                $Min_v := Min_v \cup \{$sender's ID of $R\}$;
    { Stage C: }
    **for** $i := 1$ to $2^k$ **do**
        **if** $v.id = i$ **then**
            **if** receive($R$)=false **then** become *warned*
        **else if** $i \in Min_v$ **then** send <ID(v)>
        **else if** $v.id \geq 2^k$ **then** send <ID(v)>
        **else** receive($R$);
    { Stage D: }
    $Q_v := N_v^k$; $i := 0$;
    **if** $v$ is the source **then begin**
        send <ID(v), visited>; $i := i + 1$
    **end**
    **while** $i < 2^{k+1} - 2$ **do begin**
        **if** receive($R$) **then begin**
            $Q_v := Q_v - \{$sender's ID of $R\}$;
            **if** $R$ is a token to $v$ **then begin**
                **if** *first* $= -1$ **then** *first* := sender's ID of $R$;
                **if** $R$ contains *warning* message **then**
                    become *warned*;
                **if** $Q_v = \emptyset$ **then**
                    send <ID(v), visited> to its neighbors and a token
                    to the node *first* (append *warning* message if *warned*);
                **else**
                    send a token to the node with the smallest ID in $Q_v$;
                $i := i + 1$
            **end**
        **end**;
        $i := i + 1$
    **end**;
    $k := k + 1$
  **until** $v$ is the source & $v$ is not *warned*;
**end**.
```

Figure A.1: pseudocode of bi-ARB

Appendix B

Simulating the Initialization Protocol

For a given n-node network, Initialization is to give a unique ID ranging from 1 to n for a set of nodes.

We simulate the Initialization Protocol called *Protocol for Unknown n: The No-Collision Detection Case* in [19]. In the process of this Initialization protocol, every node can send its ID without collision.

The network in [19], denoted by *IN*, differs from our model in the following:

- *IN* is single-hop (*IN* is an n-node complete graph).
- Each node can receive its own message (if no collision).
- Each node has no ID.

Suppose that the nodes in a subset P transmit. Each node v in *IN* gets in each round as follows:

- $|P| = 1$ (including the case that $v \in P$).
- $|P| = 0$ or $|P| \geq 2$.

that is, (i) only v transmits or (ii) other node transmits or (iii) two or more nodes transmits or no node transmit.

Our network can simulate one round of the *IN* in two rounds. In other words, it can determine whether "$|P| = 1$ (even whether $v \in P$ or not)" or "$|P| = 0$ or $|P| \geq 2$". Let *new* be a joining node. Each round in *IN* is simulated as follows (Fig B.1):

```
simulate IN
  1. $v \in P$ sends $v.id$;
  2. if new receive ID $i$ in previous round then
         new sends $i$;
     else
         new waits;
```

Figure B.1: simulation of *IN*

Each node v can determine whether "$|P| = 1$ (including the case that $v \in P$)" or "$|P| = 0$ or $|P| \geq 2$" according to the message received in the second step in Fig B.1.

Case1: v received its own ID. Then, $|P|=1$ and $v \in P$.

Case2: v received other's ID. Then, $|P|=1$ and $v \notin P$.

Case3: v received no message. Then, $|P|=0$ or $|P| \geq 2$.

Thus, it is sufficient to simulate *IN* in two rounds.

Bibliography

[1] N. Alon, A. Bar-Noy, N. Linial, and D. Peleg. A lower bound for radio broadcast. *Journal of Computer and System Sciences 43*, pages 290–298, 1991.

[2] R. Bar-Yehuda, O. Goldreich, and A. Itai, On the time-complexity of broadcast in radio networks: an exponential gap between determinism and randomization, *Journal of Computer and System Science*, no. 45, pages 104–126, 1992.

[3] S. Basagni, Distributed clustering for ad hoc networks, *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms, and Network*, pages 310–315, 1999.

[4] S. Basagni, M. Mastrogiovanni, C. Petrioli, A performance comparison of protocols for clustering and backbone formation in large scale ad hoc networks, *The 1st Internatinal Conderence on Mobile Ad-hoc and Sensor Systems*, pages 70-79, 2005.

[5] D. Brusci and M. Del Pinto. Lower bounds for the broadcast problem in mobile radio networks. *Distributed Computing 10*, pages 129–135, 1997.

[6] I. Chlamtac, A. Farago. , A new approach to the design and analysis of peer-to-peer mobile networks, *Wireless Networks*, vol. 5, no. 3, pp. 149–156, 1999.

[7] B. S. Chlebus, L. Gąsieniec, A. M. Gibbons, A. Pelc, and W. Rytter, Deterministic broadcasting in ad hoc radio networks, *Distributed Computing 15*, pages 27–38, 2002.

[8] M. Chrobak, L. Gąsieniec, and W. Rytter. Fast broadcasting and gossiping in radio networks. *Journal of Algorithms, Volume 43, Issue 2 (May 2002)*, pages 177–189, 2002.

[9] A. Czumaj and W. Rytter. Broadcasting Algorithms in Radio Networks with Unknown Topology. *in Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS'03)*, pages 492–501, 2003.

[10] D. Dubhashi, A . Mei, A. Panconesi, J. Radhakrishnan, A. Srinivasan, Fast distributed algorithms for (weakly) connected dominating sets and linear-size skeletons, *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 717–724, 2003.

[11] L. Gąsieniec, M. Christersson and A. Lingas. Gossiping with bounded size messages in ad hoc radio networks. *29th International Colloquium on Automata, Languages and Programming, (ICALP'02)*, pages 377–389, 2002.

[12] L. Gąsieniec, T. Radzik, Q. Xin. Faster Deterministic Gossiping in Directed Ad Hoc Radio Networks. *In Proc. of 9th Scandinavian Workshop on Algorithm Theory (SWAT'2004)*, pages 397–407, 2004.

[13] M. Goldberg, Packing of 14, 16, 17 and 20 Circles in a Circle, *Mathematics Magazine*, Vol. 44, No. 3 (May, 1971), pages 134–139, 1971.

[14] D. R. Kowalski and A. Pelc. Time of radio broadcasting: adaptiveness vs. obliviousness and randomization vs. determinism. *in Proc. , 10-th International Colloquium on Structural Information and Communication Complexity, SIROCCO 2003*, pages 195–210, 2003.

[15] D. R. Kowalski and A. Pelc, Optimal deterministic broadcasting in known topology radio networks, *Distributed Computing*, vol.19 no.3 (2007), pages 185–195, 2007.

[16] F. Kuhn, T. Moscibroda, T. Wattenhofer, Initializing Newly Deployed Ad Hoc and Sensor Networks, *in Proceedings of 10 Annual International Conference on Mobile Computing and Networking (MOBICOM)*, 2004.

[17] E. Kushilevitz and Y. Mansour. An $\Omega\left(D \log \frac{n}{D}\right)$ lower bound for broadcast in radio networks. *SIAM Journal on Computing, Volume 27, Issue 3 (June 1998)*, pages 702–712, 1998.

[18] S. Miyanaga, Y. Katayama, K. Wada, N. Takahashi, M. Kobayashi, and M. Morita, Efficient Clustering Algorithms for Dynamic Wireless Ad-hoc Networks with Considering Mergence and Partition of Clusters, *to appear in IEICE transactions on Information and Systems.*

[19] K. Nakano and S. Olariu. Randomized initialization protocols for radio networks. *Handbook of wireless networks and mobile computing*, pp. 195–218, 2002.

[20] T. Okuwa, W. Chen and K. Wada. An optimal algorithm of acknowledged broadcasting in ad hoc networks. *Proc. of 2nd Int'l Symp. Parallel and Distributed Computing(2003)*, pages 178–184, 2003.

[21] P.-J. Wan, K. M. Alzoubi, and O. Frieder, Distributed construction of connected dominating sets in wireless ad hoc networks, *ACM/Kluwer Mobile Networks and Applications*, MONET, bol. 9, no. 2, pp. 141–149, 2004.

[22] J. Wu and H. Li, On calculating connected dominating set for efficient routing in ad hoc wireless networks, *Telecommunication Systems*, vol. 18, no. 1/3, pp. 13–36, 2001.

Publications

1. Jiro Uchida, Wei Chen, and Koichi Wada, "Acknowledged Broadcasting and Gossiping in Ad Hoc Radio Networks", *7th International Conference on Principles of Distributed Systems (OPODIS)*, pp.223–234, December. 2003.

2. Jiro Uchida, Wei Chen, and Koichi Wada, "Acknowledged broadcasting and gossiping in ad hoc radio networks", *Theoretical Computer Science*, Volume 377, Issues 1-3, pp.43–54, May. 2007.

3. Jiro Uchida, Muzahidul A.K.M. Islam, Yoshiaki Katayama, Wei Chen, and Koichi Wada, "Construction and Maintenance of a Cluster-Based Architecture for Sensor Networks", *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, Volume 09, p. 237.3 (10 pages), January 2006.

4. Jiro Uchida, Muzahidul A.K.M. Islam, Yoshiaki Katayama, Wei Chen, and Koichi Wada, "Construction and maintenance of a novel cluster-based architecture for ad hoc sensor networks", *to appear in the Journal of Ad Hoc & Sensor Wireless Networks*, 2007.