T

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Notice to Users

Notice aux utilisateurs

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Self Stabilization in Dynamic Distributed Systems

by

Rick Youping Huang, B.Sc.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute of Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

April 30, 1993

© copyright

1993, Rick Youping Huang

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Author's Acknowledgement

Author's Acknowledgement

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-84100-1

Canada

Name: ROCK HUANG Thesis Title: Self Stabilization of Dynamic Distributed Systems
 Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Computer Science

0984

U-M-I

SUBJECT TERM

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
 Art History 0377
 Cinema 0900
 Dance 0378
 Fine Arts 0357
 Information Science 0723
 Journalism 0391
 Library Science 0199
 Mass Communications 0708
 Music 0413
 Speech Communication 0459
 Theater 0465

EDUCATION

General 0515
 Administration 0514
 Adult and Continuing 0516
 Agricultural 0517
 Art 0273
 Bilingual and Multicultural 0282
 Business 0688
 Community College 0275
 Curriculum and Instruction 0727
 Early Childhood 0518
 Elementary 0524
 Finance 0277
 Guidance and Counseling 0519
 Health 0680
 Higher 0745
 History of 0520
 Home Economics 0278
 Industrial 0521
 Language and Literature 0279
 Mathematics 0280
 Music 0522
 Philosophy of 0998
 Physical 0523

Psychology 0525
 Reading 0535
 Religious 0527
 Sciences 0714
 Secondary 0533
 Social Sciences 0534
 Sociology of 0340
 Special 0529
 Teacher Training 0530
 Technology 0710
 Tests and Measurements 0288
 Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language
 General 0679
 Ancient 0289
 Linguistics 0290
 Modern 0291
 Literature
 General 0401
 Classical 0294
 Comparative 0295
 Medieval 0297
 Modern 0298
 African 0316
 American 0591
 Asian 0305
 Canadian (English) 0352
 Canadian (French) 0355
 English 0593
 German 0711
 Latin American 0312
 Middle Eastern 0315
 Romance 0313
 Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
 Religion
 General 0318
 Biblical Studies 0321
 Clergy 0319
 History of 0320
 Philosophy of 0322
 Theology 0469

SOCIAL SCIENCES

American Studies 0323
 Anthropology
 Archaeology 0324
 Cultural 0326
 Physical 0327
 Business Administration
 General 0310
 Accounting 0272
 Banking 0770
 Management 0454
 Marketing 0338
 Canadian Studies 0385
 Economics
 General 0501
 Agricultural 0503
 Commerce Business 0505
 Finance 0508
 History 0509
 Labor 0510
 Theory 0511
 Folklore 0358
 Geography 0366
 Gerontology 0351
 History
 General 0578

Ancient 0579
 Medieval 0581
 Modern 0582
 Black 0328
 African 0331
 Asia, Australia and Oceania 0332
 Canadian 0334
 European 0335
 Latin American 0336
 Middle Eastern 0333
 United States 0337
 History of Science 0585
 Law 0398
 Political Science
 General 0615
 International Law and Relations 0616
 Public Administration 0617
 Recreation 0814
 Social Work 0452
 Sociology
 General 0626
 Criminology and Penology 0627
 Demography 0938
 Ethnic and Racial Studies 0631
 Individual and Family Studies 0628
 Industrial and Labor Relations 0629
 Public and Social Welfare 0630
 Social Structure and Development 0700
 Theory and Methods 0344
 Transportation 0709
 Urban and Regional Planning 0999
 Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture
 General 0473
 Agronomy 0285
 Animal Culture and Nutrition 0475
 Animal Pathology 0476
 Food Science and Technology 0359
 Forestry and Wildlife 0478
 Plant Culture 0479
 Plant Pathology 0480
 Plant Physiology 0817
 Range Management 0777
 Wood Technology 0746
 Biology
 General 0306
 Anatomy 0287
 Biostatistics 0308
 Botany 0309
 Cell 0370
 Ecology 0329
 Entomology 0353
 Genetics 0369
 Immunology 0793
 Microbiology 0410
 Molecular 0307
 Neuroscience 0317
 Oceanography 0416
 Physiology 0433
 Radiation 0821
 Veterinary Science 0778
 Zoology 0472
 Biophysics
 General 0786
 Medical 0760

EARTH SCIENCES

Biogeochemistry 0426
 Geochemistry 0606

Geodesy 0370
 Geology 0372
 Geophysics 0373
 Hydrology 0388
 Mineralogy 0411
 Paleobotany 0345
 Paleocology 0426
 Paleontology 0418
 Paleozoology 0985
 Psynology 0427
 Physical Geography 0368
 Physical Oceanography 0475

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
 Health Sciences
 General 0566
 Audiology 0390
 Chemotherapy 0922
 Dentistry 0567
 Education 0350
 Hospital Management 0769
 Human Development 0758
 Immunology 0982
 Medicine and Surgery 0564
 Mental Health 034
 Nursing 0569
 Nutrition 0570
 Obstetrics and Gynecology 0380
 Occupational Health and Therapy 0354
 Ophthalmology 0387
 Pathology 0571
 Pharmacology 0419
 Pharmacy 0572
 Physical Therapy 0382
 Public Health 0573
 Radiology 0574
 Recreation 0575

Speech Pathology 0460
 Toxicology 0383
 Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
 Chemistry
 General 0485
 Agricultural 0749
 Analytical 0486
 Biochemistry 0487
 Inorganic 0488
 Nuclear 0738
 Organic 0490
 Pharmaceutical 0491
 Physical 0494
 Polymer 0495
 Radiation 0754
 Mathematics 0405
 Physics
 General 0605
 Acoustics 0986
 Astronomy and Astrophysics 0606
 Atmospheric Science 0608
 Atomic 0748
 Electronics and Electricity 0607
 Elementary Particles and High Energy 0798
 Fluid and Plasma 0759
 Molecular 0609
 Nuclear 0610
 Optics 0752
 Radiation 0756
 Solid State 0611
 Statistics 0463
 Applied Sciences
 Applied Mechanics 0346
 Computer Science 0984

Engineering
 General 0537
 Aerospace 0538
 Agricultural 0539
 Automotive 0540
 Biomedical 0541
 Chemical 0542
 Civil 0543
 Electronics and Electrical 0544
 Heat and Thermodynamics 0348
 Hydraulic 0545
 Industrial 0546
 Marine 0547
 Materials Science 0794
 Mechanical 0548
 Metallurgy 0743
 Mining 0551
 Nuclear 0552
 Packaging 0549
 Petroleum 0765
 Sanitary and Municipal 0554
 System Science 0790
 Geotechnology 0428
 Operations Research 0796
 Plastics Technology 0795
 Textile Technology 0994

PSYCHOLOGY

General 0621
 Behavioral 0384
 Clinical 0622
 Developmental 0620
 Experimental 0623
 Industrial 0624
 Personality 0625
 Physiological 0989
 Psychobiology 0349
 Psychometrics 0632
 Social 0451

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

" Self Stabilization in Dynamic Distributed Systems "

submitted by Rick Youping Huang, B.Sc.,
in partial fulfilment of the requirements for the degree of
Master of Computer Science

Thesis Supervisor

Thesis Supervisor

Director, School of Computer Science
Carleton University

Date

In memory of my father Sixin Huang.

Abstract

In this thesis, we consider the problem of self stabilization in dynamic distributed systems. A distributed system is self-stabilizing if it can be started in an arbitrary global state and once started, the system regains its consistency in a finite time without any kind of outside intervention. The system is dynamic if both nodes and links can fail (and recover) at any time. After reviewing the existing stabilizing protocols, we approach this problem by studying an important and general issue that is topology broadcast. We present two stabilizing event driven wave algorithms for topology broadcast. Using message passing as communication primitive, these two topology wave algorithms guarantee that every operational processor in the network will have the correct topological information on the entire network within a finite time. The second wave algorithm is optimized with respect to topological changes.

To show the generality of the proposed stabilizing wave algorithms, we apply one of them to the problems of spanning tree construction and leader election. These stabilizing wave algorithms provide a general paradigm to the design of other stabilizing protocols in dynamic distributed systems.

Acknowledgements

My heartfelt thanks go to my thesis advisors Prof. Nicola Santoro and Prof. Hasan Ural. This work would not have been possible without ingenious guidance and tireless support from Nicola Santoro. His inspiration, enthusiasm, and encouragement has been a constant resource for me. I have greatly enjoyed and benefited this wonderful and stimulating period made by Nicola Santoro, which should have been very trying. I am very grateful to Prof. Hasan Ural for his initiation of this work, continued support and generosity. Many valuable discussions with Hasan Ural greatly improve this thesis.

I would like to express my gratitude to my thesis exam committee members: Prof. Danny Krizanc, Prof. Ivan Stojmenovic, and Prof. Sivarama Dandamudi for their suggestions for the improvement of this thesis.

I would like to thank Scott Reynolds and Erman Efique for their proof-reading some chapters of the draft and laughing at my dumb jokes.

Above all, the understanding, encouragement, and support from my mother and sisters have been a great driving force behind me. Finally, I thank my loving wife, Annie Liang, for helping me in more ways than I can list here throughout this adventure. Without these supports from my family, this thesis would not have been written.

Table of Contents

Abstract.....	i
Acknowledgements.....	ii
List of Figures.....	vi
List of Tables.....	vii
List of Symbols.....	viii
Chapter 0 Introduction.....	1
0.1 The Problem	2
0.2 An Example.....	3
0.3 Self Stabilization in Dynamic Distributed Systems.....	5
0.4 Motivations	6
0.5 Thesis Contribution.....	8
0.6 Thesis Organization.....	9
Chapter 1 The Model Classification and Overview of Stabilizing Algorithms.....	11
1.1 Introduction.....	11
1.2 The Model Classification for Stabilizing Algorithms ..	12
1.3 Stabilizing Shared Variable Algorithms.....	14
1.3.1 Semi-Uniform Model.....	14
1.3.1.1 Central Demon	15
1.3.1.2 Distributed Demon	17
1.3.1.3 Read/Write Demon	18
1.3.2 Id-Based Model.....	20
1.3.2.1 Shared Register Algorithm	20
1.3.2.2 Shared Memory Algorithm.....	22
1.4 Stabilizing Message Passing Algorithms.....	23
1.4.1 Stabilizing Snapshot Algorithm.....	23
1.4.2 Stabilizing Synchronizer.....	25
1.5 Topology Update and Self Stabilization.....	27
1.5.1 Topology Broadcast with Sequence Numbers	28
1.5.2 Topology Update with Multiple Copies.....	29
1.6 Chapter Summary	32

Chapter 2	The Model and Assumptions.....	35
2.1	The System Structure.....	35
2.2	The Fault Model	36
2.3	The Burst of Topological Changes.....	38
2.4	The Topological Changes	39
Chapter 3	The Stabilizing Topology Wave Algorithm.....	43
3.1	Introduction.....	43
3.2	Representation of Knowledge and Messages.....	46
3.3	The Wave Rules.....	51
3.4	The Wave Algorithm	59
3.5	Proof of Correctness	65
3.6	Time and Message Complexities.....	76
3.7	On Reset and Bounded Counters.....	78
3.7.1	Description of Reset	79
3.7.2	Explanation of Reset.....	80
3.7.3	The Correctness and Complexity of Reset.....	83
3.7.4	Comparison with related work	84
3.8	Chapter Summary	86
Chapter 4	The Optimized Stabilizing Wave Algorithm.....	88
4.1	Introduction.....	88
4.2	Representation of Knowledge and Messages.....	89
4.3	The Wave Rules.....	92
4.4	The Optimized Wave Algorithm.....	97
4.5	Proof of Correctness	102
4.6	Chapter Summary	113
Chapter 5	A Stabilizing Spanning Tree Construction and Leader Election Algorithm....	114
5.1	Introduction.....	114
5.2	Representation of Information and Messages.....	116
5.3	The Wave Rules.....	118
5.4	The Spanning Tree Construction and Leader Election Algorithm	123
5.5	Proof of Correctness	128
5.6	On Reset Primitive and Bounded Counters	134
5.7	Complexity Measures	135

5.8	Chapter Summary	137
Chapter 6	Conclusions.....	138
6.1	Summary and Discussion on the Results	138
6.2	Future Work.....	141
Appendix	Visualization of Waves	143
Glossary		144
Bibliography		148

List of Figures

Figure 0.1	An Example of A Stabilizing Protocol for A Ring Network	3
Figure 0.2	The Legitimate Configurations in A Ring Network	4
Figure 1.1	Multiple Tables Data Structure in Node i for Topology Update	29
Figure 1.2	An Example of messages increasing exponentially	30
Figure 2.1	The Hierarchy of the Models of Dynamic Systems	41
Figure 3.1	The Events and Actions in The Stabilizing Wave Algorithm	59
Figure 3.2	The Communication Delay in an Asynchronous System	68
Figure 3.3	Topological changes in two adjacent operational nodes	70
Figure 3.4	The Process of Remote Inconsistent Information	72
Figure 3.5	The Consistent Set in the Network	73
Figure 4.1	Topological Changes in Two Adjacent Operational Nodes	104
Figure 4.2	The Process of Remote Inconsistent Information	107

List of Tables

Table 0.1	The States of Processors and the State Transition	4
Table 1.1	The Complexity Measures of Existing Protocols	33
Table 2.1	The Models and Assumptions of the Computation	42
Table 3.1	The Time Complexities of Topology Broadcast Algorithms	78
Table 3.2	The Message Complexities of Topology Broadcast Algorithms	79
Table 3.3	The Comparison of the Models of Reset	86
Table 3.4	The Space Complexities of Topology Broadcast Algorithms	87
Table 5.1	The Time Complexities of Spanning Tree Algorithms	137
Table 5.2	The Space and Message Complexities of Spanning Tree Algorithms	137

List of Symbols

G	An undirected connected graph
V	A set of nodes in G
E	A set of edges in G
N	The <i>a priori</i> bound on the maximum number of nodes in G
D	The upper bound of the diameter of the graph G
$E(G)$	The bound on the maximum number of edges in G
$\varepsilon(G)$	The ratio of the network expansion
Φ	A computation model
$\Phi_{\emptyset}^{\infty}$	The model in which every node has no knowledge, and there is no <i>a priori</i> bound on the maximum number of the nodes in the network
$\Phi_{\varepsilon(G)}^{\infty}$	The model in which every node has an estimate on the ratio $\varepsilon(G)$ of the network expansion, but there is no <i>a priori</i> bound on the maximum number of the nodes in the network
$\Phi_{\emptyset}^N$	The model in which every node has no knowledge, but there exists <i>a priori</i> bound N on the maximum number of the nodes in the network
$\Phi_{N,D}^{N,D}$	The model in which every node has the initial knowledge on the <i>a priori</i> bound N on the maximum number of nodes and the bound D of the diameter of the network
Δ	The communication delay in the network
$//$	A dyadic operator for manipulation of two sets

$Adjacent(i)$	A set of identities of the currently operational adjacent nodes of node i
$pred(i)$	The identity of the node from which node i has received a message and obtained the current topological information
$succ(i)$	A subset of neighbors of node i to which it sends information messages
$resp(i)$	A subset of $succ(i)$ from which node i has received a feedback message with the correct wave number and $\mathcal{A} = \text{true}$.
W_i	The wave number at node i
$\mathcal{K}_i$	The topological knowledge set at node i
$Neighbor_{\mathcal{K}_i}(v)$	The set of the nodes incident to v , where $\mathcal{K}_i \supseteq Neighbor_{\mathcal{K}_i}(v)$
$\langle W, \mathcal{K} \rangle$	An information message
$\langle W, \mathcal{A} \rangle$	An feedback (acknowledgement) message
W	The wave number of a message
$\mathcal{K}$	The topological information carried by an information message
$\mathcal{A}$	The Boolean variable in feedback (acknowledgement) messages
$\mathcal{K}_{reset}$	An element in the reset message
$\langle \mathcal{K}_{\Delta} \rangle$	A message containing the optimized topological information
$\mathcal{K}\mathcal{W}_i[v, u]$	An element in the knowledge set at node i , and $\mathcal{K}\mathcal{W}_i[v, u] = \langle \mathcal{K}_i[v, u], W_i[v, u] \rangle$
$\mathcal{K}\mathcal{W}[v, u]$	An element in the message $\langle \mathcal{K}_{\Delta} \rangle$, and $\mathcal{K}\mathcal{W}[v, u] = \langle \mathcal{K}[v, u], W[v, u] \rangle$
$\langle W, K_v \rangle$	The information message containing a wave number and the lowest value of node identity
K_v	The lowest value of node identity
$K_{v(i)}$	The lowest value of node identity known to node i

$LC_i(\mathcal{K})$	The knowledge local consistency at node i
$GC_i(\mathcal{K})$	The knowledge global consistency at node i
$LI_i(\mathcal{K})$	The knowledge local inconsistency at node i
$RI_i(\mathcal{K})$	The knowledge remote inconsistency at node i
$LocalTopology_i$	The local topology at node i
$RemoteAmbiguity_i$	A set of the links with ambiguous status which is remote to node i
$N(S_i)$	The operational neighbors of S_i

Chapter 0

Introduction

Having experienced long and cold winters, we know that doors are important for the entrances of buildings. Observe the behavior of a door protocol in the following scenarios:

The door needs to be closed when it is not in use. If a person wants to pass through the door, then she/he must open it, walk through, and close it. This is the correct behavior. Initially, if the door is closed, the protocol can maintain the correct behavior in such a scenario. However, if there exists an unusual scenario such that initially the door is not closed or someone made a mistake, forgot to close it, after she/he left, then the door might remain open forever. In the latter scenario, the door protocol does not function any more.

We say that the door protocol described above is non *self stabilizing*. It does not work correctly with some unexpected initial situations in which some events happen accidentally. However, if a spring is installed properly, then it can overcome the problem by restoring the door back to a closed position after being opened.

Self Stabilization has been known and used in mathematics for many years, e.g., the Newton-Raphson method for finding roots of functions, in which for many functions whatever the initial estimate for the root is*, eventually will converge the iteration to the correct root.

* There are some exceptions on the initial estimates.

0.1 The Problem

In computer science, the study of *self stabilization* mainly focuses on distributed systems. It has gained a great deal of attention since it started with the fundamental paper from Dijkstra [Dij74].

A distributed system consists of a set of independent and distinct entities, called processors. Each processor can communicate directly with some other processors called its neighbors. The communication between the processors in a distributed system can take place via accessing shared memory or via exchanging message.

A distributed system is *self stabilizing* if it has a property such that it can be started from any arbitrary state and once started, it regains its consistency by itself, without any kind of an outside intervention.

A system that is self stabilizing is robust in face of topological changes such as the failures and recoveries of links and processors. For a system to regain its consistency within a finite time, topological changes must cease for a “sufficient” period of time. In another words, either topological changes cease forever or topological changes occur in burst, and the intermediate period between one burst of topological changes and the next one is “long enough”, the system will stabilize within a finite time.

0.2 An Example

We show a simple example of a stabilizing protocol, which makes a distributed system converge to its legitimate configuration in a finite time, starting from an arbitrary configuration.

The topology of network is a ring, there are only two processors in the ring (see Figure 0.1). These two processors can communicate with each other by reading the information from the other processor's memory and writing information into its own memory. Read and write can not be performed simultaneously at one location.

A ring network with two processors P_0 and P_1 .

Figure 0.1 An Example of A Stabilizing Protocol for A Ring Network

In this protocol, there are three states $\{0, 1, 2\}$ for each processor, and the transition relation δ_i for two processors, say $i \in \{0, 1\}$, is described in Table 0.1, where a is the old state of processor P_i , b is the state of the other processor, and $\delta_i(a, b)$ is the new state of processor P_i , where $i = 0, 1$.

States	a	b	$\delta_i(a, b)$
	0	1	2
	1	2	0
	2	0	1
	0	0	1
	1	1	2
	2	2	0

Table 0.1 The States of Processors and the State Transition

Since each processor can be in three different states $\{0, 1, 2\}$, there are nine possible configurations for the system. The following set of configurations can be defined as legitimate ones.

$$\{(0, 1), (2, 1), (2, 0), (1, 0), (1, 2), (0, 2)\}$$

Thus, we have the relations among these legitimate configurations (see Figure 0.2).

Figure 0.2 The Legitimate Configurations in A Ring Network

Notice that processor P_1 or P_0 is executing the state transition mutual exclusively. Only one processor can be active (enabled) at any given time, the privilege of being enabled is rotated on the ring. It implies that the above stabilizing protocol yields mutual exclusion.

For the illegal configurations such as $(0, 0)$, $(1, 1)$, and $(2, 2)$, no matter which processor is active, i.e.,

$$(1, 0) \xleftarrow{0} (0, 0) \xrightarrow{1} (0, 1)$$

$$(2, 1) \xleftarrow{0} (1, 1) \xrightarrow{1} (1, 2)$$

$$(0, 2) \xleftarrow{0} (2, 2) \xrightarrow{1} (2, 0).$$

it will guarantee the system to come back to a legitimate configuration within one move.

0.3 Self Stabilization in Dynamic Distributed Systems

With respect to the communication links and processors, distributed systems can be classified as two different types of systems:

(1) *Static System*

The links and processors that are operational remain operational during the computation period and no new links arise.

(2) *Dynamic System*

The links and processors of the system can not only crash but also recover. In another words, the communication links and processors in the system can go down or come up at an arbitrary number of times; and they can stay in the same status for an arbitrary length of time.

In a dynamic system, if all the links incident to a processor fail, the processor cannot communicate with any of its neighbors. For the rest of the active connected processors in the system, the isolated processor is viewed as a failed node. Conversely, the failure of a processor results in the failures of all the links incident to the failed processor. It implies

that a dynamic system must be able to tolerate the deletion and addition not only of the links but also of the processors without the need of initialization.

Clearly, *self stabilization* is very useful in dynamic systems for maintaining its consistency with respect to its nature, that is topological changes. It makes the system more tolerant and robust.

To approach this problem in dynamic distributed systems, we consider an important and general issue that is *topology broadcast*. The objective is to design a protocol which can guarantee that every operational processor in the network obtains the correct information on the entire network within a finite time, provided the intermediate period between one burst of topological changes and the next one is long enough.

0.4 Motivations

The *motivations* for achieving self stabilization of a dynamic system are mainly the following:

The nature of dynamic systems

In a dynamic system, the links and processors in the system can go down or come up at an arbitrary number of times. The sequence of topological changes might not be finite, that is topological changes do not cease forever. In reality, there always exist some potential failures of the links and processors in a communication system. In the sense that all the existing communication systems are dynamic.

High cost caused by outside intervention

Due to the failures of the links and processors in the system, the cost for periodic maintenance will be extremely high, if it has to be done by outside intervention.

In particular, for a decentralized wide area network, after some failures of the network, manually restoring and recovering the network requires an immense amount of coordination and effort. For example, the recovery of AT&T network which was caused by some internal failure was very costly. The time required by the outside intervention could be much longer than that required by a stabilizing system.

Distributed checking and correcting

In a distributed system, each processor executes a program to perform some function simultaneously. In some situations, it is required to bring a distributed program back to a legitimate state. Therefore, the distributed system should be able to detect the error and correct it. This problem is related to debugging in a distributed system and to the membership problem.

No initialization is required for applications

By incorporating self stabilization, the applications do not have to start in a known global state. Self stabilization guarantees the system to converge to a legitimate state, regardless of arbitrary initial states. As a result, it relaxes and simplifies the requirement of the starting condition.

0.5 Thesis Contribution

In this thesis, we review the literature on self stabilization from two aspects: (i) the communication primitives; (ii) the computation models. Most of the early work on self stabilization employ shared memory or registers as their communication primitives; in the context of distributed systems, a more natural approach for the design of a stabilizing system is by using message passing primitive. This is the communication primitive we are interested in. In our survey, we review the existing solutions by classifying them with respect to the computation models used, and examine their advantages and drawbacks.

The original and innovative contributions of this thesis are as follows:

- (1) We present a self stabilizing wave topology algorithm for a dynamic network which employs message passing as its communication primitive. Our algorithm is *event driven*, in the sense that every operational node in the network takes some actions in response to *events* such as detection of topological changes or reception of the messages. The stabilizing topology wave algorithm guarantees that every operational node in network will have the correct topological information on the entire network, provided the topological changes cease forever, or the interval between one burst of topological changes and the next one is long enough.
- (2) The proposed algorithm uses sequence numbers to distinguish new and old topological information for topology update. We discuss the relationship between having bounded counter and the reset problem with respect to the assumptions of the model, and show a version of our wave algorithm with bounded counters. We conclude that the solution for the problem of reset depends on the following factors:
 - the ratio of network expansion;

- the diameter of the network;
 - the communication delay in an asynchronous network.
- (3) Based on this stabilizing wave algorithm, we propose an optimized stabilizing wave algorithm that limits the information carried by the messages only to topological changes. By attaching a sequence number to each link, it simplifies the process of a node deciding on the consistency between the received topological information and the information known to the node, especially in the situation where there exists ambiguous information within the same wave.
- (4) To show the generality of our approach, we apply our stabilizing wave algorithm to the problems of a spanning tree construction and a leader election, obtaining a stabilizing protocol for both spanning tree construction and a leader election. The implication is that our proposed stabilizing wave algorithms can be used as building blocks to the design of other stabilizing protocols such as distributed reset and mutual exclusion in dynamic distributed systems.

0.6 Thesis Organization

This thesis is organized as follows:

In Chapter 1, we review the basic self stabilizing protocols. They are classified with respect to their models of computation and communication primitives. Since the mechanisms for topology broadcast in dynamic networks can be easily transformed and applied to other problems, the problem of topology broadcast is addressed.

In Chapter 2, we explicitly describe our model and assumptions with respect to the system structure, the fault model, and the nature of a dynamic network

In Chapter 3, we introduce a self stabilizing topology wave algorithm for a dynamic network and formally describe it. Wave numbers are employed as sequence numbers to differentiate the old and new topological information. Further, we prove its correctness and discuss the issue on reset and bounded counters under different assumptions.

In Chapter 4, we derive a refined version of the stabilizing wave algorithm. The resulting algorithm is optimized with respect to topological changes and information carried by the messages.

Chapter 5 shows one of the applications of the topology wave algorithms; that is a stabilizing wave algorithm for a spanning tree construction and a leader election in a dynamic network.

Finally, in Chapter 6, we summarize the results of this thesis, give the concluding remarks, and discuss the areas of future work.

Chapter 1

The Model Classification and Overview of Stabilizing Algorithms

1.1 Introduction

In this chapter, we will review the basic self stabilizing protocols and classify them with respect to their computation models and communication primitives.

In 1974, Dijkstra introduced the concept of the *self stabilization* and presented three stabilizing mutual-exclusion protocols for ring networks. As stated in Chapter 0, a distributed system is self stabilizing, if the system can be started in any possible global state and once started, the system regains its consistency by itself, without any kind of an outside intervention.

In distributed systems, the communication among the processors can occur via two primitives: accessing shared variables and exchanging messages. Based on these two communication primitives, there exist two types of self stabilizing algorithms:

- (i) *shared variable,*
- (ii) *message passing.*

In Section 1.2, we briefly outline and classify the models of stabilizing algorithms according to the behavior of the processors in distributed systems. In Sections 1.3 and 1.4, we describe and examine the stabilizing protocols with respect to above two communication primitives (i.e., *shared variables* and *message passing*). In Section 1.5, we address the problem of topology broadcast in a dynamic system and describe two basic topology broadcast algorithms. In Section 1.6, we compare the results among the existing protocols and summarize this chapter.

1.2 The Model Classification for Stabilizing Algorithms

Let $G=(V, E)$ be an undirected connected graph, where V is a set of nodes corresponding to processors and E is a set of edges corresponding to bidirectional communication links. Each processor can communicate directly with some other processors, called its neighbors or adjacent processors.

As we are concerned with the representation of a distributed system, we sometimes use graph, node and edge to refer network, processor and link, respectively. Therefore, graph and network, node and processor, edge and link will be used interchangeably hereafter.

The assumptions in the most related work are as follows: the failures are *detectable* and the communication delays are *finite*. The impossibility for any non-trivial computation has been proved by [FLP85], if the failures are undetectable and the communication delays are infinite. Another common assumption is that a node in the dynamic system has the knowledge about its neighbors or the links incident to it, as in [Ag90], [AKY90], and [KP90]. Similarly, FIFO channels are widely employed in many different models.

In addition, some synchrony assumption is made and the “underlying pulse” mechanism is employed to stabilize the system and regain the consistency, as in [SL86] and [HMR88]. As will be described later in Chapter 2, our asynchrony assumption is very general and realistic.

The computation models of the existing stabilizing algorithms can be classified as follows:

(i) *Semi-Uniform Model :*

In a semi-uniform model, exactly one single processor is designated as a special processor and all the rest processors are identical. Clearly, this is a restriction on its self stabilization property. Since the special processor can not be deleted, the semi-uniform systems are not dynamic. As will be reviewed later in this chapter, some existing protocols ([Dij74], [KP90], [DIM90], and [DIM91a]) employ this type of model as their computation models.

(ii) *Uniform Model :*

There is no special processor and all the processors are anonymous. Basically, some ranking protocols are employed in these uniform models to assign a distinct identity to each processor in the network. It results in randomized rather than deterministic stabilizing algorithms, as in [IJ89], [IJ90], [DIM91b], [AE91], and [AEH92]. In this thesis, we are not concerned with this type of model.

(iii) *Id-Based Model :*

Every processor has a distinct identity. This type of model can be easily converted to the uniform model by randomizing the identities of its processors. It is used in [AG90] and [AKY90].

1.3 Stabilizing Shared Variable Algorithms

With shared variable primitive, the processors in the system communicate only by accessing the shared registers or memory. Most of existing stabilizing algorithms focus on the system using this type of communication primitive, among them are [Dij74], [Kru79], [Lam84], [BGW87], [Bur87], [BP88], [IJ90], [DIM90], [AG90], and [AKY90].

In the following sub-sections, we review the basic self stabilizing shared variable algorithms with respect to the computation models described in Section 1.2. In Section 1.3.1, we describe three *semi-uniform* stabilizing algorithms. In Section 1.3.2, we present two *id-based* shared variable stabilizing algorithms.

1.3.1 Semi-Uniform Model

In this section, we examine semi-uniform stabilizing protocols. In this type of model, there exists a special processor, and processors are activated by a scheduler. Whenever a processor is activated by the scheduler, it executes a single atomic step in which the execution is uninterrupted. Based on the scheduler, there are three common demons:

- (a) Central Demon [Dij74];

- (b) Distributed Demon ([BGW89] and [Bur87]);
- (c) Read/Write Demon [DIM90].

It is important to point out that the two demons (central and distributed) presented in Sections 1.3.1.1 and 1.3.1.2 mainly focus on achieving self stabilizing *mutual exclusion* protocols in a ring network. For the sake of completeness, we review these two stabilizing protocols as the pioneer work.

1.3.1.1 Central Demon

Dijkstra [Dij74] presents a stabilizing algorithm in which there exists a central demon as a controlling element (i.e., the scheduler) to drive the system toward stability in a *ring* network.

Essentially, on each link operating between two processors, say i and j , there are two registers r_{ij} and r_{ji} , processor i writes its own state into r_{ij} and read the state of processor j from register r_{ji} , similarly processor j writes its own state into r_{ji} and read the state of processor i from register r_{ij} . Therefore, any processor on the ring can obtain all the states of its neighbors by accessing these shared registers.

At any time, only one processor is activated at random by the central demon. The activated processor executes an atomic step which is composed of reading the states of all its neighbors, deciding whether to change its state, and then moving to its new state.

During execution of this step no other processor is allowed to take any action. The example shown in Section 0.2 is a simple stabilizing protocol with central demon.

There are three drawbacks with Dijkstra's algorithm in a distributed system:

- (1) Implementing a central demon as a separate sequential machine violates the philosophy of distributed computing.
- (2) The requirement for central control is very restrictive, i.e., only one processor is activated, and the execution of a state transition in one processor disables the execution of other state transition in its neighboring processors.
- (3) Although the central demon can be implemented by introducing local mutual exclusion mechanisms that permit a processor to execute only when its adjacent processors are disabled from execution, this implementation would require the use of arbiters, which are better avoided.

Nevertheless, this is the original work which has initiated the studies of self stabilization in computer science. Another important observation made by Dijkstra is that there is no uniform self stabilizing protocol for achieving self stabilization in a ring with a composite number of processors. Further, with respect to this result in ring networks, Burns and Pachl [BP89] have proven that there exists a self stabilizing protocol without using the special processor, if the number of the processors in the ring network is prime. Recently, other self stabilizing protocols dealing with ring orientation have been proposed [IJ91] .

1.3.1.2 Distributed Demon

The motivation for designing a distributed demon is to overcome the drawbacks of the central demon. It is desirable to have a stabilizing token protocol whose correctness is not based on central demon, i.e., the state transitions of the stabilizing token protocol are noninterfering. In another words, if a state transition is enabled, it continues to be enabled regardless of the execution of other state transitions, until the execution of the state transition completes.

In the distributed demon [BG89], a subset of processors employ the above method to write their states into their own registers and to read all the states of their neighbors from their registers. All activated processors execute a single atomic step simultaneously. That is, they all read the states of all their neighbors, decide whether to change their state, and then move their new states.

To obtain mutual exclusion in a ring network, the processors are expected to circulate exactly one token. The token protocol is self-stabilizing, in the sense that, regardless of the number of tokens that initially exist, it guarantee the token system to reach a state in which only one token exists.

In general, a stabilizing token protocol is twofold:

- (1) Token protocols have many applications in a distributed environment, and they are widely used in local area networks. Thus, it is always desirable to make them as fault tolerant as possible.

- (2) The traditional method for ensuring fault tolerance of a token protocol is by adding fault detection and recovery procedures to it, which seems to result in a very complicated protocol. Clearly, a stabilizing token protocol can achieve the desirable fault tolerant properties, at the same time, it can keep itself inherently simple.

1.3.1.3 Read/Write Demon

Building a self stabilizing spanning tree is an important approach to obtain the property of self stabilization in a dynamic system. In this section, we review a self stabilizing spanning tree construction protocol [DIM90].

The model of this read/write demon is *semi-uniform*, i.e., there exists a special processor to break the symmetry.

The processors communicate via accessing shared registers. However, it is required that all shared registers are serializable with respect to read/write operations and only one atomic step is executed at any time.

Algorithm: Stabilizing Spanning Tree with Read/Write Atomicity

{ **Model** : $G = (V, E)$ (an undirected connected graph) *semi-uniform*;

Objective: to construct and maintain a spanning tree;

Assumptions : Read/Write Atomicity.

r is the maximum number of degree of a node in G , and there is an arbitrary ordering for each processor i to read the registers of its neighbors so that node i can select one node among its neighbors; in case, the neighbors have the same distance to the root. }

```

variable   first_found:                boolean
              dist :                      integer
              irji (father, distance):    integer ;
              /* the latest status of register  $r_{ji}$  , if father field is 1, processor  $i$  is the
                 father of processor  $j$ . */

begin
  do forever

    (root)
       $\Rightarrow$  for  $m:=1$  to  $r$  do write  $r_{im} := \langle 0, 0 \rangle$  od;

    [] else  $\Rightarrow$  /* for the rest processors in the dynamic network */
      for  $m:=1$  to  $r$  do  $ir_{mi} := \text{read}(r_{mi})$  od;
      first_found := false;
      dist :=  $\min(ir_{mi}.distance) + 1$ ; /* to compute the shortest distance */
      for  $m:=1$  to  $r$  do
        if not first_found  $\wedge$   $ir_{mi}.distance = dist - 1$ 
           $\Rightarrow$  write  $r_{im} := \langle 1, dist \rangle$ ; first_found := true;
        [] else
           $\Rightarrow$  write  $r_{im} := \langle 0, dist \rangle$ ;
        fi ;
      od

    od /* forever */
  end

```

Stabilizing Spanning Tree Algorithm with Read/Write Atomicity

Clearly, the special (i.e., *root*) processor is to ensure that the distance between itself and its operational neighbors updated with respect to the topological changes. This protocol is semi-uniform and using shared registers. Its stabilization time complexity is $O(D)$, where D is the unknown upper bound of the diameter of the network.

1.3.2 Id-Based Model

As stated in Section 1.2, every processor has a distinct identity in *id-based* model. In this section, we show two *id-based* stabilizing protocols ([AG90] and [AKY90]), these two protocols cope with the problems in dynamic systems, and they employ shared variables as their communication primitives.

1.3.2.1 Shared Register Algorithm

Arora and Gouda [AG90] present a self stabilizing spanning tree construction algorithm. It assumes that each node has an unique identity and the *a priori* bound N on the number of nodes in the dynamic system is known. Processors in the system can communicate by the single-write and multiple-read atomic registers. Each processor i has the initial information $Adjacent(i)$ that is a set of identities of the operational neighbors of i . Every operational node obtains the knowledge about the status of all its adjacent nodes by reading the registers of its neighbors.

The objective of this algorithm is to construct and maintain the spanning tree in a dynamic system. To maintain the spanning tree, certain local conditions are defined so that if violated in the neighborhood of a node, then the node detects it and restart (reset) the algorithm by considering itself the root of a single tree.

Algorithm: Arora and Gouda's Spanning Tree Construction

{ **Model** : $G=(V, E)$ (an undirected connected graph) *Id-based*;

Objective: to construct and maintain a spanning tree whose root has the highest identity value;

Assumptions : The *a priori* bound N on the maximum number of nodes in the graph is known.

$Adjacent(i)$ is a set of identities of its neighbors, which is maintained by a lower level self stabilizing mechanism. }

variable $root(i), f(i)$: $1.. N$; /* $root(i)$ is the present root of i , $f(i)$ is the father of i */

$d(i)$: integer; /* the distance between the root and the processor i */

parameter j : $1.. N$;

 /* the other processor j , where N is the known *a priori* bound of the number of nodes */

begin

do forever

$(root(i) < i) \vee (f(i) = i \wedge (root(i) \neq i \vee d(i) \neq 0)) \vee$
 $(f(i) \notin (Adjacent(i) \cup \{i\}) \vee d(i) \geq N)$

$\Rightarrow root(i) := i; f(i) := i; d(i) := 0;$

[] $(f(i) = j) \wedge (j \notin Adjacent(i)) \wedge (d(j) < N) \wedge$
 $((root(i) \neq root(j)) \vee (d(i) \neq d(j) + 1))$

$\Rightarrow root(i) := root(j); d(i) := d(j) + 1;$

[] $(root(i) < root(j) \wedge j \in Adjacent(i) \wedge d(j) < N) \vee$
 $(root(i) = root(j) \wedge j \in Adjacent(i) \wedge d(j) + 1 < d(i))$

$\Rightarrow root(i) := root(j); f(i) := j; d(i) := d(j) + 1;$

od;

end

Arora and Gouda's Spanning Tree Construction Algorithm

To reset the network, sequence numbers are used to distinguish between old and new messages. The stabilization time complexity depends on the bound of the network size $O(N^3)$, where N is the known upper bound on the maximum number of nodes in the network.

1.3.2.2 Shared Memory Algorithm

Afek, Kuten and Yung [AKY90] propose a self stabilizing spanning tree construction and maintenance algorithm in the dynamic network [AAG87]. The size of the network is unknown. Sufficient information is maintained at the nodes, so that illegal states can be detected locally by an operational node which can restart the algorithm.

In their model, it is assumed that every processor has a unique identity and the initial knowledge about its neighbors and adjacent links. The processors communicate by reading the memory of neighboring processors and writing its own local memory.

The algorithm can be briefly described as follows:

Every node tries to construct a tree in the network from itself. The tree with a larger identity node as its root overruns the tree with a lower identity node. Eventually, the tree of the larger identity node overruns all the other trees.

To compute and maintain a stabilizing spanning tree, some local predicates are defined for each node. In addition, a legal global state is defined and maintained for the entire dynamic system. Based on these predicates, each node maintains its local consistency. If the conditions are violated, it starts a global correction by leaving the tree and sending a request to the node with the larger identity. In order to join another tree, whose root identity is larger, a node has to propagate a request message along the new tree branches to the root and receive a grant message back. Upon receipt of the grant message, the initiator of the request joins the tree.

The time complexity is $O(N^2)$. The amount of space consumed per processor for this algorithm is $O(r \log N)$, where N is the unknown bound on the maximum number of the processors of the network, and r is the maximum degree of a processor in the network. A salient feature of this protocol is that it does not require the counters and the diameter of the network is unknown.

1.4 Stabilizing Message Passing Algorithms

From the previous sections, we observe that the stabilizing protocols described above depend on the communication media, that is shared registers or memory. Obviously, in the context of distributed systems, the natural approach is to use message passing primitive for the design of stabilizing protocols. In Sections 1.4.1 and 1.4.2, we describe two stabilizing algorithms which are proposed for message passing systems.

1.4.1 Stabilizing Snapshot Algorithm

Katz and Perry [KP90] propose that non-stabilizing systems can be stabilized by periodically performing snapshot and doing centralized checking and correction. It assumes the special node as a root, which implies that the model is *semi-uniform*. In this proposed stabilizing snapshot algorithm, a global counter is employed to differentiate the different snapshots.

The overview of the stable snapshot (*Stablesnap*) algorithm is as follows:

Stablesnap algorithm is based on an iterated version of the single snapshot algorithm by Chandy and Lamport [CL85]. The special processor has the following responsibilities:

- (1) Initiate each new iteration of the single snapshot algorithm by sending messages (maker) to its adjacent nodes;
- (2) Avoid the deadlock situation, due to the arbitrary initial state, by generating prod messages;
- (3) Collect the global state by receiving the report messages which contain local snapshots.

For a processor in the network, upon receipt of a maker message, the processor invokes the single snapshot algorithm to record its local state, which is the state of its adjacent links, and it propagates the maker message. Subsequent makers ensure the channel recording to terminate. A processor terminates when it has recorded one maker on each incoming channel.

Accurate global snapshots can be obtained only by the special processor. In addition, the original snapshot algorithm is stabilized by taking the following two actions:

- (a) Due to the arbitrary initial state, the special processor sends prod messages to avoid the deadlock. *Sequence numbers* are employed by each processor to distinguish the old and new local snapshots;
- (b) Each message also contains a path which is a sequence of processor identities to avoid infinite propagation.

As in other stabilizing algorithms, this stabilizing snapshot algorithm stresses that the desired stabilizing property is eventually established. Further, Katz and Perry show a

bounded counter stabilizing snapshot algorithm which requires the *a priori* bound N on the maximum number of nodes in network to be known.

The time complexity is $O(N^2)$, where N is the bound on the maximum number of the nodes in the system.

1.4.2 Stabilizing Synchronizer

The stabilizing synchronizer (*resynchronizer*) is proposed by Awerbuch and Varghese [AV91] to achieve self stabilization in a dynamic system. It is a self stabilizing version of a *synchronizer* [Awe85].

The original synchronizer is to simulate any synchronous system by employing a *pulse* number at each processor in the asynchronous system. The pulse number is synchronized such that the difference of the pulse numbers between a processor and any its operational neighbors is at most one. In the non-stabilizing version of the synchronizer, it depends on the initialization of the system. Based on this fact, it can increment its pulse number from n to $n + 1$ after all its neighbors have reached n . By using the synchronizer, the computation in a synchronous model can be simulated in an asynchronous model.

The stabilizing synchronizer (*resynchronizer*) is obtained by adding two phases:

- *resynch phase* ;
- *termination-detection phase* .

Resynch phase is put in the beginning, when the system starts. Its objective is to cope with the arbitrary initial state. During the *resynch* phase, since there exist arbitrary pulse numbers and messages in transit on links in the initial state, the pulse number at each

processor is synchronized by lowering its pulse to catch up with the lower ones of its operational neighbors.

Termination-detection phase is extended at the end to ensure the termination of the simulated synchronous computation and to keep the counters (*pulse number*) bounded. In this termination-detection phase, the resynchronizer waits long enough that the simulated computation is finished at each node, then, it wrap the counters around afterward. The reason is that it intends to avoid the situation in which the simulated computation and the counter reset are performed simultaneously.

In general, this stabilizing synchronizer can convert a deterministic synchronous protocol for static networks into a self stabilizing version for dynamic networks.

The time complexity is $O(D)$, where D is the upper bound on the diameter of the dynamic network. To reset the counters (*pulse number*), it requires the bound on the diameter of the dynamic network D to be known.

To summarize the stabilizing algorithms described in Sections 1.3.2.2, 1.4.1, and 1.4.2, the shared memory stabilizing algorithm proposed by Afek, *et. al.* [AKY90] can be viewed as *local checking and global correction*, Katz and Perry's stabilizing snapshot algorithm [KP90] is *global checking and global correction*; these two proposed methods have the time complexities $O(N^2)$. Awerbuch and Varghese's stabilizing resynchronizer is so called *local checking and correction*, which reduces the time complexity to $O(D)$.

1.5 Topology Update and Self Stabilization

After reviewing the stabilizing protocols presented in the previous sections, we notice that an important and aggressive approach to study self stabilization is by investigating the problem of topology broadcast in a dynamic system. The issue is on the awareness of the topological changes and the knowledge at each operational node in a dynamic network. The objective of topology broadcast algorithm is to ensure every operational node to have the correct topological information on the entire network.

As will be shown later, it is a general approach for dealing with the stable computation in a dynamic system. It can be easily extended and applied to other stabilization issues such as spanning tree construction and distributed reset.

Due to the nature of a dynamic system, a topology broadcast algorithm must be capable of incorporating any new information during the execution so that a consistent and correct view of the network topology is maintained at each operational processor in the network. Further, if topological changes occur very frequently in a short period of time, processors should distinguish the old and new information related to a specific status of a link or a processor.

In Section 1.5.1, we briefly describe a topology broadcast algorithm that uses sequence numbers to distinguish the old and new topological information. In Section 1.5.2, we introduce a topology broadcast algorithm without using sequence numbers, and show one of its applications. We will give remarks on these two existing topology broadcast algorithms, respectively.

1.5.1 Topology Broadcast with Sequence Numbers

Finn [Fin79] proposes a topology broadcast algorithm for a dynamic network. It is mainly concerned about the *connectivity* of a dynamic network.

When the system starts, every operational node computes its local information on the connectivity and exchanges it with all its operational neighbors. These nodes continue to send the received information to their neighbors. To each processor, the information is updated or increased by successive accumulations of the local information. In addition, sequence numbers are used to distinguish the old and new information. Finn proposes two versions of the algorithm. The second one is to use bounded sequence numbers and to send only the differences of the version numbers.

Since its goal is to compute the connectivity in a dynamic network, the process of the received information brought in by a message is simple. However, the algorithm can not handle the ambiguous information within the same sequence number, that is the information brought in by the received message and the information known to the node is inconsistent.

Moreover, Soloway and Humblet [SH86] have shown that the version of bounded sequence numbers with Finn's algorithm can not correctly complete, as it can generate an infinite number of restart after all topological changes cease. Therefore, the algorithms such as [Seg81] that are based on Finn's proposal of sending bounded (difference) of sequence numbers may not be correct either.

1.5.2 Topology Update with Multiple Copies

To avoid using sequence numbers to distinguish the old and new information, a topology broadcast algorithm proposed by Spinelli and Gallager [SG89] is to keep multiple information copies at each node. Each node maintains its own topology table and keeps the copies of the topology tables of all its neighbors (see Figure 1.1). The purpose of maintaining multiple tables in the protocol is to reconcile any discrepancies from these tables (topological information) so that each node eventually learns and obtains the correct topological information of entire dynamic system.

Figure 1.1 Multiple Tables Data Structure in Node i for Topology Update

Most assumptions with respect to the dynamic system in their model are the same as [AAG90], and a node failure is represented by the failures of the links adjacent to the node. Here is a set of rules for governing message broadcasting and table updating:

- (1) When the main table is changed, a message containing the new change is sent on each operational adjacent link;
- (2) When a link becomes operational, two nodes on both ends of the link exchange their *entire* main topology tables;

- (3) Any adjacent link status change is entered the node main topology table;
- (4) When a single link status message is received from a neighbor, the port topology table associated with the neighbor is updated to reflect the change.
- (5) When a entire main topology table is received from a neighbor, the port topology table associated with the neighbor is updated and the main table is updated to indicate the link is up.

Whenever the main or port topology tables is modified due to rules (3), (4), and (5), the main topology table is maintained by applying a update algorithm similar to Dijkstra's *the shortest path algorithm*, i.e., by computing the shortest distance to the topological changes to determine the correct information and disregard the ambiguous information.

The amount of space required per node is $O(r E(G) \log N)$, where $E(G)$ is the bound on the maximum number of links, N is the unknown *a priori* bound on the maximum number of nodes, and r is the maximum degree of a node. In general cases, message complexity is $E(G)$ per topological change. However, in some special case, the message complexity of this algorithm increases exponentially with the number of links which change status $O(2^k E(G))$, where k is the number of links whose status have been changed.

An example is illustrated by Figure 1.2 in which the message complexity grows exponentially

Figure 1.2 An Example of messages increasing exponentially

It involves two link status changes. Node b detects that link A is down when it receives a message on link B . When link B crashes, node b decides link A is up. However, when a message eventually arrives from node f , node b decides link A is down again. Observe that node b receives three (i.e., 2^2-1) messages regarding the status of link A . Consider node a , there are three messages sent over link C to node a . Assume that they arrive before those transmitted via node e , after node a receives these three messages from node b , link C fails. Then node a examines its port topology table associated with node e and decides that link A is up. Now let another three messages sent by node e arrive node a . Therefore, there are totally seven (2^3-1) topological status changes concerning link A .

By applying Spinelli and Gallager's topology broadcast mechanism, Anagnostou, El-Yaniv, and Hadzilacos [AEH92] propose a self stabilizing diffusion scheme for the *unique naming* problem.

In a nutshell, it can be described as follows:

For a processor to disseminate a piece of information to all other processors, it creates a token containing the information and some other control information. It then sends the copies of the token to all its neighbors, which continue to send them to their neighbor and so on, until all the processors receive the copies of the token. The returned token (acknowledgement) retraces its path in reverse. When the initiator of the diffusion receives all token copies, it learns that the computation has been completed. The table of each node keeps track of information to which node it has passed the token tuple and from which node it expects back.

With respect to the ambiguous information and tokens with the same name being received, it can be resolved by computing and comparing the length of routes in between

the processor and the initiator, i.e., the shortest path (minimum-hop) solution. In the their proposed algorithm, the token is passed from one processor to another by writing it into a shared register between them. All the processors in the network are anonymous. The computation model is *uniform* by randomizing the identities of processors (with a ranking function).

It inherits the problem of unbounded message complexity under the same situation, in which multiple link status changes occur very frequently over a short period of time. As described earlier in this section, message complexity is increasing exponentially.

To remedy the problem of unbounded message in Spinelli and Gallager's algorithm [SG89], Gafni proposes a solution to reduce the message complexity. It seems to us that the proposed protocol uses the underlying pulse mechanism to reduce the number of messages. In the context of an asynchronous distributed dynamic system, the requirement of the underlying pulse mechanism is restrictive.

1.6 Chapter Summary

In this chapter, we have reviewed the existing stabilizing protocols with respect to their communication primitives and classify their computation models. Further, we have addressed a general issue on self stabilization in a dynamic system, that is topology broadcast. Two existing basic topology broadcast algorithms have been presented.

The comparison among the existing stabilizing protocols is as follows:

Time complexity	Method	Comments
$O(N^3)$	[AG90]	using shared variables (id-based)
$O(N^2)$	[KP90]	stabilizing snapshot (semi-uniform)
$O(N^2)$	[AKY90]	using shared memory (id-based)
$O(D)$	[DIM90]	using shared variables (semi-uniform)
$O(D)$	[AV91]	using message passing (D is known)
$O(D)$	[Fin79]	topology broadcast with counters
$O(D)$	[SG89, AEH92]	topology broadcast without counters
Message complexity		
$O(E(G))$	[Fin79]	Wave algorithm using counters
$O(2^k E(G))$	[SG89, AEH92]	Using multiple copies for topology update

Table 1.1 The Complexity Measures of Existing Protocols

Note: N is the *a priori* bound on the maximum number of nodes in the G , r is the maximum degree of a node, D is the upper bound on the diameter of G , $E(G)$ is the bound of the maximum number of edges of G , and k is the number of topological changes

Most of the existing stabilizing algorithms have focused on the distributed systems using shared variables as their communication primitives. The paradigm should not be restrictive, since it does not depend on the communication media employed by the processors in the distributed systems. However, self stabilization in message passing systems has not been extensively studied.

It is natural to explore and investigate the problem of self stabilization in message passing dynamic systems. In addition, with respect to the communication primitives, it is

clear that the mechanisms for achieving self stabilization in message passing systems are more general.

Chapter 2

The Model and Assumptions

In this chapter, we present our model of the computation from four aspects: the system structure, the fault model, the burst of topological changes, and the nature of the topological changes. Further, we compare it with the models of the related work.

2.1 The System Structure

Let $G=(V, E)$ be an undirected connected graph, as described in Chapter 1. In our topology broadcast model, each node in the graph has a distinct identity and the initial knowledge about its neighbors. To represent this knowledge, a set of identities of all the operational adjacent nodes is employed, as in [AG90] and [AKY90].

Each processor in the dynamic network sends signals to all its neighbors periodically and waits to receive from all neighbors so that it can maintain a set of identities of the currently operational adjacent nodes; we will denote the set maintained at processor i in the network by $Adjacent(i)$. This can be done in practice, e.g. “detecting” failures, by using timeout mechanism. We assume that any operational processor in the system will acknowledge the signal within a finite time. A reasonable period of time is selected such that one node can receive an acknowledge message from an operational adjacent node.

2.2 The Fault Model

We assume that the failures are *detectable*. This assumption is the same as in all the previous work (e.g. [AAG87], [AC90], [AKY90], [DIM90] and [DIM91]).

Its main reason for the above assumption rests on the fact that if failures are undetectable or if their detection may be delayed indefinitely, no network can be reset or brought back to a legitimate configuration. Further, Fischer, Lynch and Paterson [FLP85] have proven the impossibility of any non-trivial computation in such a case.

In practice, the current status of the processor and its incident links are known to all its neighbors. Every node has the initial knowledge about all its neighbors. This assumption is not restrictive. As stated in Section 2.1, it can be achieved with timeout mechanism.

Processors communicate only by exchanging messages via the links of the network. There is no common memory or registers shared by the nodes and there is no global clock. Messages sent over a link arrive error-free at the other end, after an arbitrary but finite delay. If there is a link failure, the messages in transit on the link may be lost. Moreover, if there is a node failure, the messages in transit on the adjacent links of the node may be lost. Therefore, a node failure can be simulated by the failures of all its adjacent links.

There are two types of models in message passing systems:

(1) *Synchronous System*

There exists a global clock in the system. The message transmission and reception are synchronized. That is the message sender is blocked until the receiver is ready to receive the message. The message delay is bounded.

(2) *Asynchronous System*

The system has no global clock. The sender is not blocked, and the receiver may take an arbitrary long to receive the message after it has been sent.

This thesis considers an asynchronous distributed system with *finite* communication delay. The asynchrony assumption is very realistic, which makes our studies more general. Obviously, the results for the models with synchrony assumptions or with an underlying *pulse* mechanism, as in [AY91], [APV91], and [AEH92], can be easily obtained from this general approach.

Each communication link is a FIFO channel: in the absence of failures, messages sent on a link are received in the same order they were sent ([Fin79], [SG89], [KP90], and [DIM91b]), i.e., messages might be lost but not put out of order. Clearly, messages cannot be received before they are sent.

Since the capacity of the communication link is always bounded in the real world, the capacity of each link is $O(1)$. It implies that only a constant number of messages are in transit on a link at a given time, as in [KP90] and [AY92].

In this thesis, we will present the wave algorithms which are *event driven*. With respect to the events, there are two types:

- (a) *Detection of topological changes.*
- (b) *Reception of information messages.*

Normally, an operational processor takes the actions in response to the events according to their time of occurrence (i.e., local time ordering). If multiple events occur in one processor at the same time, the ordering of processing is as follows:

- If both types of events occur simultaneously, the action(s) in response to the detection of topological changes take prior over the ones in response to the reception of a message.
- If a processor detects more than one topological changes at the same time, it will process them simultaneously.
- If a processor receives more than one messages simultaneously, it will process them in an arbitrary order.

A topological changes must occur before any processors detects it. The interval, which is *from* the time when the activities of a processor is triggered by an event *to* the end of the execution of the corresponding actions, is an atomic and indivisible *step* for a processor.

2.3 The Burst of Topological Changes

The graph topology is dynamic: the links and processors in the network can be down or up at an arbitrary number of times; and they can stay in the same status for an arbitrary length of time as well.

With respect to topological changes, there are three possible situations:

(i) *Finite*

The sequence of topological changes is finite; that is the topological changes eventually cease. This is the same model as in [AAG87], [AG90], [AKY90], and [DIM90].

(ii) *Infinitely discrete*

The topological changes occur in “burst” and the period in between one burst and the next one is long enough so that the system stabilizes ([DIM91] and [APV91]).

(iii) *Infinitely continuous*

The network is continuously changing so that the system does not self stabilize [AY92].

In this thesis, since the focus of our studies is on self stabilization, the computation that we are concerned about is in dynamic systems with situation (i) or (ii). We are not interested in the situation (iii) in which the topological changes occur so frequently that the system can not ever self stabilize.

2.4 The Topological Changes

A dynamic network allows not only for the processors deletion but also for the processors addition in the system. As discussed later, the complexity of our broadcast algorithms depend on the existence of *a priori* bound on the maximum number of processors in the system and the ratio of the network expansion.

We allow the failures for arbitrary link, provided that the network is not partitioned. This is the same as in [KP90] and [Ric92]. When all the adjacent links of a node fail, the node is considered to be failed. Clearly, the converse is true as well. Thus, with respect to the failure of a node, there are two scenarios:

- (a) The processor itself fails, which results in the failures of all its adjacent links;
- (b) All the links incident to the node fail, and the processor is isolated. Thus, for the other connected active processors in the network, the disconnected processor is viewed as a failed node.

When a processor becomes operational, it must attach to an active processor. For a processor to become operational, at least one of the links incident to it must be operational. Based on this fact, the implication is that the up (operational) processor is attached to an active adjacent processor, and it can communicate with it via the operational link .

To use the changes of the link status for representing the change of the node status, we assume that, when a processor i becomes operational, it resets the status of all the links incident to it be down (failed) in its local information immediately, before preceding any other actions. This is not restrictive, since an operational node has the initial knowledge on all the links incident to it. Then the node i detects which link is operational by checking the set of currently operational nodes $Adjacent(i)$, which will trigger the node to update its local information and broadcast the topological information, as will be presented later.

With respect to node addition, the number of processors in the system changes dynamically. On one hand, if there is no *a priori* bound for the maximum number of node in the graph, it is impossible for any topology algorithm to keep bounded its message sequence numbers (i.e., counters) and the amount of memory required per processor. On

the other hand, if there exists some *a priori* bound N on the maximum number of processors which can potentially join in the network, it is possible to make the amount of memory required per processor and the counters bounded. This bound, however, might not be known to the processors. If we know the *a priori* bound N on the maximum number of processors in the system, the case is trivial and the corresponding result can be easily obtained from our results.

Therefore, there are the following scenarios for the model of computation (Φ):

The subscript of Φ is the knowledge known to every node, and the superscript indicates whether there exists *a priori* bound on the maximum number of the nodes in the system. The hierarchy of the models of dynamic systems is as follows (see Figure 2.1).

Figure 2.1 The Hierarchy of the Models of Dynamic Systems

- $\Phi_{\emptyset}^{\infty}$: The model in which every node has no knowledge, and there is no *a priori* bound on the maximum number of the nodes in the network ;
- $\Phi_{\varepsilon(G)}^{\infty}$: The model in which every node has an estimate on the ratio $\varepsilon(G)$ of the network expansion, but there is no *a priori* bound on the maximum number of the nodes in the network;
- $\Phi_{\emptyset}^N$: The model in which every node has no knowledge, but there exists *a priori* bound N on the maximum number of the nodes in the network;
- $\Phi_{N,D}^{N,D}$: The model in which every node has the initial knowledge on the *a priori* bound N on the maximum number of nodes and the bound D of the diameter of the network.

Table 2.1 illustrates the requirements and assumptions on the models.

The Model of Computation	Topological changes cease	The interval is sufficient but finite	Fault Detectable
$\Phi_{\emptyset}^{\infty}$ $\Phi_{\varepsilon(G)}^{\infty}$ No Bound (this thesis)	Yes	Yes	Yes
$\Phi_{\emptyset}^N$ <i>a priori</i> Bound (this thesis)	Yes	Yes	Yes
[AY92]'s Model	No	No	Yes
Fisher, Lynch & Paterson's have proven the impossibility for any non-trivial computation [FLP85]			NO

Table 2.1 The Models and Assumptions of the Computation

Chapter 3

The Stabilizing Topology Wave Algorithm

3.1 Introduction

In a dynamic system, topological changes may occur at any time. A distributed algorithm, namely topology algorithm, is designed to guarantee that each node is made aware of the topological changes so that the system can stabilize within a finite time, provided that (1) the topological changes eventually cease forever; or (2) the interval between one burst of topological changes and the next one is long enough but finite.

An efficient and important approach to achieve the above objective is via message broadcasting. Normally the messages contain the topological and control information. Therefore, the topological information at each node is kept current by passing these topology update messages periodically or in response to a topological change.

There are some issues that should be considered for topology broadcast:

- (i) When a link goes down, a message containing topological information in transit on the link may be lost; in the worst scenario, it divides the network into two disconnected sets of processors.
- (ii) To maintain a consistent and correct view of the network topology for each operational processor in the network, a topology algorithm must be capable of incorporating any new information during the execution.

- (iii) In a short period of time, if topological changes occur very frequently, processors should distinguish the old and new information related to a specific status of a link or a processor. A topology algorithm must efficiently ensure that each up processor is aware of the latest topological information within a finite time.

As reviewed in Chapter 1, there are two basic topology broadcast algorithms for dynamic networks:

- (i) Finn's algorithm [Fin79] employs sequence numbers to distinguish the new and old topological information. However, the algorithm can not handle the conflicting information, that is the information from the topology update messages with the same sequence numbers is ambiguous. Moreover, Soloway and Humblet [SH86] demonstrate that the version of bounded sequence numbers with Finn's algorithm can not correctly terminate.
- (ii) Spinelli and Gallager [SG89] propose a topology maintenance algorithm for arbitrary dynamic network. It requires each node to keep not only its own table for global topological information but also all its neighbors' topological information tables, the amount of memory required per node is fairly high $O(r N^2 \log N)$, where N is the *a priori* bound on the maximum number of nodes in the network and r is the maximum degree of a node. In the case, the size of network changes with time (i.e., there is no *a priori* bound on the maximum number of nodes), its memory requirement is unbounded. As described in Section 1.5.2, its message complexity is growing exponentially with respect to topological changes in some cases.

To broadcast the topological information in a dynamic network, we propose a stabilizing wave algorithm. Basically, we transform the information diffusion algorithm

for a static network by [SL85] to a stabilizing topology wave mechanism for an asynchronous dynamic distributed system.

The behavior of the original information diffusion algorithm can be described as follows:

Every processor sends to its neighbors its local knowledge which consists of the information about its neighbors and itself. Upon receipt of such information, a processor updates and forwards it along with its own local knowledge. The knowledge is updated or increased by successive accumulations of the local information. The algorithm will terminate after every processor has received the local information of all the processors in the network. The messages sent by all the processors form a wave whose effect is the broadcast of the network topology.

Obviously, the mechanism from [SL85] does not tolerate any type of failure. In the rest of this chapter, we present a stabilizing wave algorithm in which a topological information message contains the entire information of the network accumulated by the sender. Our algorithms are *event driven*, i.e., a topological information message is originated in response to a topological change. Wave numbers are used to distinguish the old and new information.

In Section 3.2, we show how the topological knowledge and messages are represented in our algorithm. Sections 3.3 and 3.4 are devoted for presenting a stabilizing wave algorithm for topology broadcast, in which the processors communicate by exchanging the entire topological information available to them. In Section 3.5, we prove the correctness of the stabilizing wave algorithm. In Section 3.6, we show time and message complexities. In Section 3.7, we discuss the issues on reset and bounded

counters with respect to the dynamic network, and show space complexity measures. Finally, in Section 3.8, we conclude and summarize the chapter.

3.2 Representation of Knowledge and Messages

The topological knowledge of the dynamic system known to a node i in the network $\mathcal{K}_i$ is represented by a two-dimensional adjacency matrix ($K \times K$), where $K \leq N$ and K is changing with time (a non-decreasing integer), N is the unknown the *a priori* bound on the maximum number of nodes in the graph G .

In the matrix $\mathcal{K}_i$ of node i , $\mathcal{K}_i[v, u]$ represents the information about a bidirectional link between node v and node u :

$$\mathcal{K}_i[v, u] = \begin{cases} 0 & \text{the link between } v \text{ and } u \text{ is failed (down) ;} \\ 1 & \text{the link between } v \text{ and } u \text{ is operational (up) ;} \\ 2 & \text{the status of the link } (v, u) \text{ is ambiguous, } v, u \neq i ; \\ \text{Undefined} & \text{the existence of the link between } v \text{ and } u \text{ is unknown;} \end{cases}$$

where $v, u \in \mathcal{K}_i$.

By definition, the status of any links incident to node i must not be ambiguous.

For a node v in the network, it is represented by $\mathcal{K}_i[v, v]$. The node status is as follows:

$$\mathcal{K}_i[v, v] = \begin{cases} 0 & \text{processor } v \text{ is failed (down);} \\ 1 & \text{processor } v \text{ is operational (up);} \\ 2 & \text{the status of processor } v \text{ is ambiguous, } v \neq i; \end{cases}$$

where $v \in \mathcal{K}_i$.

With respect to the knowledge $\mathcal{K}_i$ known to processor i , given a processor v , $Neighbor_{\mathcal{K}_i}(v)$ denotes the set of all the nodes, which are incident to v , where $v \in \mathcal{K}_i$, that is

$$Neighbor_{\mathcal{K}_i}(v) \stackrel{\text{def}}{=} \bigcup_{\mathcal{K}_i[v,u] \text{ is defined}} \{u\}$$

Clearly, $Neighbor_{\mathcal{K}_i}(v)$ is a subset of $\mathcal{K}_i$, that is $\mathcal{K}_i \supseteq Neighbor_{\mathcal{K}_i}(v)$.

Notice that the status of a node v is ambiguous (i.e., $\mathcal{K}_i[v,v] = 2$), if and only if for all the links incident to the node v , the following condition holds:

$$\left(\bigwedge_{u \in Neighbor_{\mathcal{K}_i}(v)} \mathcal{K}_i[v,u] \neq 1 \right) \wedge \left(\bigvee_{u \in Neighbor_{\mathcal{K}_i}(v)} \mathcal{K}_i[v,u] = 2 \right)$$

where $v \notin Adjacent(i)$.

As stated in the model (Chapter 2), all the topological changes are represented by the changes of the status of links. When a node, say i , becomes operational, it resets the status of all the local links to be 0 (down) in its local information $\mathcal{K}_i$ immediately; and the following condition holds:

$$\bigwedge_{j \in Neighbor_{\mathcal{K}_i}(i)} \mathcal{K}_i[i,j] = 0$$

Then it detects which links are operational by checking the $Adjacent(i)$. Therefore, it will trigger the conditions, which will be shown in Section 3.3, for initiating new waves to broadcast the current topological information. The implication is that our stabilizing wave algorithm is *event driven*.

By definition, each node has the initial knowledge about all its neighbors and adjacent links. With respect to the local status of all the links incident to node i , the local topology of i is denoted by $LocalTopology_i$, that is

$$LocalTopology_i = \{LT_i[j]\},$$

$$\text{where } LT_i[j] = \begin{cases} 1 \text{ (up)} & j \in Adjacent(i) ; \\ 0 \text{ (down)} & j \notin Adjacent(i) \text{ and } j \in Neighbor_{\mathcal{K}_i}(i) ; \\ \text{Undefined} & \text{Otherwise .} \end{cases}$$

Given a node i , let

$\langle W_i, \mathcal{K}_i \rangle$ be the wave number and topological knowledge known to node i .
 $\mathcal{K}_i[i,j]$ and $\mathcal{K}_i[v,u]$ are elements in $\mathcal{K}_i$ (i.e., $\mathcal{K}_i[i,j], \mathcal{K}_i[v,u] \in \mathcal{K}_i$);

and let

$\langle W, \mathcal{K} \rangle$ be an information message currently received by i .
 $\mathcal{K}[i,j]$ and $\mathcal{K}[v,u]$ are elements in $\mathcal{K}$ (i.e., $\mathcal{K}[i,j], \mathcal{K}[v,u] \in \mathcal{K}$).

There are two types of messages in the systems:

- (i) Information message is a couple $\langle W, \mathcal{K} \rangle$, where
 - (a) W is an integer, which denotes the wave number to distinguish old and new messages;
 - (b) $\mathcal{K}$ is the adjacency matrix, which denotes the topological knowledge known to the sender. In general, this is the result of accumulations of local knowledge of the sender.
- (ii) Feedback message is a couple $\langle W, \mathcal{A} \rangle$, where

- (a) W is an integer, which denotes the wave number of the received message that the processor is acknowledging;
- (b) $\mathcal{A}$ is a boolean variable.

Upon receipt of one information message $\langle W, \mathcal{K} \rangle$, a processor i will reply with a feedback message $\langle W, \mathcal{A} \rangle$, where W is the same as the wave number in the received information message, and $\mathcal{A}$ is defined as follows:

$$\mathcal{A} = \begin{cases} \text{true} & \begin{array}{l} \text{the information } \mathcal{K} \text{ brought in by the received message is new to} \\ \text{the node } i, \text{ and the local knowledge } \mathcal{K}_i \text{ is not new to the sender,} \\ \text{i.e., } W_i \leq W \text{ and } \mathcal{K} \supset \mathcal{K}_i; \end{array} \\ \text{false} & \begin{array}{l} \text{the information } \mathcal{K} \text{ brought in by the received message is exactly} \\ \text{the same as the local knowledge } \mathcal{K}_i, \\ \text{i.e., } W_i = W \text{ and } \mathcal{K} = \mathcal{K}_i. \end{array} \end{cases}$$

The purpose of feedback message is to construct a forest for reset, which will be described in later in this chapter. Every operational processor keeps track of the information on its adjacent nodes so that all the selected links form a forest.

These information is captured by the following variables:

- $pred(i)$: the identity of the node from which node i has received a message and obtained the current topological information. This node is called as a *predecessor*.
- $succ(i)$: a sub-set of neighbors of node i to which it sends information messages, and $Adjacent(i) \supseteq succ(i)$. The nodes in this set are called as *successors*.
- $resp(i)$: a subset of $succ(i)$ from which node i has received a feedback message with the correct wave number, and $\mathcal{A} = \text{true}$.

Before we present the wave rules, we define the notations associated with the knowledge. The definitions for the notations are given as follows:

- (i) *Local Consistency* ($LC_i(\mathcal{K})$): With respect to any local topology ($LocalTopology_i$), the topological knowledge $\mathcal{K}$ brought in by the received message is consistent with the topological knowledge $\mathcal{K}_i$ known to node i ; that is

$$LC_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigwedge \mathcal{K}[i, j] = \mathcal{K}_i[i, j]$$

- (ii) *Global Consistency* ($GC_i(\mathcal{K})$): With respect to the status of any link in the network, $\mathcal{K}$ is consistent with $\mathcal{K}_i$; that is

$$GC_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigwedge \mathcal{K}[v, u] = \mathcal{K}_i[v, u]$$

- (iii) *Local Inconsistency* ($LI_i(\mathcal{K})$): With respect to some local topology ($LocalTopology_i$), $\mathcal{K}$ is inconsistent with $\mathcal{K}_i$; that is

$$LI_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigvee \mathcal{K}[i, j] \neq \mathcal{K}_i[i, j]$$

- (iv) *Remote Inconsistency* ($RI_i(\mathcal{K})$): With respect to the status of remote links which is not incident to i (i.e., the status of the link is not local topological information), $\mathcal{K}$ is inconsistent with $\mathcal{K}_i$; that is

$$RI_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigvee_{v, u \neq i} \mathcal{K}[v, u] \neq \mathcal{K}_i[v, u]$$

With respect to $RI_i(\mathcal{K})$, the ambiguous topological information for node i must be remote (non local) to i , that is

$$RemoteAmbiguity_i = \{ RA_i[v, u] \}$$

$$\text{where } RA_i[v, u] = \begin{cases} 2 \text{ (ambiguous)} & \text{where } RI_i(\mathcal{K}) \text{ holds for } v \text{ and } u, \\ \text{Undefined} & \text{Otherwise.} \end{cases}$$

Clearly, $LC_i(\mathcal{K}) = \neg LI_i(\mathcal{K})$, $GC_i(\mathcal{K}) = \neg (LI_i(\mathcal{K}) \vee RI_i(\mathcal{K}))$, and

$$LocalTopology_i \cap RemoteAmbiguity_i = \emptyset.$$

Notice that, for notations (iii) and (iv), either v or u might be equal to i . While *Local Consistency* ($LC_i(\mathcal{K})$) is implied by *Global Consistency* ($GC_i(\mathcal{K})$), the converse is clearly not true. Hereafter, $LC_i(\mathcal{K})$, $GC_i(\mathcal{K})$, $LI_i(\mathcal{K})$ and $RI_i(\mathcal{K})$ are used to represent the above notations, respectively.

Manipulation of topological information is a *dyadic* operation, which is denoted by $//$. Both operands are the knowledge sets. The set on the left-hand of the symbol is updated with respect to the set on the right-hand of the symbol.

Given two knowledge sets $\mathcal{K}_1$ and $\mathcal{K}_2$, the operation $\mathcal{K}_1 // \mathcal{K}_2$ represents the updating of $\mathcal{K}_1$ with respect to $\mathcal{K}_2$; that is

$$\mathcal{K}_1 // \mathcal{K}_2 \stackrel{\text{def}}{=} (\mathcal{K}_1 - \mathcal{K}_2) \cup \mathcal{K}_2.$$

3.3 The Wave Rules

In order to broadcast and maintain topological information in a dynamic system, several wave rules have been designed for the stabilizing topology broadcast wave algorithm. They are three different types: *Making Waves*, *Propagating Waves*, and

Eliminating Waves. These rules are aimed for governing how the topological information should be updated at each node.

Following is a brief description of the three types of rules.

Making Waves

This wave rule determines how to initiate new waves in response to the *events* such as detection of topological changes and reception of information messages in the network. Since, as mentioned before, a node failure can be simulated by the failures of all its adjacent links, all the topological changes are represented by the changes of the links status.

Propagating Waves

Upon receipt of the information messages, the processors update and broadcast the information by applying these rules.

Eliminating Waves

It is aimed to eliminate the “outdated” information and reduce the number of messages.

Association with each rule, there is a condition (i.e., a predicate). It formally and precisely describe the scenario to which the rule can be applied.

In the rest of this section, we present the more detail description of the three types of rules and exhibit the behavior of a processor, say i , in a dynamic distributed system governed under these rules:

Making Waves

Rule 3.1.1 When a node detects a change of status in one of the links incident to it, (i.e., the link goes down or comes up), it updates the local information and starts a new wave to broadcast this new information.

Condition 3.1.1

$$(\exists j \mid (\mathcal{K}_i[i,j] = 1 \wedge j \notin \text{Adjacent}(i)) \vee (\mathcal{K}_i[i,j] = 0 \wedge j \in \text{Adjacent}(i)))$$

As mentioned in Section 3.2.1, $\text{Adjacent}(i)$ is a set of the identities of the current operational adjacent nodes maintained at node i . By definition, the changes in the status of the links incident to i are reflected in $\text{Adjacent}(i)$ immediately:

- (a) when link (i,j) fails, j will not be in $\text{Adjacent}(i)$; but it was in $\text{Adjacent}(i)$ before the failure, and therefore $\mathcal{K}_i[i,j] = 1$ (up).
- (b) when link (i,j) becomes operational, j will be in $\text{Adjacent}(i)$; but it was not in $\text{Adjacent}(i)$ before the recovery, and therefore $\mathcal{K}_i[i,j] = 0$ (down).

This rule (*Making Waves*) covers the the following five situations:

- (i) A link fails;
- (ii) A link becomes operational, that is a failed link has been repaired;
- (iii) A node fails which results in the failures of all the adjacent links of the node;
- (iv) A failed node becomes operational, that is a crashed processor has been repaired;
- (v) The system starts.

In situations (iv) and (v), when the system starts up, or a node becomes operational, the operational nodes drive the system toward self stabilization. It resets the status of the links incident to it in its local topological information immediately, as described in Section 3.3. Then this condition (3.1.1) triggers the node to initiate a new wave that broadcasts the topological information.

Rule 3.1.2 Upon receipt of an information message, the node compares the topological knowledge $\mathcal{K}$ brought in by the received message and the local topological information $\mathcal{K}_i$. If one of three following conditions holds:

- (i) the message wave number is higher than the local wave number at the node ($W_i < W$),
and $\neg((\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K}))$;
- (ii) the message wave number is same as the local wave number at the node ($W_i = W$),
and $(\neg((\mathcal{K}_i \supseteq \mathcal{K}) \vee (\mathcal{K} \supset \mathcal{K}_i)) \wedge GC_i(\mathcal{K})) \vee LI_i(\mathcal{K})$;
- (iii) the message wave number is lower than the node wave ($W_i > W$),
and $\neg(\mathcal{K}_i \supseteq \mathcal{K}) \wedge GC_i(\mathcal{K})$.

It updates its local topological knowledge $\mathcal{K}_i$ with respect to the current local topological information (i.e., $LocalTopology_i$) and the received information $\mathcal{K}$. Moreover, it propagates the updated topological information by creating a new wave $\max(W_i, W) + 1$.

Condition 3.1.2

$$\begin{aligned}
 & ((W_i < W) \wedge \neg((\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K}))) \vee \\
 & ((W_i = W) \wedge ((\neg((\mathcal{K}_i \supseteq \mathcal{K}) \vee (\mathcal{K} \supset \mathcal{K}_i)) \wedge GC_i(\mathcal{K})) \vee LI_i(\mathcal{K}))) \vee \\
 & ((W_i > W) \wedge \neg(\mathcal{K}_i \supseteq \mathcal{K}) \wedge GC_i(\mathcal{K}))
 \end{aligned}$$

To explain this rule, we show how the information is managed and stored in some situations covered by Rule 3.1.2.

- (i) Due to the arbitrary initial state of the system or some very frequent remote topological changes, the wave number of the received messages from the remote node are higher than the local one. Thus, it is possible for a node to receive a message containing some local information with a higher wave number ($W_i < W$) which is inconsistent (i.e., $LI_i(\mathcal{K})$). This is one of the situations handled by the first sub-condition which triggers the node to create a new wave for clarifying the ambiguity on the local topological information .
- (ii) The second sub-condition is to handle the topological information $\mathcal{K}$ brought in by the received message with the same wave number ($W_i = W$), and there exists some local inconsistent information (i.e., $LI_i(\mathcal{K})$), or the received message contains the information $\mathcal{K}$ which satisfies the predicate

$$\neg ((\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K})), \text{ and } GC_i(\mathcal{K}).$$
- (iii) The information $\mathcal{K}$ brought in by a message with a lower wave number ($W_i > W$) will be merged with the local information $\mathcal{K}_i$ creating a new wave to propagate the accumulated information, if the information $\mathcal{K}$ is not known to the receiver (node i), and $\mathcal{K}$ is consistent with the local information $\mathcal{K}_i$ (i.e., $GC_i(\mathcal{K})$). Thus, the system can stabilize more efficiently.

Propagating Waves

Rule 3.1.3 After receiving an information message and comparing its own knowledge $\mathcal{K}_i$ with the received one $\mathcal{K}$, if the node detects one of these two conditions holds:

- (i) the received message has a higher wave number ($W_i < W$) and
 $((\mathcal{K} \supset \mathcal{K}_i) \vee (\mathcal{K}_i \supseteq \mathcal{K}));$ or
- (ii) $(W_i = W) \wedge ((\mathcal{K} \supset \mathcal{K}_i) \vee (LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K})));$

then it continues to propagate the larger knowledge set with the wave number $W = \max(W_i, W)$ to all its adjacent nodes.

Condition 3.1.3

$$((W_i < W) \wedge ((\mathcal{K} \supset \mathcal{K}_i) \vee (\mathcal{K}_i \supseteq \mathcal{K}))) \vee \\ (W_i = W) \wedge ((\mathcal{K} \supset \mathcal{K}_i) \vee (LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K})))$$

If the following condition holds $(W_i < W) \wedge (\mathcal{K}_i \supseteq \mathcal{K})$, it implies that the node already possesses the information brought in by the received message with a higher wave, so it only has to upgrade its wave number to the wave number of the received message W and to propagate its local information with the upgraded wave number.

If the local information is a subset of the information brought in by the received message $(\mathcal{K} \supset \mathcal{K}_i)$, and the message wave number is not lower than the local wave number $(W_i \leq W)$, then the receiver updates its local knowledge $\mathcal{K}_i$ to be the received information $\mathcal{K}$, and sets its local wave number to be W . Further, it propagates the new local information with the wave number W .

If the wave number of the received message is the same as the local wave number $(W_i = W)$, and there exists remote inconsistency between the information brought in by the

received message and the local topological information (i.e., $LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K})$), then the node updates its local knowledge with respect to its local topology and sets the status of the remote inconsistent information to be ambiguous. Further, it propagates the updated local information with the same wave number. During the propagation of the message contains the information with some ambiguous status, there are two scenarios:

- (a) It must be inconsistent to some operational nodes with respect to their local topological information ($LocalTopology_i$), that is $LC_i(\mathcal{K})$. Thus, it results in the nodes to initiate a new wave for clarifying their local information (Rule 3.1.2).
- (b) Otherwise, there must be messages with a higher wave number in transit on some links of the network. Since the ambiguous information is not inconsistent with any operational nodes, it indicates that the nodes in scenario (a) must have been down. Therefore, some new waves must be created to broadcast the topological changes (Rule 3.1.1).

Eliminating Waves

Rule 3.1.4 When a node receives a message, it will discard the message. If one of these two situations occurs:

- (i) the wave number W is not higher than the wave number W_i at node i ($W_i \geq W$), and no new topological information is received ($\mathcal{K}_i \supseteq \mathcal{K}$);
- (ii) ($W_i > W$) and there exists $LI_i(\mathcal{K})$ or $RI_i(\mathcal{K})$;

Condition 3.1.4

$$((W_i \geq W) \wedge (\mathcal{K}_i \supseteq \mathcal{K})) \vee ((W_i > W) \wedge (LI_i(\mathcal{K}) \vee RI_i(\mathcal{K})))$$

For situation (ii), since there exists some inconsistent information between $\mathcal{K}_i$ and $\mathcal{K}$, there must be a wave with a higher wave number that has been initiated for clarifying this ambiguity (Rule 3.1.2). Therefore, the messages under Condition 3.1.4 should be eliminated.

With respect to all the above rules, let us examine the predicates of all the conditions:

(i) $(W_i < W)$:

$$\neg((\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K})) \quad < \text{Condition 3.1.2.} >$$

$$(\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K}) \quad < \text{Condition 3.1.3.} >$$

(ii) $(W_i = W)$:

$$(\neg((\mathcal{K}_i \supseteq \mathcal{K}) \vee (\mathcal{K} \supset \mathcal{K}_i)) \wedge GC_i(\mathcal{K})) \vee LI_i(\mathcal{K}) \quad < \text{Condition 3.1.2.} >$$

$$(\mathcal{K} \supset \mathcal{K}_i) \vee (LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K})) \quad < \text{Condition 3.1.3.} >$$

$$(\mathcal{K}_i \supseteq \mathcal{K}) \quad < \text{Condition 3.1.4.} >$$

(iii) $(W_i > W)$:

$$\neg(\mathcal{K}_i \supseteq \mathcal{K}) \wedge GC_i(\mathcal{K}) \quad < \text{Condition 3.1.2.} >$$

$$(\mathcal{K}_i \supseteq \mathcal{K}) \vee (LI_i(\mathcal{K}) \vee RI_i(\mathcal{K})) \quad < \text{Condition 3.1.4.} >$$

Note that $GC_i(\mathcal{K}) = \neg(LI_i(\mathcal{K}) \vee RI_i(\mathcal{K}))$.

It is not difficult to verify that these predicates cover all the possible situations.

3.4 The Wave Algorithm

In this section, we will formally describe the algorithm. Before doing so, we will depict* its salient features introduced in some details in the previous section with respect to *the different types of events* (see Figure 3.1).

Figure 3.1 The Events and Actions in The Stabilizing Wave Algorithm

Detection of topological changes

Initially, every operational node in the network starts the system by initiating a new wave to broadcast the topological information messages to all the operational nodes,

* For visualization, see appendix

according to Rule 3.1.1 (*Making Waves*). These waves drive the system toward self stabilization. Our self stabilizing algorithm does not depend on the correct initialization: i.e., its correctness is insensitive to the initial states of the processors and to the initial messages already in transit on the links. For example, as will be shown later, the arbitrary initial wave numbers at the operational nodes will be stabilized, and the inconsistent initial topological information will be corrected eventually.

After the system started, if there are any topological changes such as the addition and deletion of nodes or links, new waves are created based on the same rule (Rule 3.1.1). This implies that our stabilizing wave algorithm is event driven.

In general, whenever a node, say i , detects a topological change, it updates the local information $\mathcal{K}_i$ with respect to the local topology ($LocalTopology_i$) and initiates a new wave to broadcast the updated information. With $\mathcal{K}_i$, there are the following scenarios for both the arbitrary initial states and non-initial states:

- (i) $\mathcal{K}_i = \emptyset$, which indicates that, for node i , it starts from scratch;
- (ii) $\mathcal{K}_i \neq \emptyset$, in this case, the local knowledge $\mathcal{K}_i$ is twofold:
 - (a) If the information $\mathcal{K}_i$ is “outdated”, it will be corrected in a finite time,
 - (b) If the information in $\mathcal{K}_i$ is correct, then it can speed up the process of the system toward the stabilization. For example, in the end of one burst of the topological changes, if only one or a few topological changes occur, it is unnecessary for *all* the operational nodes in the network to restart from the beginning.

Reception of an information message

An operational node, say i , compares the topological knowledge $\mathcal{K}$ brought in by the received message with its own local knowledge $\mathcal{K}_i$. As a result, it will take one of the following actions:

- (i) Update its local topological knowledge with respect to the local link status (*LocalTopology*) and make a new wave to broadcast the accumulated information or to clarify local knowledge inconsistency $LI_i(\mathcal{K})$ (Rule 3.1.2);
- (ii) Propagate a wave (Rule 3.1.3);
- (iii) Eliminate a wave (Rule 3.1.4).

During the execution of the protocol, a node accumulates topological knowledge about the network from the messages containing the information which is consistent with its local links status (i.e., certification of information), and it continues to broadcast or propagate the information (Rules 3.1.2 and 3.1.3).

Due to the arbitrary initial state of the system and the nature of a dynamic network (as described in Chapter 2), a node can also receive the information which is inconsistent or “outdated” with respect to its local links’ status. The “outdated” information (i.e., contained in a message with a lower wave number and such that $((\mathcal{K}_i \supsetneq \mathcal{K}) \vee (LI_i(\mathcal{K}) \vee RI_i(\mathcal{K})))$ will be discarded (Rule 3.1.4). If the discarded information is correct, the stabilizing wave algorithm guarantees that the messages containing the correct information will eventually arrive at each node within a finite time according to Rule 3.1.3.

Reception of a feedback message

After sending messages to its neighbors, a node normally waits for the feedback messages $\langle W, \mathcal{A} \rangle$. Upon receipt of feedback messages from one of its adjacent nodes, it checks the wave number with its own local wave number. If they do not match, which implies that the feedback message is outdated. Otherwise, it checks the type of the feedback message $\mathcal{A}$ (described in Section 3.2). Based on the feedback message with the same wave number as the local one, a node captures the information and selects a subset of its adjacent links so that all the selected links form a forest among the operational nodes which will be used for reset. When a node receives all the feedback messages from its successors (this is implemented by using a variable $resp(i)$ to keep track of the feedback messages, as stated in Section 3.2.), it sends a feedback message back to its predecessor, provided it has a predecessor. In some cases, for instance, the node initiates a higher wave to broadcast new information or to clarify an ambiguity, there is no feedback messages for the received messages.

As stated in Chapter 2, the sequence of topological changes is finite or infinitely discrete. Eventually, the initiators of the last wave covering all the operational nodes in the network receive all the feedback messages from their neighbors respectively. Based on the captured information, they can determine whether or not the last wave for broadcasting the topological knowledge has been completed.

We will now present the algorithm. Once the system starts, processors execute this program as an underlying computation for topology broadcast and self stabilization.

The formal presentation of the self stabilizing wave algorithm is as follows:

Topological Change

(Condition 3.1.1) /* Event : A topological change occurs */

/* where $(\exists j \mid (\mathcal{K}_i[i,j] = 1 \wedge j \notin \text{Adjacent}(i)) \vee (\mathcal{K}_i[i,j] = 0 \wedge j \in \text{Adjacent}(i))) \vee \text{Making Waves} */$

$\Rightarrow$ $W_i := W_i + 1;$
 $\mathcal{K}_i := \mathcal{K}_i // \text{LocalTopology}_i;$
 $\text{succ}(i) := \text{Adjacent}(i); \text{pred}(i) := \emptyset; \text{resp}(i) := \emptyset;$
for all $u \in \text{Adjacent}(i)$ **do** send $\langle W_i, \mathcal{K}_i \rangle$ to u **od**

Upon receipt of $\langle W, \mathcal{K} \rangle$ from v

[] (Condition 3.1.2) /* Upon receipt of $\langle W, \mathcal{K} \rangle$ from v */

/* where $((W_i < W) \wedge \neg((\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K}))) \vee$
 $((W_i = W) \wedge ((\neg((\mathcal{K}_i \supseteq \mathcal{K}) \vee (\mathcal{K} \supset \mathcal{K}_i)) \wedge GC_i(\mathcal{K})) \vee LI_i(\mathcal{K}))) \vee$
 $((W_i > W) \wedge \neg(\mathcal{K}_i \supseteq \mathcal{K}) \wedge GC_i(\mathcal{K}));$ Making Waves */

$\Rightarrow$ $W_i = \max(W_i, W) + 1;$
 $\mathcal{K}_i := \mathcal{K}_i // \mathcal{K} // \text{LocalTopology}_i;$
 $\text{succ}(i) := \text{Adjacent}(i); \text{pred}(i) := \emptyset; \text{resp}(i) := \emptyset;$
for all $u \in \text{Adjacent}(i)$ **do** send $\langle W_i, \mathcal{K}_i \rangle$ to u **od**

[] (Condition 3.1.3) /* Upon receipt of $\langle W, \mathcal{K} \rangle$ from v */

/* where $((W_i < W) \wedge ((\mathcal{K} \supset \mathcal{K}_i) \vee (\mathcal{K}_i \supseteq \mathcal{K}))) \vee$
 $(W_i = W) \wedge ((\mathcal{K} \supset \mathcal{K}_i) \vee (LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K})))$; Propagating Waves */

$\Rightarrow$ $\text{succ}(i) := \text{Adjacent}(i); \text{pred}(i) := \emptyset; \text{resp}(i) := \emptyset;$
if $(W_i \leq W) \wedge (\mathcal{K} \supset \mathcal{K}_i)$
 $\Rightarrow \mathcal{K}_i := \mathcal{K}; \text{succ}(i) := \text{succ}(i) - \{v\}; \text{pred}(i) := v;$
if $\text{succ}(i) = \emptyset \Rightarrow$ send $\langle W, \text{true} \rangle$ to $\text{pred}(i)$ **fi**
[] $(W_i = W) \vee (LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K}))$
 $\Rightarrow \mathcal{K}_i := \mathcal{K}_i // (\mathcal{K} // \text{RemoteAmbiguity}_i) // \text{LocalTopology}_i;$
fi;


```

     $W_i := W;$ 
    for all  $u \in succ(i)$  do send  $\langle W_i, \mathcal{K}_i \rangle$  to  $u$  od;

    [] (Condition 3.1.4 ) /* Upon receipt of  $\langle W, \mathcal{K} \rangle$  from  $v$  */
    /* where  $((W_i \geq W) \wedge (\mathcal{K}_i \supseteq \mathcal{K})) \vee$ 
        $((W_i > W) \wedge (LI_i(\mathcal{K}) \vee RI_i(\mathcal{K})))$ ; Eliminating Waves */

     $\Rightarrow$  if  $(W_i = W) \wedge (\mathcal{K} = \mathcal{K}_i)$ 
         $\Rightarrow$  send  $\langle W, \text{false} \rangle$  to  $v$ ;
        if  $v \in succ(i) \Rightarrow succ(i) := succ(i) - \{v\}$  fi;
    fi;

```

Upon receipt of $\langle W, \mathcal{A} \rangle$ from v

```

    if  $(W_i = W)$ 
     $\Rightarrow$  if  $(\mathcal{A} = \text{true}) \Rightarrow resp(i) := resp(i) \cup \{v\};$ 
        []  $(\mathcal{A} = \text{false}) \Rightarrow succ(i) := succ(i) - \{v\};$ 
    fi;
    if  $(resp(i) = succ(i)) \wedge (pred(i) \neq \emptyset)$ 
     $\Rightarrow$  send  $\langle W, \text{true} \rangle$  to  $pred(i)$ ; /* the node becomes a root of a sub-tree */
    []  $(resp(i) = succ(i)) \wedge (pred(i) = \emptyset)$ 
     $\Rightarrow$  reset-wave( $i$ ); /* For model  $\Phi_{\sigma}^{\infty}$ , reset-wave procedure is invoked. */
    [] else  $\Rightarrow$  skip;
    fi;
[] else  $\Rightarrow$  skip; /*  $W_i \neq W$  */
fi

```

The Stabilizing Topology Wave Algorithm

3.5 Proof of Correctness

Before proving the correctness of the algorithm, we will introduce some terminologies and definitions which will be used in this section.

Consistent set : At time t , a set of nodes in which every processor has correct and consistent information of the local topology of all processors in the set.

Completion of the algorithm : A global system state in which no processor is executing any step and no messages are in transit.

To prove the correctness of the algorithm presented in Section 3.4, we begin with some technical lemmas.

Lemma 3.1 *Given two operational nodes, i and j , connected by an up link, without loss of generality, let $W_i \geq W_j$. If $W_i > W_j$, then*

- (i) *there is a message in transit from i to j on the link (i,j) with wave number W_i ; and*
- (ii) *if the link (i,j) does not fail before the message arrives, the wave number of j upon processing of the message will be greater than or equal to W_i ($W_j \geq W_i$).*

Proof. Part (i). We will consider two cases depending on whether or not at least one of the two node is (re)starting.

If one of the nodes is (re)starting, by definition, then the link (i, j) must be now coming up, thus, both i and j will send a message to all their operational neighbors.

If neither node is (re)starting; then, since $W_i > W_j$, node i increases its wave number only in these two situations: either after detecting a local change or after receiving a message. In the former situation, it sent a message with the new wave number W_i to all its operational neighbors including j (Rule 3.1.1); because the wave number of node j is less than the

wave number at node i ($W_j < W_i$). j has obviously not received such a message which is therefore in transit from i to j on link (i, j) . In the latter situation, node i sent the message with wave number to all its operational neighbors (Rule 3.1.2); the only exception would be that if the received message had the information such that $(W_i < W)$ and $(\mathcal{K} \supset \mathcal{K}_i)$, then node i propagated the information $\mathcal{K}$ to all its operational neighbors except the sender of the message. Clearly, the sender is not node j , since $W_j < W_i$.

Part (ii). Clearly, based on the proof of (i), if the link (i, j) does not fail before the message from node i arrives at node j , if $W_i > W_j$ still holds, node j takes action according to Rule 3.1.2 or 3.1.3, and updates its wave number to a value $W_j \geq W_i$. $\square$

Notice that, due to the communication delay, two nodes at both sides of a recovered link may not act simultaneously.

Lemma 3.2 *If any two operational nodes have different wave numbers, then there exists at least one message, whose wave number is greater than or equal to the higher one, in transit on one of the operational links in the network.*

Proof. By definition, all the operational nodes in the network are connected (i.e., the communication graph is connected). By repeated application of Lemma 3.1 (i), the lemma follows. $\square$

Lemma 3.3 *Given an arbitrary time t , let W_{max} be the highest wave number among the operational nodes. Then, there exists a delay Δ_{tw} such that, if no topological changes occur in the interval $(t, t + \Delta_{tw})$, then at time $t + \Delta_{tw}$, for any every operational node v will have a wave number $W_v \geq W_{max}$.*

Proof. Assume that topological changes cease forever, we will show that, in this case, this lemma holds. We will then show that it is sufficient that topological changes cease for

just a finite time, thus proving that the lemma holds also for the case in which the sequence of topological changes is infinitely discrete.

Let k be a processor with the highest wave number (i.e., $W_k = W_{max}$) among all the operational nodes at time t , by Lemma 3.1 (i), there exist messages with wave number W_k in transit on the operational links from k to all its operational neighbors with a smaller wave numbers. Let Δ_{w1} be the longest communication delay experienced by the messages from k to all its operational neighbors with a smaller wave numbers. Since no topological changes occur, then, after a finite time Δ_{w1} , the wave numbers of all the operational neighbors of k will be greater than or equal to W_{max} , by Lemma 3.1 (ii).

By repeating the above argument and the application of Lemma 3.2, since the graph $G=(V, E)$ is connected, and each link is bidirectional. Let Δ_{wi} be the longest communication delay experienced by the messages in transit between the nodes at distance $i-1$ and the nodes at distance i from k (see Figure 3.2). By applying the same argument as for the case $i=1$, within Δ_{wi} additional time, all operational neighbors of k at distance less than or equal to i will have a wave number which is greater than or equal to W_{max} .

Figure 3.2 The Communication Delay in an Asynchronous System

Then, at time $t + \Delta_{tw}$, every operational node v will have a wave number ($W_v \geq W_{max}$),

where
$$\Delta_{tw} = \Delta_{w1} + \Delta_{w2} + \Delta_{w3} + \dots + \Delta_{wd} = \sum_{i=1}^d \Delta_{wi}, \quad (3.5.1)$$

and d is the current diameter of the network, Δ_{wi} is the longest communication delay experienced by the messages in transit between the nodes at distance $i-1$ and the nodes at distance i from k .

In other case, since Δ_{tw} is finite for any t , for the lemma to hold, it is sufficient that topological changes cease for a finite amount of time Δ , provided $\Delta > \Delta_{tw}$. Thus, the lemma holds also for infinitely discrete case. $\square$

Lemma 3.4 *Let i and j be two operational nodes connected by an up link. At an arbitrary time t , there exists a delay Δ_{LL} such that, if no topological changes occur in the interval $(t, t + \Delta_{LL})$, then at time $t + \Delta_{LL}$, both i and j will have the correct information on the local topology of i and j .*

Proof. We will consider two cases depending on whether $W_j \neq W_i$ or $W_j = W_i$.

Case 1: Let the wave numbers of node i and j be different ($W_j \neq W_i$), without loss of generality, let $W_i > W_j$. By Lemma 3.1, there exists a message with W_i from i to j in transit on link (i, j) . Let Δ_{L1} be the delay experienced by the message in transit from i to j . If there are no topological changes, the message will arrive after a finite time delay Δ_{L1} . Upon receipt on the message from i with information $\mathcal{K}$ (where $\mathcal{K} = \mathcal{K}_i$), node j compares $\mathcal{K}$ with its local information $\mathcal{K}_j$. If $\neg((\mathcal{K} \supseteq \mathcal{K}_j) \vee (\mathcal{K}_j \supseteq \mathcal{K}))$ holds, node j updates its local information with respect to $\mathcal{K}_i$ and $LocalTopology_j$. As a result, any received information which is inconsistent with the local topology of j ($LocalTopology_j$) will be clarified by the new wave initiated by j , by Rule 3.1.2; that is

(Condition 3.1.2)

$$(W_j < W) \wedge \neg ((\mathcal{K} \supseteq \mathcal{K}_j) \vee (\mathcal{K}_j \supseteq \mathcal{K}))$$

$$\Rightarrow W_j = \max(W_j, W) + 1; \mathcal{K}_j := \mathcal{K}_j \parallel \mathcal{K} \parallel LocalTopology_j;$$

for all $u \in Adjacent(i)$ do send $\langle W_j, \mathcal{K}_j \rangle$ to u od ;

Let Δ_{L2} be the delay experienced by the message from j and i . If still there are no topological changes, this message will arrive at i within a finite delay Δ_{L2} . Thus, within a finite time Δ_{tL} ($\Delta_{tL} \geq \Delta_{L1} + \Delta_{L2}$), both i and j have the correct information on $LocalTopology_i$ and $LocalTopology_j$.

Case 2: Let the wave numbers of node i and j be the same ($W_i = W_j$), there are two scenarios:

- (a) Both i and j have the correct information on $LocalTopology_i$ and $LocalTopology_j$, the lemma holds.
- (b) If at least one of i and j does not have the correct information on $LocalTopology_i$ and $LocalTopology_j$, we show that there are messages in transit which will correct it.

Figure 3.3 Topological changes in two adjacent operational nodes

Without loss of generality, let us examine $\mathcal{K}_i$ and $\mathcal{K}_j$ with respect to $LocalTopology_i$ and $LocalTopology_j$. If there exists inconsistent information on (j, w) between $\mathcal{K}_i$ and $\mathcal{K}_j$ (see Figure 3.3). Note that the ambiguities could exist symmetrically.

The only situations in which both i and j have the same wave numbers ($W_i = W_j$) but inconsistent information are the following:

- The initial state of the system;
- Both i and j received information messages containing information on link (j,w) .
(Clearly, the received messages were not from the same initiators, since $\mathcal{K}_i[j,w] \neq \mathcal{K}_j[j,w]$;
- Node i received a message containing information on link (j,w) , and the status of (j,w) changed at the same time. Therefore, both i and j increased their wave numbers.

It is not difficult to see that, in any one of the these scenarios, according to Rules 3.1.1, 3.1.2, and 3.1.3, there must exist at least one message in transit between j and i on link (i, j) . Since the information is remote topological information for node i , thus, node i either sends a request (by updating the status of the link (j,w) to be ambiguous) or propagates the information to all its operational neighbors including j , according to the Rule 3.1.3, that is

(Condition 3.1.3)

$$((W_i < W) \wedge ((\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K}))) \vee$$

$$((W_i = W) \wedge ((\mathcal{K} \supset \mathcal{K}_i) \vee (LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K}))))$$

$$\Rightarrow \text{ if } (W_i \leq W) \wedge (\mathcal{K} \supset \mathcal{K}_i)$$

$$\Rightarrow \mathcal{K}_i := \mathcal{K};$$

$$[] (W_i = W) \vee (LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K}))$$

$$\Rightarrow \mathcal{K}_i := \mathcal{K}_i // (\mathcal{K} // RemoteAmbiguity_i) // LocalTopology_i ;$$

fi;

$$W_i := W;$$

for all $u \in succ(i)$ **do** send $\langle W_i, \mathcal{K}_i \rangle$ to u **od**

Upon receipt of this message from i , node j initiates a new wave to clarify the ambiguity, according to the Rule 3.1.2 (see Figure 3.4), that is

(Condition 3.1.2)

$(W_j = W) \wedge LI_j(K)$

$\Rightarrow W_i = \max(W_i, W) + 1;$

$K_j := K_j // K // LocalTopology_i;$

for all $u \in Adjacent(j)$ **do** send $\langle W_j, K_j \rangle$ to u **od**

Figure 3.4 The Process of Remote Inconsistent Information

Assume that no failures occur during the entire process, then within a finite time Δt , node i will correct the ambiguity about (j,w) . The same process will actually resolve all mutual ambiguities between i and j ; that is both i and j have the correct information on $LocalTopology_i$ and $LocalTopology_j$. Hence, the lemma also holds for the case in which the wave numbers of i and j are the same ($W_i = W_j$).

This now completes the proof of Lemma 3.4. $\square$

From Lemma 3.4, we notice that the process of clarifying the ambiguities with respect to the local topological information. To a processor, it actually is a *certification* process with respect to its local topology.

Theorem 3.1 *For any arbitrary time t , there exists a delay Δ_{tc} such that, if no topological changes occur in the interval $(t, t + \Delta_{tc})$, then at time $t + \Delta_{tc}$, every operational node v will have the correct topological information on the entire network.*

Proof. As the proof of Lemma 3.3, we will firstly assume that topological changes cease forever and will show that, in this case, the theorem holds. Then, we will prove that it is sufficient that topological changes cease for just a finite time, the theorem holds also for the case in which the sequence of topological changes is infinitely discrete.

Assume that topological changes cease forever. By Lemma 3.4, at any arbitrary time t , then there exists a finite delay Δ_{tl} such that, if no topological changes occur in the interval $(t, t + \Delta_{tl})$, two incident nodes connected by an operational link will have the correct information on their local topology. Let k be a processor with the highest wave number (i.e., $W_k = W_{max}$) among all the operational nodes at time t , and let Δ_{k1} be the longest communication delay experienced by the messages in transit between k and all its operational neighbors $N(k)$. By Lemma 3.4, within a finite time Δ_{k1} , all the operational neighbors of k (including k) will have the correct information on their local topological information *LocalTopology*. By definition, all the operational neighbors of k (including k) is a *consistent set* (see Figure 3.5). Let it be called S_1 , which can be viewed a “big” node.

Figure 3.5 The Consistent Set in the Network

By repeating the above argument, for all the operational nodes at the distance of i to node k , that is all the operational neighbors at the distance of $i-1$ to S_i , let Δ_{k_i} be the longest communication delay experienced by the messages in transit between S_i and all its operational neighbors $N(S_i)$ (see Figure 3.5), during this period of time, $N(S_i)$ and S_i become a new consistent set S_{i+1} (i.e., $S_i \cup N(S_i) \Rightarrow S_{i+1}$), then at time $t + \Delta_{k1} + \dots + \Delta_{k_i}$, all the operational nodes in S_{i+1} will have the correct information on their local topological information. Thus, by the time $(t + \Delta_{tc})$, all the operational nodes will be in the same consistent set S_d , that is the certification process of all the operational nodes in the network will be completed, where

$$\Delta_{tc} = \Delta_{k1} + \Delta_{k2} + \Delta_{k3} + \dots + \Delta_{kd} = \sum_{i=1}^d \Delta_{k_i} \quad (3.5.2)$$

and d is the current diameter of the network, every operational node v will have correct information on the entire network. Thus, the theorem holds for the case in which topological changes cease forever.

In other case, since Δ_{tc} is finite for any t , it is sufficient that topological changes cease for a finite amount of time Δ , provided $\Delta \geq \Delta_{tc}$. Hence, the theorem holds also for infinitely discrete case. $\square$

Now we know that, within a finite amount of time, all the operational nodes will obtain the correct topological information on the entire network, if topological changes cease.

Theorem 3.2 (No Deadlock) *There is no deadlock for the stabilizing wave algorithm.*

Proof. According to Rules 3.1.2, 3.1.3 and 3.1.4, we have the following predicates:

(i) $(W_i < W)$:

$$\neg((\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K})) \quad < \text{Condition 3.1.2} >$$

$$(\mathcal{K} \supseteq \mathcal{K}_i) \vee (\mathcal{K}_i \supset \mathcal{K}) \quad < \text{Condition 3.1.3} >$$

(ii) $(W_i = W) :$

$$(\neg((\mathcal{K}_i \supseteq \mathcal{K}) \vee (\mathcal{K} \supset \mathcal{K}_i)) \wedge GC_i(\mathcal{K})) \vee LI_i(\mathcal{K}) \quad < \text{Condition 3.1.2.} >$$

$$(\mathcal{K} \supset \mathcal{K}_i) \vee (LC_i(\mathcal{K}) \wedge RI_i(\mathcal{K})) \quad < \text{Condition 3.1.3.} >$$

$$(\mathcal{K}_i \supseteq \mathcal{K}) \quad < \text{Condition 3.1.4.} >$$

(iii) $(W_i > W) :$

$$\neg(\mathcal{K}_i \supseteq \mathcal{K}) \wedge GC_i(\mathcal{K}) \quad < \text{Condition 3.1.2.} >$$

$$(\mathcal{K}_i \supseteq \mathcal{K}) \vee (LI_i(\mathcal{K}) \vee RI_i(\mathcal{K})) \quad < \text{Condition 3.1.4.} > .$$

$$\text{Note that } GC_i(\mathcal{K}) = \neg(LI_i(\mathcal{K}) \vee RI_i(\mathcal{K})).$$

It is not difficult to verify that these predicates cover all the possible cases. Each of these predicates results in a processor to take the corresponding actions, upon receipt of an information message. Thus, the operational processors never stop processing the received messages; in other words, there are no blocked messages. At the same time, since the algorithm is event driven, the processors are never blocked waiting for messages.

If there are no messages in transit in the system, by definition, the algorithm completes. $\square$

Finally what we have to prove is that the algorithm can not keep on generating messages forever, in the case that there are no more topological changes after a finite time.

Theorem 3.3 (*No livelock and Completion*) *For any arbitrary time t , there exists a delay Δ_t such that, if no topological changes occur in the interval $(t, t + \Delta_t)$, then the algorithm completes at time $t + \Delta_t$.*

Proof. By applying the same argument as the one in Lemma 3.3, we will firstly assume that topological changes cease forever and will show that, in this case, the theorem holds. Then, we will prove that it is sufficient that topological changes cease for just a finite time; then, the theorem holds also for the case in which the sequence of topological changes is infinitely discrete.

Assume that topological changes cease forever. By Theorem 3.1, if topological changes cease at time t , within a finite time Δ_{tc} , all nodes will have correct topological information on the entire network. Thus, to prove the theorem, it is sufficient to show that, within a finite time, there will no messages in transit.

At time $t + \Delta_{tc}$, for any received messages at any operational node, the following conditions hold, that is

$$((W_i \geq W) \wedge (\mathcal{K}_i \supseteq \mathcal{K})) \vee ((W_i > W) \wedge ((\mathcal{K}_i \supseteq \mathcal{K}) \vee (LI_i(\mathcal{K}) \vee RI_i(\mathcal{K})))),$$

and $(W_i < W) \wedge (\mathcal{K} = \mathcal{K}_i)$. Conditions 3.1.1 and 3.1.2 do not hold.

Let W_{max} be the highest wave number at time $t + \Delta_{tc}$, and let v be a processor with a wave number $W_v < W_{max}$. Since there are no topological changes, at time $t + \Delta_{tc}$, and every node has correct information on the entire network, upon receipt of a message with wave number W_{max} , the condition $(W_v < W_{max}) \wedge (\mathcal{K} = \mathcal{K}_i)$ holds at v , node v updates its wave number to be W_{max} and propagates the received information, by Rule 3.1.3.

Further, if topological changes cease, at time $t + \Delta_{tc}$, all nodes will have correct information on the entire network, and Conditions 3.1.1 and 3.1.2 do not hold. Thus, no messages will be sent with wave numbers which are greater than W_{max} .

At time $t + \Delta_{tc}$, let Δ_{tm} be the longest communication delay experienced by the messages on all the operational links in the network, by applying the same argument as the one in Lemma 3.3 (see Figure 3.2), then, at time $t + \Delta_t$, all the operational nodes will have the same wave number W_{max} , where $\Delta_t = \Delta_{tc} + \Delta_{tm}$, Δ_{tc} is from Equation 3.5.2,

$$\Delta_{tm} = \Delta_{m1} + \Delta_{m2} + \Delta_{m3} + \dots + \Delta_{md} = \sum_{i=1}^d \Delta_{mi}, \quad (3.5.3)$$

d is the current diameter of the network, and Δ_{mi} is the longest communication delay experienced by the messages in transit between the nodes at distance $i - 1$ and the nodes at distance i from the node with the highest wave number W_{max} . Clearly, by Rule 3.1.4, no messages will still be in transit in the system after time $t + \Delta_t$.

Since $\Delta_{tc} + \Delta_{tm}$ is finite for any t , it is sufficient that topological changes cease for a finite amount of time Δ_t , provided $\Delta_t \geq \Delta_{tc} + \Delta_{tm}$. Thus, the theorem holds also for infinitely discrete case. $\square$

Theorems 3.1, 3.2, and 3.3 prove the correctness of the algorithm.

Notice that the correctness of our algorithm is insensitive to the arbitrary initial state of the system; that is the system will regain its consistency within a finite time, regardless of its initial state of the system.

3.6 Time and Message Complexities

Time Complexity

Based on Theorem 3.3, we know that the stabilizing time depends on the actual diameter of the network and the communication delay Δ , where $\Delta \geq \Delta_{tc} + \Delta_{tm}$. (Equations 3.5.2 and 3.5.3).

Since our stabilizing topology wave algorithm is aimed for an asynchronous dynamic network, where communication is via message passing, it is more general than most of the existing topology broadcast algorithms that employ the underlying pulse mechanism. The algorithm is correct, in the sense that it guarantees that every operational node in network will have the correct topological information on the entire network, provided the topological changes cease forever, or the interval between one burst of topological changes and the next

one is long enough. Our proposed protocol does not rely on *a priori* knowledge of the size of the network, whose size expands with time. Our algorithm differs from the techniques which use a “pulse mechanism” (eg., [SH86] and [HMR88]), in fact, it is *event driven*, i.e., every operational node in the network takes some immediated actions in response to *events* such as detection of topological changes or reception of a message.

The complexity results for our protocol are shown in Table 3.1 and 3.2, where N is the *a priori* bound on the maximum number of nodes in the network, D is the bound of the diameter of the network, r is the maximum degree of a node in G , $E(G)$ is the *a priori* bound on the maximum number of edges of G , and k is the number of topological changes. The vaules such as N , D , and $E(G)$ are used only for comparison purpose since the other algorithms assume their existence; as mentioned above, we do not make any such assumption. Therefore, to compare with the related work, we measure the time complexity with respect to a synchronous model.

Here is the comparison with other topology broadcast algorithms:

Reference	Time Complexity	Using Counters	Model and Comments
[Fin79]	$O(D)$	Yes	$\Phi_{\sigma}^N, \Phi_{N,D}^{N,D}$ Wave Algorithm.
[SG89]	$O(D)$	No	$\Phi_{\sigma}^N, \Phi_{N,D}^{N,D}$ Using the shortest path
this thesis	$O(D)$	Yes	$\Phi_{\sigma}^{\infty}, \Phi_{E(G)}^{\infty}, \Phi_{\sigma}^N$ Wave algorithm

Table 3.1 The Time Complexities of Topology Broadcast Algorithms

Message Complexity

As stated in the proof of the correctness and the time complexity measures, it is clear that the message complexity for the stabilizing wave algorithm is $O(E(G))$ per topological change. The message complexity for the mechanisms which use the underlying pulse for topology broadcast is $O(D \cdot E(G))$ per topological change, as in [Gaf87], [AV91], and [AEH92].

Reference	Message Complexity (per change)	Comments
[Fin79]	$O(E(G))$	Using sequence number for connectivity
[SG89]	$O(2^k E(G))$	Topology Broadcast with multiple copies
[Gaf87]	$O(D \cdot E(G))$	Using <i>Pulse</i> mechanism to reduce message complexity
this thesis	$O(E(G))$	Stabilizing Wave Algorithms

Table 3.2 The Message Complexities of Topology Broadcast Algorithms

From the tables, it follows that our solution is efficient also for non-expanding networks.

3.7 On Reset and Bounded Counters

As stated in Chapter 2, since the size of the network is changing with time, if there is no *a priori* bound on the maximum number of nodes in the graph, it is impossible for any topology algorithm to keep bounded its message sequence numbers (i.e., counters) and the amount of memory required per processor in the model Φ_o^∞ .

For the model Φ_o^N , the number of nodes in the network is unknown but bounded, unlike [AV91] and [AEH92] which assume the bound of the diameter of the network is

known (i.e., $\Phi_{N,D}^{N,D}$). The counters can be bounded in this model. The approach that we propose for reset is using the forest formed during the computation to keep the counters bounded.

3.7.1 Description of Reset

Recall that in Section 3.2, we design feedback messages and the variables which are aimed for capturing the information on topology broadcast. For the convenience of the readers, we recap the definitions of the feedback message and the variables:

Feedback message is a couple $P = \langle W, \mathcal{A} \rangle$, where

- (a) W is an integer, which denotes the wave number of the received message that the processor is acknowledging;
- (b) $\mathcal{A}$ is a boolean variable.

Upon receipt of one information message $\langle W, \mathcal{K} \rangle$, a processor i will reply with a feedback message $\langle W, \mathcal{A} \rangle$, where W is the same as the wave number in the received information message, and $\mathcal{A}$ is as follows:

$$\mathcal{A} = \begin{cases} \text{true} & \begin{array}{l} \text{the information } \mathcal{K} \text{ brought in by the received message is new to} \\ \text{the node } i, \text{ and the local knowledge } \mathcal{K}_i \text{ is not new to the sender,} \\ \text{i.e., } W_i \leq W \text{ and } \mathcal{K} \supset \mathcal{K}_i; \end{array} \\ \text{false} & \begin{array}{l} \text{the information } \mathcal{K} \text{ brought in by the received message is exactly} \\ \text{the same as the local knowledge } \mathcal{K}_i, \\ \text{i.e., } W_i = W \text{ and } \mathcal{K} = \mathcal{K}_i. \end{array} \end{cases}$$

The purpose of feedback message is only to construct a forest for reset. Every operational processor selects a subset of the operational links incident to it to form a forest.

Given a processor i , these information is captured by the following variables:

$pred(i)$: the identity of the node from which node i has received a message and obtained the current topological information. This node is called as a *predecessor*.

$succ(i)$: a sub-set of neighbors of node i to which it sends information messages, and $Adjacent(i) \supseteq succ(i)$. The nodes in this set are called as *successors*.

$resp(i)$: a subset of $succ(i)$ from which node i has received a feedback message with the correct wave number, and $\mathcal{A} = \text{true}$.

3.7.2 Explanation of Reset

Notice that, after the topological changes cease forever or during the intermediated period between two burst of the topological changes, the “successful” wave in which all the operational nodes in the network receive the correct topological information on the entire network may form a forest,

The outline of the process of the forest construction associated with each condition in the algorithm is as follows:

Topological Change

(Condition 3.1.1) /* Event : A topological change occurs */
 $\Rightarrow succ(i) := Adjacent(i) ; pred(i) := \emptyset ; resp(i) := \emptyset ;$

Upon receipt of $\langle W, \mathcal{K} \rangle$ from v

[| (Condition 3.1.2) /* Upon receipt of $\langle W, \mathcal{K} \rangle$ from v */
 $\Rightarrow succ(i) := Adjacent(i) ; pred(i) := \emptyset ; resp(i) := \emptyset ;$

[| (Condition 3.1.3) /* Upon receipt of $\langle W, \mathcal{K} \rangle$ from v */

```

⇒      succ(i) := Adjacent(i); pred(i) := ∅; resp(i) := ∅;
      if (Wi ≤ W) ∧ (K ⊃ Ki)
        ⇒      Ki := K; succ(i) := succ(i) - {v}; pred(i) := v;
      [] (Wi = W) ∨ (LCi(K) ∧ RIi(K))
        ⇒      .....
      fi;

```

```

[] (Condition 3.1.4) /* Upon receipt of <W, K> from v */
⇒      if (Wi = W) ∧ (K = Ki)
        ⇒      send <W, false> to v;
        if v ∈ succ(i) ⇒ succ(i) := succ(i) - {v} fi;
      fi;

```

Upon receipt of <W, $\mathcal{A}$ > from v

```

if ( Wi = W )
⇒      if (  $\mathcal{A}$  = true ) ⇒ resp(i) := resp(i) ∪ {v} ;
        [] (  $\mathcal{A}$  = false ) ⇒ succ(i) := succ(i) - {v} ;
      fi;
      if ( resp(i) = succ(i) ) ∧ ( pred(i) ≠ ∅ )
        ⇒      send <W, true> to pred(i); /* the node becomes a root of a sub tree */
        [] ( resp(i) = succ(i) ) ∧ ( pred(i) = ∅ )
          ⇒      reset-wave(i); /* For model  $\Phi_{\emptyset}^m$ , reset-wave procedure is invoked */
      [] else ⇒ skip;
    fi;
[] else ⇒ skip; /* Wi ≠ W */
fi

```

The Process of a Forest Construction in Stabilizing Topology Wave Algorithm

2 OF 2

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

When a node decides that the computation completes, it initiates a reset wave by sending reset messages along the tree.

When a nodes along the tree receive the reset message it resets its local counter and propagates the reset message along the tree. Eventually, this reset wave forms a diffusion computation which keeps all the counters bounded.

procedure reset-wave(i)

{ processor i sends a reset message to keep the counters bounded within its sub-tree }

begin

for all $u \in succ(i)$ /* $\mathcal{K}_i$ is replaced by a special symbol for any reset messages. */

do send $\langle W_i, \mathcal{K}_{reset} \rangle$ to u **od**;

$W_i := 0$;

end

The Procedure of Reset-wave

Upon receipt of $\langle W, \mathcal{K}_{reset} \rangle$ from node v

if ($W_i = W$) $\wedge$ ($resp(i) = succ(i)$) $\wedge$ ($pred(i) = v$)

$\Rightarrow$ reset-wave(i); /* For model Φ_o^N , reset-wave procedure is invoked. */

[] else

$\Rightarrow$ skip;

fi

Upon receipt of a reset message

3.7.3 The Correctness and Complexity of Reset

Lemma 3.5 *Given an arbitrary time t , then there exists a finite delay Δ_{tr} such that, if no topological changes occur in the interval $(t, t + \Delta_{tr})$, then the algorithm can reset all the counters at time $t + \Delta_{tr}$.*

Proof. From Theorem 3.3, if no topological changes occur, at time $t + \Delta_{tc}$, any every operational node v will have the correct topological information on the entire network. By applying the same argument as the one of Theorem 3.3, clearly, if no topological changes occur in the interval $(t + \Delta_{tc}, t + \Delta_{tr})$, the wave numbers of all the operational nodes will have been reset at time $t + \Delta_{tr}$, where

$$\Delta_{tr} = \Delta_{tc} + \sum_{i=1}^d \Delta_{ri}$$

d is the current diameter of the network, and Δ_{ri} is the longest communication delay experienced by the reset messages in transit between the nodes at distance $i - 1$ and the nodes at distance i from the set of initiators of the reset wave. $\square$

Lemma 3.6 *The above reset procedure is used to keep the counters bounded. The total number of the messages for each reset is at most $N - 1$, where N is the upper bound of the number of nodes in the model Φ_o^N .*

Proof. The relation of $\text{succ}(i)$ and $\text{pred}(i)$ form a set of trees (a forest) and reset messages are only sent on such order. Therefore, the number of the messages for a reset wave is at most $N - 1$. $\square$

3.7.4 Comparison with related work

For the convenience of reader, we recap the models with respect to the different assumptions, which are described in Chapter 2. The subscript of Φ is the knowledge known to every node, and the superscript is the indicator of the existence of the bound of the maximum number of the nodes in the network. The following figure is the hierarchy of the models of dynamic systems:

- $\Phi_{\emptyset}^{\infty}$: The model in which every node has no knowledge, and there is no *a priori* bound on the maximum number of the nodes in the network ;
- $\Phi_{\epsilon(G)}^{\infty}$: The model in which every node has an estimate on the ratio $\epsilon(G)$ of the network expansion, but there is no *a priori* bound on the maximum number of the nodes in the network;
- $\Phi_{\emptyset}^N$: The model in which every node has no knowledge, but there exists *a priori* bound N on the maximum number of the nodes in the network;
- $\Phi_{N,D}^{N,D}$: The model in which every node has the initial knowledge on the *a priori* bound N on the maximum number of nodes and the bound D of the diameter of the network.

The reset issue has been addressed by many papers. However, most of them can only cope with the model with very restrictive assumptions such as $\Phi_{N,D}^{N,D}$ and Φ_{σ}^N . As shown in previous sections, our approaches can be applied to Φ_{σ}^N . In Chapter 6, we will discuss how to reset with respect to the model $\Phi_{\epsilon(G)}^{\infty}$, which indicates that our proposed method is more general than others.

Here is a comparison among the models to which the reset methods can be applied:

Reference	The Models and Assumptions for Reset
[Awe91]	$\Phi_{N,D}^{N,D}$
[AEH92]	Φ_{σ}^N and $\Phi_{N,D}^{N,D}$
this thesis	Φ_{σ}^N and $\Phi_{\epsilon(G)}^{\infty}$

Table 3.3 The Comparison of the Models of Reset

Space Complexities

Consider the space required for each node during the execution of this stabilizing wave algorithm, since each node has to store the entire topological knowledge at its site. Further, the model we consider for space complexity measures is Φ_{σ}^N .

Reference	Space complexity memory required per node after stabilization	Comments (with model Φ_{σ}^N)	Using Counters
[Fin79]	$O(N^2 \log N)$	The messages wave algorithm	Yes
[SG89, AEH92]	$O(r N^2 \log N)$	Using the shortest path & multiple copies	No
this thesis	$O(N^2 \log N)$	The stabilizing wave algorithm	Yes

Table 3.4 The Space Complexities of Topology Broadcast Algorithms

Since the topological information carried by an information message is the entire knowledge known to the sender, the amount of space required per information message is the same as the one for each node.

3.8 Chapter Summary

In this chapter, we introduce a self stabilizing topology wave algorithm, in which the information messages contain the entire topological knowledge known to the sender. Although the approach is not optimized with respect to the information, the algorithm is very practical and efficient. The stabilizing wave algorithm can guarantee that every operational processor obtains the correct topological information on the entire network within a finite time, provided the burst of topological changes in a dynamic network is finite or infinitely discrete.

Since the stabilizing wave algorithm employs wave numbers as sequence numbers to distinguish old and new topological information, the version of bounded counters is also presented.

Based on this algorithm, in Chapter 4, an optimized stabilizing topology broadcast algorithm is introduced to reduce the space requirement per message. The information message contains the topological changes rather than the entire topological information of the sender.

Chapter 4

The Optimized Stabilizing Wave Algorithm

4.1 Introduction

In the stabilizing wave algorithm presented in Chapter 3, whenever a node detects that a topological change occurs, it broadcasts the messages containing the entire accumulated local knowledge. Therefore, the space requirements of information messages are very high. For the model Φ_{σ}^N in which there exist *a priori* bound N on the maximum number of nodes in the network, the number of bits to represent this knowledge in each message could be $O(N^2 \log N)$.

Under the same model and assumptions that are described in Chapter 2, we present an optimized stabilizing topology wave algorithm for a dynamic network. The characteristics of this algorithm is that, in each wave, the topological information carried by an information message is limited to the changes of the local topological information of the initiator. To correctly accommodate this modification, it is crucial for every link in the network to have a wave number so that every node can identify the new and old information with respect to any link in the network (i.e., the elements of the entire topological knowledge).

The rest of this chapter is organized as follows. In Section 4.2, we describe the representation of the topological knowledge and the messages. In Sections 4.3 and 4.4, we show how the wave rules accommodate for the optimization, and formally present

the optimized stabilizing wave algorithm. In Section 4.5, we prove its correctness of the optimized algorithm. Finally, in Section 4.6, the chapter summary is given.

4.2 Representation of Knowledge and Messages

In this optimized wave algorithm, the representation of topological knowledge is similar to the one described in Section 3.2. Each element of the adjacency matrix is a couple. It can be described as follows:

Given a processor i , the knowledge set of i is represented by a matrix $\mathcal{K}_i$. Each element of the matrix $\mathcal{K}_i$ is a couple $\mathcal{KW}_i[v,u] = \langle \mathcal{K}_i[v,u], W_i[v,u] \rangle$, where $\mathcal{K}_i[v,u]$ is the status of a bidirectional link between node v and node u , and $W_i[v,u]$ is a wave number of the link (v,u) . In practice, it can be implemented as three-dimensional array .

The value of the link status $\mathcal{K}_i[v,u]$ is defined the same as in Chapter 3, that is:

$$\mathcal{K}_i[v,u] = \begin{cases} 0 & \text{the link between } v \text{ and } u \text{ is failed (down) ;} \\ 1 & \text{the link between } v \text{ and } u \text{ is operational (up) ;} \\ 2 & \text{the status of the link } (v,u) \text{ is ambiguous, } v, u \neq i ; \\ \text{Undefined} & \text{the existence of the link between } v \text{ and } u \text{ is unknown;} \end{cases}$$

where $v, u \in \mathcal{K}_i$.

By definition, the status of any links incident to node i must not be ambiguous, since a node has the initial knowledge about the links incident to it.

For a node v in the network, it is represented by $\mathcal{K}_i[v,v]$. The node status is defined below:

$$\mathcal{K}_i[v, v] = \begin{cases} 0 & \text{processor } v \text{ is failed (down);} \\ 1 & \text{processor } v \text{ is operational (up);} \\ 2 & \text{the status of processor } v \text{ is ambiguous, } v \neq i; \end{cases}$$

where $v \in \mathcal{K}_i$.

As stated in Chapter 3, given a processor i , $Neighbor_{\mathcal{K}_i}(v)$ denotes a set of nodes incident to v , where $v \in \mathcal{K}_i$, that is

$$Neighbor_{\mathcal{K}_i}(v) \stackrel{\text{def}}{=} \bigcup_{\mathcal{K}_i[v, u] \text{ is defined}} \{u\}.$$

Clearly, $Neighbor_{\mathcal{K}_i}(v)$ is a subset of $\mathcal{K}_i$, that is $\mathcal{K}_i \supseteq Neighbor_{\mathcal{K}_i}(v)$.

Notice that the status of a node is ambiguous (i.e., $\mathcal{K}_i[v, v] = 2$), if and only if

$$\left(\bigwedge_{u \in Neighbor_{\mathcal{K}_i}(v)} \mathcal{K}_i[v, u] \neq 1 \right) \wedge \left(\bigvee_{u \in Neighbor_{\mathcal{K}_i}(v)} \mathcal{K}_i[v, u] = 2 \right),$$

where $v \notin Adjacent(i)$.

Another element of $\mathcal{KW}[v, u]$ is the wave number $W_i[v, u]$, which is used as a sequence number to identify the new and information on (v, u) .

Given a link (v, u) , $W_i[v, u]$ is defined as follows:

$$W_i[v, u] = \begin{cases} \text{an integer} & \text{which denotes the wave number of link } (v, u), u \in Neighbor_{\mathcal{K}_i}(v), \\ \text{Undefined} & v, u \in \mathcal{K}_i \text{ and } u \notin Neighbor_{\mathcal{K}_i}(v). \end{cases}$$

For a node (v, v) , its wave number should be the highest one among all its local links, i.e., $W_i[v, v] = \max(W_i[v, u] \mid u \in Neighbor_{\mathcal{K}_i}(v))$.

When processor i becomes operational, it resets the status of all the local links to be 0 (down) in its local information $\mathcal{K}_i$ immediately, before preceding any other actions; and the following condition holds:

$$\bigwedge_{j \in \text{Neighbor}_{\mathcal{K}_i}(i)} \mathcal{K}_i[i, j] = 0.$$

Then a node detects which links are operational by checking the set of the currently operational nodes. Therefore, the changes of link status are used to represent the changes of node status.

Information message $\mathcal{K}_\Delta$ is a set of couples, which represents the topological changes; that is $\mathcal{K}_\Delta = \{\langle \mathcal{KW}[v, u] \rangle\}$, where $\mathcal{KW}[v, u] = \langle \mathcal{K}[v, u], W[v, u] \rangle$, $\mathcal{K}[v, u]$ is the status of the link between v and u , $W[v, u]$ is the wave number of link $[v, u]$.

It is important to point out that information messages are designed for two purposes:

- (a) Transfer the topological information, where the status of a link (v, u) is either *up* or *down*;
- (b) Clarify some ambiguous topological information, where the status of a link (v, u) is *ambiguous*.

Given a processor i , let $\mathcal{K}_i$ be the local knowledge at node i ,

$$\begin{aligned} &\langle \mathcal{K}_i[v, u], W_i[v, u] \rangle \text{ and } \langle \mathcal{K}_i[i, j], W_i[i, j] \rangle \text{ are elements in } \mathcal{K}_i, \text{ i.e.,} \\ &\langle \mathcal{K}_i[v, u], W_i[v, u] \rangle, \langle \mathcal{K}_i[i, j], W_i[i, j] \rangle \in \mathcal{K}_i; \end{aligned}$$

and let $\langle \mathcal{K}_\Delta \rangle$ be the received message,

$$\begin{aligned} &\langle \mathcal{K}[v, u], W[v, u] \rangle \text{ and } \langle \mathcal{K}[i, j], W[i, j] \rangle \text{ are elements in } \mathcal{K}_\Delta, \text{ i.e.,} \\ &\langle \mathcal{K}[v, u], W[v, u] \rangle, \langle \mathcal{K}[i, j], W[i, j] \rangle \in \mathcal{K}_\Delta. \end{aligned}$$

As defined in Chapter 3, the notations on the consistency of knowledge such as $LC_i(\mathcal{K})$, $GC_i(\mathcal{K})$, $LI_i(\mathcal{K})$, and $RI_i(\mathcal{K})$ are defined as follows:

(i) *Local Consistency* ($LC_i(\mathcal{K})$):

$$LC_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigwedge \mathcal{K}[i, j] = \mathcal{K}_d[i, j]$$

(ii) *Global Consistency* ($GC_i(\mathcal{K})$):

$$GC_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigwedge \mathcal{K}[v, u] = \mathcal{K}_d[v, u]$$

(iii) *Local Inconsistency* ($LI_i(\mathcal{K})$):

$$LI_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigvee \mathcal{K}[i, j] \neq \mathcal{K}_d[i, j]$$

(iv) *Remote Inconsistency* ($RI_i(\mathcal{K})$):

$$RI_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigvee_{v, u \neq i} \mathcal{K}[v, u] \neq \mathcal{K}_d[v, u]$$

where $LC_i(\mathcal{K}) = \neg LI_i(\mathcal{K})$, and $GC_i(\mathcal{K}) = \neg (LI_i(\mathcal{K}) \vee RI_i(\mathcal{K}))$.

4.3 The Wave Rules

Similarly, the wave rules, which are presented in Section 3.3, have been modified to facilitate the optimized stabilizing topology wave algorithm. Since there is a wave number attaching to each link in the network, which uniquely identifies the old and new information on a link, it simplifies the process of the received topological information for an operational

processor. As described in the model of computation (Section 3.2), the failure of a node is represented by the failures of the adjacent links of the node. Therefore, all the topological changes can be represented by the changes of the status of the links.

The wave rules are classified as *making waves*, *propagating wave*, and *eliminating wave*, and are aimed for achieving the same goal as described in Section 3.3; that is to govern how the topological information should be updated and propagated at each node

For the sake of simplicity, we reveal the rules and the behavior of a processor, say i , in different scenarios with respect to these rules:

Making Waves

Rule 4.1.1 When node i detects that one of its adjacent links (i,j) goes down, it initiates a new wave $(W_i[i,j] + 1)$ to broadcast this new information (two initiators) .

Condition 4.1.1

$$(\exists j \mid \mathcal{K}_i[i,j] = 1 \wedge j \notin \text{Adjacent}(i)).$$

As described in Section 3.3, when link (i,j) fails, j will not be in $\text{Adjacent}(i)$; but it was in $\text{Adjacent}(i)$ before the failure, and therefore $\mathcal{K}_i[i,j] = 1$ (up). This rule is applied to these two situations:

- (i) a link fails;
- (ii) a node fails which results in the failures of all the adjacent links of the node.

Rule 4.1.2 When node i detects that one of the incident links (i,j) comes up, it initiates a new wave $(W_i[i,j] + 1)$ to broadcast this new topological information (two initiators) .

Condition 4.1.2

$$(\exists j \mid \mathcal{K}_i[i,j] = 0 \wedge j \in \text{Adjacent}(i)).$$

As described in Section 3.3, when link (i,j) becomes operational, j will be in $\text{Adjacent}(i)$; but it was not in $\text{Adjacent}(i)$ before the recovery, and therefore $\mathcal{K}_i[i,j] = 0$ (down).

It is important to point out that in this optimized algorithm, when a link becomes operational, the two nodes at both side of the up link exchange their entire topological knowledge to ensure the correctness and completeness of the topological knowledge at each node.

This rule is applied to the situations listed below:

- (i) a link becomes operational, that is a crashed link has been repaired, it was down before;
- (ii) a node becomes operational, that is a crashed processor has been repaired, and at least one of the links incident to the up node must become operational;
- (iii) the system starts;

For situations (ii) and (iii), the up nodes reset the status of the links incident to it in its local topological information immediately, as described in Chapter 2. Then the operational nodes initiate a new wave to broadcast the topological information by applying this rule.

Rule 4.1.3 When node i receives an information message, if one of these two conditions holds, a new wave is created by i to broadcast the topological information.

- (i) $(W_i[i,j] < W[i,j]) \wedge LI_i(\mathcal{K})$;
- (ii) $(W_i[i,j] = W[i,j]) \wedge LI_i(\mathcal{K})$.

Condition 4.1.3

$$(W_i[i,j] \leq W[i,j]) \wedge LI_i(\mathcal{K})$$

When node i receives some inconsistent information on a local link (i,j) with a wave number, which is not lower than the wave number at node i ($W_i[i,j] \leq W[i,j]$), then i creates a new wave to broadcast the topological information of the current status of the local link and to clarify the ambiguity with a higher wave number ($W[i,j] + 1$).

Propagating Waves

Rule 4.1.4 Node i uses the same wave number brought in the received messages to propagate the topological information, if it is in one of these three scenarios:

- (i) the information in the received message with a higher wave,
i.e., $(W_i[v,u] < W[v,u])$, and $RI_i(\mathcal{K}) \vee GC_i(\mathcal{K})$;
- (ii) the received information is remote inconsistent, and the wave number of the received message has the same wave number as the local wave number,
i.e., $(W_i[v,u] = W[v,u])$, and $RI_i(\mathcal{K})$;
- (iii) the information from the messages is defined and it is new to node i , that is $\mathcal{K}_i[v,u]$ is undefined,
i.e., $(\mathcal{K}[v,u] \text{ is defined}) \wedge (u \notin \text{Neighbor}_{\mathcal{K}_i}(v))$.

Condition 4.1.4

$$\begin{aligned}
& ((W_i[v,u] < W[v,u]) \wedge (RI_i(\mathcal{K}) \vee GC_i(\mathcal{K}))) \vee \\
& ((W_i[v,u] = W[v,u]) \wedge RI_i(\mathcal{K})) \vee \\
& ((\mathcal{K}[v,u] \text{ is defined}) \wedge (u \notin Neighbor_{\mathcal{K}_i}(v)))
\end{aligned}$$

Eliminating Waves

Rule 4.1.5 When node i receives a message, it will discard the information in the received message. If it is in any of the these two scenarios:

- (i) $(W_i[v,u] = W[v,u]) \wedge GC_i(\mathcal{K});$
- (ii) $(W_i[v,u] > W[v,u]);$

Condition 4.1.5

$$((W_i[v,u] = W[v,u]) \wedge GC_i(\mathcal{K})) \vee (W_i[v,u] > W[v,u])$$

With respect to all the above rules, let us examine the predicates of all the conditions:

- (i) $(W_i[v,u] < W[v,u])$

$$LI_i(\mathcal{K})$$

< Condition 4.1.3 >

$$RI_i(\mathcal{K}) \vee GC_i(\mathcal{K})$$

< Condition 4.1.4 >

- (ii) $(W_i[v,u] = W[v,u])$

$$LI_i(\mathcal{K})$$

< Condition 4.1.3 >

$$RI_i(\mathcal{K})$$

< Condition 4.1.4 >

$$GC_i(\mathcal{K})$$

< Condition 4.1.5 >

(iii) $(W_i[v, u] > W[v, u])$ < Condition 4.1.5 >

(iv) $(\mathcal{K}[v, u] \text{ is defined}) \wedge (u \notin \text{Neighbor}_{\mathcal{K}_i}(v))$ < Condition 4.1.4 >

It is not difficult to verify that these predicates cover all the possible situations.

4.4 The Optimized Wave Algorithm

Before formally describing the optimized wave algorithm, we reveal the behavior of a processor with respect to the two types of events, i.e., *detection of topological changes* and *reception of information messages*.

Detection of topological changes

By definition, when a node becomes operational, at least one of the incident links must become operational. Observe that a new node can not obtain the entire topological information by receiving only the topological changes brought in by an incoming message. To overcome this problem, when a node detects that one of local links comes up, it has to send a message containing the entire topological knowledge known to the node.

It implies that, when a link becomes operational, the two incident nodes take the following actions:

- (i) the two nodes exchange their entire local topological knowledge;

- (ii) they continue to broadcast only the difference among their topological knowledge to their respective neighbors;
- (iii) if there exists any inconsistent topological information within the same wave, a request for clarifying the ambiguity is sent with the same wave number.

When the system starts, there are a number of processors which become operational, and these operational nodes form a connected component. It is in this component that all the up processors take the above actions to broadcast their local topological knowledge. After the system starts, when a node comes up, the scenario is the same. That is the new node exchanges its entire local topological knowledge with its active (operational) adjacent nodes in the connected component.

Reception of an information message

When a node i receives a message regarding (v, u) , $(\langle \mathcal{K}[v, u], W[v, u] \rangle \in \mathcal{K}_\Delta)$, there are the following scenarios.

- (i) If $\langle \mathcal{K}[v, u], W[v, u] \rangle$ is defined, and $\langle \mathcal{K}_i[v, u], W_i[v, u] \rangle$ is undefined (i.e., $u \notin \text{Neighbor}_{\mathcal{K}_i}(v)$), it implies that link (v, u) must not be local to i , then the node updates the local topological knowledge $\mathcal{K}_i$

$$\mathcal{K}_i[v, u] := \mathcal{K}[v, u], W_i[v, u] := W[v, u].$$

- (ii) If both $\langle \mathcal{K}[v, u], W[v, u] \rangle$ and $\langle \mathcal{K}_i[v, u], W_i[v, u] \rangle$ are defined (i.e., $u \in \text{Neighbor}_{\mathcal{K}_i}(v)$), there are the following three cases:

- (1) If $W_i[v, u] < W[v, u]$, then there are three sub-cases:

-
- (a) If there exists an inconsistent information that is *remote* to node i (i.e., $RI_i(\mathcal{K})$), then i updates the local knowledge ($\mathcal{K}_i[v,u] := \mathcal{K}[v,u]$, $W_i[v,u] := W[v,u]$), and then propagates the information;
 - (b) If there exists an inconsistent information that is *local* topological information (i.e., $LI_i(\mathcal{K})$), (for the local information, it should not be ambiguous, that is $\mathcal{K}_i[i,j] \neq 2$), then node i initiates a new wave ($\mathcal{K}[i,j] := \mathcal{K}_i[i,j]$, $W_i[i,j] := W[i,j] := W[i,j] + 1$) to clarify the local inconsistent information $LI_i(\mathcal{K})$;
 - (c) If the received information is consistent with the local information, (i.e., $GC_i(\mathcal{K})$), then the node upgrades its wave number ($W_i[v,u] := W[v,u]$) and propagates the information with the higher wave number.
- (2) If ($W_i[v,u] = W[v,u]$), then there are three sub-cases:
- (a) If there exists an inconsistent information that is *remote* to node i , (i.e., $RI_i(\mathcal{K})$), then i updates $\mathcal{K}_i[v,u]$ (i.e., $\mathcal{K}_i[v,u] := \mathcal{K}[v,u] := 2$), and uses the same wave number to request for the clarification of the non-local ambiguity;
 - (b) If there exists an inconsistent information that is *local* to node i , (i.e., $LI_i(\mathcal{K})$), then i initiates a new wave to clarify the ambiguity on the local information (i.e., $\mathcal{K}[i,j] := \mathcal{K}_i[i,j]$, $W_i[i,j] := W[i,j] := W_i[i,j] + 1$). This case is the same as the one described in case (1.b).

- (c) If the received information is consistent with the information known to i (i.e., $GC_i(\mathcal{K})$), then $\mathcal{KW}[v,u]$ (i.e., $\langle \mathcal{K}[v,u], W[v,u] \rangle$) is discarded and removed from the information;
- (3) If $(W_i[v,u] > W[v,u])$, then the information in the received message $\mathcal{KW}[v,u]$ (i.e., $\langle \mathcal{K}[v,u], W[v,u] \rangle$) is discarded.

The above are the salient features of the optimized wave algorithm. The wave rules described in previous section such as *making waves*, *propagating waves*, and *eliminating waves* govern how an operational processor must handle the received information messages.

Since the sequence of topological changes is finite or infinitely discrete, every operational nodes will eventually have the correct topological information on the entire network, as will be shown later.

We now present the optimized stabilizing algorithm. Once the system starts, processors execute this scheme as an underlying computation for topology broadcast and self stabilization.

In general, when an operational processor detects some topological changes, it broadcasts the topological changes by applying Rule 4.1.1 or 4.1.2. Upon receipt of an information message, a processor handles the received information according to Rules 4.1.3, 4.1.4, and 4.1.5.

The formal presentation of the optimized wave algorithm is as follows:

Topological Change

(Condition 4.1.1)

/ (i,j) is down, i.e., $(\exists j \mid \mathcal{K}_i[i,j] = 1 \wedge j \in \text{Adjacent}(i))$: Making Waves */*

$\Rightarrow \mathcal{KW}_i[i,j] := \langle 0, W_i[i,j] + 1 \rangle;$
for all $q \in \text{Adjacent}(i)$ **do** send $\langle \mathcal{KW}_i[i,j] \rangle$ to q **od**

[] (Condition 4.1.2)

/ (i,j) is up, i.e., $(\exists j \mid \mathcal{K}_i[i,j] = 0 \wedge j \in \text{Adjacent}(i))$: Making Waves */*

$\Rightarrow \mathcal{KW}_i[i,j] := \langle 1, W_i[i,j] + 1 \rangle$
 send a message $\langle \mathcal{K}_i \rangle$ to j ; */* the entire local information. */*
for all $q \in \text{Adjacent}(i) - \{j\}$ **do** send $\langle \mathcal{KW}_i[i,j] \rangle$ to q **od**

Upon receipt of an info message $\langle \mathcal{K}_\Delta \rangle$

[] (Condition 4.1.3) */* Upon receipt of $\langle \mathcal{K}_\Delta \rangle$ from p */*

/ where $(W_i[i,j] \leq W[i,j]) \wedge LI_i(\mathcal{K})$: Making Waves */*

$\Rightarrow \mathcal{KW}_i[i,j] := \langle \mathcal{K}_i[i,j], W_i[i,j] + 1 \rangle$
for all $q \in \text{Adjacent}(i)$ **do** send $\langle \mathcal{KW}_i[i,j] \rangle$ to q **od**

[] (Condition 4.1.4) */* Upon receipt of $\langle \mathcal{K}_\Delta \rangle$ from p : Propagating Waves */*

$\Rightarrow$ **if** $((W_i[v,u] < W[v,u]) \wedge (RI_i(\mathcal{K}) \vee GC_i(\mathcal{K}))) \vee$
 $((\mathcal{K}[v,u] \text{ is defined}) \wedge (u \notin \text{Neighbor}_{\mathcal{K}_i}(v)))$
 $\Rightarrow \mathcal{KW}_i[v,u] := \langle \mathcal{K}[v,u], W[v,u] \rangle;$
for all $q \in \text{Adjacent}(i) - \{p\}$ **do** send $\langle \mathcal{KW}_i[v,u] \rangle$ to q **od**

[] $(W_i[v,u] = W[v,u]) \wedge RI_i(\mathcal{K})$

$\Rightarrow \mathcal{KW}_i[v,u] := \langle 2, W_i[v,u] \rangle;$ */* set up a request */*
for all $q \in \text{Adjacent}(i)$ **do** send $\langle \mathcal{KW}_i[v,u] \rangle$ to q **od**

fi;

[] (Condition 4.1.5) */* Upon receipt of $\langle \mathcal{K}_\Delta \rangle$ from p */*

/ where $((W_i[v,u] = W[v,u]) \wedge GC_i(\mathcal{K})) \vee (W_i[v,u] > W[v,u])$: Eliminating Waves */*

$\Rightarrow$ skip; */* the information in messages is eliminated */*

The Optimized Topology Wave Algorithm

4.5 Proof of Correctness

To prove the correctness of the algorithm presented in Section 4.4, we begin with some technical lemmas.

Lemma 4.1 *Given two operational nodes, i and j , connected by an operational link, if the wave numbers for the up link (i,j) are different, without loss of generality, let $W_i[i,j] \geq W_j[i,j]$. If $W_i[i,j] > W_j[i,j]$, then*

- (i) *there is a message in transit from i to j on the link (i,j) with wave number $W_i[i,j]$;*
- (ii) *if the link (i,j) does not fail before the message arrives, the wave number of both i and j upon processing of the message will be equal to $W_i[i,j]$.*

Proof. Part (i). We will consider two cases depending on whether or not at least one of the two node is (re)starting.

(1) By definition, link (i,j) is just becoming operational. This could be caused by the (re)starting of i or j , or both. Node i and j exchange their local information on link (i,j) (by Rule 4.1.2). Since $W_i[i,j] > W_j[i,j]$, the message must be in transit from i to j on the link (i,j) .

(2) Node i received a message from one of its neighbors which contains information on link (i,j) with wave number $W[i,j]$ (where $W[i,j] > W_j[i,j]$), with respect to (i,j) , either of these two conditions must hold:

- (a) the information is consistent with $\mathcal{K}_i[i,j]$ (i.e., $(W_i[i,j] \leq W[i,j]) \wedge GC_i(\mathcal{K})$);
- (b) the information is inconsistent with $\mathcal{K}_i[i,j]$ (i.e., $(W_i[i,j] < W[i,j]) \wedge LI_i(\mathcal{K})$).

A message sent from i to j in either of the above two situations, by Rule 4.1.3 or Rule 4.1.4. Since $W_i[i,j] > W_j[i,j]$, the message must be in transit from i to j on the link (i,j) .

Part (ii). If the link does not fail before the message from node i arrives at node j . If $W_i[i,j] > W_j[i,j]$ still holds, node j takes the actions according to Rule 4.1.4, and sets its wave number to a value $W_j[i,j] := W_i[i,j]$. $\square$

By definition, nodes i and j have the correct status of link (i,j) .

Lemma 4.2 *Given two operational nodes, i and j , connected by an operational link at time t , if there exists a link (j,w) such that $W_i[j,w] \neq W_j[j,w]$, then*

- (i) *there exists a message in transit on the link (i,j) with wave number $W_i[j,w]$;*
- (ii) *if the link (i,j) does not fail before the message of Condition (i) arrives, the wave numbers of both i and j upon processing of the message will be greater than or equal to $\max(W_i[j,w], W_j[j,w])$.*
- (ii) *there exists a finite delay Δ_{tL} such that, if no topological changes occur in the interval $(t, t + \Delta_{tL})$, both i and j will have correct information on link (j,w) at time $t + \Delta_{tL}$.*

Proof. Part (i). We will distinguish the case in which at least one of the two node is (re)starting from any other cases.

If one of the nodes is (re)starting then, by definition, then the link (i,j) must be now coming up, thus, both i and j will exchange their local information with each other.

If neither node is (re)starting; since $W_i[j,w] \neq W_j[j,w]$, there are two cases, which are $(W_i[j,w] > W_j[j,w])$ and $(W_i[j,w] < W_j[j,w])$ (see Figure 4.1):

Figure 4.1 Topological Changes in Two Adjacent Operational Nodes

- (1) Let $W_j[j, w] > W_i[j, w]$. Node j increases its wave number only in these two scenarios: either after detecting the change of status of link (j, w) or after receiving an information message. In the former scenario, it sent a message with the new wave number $W_j[j, w]$ to all its operational neighbors including i (Rule 4.1.1 or 4.1.2); because the wave number of node i is less than the wave number at node j ($W_j[j, w] > W_i[j, w]$), i has obviously not received such a message which is therefore in transit from j to i on link (i, j) . In the latter scenario, upon receipt of the message, node j sent the message with its wave number $W_j[j, w]$ to all its operational neighbors (Rule 4.1.3 or 4.1.4); the only exception would be that if the received message had the information $(W_j[j, w] < W[j, w]) \wedge GC_j(\mathcal{K})$, then node j propagated the information on (j, w) to all its operational neighbors except the sender of the message. Since $W_j[j, w] < W_i[j, w]$, clearly the sender is not node i .
- (2) Let $W_j[j, w] < W_i[j, w]$. Node i increases its wave number only if it received a message containing information on link (j, w) . According to Rule 4.1.3 or 4.1.4, node i sent the message with wave number $W_i[j, w]$ to all its operational neighbors; the only exception would be that if the received message had the information $(W_i[j, w] < W[j, w]) \wedge GC_i(\mathcal{K})$, then node i propagated the information on (j, w) to all its operational neighbors except the sender of the message. Since $W_j[j, w] < W_i[j, w]$, the sender clearly is not j . Since $W_j[j, w] < W_i[j, w]$, j has obviously not received such a message which is therefore in transit from i to j on link (i, j) .

Part (ii). Clearly, from part (i), if the link (i, j) does not fail before the message arrives from the node with higher wave number at the node with lower wave number, if $W_i[j, w] \neq W_j[j, w]$ still holds, the node with lower wave number takes action according to Rule 4.1.3 or 4.1.4, and updates its wave number to be greater than or equal to $\max(W_i[j, w], W_j[j, w])$.

Part (iii). From the model and the assumptions, the sequence of topological changes is finite or infinitely discrete. Assume firstly that topological changes cease forever; we will show that, in this case, this lemma holds. We will then show that it is sufficient that topological changes cease for just a finite time; then proving that the lemma holds also the case in which the sequence of topological changes is infinitely discrete.

If the topological changes cease forever, at time t , between node i and j , from proof (i) and (ii), there exists a message in transit. Let Δ_{L1} be the communication delay experienced by the messages from i to j , and let Δ_{L2} be the communication delay experienced by the messages from j to i .

For case (1) of part (i), at time $t + \Delta_{L2}$, upon receipt on $\langle \mathcal{K}_\Delta \rangle$ containing $\{\mathcal{K}\mathcal{W}_j[j, w]\}$ from j , node i will have the correct information on link (j, w) (Rules 4.1.3 and 4.1.4).

For case (2) of part (i), at time $t + \Delta_{L1}$, upon receipt on $\langle \mathcal{K}_\Delta \rangle$ containing $\{\mathcal{K}\mathcal{W}_i[j, w]\}$ from i , if the information is inconsistent with the local information $\mathcal{K}_j[j, w]$, (i.e., $LI_j(\mathcal{K})$ holds), node j updates the wave number ($W_j[j, w] := W_i[j, w] + 1$), by Rules 4.1.3 and 4.1.4, at time $t + \Delta_{L1} + \Delta_{L2}$, both i and j will have correct information on link (j, w) (see Figure 4.1).

Otherwise, the information is consistent with the local information $\mathcal{K}_i[j, w]$ (i.e., $LC_i(\mathcal{K})$ holds), at time $t + \Delta_{L1}$, upon receipt on $\langle \mathcal{K}_\Delta \rangle$ containing $\{\mathcal{K}\mathcal{W}_j[j, w]\}$ from j , both i and j will have the correct information on link (j, w) (Rules 4.1.3 and 4.1.4).

Figure 4.2 The Process of Remote Inconsistent Information

In other case, since $\Delta_{L1} + \Delta_{L2}$ is finite for any t , it is sufficient that topological changes cease for a finite amount of time Δ_{tL} , provided $\Delta_{tL} \geq \Delta_{L1} + \Delta_{L2}$. Thus, the lemma holds also for infinitely discrete case. $\square$

Lemma 4.3 *Given two operational nodes, i and j , connected by an operational link, if there exists a link (j,w) such that $W_i[j,w] = W_j[j,w]$ and $\mathcal{K}_i[j,w] \neq \mathcal{K}_j[j,w]$, then*

- (i) *there is a message in transit on the link (i,j) with wave number $W_i[j,w]$;*
- (ii) *at any time t , if no topological changes occur in the interval $(t, t + \Delta_{tL})$, both i and j will have correct information on link (j,w) .*

Proof. Part (i). We show that there are messages in transit which will correct it.

For two given incident nodes i and j , both of them have the same wave numbers ($W[j,w] = W_i[j,w]$) but inconsistent information on link (j,w) ($\mathcal{K}_i[j,w] \neq \mathcal{K}_j[j,w]$). It occurs only in one of the following scenarios:

- The initial state of the system;
- Both i and j received information messages containing information on link (j,w) . Clearly, the received messages were not from the same initiators, since $\mathcal{K}_i[j,w] \neq \mathcal{K}_j[j,w]$;

- Node i received a message containing information on link (j, w) , and the status of (j, w) changed at the same time. Therefore, both i and j increased their wave numbers.

It is not difficult to see that, in any one of the these scenarios, according to Rules 4.1.1, 4.1.2, 4.1.3, and 4.1.4, there must be at least one message containing the information on (j, w) in transit on link (i, j) .

Part (ii). Since the information on (j, w) is remote topological information for node i , it takes the actions according to Rule 4.1.4, which results in i to send a request the clarification of the inconsistent information on (j, w) , that is

[] (Condition 4.1.4)

$\Rightarrow$

[] $(W_i[j, w] = W[j, w]) \wedge RI_i(\mathcal{K})$

$\Rightarrow \mathcal{KW}_i[j, w] := \langle 2, W_i[j, w] \rangle; /* \text{ set up a request } */$

for all $q \in \text{Adjacent}(i)$ do send $\langle \{\mathcal{KW}_i[j, w]\} \rangle$ to q od

fi;

Upon receipt the message (request) sent from i , according to Rule 4.1.3, node j initiates a new wave to clarify the ambiguity about link (j, w) , since $(\mathcal{K}_i[j, w] \neq \mathcal{K}_j[j, w])$ (see Figure 4.2). That is

[] (Condition 4.1.3) $/* \text{ where } (W_i[j, w] \leq W[j, w]) \wedge LI_i(\mathcal{K}) */$

$\Rightarrow \mathcal{KW}_j[j, w] := \langle \mathcal{K}_i[j, w], W_i[j, w] + 1 \rangle$

for all $q \in \text{Adjacent}(j)$ do send $\langle \{\mathcal{KW}_j[j, w]\} \rangle$ to q od

Assuming that no topological changes occur during the entire process; then within a finite time Δ_{tL} ($\Delta_{tL} \geq \Delta_{L1} + \Delta_{L2}$), which is the communication delay experienced by the messages in transit between i and j , if $(W_i[j, w] = W_j[j, w])$ still holds, node i will correct the ambiguity about (j, w) . The same process will actually resolve all mutual ambiguities

between i and j . Hence, at time $t + \Delta_{IL}$, both i and j have the correct information on (j, w) .
 $\square$

Observe that the scenarios described in Lemmas 4.2 and 4.3 can occur symmetrically with respect two incident nodes i and j , and Lemmas 4.2 and 4.3 also hold. Due to the communication delay, the nodes at both sides of a recovery link may not act simultaneously.

Lemma 4.4 *Let i and j be two operational nodes connected by an operational link. At an arbitrary time t , there exists a delay Δ_{IL} such that, if no topological changes occur in the interval $(t, t + \Delta_{IL})$, then at time $t + \Delta_{IL}$, both i and j will have the correct information on the local topology of i and j .*

Proof. By Lemmas 4.1, 4.2 and 4.3, the lemma follows. $\square$

Lemma 4.5 *If any two operational nodes have different wave numbers, with respect to one of their local links, without loss of generality, let v and u be these two nodes and (v, w) be the link, (i.e., $W_v[v, w] \neq W_u[v, w]$), then there exist at least one message, whose wave number is greater than or equal to the higher one, in transit on one of the operational links in the network.*

Proof. By definition, all the operational nodes in the network are connected (i.e., the communication graph is connected). By repeated the application of Lemmas 4.1 (i) and 4.2, the lemma follows. $\square$

We will employ the same approach as the one for Lemma 3.3 to prove the following lemma.

Lemma 4.6 *Given an arbitrary time t , let $W_{max}[i,j]$ be the highest wave number of link (i,j) among the operational nodes. There exists a delay Δ_{tw} such that, if no topological changes occur in the interval $(t, t + \Delta_{tw})$, then, at time $t + \Delta_{tw}$, every operational node v will have a wave number $W_v[i,j] \geq W_{max}[i,j]$.*

Proof. Assume that topological changes cease forever; we will show that, in this case, the lemma holds. We will then show that it is sufficient that topological changes cease for just a finite time, thus proving that the lemma holds also for the case in which the sequence of topological changes is infinitely discrete.

Let k be a processor with the highest wave number (i.e., $W_k[i,j] = W_{max}[i,j]$) among all the operational nodes at time t ; by Lemmas 4.1 and 4.2, there exist messages with wave number $W_k[i,j]$ in transit on the operational links from k to all its operational neighbors with a smaller wave numbers. Let Δ_{w1} be the longest communication delay experienced by the messages from k and all its operational neighbors with a smaller wave numbers. Since no topological changes occur, then, after time Δ_{w1} , by Lemmas 4.1 and 4.2, the wave numbers of all the operational neighbors of k will be greater than or equal to $W_{max}[i,j]$.

We can now prove that at any distance from k , all operational neighbors of k at distance less than or equal to i will have a wave number which is greater than or equal to $W_{max}[i,j]$. Since the graph $G = (V, E)$ is connected, and each link is bidirectional. Let Δ_{wi} be the longest communication delay experienced by the messages in transit between the nodes at distance $i-1$ and the nodes at distance i from k . By applying the same argument as for the case $i = 1$, within Δ_{wi} additional time, all operational neighbors of k at distance less than or equal to i will have a wave number which is greater than or equal to $W_{max}[i,j]$.

Then, by the time $(t + \Delta_{tw})$, where

$$\Delta_{tw} = \Delta_{w1} + \Delta_{w2} + \Delta_{w3} + \dots + \Delta_{wd} = \sum_{i=1}^d \Delta_{wi}.$$

and d is the current diameter of the network, every operational node v will have a wave number $W_v[i,j] \geq W_{max}[i,j]$. This lemma holds for the case in which topological changes cease infinitely.

Since Δ_{pw} is finite for any t , for the lemma to hold, it is sufficient that topological changes cease for a finite amount of time Δ , provided $\Delta > \Delta_{pw}$. Therefore, the lemma holds also for infinitely discrete case. $\square$

By applying the same argument as Theorem 3.1, we have the following theorem.

Theorem 4.1 *For any arbitrary time t , there exists a delay Δ_{tc} such that, if no topological changes occur in the interval $(t, t + \Delta_{tc})$, then at time $t + \Delta_{tc}$, every operational node v will have the correct topological information on the entire network.* $\square$

Note that $\Delta_{tc} = \Delta_{k1} + \Delta_{k2} + \Delta_{k3} + \dots + \Delta_{kd} = \sum_{i=1}^d \Delta_{ki}$, d is the current diameter of the network, and Δ_{ki} has the same definition as the one in Theorem 3.1.

Now we know that, after a finite time, all the operational nodes will obtain the correct topological information on the entire network. By using the same argument as the one in Theorem 3.2, we can prove there is no deadlock and livelock for this optimized wave algorithm.

Theorem 4.2 (No Deadlock) There is no deadlock for the optimized stabilizing wave algorithm.

Proof. According to Rules 4.1.3, 4.1.4, and 4.1.5; we have the following predicates:

(i) $(W_i[v,u] < W[v,u])$

$LI_i(\mathcal{K})$

< Condition 4.1.3 >

$RI_i(\mathcal{K}) \vee GC_i(\mathcal{K})$

< Condition 4.1.4 >

(ii) $(W_i[v,u] = W[v,u])$

$LI_i(\mathcal{K})$	< Condition 4.1.3 >
$RI_i(\mathcal{K})$	< Condition 4.1.4 >
$GC_i(\mathcal{K})$	< Condition 4.1.5 >
(iii) $(W_i[v, u] > W[v, u])$	< Condition 4.1.5 >
(iv) $(\mathcal{K}[v, u] \text{ is defined}) \wedge (u \notin Neighbor_{\mathcal{K}_i}(v))$	< Condition 4.1.4 >

It is not difficult to verify that these predicates cover all the possible cases. Each of these predicates results in a processor to take the corresponding actions, upon receipt of a message. Thus, the operational processors never stop processing the received messages; in other words, there are no blocked messages. At the same time, since the algorithm is event driven, the processors are never blocked waiting for messages.

If there are no messages in transit in the system, by definition, the algorithm completes. $\square$

Theorem 4.3 (*No livelock and Completion*) *For any arbitrary time t , there exists a delay Δ such that, if no topological changes occur in the interval $(t, t + \Delta)$, then the algorithm completes at time $t + \Delta$.*

Proof. We will firstly assume that topological changes cease forever and will show that, in this case, the theorem holds. Then, we will prove that it is sufficient that topological changes cease for just a finite time, thus, the theorem holds also for the case in which the sequence of topological changes is infinitely discrete.

Assume that topological changes cease forever. By Theorem 4.1, if topological changes cease at time t , within a finite time Δ_{tc} , all nodes will have correct topological information on the entire network. Thus, to prove the theorem, it is sufficient to show that, within a finite time, there will no messages in transit.

At time $t + \Delta_{tc}$, for any received messages at any operational node, the following conditions hold, that is,

$$((W_i[v,u] < W[v,u]) \vee GC_i(\mathcal{K})) \vee ((W_i[v,u] = W[v,u]) \wedge GC_i(\mathcal{K})) \vee (W_i[v,u] > W[v,u]).$$

Conditions 4.1.1, 4.1.2, and 4.1.3 do not hold.

Given an arbitrary link (v,u) , let $W_{max}[v,u]$ be the highest wave number at time $t + \Delta_{tc}$, and v be a processor with a wave number $W_v[v,u] < W_{max}[v,u]$. By Lemma 4.6, since there are no topological changes, at time $t + \Delta_{tc}$, and every node has correct information on the entire network. At time $t + \Delta_{tc}$, upon receipt of a message with wave number $W_{max}[v,u]$, since $(W_v[v,u] < W_{max}[v,u]) \wedge (\mathcal{K} = \mathcal{K}_i)$ holds, by Rule 4.1.4, node v updates its wave number to be $W_{max}[v,u]$ and propagates the received information.

Since topological changes cease, at time $t + \Delta_{tc}$, all nodes will have correct information on the entire network, and Conditions 4.1.1, 4.1.2, and 4.1.3 do not hold. Thus, no messages will be sent with wave numbers which are greater than $W_{max}[v,u]$.

For any arbitrary link (v,u) , at time $t + \Delta_{tc}$, let Δ_{tm} be the longest communication delay experienced by the messages on all the operational links in the network. By applying the same argument as the one in Lemma 3.3 (see Figure 3.2), then, at time $t + \Delta$, all the operational nodes will have the same wave number $W_{max}[v,u]$, where $\Delta = \Delta_{tc} + \Delta_{tm}$, Δ_{tc} is

from Theorem 4.1,
$$\Delta_{tm} = \Delta_{m1} + \Delta_{m2} + \Delta_{m3} + \dots + \Delta_{md} = \sum_{i=1}^d \Delta_{mi},$$

d is the current diameter of the network, and Δ_{mi} is the longest communication delay experienced by the messages in transit between the nodes at distance $i - 1$ and the nodes at distance i from the node with the highest wave number $W_{max}[v,u]$. Clearly, by Rule 4.1.5, no messages will still be in transit in the network.

In other case, since $\Delta_{tc} + \Delta_{tm}$ is finite for any time t , it is sufficient that topological changes cease for a finite amount of time Δ , provided $\Delta \geq \Delta_{tc} + \Delta_{tm}$. Thus, the theorem holds also for infinitely discrete case. $\square$

Theorems 4.1, 4.2, and 4.3 prove the correctness of the optimized algorithm.

4.6 Chapter Summary

In this chapter, we have presented the optimized stabilizing wave algorithm. The time and message complexities are the same as the analysis presented in Section 3.8. In the optimized wave algorithm, what we are mainly concerned about is the topological information carried by the messages. By limiting the information in a message with respect to the topological changes, the process of the received topological information at each node is simplified.

There are two advantages of the optimized algorithm with respect to the one presented in Chapter 3:

- (1) It is clear that the topological information carried by messages are reduced, since they mainly represent the topological changes of the local topology of the originators. Therefore, the space requirement per message is reduced and the information is optimized, which makes the communication in a distributed system more efficient.
- (2) When a node receives an information message, it only has to update its local topological knowledge with respect to the changes. This simplifies the implementation of our stabilizing wave algorithms and makes it more practical, in the sense that unnecessary checking with other unchanged topological information is avoided.

Chapter 5

A Stabilizing Spanning Tree Construction and Leader Election Algorithm

5.1 Introduction

The two wave algorithms presented in the previous two chapters are very general. We can apply them to a variety of problems, and preserve the property of self stabilization at the same time. These problems include:

- (1) The *Token Management* problem, where an unique token circulates through the system in a fair way and can be seized by any processor that wishes to capture it. Further, it can yield *Mutual Exclusion* [IJ90] and [DIM90].
- (2) The *Unique Naming* problem where each processor must select a distinct identity [AEH92].
- (3) The *Ranking* Problem which is a special case of unique naming where the name of each processor must be an integer [DIM91a] and [AEH92].
- (4) The *Distributed Reset* problem which basically is a diffusion computation [AG90] and [APV91].
- (5) The *Spanning Tree* problem where every processor selects a subset of its adjacent links so that all the selected links form a spanning tree [AKY90] and [DIM90].
- (6) The *Leader Election* problem in which exactly one processor enters a distinguished state as a leader [AG90], [AKY90], and [DIM91a].

In this chapter, we show one of the important applications of the stabilizing wave topology algorithms. The objective of this application is to compute a stabilizing spanning tree and to elect a leader among all the operational nodes in a dynamic system. The spanning tree can be employed to control the message broadcasting and reduce the redundancy of messages. When the execution of the algorithm completes, the elected leader, which is a node with the lowest value among all the nodes known as operational nodes, will be the root of the constructed spanning tree.

When the execution of the topology wave algorithm completes, every node will have the correct information on the entire network. It is well known that, based on the topological information of the entire network, every node can compute the spanning tree and elect a leader. Alternatively, we apply the same mechanism to tackle the problems of spanning tree construction and a leader election, but not to compute the topological information on the entire network. Therefore, the amount of information needed to solve these two problems is much less than that required for network topology. In fact, we replace the local topological knowledge $\mathcal{K}$ by the smallest value known by a node. Instead of dealing with the topological knowledge $\mathcal{K}$, every node computes only *the lowest value* of the node identity among all the nodes known as operational nodes and broadcast it in the same fashion.

Consequently, the wave numbers are inherited as sequence numbers to distinguish the new and old information on the leader. The initiator of a “successful” wave, in which all the operational nodes in the network have the correct information on the lowest value of the identity of an operational node, becomes the root of the spanning tree, when the “successful” wave completes. As will be shown later, the correctness of this application is insensitive to the initial state of the network. The initial state consists of the different wave

numbers and messages in transit on the links of network. With respect to the same model described in Chapter 2, the following algorithm can construct a spanning tree and elect a leader within a *finite time*, which depends on the current diameter and the communication delays of the network, as shown in Section 3.7.

The organization of this chapter is similar to the one of the previous two chapters. In Section 5.2, we describe the representation of the information and messages. In Section 5.3, we present the wave rules for computing spanning tree. In Section 5.4, we formally describe the stabilizing spanning tree construction and leader election algorithm. In Section 5.5, we prove the correctness of this algorithm. In Sections 5.6 and 5.7, we discuss some related issues such as reset and complexity measures. Finally, in Section 5.8, we summarize the chapter.

5.2 Representation of Information and Messages

As an application, the representation of the information in the spanning tree construction and the leader election algorithm is similar to that presented in the Section 3.2. The only change is to replace the remote knowledge at each node by the lowest value K_v known to a node, where K_v denotes the knowledge on the lowest value.

Given a processor i , let $K_{v(i)}$ be the lowest identity known to i , and let $\mathcal{K}_i$ be a set that represents all the links incident to i , that is $\mathcal{K}_i = \{ \mathcal{K}_i[j] \}$, where

$$\mathcal{K}_i[j] = \begin{cases} 0 & \text{the link between } i \text{ and } j \text{ is failed (down) ;} \\ 1 & \text{the link between } i \text{ and } j \text{ is operational (up) ;} \\ \text{Undefined} & \text{the existence of the link between } i \text{ and } j \text{ is unknown.} \end{cases}$$

With respect to the knowledge $\mathcal{K}_i$ known to node i , $Neighbor_{\mathcal{K}_i}(i)$ denotes the set of all the nodes incident to i , that is

$$Neighbor_{\mathcal{K}_i}(i) \stackrel{\text{def}}{=} \bigcup_{\mathcal{K}_i[j] \text{ is defined}} \{j\}.$$

As described in Chapter 3, there are two types of messages:

- (i) Information message is a couple $\langle W, K_v \rangle$, where
 - W is an integer, which denotes the wave number to distinguish old and new messages;
 - K_v is the smallest identity known to the sender of the message.
- (ii) Acknowledgment message is a couple $\langle W, \mathcal{A} \rangle$, where
 - W is an integer, which denotes the wave number of the received message that the processor is acknowledging;
 - $\mathcal{A}$ is a boolean variable.

Upon receipt of an information message $\langle W, K_v \rangle$, a processor will always reply an acknowledgement message $\langle W, \mathcal{A} \rangle$, where W is the same as the wave number in the received information message, and $\mathcal{A}$ is defined as follows:

$$\mathcal{A} = \begin{cases} \text{true} & \text{the lowest value } K_v \text{ brought in by the received message} \\ & \text{is accepted by the receiver to be the new leader;} \\ \text{false} & \text{the lowest value } K_v \text{ brought in by the received message} \\ & \text{is not accepted by the receiver as the new leader.} \end{cases}$$

Every processor keeps track of the information on its operational neighbors, with respect to the received information messages, the sent information messages, and the received acknowledgement messages. In another words, during the process of computing

the lowest value of the identity among the operational nodes, every node selects a subset of its operational links incident to it. When the execution of the algorithm completes, the set of selected links forms a spanning tree among all the operational nodes.

As mentioned in Chapter 3, the information is captured by the following variables:

pred(i): the identity of the node from which node *i* has received a message and obtained the current topological information;

succ(i): a sub-set of neighbors of node *i* to which it sends information messages, and $Adjacent(i) \supseteq succ(i)$;

resp(i): a subset of *succ(i)* from which node *i* has received an acknowledgement message with the correct wave number and $\mathcal{A} = \text{true}$.

5.3 The Wave Rules

As mentioned in Section 5.1, this algorithm is an application of the stabilizing wave algorithm presented in Chapter 3. The model of computation is the same as the one described in Section 3.2, all the topological changes are represented by the changes of the links status. All the wave rules (*making waves*, *propagating waves* and *eliminating waves*), which are employed in the previous two chapters, are modified respectively for this application.

The wave rules are illustrated as follows (for the sake of simplicity, we present these rules with respect to node *i*):

Making Wave

Rule 5.1.1 When node i detects that one of the links (i,j) incident to i comes up, if $\mathcal{K}_i[j] = 0 \wedge j \in \text{Adjacent}(i)$ holds, the node takes the corresponding actions based on the change.

- (i) If i detects itself becomes operational, it initiates a new wave to broadcast the lowest identity value among its operational neighbors (including itself);
- (ii) Otherwise, it uses the same wave to compute the lowest value with respect to the topological change.

Condition 5.1.1

$$(\exists j \mid \mathcal{K}_i[j] = 0 \wedge j \in \text{Adjacent}(i))$$

By definition, when link (i,j) becomes operational, j will be in $\text{Adjacent}(i)$; but it was not in $\text{Adjacent}(i)$ before the recovery, and therefore $\mathcal{K}_i[j] = 0$ (down). This rule is applied to these three cases:

- (i) a link becomes operational, that is a crashed link has been repaired, it was down before;
- (ii) a node becomes operational, that is a crashed processor has been repaired, and at least one of its incident links must become operational;
- (iii) the system starts.

For cases (ii) and (iii), the up nodes reset the status of the links incident to it in its local topological information immediately, as described in Chapter 2, thus, the following condition holds:

$$\bigwedge_{j \in \text{Neighbor}_{\mathcal{K}_i}(i)} \mathcal{K}_i[j] = 0$$

The reason for distinguishing cases (i) and (ii) from case (iii) is that every node must certify its local information, when it starts. The objective is to ensure that the global information on the spanning tree and the leader is consistent.

Propagating Waves

Rule 5.1.2 When node i detects that one of its adjacent links (i,j) goes down, and the following condition holds

$$(\mathcal{K}_i[j] = 1) \wedge (j \notin \text{Adjacent}(i)) \wedge (j = \text{pred}(i)),$$

then node i uses the same wave number to re-construct the tree.

Condition 5.1.2

$$(\exists j \mid (\mathcal{K}_i[j] = 1) \wedge (j \notin \text{Adjacent}(i)) \wedge (j = \text{pred}(i))).$$

This indicates that the link between node i and its parent node j is down. In this case, node i recomputes the spanning tree by starting from itself. If the root of the tree fails, it results in all the operational neighbors, which are its children, to recompute the spanning tree.

Rule 5.1.3 When a node i detects that one of its incident links (i,j) goes down, and the condition $(\mathcal{K}_i[j] = 1) \wedge (j \notin \text{Adjacent}(i)) \wedge (j \neq \text{pred}(i))$ holds, node i updates the $\text{sucr}(i)$ and $\text{resp}(i)$ with respect to the change.

Condition 5.1.3

$$(\exists j \mid (\mathcal{K}_i[j] = 1) \wedge (j \notin \text{Adjacent}(i)) \wedge (j \neq \text{pred}(i)))$$

It implies that the link between node i and its parent j is down. With respect to a link failure, if the down link is not in the spanning tree, it has no impact on the current spanning tree. In this case, node i only has to update its local information with respect to the down link.

Rule 5.1.4 Upon receipt of an information message $\langle W, K_v \rangle$, node i compares its local information with the received message, if one of these three conditions holds:

- (i) $(W_i < W) \wedge (K_{v(i)} \geq K_v)$;
- (ii) $(W_i = W) \wedge (K_{v(i)} > K_v)$.

then this node propagates the information to the rest of its neighbors.

Condition 5.1.4

$$((W_i < W) \wedge (K_{v(i)} \geq K_v)) \vee ((W_i = W) \wedge (K_{v(i)} > K_v))$$

In case (i), the received message with a higher wave number contains a value, which is not higher than the one at node i . In case (ii), the received message with the same wave number carries a value, which is lower than $K_{v(i)}$. Then, this node updates its $K_{v(i)}$ and wave number, and it broadcasts the message to all its operational neighbors.

Rule 5.1.5 Upon receipt of an information message $\langle W, K_v \rangle$, if one of these two conditions holds:

- (i) $(W_i < W) \wedge (K_{v(i)} < K_v)$
- (ii) $(W_i = W) \wedge (K_{v(i)} < K_v) \wedge (pred(i) = v)$.

then node i compares the received value K_v with the local lowest value. Based on the results, it propagates the message with the higher wave and the selected values.

Condition 5.1.5

$$((W_i < W) \wedge (K_{v(i)} < K_v)) \vee, \\ ((W_i = W) \wedge (K_{v(i)} < K_v) \wedge (pred(i) = v))$$

In case (i), node i receives a message with a higher wave number containing a value which is higher than its local value $K_{v(i)}$ as well. In case (ii), node i receives a message with the same wave number from its parent, which contains a higher value. It implies that some link above this node in tree failed. Thus, node i takes actions in response to it with respect to its local value.

Eliminating Waves

Rule 5.1.6 Upon receipt of an information message $\langle W, K_v \rangle$, if a node detects that

- (i) the coming message is outdated ($W_i > W$) ;
- (ii) the received message with the same wave number containing a higher value is from a node rather than the father of the receiver;
- (iii) the received message has the same value and wave number;

then the node discards the received message and stops propagating it.

Condition 5.1.6

$$(W_i > W) \vee \\ ((W_i = W) \wedge (K_{v(i)} < K_v) \wedge (pred(i) \neq v)) \vee \\ ((W_i = W) \wedge (K_{v(i)} = K_v))$$

From the above rules, the predicates of all the conditions are as follows:

- (i) $(W_i < W)$
 - $(K_{v(i)} \geq K_v)$ < Condition 5.1.4 >
 - $(K_{v(i)} < K_v)$ < Condition 5.1.5 >
- (ii) $(W_i = W)$
 - $(K_{v(i)} > K_v)$ < Condition 5.1.4 >
 - $(K_{v(i)} = K_v)$ < Condition 5.1.6 >
 - $(K_{v(i)} < K_v) \wedge (pred(i) = v)$ < Condition 5.1.5 >
 - $(K_{v(i)} < K_v) \wedge (pred(i) \neq v)$ < Condition 5.1.6 >
- (iii) $(W_i > W)$ < Condition 5.1.6 >

It is clear that the above predicates cover all the possible situations.

5.4 The Spanning Tree Construction and Leader Election Algorithm

Before we formally present this spanning tree construction and leader election algorithm, we characterize the behavior of a processor under the rules described in previous section. Driven by the *events*, there are the following scenarios for an operational processor in the network:

Detection of topological changes

When a processor becomes operational, it initiates a new wave to construct a spanning tree starting from itself. Notice that, in the beginning, every operational node in the network starts the system by initiating a new wave to build a tree. If a node detects that a

link becomes operational, it compares the lowest identity value $K_{v(i)}$ with the node at the other side of the up link (Rule 5.1.1) and takes corresponding actions. With respect to the failure of a link, if the failed link is in the tree, all the nodes in the disconnected subtree restart to compute the lowest identity by using the same wave number (Rule 5.1.2). Otherwise (the failed link is not in the tree), two nodes at both sides of the link only have to update their local information (Rule 5.1.3).

Reception of an information message

A node, say i , compares its local information on the lowest identity $K_{v(i)}$ with the information brought in by the received message K_v . As a result, it will take one of the following actions:

- (i) Update the local information $K_{v(i)}$ with respect to the received information K_v and propagate the updated information with the same wave (Rule 5.1.4);
- (ii) Make a new wave to rebuild a spanning tree and elect a new leader, or propagate a wave (Rule 5.1.5);
- (iii) Eliminate a wave (Rule 5.1.6).

Reception of an acknowledgement $\langle W, \mathcal{A} \rangle$

The receiver takes the same approach as the one described in Section 3.4, when an operational node receives an acknowledgement message from one of its neighbors.

Notice that, when the execution of the algorithm completes, all the operational nodes and the selected operational links in the network form a spanning tree and the root of the constructed spanning tree is the elected leader (the node with the lowest identity among all the operational nodes).

The following is the formal presentation of spanning tree construction and leader election algorithm :

Topological Changes

(Condition 5.1.1)

/ When a link is UP, i.e., $(\exists j \mid \mathcal{K}_i[j] = 0 \wedge j \in \text{Adjacent}(i))$ */*

$\Rightarrow \mathcal{K}_i[j] := 1;$

$\text{if } \bigwedge_{j \in \text{Neighbor}_{\mathcal{K}_i}(i)} \mathcal{K}_i[j] = 0$ */* when node i is up */*

$\Rightarrow W_i := W_i + 1; K_{v(i)} := \min\{i, j\}; \text{resp}(i) := \emptyset;$

$\text{if } K_{v(i)} = i \Rightarrow \text{pred}(i) := \emptyset;$

$[] \text{else} \Rightarrow \text{pred}(i) := j;$

fi;

$\text{succ}(i) := \text{Adjacent}(i) - \text{pred}(i);$

for all $u \in \text{succ}(i)$ **do** send $\langle W_i, K_{v(i)} \rangle$ to u **od**

$[] \text{else} \Rightarrow \text{if } K_{v(i)} \geq j$

$\Rightarrow K_{v(i)} := j; \text{pred}(i) := j; \text{resp}(i) := \emptyset;$

$\text{succ}(i) := \text{Adjacent}(i) - \text{pred}(i);$

for all $u \in \text{succ}(i)$ **do** send $\langle W_i, K_{v(i)} \rangle$ to u **od**

$[] K_{v(i)} < j$

$\Rightarrow \text{send } \langle W_i, K_{v(i)} \rangle \text{ to } j;$

fi

fi;

$[]$ (Condition 5.1.2)

/ (When a link is DOWN, i.e., $(\exists j \mid (\mathcal{K}_i[j] = 1) \wedge (j \notin \text{Adjacent}(i)) \wedge (j = \text{pred}(i)))$ */*

$\Rightarrow \mathcal{K}_i[j] := 0;$

$K_{v(i)} := \min\{q \mid q \in (\text{Adjacent}(i) \cup \{i\})\};$

$\text{if } K_{v(i)} = i \Rightarrow \text{pred}(i) := \emptyset;$

$[] \text{else} \Rightarrow \text{pred}(i) := K_{v(i)};$

fi ;

$succ(i) := Adjacent(i) - pred(i); resp(i) := \emptyset;$
for all $u \in succ(i)$ **do** send $\langle W_i, K_{v(i)} \rangle$ to u **od**;

[] (Condition 5.1.3)

/ $(\exists j \mid (K_i[j] = 1) \wedge (j \notin Adjacent(i)) \wedge (j \neq pred(i)))$ */*

$\Rightarrow K_i[j] := 0;$

if $j \in succ(i) \Rightarrow succ(i) := succ(i) - \{j\}$ **fi**;

if $j \in resp(i) \Rightarrow resp(i) := resp(i) - \{j\}$ **fi**;

Upon Receipt of $\langle W, K_v \rangle$ from p

[] (Condition 5.1.4) */* Upon Receipt of $\langle W, K_v \rangle$ from p */*

/ where $((W_i < W) \wedge (K_{v(i)} \geq K_v)) \vee ((W_i = W) \wedge (K_{v(i)} > K_v))$ */*

$\Rightarrow W_i := W; K_{v(i)} := K_v;$

$succ(i) := Adjacent(i) - \{p\}; pred(i) := v; resp(i) := \emptyset;$

for all $u \in succ(i)$ **do** send $\langle W_i, K_{v(i)} \rangle$ to u **od**;

[] (Condition 5.1.5) */* Upon Receipt of $\langle W, K_v \rangle$ from p */*

/ where $((W_i < W) \wedge (K_{v(i)} < K_v)) \vee$
 $((W_i = W) \wedge (K_{v(i)} < K_v) \wedge (pred(i) = v))$ */*

$\Rightarrow W_i := W;$

if $(K_v < \min\{q \mid q \in Adjacent(i) \cup \{i\}\})$

$\Rightarrow K_{v(i)} := K_v; pred(i) := p;$

if $Adjacent(i) = \{p\} \Rightarrow$ send $\langle W, true \rangle$ to p **fi**;

[] $(K_v > \min\{w \mid w \in Adjacent(i) \cup \{i\}\})$

$\Rightarrow W_i := W_i + 1; K_{v(i)} := \min\{q \mid q \in (Adjacent(i) \cup \{i\})\}$

if $K_{v(i)} = i \Rightarrow pred(i) := \emptyset;$

[] **else** $\Rightarrow pred(i) := K_{v(i)};$

fi;

[] **else** */* $K_v = \min\{w \mid w \in (Adjacent(i) \cup \{i\})\}$ */*

$\Rightarrow K_{v(i)} := K_v;$

if $K_{v(i)} = i \Rightarrow pred(i) := \emptyset;$

[] **else** $\Rightarrow pred(i) := K_{v(i)};$

fi;

fi;

$succ(i) := Adjacent(i) - pred(i); resp(i) := \emptyset;$

for all $u \in succ(i)$ **do** send $\langle W_i, K_{v(i)} \rangle$ to u **od** ;

[] (Condition 5.1.6) /* Upon Receipt of $\langle W, K_v \rangle$ from p */

/* where $(W_i > W) \vee ((W_i = W) \wedge (K_{v(i)} < K_v) \wedge (pred(i) \neq v)) \vee$
 $((W_i = W) \wedge (K_{v(i)} = K_v))$ */

$\Rightarrow$ **if** $(W_i = W) \wedge (K_{v(i)} = K_v)$

$\Rightarrow$ send $\langle W, false \rangle$ to p ;

if $v \in succ(i) \Rightarrow succ(i) := succ(i) - \{p\}$ **fi**;

[] $(W_i = W) \wedge (K_{v(i)} < K_v)$

$\Rightarrow succ(i) := succ(i) \cup \{p\}$;

send $\langle W, K_{v(i)} \rangle$ to p ;

fi ;

Upon receipt of $\langle W, \mathcal{A} \rangle$ from p

if $(W_i = W)$

$\Rightarrow$ **if** $(\mathcal{A} = true)$ $\Rightarrow resp(i) := resp(i) \cup \{p\}$;

[] $(\mathcal{A} = false)$ $\Rightarrow succ(i) := succ(i) - \{p\}$;

fi;

if $(resp(i) = succ(i)) \wedge (pred(i) \neq \emptyset)$

$\Rightarrow$ send $\langle W, true \rangle$ to $pred(i)$; /* the node becomes a root of a sub-tree */

[] $(resp(i) = succ(i)) \wedge (pred(i) = \emptyset)$

$\Rightarrow$ **reset-wave**(i); /* For models Φ_{σ}^N , reset-wave procedure is invoked. */

fi;

[] **else** $\Rightarrow$ skip /* $W_i \neq W$ */

fi

The Spanning Tree and Leader Election Algorithm

5.5 Proof of Correctness

To prove the correctness of the spanning tree construction and leader election algorithm, we start with some technical lemmas. Based on the correctness of Theorem 3.1, the proof of correctness for this algorithm is trivial.

Lemma 5.1 *Given two operational nodes, i and j , connected by an up link, without loss of generality, let $W_i \geq W_j$. If $W_i > W_j$, then*

- (i) *there is a message in transit from i to j on the link (i,j) with wave number W_i ;*
- (ii) *if the link (i,j) does not fail before the message arrives, the wave number of j upon processing of the message will be greater than or equal to W_i ($W_j \geq W_i$).*

Proof. Part (i). We will distinguish the case when at least one of the two node is (re)starting from any other cases.

If one of the nodes is (re)starting then, by definition, then the link (i,j) must be now coming up, both i and j will send a message to all its operational neighbors.

If neither nodes is (re)starting; since $W_i > W_j$, node i increases its wave number only in these two situations: either after detecting a local change or after receiving a message. In the former situation, it sent a message with the new wave number W_i to all its operational neighbors including j (Rule 5.1.1); because the wave number of node j is less than the wave number at node i ($W_j < W_i$), j has obviously not received such a message which is therefore in transit from i to j on link (i,j) . In the latter scenario, node i sent the message with wave number to all its operational neighbors (Rules 5.1.4 and 5.1.5); the only exception would be that if the received message had the information such that $(W_i < W)$ and $(K_{V(i)} \geq K_V)$, then node i propagated the information K_V to all its operational neighbors except the sender of the message. Clearly, the sender is not node j , since $W_j < W_i$.

Part (ii). Clearly, based on the proof of (i), if the link (i,j) does not fail before the message from node i arrives at node j , if $W_i > W_j$ still holds, node j takes action according to Rule 5.1.4 or 5.1.5, and sets its wave number to a value $(W_j \geq W_i)$. $\square$

Lemma 5.2 *If any two operational nodes have different wave numbers, then there exists at least one message, whose wave number is greater than or equal to the higher one, in transit on one of the operational links in the network.*

Proof. By definition, all the operational nodes in the network are connected (i.e., the communication graph is connected). By repeated the application of Lemma 5.1 (i), the lemma follows. $\square$

Lemma 5.3. *Given an arbitrary time t , let W_{max} be the highest wave number among the operational nodes. Then, there exists a delay Δ_{tw} such that, if no topological changes occur in the interval $(t, t + \Delta_{tw})$, then then at time $t + \Delta_{tw}$, for every operational node v will have a wave number $W_v \geq W_{max}$.*

Proof. Assume that topological changes cease forever; we will show that, in this case, this lemma holds. We will then show that it is sufficient that topological changes cease for just a finite time, thus proving that the lemma holds also for the case in which the sequence of topological changes is infinitely discrete.

Let k be a processor with the highest wave number (i.e., $W_k = W_{max}$) among all the operational nodes at time t , by Lemma 5.1 (i), there exist messages with wave number W_k in transit on the operational links from k to all its operational neighbors. Let Δ_{w1} be the longest communication delay experienced by the message between k and all its operational neighbors. Since no topological changes occur, then, after a finite time Δ_{w1} , the wave

numbers of all the operational neighbors of k will be greater than or equal to W_{max} , by Lemma 5.1 (ii).

By repeating the above argument and the application of Lemma 5.2, since the graph $G = (V, E)$ is connected, and each link is bidirectional. Let Δ_{wi} be the longest communication delay experienced by the messages in transit between the nodes at distance $i - 1$ and the nodes at distance i from k . By applying the same argument as for the case $i = 1$, within Δ_{wi} additional time, all operational neighbors of k at distance less than or equal to i will have a wave number which is greater than or equal to W_{max} . Then, by the time $(t + \Delta_{tw})$,

$$\text{where } \Delta_{tw} = \Delta_{w1} + \Delta_{w2} + \Delta_{w3} + \dots + \Delta_{wd} = \sum_{i=1}^d \Delta_{wi},$$

and d is the current diameter of the network, every operational node v will have a wave number which is greater than or equal to W_{max} ($W_v \geq W_{max}$).

In other case, since Δ_{tw} is finite for any t , for the lemma to hold, it is sufficient that topological changes cease for a finite amount of time Δ , provided $\Delta > \Delta_{tw}$. Thus, the lemma holds also for infinitely discrete case. $\square$

Theorem 5.1 Given an arbitrary time t , then there exists a finite delay Δ_t such that, if no topological changes occur in the interval $(t, t + \Delta_t)$, then every operational node v will have the correct information on the lowest value of the identity among all the nodes known as operational in the network.

Proof. we will firstly assume that topological changes cease forever, and we will show that, in this case, the theorem holds. Then, we will prove that it is sufficient that topological changes cease for just a finite time, the theorem holds also for the case in which the sequence of topological changes is infinitely discrete.

Assume that topological changes cease forever. At any arbitrary time t , if no topological changes occur in the interval $(t, t + \Delta_{rw})$, then there exists a finite delay Δ_{rw} such that, by Lemma 5.3, every operational v node will have the wave number $W_v \geq W_{max}$. Let k be a processor with the highest wave number (i.e., $W_k = W_{max}$) among all the operational nodes at time t , and let Δ_{v1} be the longest communication delay experienced by the messages in transit between k and all its operational neighbors $N(k)$, by Rules 5.1.1, 5.1.2, and 5.1.3, at time $t + \Delta_{v1}$, all the operational neighbors of k (including k) will have the correct information on their local lowest value K_v .

By repeating the above argument, for all the operational nodes at the distance of i to node k , that is all the operational neighbors at the distance of $i - 1$ to S_1 , let Δ_{vi} be the longest communication delay experienced by the messages in transit between S_i and all its operational neighbors $N(S_i)$, then at time $t + \Delta_{v1} + \dots + \Delta_{vi}$, all the operational neighbors of S_i (i.e., $N(S_i)$) will have the correct information on their local lowest value K_v . Thus, by the time $(t + \Delta_t)$, every operational node v will have correct information on the lowest value of the identity in the network, where $\Delta_t = \Delta_{rw} + \Delta_{tv}$,

$$\Delta_{tv} = \Delta_{v1} + \Delta_{v2} + \Delta_{v3} + \dots + \Delta_{vd} = \sum_{i=1}^d \Delta_{vi},$$

and d is the current diameter of the network, Δ_{vi} is the longest communication delay experienced by the messages in transit between S_i and all its operational neighbors $N(S_i)$.

In other case, since Δ_t is finite for any t , it is sufficient that topological changes cease for a finite amount of time Δ , provided $\Delta \geq \Delta_t$. Thus, the theorem holds also for infinitely discrete case. $\square$

Theorem 5.2 (No Deadlock) There is no deadlock for the spanning tree and leader election wave algorithm.

Proof. According to Rules 5.1.4, 5.1.5 and 5.1.6, we have the following predicates:

(i) $(W_i < W)$

$(K_{v(i)} \geq K_v)$ < Condition 5.1.4 >

$(K_{v(i)} < K_v)$ < Condition 5.1.5 >

(ii) $(W_i = W)$

$(K_{v(i)} > K_v)$ < Condition 5.1.4 >

$(K_{v(i)} = K_v)$ < Condition 5.1.6 >

$(K_{v(i)} < K_v) \wedge (pred(i) = v)$ < Condition 5.1.5 >

$(K_{v(i)} < K_v) \wedge (pred(i) \neq v)$ < Condition 5.1.6 >

(iii) $(W_i > W)$ < Condition 5.1.6 >

Clearly, these predicates cover all the possible cases. Each of these predicates results in a processor to take the corresponding actions, upon receipt of a message. Thus, the operational processors never stop processing the received messages; in other words, there are no blocked messages. At the same time, since the algorithm is event driven, the processors are never blocked waiting for messages.

If there are no messages in transit in the system, by definition, the algorithm completes. $\square$

Theorem 5.3 (No livelock and Completion) For any arbitrary time t , there exists a delay Δ_{tc} such that, if no topological changes occur in the interval $(t, t + \Delta_{tc})$, then the algorithm completes at time $t + \Delta_{tc}$.

Proof. We will firstly assume that topological changes cease forever and will show that, in this case, the theorem holds. Then, we will prove that it is sufficient that topological

changes cease for just a finite time, the theorem holds also for the case in which the sequence of topological changes is infinitely discrete.

Assume that topological changes cease forever. By Theorem 5.1, if topological changes cease at time t , within a finite time Δ_t where $\Delta_t = \Delta_{rw} + \Delta_v$, all nodes will have correct information on the lowest value of the identity in the network. Thus, to prove the theorem, it is sufficient to show that, within a finite time, there will no messages in transit. At time $t + \Delta_t$, for any received messages at any operational node, the following condition hold, that is

$$((W_i \leq W) \wedge (K_{v(i)} = K_v)) \vee$$

$$((W_i = W) \wedge (K_{v(i)} < K_v) \wedge (pred(i) \neq v)) \vee (W_i > W).$$

Let W_{max} be the highest wave number at time $t + \Delta_t$, and let i be a processor with a wave number $W_i < W_{max}$. Since there are no topological changes, at time $t + \Delta_t$, and every node has correct information on the lowest value of identity in the network, upon receipt of a message with wave number W_{max} , since $(W_i < W_{max}) \wedge (K_{v(i)} = K_v)$, by Rule 5.1.4, node i updates its wave number to be W_{max} and propagates the received information.

Since topological changes cease, at time $t + \Delta_t$, all nodes will have correct information on the entire network. Thus, no messages will be sent with wave numbers which are greater than W_{max} , by Rules 5.1.1, 5.1.2, and 5.1.5.

At time $t + \Delta_{tc}$, let Δ_{tm} be the longest communication delay experienced by the messages on all the operational links in the network, by applying the same argument as the one in Lemma 3.3, then, at time $t + \Delta_{tc}$, where $\Delta_{tc} = \Delta_t + \Delta_{tm}$,

$$\Delta_{tm} = \Delta_{m1} + \Delta_{m2} + \Delta_{m3} + \dots + \Delta_{md} = \sum_{i=1}^d \Delta_{mi}$$

d is the current diameter of the network, and Δ_{mi} is the longest communication delay experienced by the messages in transit between the nodes at distance $i-1$ and the nodes at distance i from the node with the highest wave number W_{max} . All the operational nodes

will have the same wave number W_{tm} . Clearly, no messages will still be in transit in the network, by Rule 5.1.6.

In other case, since Δ_{tc} is finite for any t , it is sufficient that topological changes cease for a finite amount of time Δ , provided $\Delta_{tc} \geq \Delta_l + \Delta_{tm}$. Thus, the theorem holds also for infinitely discrete case. $\square$

Theorems 5.1, 5.2, and 5.3 prove the correctness of the spanning tree and leader election algorithm.

Given node i , when the execution of the algorithm completes, the selected links indicated by $pred(i)$ and $succ(i)$ form the spanning tree. Furthermore, based on these information, we can identify them in three cases :

- (i) $\forall$ node i , $pred(i) \neq \text{nil}$ and $succ(i) = \text{nil}$, then u is a leaf in the spanning tree.
- (ii) $\forall$ node i , $pred(i) = \text{nil}$ and $succ(i) \neq \text{nil}$, then u is the root of the spanning tree.
- (iii) $\forall$ node i , $pred(i) \neq \text{nil}$ and $succ(i) \neq \text{nil}$, then u is an internal node of tree, i.e., a root of sub-tree of the spanning tree.

5.6 On Reset Primitive and Bounded Counters

Considering the models $\Phi_{\varepsilon(G)}^{\infty}$ and $\Phi_{\bullet}^N$, after constructing a spanning tree in a dynamic system, we can use it for counter reset. The normal approach to implement the distributed reset is to use an existing spanning tree in the graph, which is a diffusion computation.

When the root of tree decides to start the computation, it initiates a reset wave which propagates along the tree towards the leaves. When a node is reached by the reset wave, the node resets its state to a default initial state and the value of the counter to be zero. When the reset wave reaches the tree leaves, it is reflected as a completion wave that is propagated along the tree back to the root. When the completion wave reaches the root of the tree, the diffusion computation completes.

By employing the above reset primitive, we can reset the counters in the system and keep them bounded. As stated in the model, the sequence of topological changes is either finite or infinitely discrete. In the latter case, the dynamic network can stay stable long enough before the next occurrence of the topological change so that the system stabilizes and reset its counters, provided the model is $\Phi_{\epsilon(G)}^{\infty}$ and $\Phi_{\bullet}^N$.

5.7 Complexity Measures

Time and message complexities are the same as the measures shown in Chapters 3 and 4.

It requires only $2(N - 1)$ message to form the reset waves.

Space Complexity

For every node in the graph, it needs $\log N$ bits to store the name of the processor, where N is the number of node in the graph. Therefore, the total requirement of memory per node is $3 \cdot N \cdot \log N$ for $Adjacent(i)$, $resp(i)$, $succ(i)$, and $pred(i)$, which is $O(N \cdot \log N)$.

The comparison of complexities among the existing protocols of spanning tree construction and a leader election on time, space and message:

Time	Method	Using Counters	Models and Comments
$O(N^3)$	[AG90]	Yes	$\Phi_{N,D}^{N,D}$ Shared registers (num of nodes bounded)
$O(N^2)$	[KP90]	Yes	$\Phi_{\emptyset}^N$ Global snapshot and correction
$O(N^2)$	[AKY90]	No	$\Phi_{\emptyset}^N$ with shared memory (id-based)
$O(D)$	[DIM90]	No	$\Phi_{N,D}^{N,D}$ with shared registers (semi-uniform)
$O(D)$	[AEH92]	No	$\Phi_{\emptyset}^N \Phi_{\emptyset}^{\infty}$ with shared registers (uniform)
$O(D)$	this thesis	Yes	$\Phi_{\emptyset}^N \Phi_{\emptyset}^{\infty}$ with message passing (id-based)

Table 5.1 The Time Complexities of Spanning Tree Algorithms

Space Complexity (bit)	Method	Comments
the memory required per node		
$O(N \log N)$	[AKY90]	with shared memory (id-based)
$O(rN \log N)$	[AEH92]	with shared registers (randomized id)
$O(N \log N)$	this thesis	with message passing (id-based)
Message Complexity		
$O(2^k E(G))$	[AEH92]	using multiple copies and the shortest path
$O(E(G))$	this thesis	stabilizing wave algorithm

Table 5.2 The Space and Message Complexities of Spanning Tree Algorithms

Note: N is the *a priori* bound on the maximum number of nodes in the G , r is the maximum degree of a node, D is the bound on the diameter of G , $E(G)$ is the *a priori* bound on the maximum number of edges of G .

5.8 Chapter Summary

In this chapter, we have shown one of the applications of the proposed topology wave algorithms, that is stabilizing spanning tree construction and leader election in a dynamic network. By using the same mechanism as in the wave algorithms, every operational processor disseminates a piece of information, which is the lowest value of the identity known to it, to all the operational processors. During this diffusion computation, each processor selects a subset of its adjacent operational links. Eventually, the set of all the links selected by the operational nodes forms a spanning tree in the system. Further, the root of the tree is the elected leader.

This stabilizing spanning tree construction and leader election protocol might not be the most efficient one in a dynamic distributed system, however, it shows clearly that our proposed wave algorithm provides a general paradigm to the design of others stabilizing protocol in a dynamic system.

Chapter 6

Conclusions

In this thesis, we have considered and studied the problem of self stabilization in a dynamic network. Generally speaking, the problem of self stabilization is concerned about how a dynamic system can regain its consistency, with respect to the topological changes in the network, without any kind of an outside intervention.

6.1 Summary and Discussion on the Results

We now summarize the results of this thesis according to the order of the chapters.

In first two chapters (0 and 1) we have introduced the problem with its background and the motivations. Briefly, the motivation of the study of self stabilization is to design a more robust system which can stabilize by itself within a finite time. In the survey, we have reviewed the basic and important stabilizing protocols being proposed. Primarily, these mechanisms can be classified with respect to their communication primitives: (i) shared variables; and (ii) message passing. Furthermore, their computation models are distinguished as *semi-uniform*, *uniform*, and *id-based*.

In Chapter 2, we have explicitly described the model and assumptions for our proposed stabilizing wave algorithms. It is depicted on four aspects such as the system structure, the fault model, the burst of topological changes, and the topological changes.

Informally, our model is an asynchronous distributed dynamic system. Comparing with the existing work, our model and asynchrony assumptions are very general and realistic.

After defining our computation model and assumptions, we have presented two stabilizing topology broadcast wave algorithms for message passing dynamic systems.

In Chapter 3, we have introduced the stabilizing topology wave algorithm. The characteristics of this algorithm is that information messages carry the *entire* local topological information known to the message sender, and the information is certified by the sender with respect to its local topology. Based on this fact, we prove the correctness of the algorithm.

Moreover, in the proposed topology wave protocols, sequence numbers and counters are employed to distinguish the old and new topological information. With respect to the space requirement, it is important to wrap the counters around. Therefore, we have discussed the reset method in some details with respect to the assumptions, and shown a version of the stabilizing topology wave algorithm with bounded counters.. It is clear that the reset primitive depends on the diameter of the network. In case, the diameter is changing with time, the reset is based on the ratio of the network expansion.

In Chapter 4, we have presented an optimized version of the previous one, in which the information carried by the messages is limited to topological changes. This modification enables every operational node to update its local topological information and to broadcast it more efficiently with respect to the changes. Consequently, the message size is adaptive and optimized comparing with the one presented in Chapter 3.

With these two stabilizing wave algorithms, we can provide a general paradigm to the design of other stabilizing protocols such as distributed reset, distributed snapshot, spanning tree construction, leader election, and mutual exclusion.

To show the generality of our proposed stabilizing algorithms, in Chapter 5, we have applied them to the problems of spanning tree construction and leader election in dynamic systems. By computing a leader and maintaining a spanning tree, the stabilizing wave algorithms serve as an underlying primitive to stabilize dynamic systems.

The objective of these two topology wave algorithms is to ensure that every operational processor in the network will have the correct and consistent topological information on the entire network within a finite time. Comparing with other related work, these two wave algorithms have the following advantages:

- (1) They are *event driven*, that is processors take actions in response to the detection of topological changes and the reception of messages. Therefore, they are practical and straightforward. Moreover, they are more general, in the sense the solutions for synchronous models can be easily obtained from the results of this thesis. In addition, we have shown a reset method to keep sequence numbers bounded. The proposed method can be applied to the models with different assumptions with respect to the nature of a dynamic network.
- (2) The information carried by the messages and the memory required by a processor are optimized in the second wave algorithm. It simplifies the process of the received topological information for every operational node.

6.2 Future Work

This thesis focuses on the problems the dynamic network, in which the system allows the addition and deletion for both links and processors. There are the following issues that need to be further studied.

Network Partition

In our model, when a processor is added, (i.e., the processor becomes operational), it is assumed that the operational processor is connected to an active processor. One of the open problems is how to regain and maintain the consistency in a dynamic network efficiently with respect to the network partition. The topological information on the entire network should be available to the operational processors of the two disconnected network as soon as they are bridged. One possible solution is to carry on the computation in the disconnected network components, respectively. Once they are connected, they should be able to merge and compile the topological information in an efficient way.

Reset Primitive and Bounded Counters

Since the sequence numbers are used in the wave algorithms, reset mechanism is an important issue that needs to be further studied. Recall that node addition is allowed in a dynamic distributed system. In the case, there is no *a priori* bound on the maximum number of nodes in the system, clearly, the reset of the counters depends on the ratio of the system expansion. If every node can obtain an estimate the ratio of network

expansion based on its own topological knowledge $\mathcal{K}$. Upon the estimate and the underlying time mechanism, a node can use a function $T = f(\varepsilon(G))$ to determine when its local wave counter should be reset, where $\varepsilon(G)$ is the estimated expansion rate with respect to the time. Notice that the dynamic distributed system could expand exponentially. With respect to these factors, we are investigating the mechanisms on how to reset the system and keep the counters bounded.

Spanning Tree Construction

Another issue is to achieve the objective of topology broadcast by using a dynamic spanning tree which is computed and maintained as an underlying mechanism, so that it can reduce the message complexity. Currently, the most efficient spanning tree construction protocol in a dynamic system is proposed by Awerbuch, Cidon and Kuten [ACK91], which is $O(N)$, where N is the number of nodes in the system.

These issues described above need more thought, and they are the areas of future research.

Appendix Visualization of Waves

Making Waves, Propagating Waves, and Eliminating Waves

Glossary

Ambiguity. With respect to a link in the network, the received information is inconsistent with the information known to the node.

Asynchronous System. The system has no global clock. The sender is not blocked, and the receiver may take an arbitrary long to receive the message after it has been sent.

Completion of the algorithm. A global system state in which no processor is executing any step and no messages are in transit.

Consistent set. At time t , a set of nodes in which every processor has correct and consistent information of the local topology of all processors in the set.

Dynamic System. The links and processors can become operational or crash in the system. That is, in the system, the communication links and processors can come up or go down at an arbitrary number of times; and they can stay in the same status for an arbitrary length of time.

Eliminating Waves. It is aimed to eliminate the “outdated” information and reduce the number of messages.

Finite Topological Changes. The sequence of topological changes is finite; that is the topological changes eventually cease.

Global Consistency ($GC_i(\mathcal{K})$). With respect to the status of any link in the network, $\mathcal{K}$ is consistent with $\mathcal{K}_i$; that is

$$GC_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigwedge \mathcal{K}[v,u] = \mathcal{K}_i[v,u]$$

Id-based Model. Every processor has a distinct identity.

Infinitely Continuous Topological Changes. The network is continuously changing so that the system does not self stabilize.

Infinitely Discrete Topological Changes. The topological changes occur in “burst” and the period in between one burst and the next one is long enough so that the system stabilizes.

Local Consistency ($LC_i(\mathcal{K})$). With respect to any local topology ($LocalTopology_i$), the topological knowledge $\mathcal{K}$ brought in by the received message is consistent with the topological knowledge $\mathcal{K}_i$ possessed by node i ; that is

$$LC_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigwedge \mathcal{K}[i,j] = \mathcal{K}_i[i,j]$$

Local Inconsistency ($LI_i(\mathcal{K})$). With respect to some local topology ($LocalTopology_i$), $\mathcal{K}$ is inconsistent with $\mathcal{K}_i$; that is

$$LI_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigvee \mathcal{K}[i,j] \neq \mathcal{K}_i[i,j]$$

$LocalTopology_i$. With respect to the local status of all the links incident to a node i ,

$LocalTopology_i = \{LT_i[j]\}$ is the local topology of i ,

$$\text{where } LT_i[j] = \begin{cases} 1 \text{ (up)} & j \in \text{Adjacent}(i) ; \\ 0 \text{ (down)} & j \notin \text{Adjacent}(i) \text{ and } j \in \text{Neighbor}_{\mathcal{K}_i}(i) ; \\ \text{Undefined} & \text{Otherwise .} \end{cases}$$

Making Waves . This wave rule determines how to initiate new waves in response to the topological changes in the network. The initiation of new waves is event driven.

Predecessor. The identity of the node from which node i has received a message and obtained the current topological information.

Propagating Waves. Upon receipt of the information messages, the processors update and broadcast the information by applying these rules.

RemoteAmbiguity_i . A set of the status of links which are ambiguous, that is $\{RA_i[v,u] \}$

$$\text{where } RA_i[v,u] = \begin{cases} 2 \text{ (ambiguous)} & \text{where } RI_i(\mathcal{K}) \text{ holds for } v \text{ and } u, \\ \text{Undefined} & \text{Otherwise.} \end{cases}$$

Remote Inconsistency ($RI_i(\mathcal{K})$). With respect to the status of some links which is not incident to i (i.e., the status of the link is not local topological information), $\mathcal{K}$ is inconsistent with $\mathcal{K}_i$; that is

$$RI_i(\mathcal{K}) \stackrel{\text{def}}{=} \bigvee_{v,u \neq i} \mathcal{K}[v,u] \neq \mathcal{K}_i[v,u]$$

Semi-uniform Model. Exactly one single processor is designated as a special processor and all the rest processors are identical

Static System. The links and processors that are operational remain operational during the computation period and no new links arise

Step. The interval, which is *from* the time when the activities of a processor is triggered by an event *to* the end of the execution of the corresponding actions, is an atomic and indivisible period of time for a processor.

Successors. A sub-set of neighbors of node i to which it sends information messages, and $\text{Adjacent}(i) \supseteq \text{succ}(i)$.

Synchronous System. There exists a global clock in the system. The message transmission and reception are synchronized. That is the message sender is blocked until the receiver is ready to receive the message. The message delay is bounded.

Uniform Model . There is no special processor and any all the processors are anonymous.

Bibliography

- [AAG87] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, IEEE, Oct. 1987.
- [AB89] Y. Afek and G. M. Brown. Self-Stabilization of the Alternating-Bit Protocol. In *Proceedings of the 8th Symposium on Reliable Distributed System*, pp. 80-83, 1989.
- [ABC86] S. Aggarwal, D. Barbara, and C. Courcoubetis. Coordinated Concurrent Activities. In *Proceedings of the 6th Workshop on Protocol Specification, Testing and Verification*, Gray Rocks, June, 1986.
- [ACK90] B. Awerbuch, I. Cidon and S. Kuten. Communication-Optimal Maintenance of Dynamic Trees. In *Proceedings of 5th International Workshop on Distributed Algorithms*, Delphi, Greece, Lecture Notes on Computer Science, 579, Springer-Verlag, Oct., 1991.
- [AE91] E. Anagnostou and R. El-Yaniv. More on the Power of Random Walks: Uniform Bounded Self-Stabilizing Protocols. In *Proceedings of 5th International Workshop on Distributed Algorithms*, pp 31-51, Delphi, Greece, Lecture Notes on Computer Science, 579, Springer-Verlag, Oct., 1991.

-
- [AEH92] E. Anagnostou, R. El-Yaniv, and V. Hadzilacos. Memory Adaptive Self-Stabilizing Protocols. In *Proceedings of the 6th International Workshop on Distributed Algorithms*, Haifa, Israel, Nov. 1992.
- [AG90] A. Arora and M. Gouda. Distributed Reset. In *Proceedings of the 10th FSTTCS*, pp316-331, Springer-Verlag (LNCS472), 1990.
- [AKY90] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self-stabilization on general networks. In *Proceedings of 4th International Workshop on Distributed Algorithms*, pp.15-28. Springer-Verlag, 1990.
- [APV91] B. Awerbuch, B. Patt-Shamir and G. Varghese. Self-Stabilization By Local Checking and Correction. In *Proceedings of Annual Symposium on Foundation of Computer Sciences*, IEEE, 1991.
- [AV91] B. Awerbuch, and G. Varghese. Distributed Program Checking: a Paradigm for Building Self-Stabilizing Distributed Protocols. In *Proceedings of Annual Symposium on Foundation of Computer Sciences*, IEEE, 1991.
- [Awe85] Baruch Awerbuch. Complexity of Network Synchronization. *Journal of the Association for Computing Machinery*, 32(4), pp.804-823, October, 1985.
- [BGW89] G. Brown, M. Gouda, and C. Wu. Token Systems that Self-Stabilize. *IEEE Transactions on Computers*, Vol 38, No. 6, June, 1989.

-
- [BP89] J.E. Burns and J. Pachl. Uniform Self Stabilizing Rings. *ACM Transactions on Programming Languages and Systems*, Vol. 11, No. 2, pp. 330-344, April, 1989.
- [Bur87] J.E. Burns. Self-Stabilizing Rings without Demons. Technical Report, GIT-ICS-87/36, Georgia Institute of Technology.
- [Chen91] N-S. Chen et al.. A self-stabilizing algorithm for constructing spanning trees. *Information Processing Letters*, pp147-151, Aug., 1991.
- [CGR87] E. Chang, G. Gonnet and D. Rotem. On the costs of self-stabilization. *Information Processing Letters* 24, pp. 311-316, Mar., 1987.
- [CL85] K. Chandy and L. Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems*, Vol. 3, No. 1, pp. 63-75, Feb., 1985.
- [Dij74] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, Vol. 17, pp. 643-644, 1974.
- [Dij82] E. Dijkstra. Self-stabilization in spite of distributed control. EWD 391, 1973, *Selected Writings on Computing: A Personal Perspective*. New York: Springer-Verlag, pp. 41-46. 1982.
- [DIM90] S. Dolev, A. Israeli, S. Moran. Self Stabilization of Dynamic Systems Assuming Only Read/Write Atomicity. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pp 103-117, Quebec City, Canada, 1990.

-
- [DIM91a] S. Dolev, A. Israeli, S. Moran. Uniform Dynamic Self-Stabilizing Leader Election. In *Proceedings of 5th International Workshop on Distributed Algorithms*, Delphi, Greece, Lecture Notes on Computer Science, 579, Springer-Verlag, Oct., 1991.
- [DIM91b] S. Dolev, A. Israeli, S. Moran. Resource Bounds for Self Stabilizing Message Driven Protocols. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, 1991.
- [Fin79] S. Finn. Resynch Procedure and a Fail-Safe Network Protocol. *IEEE Transactions on Communication*, Vol. Com -27, No.6, June, 1979.
- [FLP85] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Processor. In *Proceedings of 2nd ACM Symposium on Principles of Database Systems*, pp. 1-7. Atlanta, GA, March 1983. Revised version in *Journal of the Association for Computing Machinery*, 32(2): 374-382, April, 1985.
- [Gaf87] E. Gafni. Generalized Scheme for Topology Update in Dynamic Networks. In *Proceedings of 2nd International Workshop on Distributed Algorithms*, Amsterdam, The Netherlands, July, 1987.
- [Gou88] M. Gouda et al. . Delivery and Discrimination: The Senie Protocol. In *Proceedings of SIGCOMM' Symposium.*, pp. 292-302, 1988.
- [GHR90] M. Gouda, R. Howeli and L. Rosier. The Instability of Self stabilization. *Acta Informatica* 27, pp. 697-724, 1990.

-
- [GM91] M. Gouda and N. Multari. Stabilizing Communication Protocol. *IEEE Transactions on Computers*, Vol 40, No. 4, April 1991.
- [HMR88] J. Helary, A. Maddi and M. Raynal. Controlling Knowledge Transfers in Distributed Algorithms. Preprint
- [IJ90] A. Israeli and M. Jalfon. Token Management Schemes and Random Walks Yield Self Stabilizing Mutual Exclusion. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pp 119-131, Quebec City, Canada, 1990.
- [IJ91] A. Israeli and M. Jalfon. Self-Stabilizing Ring Orientation. In *Proceedings of 5th International Workshop on Distributed Algorithms*, Delphi, Greece, Lecture Notes on Computer Science, 579, Springer-Verlag, Oct., 1991.
- [KP90] S. Katz and K.J. Perry. Self-Stabilizing Extensions for Message-Passing Systems. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pp 91-101, Quebec City, Canada, 1990.
- [Kru79] H.S.M. Kruijer. Self-Stabilization (in spite of distributed control) in Tree-structured System. *Information Processing Letter* 8,2, pp. 91-95, 1979.
- [Lam84] L. Lamport. Solved Problems, Unsolved Problems, and Non-problems in Currency. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pp. 1-11, 1984.
- [Lam78] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communication of the A.C.M.*, 21 (7), pp558-565, 1978.

-
- [Multa89] N. Multari. Toward a Theory for Self-Stabilizing Protocols. Ph.D dissertation, Dept. of Computer Science., University of Texas at Austin, 1989.
- [Ric92] A.M. Ricciardi. The Group Membership Problem in Asynchronous Systems. Ph.D. dissertation, Dept. of Computer Science, Cornell University, Ithaca, New York, Nov., 1992.
- [SB84] B. Sarikaya and G. Bochmann. Synchronization and specification issues in protocol testing. *IEEE Transactions on Communication*, Vol. 32, No. 4, pp.389-395, 1984.
- [Seg81] A. Segall. Advances in Verifiable fail-safe routing procedures. *IEEE Transactions on Communication*. COM-29, pp 491-497.1981
- [SG89] J.M. Spinelli and R.G. Gallager. Event Driven Topology Broadcast Without Sequence Numbers. *IEEE Transactions on Communication*. Vol 37, No. 5, May 1989
- [SH86] S. Soloway and P. Humblet. On Distributed Network Protocols for Changing Topologies. M.I.T. Lab. Information Decision System. Technical Report, LIDS-P-1564, May. 1986.
- [SL86] F. Schneider and L. Lamport. Paradigms for Distributed Programs. *Distributed Systems*. Lecture Notes on Computer Science, 190, Springer-Verlag. 1986.

END

14-12-93

FIN