DISTRIBUTED SYNCHRONIZATION IN DELAYED AND TOPOLOGY VARYING NETWORKS

by

Onur Cihan

B.Sc., Electrical Engineering, Istanbul Technical University, 2007

B.Sc., Control Engineering, Istanbul Technical University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering

Boğaziçi University

2009

DISTRIBUTED SYNCHRONIZATION IN DELAYED AND TOPOLOGY VARYING NETWORKS

APPROVED BY:

Assoc. Prof. Mehmet Akar .....
(Thesis Supervisor)

Prof. Emin Anarım .....

Assoc. Prof. Fatih Alagöz .....

DATE OF APPROVAL: 28.08.2009

ACKNOWLEDGEMENTS

First and foremost, I would like to express my profound gratitude to my supervisor Dr. Mehmet Akar for his invaluable support, encouragement, supervision and useful suggestions throughout this thesis work.

I am as ever, especially indebted to my parents for their love and support throughout my life. I also wish to thank my brother, Ahmet Cihan, for his support during my study.

Finally, I thank TÜBİTAK for their financial support throughout my research.

ABSTRACT

DISTRIBUTED SYNCHRONIZATION IN DELAYED AND TOPOLOGY VARYING NETWORKS

There are numerous network applications where nodes require a common notion of time, and as such distributed synchronization is an important task in topology varying networks. In this thesis, we introduce a well known averaging based distributed synchronization algorithm and investigate its convergence conditions under varying topologies and delay.

When data are transmitted through the network, communication delays are unavoidable and it might degrade system performance. Furthermore, delay can cause a stable system go unstable unless certain conditions are met. In this thesis, the convergence of the consensus algorithm for delay varying networks is studied using properties of scrambling matrices. It is shown that delay does not affect the convergence of the algorithm so long as it is bounded. The effect of delay on convergence speed for some well known topologies is also discussed. In order to reduce convergence time, fastest converging system matrices are found for networks with symmetric connections by using Linear Matrix Inequalities. It is also shown that the fastest converging matrices for fixed topologies will not provide the fastest convergence for varying topology networks.

Consensus algorithms are investigated for networks not only with non–faulty nodes, but also with the faulty ones (or Byzantine nodes) that try to obstruct synchronization by sending wrong clock information to other nodes. It is shown that a network with Byzantine nodes will not be synchronized if the network topology and synchronization algorithm do not meet some conditions. Theoretical results are also illustrated by numerical examples.

ÖZET

GECİKMELİ VE İLİNGESİ DEĞİŞEN AĞLARDA DAĞITIK EŞ ZAMANLAMA

Birçok ağ uygulaması, düğümlerin ortak bir saat değerine sahip olmasını gerektirir ve ilingesi değişen ağlarda dağıtık eş zamanlama da dolayısıyla önemli bir iştir. Bu tezde, ortalama alma temelli bir dağıtık eş zamanlama algoritması tanıtılarak, yakınsama koşulları ilingesi değişen ve gecikmeli ağlar için incelenmiştir.

Veri ağ boyunca iletilirken, haberleşme gecikmeleri kaçınılmazdır ve bu da sistem performansını düşürebilir. Ayrıca belirli koşulların sağlanmaması halinde, gecikme kararlı bir sistemi kararsızlığa sürükleyebilir. Bu tezde çırpıcı matrislerin özelliklerini kullanarak, gecikmesi değişen ağlar için eş zamanlama algoritması incelenmiştir. Gecikmenin sınırlandırılmış olduğu durumda, eş zamanlama algoritmasının yakınsamasını etkilenmeyeceği gösterilmiştir. Bazı iyi bilinen ilingeler için gecikmenin yakınsama hızı üzerine etkisi de ayrıca tartışılmıştır. Yakınsama süresini azaltmak amacıyla, simetrik bağlantılı ağlar için en hızlı yakınsayan sistem matrisleri Doğrusal Matris Eşitsizlikleri kullanılarak bulunmuştur. En hızlı yakınsayan matrislerin, ilingesi değişen ağlarda en hızlı yakınsamayı sağlamadıkları da ayrıca gösterilmiştir.

Eş zamanlama algoritmaları yalnızca hatasız düğümler için değil, aynı zamanda diğer düğümlere hatalı saat bilgisi göndererek eş zamanlamayı engellemeye çalışan hatalı düğümler (Bizans düğümleri) için de incelenmiştir. Ağ ilingesi ve eş zamanlama algoritmasının bazı koşulları sağlamaması durumunda, Bizans düğümlerinin bulunduğu bir ağın eş zamanlanamayacağı gösterilmiştir. Teorik sonuçlar aynı zamanda sayısal örneklerle doğrulanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

A	System matrix
$\hat{A}$	Delayed system matrix
A^*	Optimal system matrix
A_D	The matrix that contains diagonal elements of A
A_{D^-}	The matrix that contains non–diagonal elements of A
C	Closed subspace of H
c_0	Projection of x on C
$D(a, R)$	Gershgorin circle centered at $x = a$ with radius R
d_0	The maximum offset between the components of $x(0)$
$E(t)$	Set of directed edges at time t
G	Adjacency matrix
H	Hilbert space
I	Identity matrix
k_{max}	The maximum number of steps to achieve desired level of synchronization accuracy
n	Number of nodes in a network
$N_i(t)$	Set of neighbors of particle i
T	Upper triangular matrix
U	Unitary matrix
V	Set of vertices
v	The eigenvector corresponding to the eigenvalue $\lambda_1 = 1$
$x(t)$	State vector
$\hat{x}(t)$	Delayed state vector
δ	Threshold value for the Algorithm CON
ϵ	Desired level of synchronization accuracy
λ	Eigenvalue of the corresponding matrix
λ_2	The second largest eigenvalue of a matrix
λ_2^*	The second largest eigenvalue of the optimal matrix

π_K	The set which contain all product of matrices of length k from a given set
σ	Set of eigenvalues
$\tau(A)$	The coefficient of ergodicity of A
τ_{ij}	Communication delay between node i and j
$\theta_i(t)$	Heading angle of node i at time t
ξ	Small positive constant

LMI	Linear Matrix Inequality
MSLEP	Minimum Second Largest Eigenvalue Problem
NIEP	Non–negative Inverse Eigenvalue Problem
RTT	Round–Trip Time
SDP	Semi Definite Programming

1. INTRODUCTION

Clock synchronization is the task of achieving a common notion of time for the nodes in the network. Many sensor network applications require the sensors share the same clock value. This is very important in networks in order to evaluate sensor data and predict future system behavior. There are lots of network applications where a global clock is needed which include [1]:

(i) Military applications: Detection of nuclear, biological, and chemical attacks and presence of hazardous materials; prevention of enemy attacks via alerts when enemy aircrafts are spotted; monitoring friendly forces, equipment and ammunition
(ii) Environmental applications: Forest fire monitoring, flood detection, and earthquake detection; monitoring ecological and biological habitats.
(iii) Civilian applications: Determining spot availability in a parking lot; active badge tracking at the workplace; surveillance for security in banks and shopping malls; highway traffic monitoring
(iv) Health applications: Tracking and monitoring doctors inside a hospital; identifying pre-defined symptoms by telemonitoring human physiological data
(v) Home and K–12 education: The intelligent home and smart kindergarten, where wireless networks are used in developmental, problem-solving environments
(vi) Scientific applications: Space and interplanetary exploration; deep undersea exploration, subatomic particle study, high-energy physics, study of cosmic radiation

In wireless networks, the operation that forms a meaningful result by combining all sensor data is called *data fusion*. For instance, in a forest fire monitoring application, different sensors detect fire at different points in time when the fire enters the range of each sensor. Sensor readings (direction or velocity) and timestamps are passed along so that fusion of such information from different sensors will add up to a global result. However, without a common notion of time, the direction of the fire cannot be accurately detected.

Without a synchronization algorithm, clocks can easily drift seconds per day, accumulating significant errors over time. Furthermore, because different clocks tick at different rates, they may not remain always synchronized although they are initially synchronized.

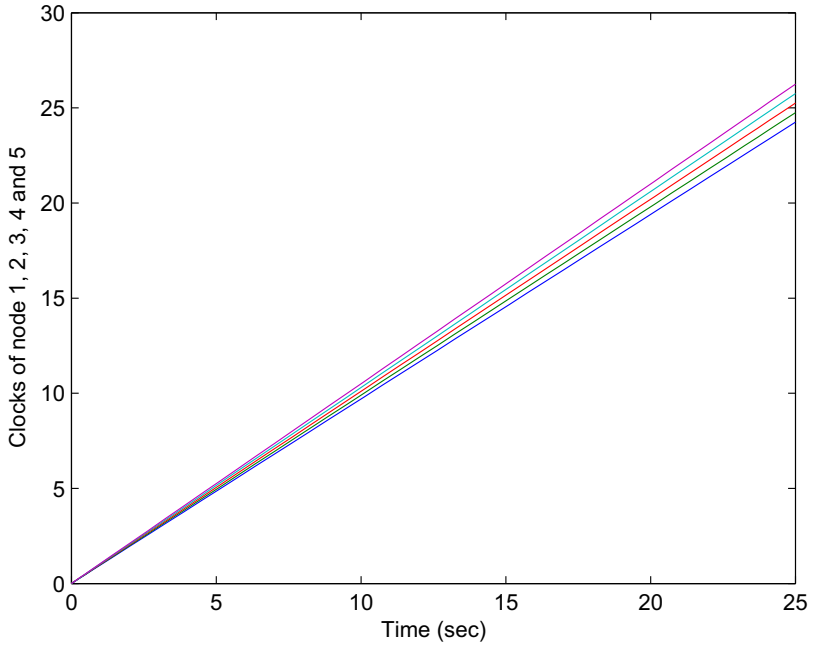

Figure 1.1. Node clock values under the lack of a synchronization algorithm

For example, in Fig 1.1, all clock values are reset at $t = 0$. However, different clock rates cause divergence in clock values. This divergence poses serious problems in applications that depend on a synchronized notion of time. Clock synchronization can be done centrally or distributively.

1.1. Centralized Synchronization

In centralized systems, there is no need for synchronized time because there is no time ambiguity. All processes set their clock to the master node's clock which is called *the global clock*. Cristian's algorithm and the Berkeley Algorithm are examples of solutions to the clock synchronization problem in centralized systems.

The basic idea underlying the Cristian's Algorithm is as follows [2]:

Assuming there is a time server which is connected to the Universal Time source, a node N in the network applies the following three steps:

(i) N requests the time from the time server.
(ii) After receiving the request, the server prepares a response time T.
(iii) The node N sets then its time to $T + RTT/2$.

Cristian's algorithm is a method for clock synchronization which can be used in many fields of distributive computer science, but the algorithm is probabilistic, in the sense that it only achieves synchronization if the round–trip time (RTT) of the request, which estimates the time elapsed for a message to be delivered to a remote place and come back again, is small compared to desired accuracy. It also suffers in implementations using a single server, making it unsuitable for distributive applications where redundancy may be significant.

In synchronization methods with a master node, the idea basically resembles the algorithm of Berkeley [3]. In this algorithm:

(i) A master is chosen via an election process, e.g. Chang and Roberts's algorithm [4], which is a useful method in decentralized distributed computing.
(ii) The master collects the time stamps coming from slaves.
(iii) The master observes the round–trip time of the messages and estimates the time of each slave and its own.
(iv) The master then averages the clock times. If any of the values are far from the average, they will be neglected.
(v) Instead of sending the updated current time back to the other nodes, the master then sends out the amount (positive or negative) that each slave must adjust its clock. This avoids further uncertainty due to round–trip time for the slave processes.

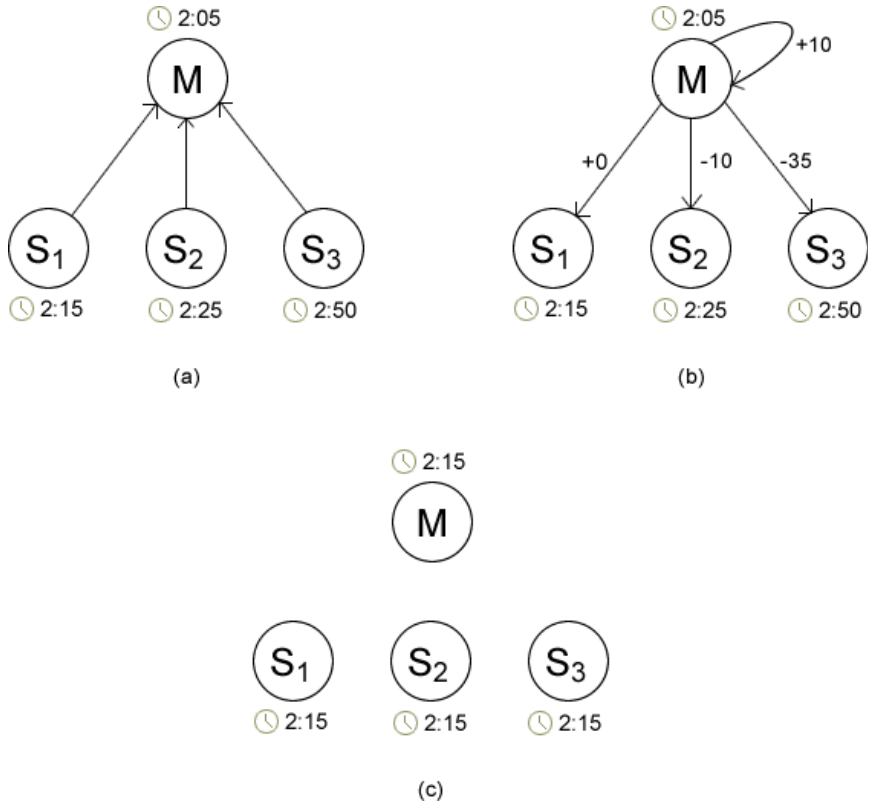

Figure 1.2. Berkeley algorithm example (a) The master node gathers clock information from the slave nodes (b) The master node sends clock differences to all nodes (c) Nodes update their values using the difference data

The application of the Berkeley algorithm in a four–node network is illustrated in Fig. 1.2. Once master node receives timestamps of three slave nodes, S_3 is considered to be faulty and therefore is not included in averaging. Master node responds all slave nodes by timestamp differences and all nodes including the master node set their clocks to the average value.

Throughout the thesis, we will concentrate on distributed synchronization.

1.2. Distributed Synchronization

In distributed systems, there is no global clock or common memory. Each processor has its own local clock and its own notion of time. In order to achieve synchronization, each node gets clock information of neighboring nodes and updates its clock

value by applying an algorithm using these values.

1.2.1. Distributed Synchronization

Viscek *et al.* suggested a distributed algorithm for self–driven particle systems [5]. It is assumed that all particles have the same speed but different headings. Heading information is shared by neighboring particles to provide a movement as a group. Each particle collects clock information from others which are within a close distance, r. Although Viscek's model is not linear, most of the convergence algorithms designed so far is the linearized version of Viscek's algorithm which is given by

$$\theta_i(t+1) = \frac{1}{1+|N_i(t)|}(\theta_i(t) + \sum_{j \in N_i(t)} \theta_j(t)) \tag{1.1}$$

where $\theta_i(t)$ is the heading angle of node i, $N_i(t)$ is the set of neighbors of particle i and $|N_i(t)|$ is the number of neighbors of node i at time t.

This algorithm can be used as a distributed clock synchronization algorithm as well. Instead of heading angles, clock information will be shared to achieve a common clock value in the network. Each node should use this algorithm to update its clock value by using the clock data of other nodes. However, a neighborhood is not necessary as long as there is a connection between nodes. Since Viscek's work, there has been a vast amount of research activity on the study of distributed consensus algorithms [6]-[20].

In 2003, a sufficient condition for convergence of the headings is given by Jadbabaie and his co–authors [6]. The proof is based on Wolfowitz's theorem on convergence of stochastic matrices products. It requires the existence of an infinite sequence of contiguous nonempty bounded time intervals across which all the agents are linked together.

Moreau analyzed the convergence of such multi–agent systems and his analysis

is based upon a blend of graph–theoretic and system–theoretic tools with the notion of convexity playing a central role [7].

Tsitsiklis *et al.* also studied the convergence of such systems, for networks with time–varying coefficients and time delays: if at time t an agent i uses the value of an agent j to update his value, it can use an outdated value with $\tau_{ij} < t$ instead of the most recent clock value [8].

Li and Wang suggested a weaker condition for the convergence of the system (1.1). They require the graph of the network to be connected in a limited number of steps if two vertices were connected to each other [9].

In this thesis, we examine the convergence of consensus algorithms in delay and topology varying networks [8], [10], [11], [12]. The algorithm we consider is the delay extended version of [14], [15]. It is proved in [8],[12] that the consensus algorithm converges so long as delay is bounded and a network connectivity condition is satisfied. In [12], Xiao and Wang proved the convergence using the concept of spanning trees discussed in [11]. We establish the same fact by showing that a finite power of the delay augmented system matrix is scrambling. On the other hand, neither [8] nor [12] considers the possible detrimental effect of delay on convergence speed. We investigate the effect of bounded delay on convergence speed and show by constructing certain topologies that delay does not always reduce convergence speed. Moreover, we also study the same convergence algorithm for networks with faulty nodes. Throughout the thesis, a distributed clock synchronization algorithm similar to (1.1) will be used.

1.3. The Organization of the Thesis

In Chapter 2, the consensus algorithm that is studied throughout the thesis is introduced, and the convergence conditions are determined. Since network mobility will cause topology to switch, the algorithm is analyzed for varying topology networks.

In Chapter 3, in addition to varying topology, we study the consensus under

bounded delay since it is very common in a wireless network. It is shown that a varying topology and delay network will be synchronized under the same assumptions as the convergence of fixed–topology and no–delay networks.

We investigate the relationship between the system matrix and the convergence speed in Chapter 4 and the effect of delay on convergence speed of the consensus algorithm. It is shown theoretically and by examples that the effect of delay on convergence speed is not always adverse. The results obtained by using the concepts we introduce in Chapter 3.

In Chapter 5, We suggest two different methods to provide a faster convergence; however these methods are not always applicable and even if it is applicable, this fast convergence does not work in topology varying networks.

We investigate the consensus algorithm for a network with faulty nodes as well. These faulty nodes cause instability in general; however, synchronization can be achieved under some assumptions that we discuss in Chapter 6. Finally, concluding remarks are given in Chapter 7.

2. AVERAGING BASED SYNCHRONIZATION

In the introduction, we discussed several algorithms that might be used for synchronization. Throughout the thesis, we will focus on averaging based consensus algorithms. In such algorithms, the nodes use not only their own values but also those of neighboring nodes in order to synchronize. Before discussing the algorithm in detail, some important information about graph representation of networks is introduced below.

2.1. Graph Representation of Networks

In communication networks, data flow is usually expressed by graph representation. In the following definitions, V denotes the set of vertices from 1 to n, $E(t)$ represents the set of directed edges at time t and $(j,i) \in E(t)$ holds if and only if there is communication between node j to node i at time t.

Definition 2.1.1. A graph is symmetric if and only if $(j,i) \in E(t)$ then $(i,j) \in E(t)$.

Definition 2.1.2. A graph is connected if there is communication, not necessarily a direct one, between any two nodes and if the graph is symmetric. Moreover, a graph is complete if it is connected and there is a direct communication between all vertices.

Definition 2.1.3. An adjacency matrix for (V,E) is defined as an $n \times n$ matrix $G = [g_{ij}]$, where $g_{ij} = 1$ if there is communication between node j to node i, and $g_{ij} = 0$ if there is no communication.

All elements of the adjacency matrices of complete graphs are one and symmetric graphs have symmetric adjacency matrices.

Example 2.1.1. Let the topology of a network be as given in Fig 2.1. This network

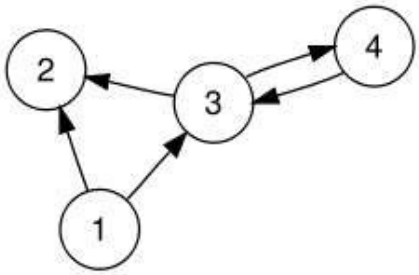

Figure 2.1. The graph representation of a four node network

has the adjacency matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{2.1}$$

2.2. The Consensus Algorithm

In discrete–time, the averaging based distributed consensus algorithm is expressed as

$$x_i(t+1) = \sum_{j=1}^{n} a_{ij}(t) x_j(t) \tag{2.2}$$

where $a_{ij}(t)$ is the non–negative weighting coefficient for the information coming from node j to node i, and $x_j(t)$ is the clock value of node j. Each node uses this equation to update its value in every step. For an n–node network, (2.2) can be represented in the matrix form

$$x(t+1) = A(t)x(t), A(t) \in \mathbb{A} \tag{2.3}$$

where $x(t) = [x_1(t), ..., x_n(t)]^T$ is the vector containing clock values of nodes and $A(t)$ is the system matrix composed of the weighting coefficients $a_{ij}(t)$. Throughout the thesis, it is assumed that the system matrices satisfy the following assumption.

Assumption 2.2.1. (i) There exists a positive constant ξ such that $a_{ii}(t) > \xi$, for all i and t.

(ii) $a_{ij}(t) \in \{0\} \cup [\xi, 1]$, for all i, j and t.

(iii) $\sum_{j=1}^{n} a_{ij}(t) = 1$, for all i and t.

Assumption 2.2.1(i) requires that each node should use its own value in its update. (ii) is related to network connectivity and ensures that we use data received from neighboring nodes with positive weights. The averaging coefficients sum up to one for each node as stated in (iii).

The main purpose of the algorithm is to converge to a common notion of time, or mathematically

$$lim_{t\to\infty} x(t) = [c, ..., c]^T \tag{2.4}$$

where c is the common clock for the nodes.

Definition 2.2.1. A matrix $A \in \Re^{n\times n}$ is said to be *row–stochastic* if the following conditions are satisfied:

(i) $a_{ij} \geq 0$, for all $1 \leq i \leq n$.

(ii) $\sum_{j=1}^{n} a_{ij} = 1$, for all $1 \leq i \leq n$.

This implies that connectivity of the nodes are non–negative and they sum up to one for each node update. Note that a row–stochastic matrix A satisfies $Ae = 1e$ where $e = [1, 1, \ldots, 1]^T$ is the eigenvector corresponding to the eigenvector $\lambda_1 = 1$.

Lemma 2.2.1. (Markov [21]) *Let x be a nonnegative vector and A a stochastic matrix. If $z = Ax$ then*

$$\max_i z_i - \min_i z_i \leq \tau(A)(\max_i x_i - \min_i x_i) \tag{2.5}$$

where

$$\tau(A) = \frac{1}{2} \max_{i,j} \sum_{k} |a_{ik} - a_{jk}|. \tag{2.6}$$

It is obvious from the definition that $\tau(A) \leq 1$ *for a row stochastic matrix* A*. This also implies that the minimal entry in* Ax *is non–decreasing over the minimal entry of* x*, and the maximal entry in* Ax *is non–increasing in the maximal entry of* x*.*

Definition 2.2.2. For a stochastic matrix A, the quantity $\tau(A)$ is called the *the coefficient of ergodicity of* A. If $\tau(A) < 1$, the matrix A is called *scrambling.*

Definition 2.2.3. A stochastic matrix A is called *ergodic* if $lim_{t\to\infty} A^t = ed^T$ for some $d \in \Re^n$ where $e = [1, 1, \ldots, 1]^T$.

A scrambling matrix is contractive in the difference of the entries in a vector. Therefore, a scrambling matrix is ergodic, but the opposite is not true in general [19].

Example 2.2.1. Let the system matrix of a four node network be

$$A = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 1/3 & 1/3 & 0 & 1/3 \end{bmatrix}. \tag{2.7}$$

The coefficient of ergodicity of the above matrix is one, so it is not scrambling. However it is ergodic, as we have

$$lim_{t\to\infty} A^t = \begin{bmatrix} 0.3158 & 0.3158 & 0.2105 & 0.1579 \\ 0.3158 & 0.3158 & 0.2105 & 0.1579 \\ 0.3158 & 0.3158 & 0.2105 & 0.1579 \\ 0.3158 & 0.3158 & 0.2105 & 0.1579 \end{bmatrix} = ed^T, \tag{2.8}$$

where $d = [0.3158, 0.3158, 0.2105, 0.1579]^T$.

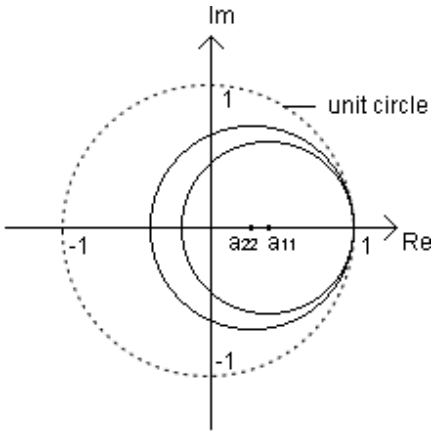

Figure 2.2. Gershgorin circles of the system matrices

2.3. Convergence of the Consensus Algorithm

Theorem 2.3.1. *(Gershgorin circle theorem [10]) Every eigenvalue of an $n \times n$ matrix A, lies within at least one of the Gershgorin circles defined by:*

$$D(a_{ii}, R_i) = \left\{ z \in C : |z - a_{ii}| \leq \sum_{j=1, j \neq i}^{n} |a_{ij}| \right\} \tag{2.9}$$

where a_{ii} is the $i-th$ diagonal element of A.

Applying Gershgorin circle theorem to the system matrix, we find that the eigenvalues are in the following region

$$\bigcup_{i=1}^{n} \{ z \in C : |z - a_{ii}| \leq 1 - a_{ii} \} . \tag{2.10}$$

Gershgorin circles are depicted in Fig 2.2. It is obvious from the figure that the union of these regions is the region surrounded by the circle with the largest radius. Since the radius is defined by $1 - a_{ii}$, the smaller diagonal element of the system matrix A defines the region where all eigenvalues are inside. Due to the Assumption 2.2.1(i), a_{ii} is positive and this region is a subset of the region surrounded by the unit circle. Since A is row–stochastic, we know that one of the eigenvalues of A is equal to one. In order to prove stability, we have to show that eigenvalue one has a multiplicity of

one. In a network, if there is a node which affects all others directly or indirectly, and the system matrix A satisfies Assumption 2.2.1, the eigenvalue one has a multiplicity of one. Then the convergence follows.

2.4. Convergence of the Algorithm for Varying Topology

In the previous section, we have shown that under the given assumptions the algorithm (2.2) converges for a fixed topology. However, topologies of wireless networks change as the communication between nodes are lost or new communication links are formed. In this section, we will examine such networks with switching topologies.

Theorem 2.4.1. ([19]) *Given $\mathbb{A} = \{A_1, ..., A_N\}$, if there is a node which affects all others directly or indirectly for each topology, then the system achieves consensus for varying topology.*

Proof. Since each topology satisfies Assumption 2.2.1, the k–th power of each system matrix where $k \leq n-1$ should be scrambling. Assuming $\mathbb{A}$ is a finite set, i.e., $\mathbb{A} = \{A_1, ..., A_n\}$, define the sets π_k, $k \geq 1$, which contain all products of matrices of length k from the set $\mathbb{A}$, e.g., $\pi_1 = \mathbb{A}$. Then there exists a $K \leq n-1$ where all matrices in π_K are scrambling. This implies that for each topology, the maximum difference between nodes will decrease in each step.

□

Example 2.4.1. Consider the following two averaging matrices for a four–node network

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/4 & 3/4 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 1/3 & 2/3 \end{bmatrix} \tag{2.11}$$

Table 2.1. The ergodicity coefficients for the matrices in the set π_3

Matrix	The coefficient of ergodicity
A_1^3	0.9583
A_2^3	0.5208
$A_1^2A_2$	0.8889
$A_2^2A_1$	0.6250
$A_2A_1^2$	0.5417
$A_1A_2^2$	0.7500
$A_1A_2A_1$	0.7500
$A_2A_1A_2$	0.6458

$$A_2 = \begin{bmatrix} 1/4 & 3/4 & 0 & 0 \\ 0 & 2/3 & 1/3 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}. \tag{2.12}$$

The system matrices A_1 and A_2 are not scrambling; however, the set π_3 can be constructed as follows:

$$\pi_3 = \{A_1^3, A_2^3, A_1^2A_2, A_2^2A_1, A_2A_1^2, A_1A_2^2, A_1A_2A_1, A_2A_1A_2\}.$$

It is straightforward to calculate the ergodicity coefficients using (2.6) for all matrices in π_3. As given in Table 2.1, they are all less than one which implies that they are scrambling and consequently ergodic. Therefore, synchronization is achieved. Starting from initial clock values $x_0 = [1, 0.9, 0.8, 0.7]^T$, in the case of system matrices switching between A_1 and A_2, the maximum difference between node values are depicted in Fig 2.3. It is a non–increasing function of time; however it is not monotonously decreasing since the system matrices are not scrambling.

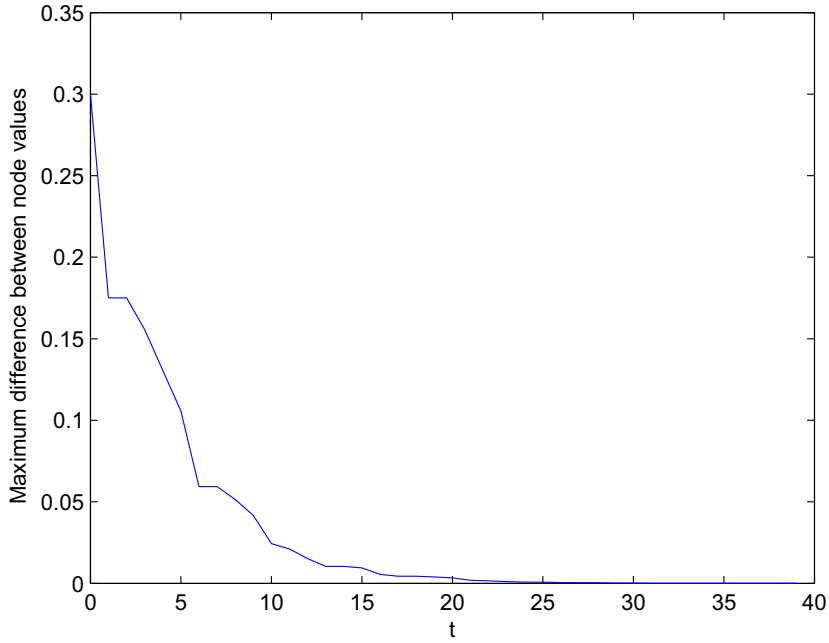

Figure 2.3. The maximum clock difference between nodes for a four node, varying topology network

2.4.1. Convergence Proof Using Graph Representation

It is proved above that the algorithm (2.2) is convergent for both fixed and topology varying networks. We have proved the convergence by using the properties of scrambling matrices so far. It is known that the selection of the averaging coefficients do not affect convergence as long as they satisfy Assumption 2.2.1. This means that the convergence is related to the network structure, not the actual values of the averaging coefficients. In this sub–section we will relate the convergence to the network structure properties (or, in mathematical terms, adjacency matrices).

Let $\mathbb{G} = \{G_1, ..., G_n\}$ be the set of adjacency matrices for the graphs associated with the matrices $A_1, ..., A_n$. Let $\mathbb{G}_k$ be the similar set for π_k. Under Assumption 2.2.1, following result can be stated [19]:

Lemma 2.4.1. *(i) If every $G \in \mathbb{G}$ is connected, then there exists an integer $K \geq 1$ for which every element of $\mathbb{G}_K$ is complete.*

(ii) Assume that every graph of $G \in \mathbb{G}$ has a node which affects all others directly or

indirectly. Then there exists an integer $K \geq 1$ for which every graph of $\mathbb{G}_K$ has a node (not necessarily the same) that reaches every other node in one step.

Proof. Suppose that every graph $G \in \mathbb{G}$ is connected. Consider an arbitrary product of the adjacency matrices $G = G_{i_1} G_{i_2} ... G_{i_K} = [g_{ij}]$ for an arbitrary index set $i_j \in \{1, 2, ..., N\}$. Then g_{ij} is given by

$$g_{ij} = \sum_{k_1=1}^{n} \sum_{k_2=1}^{n} ... \sum_{k_K=1}^{n} g_{i,k_1 k_2} ... g_{i,K_1 k_K j} \tag{2.13}$$

where $g_{k,lm}$ is the (l, m)–th component of the adjacency matrix G_k. Recall that $g_{k,lm}$ is non–zero if there is a path of length one from node m to l. Hence g_{ij} in (2.13) will be non–zero for all i, j and for $K = n - 1$, as the graphs are connected and every node is accessible from every other in at most $n-1$ steps. Thus, proof of part (i) is completed.

For the second part assume that every graph G of $\mathbb{G}$ has a node j_0 from which there exists a path to all others. From (2.13), g_{ij} will be non–zero for all i and $j = j_0$, i.e., G will have a positive column, which yields the desired result. □

Theorem 2.4.2. [19] *Given a set of matrices $\mathbb{A}$, let the associated set $\mathbb{G}$ satisfy one of the following:*

(i) Every $G \in \mathbb{G}$ is connected, or

(ii) For every graph G of $\mathbb{G}$, there is a vertex from which there is a path to all other vertices.

Then there exists a positive K where all matrices in π_K are scrambling and hence consensus is achieved.

Proof. Suppose that every graph in $\mathbb{G}$ is connected. Then by Lemma 2.4.1(i), there exists an integer $K \geq 1$ for which every element of $\mathbb{G}_K$ is complete, which implies that all the matrices in π_K are strictly positive (and also scrambling), and consensus is achieved.

Under hypothesis (ii), it follows from Lemma 2.4.1(ii) that all matrices in π_K are scrambling, hence the conclusion. □

2.5. Summary of the Chapter

In this chapter, we have introduced the distributed synchronization (consensus) algorithm that is the topic of this thesis. Mathematical preliminaries, including graph theoretic concepts, definitions of ergodicity, stochastic and scrambling matrices have been developed. Subsequently, the distributed algorithm in (2.2) is expressed as a switched system in (2.4) whose convergence properties are studied in Theorems 2.4.1 & 2.4.2 for fixed and topology varying networks. These results are mainly derived using the scrambling properties of row–stochastic matrices. Algorithm (2.2) is investigated in this chapter for topology varying case. In the next chapter, we extend these results to the delayed networks.

3. SYNCHRONIZATION IN DELAYED NETWORKS

Communication delay is unavoidable in wireless networks. It is known that delay may degrade system performance and may even induce instability. In this chapter, we introduce the delayed version of algorithm (2.2) and study its convergence properties.

3.1. Mathematical Model of Delayed Algorithm

In case of communication delay between nodes, the algorithm (2.2) can be modified as

$$x_i(t+1) = a_{ii}(t)x_i(t) + \sum_{j=1, j\neq i}^{n} a_{ij}(t)x_j(t-\tau_{ij}) \tag{3.1}$$

where τ_{ij} is the communication delay between node i and j . In order to examine the effect of delay on synchronization, in addition to Assumption 2.2.1, it is assumed that the delay is bounded.

3.2. Convergence of the Consensus Algorithm Under Delayed Information

When delay between nodes is fixed, i.e., $\tau_{ij} = \tau$ for all i, j, then the update algorithm (3.1) can be written in matrix form as

$$x(t+1) = A_D x(t) + A_{D^-} x(t-\tau) \tag{3.2}$$

where $A_D = diag(a_{ii})$, $A_{D^-} = A - A_D$ and τ is the uniform delay between the nodes. By defining the augmented state vector

$$\hat{x}(t) = [x(t+\tau)^T, x(t+\tau-1)^T, ..., x(t)^T]^T,$$

the system in (3.2) can be rewritten as:

$$\hat{x}(t+1) = \hat{A}(t)\hat{x}(t) \tag{3.3}$$

where

$$\hat{A} = \begin{bmatrix} A_D & 0 & \dots & 0 & A_{D^-} \\ I & 0 & \dots & 0 & 0 \\ 0 & I & \dots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & \dots & I & 0 \end{bmatrix} \tag{3.4}$$

is a square matrix of dimension $(\tau+1)n \times (\tau+1)n$ [20].

Lemma 3.2.1. *If there exists a node that affects all others directly or indirectly when $\tau = 0$, then (3.3) achieves consensus under bounded delay τ.*

Proof. Suppose that there exists a node which affects the others directly or indirectly for $\tau = 0$. From [19], we know that A^{n-1} is scrambling. We now show that $\hat{A}^{(\tau+1)n-1}$ is scrambling which will imply that synchronization is achieved. To this end, we first compute $\hat{A}^\tau = [\hat{A}^\tau_{ij}]$:

$$\hat{A}^\tau_{ij} = \begin{cases} f_{(i,1)}(D) & \text{if } j = 1 \text{ and } i \leq \tau \\ I & \text{if } j = 1 \text{ and } i = \tau + 1 \\ 0 & \text{if } j \leq i \text{ and } j \neq 1 \\ \bar{f}_{(i,j)}(A) & \text{if } j > i \end{cases} \tag{3.5}$$

where $f_{(i,j)}(A)$ has the same adjacency matrix as A; $\bar{f}_{(i,j)}(A)$ is defined similarly but with zero diagonal elements; and $f_{(i,j)}(D)$ is a diagonal matrix. Note that the individual numeric values of the components of each of these sub–matrices are not important in showing that $\hat{A}^{(\tau+1)n-1}$ is scrambling. Note also that the first row of $\hat{A}^\tau$ is composed

of matrices that are diagonal. By straightforward algebra, we can compute $\hat{A}^{2\tau+1} = [\hat{A}_{ij}^{2\tau+1}]$ as

$$\hat{A}_{ij}^{2\tau+1} = \begin{cases} g_{(i,1)}(A) & \text{if } j = 1 \\ \bar{g}_{(i,j)}(A) & \text{if } j \leq i \text{ and } j \neq 1 \\ \bar{g}_{(i,j)}(A^2) & \text{if } j > i \end{cases} \tag{3.6}$$

where the functions $g_{(i,j)}(\cdot)$ and $\bar{g}_{(i,j)}(\cdot)$ are defined in the same way $f_{(i,j)}(\cdot)$ and $\bar{f}_{(i,j)}(\cdot)$ are defined. From (3.6), we could conclude that $\hat{A}^{2\tau+1}$ is scrambling (hence synchronization is achieved) if A were itself scrambling. This is due to the fact that there are $\tau + 1$ blocks of $n \times n$ matrices in the same form of A in the first column of $\hat{A}^{2\tau+1}$.

In case, A is not scrambling, we compute $\hat{A}^{(\tau+1)n-1} = [\hat{A}_{ij}^{(\tau+1)n-1}]$ as

$$\hat{A}_{ij}^{(\tau+1)n-1} = \begin{cases} h_{(i,1)}(A^{n-1}) & \text{if } j = 1 \\ \bar{h}_{(i,j)}^{(n-1)}(A) & \text{if } j \leq i \text{ and } j \neq 1 \\ \bar{h}_{(i,j)}^{(n)}(A) & \text{if } j > i \end{cases} \tag{3.7}$$

where $h_{(i,j)}$ functions are defined similar to $f_{(i,j)}$ and $g_{(i,j)}$. Furthermore, $\bar{h}_{(i,j)}^{(k)}(A)$ is a $k - th$ order polynomial of $\bar{h}(A)$. The first column of (3.7) consists of $\tau + 1$ blocks of $n \times n$ matrices in the same form of A^{n-1}. From [19], when there is a node which affects all other nodes directly or indirectly when $\tau = 0$, A^{n-1} is a scrambling matrix. Therefore, from (3.7), $\hat{A}^{(\tau+1)n-1}$ is scrambling if and only if A^{n-1} is scrambling. Since this is guaranteed by assumption, synchronization is achieved under bounded delay. $\square$

3.2.1. Fixed Delay and Varying Topology Networks

In this sub–section, we will assume a network with a uniform fixed delay between nodes and analyze the effect of switching between a finite number of averaging matrices $\mathbb{A} = \{A_1, A_2, ..., A_N\}$. Let $\hat{\mathbb{A}} = \{\hat{A}_1, \hat{A}_2, ..., \hat{A}_N\}$ be the set of delayed system matrices associated with $\mathbb{A}$.

Lemma 3.2.2. *Given $\mathbb{A} = \{A_1, ..., A_N\}$, if there is a node which affects all others directly or indirectly for each topology when $\tau = 0$, then the delayed system achieves consensus for $\tau_{ij} = \tau$.*

Proof. If each $A_i \in \mathbb{A}$ satisfies Assumption 2.2.1 and if there is a node which affects all others directly or indirectly when $\tau = 0$, the corresponding $\hat{A}_i^{((\tau_i+1)n-1)}$ is scrambling. Define the sets π_k, $k \geq 1$, which contain all matrix products of length k obtained from the set $\hat{\mathbb{A}}$. For some positive constant $K \leq (\tau_i+1)n-1$, all matrices in π_K are scrambling [19]. Hence a varying topology and fixed delay network achieves consensus. □

3.2.2. Varying Delay and Fixed Topology Networks

In this sub–section, we examine the convergence properties of the consensus algorithm for networks of fixed topology but under randomly changing amounts of bounded delay.

Lemma 3.2.3. *Assume a fixed topology. If there is a node which affects all other nodes directly or indirectly when $\tau = 0$, then the delayed algorithm achieves consensus for arbitrary varying delay.*

Proof. Since delay is bounded by τ_{max}, we can define the state vector

$$\hat{x}(t) = [x(t + \tau_{max})^T, x(t + \tau_{max} - 1)^T, \ldots, x(t)^T]^T$$

and the system matrices $\hat{A}_{\tau_i}$ with the dimensions, $(\tau_{max}+1)n$:

$$\hat{A}_{\tau_i} = \begin{bmatrix} A_D & 0 & \dots & 0 & A_{D^-} & 0 & \dots & 0 \\ I & 0 & \dots & \dots & \dots & \dots & \dots & \vdots \\ 0 & I & 0 & \dots & \dots & \dots & \dots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & 0 & I & 0 \end{bmatrix} \tag{3.8}$$

where A_{D^-} is the $(\tau+1)-th$ sub–matrix of the first row of $\hat{A}_{\tau_i}$. Since $\hat{A}_{\tau_i}$ have the same dimensions, a positive constant $K \leq (\tau_{max}+1)n-1$ can be found such that all matrices in π_K are scrambling as given in the proof of Lemma 3.2.2. Hence consensus is achieved. □

3.2.3. Varying Delay and Varying Topology Networks

Lemmas 1-3 are combined in the following Theorem.

Theorem 3.2.1. *If there is a node which affects all other nodes directly or indirectly for each topology, then the system achieves consensus for arbitrary varying delay networks.*

Proof. Since delay is bounded by τ_{max}, we can construct such $\hat{A}$ matrices that they have same dimensions as in the varying delay fixed topology case. For varying topology case, we will construct the $\hat{A}$ matrices not only for all τ_i, but also for all $A_i \in \mathbb{A}$. The matrices $\hat{A}_{A_i,\tau_i}^{((\tau_i+1)n-1)}$ become scrambling for all τ_i and A_i. Given

$$\hat{\mathbb{A}} = \{\hat{A}_{A_1,\tau_{min}}, ..., \hat{A}_{A_1,\tau_{max}}, \hat{A}_{A_2,\tau_{min}}, ..., \hat{A}_{A_n,\tau_{max}}\},$$

define the sets π_k, $k \geq 1$, which contain all matrix products of length k from the set $\hat{\mathbb{A}}$. As in the proofs of Lemma 3.2.2-3.2.3, there exists some positive constant

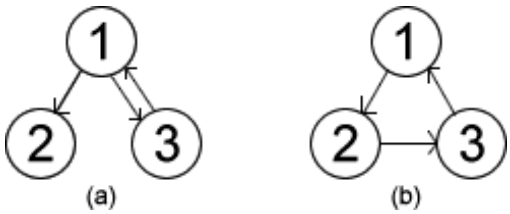

Figure 3.1. Two different topologies for a network of $n = 3$ nodes

$K \leq (\tau_{max} + 1)n - 1$ such that all matrices in π_K are scrambling. Hence a varying delay and varying topology network achieves consensus. □

Comment: The consensus conditions for the no–delay case are the necessary and sufficient conditions for the varying delay and varying topology case as well.

3.2.4. A Numerical Example

Now let us illustrate the convergence properties with a simple example. Consider two topologies of a three node network depicted in Fig. 3.1. Let the corresponding averaging matrices be

$$A_1 = \begin{bmatrix} 1/2 & 0 & 1/2 \\ 1/6 & 5/6 & 0 \\ 2/7 & 0 & 5/7 \end{bmatrix} \tag{3.9}$$

$$A_2 = \begin{bmatrix} 1/3 & 0 & 2/3 \\ 3/5 & 2/5 & 0 \\ 0 & 4/7 & 3/7 \end{bmatrix} \tag{3.10}$$

and the delay parameter switching between $\tau_{min} = 3$ and $\tau_{max} = 6$. As A_1 and A_2 both satisfy Assumption 2.2.1 and there is a node affecting the others in each case, there exists a K value such that π_K becomes scrambling. Hence from Theorem 3.2.1, it can be concluded consensus will be achieved. This is also confirmed in the simulations shown in Fig. 3.2.

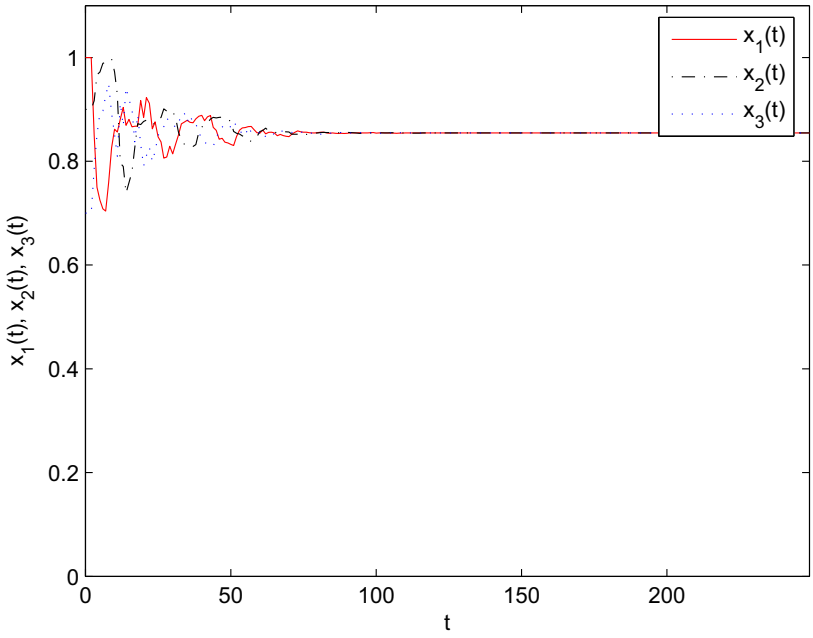

Figure 3.2. The node values for $\tau_{min} = 3 \leq \tau(t) \leq \tau_{max} = 6$ and varying topology network

3.3. Summary of the Chapter

In this chapter, we have investigated the distributed synchronization algorithm for delayed networks. We have studied the delayed version of (2.2) and discussed its convergence properties. A new state vector and augmented system matrix are introduced in the modified algorithm and this algorithm is expressed as a switched system as given in (3.3). We have proved that although the augmented system matrix $\hat{A}$ is not scrambling, a finite power of it is scrambling under Assumption 2.2.1. It is also shown that all delayed system matrices can be expressed as matrices with the same dimensions, since delay is bounded by τ_{max}. Convergence properties of delayed consensus algorithm are studied in Theorem 3.2.1 using the concept of scrambling matrices and matrix product sets. It has been shown both theoretically and by simulations that convergence is not affected by varying topology and varying delay so long as delay is bounded.

4. THE CONVERGENCE SPEED

In a network, the total time required to synchronize a network is called *convergence time*. A consensus algorithm that requires a large number of information exchanges for synchronization will result longer convergence time. Furthermore, in sensor networks, the speed of convergence is crucial since stored energy in sensors are very limited. Therefore faster convergence is always a desired performance metric for synchronization algorithms.

In Chapter 2, it has been shown that the convergence is not related to the averaging coefficients, but the network structure. On the other hand, the averaging coefficients play an important role in the speed of convergence of the convergence algorithm. For the same network, different averaging matrices may have different convergence speeds. In this chapter, we will derive the relationship between the structure of the system matrices and the convergence speed, so that with that information we are able to speed up the convergence.

4.1. Terms Related to Convergence Speed

Given a maximum offset, d_0, between the components of the initial state vector, x(0),

$$d_0 = \max_i x_i(0) - \min_i x_i(0), \tag{4.1}$$

it is important to determine the number of steps, k, required to achieve a desired level of synchronization accuracy ϵ. In this section, we will give detailed information about two terms related with the convergence speed of consensus algorithms.

4.1.1. The Coefficient of Ergodicity

From Chapter 2, we know that the coefficient of ergodicity is not greater than one for row–stochastic matrices. We also know that the difference between maximum and minimum value of a vector decreases when it is less than one.

Lemma 4.1.1. (Markov [21]) *Let $A_1, ..., A_n$ be row–stochastic matrices. Then the following inequalities are satisfied:*

$$\tau(A_1...A_n) \leq \tau(A_1)...\tau(A_n) \tag{4.2}$$

and

$$\tau(A^k) \leq \tau(A)^k \tag{4.3}$$

for any row–stochastic matrix A.

From (2.5) and (4.3), following equation can be written

$$k_{max} = log_{\tau(A)}(\epsilon/d_0). \tag{4.4}$$

Equation (4.4) provides a bound on the convergence speed and we can say that a matrix with a smaller coefficient of ergodicity will have a smaller k_{max} value; however this does not ensure faster convergence.

Example 4.1.1. Let two system matrices A_1 and A_2 given as

$$A_1 = \begin{bmatrix} 0.1653 & 0.3607 & 0.4740 \\ 0.2931 & 0.3439 & 0.3631 \\ 0.0341 & 0.6667 & 0.2992 \end{bmatrix} \tag{4.5}$$

$$A_2 = \begin{bmatrix} 0.3460 & 0.5157 & 0.1383 \\ 0.3098 & 0.3533 & 0.3369 \\ 0.0699 & 0.5020 & 0.4281 \end{bmatrix}. \tag{4.6}$$

In this example, the first system achieves consensus in less steps than the second system. It can be mathematically shown that while $\tau(A_1) = 0.3228$ is greater than $\tau(A_2) = 0.2898$, $\tau({A_1}^5) = 0.0004$ is less than $\tau({A_2}^5) = 0.0019$. This is because of the fact that (4.3) holds for each system. □

Since the coefficient of ergodicity is not a good means to compare convergence speeds as illustrated by the above example, we have to use alternative properties of the system matrices to make better comparisons.

4.1.2. The Second Largest Eigenvalue

In Section 2.3, it is proved that the largest eigenvalue of a system matrix is one. Moreover, all other eigenvalues are less than one (in magnitude). Let λ_2 be the second largest eigenvalue.

$$\lambda_1 = 1 > |\lambda_2| \geq ... \geq |\lambda_n| \tag{4.7}$$

Then the convergence speed is related to λ_2 and a smaller λ_2 provides faster convergence.

Example 4.1.2. Consider the system matrices given in Example 4.1.1. The second largest eigenvalues of the system matrices A_1 and A_2 are $\lambda_2(A_1) = 0.1921$ and $\lambda_2(A_2) = 0.2837$, respectively. Since $\lambda_2(A_1) < \lambda_2(A_2)$, the first system converges faster.

4.2. The Effect of Delay on Convergence Speed

In control systems, it is well known that delay may degrade system performance and even cause instability. Maybe somewhat surprisingly, it is seen from Theorem

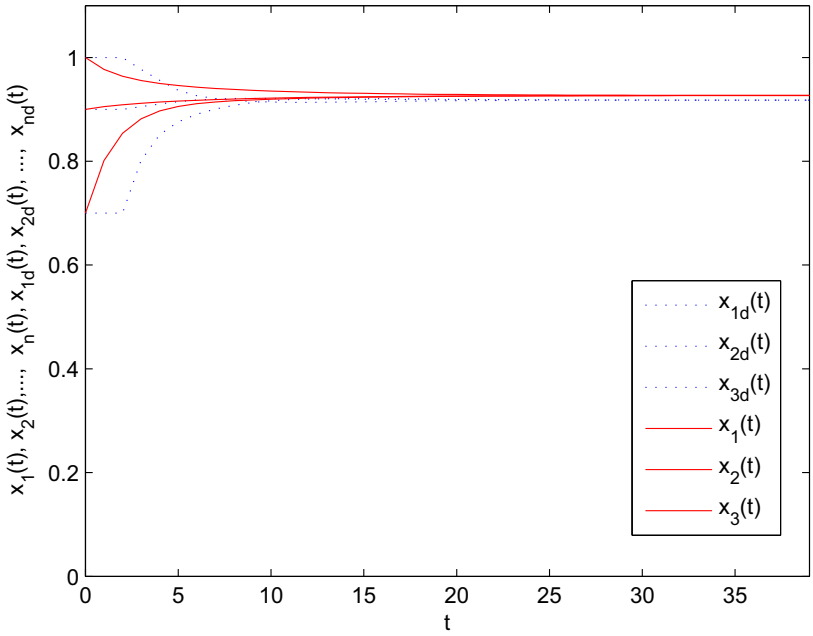

Figure 4.1. Simulation results for delay ($\tau = 2$) and no–delay networks using the system matrix in (4.8)

3.2.1 that delay does not have an adverse effect on the convergence of the distributed consensus algorithm so long as it is bounded. While this is the case, whether it has any effect on the convergence speed needs to be investigated. This is the subject of this section.

As a first step, let us consider a network of three nodes and the topology in Fig. 3.1(b) and the system matrix as follows

$$A = \begin{bmatrix} 0.924 & 0 & 0.076 \\ 0.052 & 0.948 & 0 \\ 0 & 0.507 & 0.493 \end{bmatrix}. \tag{4.8}$$

In our simple example, the second largest eigenvalue of A is $\lambda_2 = 0.8613$, while the second largest eigenvalue of the one–step delayed system has magnitude $|\bar{\lambda}_2| = 0.7722$. Surprisingly, this implies that the delayed system converges quicker, which is also confirmed by the simulation results depicted in Fig. 4.1.

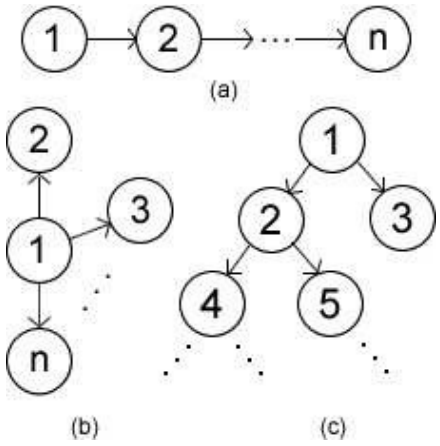

Figure 4.2. Some important topologies: a) $Chain$, b) $Master - Slave$, c) $Hierarchical$

The above example illustrates that delay may not always degrade the performance of the consensus algorithm. In the sequel, we investigate special topologies where the performance of the delayed and non–delayed algorithm is the same. Specifically, consider the topologies given in Fig. 4.2. In the first topology, each node receives previous node's clock information and updates its own clock. The next topology is known as *Master–Slave topology* where the first node shares its information with the rest. The third one is the *hierarchical topology*. In the hierarchical topology, each node receives the leading node's information in one or more number of steps. In all of these topologies, diagonal elements of the system matrix are also the eigenvalues of the matrix. The system matrices are lower triangular and eigenvalues are the diagonal elements. The largest eigenvalue of the matrix is $\lambda_1 = d_{11} = 1$ where d_{ii} is the $i - th$ diagonal element of the matrix. All of the other eigenvalues are positive and less than one.

Theorem 4.2.1. *In topologies where the system matrix is lower triangular or upper triangular, as in Fig. 4.2, delay does not decrease convergence speed of the algorithm (3.1).*

Proof. In the previous section, the system matrix of a system with delay was given in (3.4). In general, the eigenvalues of the delayed system matrix can be calculated as

the roots of the determinant

$$|\lambda^{\tau+1} I - \lambda^{\tau} A_D - A_{D^-}|$$

where τ is the uniform delay between nodes. It is straightforward to compute that the determinant is equal to $\lambda^{\tau n}(\lambda - \lambda_1)(\lambda - \lambda_2)...(\lambda - \lambda_n)$ where $\lambda_1 = 1 > |\lambda_2| \geq ... \geq |\lambda_n|$ are the eigenvalues of the same system without delay. The only difference between the eigenvalues of the delayed and the non–delayed systems is the existence of τn zero eigenvalues in the delayed case. Since this does not change the second largest eigenvalue λ_2, delay does not affect the convergence speed in such topologies. □

Theorem 4.2.1 delineates certain well known topologies where the performance of the consensus algorithm in terms of convergence speed is not adversely affected. Now let us consider a special topology where all nodes communicate with each other and use the averaging algorithm with equal weights $1/n$. In this case, all eigenvalues of the system matrix are equal to zero except for $\lambda_1 = 1$; hence consensus is achieved in one–step. The second largest eigenvalue of the delayed system is nonzero and therefore it cannot converge in one step.

4.3. Summary of the Chapter

In this chapter, we have investigated the relationship between the structure of the system matrices and the convergence speed. Two different structural properties, the coefficient of ergodicity and the second largest eigenvalue are discussed. We have found that the coefficient of ergodicity provides an upper bound, k_{max}, for the number of steps to achieve synchronization; however, the actual number of steps is far less than this upper bound in general. We have also studied the effect of the second largest eigenvalue of the system matrix A to the convergence speed, and concluded that a smaller second largest eigenvalue (in magnitude) will result faster convergence.

Subsequently, we have examined the effect of delay on the convergence speed. It

is proved that delay does not decrease convergence time for some certain topologies; moreover, a properly chosen system matrix may provide faster convergence in the delayed case.

In conclusion, it is important for a system matrix to have a smaller second largest eigenvalue for faster convergence. The derivation of system matrices that will lead to optimized performance will be analyzed in the next chapter.

5. PROPER SELECTION OF SYSTEM MATRICES

From Chapter 4, we know that the convergence speed is related with the second largest eigenvalue of the system matrix. To speed up consensus, the weighting coefficients have to be chosen properly so that the second largest eigenvalue should be as small as possible. In this chapter, we will concentrate on finding system matrices to provide faster convergence.

5.1. Non–negative Inverse Eigenvalue Problem

Definition 5.1.1. A real $n \times n$ matrix is said to be *non–negative* if each of its entries is non–negative.

Since the system matrices considered in this thesis satisfy Assumption 2.2.1(ii), they are non–negative matrices.

Definition 5.1.2. Given a list of n complex numbers, $\sigma = \{\lambda_1, ..., \lambda_n\}$, the problem of finding a non–negative $n \times n$ matrix with eigenvalues σ is called The *Non–negative Inverse Eigenvalue Problem*. If such a matrix A exists, we say that A realizes σ.

Finding necessary and sufficient conditions for a list of given eigenvalues to be realizable as the eigenvalues of a non–negative matrix has been a challenging area of research for more than sixty years and this problem is not solved yet [22]. As stated in [23], although many necessary and sufficient conditions exist, necessary conditions are too general while sufficient conditions are too specific. Let us give some definitions that the algorithm is based on.

Definition 5.1.3. Let H be a Hilbert space and x be an element in H. Let C be a closed subset of H. Any $c_0 \in C$ such that $\|x - c_0\| \leq \|x - c\|$ for all $c \in C$ is called a *projection* of x onto C.

Definition 5.1.4. A *unitary matrix* is an $n \times n$ complex matrix satisfying the condition

$U^*U = UU^* = I_n$ where I_n is the identity matrix with the dimension n and U^* is the conjugate transpose of U.

Definition 5.1.5. (Schur Decomposition). Given $A \in \mathbb{C}^{n\times n}$ with eigenvalues $\sigma = \{\lambda_1, ..., \lambda_n\}$ in any prescribed order, there exists a unitary matrix $U \in \mathbb{C}^{n\times n}$ and an upper triangular matrix $T \in \mathbb{C}^{n\times n}$ such that

$$A = UTU^* \tag{5.1}$$

and $T_{ii} = \lambda_i$, $i = 1, ..., n$.

Now we define two sets that will help us to construct the non–negative inverse eigenvalue problem (NIEP):
Let $M = \{A \in \mathbb{C}^{n\times n} | A = UTU^*$ for some unitary U and some $T \in \mathbb{T}\}$ and $N = \{A \in \Re^{n\times n} | A_{ij} \geq 0$ for all $i, j\}$. The purpose of our NIEP algorithm is to find a matrix that exists in both sets:

$$X \in M \cap N. \tag{5.2}$$

Definition 5.1.6. Suppose $U \in \mathbb{C}^{n\times n}$ is unitary and $T \in \mathbb{C}^{n\times n}$ is upper triangular. Let $\{\hat{\lambda}_1, ..., \hat{\lambda}_n\}$ be a permutation of the list of eigenvalues σ such that, among all possible permutations, it minimizes

$$\sum_{i=1}^{n} |\hat{\lambda}_i - T_{ii}|^2. \tag{5.3}$$

Define

$$\hat{T}_{ij} = \begin{cases} \hat{\lambda}_i & \text{if } i = j \\ T_{ij} & \text{otherwise} \end{cases} \tag{5.4}$$

The NIEP Algorithm:

(1) Choose a non–negative matrix $Y \in \Re^{n\times n}$ with random elements,

(2) Calculate Schur decomposition of Y: $Y = UTU^*$.

(3) Define $\hat{T} = [\hat{T}_{ij}]$

(4) $X = U\hat{T}U^*$

(5) Update $X = [x_{ij}]$ such that

$$x_{ij} = \begin{cases} \xi & \text{if } G_{ij} = 1 \text{ and } x_{ij} < \xi \\ 0 & \text{if } G_{ij} = 0 \\ x_{ij} & \text{otherwise} \end{cases} \tag{5.5}$$

where ξ is a small positive constant.

(6) If $\|X - Y\| = (\sum_{i,j} |X_{ij} - Y_{ij}|^2)^{1/2} > \epsilon$, set $Y = X$ and go to step 2.

Finding necessary and sufficient conditions for finding a matrix X with desired eigenvalues is still unsolved, thus convergence of the NIEP algorithm is not guaranteed.

5.1.1. Non–Negative Inverse Eigenvalue Problem for Row–Stochastic Matrices

So far, we have discussed finding non–negative matrices with the lack of stochasticity condition so far. In this section, we are interested in developing a solution for row–stochastic and non–negative matrices. If a non–negative matrix with a given list of eigenvalues can be constructed, we can convert the matrix into a row–stochastic matrix.

Lemma 5.1.1. [22] *Let Y be a non–negative matrix with spectrum σ and v the eigenvector corresponding to $\lambda_1 = 1$. If elements of v are all positive/negative, then $A = D^{-1}YD$ is a row–stochastic and non–negative matrix with the same spectrum of Y where D is a diagonal matrix whose diagonal elements are composed of the elements of v.*

Proof. If $v = [v_1, ..., v_n]^T$ is the eigenvector of Y corresponding to the eigenvalue $\lambda_1 = 1$,

then it satisfies

$$Yv = 1v \tag{5.6}$$

which can be represented by n equations as follows

$$\begin{aligned} y_{11}v_1 + y_{12}v_2 + ... + y_{1n}v_n &= v_1 \\ y_{21}v_1 + y_{22}v_2 + ... + y_{2n}v_n &= v_2 \\ &\vdots \\ y_{n1}v_1 + y_{n2}v_2 + ... + y_{nn}v_n &= v_n. \end{aligned} \tag{5.7}$$

By constructing D matrix, we can derive the $D^{-1}YD$ matrix as follows:

$$D^{-1}YD = \begin{bmatrix} \frac{1}{v_1} & 0 & \dots & 0 \\ 0 & \frac{1}{v_2} & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & \frac{1}{v_n} \end{bmatrix} Y \begin{bmatrix} v_1 & 0 & \dots & 0 \\ 0 & v_2 & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & v_n \end{bmatrix} \tag{5.8}$$

or

$$D^{-1}YD = \begin{bmatrix} \frac{v_1}{v_1}y_{11} & \frac{v_2}{v_1}y_{12} & \dots & \frac{v_n}{v_1}y_{1n} \\ \frac{v_1}{v_2}y_{21} & \frac{v_2}{v_2}y_{22} & \dots & \frac{v_n}{v_2}y_{2n} \\ & & \vdots & \\ \frac{v_1}{v_n}y_{n1} & \frac{v_2}{v_n}y_{n2} & \dots & \frac{v_n}{v_n}y_{nn} \end{bmatrix}. \tag{5.9}$$

Using (5.7), it is easy to see that the sum of each row of (5.9) is equal to one. □

Example 5.1.1. Let the list of desired eigenvalues be $\sigma = \{1, 0.2, 0.1, 0\}$ and adjacency matrix for the network be

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}. \tag{5.10}$$

Step 1: Setting algorithm precision $\epsilon = 10^{-14}$, the algorithm returns the following Y matrix

$$Y = \begin{bmatrix} 0.2543 & 0.2467 & 0 & 0 \\ 0 & 0.4975 & 0.2684 & 0.1827 \\ 0.0999 & 0.7295 & 0.5338 & 0.3325 \\ 0 & 0.1671 & 0 & 0.0144 \end{bmatrix}. \tag{5.11}$$

Eigenvalues of Y are exactly what is expected; however, it is not row–stochastic.
Step 2: The eigenvector corresponding to $\lambda_1 = 1$ is

$$v = [-0.1610, -0.4865, -0.8547, -0.0825]^T,$$

the digitalization matrix D is

$$D = \begin{bmatrix} -0.1610 & 0 & 0 & 0 \\ 0 & -0.4865 & 0 & 0 \\ 0 & 0 & -0.8547 & 0 \\ 0 & 0 & 0 & -0.0825 \end{bmatrix}. \tag{5.12}$$

Finally, row–stochastic matrix A can be calculated as

$$A = D^{-1}YD = \begin{bmatrix} 0.2543 & 0.7457 & 0 & 0 \\ 0 & 0.4975 & 0.4715 & 0.0310 \\ 0.0188 & 0.4153 & 0.5338 & 0.0321 \\ 0 & 0.9856 & 0 & 0.0144 \end{bmatrix}. \tag{5.13}$$

A is a row–stochastic matrix with desired eigenvalues and adjacency matrix.

5.2. Minimum Second Largest Eigenvalue Problem (MSLEP)

Assigning eigenvalues of non–negative matrices is an applicable method for system matrices; however, it has some serious drawbacks. First of all, it is not always possible

to assign eigenvalues. The problem of determining necessary and sufficient conditions for assigning desired eigenvalues is still unsolved. Moreover, the important (in terms of convergence speed) eigenvalue of system matrices is the second largest one, so the remaining eigenvalues are not important as long as they are less than λ_2.

In this section, we will study minimization problem of the second largest eigenvalues of system matrices (MSLEP).

5.2.1. MSLEP for Symmetric System Matrices

Definition 5.2.1. Let A be an $n \times n$ matrix. If A satisfies

$$A \geq 0, \quad Ae = e, \quad A = A^T, \tag{5.14}$$

where $A \geq 0$ means it is non–negative, i.e., $A = [a_{ij}]$ where $a_{ij} \geq 0$ for all i and j, and $e = [1, ..., 1]^T$. Then A is called a *doubly stochastic matrix*.

Not only the row–sum, but also the column sum of doubly stochastic matrices are equal to one. Note that doubly stochastic matrices are only suitable to topologies where the connections between nodes are bidirectional. In this sub–section, we consider the following problem: find the A matrix that gives the minimum second largest eigenvalue λ_2. This can be posed as the following optimization problem:

$$\begin{aligned} &\text{minimize} && \lambda_2 \\ &\text{subject to} && A \geq 0, \quad Ae = e, \quad A = A^T \\ &&& a_{ij} = 0 \text{ if } g_{ij} = 0. \end{aligned} \tag{5.15}$$

Here, $g_{ij} = 0$ if there is no communication between node i and node j. This problem is called *The Minimum Second Largest Eigenvalue Problem for Symmetric System Matrices*. We will introduce two different methods to solve this problem.

5.2.1.1. The Maximum–Degree Chain. There are several simple methods to obtain system matrix that is hoped to give fast convergence, if not the fastest possible.

The maximum–degree chain method is a well–known heuristic method and is applicable for Markov Chains [24]. Let d_i be the number of nodes that communicates with node i. Let $d_{max} = \max_i d_i$. The maximum–degree chain is obtained by assigning $1/d_{max}$ to every edge except the self–loops, and choosing the diagonal elements to ensure that the sum of each row is equal to one. The maximum–degree matrix is given by

$$A^{md} = \begin{cases} 1/d_{max} & \text{if } i \neq j \text{ and } g_{ij} = 1 \\ 1 - d_i/d_{max} & \text{if } i = j \\ 0 & \text{if } g_{ij} = 0 \end{cases} \tag{5.16}$$

This method is also applicable to the systems we consider in the thesis. However, Assumption 2.2.1(i) is violated for one or more nodes.

5.2.1.2. Convex and SDP Formulation of MSLEP. We first show that the minimum second largest eigenvalue is a convex function of A, on the set of doubly stochastic matrices. There are several ways to establish this. The first proof uses the variational characterization of eigenvalues. Since $\lambda_1 = 1$ with associated eigenvector $e = [1, ..., 1]^T$, we can express the second largest eigenvalue as

$$\lambda_2(A) = \sup\{u^T A u \quad | \quad \|u\|_2 \leq 1, \quad e^T u = 0\}. \tag{5.17}$$

This shows that $\lambda_2(A)$ is the pointwise supremum of a family of linear functions of A (i.e., $u^T A u$), and so is a convex function of A [24].

We can also derive convexity of λ_2 from known results for eigenvalues of symmetric matrices. The sum of any number of the largest eigenvalues of a symmetric matrix is a convex function of the matrix. In particular, the function $\lambda_1 + \lambda_2$ is convex for general symmetric matrices. Since our matrices are doubly stochastic, we have $\lambda_1 = 1$, so we

conclude that $\lambda_2 = (\lambda_1 + \lambda_2) - 1$ is a convex function.

We can also show convexity of λ_2 by expressing it as the spectral norm of A restricted to the subspace e^T:

$$\lambda_2(A) = \|(I - (1/n)ee^T)A(I - (1/n)ee^T)\|_2 = \|A - (1/n)ee^T\|_2. \tag{5.18}$$

Here, the matrix $I - (1/n)ee^T$ gives orthogonal projection on $e^\perp$, and $\|.\|_2$ denotes the spectral norm, or maximum singular value. (In this case, since the matrices are symmetric, $\|.\|_2$ is the largest magnitude eigenvalue). Equation (5.18) gives λ_2 as the norm of an affine function of A, and so is a convex function.

MSLEP for symmetric matrices can be cast as a convex optimization problem or a semi–definite programming problem.

The MSLEP is evidently a convex optimization problem, since the constraints are all linear equalities or inequalities, and the objective function is convex. Using (5.18), we can formulate the MSLEP as

$$\begin{array}{ll} \text{minimize} & \lambda_2(A) = \|A - (1/n)ee^T\|_2 \\ \text{subject to} & A \geq 0, \quad Ae = e, \quad A = A^T \\ & a_{ij} = 0 \text{ if } g_{ij} = 0, \end{array} \tag{5.19}$$

i.e., a norm minimization problem over a set of symmetric stochastic matrices.

We can express (5.19) as an SDP by introducing a scalar variable s to bound the norm of $A - (1/n)ee^T$:

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to} & -sI \preceq A - (1/n)ee^T \preceq sI \\ & A \geq 0, \quad Ae = e, \quad A = A^T \\ & a_{ij} = 0 \text{ if } g_{ij} = 0. \end{array} \tag{5.20}$$

Here the variables are the matrix A and the scalar s. The symbol $\preceq$ denotes matrix inequality, i.e., $X \preceq Y$ means $Y - X$ is positive semidefinite. (The symbol $\leq$ is used to denote elementwise inequality).

The problem (5.20) is not in one of the standard forms for SDP, but is readily transformed to a standard form, in which a linear function is minimized, subject to a linear matrix inequality (the constraint that an affine symmetric–matrix valued function be positive semidefinite), and linear equality constraints. The inequalities in (5.20) can be expressed as a single linear matrix inequality (LMI),

$$diag(A - (1/n)ee^T + sI, sI - A + (1/n)ee^T, vec(A)) \succeq 0. \tag{5.21}$$

Here, diag(.) forms a block diagonal matrix from its arguments, and vec(A) is a vector containing the $n(n+1)/2$ different coefficients in A. Since a block diagonal matrix is positive semidefinite if and only if its blocks are positive semidefinite, this single LMI is equivalent to the inequalities in (5.20).

The SDP formulation (5.20) has several important ramifications. First of all, it means that the MSLEP problem can be solved efficiently (and globally) using standard SDP solvers, at least for small or medium size problems (with number of nodes up to a thousand or so). A custom designed SDP solver for the MSLEP, which exploits the special structure of the problem (e.g., sparsity of A) would be able to solve even larger problems.

Example 5.2.1. We first consider the four simple topologies with given adjacency matrices in Table 5.1 For each topology, Table 5.2 shows the second largest eigenvalues for maximum–degree chain and the optimal A matrices. Note that the optimal A matrix need not be unique, i.e., the optimal system matrix for second topology can

Table 5.1. Adjacency matrices for four symmetric topologies

Topology	Adjacency Matrix	Topology	Adjacency Matrix
1	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$	2	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$
3	$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$	4	$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$

also be obtained as

$$\begin{bmatrix} 6/11 & 5/11 & 0 & 0 \\ 5/11 & 0 & 3/11 & 3/11 \\ 0 & 3/11 & 3/11 & 5/11 \\ 0 & 3/11 & 5/11 & 3/11 \end{bmatrix}.$$

5.2.1.3. Symmetric MSLEP for Topology Varying Case. In the previous section, we study the optimal system matrix for a fixed topology network. The optimal system matrix converges faster than the maximum degree chain matrix for a given topology; however, it is not always true for varying topologies. Let us illustrate this fact with a simple example.

Example 5.2.2. Consider the two topologies depicted in Fig. 5.1. Let the topology switch respectively between these two. The optimal and the maximum degree chain matrices are given in Table 5.3- 5.4 for each of these topologies. Since topology is switching respectively, this topology varying network system can be thought as a fixed topology network with the system matrices $A^*_{12} = A^*_1 A^*_2$ and $A^{md}_{12} = A^{md}_1 A^{md}_2$. The second largest eigenvalue of A^* is $|\lambda^*_2| = 0.4776$ which is greater than the second

Table 5.2. The second largest eigenvalue of maximum–degree chain (λ_2^{md}) and optimal A matrix λ_2^*

Topology	λ_2^{md}	λ_2^*	Optimal A matrix
1	$\sqrt{2}/2$	$\sqrt{2}/2$	$\begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \end{bmatrix}$
2	2/3	7/11	$\begin{bmatrix} 6/11 & 5/11 & 0 & 0 \\ 5/11 & 0 & 3/11 & 3/11 \\ 0 & 3/11 & 4/11 & 4/11 \\ 0 & 3/11 & 4/11 & 4/11 \end{bmatrix}$
3	2/3	3/7	$\begin{bmatrix} 1/7 & 2/7 & 0 & 2/7 & 2/7 \\ 2/7 & 3/7 & 2/7 & 0 & 0 \\ 0 & 2/7 & 1/7 & 2/7 & 2/7 \\ 2/7 & 0 & 2/7 & 3/7 & 0 \\ 2/7 & 0 & 2/7 & 0 & 3/7 \end{bmatrix}$
4	1/4	1/4	$\begin{bmatrix} 1/4 & 1/4 & 1/4 & 0 & 1/4 \\ 1/4 & 1/4 & 0 & 1/4 & 1/4 \\ 1/4 & 0 & 1/4 & 1/4 & 1/4 \\ 0 & 1/4 & 1/4 & 1/4 & 1/4 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 \end{bmatrix}$

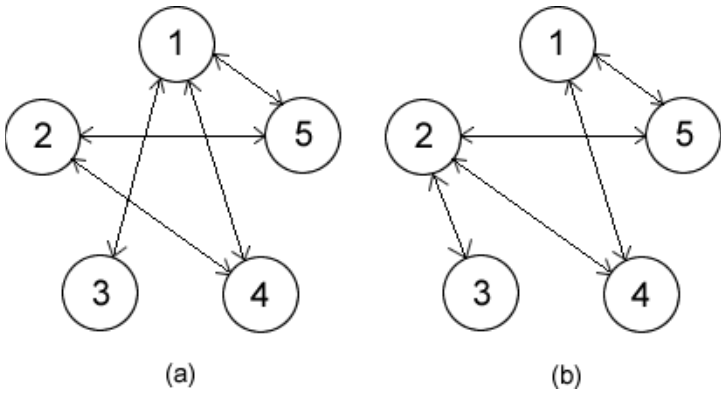

Figure 5.1. Two topologies of a five node–network where first and second node are swapped

largest eigenvalue of A^{md}, $|\lambda_2^{md}| = 0.2909$.

This means that the systems using heuristic matrices may even converge faster than the systems using the optimal matrices. This is because of the fact that eigenvalues of the multiplication of two matrices is not related to the eigenvalues of these two matrices. Therefore, optimal matrices work better in fixed topology networks.

5.2.2. MSLEP for Ordinary System Matrices

The connections between nodes need not be bidirectional in general, hence the optimal system matrices may not be symmetric. For instance, the optimal non–symmetric system matrix for the topology given in Fig. 5.1(a) is

$$A^* = \begin{bmatrix} 0.0814 & 0 & 0 & 0.0077 & 0.9109 \\ 0 & 0.5172 & 0.0491 & 0.0093 & 0.4244 \\ 0 & 0.8860 & 0.1140 & 0 & 0 \\ 0.3823 & 0.4389 & 0 & 0.1788 & 0 \\ 0.0165 & 0.5376 & 0 & 0 & 0.4459 \end{bmatrix}. \tag{5.22}$$

The second largest eigenvalue of A^* is 0.21371, while the second largest eigenvalue of the system matrix of the symmetric system is 0.7012. It is clear that second largest eigenvalue of a non–symmetric optimal matrix is smaller than a symmetric–optimal

Table 5.3. Optimal system matrices for a varying topology network

Topology 1	$A_1^* =$	0.0296	0	0	0.4852	0.4852
		0	0	0.4352	0.2824	0.2824
		0	0.4352	0.5648	0	0
		0.4852	0.2824	0	0.2324	0
		0.4852	0.2824	0	0	0.2324
Topology 2	$A_2^* =$	0	0	0.4352	0.2824	0.2824
		0	0.0296	0	0.4852	0.4852
		0.4352	0	0.5648	0	0
		0.2824	0.4852	0	0.2324	0
		0.2824	0.4852	0	0	0.2324
Varying topology	$A_{12}^* =$	0.2741	0.4709	0.0128	0.1211	0.1211
		0.3489	0.2741	0.2458	0.0656	0.0656
		0.2458	0.0128	0.3190	0.2112	0.2112
		0.0656	0.1211	0.2112	0.3280	0.2741
		0.0656	0.1211	0.2112	0.2741	0.3280

Table 5.4. The maximum degree chain matrices for a varying topology network

Topology 1	$A_1^{md} =$	$\begin{bmatrix} 1/3 & 0 & 0 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 1/3 & 2/3 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 0 & 1/3 \end{bmatrix}$
Topology 2	$A_2^{md} =$	$\begin{bmatrix} 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 1/3 & 0 & 1/3 & 1/3 \\ 1/3 & 0 & 2/3 & 0 & 0 \\ 1/3 & 1/3 & 0 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 0 & 1/3 \end{bmatrix}$
Varying topology	$A_{12}^{md} =$	$\begin{bmatrix} 2/9 & 2/9 & 1/9 & 2/9 & 2/9 \\ 1/3 & 2/9 & 2/9 & 1/9 & 1/9 \\ 2/9 & 1/9 & 4/9 & 1/9 & 1/9 \\ 1/9 & 2/9 & 1/9 & 1/3 & 2/9 \\ 1/9 & 2/9 & 1/9 & 2/9 & 1/3 \end{bmatrix}$

matrix. However, MSLEP for non–symmetric system matrices is not a convex problem and cannot be solved by LMIs [24].

5.3. Summary of the Chapter

In this chapter, we have proposed two different methods for constructing system matrices that provide faster convergence. The first method, *Non–Negative Inverse Eigenvalue Problem*, is used to determine all eigenvalues of the system matrix [22]. However, finding necessary and sufficient conditions for the existence of such matrices is still an unsolved problem. Besides, we do not need to determine all eigenvalues as long as the second largest is small enough. Therefore we have introduced another method suggested by Boyd *et al.* [24], *Minimum Second Largest Eigenvalue Problem*, which is used to find the fastest converging system matrix. The main disadvantage of this method is the requirement for connections to be symmetric. It turns out to be a non–convex problem if the connections are not symmetric. Moreover, even when the connections are symmetric, there is a restriction for the weighting coefficients to be symmetric as well, while they are not likely to be the same in practice.

In distributed synchronization, each node knows only its own connections (row i of the adjacency matrix G), not the centralized graph. Therefore the fastest converging system matrix cannot be obtained by a node. On the other hand, we have also shown that even with given centralized graphs for a topology varying network, it is a hard problem to determine the system matrix which will give the best convergence speed.

6. BYZANTINE NODES

The earlier chapters of this thesis studied consensus that can be achieved in distributed networks where nodes are *reliable*. Fault–tolerant networks are the networks that continue to operate properly in the event of failure of some of its nodes. It is important to study what coordinated behavior of process is possible under the assumption that nodes may fail. The chance of a failure occurring in some place in a distributed system may grow arbitrarily large when the number of nodes increases [25]. Most researchers limit the number of these faulty nodes to prove convergence of Byzantine networks [26],[27]. In this thesis, we investigate Byzantine faults in the context of Algorithm (2.2).

Definition 6.0.1. If there is a difference between the actual and transmitted clock data for a node, then the node is said to be *faulty (or a Byzantine node)*.

Byzantine nodes cause instability in networks and it is difficult to find algorithms that can handle these faults [26]. The following example describes a Byzantine network's behavior in lack of a fault–tolerant algorithm.

Example 6.0.1. Consider a three node network in which the third node is Byzantine. Let two non–faulty nodes share their clock information with each other as illustrated in Fig. 6.1. The Byzantine node clock value x_3 is not a function of time, and the first node receives that clock value incorrectly, e.g., k_1x_3. Similarly, the second node receives that value as k_2x_3. k_1 and k_2 are called *Byzantine constants* and they are different than one. Figs. 6.2 and 6.3 depict the simulation results of the network for initial clock values $x_0 = [50, 75, 100]^T$, Byzantine constants $k_1 = 1.1, k_2 = 0.9$ and weighting coefficients $a_{ij} = 1/3$ for $i = 1, 2$ and $j = 1, 2, 3$. Note that Byzantine nodes do not update their clock values using an algorithm.

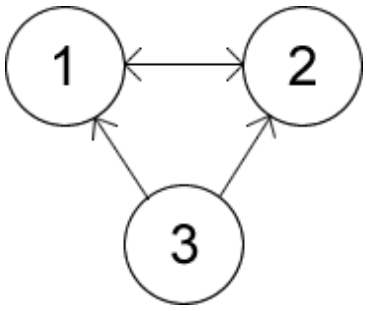

Figure 6.1. A three node network with a Byzantine node

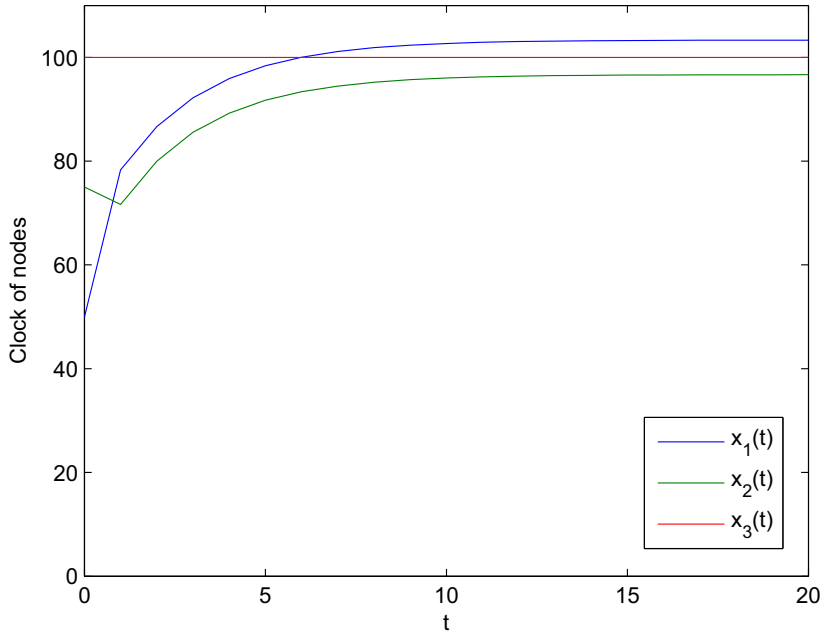

Figure 6.2. Simulation results for a three node network with a Byzantine node

6.1. Single Faced Byzantine

A Byzantine node shares wrong clock information with non–faulty nodes. If a Byzantine node sends the same (and of course incorrect) clock information to non–faulty nodes, it is called *single faced Byzantine*. All Byzantine constants are equal for a single faced Byzantine network.

Let the number of Byzantine nodes be m in an n node network. By defining the

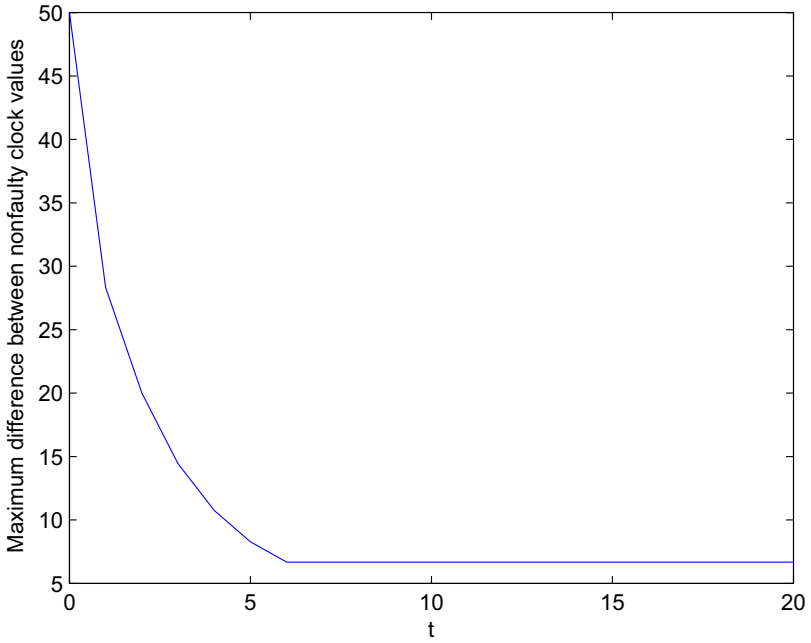

Figure 6.3. The maximum difference between clock values for a network with Byzantine nodes

clock vector

$$x(t) = [x_1(t), ..., x_{n-m}(t), x_{b_1}, ..., x_{b_m}]^T,$$

the algorithm given in (2.4) can be used for the single faced Byzantine networks. The structure of the system matrix should be in the following form:

$$A = \begin{bmatrix} A_1 & A_2 \\ 0 & I \end{bmatrix} \tag{6.1}$$

where adjacency matrices of $A_{1_{(n-m)\times(n-m)}}$ and the non–Byzantine subsystem are the same, and $A_{2_{(n-m)\times m}}$ is a nonzero matrix and composed of weighting coefficients for the clock data received by the Byzantine nodes.

Theorem 6.1.1. *If all single faced Byzantine nodes send their clock information to all non–faulty nodes and non–faulty nodes take average of received clock values, then non–faulty nodes converge to the average of Byzantine clocks even if the non–Byzantine*

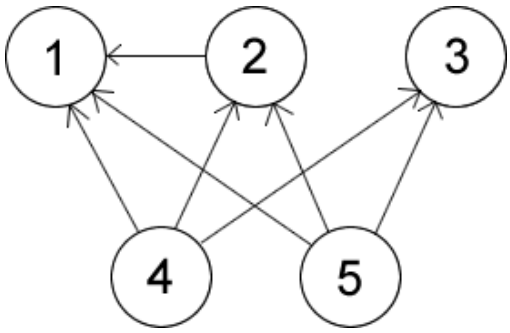

Figure 6.4. A five–node network with two single faced Byzantine nodes

system is not convergent.

Proof. It is well known from the previous sections that the system matrix is row–stochastic and therefore the maximum difference between non–faulty node clock values is guaranteed to be non–increasing. When we take the limit of infinite power of the system matrix, we get

$$lim_{k\to\infty}A^k = \begin{bmatrix} A_1^k & (A_1^{k-1} + ... + A_1 + I)A_2 \\ 0 & I \end{bmatrix} = \begin{bmatrix} 0 & (I - A_1)^{-1}A_2 \\ 0 & I \end{bmatrix}. \tag{6.2}$$

It is straightforward to see from (6.2) that all non–faulty nodes will lose their clock information at the end and therefore $(I - A_1)^{-1}A_2$ is a row stochastic matrix. If every non–faulty node uses the same averaging coefficients for the clock information sent by Byzantine nodes, then A_2 matrix can be decomposed into following two parts

$$A_2 = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & k_{n-m} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \\ \vdots & & \vdots \\ 1 & \dots & 1 \end{bmatrix} \tag{6.3}$$

where $k_i \leq 1$, $i = 1, ..., n-m$ are the weighting coefficients for the information received by the i–th non–faulty node. Let $\tilde{A}_1 = (I - A_1)^{-1}diag\{k_1, ..., k_{n-m}\}$. Since $(I - A_1)^{-1}A_2 = \hat{A}_1 e_{n-m} e_m^T$ is row–stochastic, every row–sum of $\tilde{A}_1$ is equal to $1/m$ and therefore the non–faulty nodes converge to the mean of received Byzantine clock values.

□

Example 6.1.1. Consider the three node network depicted in Fig. 6.4. Let the fourth node send its clock information 50 per cent of its original clock and the fifth node send 90 per cent of its original clock value. Choosing the averaging coefficients as suggested in Theorem 6.1.1, the system matrix becomes

$$A = \begin{bmatrix} 1/4 & 1/4 & 0 & 1/4 & 1/4 \\ 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{6.4}$$

Figs. 6.5 and 6.6 show the simulation results of the network for the initial clock values $x_0 = [60, 70, 80, 90, 100]^T$. The results confirm the convergence of the non–faulty

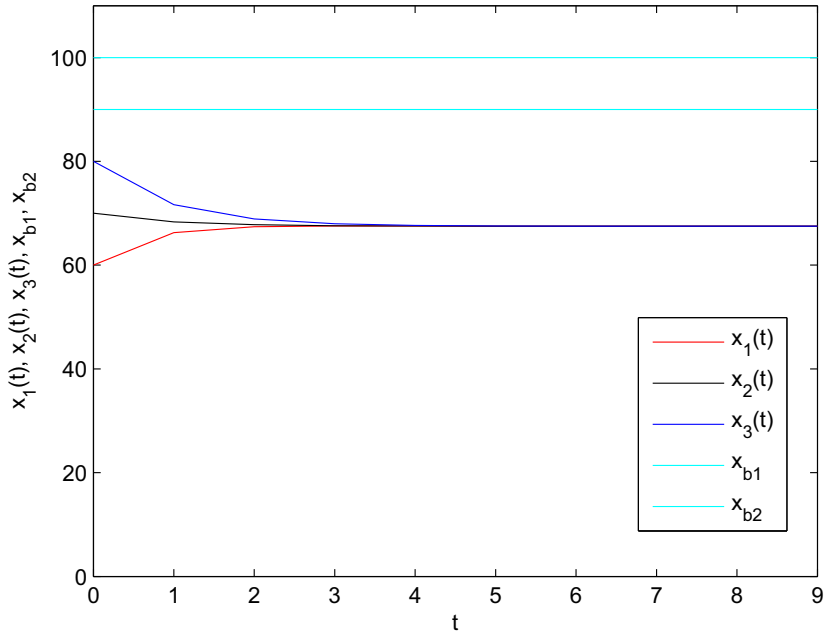

Figure 6.5. Simulation results for a five–node network with two single faced Byzantine nodes

nodes to the mean of received Byzantine clock information, that is

$$x_\infty = \frac{x_{b_1} + x_{b_2}}{2} = \frac{90\frac{50}{100} + 100\frac{90}{100}}{2} = 67.5$$

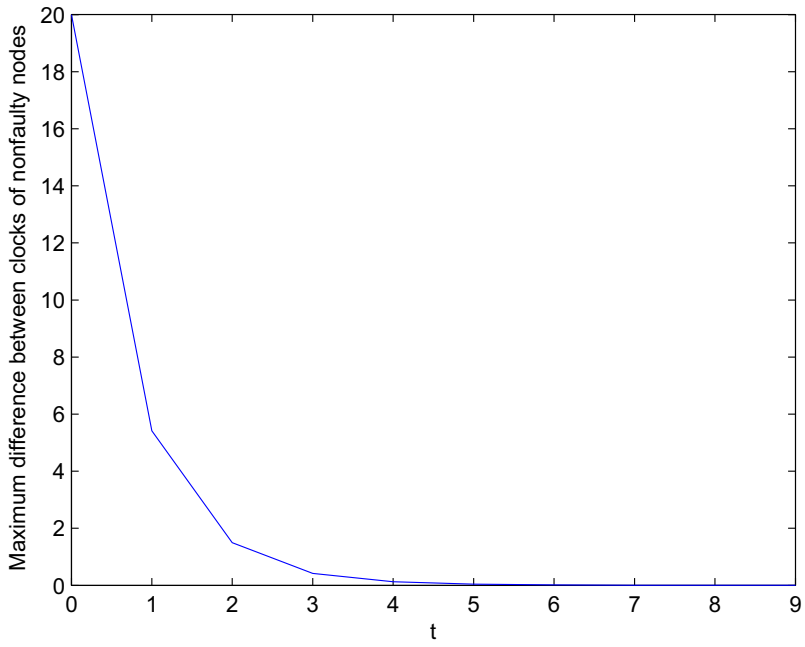

Figure 6.6. The maximum difference between clocks of non–faulty nodes for a five–node network with two single faced Byzantine nodes

Also note that the non–Byzantine subsystem was not convergent by its own.

6.2. Double Faced Byzantine

If a Byzantine node sends different (and wrong) clock information to each non–faulty node, this node is called to be a *double faced Byzantine node*. This type of nodes almost always cause instability in networks. Lamport suggested some algorithms to solve Byzantine Generals Problem, which is also applicable to double faced Byzantine networks [26]. In these algorithms, it is assumed that all nodes communicate with each other (network graph is complete).

6.3. The Algorithm CON

The Algorithm CON is the simplest convergence algorithm for double faced Byzantine networks. The algorithm is given as follows:

$$x_i(t+1) = \frac{\sum_{j \in N_i(t)} x_j(t) + \sum_{k \notin N_i(t)} x_i(t)}{n} \tag{6.5}$$

where $x_i(t)$ is the local clock value of node i at time t, $N_i(t)$ is the set of nodes that shares a value not different from $x(i)$ more than δ.

The algorithm can be also expressed as the following: Each process reads the value of every process's clock and sets its own clock to the average of these values that are not different from its own value by more than δ.

Here, δ has to be chosen properly to achieve synchronization. A very large δ may result in instability since Byzantine nodes will not be detected. Similarly, a very small δ will also result in instability since nodes may not update their own values. The selection of δ will not be discussed in this thesis.

Example 6.3.1. Consider a seven–node network with two double faced Byzantine nodes. Let the network graph be complete (all nodes communicate with each other). The CON algorithm provides bounded convergence as depicted in Fig. 6.7 and 6.8. The bound of convergence is directly related to δ which is chosen to be 18 in the simulations.

6.4. The Algorithm COM

The second convergence algorithm for double faced Byzantine networks is called *The Algorithm COM* [26]. In this algorithm, nodes receive all clock information in the network and set their local clocks to the median of these values. The main disadvantage of the Algorithm COM is that if a byzantine node's clock information is the median value for the corresponding node, the node will update its clock with incorrect

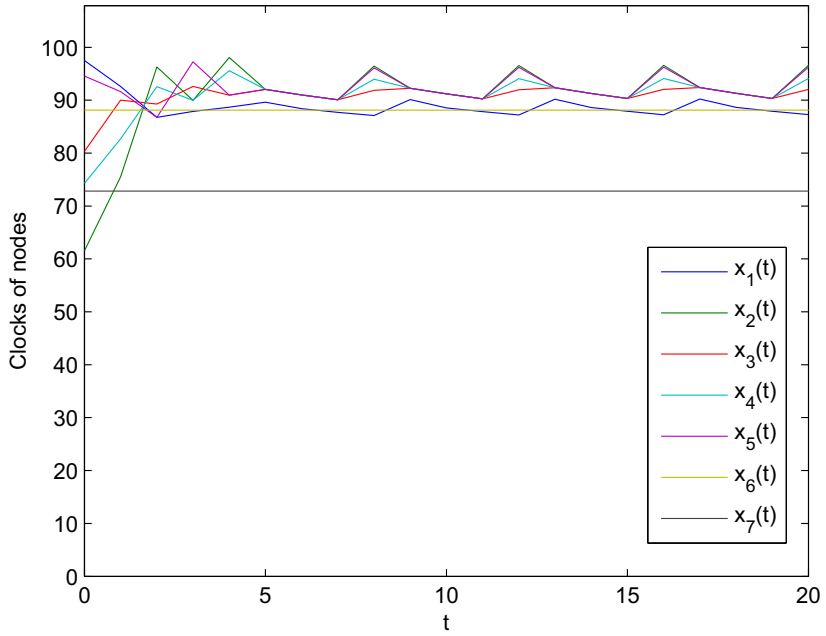

Figure 6.7. Simulation results for a CON algorithm applied seven–node network with two double faced Byzantine nodes

information for that step. However, when no byzantine information is the median of a non–faulty node's received clocks, nodes equalize their clocks for that step. It is clear from these examples that a permanent consensus is not ensured since this algorithm is for the networks where the Byzantine nodes are double faced.

6.5. Summary of the Chapter

In this chapter, we have discussed the convergence algorithm (2.2) for the networks where faulty nodes are present. It is shown by simulations that Byzantine nodes cause instability in the networks. For single faced Byzantine networks, a convergence condition that does not require any restrictions about non–faulty network topology is proposed. However, the convergence condition states that all Byzantine nodes have to share their clock information with all non–faulty nodes in the network, which is not possible in practice. Double-faced Byzantine networks have been also discussed and the convergence algorithms, The Algorithm CON and COM have been investigated. These algorithms require the network graph to be complete which is a very strict assumption.

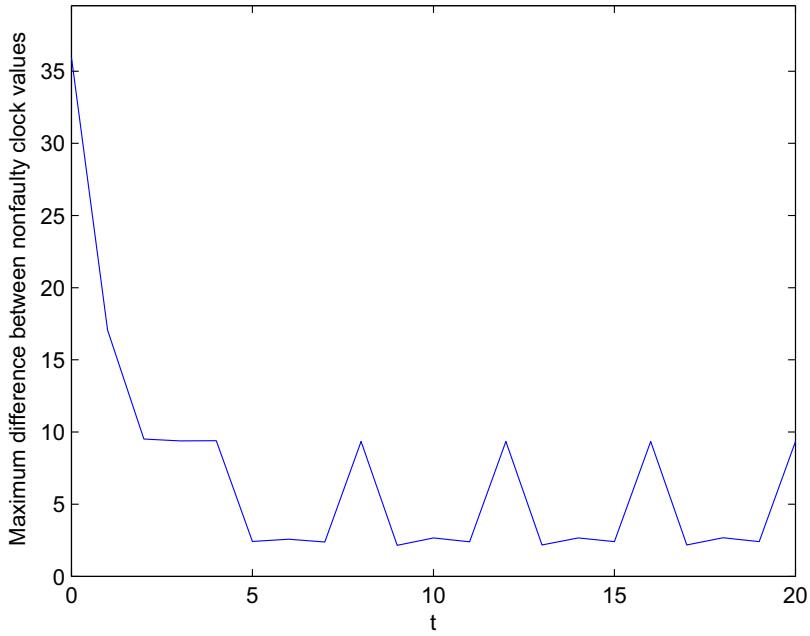

Figure 6.8. The maximum difference between clocks of non–faulty nodes for a CON algorithm applied seven–node network with two double faced Byzantine nodes

We have shown by simulations that the Algorithm CON provides a bounded convergence if the threshold δ is chosen properly, and we have commented on the convergence of the Algorithm COM.

7. CONCLUSIONS

In this thesis, clock synchronization, which is one of the most important tasks in wireless sensor networks, is studied. There are a lot of applications where sensors require using a common notion of time. Since most of the wireless networks are mobile, synchronization of varying topology networks is an important area of research. Throughout the thesis, averaging based distributed synchronization algorithms for both fixed and varying topologies are discussed. Convergence conditions of this synchronization algorithm are investigated in Chapter 2.

Communication delays are unavoidable when data are transmitted through a media on wireless networks. Delay can reduce system performance and it can also turn a stable system into an unstable one if certain conditions are not satisfied. The effect of delay on convergence of consensus algorithm has been studied in Chapter 3 and it is shown that convergence is not adversely affected so long as it is bounded. In order to analyze the effect of delay on convergence speed, some well known topologies are covered in Chapter 4. It is shown that delay does not always have an adverse effect on convergence speed.

In Chapter 5, we have focused on reducing the convergence time for clock synchronization. Fastest converging system matrices have been proposed for networks with symmetric connections by using LMIs and for non–symmetric connections, non–negative inverse eigenvalue problem has been examined.

In Chapter 6, we have discussed faulty nodes which are also known as Byzantine nodes. These nodes cause instability if clock synchronization algorithm is not chosen properly. Sufficient conditions for synchronization of single faced Byzantine networks have been proposed. Future work will concentrate on finding the conditions of distributed consensus algorithms when double faced Byzantine nodes are present in the network.

REFERENCES

1. Sundararaman, B., U. Buy and A. D. Kshemkalyani, Clock synchronization for wireless sensor networks: a survey, *Ad Hoc Networks*, 3(3):281–323, 2005.

2. Cristian, F., Probabilistic clock synchronization, *Distributed Computing*, 3(3):146-158, Jul. 1986.

3. Gusella, R. and S. Zatti, The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD, *IEEE Transactions on Software Engineering*, 15(7):847-853, Jul. 1989.

4. Chang, E. and R. Roberts, An improved algorithm for decentralized extrema–finding in circular configurations of processors, *Communications of the ACM*, 22:281-283, 1979.

5. Vicsek, T., A. Czirok, I. Ben-Jacob, I. Cohen and O. Schochet, Novel type of phase transitions in a system of self–driven particles, *Physics Review Letter*, 75:1226-1229, 1995.

6. Jadbabaie, A., J. Lin and A. S. Morse, Coordination of groups of mobile autonomous agents using nearest neighbor rules, *IEEE Transactions on Automatic Control*, 48(6):988–1001, Jun. 2003.

7. Moreau, L., Stability of multiagent systems with time–dependent communication links, *IEEE Transactions on Automatic Control*, 50(2):169–182, Feb. 2005.

8. Blondel, V. D., J. M. Hendrickx, A. Olshevsky and J. N. Tsitsiklis, Convergence in multiagent coordination, consensus and flocking, In *Proc. IEEE Conf. Decision and Control*, Seville, Spain, Dec. 2005.

9. Li, S. and H. Wang, Multi–agent coordination using nearest neighbor rules: revis-

iting the vicsek model, *CoRR*, cs.MA/0407021, 2004.

10. Olfati-Saber, R. and R. M. Murray, Consensus problems in networks of agents with switching topology and time–delays, *IEEE Transactions on Automatic Control*, 49(9):1520–1533, Sep. 2004.

11. Ren, W. and R.W. Beard, Consensus Seeking in Multi–agent Systems Under Dynamically Changing Interaction Topologies. *IEEE Transactions on Automatic Control*, 50(5):655–661, 2005.

12. Xiao, F. and L. Wang, Consensus protocols for discrete–time multi–agent systems with time–varying delay. *Automatica*, 44(10):2577–2582, 2008.

13. Tsitsiklis, J. N., D. P. Bertsekas and M. Athans, Distributed asynchronous deterministic and stochastic gradient optimization algorithms, *IEEE Trans. Automatic Control*, 31(9):803-812, Sep. 1986.

14. Olshevsky, A. and J. N. Tsitsiklis, On the Nonexisence of Quadratic Lyapunov Functions for Consensus Algorithms, *IEEE Trans. Automatic Control*, 53(11):2642–2645, Dec. 2008.

15. Akar, M. and R. Shorten, Distributed Probabilistic Synchronization Algorithms for Communication Networks, *IEEE Trans. Automatic Control*, 53(1):389–393, Feb. 2008.

16. Liu, Y. and J. Zhu, Consensus Research of Modified Multiagent Networks with Time Delay, *International Conference on Natural Computation*, pp. 339–342, 2008.

17. Wang, W. and J. J. E. Slotine, Contraction analysis of time delayed communications and group cooperation, *IEEE Transactions Automatic Control*, 51(9):712–717, Apr. 2006.

18. Seuret, A., D. V. Dimarogonas and K. H. Johansson, Consensus under Communication Delays, *47th IEEE Conference on Decision and Control*, Cancun, Mexico,

pp. 4922–4927, Dec. 2008.

19. Akar, M. and R. Shorten, Deterministic synchronization algorithms and convergence rates, *Proc. of IEEE International Conference On Networking, Sensing and Control*, Ft. Lauderdale, Florida, U.S.A., April 2006.

20. Wang, L. and F. Xiao, A New Approach to Consensus Problems for Discrete–Time Multiagent Systems With Time–Delays, *Proceedings of the 2006 American Control Conference*, Minneapolis, Minnesota, USA, Jun. 14–16, 2006.

21. Hartfiel, D. J., *Markov set–chains*, Springer, 1998.

22. Orsi, R., Numerical Methods For Solving Inverse Eigenvalue Problems For Nonnegative Matrices, *SIAM Journal on Matrix Analysis and Applications*, 28(1):190–212, 2006.

23. Chu, M. T. and G. H. Golub, *Structured inverse eigenvalue problems*, Acta Numerica, pp. 1–71, Nov. 2002.

24. Boyd, S., P. Diaconis and L. Xiao, *Fastest Mixing Markov Chain on A Graph*, SIAM Review, 46(4):667–689, Dec. 2004.

25. Tel, G., Byzantine Clock Synchronization, *Introduction to distributed algorithms*, Cambridge University Press, 2000.

26. Lamport, L. and P. M. Melliar-Smith, Byzantine Clock Synchronization, *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pp.68–74, Aug. 27–29, 1984.

27. Castro, M. and B. Liskov, Practical Byzantine fault tolerance, *Proceedings of the third symposium on Operating systems design and implementation*, pp.173–186, Feb. 1999.