Byzantine Agreement and Network Failures

by

Franky S. Ling

B.C.S., University of Windsor, 1983

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
April 1986

Approval

Name: Franky S. Ling

Degree: Master of Science

Title of Thesis: Byzantine Agreement and Network Failures

Examining Committee:

Chairman: Dr. Joseph Peters

Dr. Tiko Kameda
Senior Supervisor

Dr. Binay Bhattacharya

Dr. Arthur Lee Liestman

Dr. John Ellis
External Examiner
Department of Computer Science
University of Victoria

Apr 7, 1986
Date Approved

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Byzantine Agreement and Network Failures

Author:

(signature)

Siu Ming Franky Ling

(name)

16 April 1986

(date)

Abstract

Achieving consistency is one of the important issues in distributed computing. In a distributed environment, consistency comes in different forms, e.g., processors must take the same action at some specified time, different clocks should give approximately the same value, processors have to agree on a critical value, etc. Under the worst kind of failure behavior, this problem has been abstracted as the Byzantine Agreement problem. A new Byzantine Agreement algorithm is presented which uses fewer messages than other currently known algorithms under a certain range of parameters and terminates in an optimal number of rounds of message exchanges. Previous research dealt mainly with processor failures and treated communication/link failures as a special case of processor failure. A necessary and sufficient condition is presented for reaching agreement under simultaneous failure of k processors and l links in terms of a graph property, namely, the connectivity function of a graph. Byzantine failure is assumed, i.e., no assumptions are made on the behavior of a faulty component, and in particular, malicious actions taken by faulty components to prevent nonfaulty processors from reaching agreement are taken into account.

Connectivity function is valuable in recognizing the degree of tolerance of a distributed system. Unfortunately, however, it is shown that computing the fixed-pair connectivity function is, in general, *NP*-complete. Among several classes of

graphs investigated, it is shown that the problem remains *NP*-complete for bipartite, chordal, and split graphs; and is solvable in polynomial time for n-cubes, series-parallel graphs, and other graphs with bounded vertex connectivity.

To my parents and Aunt Siu Ching.

Acknowledgements

I wound like to express my sincere gratitude to my Senior Supervisor, Tiko Kameda, for giving me his invaluable advice and guidance throughout the research and his patience in proofreading my thesis.

I would also like to thank the committee members, Binay Bhattacharya, Arthur Liestman, Joseph Peters, and External Examiner, John Ellis, for their proofreading of my thesis and helpful comments and suggestions made. Some ideas in algorithm SPCon(), which is described in Chapter 5, were suggested by Binay Bhattacharya.

Finally, I thank the School of Computing Science at Simon Fraser University for the financial support given to me, and its staffs, especially Ethel Inglis, in giving me the assistance needed.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis is concerned with the fault tolerance of a distributed system, in particular, with Byzantine Agreement and communication failures.

In Chapter 2, we give a brief survey of the Byzantine Agreement problem, which deals with the reaching of an agreement under the worst kind of failure behavior, called Byzantine failure. In Chapter 3, a new Byzantine Agreement algorithm is presented which uses a smaller number of messages than the currently known algorithms for certain ranges of parameters.

In dealing with the Byzantine Agreement problem, most researchers assume that link failure is a special case of processor failure; a link failure is considered as the failure of one of the end processors of the link. In Chapter 4, we separate processor and link failures and give necessary and sufficient conditions for a system to tolerate t processor and l link failures in reaching Byzantine Agreement. A protocol is presented for a Byzantine Agreement to tolerate t processor and l link failures under the sufficiency conditions. One of the necessary and sufficient conditions is stated in terms of a graph property, the *connectivity function*.

In Chapter 5, we study the connectivity function, which defines the number of

vertices and edges whose removal disconnects a graph. It generalizes the concepts of vertex and edge connectivity. Our interest is chiefly in the algorithmic complexity of the problem. We show that the problem of computing the *fixed-pair connectivity function* is, in general, *NP*-complete. Important properties of the connectivity function are investigated and the connectivity functions for certain classes of graphs are determined.

Chapter 6 contains a summary of the results in this thesis and states some open problems.

Chapter 2

Byzantine Agreement

2.1. Introduction

In a distributed system, various faults can occur, e.g., faults caused by component malfunctioning and faulty software, and communication error caused by noise in the communication medium. A single fault happening at an unfortunate time could be disastrous. Thus, one important design issue for a reliable system is how to keep a system functioning properly in the presence of faults.

Many fault tolerant algorithms assume a benign form of failure, called the fail-stop mode. In this mode, once a component fails, it immediately stops functioning and does nothing else. In reality, however, some malicious form of failure can happen, e.g., a faulty component could behave illogically, and a '1' sent by a processor could be interpreted as a '0' by a receiver due to noise in the communication medium. Therefore, a system with a high degree of reliability must tolerate "malicious" faults. Many fault tolerant algorithms do not work if "malicious" failures occur. In a distributed, message passing environment, "malicious" faults have been formalized as the Byzantine failure, which makes no assumptions on the behavior of a faulty component, e.g., a faulty processor can send conflicting information to others, or modify a message while it is relaying the message. Hence, algorithms that handle

Byzantine failures cannot depend on the behavior of a faulty component at all. One way of picturing a system with Byzantine failures is to imagine that there is a demon, with the ability to control the faulty components, to prevent correct processors from working properly. The faulty components, being undetected by correct processors and controlled by the demon, send confusing messages to correct processors.

One useful concept in handling Byzantine failures is that of Byzantine Agreement. Byzantine Agreement is said to have been reached if all correct processors agree on the same value. The value here may be quite general, including the description of an action that should be taken by all correct processors. Some applications are: reliable broadcast which requires that all correct receivers receive the same message; a commit protocol in a distributed database which requires the result of a transaction that involves data at different processors to be consistently installed (either all changes due to a transaction or nothing at all should be reflected in the database). The paper [16] discusses applications of Byzantine Agreement in database system; and replicated computing, in which several processors may be dedicated to compute the same function on an input, and it is essential for all of them to obtain the same input.

A great deal of research has been done on the Byzantine Agreement problem and its variants. The results obtained range from lower bounds to efficient algorithms. We give in this chapter a brief description of some major results.

2.2. Model and Definitions

A *system* consists of a set $P = \{p_1, p_2, \cdots, p_n\}$ of n processors and a communication network which connects the processors. Processors communicate only by sending and receiving messages through the network. Each processor is modeled by an infinite state machine. Initially, each processor p_i is in its initial state and has an initial value $v_i \in V$, where V is a set of values. Note that processors' initial values may be different. For simplicity, we assume that $V = \{0,1\}$. The extension of V to an arbitrary value set is straightforward, since any value can be encoded in binary representation. A *protocol* defines the next state of each processor and the messages to be sent, as a function of the current state and the messages received. A processor is said to be *correct* if it follows the protocol correctly, and is *faulty* otherwise. At the end of protocol execution, every correct processor decides on a value. A *Byzantine Agreement* is said to have been reached if B1 and B2 below are satisfied:

B1. All correct processors decide on the same value.

B2. If all correct processors have the same initial value v, then all correct processors decide on v.

A *weak Byzantine agreement* is said to have been reached when B1 above and B2' below are satisfied:

B2'. If all processors are correct and their initial values are all equal to v, then all processors decide on v.

A protocol is *t-resilient* if, for all executions of the protocol, it terminates in a finite number of state transitions and guarantees that an agreement is reached despite the presence of up to t faulty processors.

Failure Modes.

We list three commonly discussed failure modes:

Fail-stop: once a component fails, it immediately stops functioning, i.e., it neither sends nor receives any message during the rest of a protocol execution.

Omission failure: the only faulty behavior is to omit to execute some step(s) of a protocol.

Byzantine failure: no assumption is made on the behavior of faulty components, i.e., they can act maliciously.

Network Assumptions

We first assume that every two processors are connected by a reliable bidirectional communication link, i.e., the underlying graph of the network is a complete graph and links never fail. We will discuss link failure and the structure of the network later. A message may pass through several nodes of a network, we assume that the receiver of a message can identify the immediate sender, i.e., the last node which relay the message.

2.3. Synchronous Systems

In a synchronous system, execution steps of processors are synchronized. This requires that their local clocks be synchronized within some bound, processors' response time be bounded, and there exists an *a priori* bound on message delay.

It is convenient to divide protocol execution into *rounds*. At the beginning of a

round, each processor sends messages to some processors, possibly to all including itself. It then retrieves all messages that were sent to it in this round and performs its computation.

Let t be an upper bound on the number of faulty processors with Byzantine failure in a system. Pease, Shostak, and Lamport [20] proved that, with n processors, it is impossible to reach Byzantine Agreement if $n \leqslant 3t$. If $n \geqslant 3t+1$, Byzantine Agreement is always possible, which they proved by an algorithm for Byzantine Agreement. Their algorithm terminates in $t+1$ rounds and uses $O(n^{t+1})$ messages.

As the possibility of achieving Byzantine Agreement has been established, it is desirable to seek efficient algorithms and lower bounds for the problem. Fischer and Lynch [13] showed that $t+1$ is a lower bound on the number of rounds needed for any t-resilient Byzantine Agreement protocol. Since there are algorithms that terminate in $t+1$ rounds [20], this bound is tight. As for message complexity, Dolev and Reischuk [10] showed that $\Omega(nt)$ messages are necessary for any t-resilient Byzantine Agreement protocol. As mentioned above, the algorithm in [20] uses $O(n^{t+1})$ messages. Therefore, it is impractical if t is large, since the number of messages sent is probably too large. The most efficient known algorithm in terms of message complexity, described in [9], uses a polynomial number of messages, with a total of $O(nt^2 \log t)$ bits, but it requires $2t+3$ rounds, which is not optimal. Since there exist algorithms that terminate in $t+1$ rounds, and also algorithms that use a polynomial number of messages, it would be interesting to know whether Byzantine Agreement could be reached in $t+O(1)$ rounds of message exchanges, while

exchanging a polynomial number of messages. This problem still remains open. It is noted that the algorithm described in [11] uses $O(nt^3 \log t)$ message bits and terminates in $t+1$ rounds, but the algorithm requires that a system have $2t^2+3t+5$ processors, which is large compared to $3t+1$.

An agreement is said to be *simultaneous* [5] (called *immediate* in [11]), if all correct processors decide in the same round on a value satisfying the conditions of an agreement. If we allow processors to make their decisions in different rounds, an agreement is said to be *eventual*. Clearly, simultaneous agreement is at least as hard as eventual agreement. In a protocol execution instance, there could be no faulty processor, or on the average, the actual number of faulty processors f could be much less than t. It is reasonable to inquire whether a t-resilient protocol can stop earlier if f is less than t. This property is known as *early-stopping* in [11]. It is shown in [11] that $\min(f+2, t+1)$ is a lower bound on the number of rounds for any eventual early-stopping t-resilient Byzantine Agreement protocol. Surprisingly, for simultaneous agreement, the lower bound of $t+1$ rounds still applies for early-stopping. It is shown in [5] that $t+1$ is also a lower bound on the number of rounds for early-stopping simultaneous weak Byzantine Agreement protocol with just fail-stop mode of failures. This means that no simultaneous-agreement t-resilient protocol can stop earlier than $t+1$ rounds even when there is no failure during protocol execution. By using a polynomial number of messages, eventual agreement can be reached [11] in $\min(2f+5, 2t+3)$ rounds when $n = 3t+1$, and in $\min(f+2, t+1)$ rounds when $n \geq 2t^2+3t+5$.

An *authentication protocol* enables a correct receiver to verify that a received message is genuine and to identify the original sender of the message. This is accomplished by adding to a message to be sent by a processor an unforgeable *digital signature* [21]. Thus, by employing an authentication protocol, the behavior of a faulty processor in relaying a message can be restricted to simply dropping the message. If a faulty processor has modified a message, a correct receiver can verify that the relay is faulty. But it is possible for faulty processors to collude and to send conflicting information to others. By using authentication, algorithms exist that can tolerate an arbitrary number of failures [20]. But even with authentication, the lower bound of $t+1$ rounds of message exchanges still applies [12]. A lower bound on the number of digital signatures needed for any authenticated Byzantine Agreement protocol is $\Omega(nt)$ [10]. Unlike the unauthenticated case, where it is unknown whether Byzantine Agreement can be reached in $t+O(1)$ rounds using a polynomial number of messages when $n=3t+1$, an authenticated t-resilient agreement protocol exists that terminates in $t+1$ rounds and exchanges $O(nt)$ messages [12].

Although authenticated Byzantine Agreement algorithms are able to tolerate any number of faults and are more efficient than their unauthenticated counterparts, an authentication protocol incurs extra cost for an authenticated Byzantine Agreement algorithm, and there is no known authentication protocol that is absolutely unbreakable. For a system with no malicious act expected, a checksum scheme probably can provide a good resilience to failures.

Most authenticated agreement algorithms are efficient, simple to implement, and easy

to comprehend. Srikanth and Toueg [22] devised a clever scheme that can transform an authenticated agreement algorithm into an unauthenticated one. The method uses two rounds of unauthenticated message exchanges to replace one round of authenticated message exchange. Thus their method preserves the simplicity of an authenticated algorithm in an unauthenticated one and still uses a polynomial number of messages. For the transformed algorithms to work correctly, a system must have $3t + 1$ processors; this is a requirement for any unauthenticated protocol.

2.4. Asynchronous Systems

In a synchronous system, we assumed that all processors execute a round of message exchange at the same logical global time. There are many reasons for a system to behave asynchronously. For example, a heavily loaded processor could respond slowly, messages sent through a congested network could be delayed, and in datagram service, messages are not guaranteed to be delivered in the order sent, if at all.

Fischer, Lynch, and Paterson [14] proved that, in a completely asynchronous system (i.e., processor response time is arbitrarily long, an *a priori* bound on message delay is precluded, and messages may not be delivered in the order sent), no deterministic agreement protocol can reach weak Byzantine Agreement in the presence of one fail-stop mode failure. In a later paper [8], Dolev, Dwork, and Stockmeyer extended the above result. Still assuming fail-stop mode, they identified five critical system parameters:

1. processors synchronous(F) or asynchronous(U),
2. communication synchronous(F) or asynchronous(U),

3. message order synchronous(F) or asynchronous(U),

4. broadcast transmission(F) or point-to-point transmission(U),

5. atomic receive/send(F) or separate receive and send(U).

In the above listing F means favorable while U means unfavorable. Broadcast means that broadcasting takes place in bounded time. Atomic receive/send means that the time for receiving, processing and sending of a message is bounded by a constant. In total, there are $32 = 2^5$ cases as parameterized by the above 5 parameters. For n processors, they identified four groups of cases where n-resilience (fail-stop mode) is possible, but the weakening of any parameter from favorable to unfavorable is sufficient to make t-resilience unachievable (t is either 1 or 2). These four "minimal" groups are (they cover 17 out of 32 cases):

1. synchronous processors and synchronous communication,

2. synchronous processors and synchronous message order,

3. broadcast transmission and synchronous message order,

4. synchronous communication, broadcast transmission and atomic receive/send.

When Byzantine failure is considered, Case 1 can be solved; in fact, it is the synchronous model assumed in most literature. The paper [1] describes an algorithm to handle Byzantine failures in an asynchronous system (case 4: processor U and communication F) by imposing a phase protocol (it generates artificial phases). No Byzantine Agreement protocol has yet been found for case 2 or case 3.

2.5. Network Failures

In the preceding sections, we assumed that a network is completely connected and communication is reliable. Actually, what a Byzantine Agreement protocol needs is a way to guarantee that any two correct processors in a system can communicate reliably and can identify the original sender of a received message if the sender is correct. Dolev [6] showed that, in the presence of up to t faulty processors in a system, a necessary and sufficient condition for this purpose is that the vertex connectivity for the underlying graph of a network is $2t + 1$. This result also holds when the structure of the network is unknown to the processors [7].

When discussing communication link failures, most researchers assume that one of the two end nodes of the faulty link is faulty. In Chapter 4 of this thesis, we separate processor and link failures, and describe necessary and sufficient conditions for a system to tolerate t faulty processors and l faulty links. The conditions are: $n \geq 3t + 1$ and the fixed-pair connectivity function f for every pair of distinct processors s and t which participate in the protocol satisfy either (1) $f(G,s,t,2t) \geq 2l+1$ or (2) $f(G,s,t,a) \geq 2l+1$ and $f(G,s,t,a+1)$ is undefined, where G is the underlying graph of the network and a is any arbitrary positive integer. For a definition of f, please see Chapter 5. Informally, the function defines the number of vertices and edges that must be removed in separating two vertices s and t, e.g., if $f(G,s,t,a) = b$, we can separate s and t by removing a vertices and b edges from G.

Chapter 3

A Byzantine Agreement Algorithm

3.1. Introduction

Two important measures of the efficiency of a Byzantine Agreement algorithm are the number of rounds of message exchanges needed and the amount of information exchanged. The discussion in this chapter concentrates on unauthenticated algorithms. Let n be the number of processors in a system and t be the maximum number of faulty processors that an algorithm can tolerate. It is shown in [13] that $t+1$ is a lower bound on the number of rounds needed, and in [10] that $\Omega(nt)$ is a lower bound on the number of messages required by any Byzantine Agreement algorithm. The first published Byzantine Agreement algorithm [20] runs in $t+1$ rounds, and is therefore optimal with respect to the number of rounds, but the algorithm needs to exchange $O(n^{t+1})$ *messages*. The algorithm in [9] uses a polynomial number of *bits*, $O(t^3 \log t)$, in the exchanged messages, but the number of rounds required in the worst case is $2t+3$, which is roughly twice the optimal. These algorithms work correctly when $n \geq 3t+1$. It has been an open problem whether Byzantine Agreement can be made with a polynomial number of messages in less than $2t+3$ rounds. Another interesting problem is to find the trade-off between the number of rounds and the number of messages needed. One might expect that when a system has more correct processors, the system would handle failures efficiently. The number of correct

processors in a system could well be added to the above trade-off factors. In fact, the algorithm described in [11] uses $O(nt^3 \log t)$ message bits and runs in $t+1$ rounds, when the number of processors $n = 2t^2 + 3t + 5$. Let $t = d^2+2d$, where d is a positive integer, and $n = 9t - 12d + 1 = 9t - \sqrt{t+1} + 13$. We show in this chapter that Byzantine Agreement can be reached in the presence of t processor failures in $d^2 + 3d + 4 = t + \sqrt{t+1} + 3$ rounds using a total of $O(t^{\sqrt{t+3}} \log t)$ message bits. Currently known t-resilient Byzantine Agreement algorithms with $n < 2t^2 + 3t + 5$ that terminate in less than $2t + 3$ rounds need to exchange $O(n^{t+1})$ messages. With a slight modification, our algorithm can terminate in $t + 1$ rounds with a small increase in message bits.

We give an informal description of our Byzantine Agreement algorithm before presenting it. Our algorithm has two levels - *upper level* and *lower level*. Processors are divided into disjoint groups, each of size $3d+1$. At the upper level, each group behaves as a "processor" and executes a known Byzantine Agreement algorithm; we use the algorithm in [20]. If each group has one processor, then our algorithm is essentially the same as the algorithm in [20].

Depending on a parameter of our algorithm, a group can have more than one processor. The first question to be answered is: How do two groups communicate? We require that all correct processors in a group, say group g, first agree on what "group message" to send and then all correct processors of group g send the same "group message" to all processors in the system.

A processor, say p, outside group g will receive as many correct "group messages"

from g as there are correct processors in group g. If the majority of processors in group g are correct, then the majority of the messages that p receives are correct and can be used as group g's "group message".

The lower level accomplishes the goal of reaching an agreement among correct processors in a group on what "group message" to send by invoking a known Byzantine Agreement algorithm; we also use the one in [20]. If a group has at most d faulty processors, the goal can be achieved if and only if the group has at least $3d+1$ processors. This is why we call a group with $3d+1$ processors *correct* if its has no more than d faulty processors; otherwise it is said to be *faulty*.

If our algorithm can handle at most h faulty groups, the maximum number of faulty processors, t, our algorithm can handle must satisfy $\lfloor \frac{t}{d+1} \rfloor \leq h$ (a group is faulty if it has d+1 or more faulty processors); otherwise t faulty processors may be distributed among groups in such a way that more than h groups are faulty. Thus, $t \leq h(d+1)+d$.

The Byzantine Agreement algorithm of [20] terminates in t+1 rounds and exchanges $O((3t+1)^{t+1})$ messages, if n=3t+1. By splitting processing into two levels, the upper level with 3h+1 groups terminates in h+1 upper level rounds and exchanges $O((3h+1)^{h+1})$ "group messages", and each lower level agreement (among 3d+1 processors) embedded in each of the h+1 upper level rounds terminates in d+1 rounds and exchanges $O((3d+1)^{d+1})$ messages. In the next section we present our algorithm and in Section 3.4 we discuss complexity in more detail.

Figure 3-0 gives an example of the algorithm to be presented. In this example h,d=1, and therefore there are four groups, each with four processors.

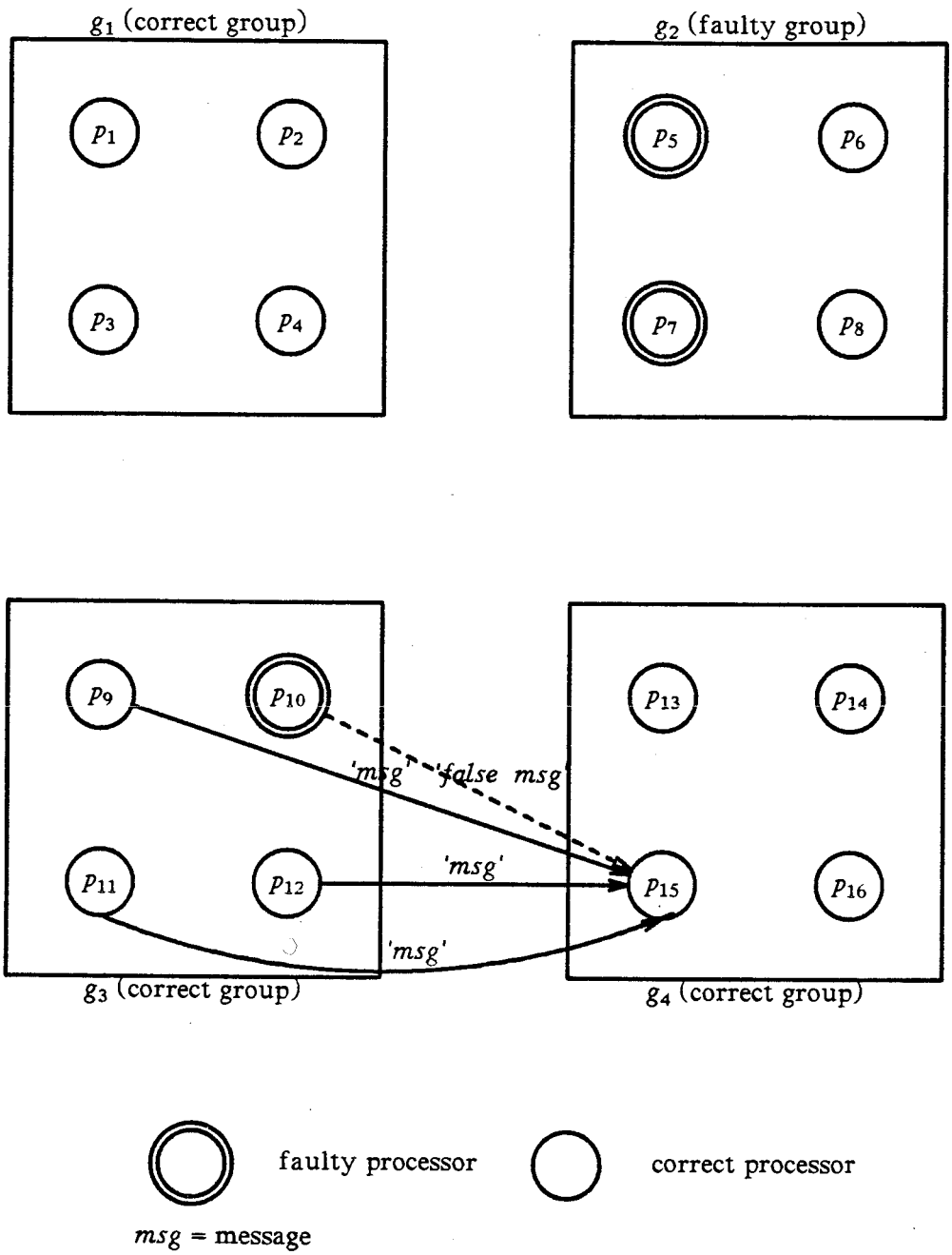

Figure 3-1: A scenario of algorithm GrByz().

3.2. Algorithm

Let h, d be positive integers. The algorithm we present in this section enables Byzantine Agreement to be reached in the presence of up to $hd + h + d$ processor failures, provided that the system has $(3h + 1)(3d + 1)$ processors. The algorithm assumes the following: each processor can directly communicate with every other processor reliably and the processors are divided into $3h + 1$ disjoint groups, each with $3d + 1$ processors, and each processor can recognize to which group any processor belongs. A processor is said to be *correct* if it follows the algorithm; otherwise it is *faulty*. A group is said to be *correct* if it has no more than d faulty processors; otherwise it is *faulty*.

To present our algorithm, we first introduce the following notation.

P = set of all processors,
g_i = group i,
$G = \{g_1, \cdots, g_{3h+1}\}$, set of all groups,
$P(g_i)$ = set of all processors in group g_i.

The following procedures will be called by algorithm GrByz():

Majority(X) : if over half of the elements in the vector $X = (x_1, x_2, \cdots, x_{3d+1})$, where each x_j is a set, are equal to some x_j, $1 \leqslant j \leqslant 3d+1$,
return x_j;
otherwise return *Nil*.

Byz$(P(g),d,v)$: returns v';

Byz$(P(g),d,v)$ is a Byzantine Agreement protocol that can tolerate up to d faulty processors, where $P(g)$ is the set of participating processors, and v is the initial value of a processor in $P(g)$ in upper level round 1. Note that each participating processor

may have different v, and v is a message set in later upper level rounds in algorithm GrByz(). v' is the agreed "value" that satisfies the Byzantine Agreement conditions. The algorithm we chose [20] can be applied to an arbitrary value set, i.e., v is not restricted to {0,1}. In upper level round 1, we assume that v is either 0 or 1. Each processor p executes the following algorithm. Let g be the group to which p belongs.

Algorithm GrByz(P, G, h, d);
begin

upper level round 1: /* Agree on group g's initial value */

Let v be the initial value of p;
v' := Byz($P(g)$, d, v); /* v' is used as group g's initial value */
send message set $\{g{:}v'\}$ to all processors including itself;

upper level round R+1, where $1 \leqslant R \leqslant h$:

for each group $g_j \in G$; /* get the message set from g_j by applying Majority() */
begin
Let $q_1, q_2, \cdots\cdots, q_{3d+1}$ be the $3d$+1 processors in group g_j,
and M_{q_i}, where $1 \leqslant i \leqslant 3d+1$, be the set of messages
that p received from processor q_i in upper level round R;
$X := (M_{q_1}, M_{q_2}, \cdots, M_{q_{3d+1}})$;
GrM_{g_j} := Majority(X);
/* if g_j is correct, GrM_{g_j} is the message set from g_j */
end;
/* Agree with all processors in $P(g)$ on
the received message set from each group */
Perform the following $3h$+1 computations in parallel using d+1 rounds:
begin
$AgreedGrM_{g_1}$:= Byz($P(g)$, d, GrM_{g_1});
$AgreedGrM_{g_2}$:= Byz($P(g)$, d, GrM_{g_2});
...
$AgreedGrM_{g_{3h+1}}$:= Byz($P(g)$, d, $GrM_{g_{3h+1}}$);
end;
send message set $\{g{:}m \mid m \in AgreedGrM_{g_j}, 1 \leqslant j \leqslant 3h+1\}$
to all processors including itself;

upper level round h+2: /* make decision in this round */

for each group $g_j \in G$;
begin
Let $q_1, q_2, \cdots, q_{3d+1}$ be the $3d+1$ processors in group g_j,
and M_{q_i}, where $1 \leq i \leq 3d+1$, be the set of messages
that p received from processor q_i in upper level round h+1;
$X := (M_{q_1}, M_{q_2}, \cdots, M_{q_{3d+1}})$;
GrM_{g_j} := Majority(X);
end;
$M^* := \{m \mid m \in GrM_{g_j}, 1 \leq j \leq 3h+1\}$;
v' := value obtained by applying the algorithm in [20]
on the message set M^*;

end.

3.3. Proof of Correctness

Let T be the set of correct processors and T_g be the set of correct groups. The following lemma says that if all correct processors in a correct group $g_j \in T_g$ send the same set of messages in upper level round R, then every correct processor will compute the same set of messages in upper level round $R+1$ by applying Majority() to the set of messages received from all the processors in group g_j in upper level round R.

Lemma 1: Let g_j be a correct group and suppose all correct processors of g_j send message set M to all processors in P in upper level round R. Then each correct processor $q \in T$ obtains $GrM_{g_j} = M$ by applying Majority(X) in upper level round $R+1$.

Proof. Since there are at least $2d+1$ correct processors and at most d faulty processors in g_j, any correct processor receives at least $2d+1$ correct messages carrying M. □

The next lemma asserts that a correct group (more precisely, all the correct processors in a correct group; g_b in Lemma 2 below) will send the same message set to every processor in upper level round $R+1$ (part (a)). In addition, if it receives a set of messages from a correct group in upper level round R, it will relay them faithfully (part (b)).

Lemma 2: Let γ be a string of names of groups, and l be the number of names in γ. For $p \in T$, define $\sigma_p(g_b{:}\gamma) = v$, if and only if there is a message $m = g_b{:}\gamma{:}v \in GrM_{g_b}$, where GrM_{g_b} is the message set that p computes by applying Majority(X) in upper level round $l+2$ (see upper level round R+1 in the above algorithm). Let p, q be any two correct processors and let $g_b \in T_g$.

(a) $\sigma_p(g_b{:}g_c{:}\gamma) = \sigma_q(g_b{:}g_c{:}\gamma)$ for any $g_c \in G$.

(b) $\sigma_p(g_b{:}g_c{:}\gamma) = \sigma_p(g_c{:}\gamma)$, if $g_c \in T_g$.

Proof.

(a) Since g_b is a correct group, all correct processors in it agree on the same message set before sending it (see the step Byz() in GrByz). By Lemma 1, every correct processor gets the same message set from g_b by applying Majority(), therefore $\sigma_p(g_b{:}g_c{:}\gamma) = \sigma_q(g_b{:}g_c{:}\gamma)$.

(b) Now we consider the case where $g_c \in T_g$. Since g_c is a correct group, all correct processors in g_c agree on $\gamma{:}v$ before sending $g_c{:}\gamma{:}v$, and, by Lemma 1, every processor receives $g_c{:}\gamma{:}v$ in upper level round l+2. There are at most d faulty processors in g_b, therefore, by applying Byz() in upper level round l+2, every correct processor in g_b agrees on $g_c{:}\gamma{:}v$. Since the value sent by the correct processors in a correct group

(e.g., g_b) is the agreed value prefixed by the group's name, each correct processor p obtains the message $m = g_b{:}g_c{:}\gamma{:}v$ from g_b. □

Before proving Theorem 4, we state the following sufficient conditions [20] for achieving a Byzantine Agreement from a message set M^* obtained after $t+1$ rounds of message exchanges in the presence of t faults.

C1. Every correct processor relays all the messages it received from previous rounds to every processor, and the relayed messages are prefixed by its name. It also sends the messages to itself for the purpose of recording its history.

C2. If a processor is correct, it relays the message without modifying it.

C3. Every correct processor sends its initial value to every processor in the first round.

C4. The total number of processors is at least $3t+1$ and the total number of faulty processors is at most t.

In applying these conditions to groups of processors, C2 cannot be used, since the message out of a faulty group may not be uniquely defined. Since a message sent by a faulty processor is arbitrary, condition C2 can be replaced by C2' below.

C2'. If a processor is correct, it relays the messages from correct processors without modifying it.

Lemma 3: Let the number of faulty processors be not more than $hd+h+d$. Algorithm GrByz(P, G, h, d) satisfies C1, C2', C3, and C4 if we replace "processor" by "group" in C1, C2', C3, and C4.

Proof. Since there are at most $hd+h+d$ faulty processors, and a group is faulty if it has more than d faulty processors, there are at most h faulty groups and C4 is

satisfied. Let the agreed value v' obtained by executing Byz() in upper level round 1 for each group be the group's initial value. We prove by induction on the number R of upper level rounds that C1, C2', and C3 are also satisfied. If $R = 1$, every correct processor in a correct group g agrees on the same value v' before sending $\{g{:}v'\}$ to all processors. By Lemma 2, C1, C2', and C3 are satisfied. Assume the lemma is true for $R \leqslant R'$, where $R' \geqslant 1$. We now prove the case where $R = R' + 1$. By Lemma 2, which says that a correct group sends the same message set to all processors and relays message sets from correct groups faithfully received in upper level round R', thus C1 and C2' are satisfied for upper level round R. And finally, C3 is satisfied by upper level round 1 of the algorithm. □

Theorem 4: Let the number of processors be $(3h + 1)(3d + 1)$. If the number of faulty processors is at most $hd + h + d$, then algorithm GrByz(P, G, h, d) guarantees Byzantine Agreement.

Proof. Since M^* is obtained after $h + 1$ rounds of message exchanges by groups, and by Lemma 3, algorithm GrByz(P, G, h, d) satisfies C1, C2', C3, and C4, and every correct processor decides on the same value by applying the algorithm of [20] on M^*, therefore condition B1 of the Byzantine Agreement conditions is satisfied. Assume that every correct processor has the same initial value v. Then by executing Byz() in upper level round 1, the initial value of every correct group is v. Therefore, the decision obtained from M^* in upper level round $d + 2$ is also v, and condition B2 of the Byzantine Agreement conditions is also satisfied. □

3.4. Complexity

Theorem 5: Let h, d be positive integers. The number of rounds of message exchanges performed by GrByz(P, G, h, d) is $(h+1)(d+2)$.

Proof. In upper level round R+1, $1 \leq R \leq h$, $3h$+1 Byz()s are invoked. Each Byz($P(g)$, d, M) runs in $d+1$ rounds [20]. Since two Byzantine Agreement algorithms Byz($P(g)$, d, GrM_{g_i}) and Byz($P(g)$, d, GrM_{g_j}), where $i \neq j$, have no interactions, they can run in parallel. Thus, the $3h$+1 Byz()s in an upper level round can run in d+1 rounds. When $R = 0$, only one Byz() is invoked, which needs $d+1$ rounds. Adding the round of sending "*AgreedGrM*", each upper level round has d+2 rounds. There is no sending and receiving in upper level round $h+2$. Therefore, the total number of rounds is $(h+1)(d+2)$. □

Theorem 6: Let the number of processors be $n = (3h+1)(3d+1)$ and $hd+h+d$ be the upper bound on the number of faulty processors in P. Algorithm GrByz(P, G, h, d) reaches Byzantine Agreement using $O(d(3d+1)^{d+2}(3h+1)^2 \log d + (3d+1)^{d+2}h(3h+1)^{h+1} \log h + d^2h(3h+1)^{h+2} \log h)$ message bits.

Proof. Consider the message bits sent by a correct processor p in upper level round R+1, $1 \leq R \leq h$. p participates in $3h$+1 Byz()s, and in each Byz(), p sends

$$(3d+1)^{d+1} \qquad (1)$$

messages. Let *msg* be a message sent in a Byz() in upper level round R+1. *msg* consists of two parts: the route part and the value part. The route part is a sequence of processor names via which the message has been routed. The value part is the message set GrM_{g_i}, for some $g_j \in G$. The route part of *msg* has at most d+1 processor names in it. Now consider how many bits are needed to encode GrM_{g_j}. Each valid message in GrM_{g_j} has its route part consisting of R group names and the

first group name in the route part must be g_j. Thus, there are $O((3h+1)^{R-1})$ messages in GrM_{g_j}. The value part of a message in GrM_{g_j} is either 0 or 1. Thus GrM_{g_j} can be encoded in

$$O((3h+1)^{R-1}R \log h)) \text{ bits} \tag{2}$$

and *msg* can be encoded in

$$x = O((3h+1)^{R-1}R \log h + (d+1)\log d)) \text{ bits.} \tag{3}$$

Therefore, by (1), in each Byz() in upper level round R, p sends $O(x(3d+1)^{d+1})$ bits. Since there are $3h+1$ Byz()s, p sends

$$O(x(3d+1)^{d+1}(3h+1)) \text{ bits.} \tag{4}$$

Let $M = \{g{:}m | m \in AgreedGrM_{g_j}, 1 \leqslant j \leqslant 3h+1\}$ be the message set that p sends in the last round of upper level round $R+1$. The route part of a message in M consists of $R+1$ group names. The value part is either 0 or 1. M can be encoded in $y = O((3h+1)^R R \log h)$ bits. Since p sends M to all processors, p sends $O((3h+1)(3d+1)y)$ bits in the round of sending M. In upper level round $R+1$, including (4) p thus sends $O(x(3d+1)^{d+1}(3h+1) + (3h+1)(3d+1)y)$

$= O((3h+1)^R(3d+1)^{d+1}R \log h + (d+1)(3d+1)^{d+1}(3h+1)\log d + (3d+1)(3h+1)^{R+1} R \log h)$ bits.

Summing over all upper level rounds, the total number of message bits sent by p is of

$O(d(3d+1)^{d+1}(3h+1)\log d + (3d+1)^{d+1}\log h\Sigma_{R=0}^{h} R(3h+1)^R + d(3h+1)\log h\Sigma_{R=0}^{h} R(3h+1)^R)$

$= O(d(3d+1)^{d+1}(3h+1)\log d + (3d+1)^{d+1}h(3h+1)^h\log h + d\cdot h(3h+1)^{h+1}\log h).$

Since there are $(3h+1)(3d+1)$ processors, the total number of message bits exchanged by GrByz(P, G, h, d) is of

$$O(d(3d+1)^{d+2}(3h+1)^2\log d + (3d+1)^{d+2}h(3h+1)^{h+1}\log h + d^2h(3h+1)^{h+2}\log h).$$

□

We can eliminate one round out of a total of $d+2$ rounds of message exchanges in each upper level round, with a small increase in message bits. Thus GrByz() can terminate in an optimal number of rounds. We prove this in the next theorem.

> **Theorem 7:** Let the number of processors be $n = (3h+1)(3d+1)$ and $hd + h + d$ be the upper bound on the number of faulty processors. Byzantine Agreement can be reached in $hd + h + d + 1$ rounds using $O(d(3d+1)^{d+2}(3h+1)^3\log d + (3d+1)^{d+2}h(3h+1)^{h+2}\log h + d^2h(3h+1)^{h+3}\log h)$ message bits.

Proof. Each upper level round in algorithm GrByz(P, G, h, d) has $d+2$ rounds. $d+1$ rounds for the procedures Byz() to agree on the groups' messages received in a previous upper level round, (or the initial value in upper level round 1), and an extra round to send the agreed message set to all processors. We now show that this extra round can be eliminated. The agreed value (*AgreedGrM*) agreed by a processor in procedure Byz() is based on the messages it received in round $d+1$ of the procedure Byz(). If all correct processors in a correct group, say group g, send the message set in round $d+1$ of the procedure Byz() to all processors in P, instead of to only the processors within group g, other correct processors (outside group g) can agree on the same value (*AgreedGrM*) as the correct processors do in group g. If group g is faulty, it does not matter whether the extra round is in the algorithm or not, since

faulty groups can do anything. Since we have h+1 upper level rounds engaged in message exchanges, after the elimination of an extra round in each upper level round, the total number of rounds is $(h+1)(d+1) = hd + h + d + 1$.

We now consider the extra message bits introduced in eliminating the extra round. During the last round of Byz(), instead of sending the message set to $3d$+1 processors in the group to which it belongs, p now sends the message set to all $(3h + 1)(3d + 1)$ processors. Therefore, we increase the total number of message bits at most by a factor of $3h$+1. The total number of message bits is therefore of $O(d(3d+1)^{d+2}(3h+1)^3 \log d + (3d+1)^{d+2}h(3h+1)^{h+2} \log h + d^2h(3h+1)^{h+3} \log h)$. □

The message complexity of algorithm GrByz() is a function of h and d. Let us consider the values of h and d such that the message bits used by algorithm GrByz() is minimized. Given t, the number of faulty processors we want to tolerate, in order to get GrByz() to work correctly, h and d must satisfy the inequality:

$$t \leqslant hd + h + d. \tag{5}$$

For simplicity, let $t = c^2 + 2c$, where c is an integer greater than 1. We claim that when $h = d = c$ the message bits used by GrByz() is very close to the minimum when compared to other values of h and d under constraint (5) above.

To prove this claim, let $O(m(h,d))$ be the message complexity of GrByz(), where $m()$ is a function of h and d. For simplicity we use an approximate formula $m(h,d) = (3d+1)^{d+2}h(3h+1)^{h+2}$ (see Theorem 7). Since h and d must satisfy (5), let $d = c-q$, where q is a real number less than c. Since $m()$ increases as h and d

increase, to get smaller message complexity, we want to get as smaller h as possible under constraint (5), therefore we can express h as $c+q(c+1)/(c-q+1)$ and $m(h,d)$ as a function of q. We want to show that the message complexity increases as q increases. This means that the first derivative $\frac{dm(q)}{dq} > 0$. This can be shown as follows:

$$\frac{dm}{dq} = -h(3h+1)^{h+2}(3d+1)^{d+2}\left(\frac{3(d+2)}{3d+1} + ln(3d+1)\right)$$
$$+ (3d+1)^{d+2}h(3h+1)^{h+2}\frac{(c+1)^2}{(c-q+1)^2}\left(\frac{3(h+2)}{3h+1} + ln(3h+1)\right)$$
$$+ (3d+1)^{d+2}(3h+1)^{h+2}\frac{(c+1)^2}{(c-q+1)^2} > 0,$$

for $1 \leqslant q \leqslant c$, because the term $\frac{(c+1)^2 3(h+2)}{(c-q+1)^2(3h+1)}$ increases faster than the term $\frac{3(d+2)}{3d+1} = \frac{3(c-q+2)}{3(c-q)+1}$ as q increases in the interval $0 \leqslant q \leqslant c$ and when $q=0$ the two terms are equal.

The case for $d = c+q$ can be treated similarly.

When $h = d$, we have the following corollary.

Corollary 7.1: Let the number of processors be $n = (3d+1)^2$ and d^2+2d be the upper bound on the number of faulty processors. Byzantine Agreement can be reached in d^2+2d+1 rounds using $O(d(3d+1)^{d+6}\log d + d(3d+1)^{2d+4}\log d)$ message bits.

3.5. Concluding Remarks

We assume that $n = (3h+1)(3d+1)$ processors are divided into $(3h+1)$ groups. This can easily be arranged if all processor names are unique.

In a large network that spans a large geographical area, the message delay time is

longer and the communications cost is higher for a pair of widely separated nodes than for a pair of close nodes. Another advantage of our algorithm is that by grouping processors that are close to each other, two distant nodes that belong to different groups need to communicate only once in an upper level round of our algorithm. In contrast to other Agreement algorithms in which every pair of nodes must communicate with each other in each of the $t+1$ rounds, in our algorithm, with $h = d$ two processors in two different groups need only $\sqrt{t+1}$ rounds of message exchanges.

Chapter 4

Network Failures

4.1. Introduction

In this chapter, we present a protocol for two nodes, u and v, of a network to communicate reliably in the presence of up to t processor and l link Byzantine failures. The protocol works under the condition that the *fixed-pair connectivity function* f (for a definition of f, please see the next chapter) satisfies either (1) $f(G,u,v,2t) \geq 2l+1$ or (2) $f(G,u,v,a) \geq 2l+1$ and $f(G,u,v,a+1)$ is undefined, where G is the underlying graph of the network and a is any positive integer. The second case deals with the situation in which there are $2l+1$ direct links between u and v. In that situation, if only up to l links are faulty, the correct message can be obtained by a simple majority applied on the messages received from the $2l+1$ direct links. We will also prove that the above condition is a necessary condition for u and v to communicate reliably in the presence of up to t processor and l link Byzantine failures.

Most previous research on Byzantine Agreement assumed that link failure is a special case of processor failure, and if link failure occurs, one of the two end nodes of the faulty link is regarded as faulty. We separate processor and link failures here. If a Byzantine Agreement algorithm can tolerate up to t processor failures

when no link failure occurs, then by using our protocol for communication, Byzantine Agreement can be achieved in the presence of up to t processor and l link Byzantine failures, provided the network satisfies the sufficient condition for reliable communication. If two correct processors cannot communicate reliably, no agreement can be achieved. By combining the necessary and sufficient condition for the reaching of Byzantine Agreement in the presence of up to t processor failures without link failure and the necessary and sufficient condition for reliable communication in the presence of up to t processor and l link failures, a necessary and sufficient condition for Byzantine Agreement under t processor and l link failures is obtained.

4.2. Protocol

If a network is not completely connected and the vertex connectivity of the network is at least $2t + 1$, a receiver can get a correct message from a correct sender, even though there are up to t faulty processors in the network, by executing the protocol described in [7] (from now on we will refer to it as the SR protocol). The function of the SR protocol is to achieve reliable communication among correct processors. It can be used in Byzantine Agreement protocols for communication when the network is not completely connected.

The SR protocol works under the following assumptions:

A1. The processors are arranged in a network with the vertex connectivity of the network at least $2t + 1$. Each communication link is bidirectional, i.e., message can be sent in both direction.

A2. Every processor knows the names of all the members of the network and all names are unique.

A3. Processors communicate only by sending messages along links.

A4. Every processor can identify the neighbor from which it receives each message.

A5. A message includes the route through which it was delivered.

A6. A correct processor relays a message only after it appends to the message the name of the processor from which it received the message, and a receiver upon receiving a message appends to the message the name of the last relay.

A7. A correct processor relays messages without either altering them or eavesdropping on their values.

A8. There exists an *a priori* upper bound on the delay in relaying a message by a correct processor.

A9. There exists an upper bound, t, on the number of faulty processors in the system.

We prove that if A1 and A9 are changed to A1' and A9' below, respectively, two correct processors can communicate reliably in the presence of t processors and l link Byzantine failures.

A1'. The fixed-pair connectivity function f for every two nodes u and v which participate in the protocol satisfies either (1) $f(G,u,v,2t) \geq 2l+1$ or (2) $f(G,u,v,a) \geq 2l+1$ and $f(G,u,v,a+1)$ is undefined, where G is the underlying graph of the network and a is any arbitrary positive integer.

A9'. There exist upper bounds, t, on the number of faulty processors and l, on the number of faulty links in the system.

The SR protocol, modified as follows, will be called the MSR protocol.

The sending processor sends the message to all its neighbors, and every relay

processor broadcasts the message to all its neighbors, except the one from which the message was received, in accordance with A6.

The receiving processor executes the following algorithm T (a constant, which is a maximum message delay) units of time after the sender sent the message have elapsed.

Algorithm: receive(MessagesReceived, Value);

If there exists a set of nodes N', $|N'| = t$, and a set of links E', $|E'| = l$, such that all messages in MessagesReceived that did not pass through $N' \cup E'$ contain the same value v, then $Value \leftarrow v$;

else $Value \leftarrow \emptyset$.

Lemma 1: Under assumptions A1', A2-A8, and A9', a receiver can get the correct value sent from a correct sender by executing procedure receive().

Proof. A8 implies that all messages that pass through a route which has no faulty processors or links will arrive within some fixed time interval T from the time the sender sent the message.

A6 implies that if a message was routed through a route that has at least one faulty processor or faulty link, the name of at least one faulty processor or faulty link in the route will appear in the message.

Under the assumptions, the existence of N' and E' such that the messages that did not pass through N' and E' contain the correct value sent from the correct sender is clear. Let N' be the set of faulty nodes and E' be the set of faulty links. Then

all messages that did not pass through $N' \cup E'$ are correct messages sent from the correct sender.

Suppose v is the value sent by a correct sender and a receiver gets $v' \neq v$. But this could only happen if all the routes for the messages which did not pass through $N' \cup E'$ contained at least one faulty processor or link. Let N'' be the set of faulty processors and E'' be the set of faulty links. Then $|N''| \leqslant t$ and $|E''| \leqslant l$ by A9'. Therefore, every message must go through $N' \cup E' \cup N'' \cup E''$, and $N' \cup E' \cup N'' \cup E''$ separates the sender and the receiver. However, $|N'| + |N''| \leqslant 2t$ and $|E'| + |E''| \leqslant 2l$ contradict A1'. □

By Lemma 1, Byzantine Agreement can be reached in the presence of up to t processor and l link failures if A1' is satisfied. We now show that if A1' is not satisfied, then two correct processors cannot communicate reliably under t processor and l link failures. This also implies that no Byzantine Agreement can be achieved.

> **Lemma 2:** Let u, v be two correct processors (nodes) of a network. Suppose that the fixed-pair connectivity function f does not satisfy (1) $f(G,u,v,2t) \geqslant 2l+1$, and there does not exist a positive integer a such that (2) $f(G,u,v,a) \geqslant 2l+1$ and $f(G,u,v,a+1)$ is undefined, where G is the underlying graph of the network. Then u and v cannot communicate reliably in the presence of t processor and l link Byzantine failures.

Proof. If neither (1) nor (2) is satisfied, then there exists a cut set of $2t$ nodes and $2l$ links the removal of which separates u and v. Let the $2t$ nodes of the cut set be $\{p_1, p_2, \cdots, p_{2t}\}$ and the $2l$ links in the cut set be $\{e_1, e_2, \cdots, e_{2l}\}$. Partition the cut set into two sets A and B such that $A = \{p_1, p_2, \cdots, p_t\} \cup \{e_1, e_2, \cdots, e_l\}$ and $B = \{p_{t+1}, p_{t+2}, \cdots, p_{2t}\} \cup \{e_{l+1}, e_{l+2}, \cdots, e_{2l}\}$. Suppose node u sends the value x to v.

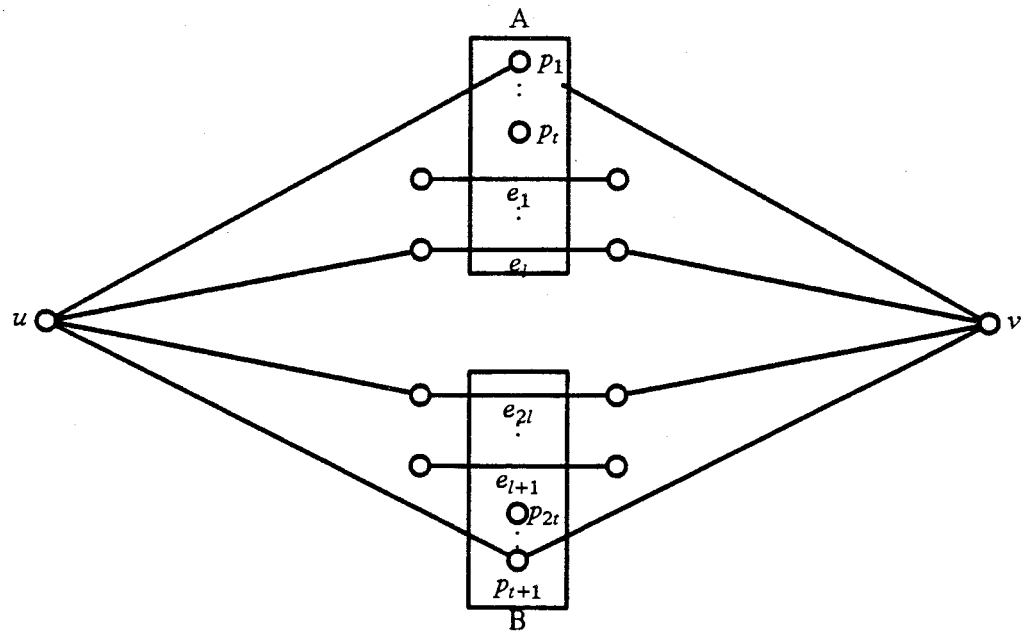

Figure 4-1: An Example of Unreliable Communication

Since there are t faulty nodes and l faulty links in the system, the elements in A could all be faulty. In relaying x to v, all the elements in A change x to x'. Since A and B are symmetric, and both could be faulty, v cannot identify which message is correct. □

> **Theorem 3:** Byzantine Agreement is achievable in a synchronous system in the presence of up to t processor and l link Byzantine failures if and only if $n \geq 3t + 1$ and the fixed-pair connectivity function f for every pair of participating nodes u and v satisfies either (1) $f(G,u,v,2t) \geq 2l + 1$ or (2) $f(G,u,v,a) \geq 2l + 1$ and $f(G,u,v,a+1)$ is undefined.

Proof. This theorem follows from the fact that Byzantine Agreement can be reached if and only if $n \geq 3t + 1$ [20] in the absence of link failure and from the above two lemmas. □

Chapter 5

The Connectivity Function

5.1. Introduction

One reliability measure of a network is its connectedness: That is, the number of nodes, or links, or both, which must be removed from a network in disconnecting it. The *connectivity function* represents, in addition to the removal of a nodes from a network N, the number of links which must be removed from N to make N disconnected. When $a = 0$, its value is the edge connectivity of N, and when a is the vertex connectivity of N, its value is 0; thus, the connectivity function is a generalization of the edge and vertex connectivities.

This chapter discusses the connectivity function and is organized as follows. In Section 2, we introduce the graph-theoretic terminology and definitions needed in later sections. In Section 3, we describe some properties of the connectivity function. Section 4 deals with the problem of constructing graphs with given connectivity functions. Although the vertex and edge connectivities can be determined in polynomial time, it is proved in Section 5 that the problem of computing the fixed-pair connectivity function is *NP*-complete for general graphs; in addition, it is proved that the problem remains *NP*-complete for bipartite, chordal, and split graphs. In the weighted case, the problem remains *NP*-complete even for complete graphs. An

approximation problem for the connectivity function is also proved *NP*-hard. In Section 6, we present polynomial time algorithms for determining the fixed-pair connectivity function of certain graphs, including vertex-connectivity-bounded graphs, certain "regularly structured" graphs, such as complete graphs and *n-cubes*, and series-parallel graphs. In Section 7, we consider a path problem that is related to the connectivity function.

5.2. Terminology and Definitions

A *graph* $G=(V,E)$ consists of a finite non-empty *vertex set* $V=V(G)$ and an *edge multi-set* $E=E(G)$. An *edge* of G is an unordered pair $\{u,v\}$ of vertices such that $u,v \in V$ and $u \neq v$. If $e=\{u,v\} \in E(G)$, we say, variously, that e *joins* u and v, u and v are *adjacent*, e is *incident on* u and v, and u and v are *incident with* e. The *degree* of a vertex v is the number of edges incident on v. *Multiple edges* are two or more edges that join the same pair of vertices. A graph is a *multigraph* if multiple edges are allowed; otherwise it is *simple*.

Two simple graphs $G=(V,E)$ and $G'=(V',E')$ are *isomorphic* if there is a one-to-one mapping f from V onto V' that preserves adjacency, i.e., $\{u,v\} \in E$ if and only if $\{f(u),f(v)\} \in E'$. A *subdivision* of an edge $\{u,v\}$ of a graph G is an operation that replaces $\{u,v\}$ with a new vertex w and two edges, $\{u,w\}$ and $\{w,v\}$. Two graphs are *homeomorphic* if both can be obtained from the same graph by sequences of edge subdivisions.

Let $G=(V,E)$ be a simple graph. A *path* p in G is a sequence of vertices $v_1,v_2,\cdots,v_k$ such that $\{v_i,v_{i+1}\} \in E$ for $i \in \{1,\cdots,k-1\}$. Vertices v_1 and v_k are

called the *ends* of the path p. If u and v are the ends of a path, we call the path a (u,v)-path. A path is *simple* if all its vertices are distinct. Two (u,v)-paths are *vertex disjoint* if the two paths have no vertex in common except for u and v, and are *edge disjoint* if the two paths have no edge in common. A path $v_1,v_2,\cdots,v_k$ is called a *cycle* if $v_1 = v_k, k \geq 2$, and it has no repeated edges. A cycle is *simple* if the vertices $v_1,v_2,\cdots,v_{k-1}$ are distinct. The *length* of a path or a cycle p is the number of edges in p, i.e., $length(p) = k-1$. We sometimes refer to the vertex set $V(p) = \{v_i | i \in \{1,2,\cdots,k\}\}$ and the edge set $E(p) = \{\{v_i,v_{i+1}\} | i \in \{1,\cdots,k-1\}\}$ of a path or a cycle p.

A vertex v is *reachable* from another vertex u if there is a (u,v)-path. A graph G is *connected* if every vertex is reachable from every other vertex in G, and is *disconnected* if G has two vertices u and v such that v is not reachable from u. A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A *component* is a maximally connected subgraph. If $G' = (V', E')$ is a maximal subgraph of G with vertex set V', i.e., $E' = \{\{u,v\} | u,v \in V', \{u,v\} \in E\}$; we say G' is a subgraph of G *induced by* V'. For $v \in V, V' \subset V$ and $E' \subseteq E$, we use $G - v$ and $G - V'$ to denote subgraphs of G induced by $V - \{v\}$ and $V - V'$, respectively, and $G - E'$ denotes the subgraph $(V, E - E')$.

A set V' of vertices is a *vertex cut set* of a connected graph $G = (V, E)$ if $G - V'$ has more than one component; in this case the removal of the vertex set V' is said to *disconnect* G. The *edge cut set* E' of edges and the *mixed cut set* (A,B), where $A \subset V$ or $B \subseteq E$, are defined similarly.

Let s and t be two distinct vertices of a graph $G=(V,E)$. A vertex set V', where $s,t \notin V'$, is an *(s,t) vertex cut set* if the removal of V' from G separates s and t. The *(s,t) edge cut set* and the *(s,t) mixed cut set* (A,B), where $s,t \notin A, A \subset V$, and $B \subseteq E$, are defined similarly.

The *fixed-pair vertex connectivity* for two nonadjacent vertices s and t of G, denoted by $\kappa(G,s,t)$, is the smallest number of vertices, not including s and t, whose removal separates s and t. The *vertex connectivity* of G, denoted by $\kappa(G)$, is the smallest number of vertices whose removal disconnects G or results in a trivial graph (a single vertex); clearly it is equal to $min\{\kappa(G,s,t) | s,t \in V, \{s,t\} \notin E\}$ if $G \neq K_n$.

The *fixed-pair edge connectivity* between two vertices s and t of G, denoted by $\lambda(G,s,t)$, is the minimum number of edges whose removal separates s and t. The *edge connectivity* of G, denoted by $\lambda(G)$, is the minimum number of edges whose removal disconnects G; it is equal to $min\{\lambda(G,s,t) | s,t \in V, s \neq t\}$.

The *fixed-pair connectivity function* for two vertices s and t of G, denoted by $f(G,s,t,a)$, equals b if there exists some set of a vertices and b edges whose removal separates s and t and there is no set of $a-1$ vertices and b edges or of a vertices and $b-1$ edges with this property. Similarly, we define the *global connectivity function* $F(G,a)$: $F(G,a)=b$ if there exists some set of a vertices and b edges whose removal disconnects G and there is no set of $a-1$ vertices and b edges or of a vertices and $b-1$ edges with this property.

Since $F(G,0)=\lambda(G)$ and $F(G,\kappa(G))=0$ by definition, the connectivity function generalizes the concepts of edge and vertex connectivities.

Throughout this thesis we discuss only the connectivity functions of connected graphs. For convenience, from here on, until and unless otherwise stated, a graph, a path, and a cycle means a simple graph, a simple path, and a simple cycle, respectively.

5.3. Properties of Connectivity Function

In this section, we consider a graph $G=(V,E)$ containing two distinguished vertices s and t.

> **Lemma 1:** Let $G_v = G-v$, a be a positive integer, and b be a nonnegative integer. If $f(G_v,s,t,a) \geq b$ for all $v \in V-\{s,t\}$ such that $v \neq s,t$ then $f(G,s,t,a) > b$.

Proof. It is clear that $f(G,s,t,a) \geq b$. Suppose $f(G,s,t,a) = b$ and let (A,B) with $|A|=a$ and $|B|=b$ be a mixed cut-set which separates s and t in G. Then for some $v \in A$, $(A-\{v\},B)$ also separates s and t of G_v, but $|A-\{v\}|=a-1$ and $|B|=b$ imply that $f(G_v,s,t,a-1)=b$, contradicting the assumption. □

Let $G=(V,E)$ be a multigraph. We define a graph operation *edge contraction*, denoted by $ec(G,u,v)$, where u and v are in G and are adjacent, as follows: $ec(G,u,v)$ identifies u and v with all edges and adjacency preserved except for the edges between u and v, which are removed.

Note that the edge contraction operation is different from the *elementary contraction operation* which also removes the resulting duplicate edges, if any. The following lemma concerns the connectivity function of a graph after an edge contraction operation.

Lemma 2: Let v be a vertex adjacent to s such that $v \neq t$, and let $G' = ec(G,s,v)$. If there exists a mixed cut-set (A,B) that separates s and t in G', then (A,B) also separates s and t in G.

Proof. If the lemma were not true, then after deleting (A,B) from G, there would be a path P from s to t in $G-A-B$. Whether v is on P or not, P or $ec(P,s,v)$ is also a path from s to t in $G'-A-B$, a contradiction. □

The above lemma implies that if $f(G',s,t,a) = b'$ and $f(G,s,t,a) = b$, then $b' \geq b$.

Theorem 3: $f(G,s,t,a) \geq b$ if and only if condition 1 below holds and there exists a vertex v adjacent to s, where $v \neq t$, satisfying conditions 2 and 3.

1. $f(G_v,s,t,a-1) \geq b$,
2. $f(G',s,t,a) \geq b$, where $G'=ec(G,s,v)$, or there is no mixed cut-set of $a-1$ vertices and b edges or of a vertices and $b-1$ edges that separates s and t of G',
3. Let X^* be the set of all nonempty subsets of E', where E' is the set of all edges that are incident on both s and t. For each $X \in X^*$, $f(G'',s,t,a) \geq b-|X|$, where $G'' = G - X$.

Proof. The necessity is implied by a definition of connectivity function and Lemma 2. We now prove the sufficiency. If the theorem were not true, then there would exist a mixed cut set (A,B), such that $|A| = a$ and $|B| \leq b-1$, without violating the three conditions. By condition 3, no edge in B is incident to both s and v. There are two cases to consider: either v is in A or not. If v is in A, then $(A-v,B)$ separates s and t in G_v, contradicting condition 1. If v is not in A, we claim that (A,B) would also separate s and t in G', a contradiction to condition 2. Since v is adjacent to s and edge $\{s,v\}$ is not in B, so (A,B) separates s and v from t in G, and therefore it separates s from t in G'. □

5.4. Construction of Graphs with Given Connectivity Functions

Let a be a positive integer. It is shown in [3] that every decreasing function from $\{0,1,\ldots,a\}$ into the non-negative integers such that $F(a)=0$ is the connectivity function for some graph, but the construction of such a graph requires a large number of vertices.

In the case of a fixed number n of vertices, $F(j)=l$ means that every vertex must be of degree at least $j+l$, and so the number of edges $m \geq \lceil \frac{(j+l)n}{2} \rceil$. If $m = \lceil \frac{(j+l)n}{2} \rceil$ for some positive integer j, then a graph $G=(V,E)$ with $\kappa(G) = j+l = \lfloor 2m/n \rfloor$, $|V|=n$, and $|E|=m$ can be constructed [4] and this implies that $F(j) \geq l$ for G.

Let S be a sequence of ordered pairs $(i,F(i))$ where i and $F(i)$ are non-negative integers such that $F(i)$ is a strictly decreasing function of i. Take a pair $(i', F(i'))$ in S such that $i'+F(i')$ is maximum, and let $a = i' + F(i')$. Construct a graph $G=(V,E)$ with its global connectivity function F' satisfying $F'(G,i') = F(G,i')$. Since $\kappa(G'-V') \geq a - |V'|$ for any subset $V' \subset V$, therefore, we have $F'(j) \geq a-j$. On the other hand, by the definition of the vertex connectivity, $j+F(j) \leq a$ holds. It follows that $F'(j) \geq F(j)$ for each pair $(j,F(j))$ in S.

As shown in the next section, the evaluation of a fixed-pair connectivity function is, in general, *NP*-hard. Therefore, given an arbitrary graph G containing vertices s, t and a pair (i,j) of positive integers, the problem of constructing a graph G' from G such that $f(G',s,t,i) = j$ by adding the minimum number of edges to G is also *NP*-hard.

5.5. *NP*-Completeness

Here we show that computing a fixed-pair connectivity function is, in general, *NP*-hard. We pose this problem as a decision problem as follows:

Fixed-pair Connectivity Function, *FPC*.

INSTANCE: Graph $G=(V,E)$, $s,t \in V$, an integer a, $0 < a \leqslant |V|-2$, and positive integer b.

QUESTION: Does there exist a subset $V' \subset V$ with $s,t \notin V'$ and $|V'|=a$, such that $\lambda(G-V',s,t) \leqslant b$, i.e., the edge connectivity between s and t in $G-V'$ is less than or equal to b?

We first prove the *NP*-completeness of a similar problem, *MMF*, viz., finding the minimum maximum flow over all subnetworks induced by deleting a nodes from a network.

Min-Max Flow of Subnetworks, *MMF*.

INSTANCE: Network W with node set N and link set L, $s,t \in N$, and positive integers a and b.

QUESTION: Does there exist a set of nodes $N' \subseteq N$ with $|N'|=a$, and $s,t \notin N'$ such that, when the nodes in N' are removed from W, the flow from s to t is at most b?

Theorem 4: MMF is *NP*-complete.

Proof. Since a maximum flow of a network can be found in polynomial time for each choice of N', this problem is in *NP*.

To complete the proof, we now show that the "minimum cut into equal-sized subsets" problem, which is known to be *NP*-complete [15], is polynomially reducible to MMF.

Minimum Cut into Equal-Sized Subsets, *MCE*.

INSTANCE: Graph $G=(V,E)$, two distinguished vertices $s,t \in V$, and a positive integer M.

QUESTION: Is there a partition $V=P_1 \cup P_2$, $P_1 \cap P_2=\emptyset$, such that $|P_1|=|P_2|$, $s \in P_1, t \in P_2$, and $|\{\{u,v\} \in E \mid u \in P_1, v \in P_2\}| \leqslant M$?

Theorem 5: MCE α MMF.

Proof. Let $G=(V,E)$ and M be an instance of MCE, where $V=\{v_1,...,v_{n-2},s,t\}$, and without loss of generality let $n=|V|$ be even.

Transformation:

$$flow = b = M + (n/2-1)\,n + (n/2-1)\,n^2,$$

$$a = n/2 - 1.$$

Network $W=(N,L)$

$$N = \{s',t'\} \cup V \cup C,$$
$$\text{where } C = \{c_1,...,c_{n-2}\},$$
$$L = \cup \{\{s',s\},\{t,t'\}\}$$
$$\cup \{\{s',c_i\} \mid 0 < i < n-1\}$$
$$\cup \{\{c_i,v_i\} \mid 0 < i < n-1\}$$
$$\cup \{\{v_i,t'\} \mid 0 < i < n-1\}$$
$$\cup \{\{c_i,t'\} \mid 0 < i < n-1\},$$

with capacities of the links as follows:

$cap(e) = 1$ for $e \in E$,
$cap(e) = n^2 + 4n$ for $e = \{s',c_i\}$,
$cap(e) = 2n$ for $e = \{c_i,v_i\}$ or $\{s,s'\}$,
$cap(e) = n$ for $e = \{v_i,t'\}$ or $\{t',t\}$,
$cap(e) = n^2$ for $e = \{c_i,t'\}$.

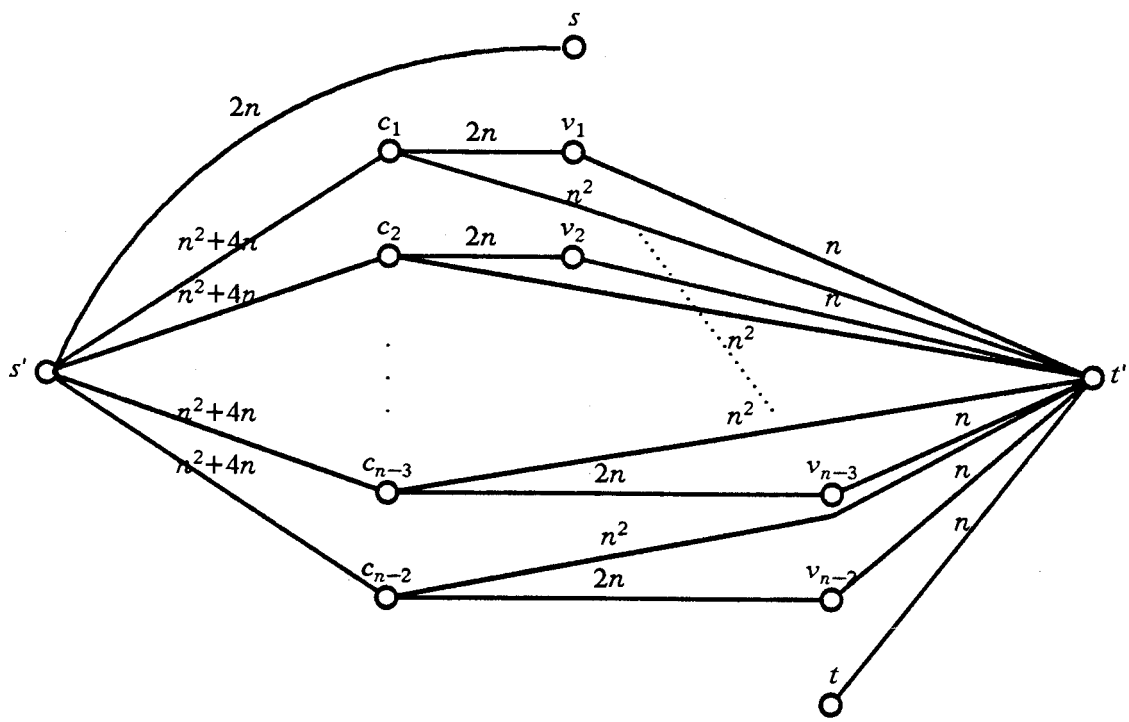

Figure 5-1: Construction of $W=(N,L)$

Since we add only n nodes in constructing W from graph G, and the capacity for each link is of $O(n^2)$, the transformation is polynomial.

Suppose in a given instance of MCE, M', $M' \leq M$, the vertex set V is partitioned into two equal-sized disjoint sets V_1 and V_2, where $V_1 = \{s,v_1, \cdots ,v_h\}$, $h = \frac{n}{2} - 1$, and $V_2 = V - V_1$. Define a cut is the set of edges that has one end in V_1 and the other end in V_2. The size of a cut is the number of edges in the cut. We can then construct a flow from s' to t' of at most b units after deleting $a = \frac{n}{2} - 1$ nodes in W.

Let $N' = \{c_{n/2},\ldots,c_{n-2}\}$, and $P_1 = C - N' \cup V_1 \cup \{s'\}$ and $P_2 = V_2 \cup \{t'\}$ be a partition of the vertices of $W-N'$.

After removing N' from W, the edge cut separating V_1 from V_2 has size M'. Similarly the edge cuts which separate V_1 from t', and $C-N'$ from t' have sizes $(n/2-1)n$ and $(n/2-1)n^2$, respectively.

Therefore, the size of any edge cut separating P_1 from P_2 and so the flow from s' to t' is at most $M' + (\frac{n}{2}-1)n + (\frac{n}{2}-1)n^2 \leqslant b$ by the *max-flow-min-cut-theorem*.

Suppose there is a min-max flow of b' units from s' to t' in $W-N'$, where $b' \leqslant b$ and $|N'| = \frac{n}{2} - 1 = a$.

We claim that $N' \subset C$. This is obvious because deleting any c_i will delete at least n^2 units of flow, while deleting any other node not in C will remove at most n units of flow.

Let $N' = \{c_{p(n/2)}, \cdots, c_{p(n-2)}\}$, where p is a permutation on $\{n/2, n/2+1, \cdots, n-2\}$.

From the construction of W, if the flow is maximum, the links (c_i,t'), (v_i,t'), $c_i \notin N'$, can all have flow equal to their capacities since the capacities of the edges (s',c_i) are sufficiently high.

But the total flow from $c_i \notin N'$ to t', $c_i \notin N'$, is then $(\frac{n}{2}-1)n^2$ and the total flow from v_i to t', $c_i \notin N'$, is $(\frac{n}{2}-1)n^2$. Therefore, the remaining M' units of flow, $M' = b' - (n/2-1)n - (n/2-1)n^2 \leqslant M$, must come from $V_2 = \{v_i \mid c_i \in N'\}$. Since V_2 is

adjacent only to $V_1 = V - V_2$ and t', and again because of the sufficiently large capacities of (s',c_i) and (c_i,v_i), the flows in the edges (v_i,v_j), where $c_i \notin N'$ and $c_j \in N'$, are equal to their capacities. Therefore, the edges between V_1 and V_2 are part of a cut in $W - N'$ and are also a cut of size at most M in the instance of MCE. □

Intuitively, deleting c_i from W implies that we put vertex v_i of graph G (an instance of MCE) into P_2. For simplicity the links in the construction of W are undirected. Since an undirected link $\{u,v\}$ is equivalent to two directed links (u,v) and (v,u) in network flow problems, our results hold also for the directed case.

Theorem 6: The FPC problem is *NP*-complete.

Proof. To show that the problem is in *NP*, we nondeterministically delete a vertices and compare the edge connectivity of the resulting graph with b. Since the edge connectivity can be found in polynomial time, this nondeterministic algorithm has polynomial time complexity.

We now prove that it is *NP*-hard by transforming MMF to the fixed-pair connectivity function problem.

Transformation:

We transform the network $W=(N,L)$ of an instance of MMF into a graph $G = (V, E)$, where $V = N$. For each link $e' = \{u,v\}$ in L with $cap(e') = c$, introduce c edges between u and v in G. To avoid multiple edges, insert a vertex in the middle of each edge

thus introduced. Since the instances of MMF, which were derived from MCE, have edge capacity of $O(n^2)$, therefore the transformation is polynomial.

Since both the edge connectivity and the maximum flow are equal to the minimum cut, the *NP*-hardness of the FPC follows. □

5.5.1. Bipartite Graphs

A graph $G=(V,E)$ is *bipartite* if its vertex set V can be partitioned into two sets U and W such that every edge of G has one end in U and the other end in W, or, equivalently, G is bipartite if every cycle of G has even length [17].

We can form a bipartite graph from a graph G by doubling the length of every cycle of G, for example, by replacing each edge of G with a path of length two. Intuitively, such expansion will preserve the value of the fixed-pair vertex connectivity, the fixed-pair edge connectivity, and the fixed-pair connectivity function as well. We formalize these ideas in the proof of the next theorem.

Theorem 7: The fixed-pair connectivity function problem, FPC, is *NP*-complete for bipartite graphs.

Proof. It is easy to show that FPC belongs to *NP*: Nondeterministically remove a vertices and determine the edge connectivity of the resulting graph. We now show that any instance of FPC for a general graph can be transformed into an instance of FPC for a bipartite graph. Let $G=(V,E)$, two distinguished vertices s and t, and two nonnegative integers a and b be an instance of FPC. We transform G into a bipartite graph $G'=(V',E')$ such that $f(G,s,t,a)=f(G',s,t,a)$. Since an edge joining s

and t must be in any (s,t) mixed cut set, we can assume without lost of generality that no edge is incident with both s and t. More formally, define G' as follows:

$$G' = (V', E'),$$
$$V' = V \cup V'',$$
$$V'' = \{w_e | e \in E\},$$
$$E' = \{\{u,w_e\},\{w_e,v\} | e = \{u,v\} \in E\}.$$

Clearly, the transformation is polynomial. Each $e \in E'$ has one end in V and the other end in V'', hence G' is bipartite. It is also easy to see that $P = s,v(1),v(2),\cdots,v(k),t$ is a path of G if and only if $P' = s,w_{\{s,v(1)\}},v(1),w_{\{v(1),v(2)\}},\cdots,w_{v(k),t},t$ is a path of G'. We call P' the *expanded version* of P. Let (A,B) be an (s,t) mixed cut set of G and B' be the set of edges formed from B by replacing each edge $e = \{u,v\} \in B$ with either $\{u,w_e\}$ or $\{w_e,v\}$, but not both. We thus have $|B'| = |B|$. If an (s,t)-path P of G contains some element of A or B, then the expanded version of P in G' contains some element of A or B'. Therefore, (A,B') is an (s,t) mixed cut set of G' with $|B'| = |B|$. It follows that $f(G,s,t,a) \geq f(G',s,t,a)$. To prove $f(G,s,t,a) \leq f(G',s,t,a)$, let (A',B') be an (s,t) mixed cut set of G', and P be an (s,t)-path of G such that P' is the expanded version of P. If P' contains some $u \in V$ or $w_{\{u,v\}} \in V'$, then P must also contain u or both u and v, respectively. So let A be a vertex set formed from A' by replacing each vertex $b_{\{u,v\}} \in V' \cap A'$ with u if $u \neq s,t$, and B be an edge set formed from B' by replacing each edge $\{w_e,v\} \in E'$ with e. Then, (A,B) is clearly an (s,t) mixed cut set of G with $|A| \leq |A'|$ and $|B| \leq |B'|$, and this completes the proof. □

5.5.2. Chordal and Split Graphs

A graph is *chordal* if for every cycle of length greater than three there is a *chord*, where a chord is an edge joining two non-consecutive vertices of the cycle. A graph is a *split graph* if both it and its complement are chordal. Split graphs form a proper subclass of chordal graphs.

We show here that FPC is *NP*-complete for the split graphs.

Theorem 8: The fixed-pair connectivity function problem, FPC, is *NP*-complete for split graphs which are also edge graphs.

Proof. It is easy to show that FPC belongs to *NP*. Let $G=(V,E)$, two distinguished vertices s and t, and two nonnegative integers a and b be an instance of FPC. We transform $G=(V,E)$ into a split graph $G'=(V',E')$ which is also an edge graph such that $f(G,s,t,a) \leqslant b$ if and only if $f(G',s,t,a) \leqslant n^3 b + n^2$, where $n=|V|$. We triangulate G by adding n^3-1 copies of a path of length two to each adjacent pair of vertices of G and an edge to each non-adjacent pair of distinct vertices of G. Intuitively, the n^3-1 copies of a path added to each adjacent pair of vertices of G increase the capacity of an edge of G so much that the subsequently added edge to each non-adjacent pair of G with capacity one is negligible with respect to the amount of flow it can flow. More formally G' is defined as follows:

$$
\begin{aligned}
&G'=(V',E')\\
&V'=V \cup V'',\\
&V''=\{w_{e,i} \mid e \in E, 1 \leqslant i < n^3\},\\
&E'=E \cup E_1 \cup E_2,\\
&E_1=\{\{u,v\} \mid u,v \in V, u \neq v, \{u,v\} \notin E\}\\
&E_2=\{\{u,w_{e,i}\}, \{w_{e,i},v\} \mid e=\{u,v\}, e \in E, 1 \leqslant i < n^3\}.
\end{aligned}
$$

It is clear that the transformation is polynomial. First we show $f(G,s,t,a) \leqslant b$ if and only if $f(G',s,t,a) \leqslant n^3b + n^2$. It is easy to see that $s,v1,\cdots,vk,t$ is a path of G if and only if $P' = \{ s,v1,\cdots,vk,t,\ s,w_{\{s,v1\},1},v1,\cdots,vk,w_{\{vk,t\},1},t,\ s,w_{\{s,v1\},2},v1,\cdots,vk,w_{\{vk,t\},2},t,\ \cdots,\ s,w_{\{s,v1\},n^3-1},v1,\cdots,vk,w_{\{vk,t\},n^3-1},t\}$ is a set of paths of $G'-E''$. Since $|E''| < n^2$ and $f(G'-E'',s,t,a) = n^3 f(G,s,t,a)$, we get $f(G',s,t,a) \leqslant n^3b + n^2$. To prove $f(G,s,t,a) \leqslant b$, let (A,B) be a minimal (s,t) mixed cut set of G' with $|A| = a$ and $|B| \leqslant n^3b + n^2$. Similar to the *NP*-completeness proof for bipartite graph, by removing $B \cap E''$ from the mixed cut set (A,B), where $|B \cap E''| < n^2$, we get $f(G,s,t,a) \leqslant b$.

Finally, we prove by contradiction that G' is a split graph. Let C be a chordless cycle of G' with length greater than three. In G', the vertex set V forms a clique and V'' is an independent set. Therefore, C has at most two vertices from V and has at least two vertices from V''. Let $v \in V'' \cap V(C)$ and $a,b \in V(C)$ be adjacent to v. Since $v \in V''$, its adjacent vertices are all in V; but $a,b \in V$ implies that there is a chord $\{a,b\}$ in C, a contradiction, hence G' is chordal. Similarly, it is easy to see that the complement of G', $\bar{G}'$, with V being an independent set and V'' being a clique, is also chordal. It follows that G' is split. □

5.5.3. Weighted version

In the definition of the connectivity function, we assumed the edges of a graph had equal weight. In some instances, for example, the problem of separating two nodes of a network with minimum cost, it is useful to allow edges to have different weights. This consideration gives rise to the weighted version of the connectivity function.

We now prove that the weighted version is *NP*-complete even for complete graphs. Let $G=(V,E)$ be a graph and w be a mapping from E into **N**. The weighted fixed-pair connectivity function and the corresponding decision problem are defined as follows:

> Given a graph $G=(V,E)$ with two distinguished vertices $s,t \in V$ and edge weight $w: E \rightarrow \mathbf{N}$, the *weighted fixed-pair connectivity function* $wf(G,s,t,a)$ equals b if there exists an (s,t) mixed cut set (V',E') with $|V'|=a$ and $\Sigma_{e \in E'} w(e)=b$ and there is no (s,t) mixed cut set (V'',E'') with $|V''|=a-1$ and $\Sigma_{e \in E''} w(e)=b$, or with $|V'|=a$ and $\Sigma_{e \in E''} w(e) < b$.

Weighted Fixed-Pair Connectivity Function Problem, *WFPC.*

INSTANCE: Graph $G=(V,E)$ with two distinguished vertices s and t, a function $w: E \rightarrow \mathbf{N}$, and two nonnegative integers a and b.

QUESTION: Does there exist a subset $V' \subset V$ with $s,t \notin V'$ and $|V'|=a$, such that there is an (s,t) edge cut set E' in $G-V'$ with $\Sigma_{e \in E'} w(e) \leqslant b$?

> **Theorem 9:** The weighted fixed-pair connectivity function problem, *WFPC*, is *NP*-complete for complete graphs.

Proof. Based on the same argument as in the *NP*-completeness proof of FPC, we have $WFPC \in NP$. Let $G=(V,E)$, two distinguished vertices s and t, and two nonnegative integers a and b be an instance of FPC. We transform G into a weighted complete graph $G'=(V',E')$ such that $f(G,s,t,a) \leqslant b$ if and only if $wf(G',s,t,a) \leqslant n^3 b + n^2$, where $n=|V|$. Define G' as follows:

$G'=(V',E')$,
$V'=V$,

$$E' = E \cup \{\{u,v\} | u,v \in V, u \neq v, \{u,v\} \notin E\},$$
$$w(e) = \begin{cases} n^3 \text{ if } e \in E, \\ 1 \text{ otherwise.} \end{cases}$$

Clearly, the transformation is polynomial. It is straightforward to verify that G' is complete and $f(G,s,t,a) \leqslant b$ if and only if $wf(G',s,t,a) \leqslant n^3 b + n^2$. □

Several classes of graphs contain the complete graphs as members; we list some of them in the following corollary.

Corollary 9.1: The weighted fixed-pair connectivity function problem, *WFPC*, is *NP*-complete for edge graphs, interval graphs, strongly chordal graphs, and cographs.

5.5.4. Approximation

Since the evaluation of the fixed-pair connectivity function is *NP*-hard as shown in Theorem 6, looking for an optimal cut set is, in general, computationally intractable. We might want to know if there exists a polynomial approximation algorithm that can find a vertex set $C \subset V - \{s,t\}$ of size a such that $|\lambda(G-C,s,t) - f(G,s,t,a)| < k$, where k is a fixed positive integer. We show here that finding such an approximate solution is as difficult as finding an optimal solution.

Theorem 10: Let $G = (V,E)$ be an arbitrary graph containing vertices s and t, and let a, k be an arbitrary pair of nonnegative integers. If $P \neq NP$, then no polynomial-time algorithm A can find a vertex set C, with $|C| = a$, such that $|\lambda(G-C,s,t) - f(G,s,t,a)| < k$.

Proof. For a given graph $G = (V,E)$, let $f(G,s,t,a) = b$ and C be the solution computed by A. First transform $G = (V,E)$ into $G' = (V',E')$ as follows. Intuitively, we replace each original edge with k edges and place a new vertex in the middle of each new edge.

$V' = V \cup \{w_{e,i} | 1 \leq i \leq k, e \in E\}$,

$E' = \{\{u, w_{\{u,v\},i}\}, \{w_{\{u,v\},i}, v\} | 1 \leq i \leq k, \{u,v\} \in E\}$.

The graph G' is clearly bipartite and since the capacity of each edge is a multiple of k, by a proof analogous to that for Theorem 7, it is easy to show that $f(G',s,t,a) = kb$. Now let C, where $|C| = a$, be the solution obtained by applying A to G'. If $C \subset V$, then $\lambda(G'-C,s,t)$ is a multiple of k, so let $\lambda(G'-C,s,t) = kj$ for some positive integer j. We have $|kj - f(G',s,t,a)| < k$, i.e., $|kj - kb| < k$; this implies that $kj = kb$ and, therefore, C is an optimal solution for G as well. If $C \not\subset V$, let $v' \in C-V$ and replace v' with one of its adjacent vertices which are in V. Let C' be the set C with every vertex not in V replaced by a vertex in V as above. It is clear that $\lambda(G-C',s,t) \leq \lambda(G-C,s,t)$ from the transformation, and therefore C' is also an optimal solution for both G and G'. □

5.6. Polynomially Solvable Cases

Presented in this section are certain classes of graphs, the connectivity functions of which can be computed in polynomial time. The following Theorem will be frequently referred to in later sections.

> **Theorem 11:** [Menger's Theorem] The minimum number of edges (vertices) separating two (nonadjacent) vertices s and t is the maximum number of edge (vertex) disjoint (s,t) paths.

There are well known polynomial time algorithms for computing the edge connectivity and vertex connectivity, which are special cases of the connectivity function. Many efficient fixed-pair edge connectivity and fixed-pair vertex

connectivity algorithms employ some maximum network flow algorithms to find the maximum number of disjoint paths connecting two vertices. For the edge and vertex connectivities, a common approach is to take the minimum value of the fixed-pair connectivities over a set of vertex pairs. With this approach, higher efficiency is normally achieved by minimizing the number of calls to a maximum flow algorithm that computes a fixed-pair connectivity.

By virtue of the strictly decreasing property of the connectivity function, we have the following lemma.

Lemma 12: Let $G=(V,E)$ be a graph with two distinguished vertices $s,t \in V$. If the weighted fixed-pair connectivity function $wf(G,s,t,0)=\kappa(G,s,t)$, then $wf(G,s,t,a)=\kappa(G,s,t)-a$ for all a such that $0 < a \leqslant \kappa(G,s,t)$.

Proof. By definition, $wf(G,s,t,\kappa(G,s,t))=0$ and $wf(G,s,t,a-1) > wf(G,s,t,a)$ for all a, $0 \leqslant a \leqslant \kappa(G,s,t)$. It follows that $wf(G,s,t,a)=\kappa(G,s,t)-a$, where $0 < a \leqslant \kappa(G,s,t)$.

□

Corollary 12.1: If $\lambda(G,s,t)=\kappa(G,s,t)$, then $f(G,s,t,a)=\kappa(G,s,t)-a$ for all a, $0 \leqslant a \leqslant \kappa(G,s,t)$. Similarly, if $\lambda(G)=\kappa(G)$, then $F(G,a)=\Gamma-a$ for all a, $0 \leqslant a \leqslant \kappa(G)$.

Lemma 12 and its corollary are useful in recognizing the connectivity function for graphs with certain "regular structures". Take a complete graph $G=K_n$ with n vertices, for example. Its vertex connectivity $\kappa(G)=\lambda(G)=n-1$, and thus the global connectivity function is obtained immediately by Corollary 12.1 . As every pair of distinct vertices of a complete graph are adjacent, κ(G,s,t) is undefined. Since every (s,t) mixed cut set must include the edge $\{s,t\}$, this edge can be taken out from G

before applying Corollary 12.1. We thus obtain $\lambda(G-\{\{s,t\}\},s,t) = \kappa(G-\{\{s,t\}\},s,t) = n-2$ and $f(G-\{\{s,t\}\},s,t,a) = n-2-a$ for all a, $0 \leqslant a \leqslant n-2$. After edge $\{s,t\}$ is restored to $G-\{\{s,t\}\}$, we get $f(G,s,t,a) = n-1-a$ for all a, $0 \leqslant a \leqslant n-2$. In contrast with the result here, the weighted fixed-pair connectivity function problem is *NP*-complete for class of complete graphs as shown in Theorem 9.

5.6.1. Graphs with Bounded Vertex Connectivity

Let $G=(V,E)$ be a graph with two distinguished vertices $s,t \in V$, having the fixed-pair connectivity function satisfying $f(G,s,t,a) = b$ for a pair of integers a, b. Let (A,B) be an (s,t) mixed cut set for G with $|A| = a$ and $|B| = b$. As $f(G,s,t,a) = min_{V' \subseteq V}\{\lambda(G-V',s,t) | s,t \notin V', |V'| = a\}$, one way to calculate $f(G,s,t,a)$ is to systematically check $\lambda(G',s,t)$ for every induced subgraphs G' of G with a fewer vertices and with $s,t \in V(G')$. If $\kappa(G,s,t)$ is bounded by a constant k, then $f(G,s,t,a)$ is undefined for $a > k$. Therefore, for any a, in calculating $f(G,s,t,a)$, the number of such G' whose edge connectivity we need to check is bounded by $\binom{n}{k}$, where $n = |V|$. The following lemma immediately follows.

> **Lemma 13:** Let $G=(V,E)$ be a graph with two distinguished vertices $s,t \in V$, the weighted fixed-pair connectivity function $wf(G,s,t,a)$ is computable in polynomial time for graphs whose $\kappa(G,s,t)$ is bounded by a constant.

> **Corollary 13.1:** The weighted fixed-pair connectivity function is computable in polynomial time for grid graphs, wheel graphs, and degree bounded graphs.

As every planar graph $G=(V,E)$ with $|V| \geqslant 4$ has at least four vertices of degree not exceeding five (Cor. 11.1 (e) in [17]), we have $\kappa(G) \leqslant 5$. Therefore the global

connectivity function $F(G,a)$ is undefined for all integers a satisfying $a \geqslant j$ for some integer $j \leqslant 5$. The weighted or unweighted global connectivity function for a planar graph can thus be computed in polynomial time even by an exhaustive search.

5.6.2. Graphs n-cube Q_n

For a positive integer n, the *n-cube*, denoted by Q_n, is a graph consisting of 2^n vertices, each of which is labelled $v_n v_{n-1} \ldots v_1$, where $v_i \in \{0,1\}$. Two vertices of Q_n are adjacent if and only if their labels differ in exactly one digit.

Theorem 14: For all positive integers n, the vertex connectivity, $\kappa(Q_n)$, of the n-cube Q_n, is equal to n.

Proof. Since each vertex of Q_n has exactly n neighbors, $\kappa(Q_n) \leqslant n$ holds. We prove $\kappa(Q_n) \geqslant n$ by induction on n, the *dimension* of the n-cube.

As $Q_1 = K_2$ by definition, so $\kappa(Q_1) = 1$, and the theorem is true for $n = 1$. Assuming the theorem is true for all $n \leqslant r$ for some $r \geqslant 1$, we consider the case where $n = r+1$.

Let $n = r + 1$. We assume $\kappa(Q_n) < n$ and derive a contradiction.

If $\kappa(Q_n) < n$, then there is vertex cut set V' with $|V'| = n-1$. After V' is removed from Q_n, $Q_n - V'$ has more than one component. Assume $Q_n - V'$ has $x \geqslant 2$ components, and let $S_1, S_2, \cdots, S_x$ be the components of $Q_n - V'$. We claim that $x = 2$ and if $u = a_n, \cdots, a_1 \in S_1$ and $v = b_n, \cdots, b_1 \in S_2$, then $a_k \neq b_k$ for all k, $1 \leqslant k \leqslant n$. To prove the above claim consider any two distinct components S_i and S_j. We first prove that the first digit a_n of any vertex $u = a_n, \cdots, a_1 \in S_1$ must be

different from the first digit b_n of any vertex $v = b_n, \cdots, b_1 \in S_2$. Assume to the contrary that $a_n = b_n = 0$. The case where $a_n = b_n = 1$ can be treated similarly. Let V^0 consists of all the vertices of Q_n with their first digits equal to 0. Observe that the subgraph induced by V^0 is isomorphic to Q_{n-1}. We note the following:

(i) The cut set V' must have at least $n-1$ vertices with their first digits equal to 0. Otherwise, let $U' = V' \cap V^0$, where $|U'| < n-1$. For each k, $1 \leq k \leq x$, let V_k^0 be the vertices of S_k belonging to V^0. Then $V^0 = V_1^0 \cup \cdots V_x^0 \cup U'$. Furthermore, S_i and S_j are separate in the graph $Q_n - U'$. But then U', with $|U'| < n-1$, separates $u \in V_i^0$ from $v \in V_j^0$ in the graph induced by V^0, which is a Q_{n-1}, a contradiction to the induction hypothesis $\kappa(Q_{n-1}) = n-1$.

(ii) The cut set V' must include at least one vertex with the first digit equal to 1. By the induction hypothesis, there are $n-1$ vertex disjoint paths from $1,a_{n-1}, \cdots, a_1$ to $1,b_{n-1}, \cdots, b_1$ such that all vertices in the paths have their first digits equal to 1. Since u is adjacent to $1,a_{n-1}, \cdots, a_1$ and v is adjacent to $1,b_{n-1}, \cdots, b_1$, V' is not a vertex cut set unless it has at least one vertex with the first digit equals to 1.

From (i) and (ii) above, it follows that $|V'| \geq n$, a contradiction to our original assumption, thus we must conclude $a_n \neq b_n$. Since the above argument is valid for any a_k, $1 \leq k \leq n$, it follows that $a_k \neq b_k$ for all k, $1 \leq k \leq n$, and there can be only 2 components in $Q_n - V'$, i.e., $x = 2$.

For a vertex u of Q_n, there is only one vertex in Q_n with its i^{th} digit different

from the i^{th} digit of u for all i, $1 \leqslant i \leqslant n$. Therefore, the number of vertices in S_i, where $i = 1,2$, is one. But then the equality $|S_1| + |S_2| + |V'| = |V(Q_n)|$ does not hold, since $|S_1| + |S_2| + |V'| = n + 1$, whereas $|V(Q_n)| = 2^n$, a contradiction. Thus the theorem follows. □

Corollary 14.1. For all positive integers n, the edge connectivity $\lambda(Q_n)$, the fixed-pair vertex connectivity $\kappa(Q_n,s,t)$ for two non-adjacent vertices s, t of Q_n, and the fixed-pair edge connectivity $\lambda(Q_n,u,v)$ for any two distinct vertices u, v of Q_n are all equal to n.

Proof. Since the vertex connectivity of an n-cube cannot exceed the edge connectivity, the fixed-pair vertex connectivity, and the fixed-pair edge connectivity of the n-cube, and since each vertex of an n-cube has exactly n neighbors, none of the above connectivities can exceed n. □

Since $\kappa(Q_n) = \lambda(Q_n)$, and $\kappa(Q_n,u,v) = \lambda(Q_n,u,v)$ for two non-adjacent vertices u and v of Q_n, the value of the global and fixed-pair connectivity functions for an *n-cube* can be computed by Lemma 12.

5.6.3. Series-Parallel Graphs

Two distinct edges of a graph are said to be in *series* if they are incident on the same vertex ("middle vertex") of degree two and are *parallel* if they join the same pair of distinct vertices.

A *series-parallel multigraph* is defined recursively as follows:

A graph consisting of two vertices joined by an edge is series-parallel. G is series-parallel if a graph obtained from G by replacing a pair of series edges together with the middle vertex or a pair of parallel edges of G by an edge is series-parallel. □

A *series-parallel simple graph* is defined as a series-parallel multigraph without multiple edges between any pair of vertices.

We now define the class of graphs that are "suppressible to an edge". We follow the definitions in [18]; for a characterization, please see [18].

Let $G = (V, E)$ be a simple graph. A vertex $v \in V$ is *suppressible* if v has degree 2. If v is suppressible, an *elementary suppression* $\sim v$ applied to G results in a graph as follows: $G \sim v = (V - v, E \cup \{u,w\})$, where u and w are the two vertices adjacent to v; to keep the graph simple, edge $\{u,w\}$ is only added if u and w are not already adjacent. A *suppression* S of G is a sequence of elementary suppressions. The result of applying S to G is denoted by $S(G)$. In this case we say G is *suppressible* to $S(G)$. A *total suppression* of G is a suppression of G that cannot be extended.

It is obvious that if a simple graph is series-parallel, then the graph is suppressible to an edge, and vice versa. We introduce suppressible graphs merely for the purpose of proving the correctness of some algorithms. From here on, the terms series-parallel graph and graph that is suppressible to an edge will be used interchangeably.

The algorithm we present next computes the weighted fixed pair connectivity function $wf(G,s,t,a)$ for a graph G which is suppressible to edge $\{s,t\}$. Since an edge joining s and t must be in any (s,t) mixed cut set, we can assume without loss of generality that the input graph G has no edge joining s and t.

The following procedures will be called by SPConFn():

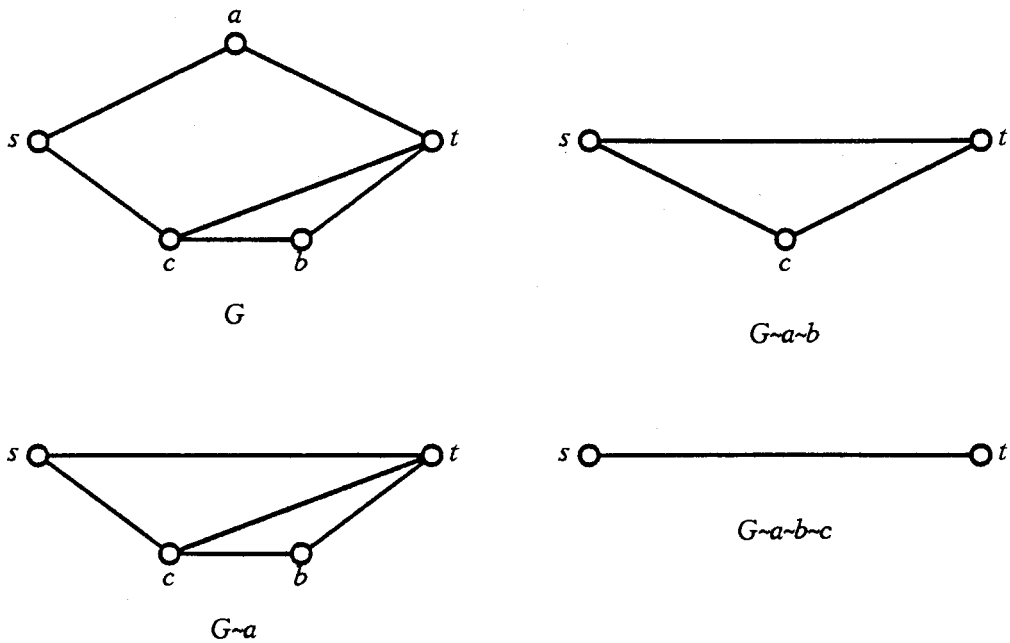

Figure 5-2: A Suppression $S = \sim a \sim b \sim c$ applied on G.

Component(G) : **return**(C, c);
Finds all the components in graph G, and returns the number, c, and the set $C = \{G_1, G_2, \cdots, G_c\}$, of all components of G.

MaxFlow(G, w, s, t) : **return**(f,*flow*);
Finds the maximum flow f from s to t and the corresponding flow pattern. w is a weight function from edge set of G into **N**. *flow* is a flow function on the edge set of graph G, i.e., *flow*(e) gives the amount of flow that passes through edge e in the computed maximum flow pattern.

Procedure SPConFn(G, s, t, a);

INPUT: Graph $G=(V,E)$ with two distinguished vertices $s,t \in V$.
a weight function w from E into **N**, and an integer $a \in$ **N**.

OUTPUT: *wf(G, s, t, a).*

begin

step 1: /* Find all the components in graph $G-\{s,t\}$
by calling Component() */
(*Comp*, *c*) := Component($G-\{s,t\}$);

step 2: (maxflow, flow) := MaxFlow(*G*, *w*, *s*, *t*);

step 3: **for** $i \in \{1, \cdots, c\}$

$\text{mflow}(G_i) := \Sigma_{e \in E_i'} \text{flow}(e)$,

where E_i' is the set of all edges $\{s,v\}$ such that $v \in V(G_i)$;

step 4: Let c' be the number of components G_i such that $\text{mflow}(G_i)>0$;

if $a > c'$ **then**

return (Message:"*wf(G,s,t,a)* is undefined because $a > \kappa(G,s,t)$")

else

/* This is to block as much flow as possible by removing *a* vertices.
The remaining flow will be blocked by removing a minimum weight
edge cut set. */

begin

select "*a*" largest mflow()'s, and
let mflow(1), mflow(2), ..., mflow(*a*)
be the "*a*" largest mflow()'s;

return(b':=maxflow $- \Sigma_{i=1}^{a}$ mflow(G_i))

end

end.

We now prove the correctness of algorithm SPConFn(). Let $\overline{G}_i$ be the subgraph of G induced by vertex set $V(G_i) \cup \{s,t\}$ and $I=\{1, \cdots, c'\}$. Assume without loss of generality that

$$\text{mflow}(G_1) \geq \text{mflow}(G_2) \geq \cdots \geq \text{mflow}(G_{c'}) > 0$$

Fact 1. Let S be a suppression of a graph G, and u, v, where $u \neq v$, be vertices of both G and $S(G)$. There is a (u,v)-path in G if and only if there is a (u,v)-path in $S(G)$.

Lemma 15: Let s and t be two distinguished vertices of a simple graph G and G be suppressible to an edge $\{s,t\}$. Then for any two vertex-disjoint (s,t)-paths P and Q of G, there is no path R, disjoint from P and Q, such that R connects a vertex in $V(P)-\{s,t\}$ with a vertex in $V(Q)-\{s,t\}$.

Proof. Let $u \in V(P)-\{s,t\}$, $v \in V(Q)-\{s,t\}$, and suppose there is a path $u,w_1,w_2,\cdots,w_i,v$ connecting u and v, such that $(V(P) \cup V(Q)) \cap \{w_1,w_2,\cdots,w_i\} = \emptyset$. Since G is suppressible to $\{s,t\}$, let S be a suppression such that $S(G)$ is edge $\{s,t\}$, and S' and S'' be subsequences of S such that $S = S' \sim uS''$. Without loss of generality, assume $\sim u$ precedes $\sim v$ in S. Since $s,t,u,v \in V(S'(G))$, by Fact 1, there are two (s,t)-paths, $(s,a_1,\cdots,u,\cdots,a_l,t)$, $(s,b_1,\cdots,v,\cdots,b_m,t)$, and a (u,v)-path $(u,w'_1,\cdots,w'_j,v)$ in $S'(G)$, and these three paths are clearly vertex-disjoint, except at s and t. Thus, u has degree at least three in $S'(G)$, and $S'(G)$ is not suppressible by $\sim uS''$, a contradiction to the assumption that G is suppressible by S. □

Fact 2. For all $i,j \in \{1,\cdots,c\}$ such that $i \neq j$, G_i and G_j are disjoint.

Proof. G_i and G_j are two different components. □

Fact 3. Each (s,t)-path is in exactly one $\overline{G}_i$.

Proof. Let P be any (s,t)-path. Then it contains one vertex in some G_i. Since the vertices of P are connected, therefore, P is in $\overline{G}_i$, and by Fact 2, P cannot be in two different $\overline{G}_i$'s. □

Fact 4. For each $i \in I$, the vertex connectivity $\kappa(\overline{G}_i,s,t)$ is at most one.

Proof. If $\kappa(\overline{G}_i,s,t) > 1$ for some i, there are two vertex disjoint (s,t)-paths p, q in $\overline{G}$. Since we assume that s and t are not adjacent, there is one vertex in $V(p)$ and one vertex in $V(q)$ both of which belong to G_i. By Lemma 15, p and q will be separated by removing s and t from $\overline{G}_i$, a contradiction that G_i is a component of $G-\{s,t\}$. □

Theorem 16: Given valid arguments, G, s, t, a, algorithm SPConFn() correctly computes $wf(G,s,t,a)$.

Proof. Let b' be the value returned by SPConFn(). The proof consists of two parts:

1. $wf(G,s,t,a) \leqslant b'$.

We need to prove that there is a vertex set $V' \subseteq V(G)-\{s,t\}$ with $|V'|=a$ and an edge set E' with total weight b' such that the removal of V' and E' from G disconnects s from t. From Fact 3, any (s,t)-path P is in $\overline{G}_i$ for some $i \in I$. If $1 \leqslant i \leqslant a$, then by Fact 4, there exists a vertex set V', $|V'|=a$, such that P must pass through some $v \in V'$. If $a < i \leqslant c'$, on the other hand, P must pass through some minimum weight edge cut set $E_i' \subseteq E(G_i)$, where $\Sigma_{e \in E_i'} \text{flow}(e) = \text{mflow}(G_i)$, by the *Maximum-Flow Minimum-Cut Theorem*. By removing V' and $\{e \in E_i' \mid a < i \leqslant c'\}$ from G, all paths from s to t are disconnected.

2. $wf(G,s,t,a) \geqslant b'$.

Let (V'',E'') be any (s,t) mixed cut set with $|V''|=a$. By Fact 4 there exists a vertex $v \in V(G_i)$ such that any (s,t)-path of $\overline{G}_i$ must pass through v. Therefore, we can

assume that $V'' \cap V(G_i) \leqslant 1$ for all $i \in I$. Then, since $\text{mflow}(G_1) \geqslant \text{mflow}(G_2) \geqslant , \cdots , \geqslant \text{mflow}(G_a)$, the maximum flow F from s to t after removing V'' from G satisfies

$$F = \text{maxflow} - \sum_{V'' \cap V(G_i)=1} \text{mflow}(G_{p(i)}) \geqslant \text{maxflow} - \sum_{i=1}^{a} \text{mflow}(G_i) = b'$$

, and by the *Maximum-Flow Minimum-Cut Theorem*, we have $\Sigma_{e \in E''} w(e) \geqslant F \geqslant b'$. □

Theorem 17: Algorithm SpConFn() for an input graph $G = (V, E)$, where $|V| = n$, runs in time $O(n \log n)$.

Proof. First note that the series-parallel graphs are a sub-class of planar graphs, and therefore $|E|$ is of $O(n)$. Step 1 finds the components of G-$\{s,t\}$, and can be done in $O(|E|) = O(n)$ by using the depth-first search. Step 2 computes a maximum flow from s to t for graph G; for (s, t)-planar graphs, i.e., graphs that can be drawn in the plane with no edges crossing each other such that vertices s and t are on the same face, a maximum flow from s to t can be found in $O(n \log n)$ [19]. Since if a graph is suppresible to edge $\{s, t\}$, it is also (s, t)-planar. Step 2 is of $O(n \log n)$. Step 3 checks each edge incident on s to determine whether the other end of the edge is in G_i and performs the summation; thus step 3 is of $O(|E|) = O(n)$. Step 4 finds the "a" largest mflow()'s, which can be done in $O(n)$ [2]. Therefore, the time complexity of algorithm SPConFn() is of $O(n \log n)$. □

Algorithm SPConFn() evaluates the weighted fixed-pair connectivity function $wf(G,s,t,a)$ for a graph G that is suppressible to edge $\{s,t\}$. Next, we consider $wf(G,s,t,a)$ for G that is not suppressible to edge $\{s,t\}$ but is suppressible to another edge $\{s',t'\}$. We state some results in [18] which are relevant to Lemma 20 that follows.

Lemma 18: For any two total suppressions S and S' of G, $S(G)$ and $S'(G)$ are isomorphic.

Lemma 19: If G is suppressible to an edge, then there is no subgraph homeomorphic to K_4, the complete graph on four vertices.

Lemma 20: Let $G=(V,E)$ be a graph with two distinguished vertices $s,t \in V$. Suppose G is not suppressible to edge $\{s,t\}$, but is suppressible to an edge $e \neq \{s,t\}$, and, in addition, every vertex $u \in V$ is on some (s,t)-path. Then, we have the following.

a) If there are two vertex disjoint (s,t)-paths P and Q in G, where $|V(P)| \geq 3$ and $|V(Q)| \geq 3$, such that $u \in V(P)-\{s,t\}$ and $v \in V(Q)-\{s,t\}$, then there exists a (u,v)-path R such that $V(R)-\{u,v\}$ is disjoint from $V(P)$ and $V(Q)$.

b) Either $wf(G,s,t,2)=0$ or $wf(G,s,t,2)$ is undefined.

Proof.

a) We prove by induction on the number of vertices. Assertion a) is clearly true for $|V| \leq 4$. Assume it is true for $|V| \leq k$, for some $k \geq 4$. We now prove that it is also true for $|V|=k+1$. Since G is suppressible to an edge and G has at least 5 vertices, some vertex $v \in V$ is suppressible. If $v \neq s,t$, then $G \sim v$ is not suppressible to $\{s,t\}$, and by the induction hypothesis, the lemma is true. There are two possible remaining cases:

Case 1. Either s or t, and not both, is suppressible.

Assume without loss of generality that s is the only suppressible vertex, and let vertices b and c be adjacent to s. If b and c are not adjacent to each other, then b and c have the same degree in $G \sim s$ as in G and so do all other vertices, and,

therefore, $G \sim s$ is non-suppressible. But $G \sim s$ has at least 4 vertices and obviously is not an edge; thus b and c must be adjacent to each other. Since there are two vertex disjoint (s,t)-paths P and Q, one must pass through b and the other through c. Paths P, Q have length greater than one, and thus $b,c \neq t$, but the path $R = (b,c)$ connects P and Q.

Case 2. Both s and t are suppressible.

Assume vertices b and c are adjacent to s, and vertices d and e are adjacent to t. If b and c or d and e are adjacent to each other, Assertion a) immediately follows as in case 1. If not, the degrees of b, c, d, e and all other vertices in $G \sim s \sim t$ remain the same as in G, so $G \sim s \sim t$ is non-suppressible. But $G \sim s \sim t$ has at least three vertices, i.e., it is not an edge, a contradiction to our assumption that G is suppressible to an edge.

b) If Assertion b) was not true, then there would be three vertex disjoint (s,t)-paths. Let P and Q be two vertex disjoint (s,t)-paths with length greater than one. From a), there would be a (u,v)-path R, with $u \in V(P)-\{s,t\}$, $v \in V(Q)-\{s,t\}$, such that $V(R)-\{u,v\}$ is disjoint from $V(P)$ and $V(Q)$. But then G would have a subgraph homeomorphic to K_4, a contradiction to Lemma 19. □

Notice that a vertex which is not on any (s,t)-path does not affect the connectivities $\kappa(s,t)$, $\lambda(s,t)$ or $f(G,s,t,a)$. To evaluate any type of connectivity between s and t, we can therefore assume without loss of generality that every vertex is on some (s,t)-path. If a graph is suppressible to edge $\{s,t\}$, we apply algorithm SPConFn(). If

a graph is suppressible to an edge e, but not to $\{s,t\}$, Lemma 20 tells us that $f(G,s,t,a)$ is only defined for $a \leqslant 2$. In this case, even an exhaustive algorithm would run in polynomial time.

By Theorem 16, algorithm SPConFn() is applicable to series-parallel simple graphs. Consider a series-parallel multigraph. Let S be a set of multiple edges joining two vertices u and v. Observe that if any $e \in S$ is in a minimal cut set, then so are all $e \in S$. Thus, we treat the multiple edges joining u and v as one edge $\{u,v\}$ with edge weight equal to $|E'|$. Let G' be the graph thus obtained from a given graph G. It is easy to see that $f(G,s,t,a) = wf(G',s,t,a)$. To apply the method for simple graphs to multigraphs, we first replace multiple edges $\{u,v\}$ by a single edge with weight $w(\{u,v\})$, representing the number of edges joining u and v. We then solve the weighted version of the connectivity function.

5.7. A Path Problem

Let $G = (V, E)$ be a graph with two distinguished vertices $s,t \in V$, $V' \subseteq V$, $E' \subseteq E$, and P be a set of (s,t)-paths. An (s,t)-path p is said to be a (V',E')-*avoiding* path if no vertex of p is in V' and no edge of p is in E'. Define a predicate Q on P as follows: $Q(P,(V',E')) = True$ if and only if P contains a (V',E')-avoiding path.

Given that the fixed-pair connectivity function $f(G,s,t,a) > b$, where a, b are positive integers, the question is: For a given positive integer k, does G have a set P of (s,t)-paths with $|P| \leqslant k$, such that P contains a (V',E')-avoiding path for all $V' \subseteq V-\{s,t\}$ and $E' \subseteq E$ with $|V'| \leqslant a$ and $|E'| \leqslant b$? Furthermore, what is the minimum value of k such that the above question has the affirmative answer for all graphs G with $f(G,s,t,a) > b$?

We have not been able to answer these questions; nevertheless, we have the following lemma, which relates Q on a graph G with Q on subgraphs of G.

> **Lemma 21:** Let $G=(V,E)$ be a graph with two distinguished vertices $s,t \in V$, a, b be positive integers, and P_v be a set of (s,t)-paths in $G-v$, where $v \in V-\{s,t\}$. If, for each $v \in V-\{s,t\}$, $Q(P_v,(V',E'))=True$ for all $V' \subseteq V-\{s,t,v\}$ and $E' \subseteq E-\{e \in E \mid e \text{ is incident on } v\}$ with $|V'| \leqslant a-1$ and $|E'| \leqslant b$, then for the path set $P=\bigcup_{v \in V-\{s,t\}} P_v$, we have $Q(P,(V',E'))=True$ for all $V' \subseteq V-\{s,t\}$ and $E' \subseteq E$ with $|V'|=a$ and $|E'|=b$.

Proof. We prove by contradiction. Let $V'' \subseteq V-\{s,t\}$ and $E'' \subseteq E$ with $|V''|=a$ and $|E''|=b$ such that P contains no (V'',E'')-avoiding path. For any $v \in V''$, the path set P_v has no path containing v. Therefore, $Q(P,(V'',E''))=False$ implies that $Q(P_v,(V''-\{v\},E''))=False$. Since $|V''-\{v\}|=a-1$, we have a contradiction. □

5.8. Summary

Table 5-0 gives a summary of the complexities of computing the fixed-pair connectivity function, FPC, and the weighted fixed-pair connectivity function, WFPC, for the graphs we have studied in this chapter. NPC means *NP*-complete, P means polynomially solvable, "?" means the problem is still unsolved, n is the number of vertices, m is the number of edges, and k is a positive constant.

Class of Graphs	FPC	WFPC
Bipartite	NPC	NPC
Chordal	NPC	NPC
Split	NPC	NPC
Edge	?	NPC
Strongly chordal	?	NPC
Interval	?	NPC
Cograph	?	NPC
Complete	P	NPC
Vertex connectivity bounded by k	$O(n^{k+1}m \log n)$	$O(n^{k+1}m \log n)$
n-cube	P	?
Series-parallel	$O(n \log n)$	$O(n \log n)$
Approximation $\lambda(G-V',s,t)$ - OPT $< k$	*NP*-hard	*NP*-hard

Table 5-1: Complexity of FPC and WFPC.

Chapter 6

Conclusion

We have presented a new (d^2+2d)-resilient Byzantine Agreement protocol which terminates in the minimum number of rounds of message exchanges and uses $O((3d)^{2d+6})$ messages, when the number of processors $n \geq 9d^2 + 6d+1$. Other currently known t-resilient agreement protocols that terminate in $t+1$ rounds need to exchange $O(n^{t+1})$ messages when $n \leq t^2+3t+5$. To compare those protocols against our protocol, let $d^2+2d=t$. For large d, we have $d \sim \sqrt{t}$ and $d \sim \sqrt{n}/3$. Therefore, in terms of t and n, the number of messages required in our protocol is $O(n^{\sqrt{t}+3})$. Our protocol also allows two distant processors in two different groups to communicate with each other in about $\sqrt{t}$ rounds of message exchanges, in contrast to some other protocols which require every pair of processors to communicate in each of the $t+1$ rounds.

To investigate the resilience of network to failures, we have introduced a measure that is more precise than considering a link failure as the failure of one of the link's end nodes. For example, if a network with $3t+1$ processors is completely connected, the fixed-pair connectivity function for every two distinct nodes u and v satisfies $f(G,u,v,2t)=t$; thus it provides us with the information that the network can tolerate t processor and $t/2-1$ link failures. By considering a link failure as a processor failure, the total

number of processor and link failures cannot exceed t.

One drawback of our protocol for reliable communication, which is described in Chapter 4, is that the number of messages it uses is exponential in the number of nodes of the network. If the network is large, it is quite inefficient. An open problem is thus to find an efficient (in terms of the number of messages used) protocol to tolerate t processor and l link failures.

As an open problem we have stated the path problem in Chapter 5. It seems that in order to find an efficient protocol to handle t processor and l link failures under the minimal sufficient condition, the path problem must be solved first. We conjecture that the answer to the path problem is $k = (a + 1)(b + 1)$.

If the conjecture is true, then there is hope for finding an efficient protocol that tolerates up to t processor and l link failures. Otherwise, if k is exponential in the number of vertices of a graph, it is unlikely that there exists an efficient protocol that can tolerate up to t processor and l link failures under the minimal sufficient condition in a general setting.

References

[1] Attiya, C., Dolev, D. and Gil, J.
Asynchronous Byzantine Consensus.
Proc. 3rd ACM Symp. on PODC , 1984.

[2] Aho, A., Hopcroft, J., and Ullman J.
The Design and Analysis of Computer Algorithms.
Addison-Wesley Publishing Co., 1974.

[3] Beineke, L. W., and Harary, F.,
The Connectivity Function of a Graph.
Mathematika 14 :pp.197-202, 1967.

[4] Boesch, F., Harary, F., and Kabell, J.,
Graphs as Models of Communication Network Vulnerability: Connectivity and Persistence.
Networks, Vol.11 :pp.57-63, 1981.

[5] Coan, B. and Dwork C.
Simultaneity is Harder than Agreement.
Proc. 5th Symposium on Reliability in Distributed Software and Database Systems , 1986.

[6] Dolev, D.
The Byzantine Generals Strike Again.
J. Algorithms , 1982.

[7] Dolev, D.
Unanimity in an Unknown and Unreliable Environment.
Proc. 22nd Annual IEEE Symp. on Foundations of Computer Science :pp.159-168, 1981.

[8] Dolev, D., Dwork, C., and Stockmeyer, L.,
On the Minimal Synchronism Needed for Distributed Consensus.
Proc. 24th Symp. on Foundations of Computer Science , 1983.

[9] Dolev, D., Fischer, M.J., Fowler, R., Lynch, N.A. and Strong, H.R.
Efficient Byzantine Agreement without Authentication.
Information and Control 3 , 1983.

[10] Dolev, D. and Reischuk, R.
Bounds on Information Exchange for Byzantine Agreement.
J. ACM , 1985.

[11] Dolev, D., Reischuk, R. and Strong, H.R.
Eventual is Earlier than Immediate.
Proc. 23rd IEEE Symp. on Foundations of Computer Science :pp.196-203, 1982.

[12] Dolev, D. and Strong, H.R.
Authenticated Algorithms for Byzantine Agreement.
SIAM J. Computing 12 , 1983.

[13] Fischer, M. and Lynch, N.
A Lower Bound for the Time to Assure Interactive Consistency.
Info. Proc. Lett. 14, 4 , 1982.

[14] Fischer, M., Lynch, N.A. and Paterson, M.
Impossibility of Distributed Consensus with One Faulty Process.
J. ACM , 1985.

[15] Garey, M. R., and Johnson, D. S.
Some Simplified NP-Complete Graph Problems.
Theoretical Computer Science 1 :pp.237-267, 1976.

[16] Garcia-Molina, H., Pittelli, F., and Davidson, S.
Applications of Byzantine Agreement in Database Systems.
ACM Transactions on Database Systems, Vol.11, No.1 , 1986.

[17] Harary, F.
Graph Theory.
Addison-Wesley, 1972.

[18] Harary, F., Krarup, J., and Schwenk, A.
Graphs Suppressible to an Edge.
Canad. Math. Bull. Vol. 15(2) :201-204, 1972.

[19] Itai, A. and Shiloach, Y.
Maximum Flow in Planar Networks.
SIAM J. on Computing, Vol.8 :pp.135-150, 1979.

[20] Pease, M., Shostak, R. and Lamport, L.
Reaching Agreement in the Presence of Faults.
J. ACM 27 :pp.228-234, 1980.

[21] Rivest, R.L., Shamir, A. and Adleman, L.
A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.
Communications of the ACM :pp.120-126, 1978.

[22] Srikath, T. K. and Toueg, S.
Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms. *Tech. Rep. 84-623, Dept. of Computer Sci., Cornell Univ.* , 1984.