ABSTRACT

Title of dissertation: ROBUST NETWORK TRUST ESTABLISHMENT FOR COLLABORATIVE APPLICATIONS AND PROTOCOLS

Georgios E. Theodorakopoulos
Doctor of Philosophy, 2007

Dissertation directed by: Professor John S. Baras
Department of Electrical and Computer Engineering

In networks without centralized control (e.g. ad-hoc or peer-to-peer networks) the users cannot always be assumed to follow the protocol that they are supposed to. They will cooperate in the operation of the network to the extent that they achieve their own personal objectives. The decision to cooperate depends on the trust relations that users develop for each other through repeated interactions. Users who have not interacted directly with each other can use direct trust relations, generated by others, in a transitive way as a type of recommendation. Network operation and trust generation can be affected by malicious users, who have different objectives, and against whom any proposed solution needs to be robust.

We model the generation of trust relations using repeated games of incomplete information to capture the repetitive operation of the network, as well as the lack of information of each user about the others' objectives. We find equilibria that provide solutions for the legitimate users against which the malicious users cannot improve their gains. The transitive computation of trust is modeled using semiring

operators. This algebraic model allows us to generalize various trust computation algorithms. More importantly, we find the maximum distortion that a malicious user can cause to the trust computation by changing the reported trust value of a trust relation.

ROBUST NETWORK TRUST ESTABLISHMENT FOR COLLABORATIVE APPLICATIONS AND PROTOCOLS

by

Georgios E. Theodorakopoulos

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:
Professor John S. Baras, Chair/Advisor
Professor Virgil D. Gligor
Professor Carlos A. Berenstein
Professor Nuno Martins
Professor Gang Qu
Professor Michel Cukier

Dedication

To my family, Garyfallia, Efthymios, and Kelly. To my Teachers, Yiorgos Hatzis, Yiorgos Kefalas, and Popi Kanellou.

Acknowledgments

My advisor, Prof. John S. Baras, was the single most influential person in the course of my graduate studies. Were it not for him, I would not even have come to the University of Maryland, let alone stay and do research. I want to thank him for his deep mathematical insight, limitless energy and enthusiasm with which he guided my work. Prof. George V. Moustakides was also a great source of help and encouragement, especially in the crucial period leading up to my Ph.D. Research Proposal Examination. Finally, I want to thank the administrative staff of ISR and the ECE department, and, in particular, Althia Kirlew and Kim Edwards for their efficiency in isolating us students from the countless administrivia.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Networks are built with the expectation that the entities participating in them will follow a predefined protocol. The protocol determines the behavior of the network entities and it is a way to share common resources to achieve a network-wide benefit. However, in networks without centralized control, the entities cannot be forced to follow the protocol. The entities are (or are controlled by) users with personal objectives that will not coincide, in general, with the network-wide ones. Since each user's benefit depends on how the other members of the network behave, it is useful for him to form an expectation about what the others will do.

We define trust as the expectation of the behavior of others, e.g., other users in a network. As an example, consider that the routing protocol in a communication network has found shortest path routes that the packets are supposed to follow. Will the other users agree to forward your packets, or will they drop them? In a peer-to-peer, or a wireless ad-hoc network each user is assumed to be a potentially selfish, or even malicious, individual who cannot be counted on to follow any protocol.

The notion of trust, as a network security rather than a philosophical concept, will for our purposes correspond to a set of relations among entities that participate in a protocol [21]. These relations are based on the evidence generated by the previous interactions of entities within a protocol. In general, if the interactions have

been faithful to the protocol, then trust will "accumulate" between these entities. Exactly how trust is computed depends on the particular protocol (application). The application determines the exact semantics of trust, and the entity determines how the trust relation will be used in the ensuing steps of the protocol. Trust influences decisions like access control, choice of public keys, etc. It could be useful as a complement to a Public Key Infrastructure (PKI), where an entity would accept or reject a public key according to the trustworthiness of the entities that vouch for it (i.e. have signed a certificate for it) – this is the idea behind PGP's Web of Trust [61]. It can also be used for routing decisions: Instead of the shortest path, we could be looking for the most trusted path between two nodes (this has been already proposed in P2P networks [37]).

We will be focusing on infrastructureless networks, in the sense that no infrastructure should be required for the trust mechanisms to operate. It is, usually, in such networks that the users cannot be enforced to abide by the protocol. In particular, ad-hoc wireless networks will be a running example of the type of network that we are considering.

Ad Hoc networks are envisioned to have dynamic, sometimes rapidly-changing, multihop topologies which are composed of bandwidth-constrained wireless links. The nodes themselves form the network routing infrastructure in an ad hoc fashion [16]. Based on these characteristics, we are imposing the following three main constraints on our scheme:

First, there is no preestablished infrastructure. The computation process cannot rely on, e.g., a Trusted Third Party. There is no centralized Public Key Infras-

tructure, Certification Authorities, or Registration Authorities with elevated privileges.

Second, evidence is uncertain and incomplete. Uncertain, because it is generated by the users on the fly, without lengthy processes. Incomplete, because in the presence of adversaries we cannot assume that all friendly nodes will be reachable: the malicious users may have rendered a small or big part of the network unreachable.

Third, the trust metric cannot impose unrealistic communication/computation requirements. We are looking for a scheme that would lend itself to an efficient implementation. In other words, it should be as light as possible since it is a complement to the real operation of the network.

We will be distinguishing between two types of trust: direct and indirect. Direct trust is generated from the interactions between the network users. Indirect trust between two users is computed in a transitive fashion based on the available direct trust information. For instance, if there exist direct trust relations between users A and B, and also between users B and C, then A can use these two relations to compute an indirect trust relation between himself and C. This computed indirect trust relation can then be used for, e.g., deciding whether to provide to C access to some of A's resources.

A high level view our general model is shown in Figure 1.1.

Our review of the work on trust has uncovered two omissions in the literature. First, the lack of a model for how trust is generated based on the interactions of the users. Secondly, the lack of a common platform that can enable comparisons between

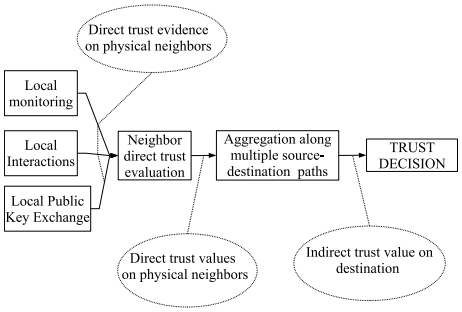

Figure 1.1: System operation

trust computation algorithms and protocols. Closely linked with this second point is that a comparison between two trust computation algorithms should definitely take into account the difficulty or otherwise that a malicious user would have in leading the computation astray.

Our contribution is the design and analysis of two formal models, one for direct trust generation, and one for indirect trust computation. Both models are expressive enough to allow reasonable flexibility for various specific applications and protocols. On top of their expressive power, both models enable us to analyze their properties. In particular, a major benefit is that we can find guarantees of robustness in the presence of malicious users. We can quantify and bound the maximum manipulation or damage that the attackers can achieve.

The direct trust generation model is based on game theory. The users are modeled to have certain actions at their disposal from which they can choose in order to maximize a payoff function. The objective function of each user is what distinguishes the different classes of users (selfish or malicious). Modeling selfish users as payoff maximizing players is very natural, and, in different forms than ours,

has been done before. However, the modeling of malicious users using game theory is a novelty of our work. The notion of Nash equilibrium, the usual solution concept in game theory, allows us to bound the payoff that the malicious users can achieve.

The indirect trust generation model is based on algebra, and, in particular, on the algebraic structure called semiring. Semirings capture the essence of the path-based nature of the indirect trust computation. The algebraic modeling viewpoint that we use is general enough to accommodate various algorithms, and to formulate axioms that all algorithms should satisfy. More importantly, within the framework of semirings we can quantify the effect of malicious users manipulating the trust relations reported by the other users.

A general requirement for our models to be applicable, as for all trust models, is that there has to be a way to associate actions with user names or user identities. In other words, we cannot have complete anonymity, because that would imply complete unaccountability. Note that this requirement does not mean that our models are susceptible to the Sybil attack [19]. Users can change identities, as long as they have *some* identity.

A more specific requirement, which applies to the game theoretic model of trust generation, is that we need network users to interact repeatedly. If the network is an ephemeral network, i.e., the user interactions are too short-lived, then there is not enough time to generate trust. In that case, the direct trust values would have to be externally imposed, e.g., in the form of digital certificates, or some kind of quick challenge-response protocol. Externally imposed trust is not naturally modeled by our game theoretic framework. However, it can certainly be used as input to the

algebra based indirect trust computation model.

This dissertation is functionally separated into two parts. The first part, consisting of chapters 2, 3, and 4, introduces and applies the game theoretic model for direct trust generation. Chapters 3 and 4 give two specific instantiations of the model, where the actions, payoffs, and equilibrium strategies of the users are specified. Chapter 3 models a scenario in resource constrained networks, where users want to forward their packets, but they also want to conserve their energy. Chapter 4 models a scenario where malicious users are trying to maximize their payoff before being discovered and isolated from the network. The second part, chapters 5, 6, and 7, is about indirect trust computation. In chapter 5, we present the algebraic operators used for the computation, and describe various algorithms in the literature from the algebraic point of view. Chapter 6 introduces our own model of semirings, together with two instantiations, and chapter 7 analyzes the robustness of semiring-based algorithms to attacks. Chapter 8 addresses practical issues for trust applications.

Chapter 2

Game Theoretic Modeling of Trust Generation

In this chapter, we describe the game theoretical framework that we will use for the generation of trust. We will model different types of users (good and malicious) by assigning them different payoffs that they try to maximize. We will use graphical games to emphasize the dependencies of a user's payoff to other users' actions. Incomplete information will be used to reflect the lack of knowledge of good users for the types of other users. Repeated games will be used to capture the duration of the network operation and the associated interactions between users, as well as the updating of the knowledge of the good users that comes from the accumulated history of observations of other users' actions. Trust will quantify the knowledge of the good users about the other users' types.

2.1 Introduction to Game Theory

In this section we give a short introduction to game theory. After describing the basic notions, we will focus on some definitions relevant to the specific areas of game theory that we will use in this dissertation. These areas are graphical games, incomplete information games, and repeated games. Highly recommended textbooks on game theory are: Osborne and Rubinstein [47], Myerson [44]. Very popular is also Fudenberg and Tirole [25].

2.1.1 Game Theory Fundamentals

Game theory studies situations where two or more individuals choose among actions so as to maximize their own welfare, knowing that everybody's actions affect everybody's welfare. Individuals are called *players*; a tuple of actions, one for each player, is called an *outcome*; and each player's welfare is represented with a real-valued *utility* function, which, as a general intuition, assigns larger numbers to outcomes that are preferred.

A simple way to describe a game is to use the *strategic form* representation. A game in strategic form is a tuple Γ of the form

$$\Gamma = (N, (A_i)_{i \in N}, (u_i)_{i \in N}) \tag{2.1}$$

where N is the non-empty set of players, which is usually identified with $\{1, \ldots, n\}$; A_i is the non-empty set of actions available to player i, which can be discrete or continuous; $u_i : \times_{j \in N} A_j \longrightarrow \mathbb{R}$ is the utility function of player i, which assigns a real number to every outcome.

Players in strategic form games choose their actions simultaneously, and then each one receives a payoff determined by the outcome that was realized and his utility function. The players are allowed to randomize among their actions, which turns the outcome into a random variable, called a *lottery* over outcomes. The received payoff is then also a random variable, but the players are assumed to receive its expected value. The variance of the payoff is never taken into account. The utility function is chosen so that its expected value reflects the players' preferences over any lotteries, not only over deterministic outcomes.

Game theory tries to find which outcomes are compatible with the assumptions about the users, that is, that the users are trying to maximize their utility function, and that they know that the outcome is affected by the other players who are also trying to maximize their own utility. All players have all the information about the game, including the utility functions of the other players. The most popular way to find such outcomes was proposed by Nash [46], and the corresponding outcomes are called Nash equilibria. A Nash equilibrium is a tuple of actions, one for each player, such that each action is a *best response* to the other players' actions: Given that all the other players will choose the action dictated by the equilibrium, no player can gain a larger payoff by deviating from his own equilibrium action. The action dictated by the equilibrium can also be a randomization between actions. In that case, we speak of a *mixed strategy* Nash equilibrium, as opposed to a *pure strategy* Nash equilibrium, which is when the equilibrium actions are deterministic.

As an example of a strategic game, consider the *Prisoner's Dilemma*. There are two players, each of which has two actions at his disposal. The payoffs are as shown in Figure 2.1. The name of this game comes from the following story [47], which also explains the payoffs: Two suspects are placed in different cells. If they both confess, each will be sentenced to three years in prison. If only one of them confesses, he will be freed, but his confession will be used to convict the other one to a four year imprisonment. If neither confesses, they will both receive a one year sentence due to some minor offense. The payoffs can be seen as the number of years that each player avoids spending in prison, out of a maximum of four. One player chooses a row and receives the left payoff in the appropriate box, the other one

	Don't Confess	Confess
Don't Confess	$3, 3$	$0, 4$
Confess	$4, 0$	$1, 1$

Figure 2.1: The Prisoner's Dilemma. A 2-player, 2-action game in strategic form. The only Nash equilibrium is (`Confess`, `Confess`), although both players can do simultaneously better by playing (`Don't Confess`, `Don't Confess`).

chooses a column and receives the payoff on the right in each box.

The paradox in this case is that whatever a player does, the other one is better off confessing. This implies that the only Nash equilibrium is (`Confess`, `Confess`), which gives a payoff of 1 to each player. However, they can both do better by playing (`Don't Confess`, `Don't Confess`). This is a pure strategy Nash equilibrium, and there are none other, either pure or mixed.

2.1.2 Graphical Games

Graphical games were introduced by M. Kearns, M. Littman, and S. Singh [31], as a more compact representation of strategic form games, where the payoff of a player depends only on a few other players' actions, presumably many fewer than the total number of players. The n-player graphical game is described by an undirected graph $G = (V, E)$ with $|V| = n$ vertices, one for each player. An edge $(i, j) \in E \subseteq V \times V$ means that i's payoff depends on j's action and vice versa. Although there are no self-loops in the graph, it is assumed that a player's payoff depends on his own action. Also, a trivial generalization of a graphical game is when

the graph is directed, with a directed edge (i, j) implying that j's payoff depends on i's action.

2.1.3 Incomplete Information Games

Incomplete information games are games where some players initially have private information about the game that other players do not know. The private information that a player has is called the *type* of the player. The type is a payoff function and a probability distribution on the possible types of other players. The set of all possible types of player i is denoted T_i, and a member of T_i is denoted by t_i. The probability distributions on the type of i that other players have need not be common: each player can believe different things about the type of player i.

In the case of two player games, we can distinguish two classes of games according to the information that each player receives. If neither player has any information about the other's type, we have the *symmetric information* case. If only one player has no information, but the other one has complete information about both players' types, then we have the *asymmetric information* case. In this latter case, the first player is called *uninformed*, and the other one is called *informed*.

The notion of the Nash equilibrium now changes to a Bayesian-Nash equilibrium, in which the uninformed player (or players) plays against a mixture, defined by his probability distribution, of the possible types of the other player (or players). In particular, such an equilibrium assigns to each type t_i of each player i a randomized or deterministic action that is optimal for i given t_i's probability distribution

Player 2 - A

Player 1 \ Player 2	T	S
T	$2, 1$	$0, 0$
S	$0, 0$	$1, 2$

Player 2 - B

Player 1 \ Player 2	T	S
T	$2, 0$	$0, 2$
S	$0, 1$	$1, 0$

Figure 2.2: Incomplete information game. Player 1 has one type, player 2 has two types (A and B). The (subjective) probability that player 1 places on player 2 being of type A is p ($1 - p$ for type B).

on the types of the other players and the actions that the equilibrium assigns to those types.

Consider the following two player incomplete information game with asymmetric information. Player 1 has only one possible type, but player 2 has two, say A and B. According to player 1, player 2 is of type A with probability p, and of type B with probability $1 - p$. Each player has two actions at his disposal: T and S. The payoff functions of player 1, and of the two types of player 2 are shown in Figure 2.2.

Each equilibrium for this game will be represented with a triplet (a, b, c), corresponding to the probabilities with which each of player 1, player 2-A, player 2-B will play T. We will use a similar notation, with a pair instead of a triplet, for the equilibria of the two constituent games with player 1's probability always shown first. The equilibria of the incomplete information game depend on the probability p. For example, it is expected that as p approaches 1, that is, player 1 is more and more certain that player 2 is of type A, the equilibria of the game will assign to player

1 the same actions that are assigned to him by the equilibria of the game between player 1 and player 2-A. There are three equilibria in the game player 1 vs player 2-A: $(1,1)$, $(0,0)$, and the mixed strategy equilibrium $(\frac{2}{3},\frac{1}{3})$. Conversely, when p approaches 0, we expect the equilibrium to be somehow related to the (unique) mixed strategy Nash equilibrium of the game between player 1 and player 2-B: $(\frac{1}{3},\frac{1}{3})$.

The Bayesian-Nash equilibria for the game of Figure 2.2 are shown below for the various value of p. We can see that there could be from one up to three equilibria, depending on the value of p. We can also verify that the equilibria for $p \longrightarrow 0,1$ resemble the equilibria of the constituent games.

Values of p	Equilibria			
$0 \leq p \leq \frac{1}{3}$	$(\frac{1}{3}, 0, \frac{1}{3(1-p)})$	xxx	xxx	xxx
$\frac{1}{3} \leq p \leq \frac{2}{3}$	$(\frac{1}{3}, 0, \frac{1}{3(1-p)})$	$(\frac{2}{3}, \frac{1}{3p}, 0)$	$(1,1,0)$	xxx
$\frac{2}{3} \leq p \leq 1$	xxx	$(\frac{2}{3}, \frac{1}{3p}, 0)$	$(1,1,0)$	$(0,0,1)$

2.1.4 Repeated Games

In repeated games, the same game (called the *stage game*), is repeatedly played in rounds, and the players remember what has happened in the past. So, their actions may depend on the accumulated history of past actions. The duration of the game may be finite or infinite. We will be focusing on the infinite case. The payoff of the repeated game is a function of the sequence of payoffs of the stage games. It is not usually chosen to be the sum, because many sequences of payoffs may have an infinite sum, so comparison would be meaningless. We will concentrate

on two repeated game payoff functions: The limit of means payoff

$$R_i = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} R_i^t, \tag{2.2}$$

and the δ-discounted payoff

$$R_i = (1-\delta) \sum_{t=1}^{\infty} \delta^{t-1} R_i^t, \tag{2.3}$$

where $\{R_i^t, t = 1, \ldots\}$ are the realized payoffs of player i at rounds $t = 1, \ldots,$ and the parameter $\delta, 0 < \delta < 1$ can be interpreted as the patience of a player. If δ is small, then only payoffs in the early rounds count; if δ is large, then payoffs of later rounds count, too. In the limit $\delta \longrightarrow 1$, the δ-discounted payoff becomes equal to the limit of means payoff.

2.1.5 Putting it all together

Our model in what follows will be a repeated graphical game with incomplete information. The users will be associated with nodes on a graph, which will be called the *game graph*, joined by edges denoting dependencies of payoffs on actions of other users. Each user will have a type, known to himself, but some users will be informed about everybody's types, while other users will only know their own. The game will be played repeatedly, and one of the main issues will be the acquisition of knowledge on the part of the uninformed players through the repeated interactions. This acquisition of knowledge will be reflected in the probabilities p that each user uses as estimates of the types of his neighbors on the game graph. Overall, the probabilities will affect what actions the players will play next, and the actions will affect how the probabilities will be updated.

2.2 Modeling Protocol Interactions of Network Users

We model networks consisting of users who try to maximize a personal benefit through their participation in the network. Associated with the network, there is a protocol that describes the available actions that each user has at his disposal. The protocol also defines a behavior that is expected of each user. However, there is no centralized control of the users' decisions, so each user is free to choose his action.

The payoff of each user reflects the benefit he derives from being part of the network, typically in terms of the services that the rest of the network provides to him through the operation of the protocol. In general, the level of service that a user receives may depend on many or all other users of the network; we model these dependencies as a graphical game. For example, if the protocol is about packet forwarding, then all users on the path from the source to the destination can affect the level of the service. However, the users may not want to follow the protocol, since that will entail certain responsibilities on their part. These responsibilities usually amount to resource consumption of some sort, such as energy consumption in the case of packet transmission in wireless networks. The cost associated with following the protocol is also reflected in the payoff, as a decrease in the derived benefit.

The users can have varying degrees of selfishness. Selfishness can be quantified as the level of service that a user desires from the network before he decides to follow the protocol himself. In the examples that we will analyze later, we will only consider users who are equally selfish, but we will point out how different degrees of

selfishness can be incorporated. Nevertheless, we will make an important distinction between selfish users, who care about the level of service they receive, and malicious users. More precisely, we can model malicious users as users whose payoff does not depend at all on the level of service that they receive. Our choice is to model them as playing a zero-sum game against good players, i.e., their objective is directly opposed to whatever the objective of the good users is.

In short, we distinguish different types of players (selfish or malicious) *through their different payoffs*, and we assume that all players choose their actions so as to maximize their payoff. This is in contrast to various game theoretic models in the literature that assign identical payoffs to all users, and then distinguish, e.g., malicious users through the different strategy that they follow. The problem with distinguishing users through assigning different strategies is that the strategy assigned to malicious users may not be a payoff maximizing strategy. This runs contrary to the rationality assumption for game theoretic players. Actually, if all players have the same payoff but follow different strategies, some players will be following suboptimal strategies, barring degenerate cases.

A crucial modeling decision is to make the payoff matrix of each user known only to himself. This reflects the lack of knowledge that users have for each other in a network without centralized control. To enhance the capabilities of the malicious users, we will assume that they know the payoffs of everybody. That is, they know who is malicious and who is only selfish. This is a game of incomplete information, and the information is asymmetric, since malicious users know the types of everyone, but each selfish user only knows his own payoff.

Since the network operates for a long time, the users choose actions repeatedly. We assume time is discrete, and users choose their actions simultaneously in rounds, which exactly follows the repeated games framework. At each round, users remember the past history of their own actions and of the actions of the users that affect their payoff. They can use this history to help them choose their action in the next round. Given the incomplete information assumption, repeated interactions can help increase the knowledge of non-malicious users regarding who is malicious and who is not. This increase in knowledge is quantified through the beliefs of users for each other's type.

In general, if we want a user i to form beliefs about other users based on their actions, these actions have to be observable by user i. Therefore, if the users who affect i's payoff are not among his one-hop network neighbors, then either there has to be a mechanism that can make observations "at a distance", or the model has to be extended to, e.g., imperfect observation games. We do not consider such extensions in this work.

2.2.1 Formal description of the model

The network is modeled as an undirected graph $G = (V, L)$, where each node in V corresponds to one user. An edge $(i, j) \in L$ means that there is a communication link between the users corresponding to nodes i and j. The set of neighbors of user i, denoted N_i, is the set of users j such that there exists an edge (i, j):

$$N_i = \{j \in V | (i, j) \in L\}. \tag{2.4}$$

The neighbors of user i are also called adjacent nodes to i, and the links that share a node are called adjacent links. Since the graph is undirected, the neighbor relationship is symmetric: $j \in N_i \Leftrightarrow i \in N_j$. The assumption for an undirected graph can be dropped, in order to model asymmetric links, but we believe the extension to be straightforward. We denote the set of Bad users by V_B, and the set of Good users by V_G. It holds that $V_B \cap V_G = \emptyset$ and $V_B \cup V_G = V$. The *type* $t_i \in \{G, B\}$ of a user i denotes whether he is Good or Bad. The Good users correspond to the selfish users of the previous section.

The graph of the game is identified with the graph of the network. Therefore, the payoff of a user is only affected by his one hop network neighbors, whose actions are also directly observable. Users have a choice between two actions: C (for Cooperate), and D (for Defect). When all users choose their actions, each user receives a payoff that depends on his own and his neighbors' actions, and his own and his neighbors' types. Observe that the user is playing the same action against all neighbors. As an extension, a user's actions could be different for different links. The whole issue is about how much granularity of control each user has. If all the user can (or wants to) do is turn a switch ON or OFF, then the only model that can be used is a single action for all neighbors.

The payoff is decomposed as a sum of payoffs, one term for each adjacent link. Each term of the sum depends on the user's own action and type, and the action of his neighbor along that link. It may also depend on the type of his neighbor along that link, but in that case the separate terms of the sum are not revealed to him. Only the total payoff (sum of all link payoffs) is revealed to the user, otherwise he

would be able to discover immediately the types of his neighbors.

The payoff of user i is denoted by $R_i(a_i|t_i)$, when i's action is a_i and i's type is t_i. We extend this notation to denote by $R_i(a_i a_j|t_i t_j)$ the payoff for i when j is a neighbor of i and j's action and type are a_j and t_j. So, the decomposition of i's payoff along each adjacent link can be written as

$$R_i(a_i|t_i) = \sum_{j \in N_i} R_i(a_i a_j|t_i t_j), \tag{2.5}$$

or, when i's payoff does not depend on the types of his neighbors,

$$R_i(a_i|t_i) = \sum_{j \in N_i} R_i(a_i a_j|t_i \cdot). \tag{2.6}$$

We assume there are no links between any two Bad users. The Bad users are supposed to be able to communicate and coordinate perfectly; hence, there is no need to restrict their interaction by modeling it in these terms. Moreover, the Bad users know exactly both the topology and the type of each user in the network. Good users only know their local topology, e.g., how many neighbors they have and what each one of them plays, but not their types.

Since there are no links between Bad users, the two games that can appear are Good versus Good, and Good versus Bad. The payoffs in matrix form are shown in Figure 2.3, where i is the row player, and j is the column player. For instance, $R_j(DC|BG)$ is the payoff of the column player j when he plays D, his type is B, the row player plays C, and the row player's type is G.

The payoff of a Bad user is the opposite of the payoff of the Good user. That is, the Good versus Bad game is a zero-sum game. Also, the Good versus Good

Bad

Good		C	D
	C	$R_i(CC\|GB), R_j(CC\|BG)$	$R_i(CD\|GB), R_j(DC\|BG)$
	D	$R_i(DC\|GB), R_j(CD\|BG)$	$R_i(DD\|GB), R_j(DD\|BG)$

Good

Good		C	D
	C	$R_i(CC\|GG), R_j(CC\|GG)$	$R_i(CD\|GG), R_j(DC\|GG)$
	D	$R_i(DC\|GG), R_j(CD\|GG)$	$R_i(DD\|GG), R_j(DD\|GG)$

Figure 2.3: The two games that can take place on a link: Good versus Bad and Good versus Good. The payoffs are shown in their most general form.

game obviously needs to be symmetric. These two observations allow us to simplify the presentation of the payoff matrices as shown in Figure 2.4.

The game is played repeatedly with an infinite horizon, and time is divided in rounds $t = 1, 2, 3, \ldots$. Actions and payoffs of user i at round t are denoted with superscript t: a_i^t and R_i^t. Each Good user i remembers his own payoff and action at each past round, and the actions of his neighbors at each past round. The n-round *history* available to user i is defined to be the collection of all this information from round 1 up to and including round n:

$$\mathcal{H}^{1\ldots n} = \{\forall j \in N_i, \forall t \in \{1, \ldots, n\} : a_j^t\} \cup \{\forall t \in \{1, \ldots, n\} : a_i^t, R_i^t\}. \tag{2.7}$$

User i then uses the n-round history to decide what his action will be in round $n + 1$. In particular, the history is used to form estimates about actions expected

Bad

Good \	C	D
C	$r_1, -r_1$	$r_2, -r_2$
D	$r_3, -r_3$	$r_4, -r_4$

Good

Good \	C	D
C	s_1, s_1	s_2, s_3
D	s_3, s_2	s_4, s_4

Figure 2.4: The two games that can take place on a link: Good versus Bad and Good versus Good. The payoffs have been simplified to reflect that the Good versus Bad game is zero-sum, and the Good versus Good game is symmetric.

of i's neighbors at round $n+1$. The estimates can either be directly about the neighbors' actions, or indirectly through first estimating their types and then the probability that given their type they will play a certain action. The expected payoff of user i for round $n+1$ will be as follows (all the probabilities are implicitly conditioned upon the n-round history):

$$\mathbb{E}[R_i^{n+1}] = \Pr(a_i^{n+1} = C) R_i^{n+1}(C) + \Pr(a_i^{n+1} = D) R_i^{n+1}(D), \tag{2.8}$$

where $R_i^{n+1}(C)$ is the expected round $n+1$ payoff for user i, given that he plays C at round $n+1$, which we compute below. A similar definition and computation holds for $R_i^{n+1}(D)$.

$$
\begin{aligned}
R_i^{n+1}(C) = & \sum_{j \in N_i} R_i(CC|GB) \cdot \Pr(a_j^{n+1} = C | t_j = B) \cdot \Pr(t_j = B) \\
& + \sum_{j \in N_i} R_i(CC|GG) \cdot \Pr(a_j^{n+1} = C | t_j = G) \cdot \Pr(t_j = G) \\
& + \sum_{j \in N_i} R_i(CD|GB) \cdot \Pr(a_j^{n+1} = D | t_j = B) \cdot \Pr(t_j = B) \\
& + \sum_{j \in N_i} R_i(CD|GG) \cdot \Pr(a_j^{n+1} = D | t_j = G) \cdot \Pr(t_j = G)
\end{aligned}
\tag{2.9}
$$

Then, user i will choose the action (C or D) that gives him the largest expected payoff. After all players choose their actions, they receive their payoffs and observe their neighbors' actions. They incorporate that information in their history, and update the probability estimates accordingly.

In the next two chapters, we will give specifics of two models that instantiate the general model we just described. The difference will be, first and foremost, in the way that Good users treat collaboration with Bad users. In the next chapter, collaboration with Bad users will be identical to collaboration with Good ones, that is, the costs and benefits will be the same in both cases. In the chapter after that, the assumption will be that a Good user prefers to collaborate with Good users, but receives a negative payoff when collaborating with Bad users. So, he will be able to detect that Bad users are among his neighbors, but he will not immediately know which ones are the Bad ones. There, the interest will be in how soon the Good user discovers which the Bad neighbors are. Additionally, upon discovering a Bad user, a Good user will be able to break the link that joins them, thus altering the game graph.

2.3 Related work

Repeated games with incomplete information were introduced by Aumann and Maschler [6]. The 2-person case is also discussed by Myerson [44], and, at greater depth, by Zamir and Forges [5].

The most relevant piece of game theoretic literature is the work by Singh, Soni, and Wellman [54]. There they deal with the case of tree games with incomplete information, i.e., the graph of the game is a tree, the players have private information, but there is no history (the game is not repeated). They provide algorithms for finding approximate Bayes-Nash equilibria. Their algorithm does not generalize to non-tree graphs in an obvious way.

Game theoretic considerations have been applied to trust computation before, but on completely different models of interaction. Kreps and Wilson [32] address the chainstore game, in which there is one player who plays everybody else in turn. A notion of reputation of this so-called "long run" player is developed, which corresponds to the expectations of the "short term" players whose turn has not come yet. This introduces a fundamental asymmetry among the players which is non-existent in our model.

Morselli et al. [41], and similarly Friedman et al. [24], assume that there is a network of users who are playing a game repeatedly. However, their model proceeds in a completely different direction than ours. They assume that users are paired at random and play a 2-person game (their example is Prisoner's Dilemma). In so doing, they do not incorporate any notion of topology (users playing against their

neighbors). Moreover, they either assume that "the system" can see what everyone is doing, as well as give this information to everybody [24], or that each player broadcasts his experience after each round (what his partner did) [41]. For the networks that we are considering (resource-constrained, and without a centralized authority) such luxuries are prohibited. Moreover, we are taking the different approach of separating the problem of building up trust values for one's neighbors based on direct interactions from the problem of computing indirect trust values which are based on neighboring users' recommendations for the rest of the users.

In the literature for inducing cooperation among network users, mostly selfish users have been studied. The work by Blanc, Liu, and Vahdat [11] is an example of providing incentives for users to cooperate (another example is Buttyán and Hubaux [15]). However, they are modeling Malicious Users as "Never Cooperative", without any further sophistication, since their main focus was discouraging selfish free-riders. There is no degree of selfishness that can approximate the behavior of our Malicious Users. For example, Félegyházi, Hubaux and Buttyán [23] assume that the payoff function of a user is non-decreasing in the throughput experienced by the user. Our Bad users do not care about their data being transmitted. For the same reason, the model proposed by Urpi, Bonuccelli, and Giordano [60] does not apply (as the authors themselves point out).

In other related work, Srinivasan, Nuggehalli, Chiasserini, and Rao [56] are using a modified version of Generous Tit For Tat (for an early famous paper in the history of Tit for Tat see [7]), but they have no notion of topology and, consequently, of neighborhoods. In their setting, each user is comparing his own frequency

of cooperation to the aggregate frequency of cooperation of the rest of the network. Altman, Kherani, Michiardi, and Molva [4] proposed a scheme for punishing users whose frequency of cooperation is below the one dictated by a certain Nash equilibrium. Aimed particularly against free-riding in wireless networks is the work by Mahajan, Rodrig, Wetherall, and Zahorjan [36], and also the one by Feldman, Papadimitriou, Chuang, and Stoica [22].

Malicious users and attacks have been mostly considered from a system perspective for particular protocols or algorithms (e.g. securing Distributed Hash Tables [55], or from a cryptographic viewpoint: key exchanges, authentication, etc.). For a summary from the perspective of P2P networks, see Ref. [17].

To the best of our knowledge, there has been no game theoretic modeling of malicious users as we describe them here. Moscibroda, Schmid, and Wattenhofer [42] have modeled malicious users in a game theoretic setting, but in a different way. The game they are considering is a virus inoculation game, in which selfish users decide whether to pay the cost for installing anti-virus software (inoculation), or not pay and risk getting infected. The malicious users declare that they have been inoculated, when in fact they have not, so as to mislead the selfish ones. After the selfish users have made their decisions, the attacker chooses an uninoculated user, uniformly at random, and infects him. The infection propagates to all unprotected users that can be reached from the initially infected users on paths consisting of unprotected users (the malicious ones are equivalent to unprotected). One major difference is that in this model the selfish users are assumed to know the topology of the network (a grid, in particular), whereas in our model they only know their local

neighborhood topology. Another difference is that the notion of Byzantine Nash Equilibrium that they consider is restricted to the strategies of the selfish users alone.

Chapter 3

Collaboration in the Face of Malice

3.1 Introduction

In this chapter, we follow the general framework described in Chapter 2. We use repeated graphical games of incomplete information to model the effect of malicious users on the operation of a network. There are two salient points introduced and used in this chapter. First, Good users do not care if they cooperate with other Bad users. They get the same costs and benefits in either case. Second, the Good users' indifference to the other players' type implies that Good users only care about estimating their neighbors' potential future actions. They estimate a neighbor's next action through the *fictitious play* process, explained in the next section. In brief, this means that each user assumes that each neighbor plays an action with the same probability independently at every round. This probability is estimated as the empirical frequency of past observations of that action.

3.2 Malicious and Legitimate User Model

For the purposes of this section, we define a game with the payoffs shown in table form in Fig. 3.1 for the two pairs of types that can arise (Good versus Good, and Good versus Bad). Using the R-notation, the payoffs for a Good player would

Bad

Good \ Bad	C	D
C	$N-E, E-N$	$-E, E$
D	$0, 0$	$0, 0$

Good

Good \ Good	C	D
C	$N-E, N-E$	$-E, 0$
D	$0, -E$	$0, 0$

Figure 3.1: The two games that can take place on a link: Good versus Bad and Good versus Good.

be as follows, independent of the type of the opponent:

$$R_i(CC|G\cdot) = N - E \tag{3.1}$$

$$R_i(CD|G\cdot) = -E \tag{3.2}$$

$$R_i(DC|G\cdot) = 0 \tag{3.3}$$

$$R_i(DD|G\cdot) = 0, \tag{3.4}$$

while for a Bad player they would be as follows, keeping in mind that Bad players only play against Good ones, not against other Bad:

$$R_i(CC|B\cdot) = E - N \tag{3.5}$$

$$R_i(CD|B\cdot) = 0 \tag{3.6}$$

$$R_i(DC|B\cdot) = E \tag{3.7}$$

$$R_i(DD|B\cdot) = 0. \tag{3.8}$$

We explain the payoffs as follows: In the example of a wireless network, a C means that a user makes himself available for communication, that is, forwarding traffic of other nodes through himself. A link becomes active (i.e., data is exchanged over it) only when the users on both endpoints of the link cooperate, that is, play C. Playing C is in line with what Good users want to achieve – good network operation – but it costs energy, since it means receiving and forwarding data. So, when both players on a link play C, the Good player (or both players, if they are both Good) receives N (for Network) minus E (for Energy) for a total of $N - E$. We assume $N > E > 0$, otherwise no player would have an incentive to play C. On the other hand, when a Good player plays C and the other player D, then the Good player only wastes his energy since the other endpoint is not receiving or forwarding any data. For this reason, the payoff is only $-E$. In general, the values of E and N provide a way to quantify the selfishness of Good users, if we allow these values to vary across users. That is, values E^i and N^i, instead of E and N, would signify user i's *personal* cost of spending energy, and his *personal* benefit from activating an adjacent link. The larger the ratio $\frac{E^i}{N^i}$, the more selfish the user.

The Bad user's payoff is always the opposite of the Good user's payoff. In particular, we do not assume any energy expenditure when the Bad users play C.

In a peer-to-peer network, a C would mean uploading high quality content (as well as, of course, downloading), and a D would be the opposite (e.g. only downloading). The benefit of cooperation is the increased total availability of files. The cost of cooperation E could be, for instance, the hassle and possible expense associated with continuing to upload after one's download is over. In a general social

network, edges would correspond to social interactions, a C would mean cooperating with one's neighbors toward a socially desirable objective (like cleaning the snow from the sidewalk in front of your house), and a D would mean the opposite of a C. The cost E and benefit N are also obvious here.

In repeated games, as we have discussed, the players are allowed to have full or partial memory of the past actions. Here, we allow the Bad users to have all information about the past (their own moves, as well as everybody else's moves since the first round). On the other hand, the Good users follow a *fictitious play* process [13], that is, they assume that each of their neighbors chooses his actions independently and identically distributed according to a probability distribution with unknown parameters (Bernoulli in this case, since there are only two actions available: C and D). So, at each round they are choosing the action that maximizes their payoff given the estimates they have for each of their neighbors' strategies. For example, if player i has observed that player j has played c Cs and d Ds in the first $c+d$ rounds, then i assumes that in round $t = c+d+1$, j will play C with probability $\frac{c}{c+d}$ and D with probability $\frac{d}{c+d}$. We denote by q_j^t the estimated probability that j will play C in round $t+1$, which is based on j's actions in rounds $1, \ldots, t$.

We choose the average payoff as our repeated game payoff function:

$$R_i = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} R_i^t. \tag{3.9}$$

Let us now calculate the expected payoff for each of the two actions of a Good user. We assume that t rounds have been completed, and Good user i is

contemplating his move in round $t+1$. The following equation is an instantiation of the generic payoff equation (2.9) for the specific payoffs shown in (3.1).

$$\begin{aligned} R_i^{t+1}(C|G) &= \sum_{j \in N_i} \left\{ q_j^t R_i(CC|G) + (1-q_j^t) R_i(CD|G) \right\} \\ &= \sum_{j \in N_i} \left\{ q_j^t N - E \right\} \\ &= N \cdot \sum_{j \in N_i} q_j^t - |N_i| \cdot E \\ R_i^{t+1}(D|G) &= 0. \end{aligned} \tag{3.10}$$

So, in order to decide what to play, user i has to compare the expected payoff that each action will bring. Action C will be chosen if and only if $\mathbb{E}R_i(C|G) \geq \mathbb{E}R_i(D|G)$, that is, iff

$$\sum_{j \in N_i} q_j^t \geq |N_i| \frac{E}{N}. \tag{3.11}$$

Therefore, the Good user will add the estimates for his neighbors and compare to the quantity $|N_i|\frac{E}{N}$ to decide whether to play C or D. This makes the implementation of the Good user behavior particularly simple, since they only need to keep track of one number for each neighbor, as opposed to the whole history of actions observed. Our main aim in this chapter is to see what the Bad users can do against this strategy. On the one hand, perhaps they can exploit its simplicity to gain the upper hand against the Good users. On the other hand, the fact that the Bad users have full knowledge of the history and can achieve perfect coordination in their actions may not be very useful here. The reason is that Good users only care about the frequencies of the actions that they observe (see Eq. (3.11)), and the Bad users will achieve nothing by more elaborate strategies. This point is worth

repeating: The only relevant decision that the Bad users can make is to choose the frequency of their Cs and Ds. Nothing more elaborate than that will be noticed by the Good users. So, we will be assuming that the Bad players can only choose a fixed probability with which they will be playing C.

The choice of payoff that we made (Eq. (3.9)) implies that only the steady state matters when discussing the sequences of actions of the users. In particular, it allows us to talk about frequencies instead of probabilities, since frequencies will have converged in the steady state. Initial (a priori) estimates will not matter. What does matter, however, is our assumption that the Good users start out by playing C. If that is not the case, the game will in general converge to an equilibrium where all users are playing D.

As a justification for the choice of fictitious play as the process that the Good users follow, we note that very similar behavior assumptions have been done when computing trust and reputation values. In this chapter, a user's reputation corresponds to the probability that he plays C. An example is the work by Buchegger, and Le Boudec [14], who, in summary, are assuming that a user can misbehave with an unknown probability θ. This probability is then estimated by gathering observations and updating a prior distribution (a Beta distribution) through Bayes' law. The Beta distribution function has also been used in Ismail and Jøsang's work [28]. It is particularly useful because it can be used to count positive and negative events, which are often used to represent a user's behavior in a network, and then compute the user's reputation as the mean value of the Beta distribution.

3.3 Searching for a Nash Equilibrium

In game theory, as we have discussed, the solution concept we are dealing with most frequently is the Nash Equilibrium. The Nash Equilibrium is a tuple of strategies, one strategy for each player, with the property that no single player would benefit from changing his own equilibrium strategy, given that everybody else follows theirs. In our case, we have already restricted the Good players' strategies to fictitious play, as shown in Eq. (3.11), so the Nash Equilibrium will be restricted in this sense.

More formally, the Nash Equilibrium in our case would be a vector $\vec{q} \in [0,1]^{|V|}$, where q_i is the frequency with which user i plays C. The subvector $\vec{q}_G$ corresponding to the Good users will contain only 0s and 1s, according to whether Eq. (3.11) is false or true, respectively. For example, if Eq. (3.11) is false for user i, then the i^{th} element of $\vec{q}$ will be zero. The subvector $\vec{q}_B$ corresponding to the Bad users will contain values in $[0,1]$ such that no other value of q_i would increase the payoff of Bad user i. The payoff of Bad user i when he is playing C with frequency q_i is

computed as follows:

$$\begin{aligned} R_i(q_i|B) = & \sum_{j \in N_i : q_j = 1} \{q_i R_i(CC|B) + (1-q_i) R_i(DC|B)\} \\ & + \sum_{j \in N_i : q_j = 0} \{q_i R_i(CD|B) + (1-q_i) R_i(DD|B)\} \\ = & \sum_{j \in N_i : q_j = 1} \{q_i (E-N) + (1-q_i) E\} \\ & + \sum_{j \in N_i : q_j = 0} \{q_i \cdot 0 + (1-q_i) \cdot 0\} \\ = & (E - N q_i) |\{j \in N_i : q_j = 1\}| \end{aligned} \tag{3.12}$$

Note that since a Bad user only has Good neighbors, and Good users only play always C or always D, the q_j ($j \in N_i$) will all be either 0 or 1.

The problem with the equation we just wrote is that it is not immediately obvious what the cardinality of the set $\{j \in N_i : q_j = 1\}$ is. This set contains the neighbors of a Bad user that play C. In general, what a Good user plays will be affected by the choice of frequencies by all the Bad users. Actually, it will not just be affected: it will be completely determined. Given a choice of frequencies $\vec{q}_B$ by the Bad users, we describe an algorithm (see Fig. 3.2) to compute the optimal responses of the Good users, which will in turn determine the payoffs for every user (Good or Bad). Our purpose in presenting this algorithm is to show how Good users' actions would affect one another and Ds would propagate through the network. It is not claimed to be the most efficient to compute what the equilibrium is.

In a few words, the algorithm starts by assuming that all Good users play C. Then, they check the validity of Eq. (3.11), and some start playing D. Then, the ones who started playing D may cause their Good neighbors who are still playing C

to also start playing D. The set S contains these users who still need to (re)check Eq. (3.11), since its validity may have changed.

EQUILIBRIUM$(G, \vec{q}_B)$

$\triangleright$ All Good users are initialized to playing C,

$\triangleright$ i.e., $\vec{q}_G$ contains only 1s.

$\triangleright$ S contains users that may switch from C to D.

1 $S \leftarrow V_G$

2 **while** $S \neq \emptyset$

3 **do** $i \leftarrow$ REMOVE(S)

4 **if** SUM$(i) < \frac{E}{N}|N_i|$

5 **then**

6 $q_i \leftarrow 0$

7 $S \leftarrow S \cup \{j | j \in N_i \wedge j \in V_G \wedge q_j = 1\}$

Figure 3.2: The algorithm EQUILIBRIUM computes the optimal actions (C or D) for the Good users, given the subvector $\vec{q}_B$ of the Bad user frequencies. The procedure SUM(i) returns the sum of the frequencies of i's neighbors. Since a Good user switching from C to D could cause other Good users to switch from C to D, the while-loop needs to run until there are no more candidates for switching.

In general, a change in the frequency q_i of a Bad player i will affect the payoffs of other Bad players, too, since it will change the optimal responses of Good users that could, e.g., be common neighbors of i and other Bad users. So, the definition of the Nash Equilibrium we gave earlier could be expanded to regard the Bad players

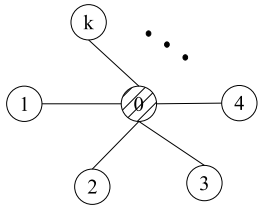

Figure 3.3: Single Bad user maximizes his own local payoff.

as a team that aims to maximize the total payoff, rather than each Bad user trying to maximize his own individual payoff. In Section 3.3.1, we will see a case when local maximization by each Bad user is equivalent to maximization of the sum of all the Bad users' payoffs. In Section 3.3.2 we will examine more closely why the two objectives are in general different and will describe a heuristic for the general case.

3.3.1 The Uncoupled Case

Let us look at the case of a single Bad player in the whole network. Since no other Bad players exist, the choice of the Bad user will only affect his own payoff. We will see what he has to do (i.e., with what frequency to play C) in order to maximize his payoff, in a tree topology where the Bad player is at the root of the tree. In Figure 3.3 we see the root (Bad user) and the one-hop neighbors only.

Assume that the Bad user – labeled user 0 – has k neighbors, labeled $1, \ldots, k$. We also assume that all the Good users will start by playing C, and will only change to D if they are forced by the Bad user. Applying Eq. (3.11) for each neighbor, we see that each expects to see a different sum of frequencies from his own neighbors

in order to keep playing C. User i expects to see a sum of frequencies that is at least $\frac{E}{N}|N_i|$. Since all of i's neighbors except user 0 are Good, they will at least start by playing C, so user i will see a sum of frequencies equal to $|N_i| - 1 + q_0$ (q_0 is the frequency with which the Bad user 0 is playing C). So, in order to make user i continue playing C, the Bad user should play C with frequency

$$q_0 \geq \frac{E}{N}|N_i| - (|N_i| - 1) = 1 - |N_i|(1 - \frac{E}{N}) \equiv t_i, \tag{3.13}$$

which is decreasing with $|N_i|$, since $E < N$. We call this quantity the *threshold* t_i:

Definition. The *threshold* t_i of a Good user i with g Good neighbors is the sum of frequencies of Cs that his $|N_i| - g$ Bad neighbors need to play, so that he keeps playing C. We assume that all his Good neighbors play C.

$$t_i = \max\left\{\frac{E}{N}|N_i| - g, 0\right\} \tag{3.14}$$

Without loss of generality, we assume that the Bad user's neighbors are labeled in increasing order of t_i. So, $t_1 < t_2 < \ldots < t_k$. By choosing $q_0 = t_k$, the Bad user guarantees that all his neighbors will be playing C. The payoff that he receives by playing $q_0 = t_k$ can be calculated using Eq. (3.12) to be equal to $R_0(t_k|B) = k(E - Nt_k)$. In general, the following holds:

$$R_0(t_j|B) = j(E - Nt_j). \tag{3.15}$$

It does not make sense for the Bad user to choose any value for q_0 other than one of the t_j, since it would not make any difference to the Good users; it would only reduce the payoff of the Bad user. So, the aim of the Bad user is to find the

value of t_j that maximizes the payoff. Note that as j increases, the first term of the product increases, but the second term decreases. This gives a bound on the acceptable values that q_0 can take. It cannot be higher than $\frac{E}{N}$, since that would make the payoff negative, when the Bad user can always guarantee a payoff of at least 0 by choosing $q_0 = 0$. In order to find the optimal t_j we compare the payoffs for all values of j, and pick the maximum. For two values of q_0, say t_l and t_m with $l < m$, the comparison boils down to:

$$\begin{aligned} R_0(t_l|B) > R_0(t_m|B) &\Leftrightarrow l(E - Nt_l) > m(E - Nt_m) \\ &\Leftrightarrow t_m > \frac{l}{m}t_l + (1 - \frac{l}{m})\frac{E}{N} \end{aligned} \tag{3.16}$$

When the Bad user chooses a value for q_0, some of his neighbors will play C and some D. The ones who play D may cause their own neighbors to start playing D, and so on, in a spirit similar to the algorithm described in the previous section. However, the Ds cannot, by propagating, influence other neighbors of the Bad user: That is a consequence of the tree topology that we have assumed, since the only path that goes between two one-hop neighbors of the Bad user, goes through the Bad user.

What happens when there are multiple Bad users in a general topology? We will examine the circumstances under which the maximization of the total sum of Bad users payoff is achieved through the local maximization of each Bad user's payoff. This local maximization is done as we have just described in Eq. (3.16). We call "Uncoupled Case" the situation described by these circumstances.

We will find it useful to define the notion of the *tolerance* of a Good user.

Definition. The *tolerance* of a Good user is the largest number of his one-hop neighbors that can play D, before he starts playing D himself.

The tolerance of a Good users is a function of $\frac{E}{N}$. To compute the tolerance of user i, assume that n of his neighbors play D, and $|N_i| - n$ play C. From Eq. (3.11), for user i to play C the following needs to hold:

$$|N_i| - n \geq |N_i|\frac{E}{N} \Leftrightarrow n \leq |N_i|(1 - \frac{E}{N}) \tag{3.17}$$

The tolerance is the largest integer n for which this equation holds, i.e., $n_{\max} = \lfloor |N_i|(1 - \frac{E}{N}) \rfloor$.

To show that local maximization is equivalent to global maximization, we need to make sure that Good players who start playing D do not cause, recursively, "too many" other Good users to play D so that the payoffs of other Bad users are affected. Going back to the algorithm EQUILIBRIUM, we can see that this will happen if and only if the nodes that play D because of a Bad user B_1 are separated by at least two nodes (three hops) from the nodes that play D because of any other Bad user. In other words, there needs to be a layer of nodes at least two nodes deep that have large enough tolerances so that they will not start playing D themselves. Since, for a fixed $\frac{E}{N}$, the tolerance of a user depends only on the number of his neighbors, nodes with a high degree that are connected to each other would provide the highest resistance to playing D. In graph theoretical terms, the greatest "aggregate" tolerance is achieved, for a given number of nodes, when the nodes are connected in a clique (maximum connectivity).

We now apply these considerations to a specific example, shown in Fig. 3.4.

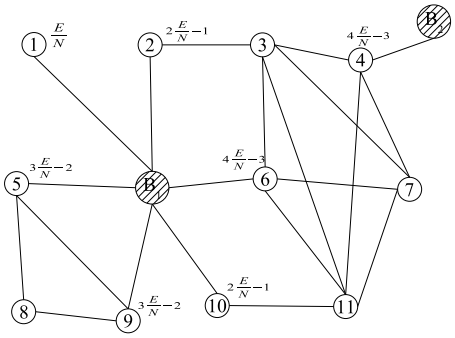

Figure 3.4: The Uncoupled Case. Shown next to each node is its threshold: The frequency of Cs it expects to see from the neighboring Bad user. It is equal to $\frac{E}{N}|N_i| - (|N_i| - 1)$, since each Good user happens to have at most one Bad neighbor.

There are two Bad users, labeled B_1 and B_2, and eleven Good users. Shown in the picture are the thresholds of the Good users who are neighbors of a Bad user. Remember that the threshold of a Good user is the frequency of Cs that the Bad neighbor should play if the Good user is to play C, assuming that all the other neighbors of the Good user play C.

The thresholds of B_1's neighbors are, in ascending order: $4\frac{E}{N} - 3, 3\frac{E}{N} - 2, 2\frac{E}{N} - 1, \frac{E}{N}$. These, along with the value 0, are the options that B_1 has for choosing his q_{B_1}. To keep things simple, we assume that $\frac{E}{N}$ is such that all the above thresholds are positive. In particular, this translates to $\frac{E}{N} > \frac{3}{4}$.

The table below shows the choices that B_1 can make and the associated payoffs.

The thresholds are computed using Eq. (3.14), and the payoffs using Eq. (3.12).

frequency	payoff
0	0
$4\frac{E}{N} - 3$	$3(N - E)$
$3\frac{E}{N} - 2$	$6(N - E)$
$2\frac{E}{N} - 1$	$5(N - E)$
$\frac{E}{N}$	0

So, to maximize his own payoff, B_1 should play C with frequency $3\frac{E}{N} - 2$. On the other hand, the maximizing frequency for B_2 is $4\frac{E}{N} - 3$, which brings him a payoff of $3(N - E)$.

How will the Good users react to these choices? In effect, we will now go through what the algorithm EQUILIBRIUM in Fig. 3.2 would do. Users 5,6, and 9 will play C, and users 1,2, and 10 will play D because B_1's choice is above or below their thresholds, respectively. User 4 will play C because of B_2's choice.

For the two-hop neighbors of the Bad users, we will have the following: User 8 will obviously play C, since both his neighbors (5 and 9) play C. For users 3 and 11, the situation is identical: They have five neighbors each (hence they need to see a sum of frequencies equal to $5\frac{E}{N}$ to play C), and exactly one of their neighbors is playing D (2 and 10, respectively). So, each sees a total sum of frequencies equal to 4, and they will play C if $4 > 5\frac{E}{N} \Leftrightarrow \frac{E}{N} < \frac{4}{5}$. Although we have constrained $\frac{E}{N}$ to be greater than $\frac{3}{4}$, there is still a range of values for $\frac{E}{N}$, namely $\frac{3}{4} < \frac{E}{N} \leq \frac{4}{5}$, for which users 3 and 11 will play C. The only remaining user is user 7, who will obviously play C, since all his neighbors are playing C.

So, to recapitulate, we have verified that the individual maximization of each Bad user's payoff results, in this case, in the global maximization of the sum of the payoffs of the Bad users.

3.3.2 The General Case and a Heuristic Solution

In this section, with the help of Fig. 3.5, we will see that local maximization of Bad user payoffs does not always correspond to maximization of the sum of their payoffs. This case particularly applies when a Good user has two or more Bad neighbors.

The situation for Bad user B_1 has not changed compared to Fig. 3.4. The maximizing frequency is still $3\frac{E}{N} - 2$, and the payoff is still $6(N - E)$. As far as B_2 is concerned, we can see that the maximizing frequency is also $3\frac{E}{N} - 2$, which will bring him a payoff of $4(N - E)$. However, if we apply the algorithm of Fig. 3.2, we will find that if B_1 sets $q_{B_1} = 3\frac{E}{N} - 2$ and B_2 sets $q_{B_2} = 3\frac{E}{N} - 2$, then things will not go "as planned" by the Bad users. Namely, user 2 will start playing D, because his threshold is not satisfied. As a result, user 3 will see a total sum of frequencies equal to $0 + 1 + 3\frac{E}{N} - 2 = 3\frac{E}{N} - 1$, which is lower than the total sum of $3\frac{E}{N}$ that he expects, and so he will start playing D. Therefore, B_2's payoff will be diminished. If we carry the calculations further, we will see that, after 3 starts playing D, neither user 6 sees the total sum of frequencies he needs to keep playing C – he only sees $0 + 1 + 1 + 3\frac{E}{N} - 2 = 3\frac{E}{N} < 4\frac{E}{N}$. So user 6 will also start playing D, which will diminish B_1's payoff, too. Eventually, all Good users except 5,8, and 9 will play D.

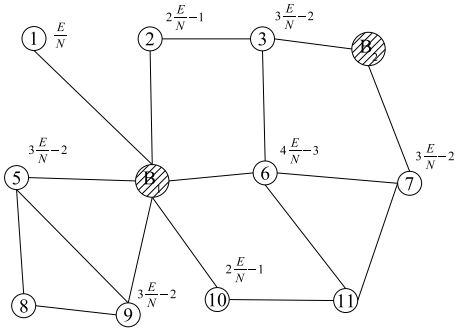

Figure 3.5: The General Case. Shown next to each node is its threshold: The frequency of Cs it expects to see from the neighboring Bad user. It is equal to $\frac{E}{N}|N_i| - (|N_i| - 1)$, since each Good user happens to have at most one Bad neighbor.

One could argue that a Malicious User would be happy with a situation like that, i.e., when most of the Good users are playing D. In that case, the network ceases to operate, which is supposed to be what a Malicious User wants. However, by playing D, the Good users do not use up their energy. Remember that another aim of the Bad users is to deplete the energy of the Good ones. Although we have not incorporated this in our model, we could set a finite limit on the amount of energy that the Good users can use. Then, it makes more direct sense for the Bad users to try and deplete the Good users energy, which seems to be infinite in our current model. Moreover, we could constrain the life of the Bad users to some finite (or probabilistic) length of time (after which they get caught, for example, and the network operates smoothly). Then, again it would make more direct sense for the Bad users to "hurry up" and waste the energy of the Good users, so that the network will not be able to operate for very long, even after the Bad users are caught. Even

in a non-wireless network, where energy is not a concern, Bad users would still try to exploit the cost of cooperation. Remember that the cost of cooperation is a fundamental characteristic of the situations we are modeling.

We now look at what the Bad users should do, in order to maximize the sum of their payoffs. We keep referring to Fig. 3.5. In general, either one of two options is likely to achieve the optimal sum: Either one of the two nodes sacrifices his own payoff to a small or large extent, so that the other one can keep playing his maximizing frequency, or they both sacrifice a part of their individually maximum payoffs, in such a way that the sum of payoffs is maximized.

We observe that the Bad user B_2 has two choices for his frequency: either 0, or $3\frac{E}{N} - 2$, which do not coincide since $3\frac{E}{N} - 2 > 3\frac{3}{4} - 2 > 0$ (we keep assuming, as previously, that $\frac{E}{N} > \frac{3}{4}$). If B_2 plays 0, then both his neighbors (3 and 7) will start playing D, and so B_2 will get a zero payoff. As a result, user 2 will expect a total sum of frequencies equal to $2\frac{E}{N}$ from B_1, which is more than the absolute maximum frequency of $\frac{E}{N}$ that a Bad user needs to play in order to keep his payoff positive. So, user 2 would also play D. Moreover, because of users 3 and 7 playing D, user 6 would need a larger frequency of Cs from B_1. We could continue this analysis, but it has become clear that the Bad users are worse off with this choice of B_2.

If B_2 chooses $3\frac{E}{N} - 2$, then both 3 and 7 will play C, if they are not affected by the choice of B_1. The payoff for B_2 would then be $4(N - E)$. We have already seen that if B_1 chooses his maximizing frequency $3\frac{E}{N} - 2$, then many users will end up playing D, thus ruining the payoffs of the Bad users. So, we turn to the next best local option for B_1, the frequency $2\frac{E}{N} - 1$, which gives him a payoff of $5(N - E)$.

We see that this keeps all neighbors of B_1 at C except user 1. Therefore, the rest of the network is not affected, and everybody keeps playing C. The sum of payoffs for the two Bad users is now $5(N-E)+4(N-E)=9(N-E)$, which is the maximum sum that can be achieved (remember that the sum of the two individual maximum payoffs is $6(N-E)+4(N-E)=10(N-E)$, and we have seen that this payoff cannot be achieved in a stable equilibrium).

We now propose a heuristic for finding a payoff as close as possible to the maximum sum of payoffs. The heuristic maximizes the payoff of the "most promising" Bad users first. In other words, the idea is to find the node whose maximum individual payoff is the largest among all individual payoffs and let him play his maximizing frequency. Then, re-evaluate the individual payoffs attainable by all other Bad users in the new situation, and again choose the "most promising" one. Finally, repeat this process until no other Bad users are left. In effect, this sacrifices the payoffs of the "least promising" Bad users for the sake of the "most promising" ones. Note that in the uncoupled case, this heuristic performs optimally.

In the topology depicted in Fig. 3.5, our heuristic would choose B_1 as the most promising Bad user, since his maximum individual payoff is $6(N-E)$, for a maximizing frequency of $3\frac{E}{N}-2$. That would cause Good users 1, 2, and 10 to play D because their thresholds would not be satisfied. Now, we come to what B_2 can do in the new situation. as we have already seen, he cannot play his individual maximizing frequency. The heuristic would make B_2 play whatever frequency is necessary to prevent the Ds from spreading to other Good users. For instance, to prevent user 3 from playing D, B_2 will need to play the maximum possible frequency ($q_{B_2}=1$),

despite the fact that B_2 would incur a negative payoff by doing that. Then, user 3 would see a total sum of frequencies of 2, and, if his expected sum of frequencies, $3\frac{E}{N}$, is lower than that, then he would play C. So, we see that the performance of the heuristic depends on the value of $\frac{E}{N}$. It can be shown that the lower that value is, the better the heuristic performs. If $\frac{E}{N}$ is low enough, we get the uncoupled case, for which we know that the heuristic coincides with the optimal solution.

Chapter 4

Detecting Malicious Users

4.1 Introduction

In this chapter, we still follow the general framework of Chapter 2 about repeated graphical games with incomplete information. The general scenario that we want to capture is the following: Good users want to cooperate with other Good users, but not with Bad users. Bad users, on the other hand, want to cooperate with Good users. The Good are unaware of who is Good and who is Bad, but since the game is played repeatedly they can gradually detect the Bad ones. We will explore strategies that the Good users can follow to detect and isolate the Bad ones. It resembles an intruder detection framework in the following sense: There is a mechanism to detect the existence of malicious users in a user's neighborhood, but cannot identify exactly which the malicious users are. They may be eavesdroppers who monitor traffic (through cooperating in the network operation) and try to learn and then leak sensitive information. The mechanism is then supposed to be able to detect the leak. As a result, the existence of malicious users would be deduced, but not their identity. Alternatively, they may inject malformed packets in the network (worms, etc.).

Note that we will not be assuming collusion among the Bad users, although this can be an extension of our model. Also, our model for the Bad users means that

they benefit from cooperating for as long as possible without getting caught. So, we do not cover situations where a single cooperation between a Bad and a Good user is enough, e.g., to destroy the whole network.

The rest of the chapter is organized as follows: In section 4.2, we give a detailed presentation of our model of the network and the specifics of the game played. After that, we analyze in section 4.3 the strategies and corresponding outcomes that can appear in the game. Section 4.4 shows some indicative simulations and compares to the analytical approach of section 4.3.

4.2 System Model

The network operates in rounds $t = 1, 2, \ldots$, and at each round t each user i chooses an action a_i^t: C (for Cooperate) or D (for Defect). Playing C corresponds to making oneself available for communication (e.g. sending/receiving data). Playing D corresponds to shutting down all communications to and from the user. After all users have chosen an action, each user learns his neighbors' actions (i.e. which neighbor played which action), and his own payoff for that round, which depends only on his own action and these of his neighbors. Note that a user's payoff is known only to him, and is never reported to others. If a Good user is able to tell that a particular neighbor of his is Bad, then he can sever the link that joins them, so as not to be affected by that neighbor's actions in the future. In Section 4.3, we will discuss how Good users can detect Bad ones.

At round t, the payoff R_i^t of a Good user i who played C equals the number of

Bad

Good \	C	D
C	$-1, +1$	$0, 0$
D	$0, 0$	$0, 0$

Good

Good \	C	D
C	$+1, +1$	$0, 0$
D	$0, 0$	$0, 0$

Figure 4.1: The two games that can take place on a link: Good versus Bad and Good versus Good.

Good neighbors who played C minus the number of Bad neighbors who played C. This reflects the preference of Good nodes to cooperate with other Good nodes and not with Bad ones. A Good user who played D receives a zero payoff regardless of the actions of the neighbors. This means that he risks no losses, but he has no gain, he learns nothing about his neighbors, and his neighbors learn nothing about him. The payoff of a Bad user who plays C is equal to the number of his Good neighbors who played C (remember that a Bad user has only Good neighbors). So, it is the negative of R_i^t.

We can translate the above considerations to payoffs. The payoffs for the two possible games (Good versus Good and Good versus Bad) are shown in Figure 4.1.

4.3 Analysis

To simplify the analysis, we will concentrate on a star topology network, where the central node is a Good user, and his neighbors are N Good users and 1 Bad (Fig. 4.2). We assume that the central node knows that he has exactly one Bad neighbor, but he does not know who that is. Note that the star topology assumption

is not unrealistic, since this is exactly the situation (from the point of view of the central node) even in a general network. We will also see that the assumption that only one Bad user exists can be removed without significant conceptual change in the analysis.

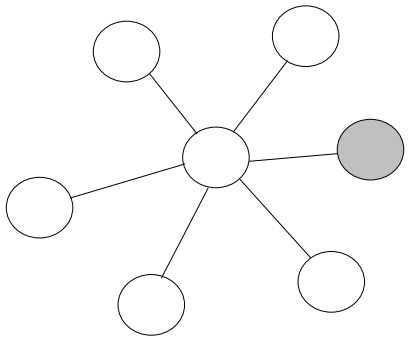

Figure 4.2: Star topology. Bad user is shaded.

Note from the System Model discussion that Good users only learn the total payoff they receive after each round, and not the per-link payoffs they receive due to their interactions with individual neighbors. So, they cannot immediately tell which of their neighbors are Good and which are Bad, but they do get some information about the types of their neighbors. In what follows, we will describe strategies for the Good and Bad players that form a Nash equilibrium. For the most part, we will be seeing things either from the central Good user's point of view, or from the point of view of the Bad user. Since in a general network all Good nodes will see themselves in the role of a central node in a (local) star topology, we are looking for strategies that are symmetrical with respect to Good nodes. That is, we want all the Good nodes to follow the same rules when choosing what to play.

Assume that the central Good user i has memory of the past history (own and neighbor moves, as well as received payoffs). Let CN_i^t (resp. DN_i^t) be the subset of i's neighbors that play C (resp. D) at round t. We assume that i plays C at round t, so i's payoff at round t is $|CN_i^t|$ if the Bad user played D, or $|CN_i^t| - 2$ if the Bad user played C (Remember that a C from a Good user gives $+1$, whereas from a Bad user it gives -1.). So, just by looking at his payoff, the central Good user i can deduce whether the Bad user played C or D at round t. The Bad user is then known to be either in the set CN_i^t or in DN_i^t. Without loss of generality, let's assume that the Bad user played C.

In the next round $(t+1)$, if the Bad user plays C again, then i can deduce that he is in the intersection $CN_i^t \cap CN_i^{t+1}$. If he plays D, then he is in $CN_i^t \cap DN_i^{t+1}$. This sequence of sets (the sequence of *hiding sets*, the initial of which is N_i) is non-increasing, but the Bad user will only be detected if it converges to a singleton set. If the behavior of the Good users is deterministic, then the Bad user can imitate a Good user, and he will never be discovered. However, if the Good nodes choose their actions in a randomized manner, they are no longer predictable.

We will look at the simplest possible randomization: each Good user plays C with probability p independently at each round. We want to compute the probability p that maximizes the central Good user's payoff. We will use the δ-discounted payoff: Given an infinite sequence of round payoffs $\{R_i^t, t = 1, 2, \ldots\}$, the game payoff for user i is $R_i = (1-\delta)\sum_{t=1}^{\infty} \delta^{t-1} R_i^t$. In our case, δ could correspond to how long the users think that the network will keep operating. To be precise, δ could be seen as the probability that the network will collapse at time $t+1$, given that it has been

operating up to and including time t.

To compute the payoff, we split the network evolution into two stages: pre-detection and post-detection of the Bad user. In the pre-detection stage, the Good users start by playing C with probability p, and we assume that the Bad user always plays C. We will see that this is the best that the Bad user can do. We again use the generic payoff equation (2.9) with the specific payoffs shown in Figure 4.1. Moreover, we know that there is exactly one Bad neighbor, so for each $j \in N_i$ the probability that j is Bad is $\Pr(t_j = B) = \frac{1}{N_i}$. Finally, the Bad users always play C, so $\Pr(a_j^{t+1} = C | t_j = B) = 1$. The expected round $t+1$ payoff for the central Good user i is

$$R_i^{n+1}(C) = \sum_{j \in N_i} R_i(CC|GB) \cdot \Pr(a_j^{n+1} = C | t_j = B) \cdot \Pr(t_j = B) \tag{4.1}$$

$$+ \sum_{j \in N_i} R_i(CC|GG) \cdot \Pr(a_j^{n+1} = C | t_j = G) \cdot \Pr(t_j = G) \tag{4.2}$$

$$= \sum_{j \in N_i} (-1) \cdot 1 \cdot \frac{1}{N_i} \tag{4.3}$$

$$+ \sum_{j \in N_i} (+1) \cdot p \cdot \frac{N_i - 1}{N_i} \tag{4.4}$$

$$= p(N_i - 1) - 1, \tag{4.5}$$

when he plays C, and

$$R_i^{t+1}(D) = 0, \tag{4.6}$$

when he plays D. Recalling that $N_i - 1 = N$ is the number of Good neighbors, the

overall expected payoff for user i at round $t+1$ is

$$R_i^{t+1} = \Pr\{a_i^{t+1} = C\} R_i^{t+1}(C) + \Pr\{a_i^{t+1} = D\} R_i^{t+1}(D) \tag{4.7}$$

$$= p(pN - 1) \tag{4.8}$$

After the Bad user has been detected, the link to him is severed and the Good nodes are free to play C forever. So, the central Good user's payoff is N from then on (+1 from each one of the N Good neighbors). Assuming that the Bad user is caught after the actions of round t_0, the total game payoff for i is

$$R_i(\delta, p, N) = (1-\delta)\{\sum_{t=1}^{t_0} \delta^{t-1} p(pN-1) + \sum_{t=t_0+1}^{\infty} \delta^{t-1} N\} \tag{4.9a}$$

$$= (1 - \delta^{t_0}) p(pN-1) + \delta^{t_0} N \tag{4.9b}$$

$$= \delta^{t_0}(N - p(pN-1)) + p(pN-1) \tag{4.9c}$$

We will now calculate the expected value of t_0 with the following argument. Since the Bad user is always playing C, his hiding set after t rounds is $N_i \cap CN_i^1 \cap CN_i^2 \cap \cdots \cap CN_i^t$. Since the Good users are playing C with probability p at every round, the size of the hiding set is reduced by a factor of p every round. Since t_0 is the detection time, it has to be that

$$p^{t_0}(N+1) = 1 \Rightarrow t_0 = -\log_p(N+1). \tag{4.10}$$

In this calculation, however, we have neglected the fact that the central Good user also plays C with probability p. When he plays D he will not be able to make any observations about his neighboring nodes, so the D-rounds will be wasted as far as detection is concerned. The above calculated t_0 is the number of *observations* that

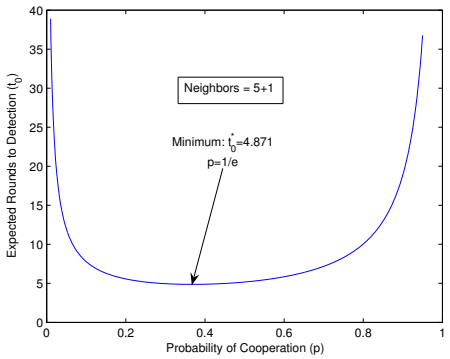

Figure 4.3: Expected number of rounds until detection.

are needed to detect the Bad user. A C will be played once every $\frac{1}{p}$ rounds, so the correct expected time of detection will be

$$t_0(p) = \frac{1}{p} \cdot (-\log_p(N+1)) = -\frac{\log_p(N+1)}{p}, \tag{4.11}$$

shown in Fig. 4.3 for $N = 5$. We can calculate that the expected number of rounds to detection is minimized for $p = \frac{1}{e}$. It is then equal to $t_0^* = e \ln(N+1)$.

Looking at Eq. (4.9b), we observe that $p(pN-1)$ (the pre-detection payoff) is always smaller than N (the post-detection payoff). This seems to imply that the Good user's payoff is maximized when δ^{t_0} is maximized, i.e. when t_0 is minimized. But this is not the case, although it is not easy to see analytically. The idea is that when δ is small enough, δ^{t_0} is very close to zero even for $t_0 = t_0^*$. As a result, R_i is practically equal to $p(pN-1)$, and it is maximized when p is maximized, i.e. for $p \to 1$.

To study the behavior of $R_i(p)$ for different values of the discount factor δ we perform numerical computations. First of all, the payoff increases monotonically with δ. This is because, regardless of the value of p, being more patient gives more

weight to the post-detection payoff, which is higher (N versus $p(pN-1)$). We can also see that the Good users can guarantee a payoff of $N-1$ (4 in this case) by choosing p very close to 1. Even though the pre-detection period can become very large, the pre-detection payoff is close to $N-1$. But we will see that in some cases, the Good can do better than that.

For a fixed value of δ the effect of an increase of p is twofold: First, the payoff during the pre-detection period is increased. Second, if p increases from 0 to $\frac{1}{e}$, then the number of rounds until detection (the length of the pre-detection period) decreases. But if p increases beyond $\frac{1}{e}$, the length of the pre-detection period increases.

It seems there are three distinct types of behavior for different intervals of δ (Fig. 4.4). When δ is small, the payoff increases monotonically with p (Fig. 4.5). This happens because a small δ implies that $\delta^{t_0} \approx 0$, and $R_i \approx p(pN-1)$ which increases monotonically with p. So, if the Good user is very impatient, then the post-detection payoff is too heavily discounted to make any difference, even if the Bad user is detected in the shortest number of rounds ($t_0^* = e\ln(N+1)$). All the Good can do in this case is increase the pre-detection payoff as much as possible by setting p to 1.

When δ is large (Fig. 4.6), δ^{t_0} is approximately equal to 1, so $R_i \approx N$ (the post-detection payoff, which is the maximum possible value of the total payoff) regardless of p. A value of δ close to 1 means that the Good users don't mind waiting; so, provided that the detection happens in finite time, the payoff is very close to N. This is reflected in Fig. 4.6 where the payoff is almost constant for a wide range of

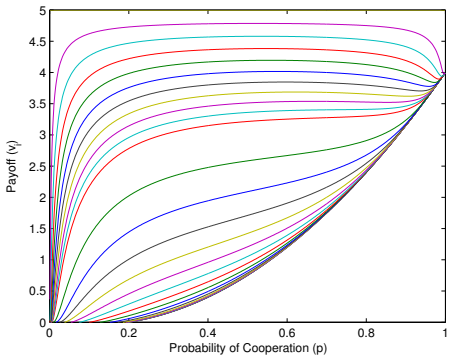

Figure 4.4: All values of δ: $0 \leq \delta \leq 0.9$, step 0.05; $0.9 \leq \delta \leq 1$, step 0.01.

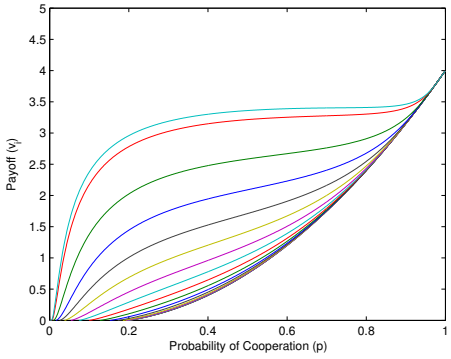

Figure 4.5: Small values of δ: $0 \leq \delta \leq 0.9$, step 0.05; 0.91.

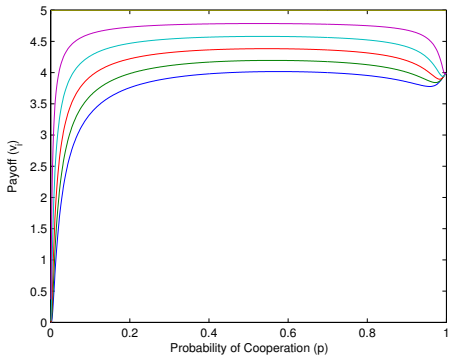

Figure 4.6: Large values of δ: $0.95 \leq \delta \leq 1$, step 0.01.

values of p, and the maximum is attained for values of p that are around 0.5 and definitely away from 1. However, when p is too close to 0 then the detection time is too large, and the pre-detection payoff is too small, so the total payoff is close to 0. When p approaches 1, the detection time is again too large (hence the observed drop in post-detection payoff). But now the pre-detection payoff is close to its maximum value of $N-1$, so the total payoff goes up again and becomes equal to $N-1$ for $p=1$.

When δ is medium (Fig. 4.7), the maximum payoff is again attained for $p=1$, as in the small δ case. However, unlike the small δ case, the payoff does not increase monotonically with p. There is a local maximum for values of p around 0.5, which is caused by the same reasons as in the large δ case, although not as pronounced.

The conclusion is that, for a given value of δ, the maximum payoff is achieved either for a probability of cooperation p equal to 1, or for some value around 0.5. The first case happens when the Good users are impatient (small δ), and so the post-detection period is too far into the future for them to care. Which means that

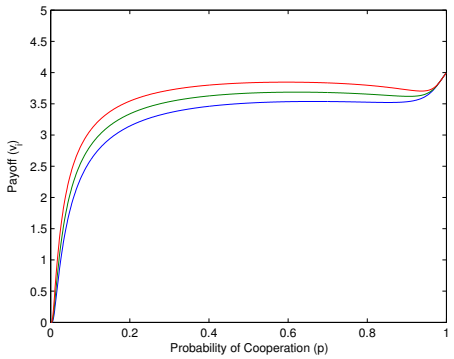

Figure 4.7: Medium values of δ: 0.92, 0.93, 0.94.

all their gains are going to come from the pre-detection period, therefore the best policy for them is to maximize the pre-detection period payoff $p(pN-1)$. The second case happens when δ is large enough for the post-detection payoff to be weighted more heavily than the pre-detection payoff. The maximum around 0.5 is happening because the post-detection payoff is maximized for $p = \frac{1}{e} \approx 0.37$ (the value of p that maximizes δ^{t_0}), but does not drop significantly for somewhat larger values of p. The pre-detection payoff is, however, monotonically increasing with p, and until around $p = 0.5$ it more than compensates for the slight drop in the post-detection payoff. After that, the post-detection payoff starts dropping faster than the pre-detection can increase.

Why does the Bad user have to play C all the time? If he plays what most Good play, he prolongs the time of detection. If he plays C, he gains payoff. If p was chosen by the Good to be larger than $\frac{1}{2}$, then these two considerations of the Bad user would both concur to playing C. However, the maximizing p may be less than $\frac{1}{2}$ for some values of δ, so it might make sense for the Bad user to play D once

in a while so as to hide a bit longer.

We will see that playing D never increases the Bad user's payoff, and it can even decrease it. Suppose the current hiding set is X, and the Bad plays D. If the central Good plays D, nothing changes. If the central Good plays C, he observes who plays D and who plays C, so by looking at the payoff he can tell what the Bad played. The new hiding set is $X \cap Y \subseteq X$, i.e. smaller than X, the Bad has gained nothing in the current period, and because of the discount factor δ the payoff of a C has become smaller. So, in effect, the Bad player is facing the exact same situation he was facing before, only he is in a smaller hiding set, and the benefit of a C is smaller. This allows us to conclude that the best thing the Bad can do is play C from the first round until he is detected. Hence, the strategies "always C" for the Bad and "C with probability p" (for the maximizing value of p) for the Good form a Nash equilibrium. Noone can do better by unilateral deviation.

4.4 Simulation for the star topology

Another observation is that we have assumed that the detection is happening at exactly the expected time. This translates to an incorrect computation of the expected payoff R_i. Using equation (4.9c) to calculate the payoff ($R_i(t)$) for a time of detection equal to t, the expected payoff over all possible times of detection (but for a fixed value of p) is

$$\mathrm{E}[R_i(t)] = \mathrm{E}[\delta^t](N - p(pN - 1)) + p(pN - 1). \tag{4.12}$$

What we have computed, however, is

$$\delta^{\mathrm{E}[t]}(N - p(pN-1)) + p(pN-1) = \tag{4.13}$$

$$\delta^{t_0}(N - p(pN-1)) + p(pN-1). \tag{4.14}$$

To estimate how far from the truth we are, we performed some indicative simulations. Shown in figure 4.8 are the comparisons of the computations to the simulations for three values of $\delta : \delta = 0.2, 0.91, 0.97$. These were chosen as representative values from each one of the three intervals of δ where the behavior of the payoff function changes. Each simulation was repeated 100 times and the mean and variance are depicted in the diagram for each value of p that the simulation was done. We see that, except for the $\delta = 0.2$ case, there is a disparity not accounted for by the variance interval. On the other hand, qualitatively the simulations are similar to the computations. For $\delta = 0.2$ they are almost identical, for $\delta = 0.91$ the simulation curve is monotonically increasing achieving its maximum at $p = 1$ (similarly to the computation), and for $\delta = 0.97$ the maximum is attained long before $p = 1$, after which the payoff drops, just as the computation predicts.

4.5 Extensions

The case of multiple Bad users in the neighborhood of a Good user is not much different in essence (there is still the concept of a hiding set), although the calculations grow longer. Even if the Good user does not exactly know how many Bad neighbors he has, he can discover after each round how many Bad users were among the cooperators. Since the set of cooperators changes in every round, the

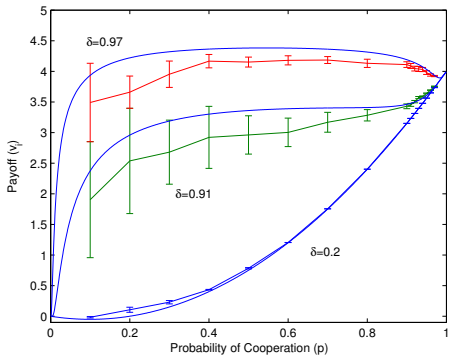

Figure 4.8: Comparison: Simulation against computation results for three representative values of the discount factor $\delta = 0.2, 0.91, 0.97$

Good user can utilize the information he gathers after each round to eventually pinpoint who the Bad users are. For example, if a set of cooperators has just one extra member compared to another set, then the type of that extra member (Good or Bad) can be immediately determined by comparing the payoffs in the two cases. Of course, this process requires extra memory capabilities on the part of the Good user (to remember what happened in the past rounds).

Chapter 5

Trust Computation: Algebraic Models and Discussion of Related Work

This chapter initiates this dissertation's second part, in which we focus on the computation of indirect trust values. We briefly summarize the relationship between direct trust generation and indirect trust computation, and we give example applications of trust computation. Most of this chapter is a discussion of related work in terms of an algebraic model. In brief, the various ways to compute trust are formalized under an algebraic terminology and their properties are discussed in terms of algebraic properties.

5.1 Introduction

Trust is a weighted binary relation between two members of a network. As an example, consider a network of intelligence gathering agents, organized in a hierarchical manner. Trust could then be the expectation of a person A (presumably high in the hierarchy) that a person B (low in the hierarchy) is honest, as opposed, e.g., to being a double agent. The weight of this relation is then a way to quantify this expectation: The greater the weight, the higher the expectation.

For an illustration associated with Public Keys, suppose that entity A wants to determine the public key associated with entity B. In this case, the trust relation

would be: "A does (or does not) believe that B's key is K_B". A and B have had no previous interactions, hence no trust relation, so A has to contact entities that have some evidence about B's key. Relevant pieces of evidence in this case are certificates binding B's key to B's identity. Also, the trustworthiness of the entities that issued these certificates should be taken into account.

In a regular Public Key Infrastructure (PKI) with a Trusted Third Party (TTP), A would now contact the TTP for B's key. Since the TTP is trusted by everyone, A would believe that B's key is what the TTP provided, and that would be the end of the story. In this work, however, we do not assume the existence of any globally trusted entity: on the contrary, everything is up to the individual nodes of the network. They themselves sign certificates for each other's keys, and they themselves have to judge how much to trust these certificates and, essentially, their issuers. If A has had previous interactions with these issuing entities, then their public keys as well as their trustworthiness will be known to A, who will now decide whether to accept K_B as B's key or not. Otherwise, the same steps will have to be repeated to establish a trust relation with the issuing entities, recursively, until A can reach a decision, which could very well be that there is not (trustworthy) enough evidence to establish the relation. This is what the trust computation algorithm does (Sec 6.3.3), but in a forward way: A first computes trust values for his one-hop neighbors, then two-hop, and so on until the destination is reached (or, in the general case, until A has computed a trust value for all other entities).

Real life interactions build trust (or distrust) between some of the members of the network. In this way, what we call *direct* trust is created, and, since not all

members of a network have direct interactions, such direct trust links do not exist between all pairs. However, members without direct interactions will also need to make trust assessments for others, as in our example above. Trust computation deals with the calculation of these *indirect* trust relations.

Ultimately, we want to combine all relevant direct trust relations and associated weights to come up with an indirect trust value (weight). For this, first of all we assume that trust is in some sense transitive: If A directly trusts B (to some degree) and B directly trusts C (to some degree), then we can derive how much A indirectly trusts C (through B). Hence, we can talk about *trust paths* from a source (A in this case) to a destination (C in this case). These paths can be of any length, not just of length 2, as in this case. A further observation is that there could be multiple trust paths from A to C, through nodes other than B, and all these paths will be relevant for the trust computation.

Several approaches to trust computation have been proposed in the literature by [58], [29], [35], and others. Unfortunately, all these attempts have been made in a relatively ad-hoc fashion. With few exceptions, no researchers have compared their own approach to the others. As a result, someone – say, a network administrator – who believes that a notion of trust would be useful to incorporate in his administrative domain, has no easy way to choose which trust metric would be more suited to his needs.

In this chapter, we show how multiple trust computation algorithms can be seen from a common viewpoint. Their properties are formalized within an algebraic framework. The benefit of this approach is that it is much easier to see the differences

and similarities that exist. Moreover, it is easier to design an algorithm that satisfies the desired set of properties for a particular situation, since the relevant properties are clearly singled out. Finally, it is easier to implement and evaluate the algorithms under a common software solution which makes use of the common framework that all algorithms are instantiations of.

The rest of the chapter includes the description of our system model; detailed expositions of published algorithms under that model; requirements that the algorithms are intuitively expected to satisfy; and algebraic properties that the algorithms may or may not have. We interpret the algebraic properties in terms of practical implications. Finally, we emphasize proposals on evaluating the robustness of the algorithms to attacks by malicious adversaries.

5.2 General framework and description of algorithms

We will be dealing with what is called *recommendation trust* in the literature, as opposed to *direct trust*. In other words, we assume that direct trust has already been built between some pairs of users in the network through real life interactions or otherwise, as mentioned earlier. Then, our area of interest is the combination of these direct trust values into indirect ones. The term *recommendation* comes from the fact that the intermediate trust values can be seen as recommendations of users for other users. Also note that, in general, the recommendation trust values of two users A and A' about a user B will differ. Different users can have different opinions about the same user.

Our model of this situation is a directed graph $G = (V, E)$ with weights on the edges, the weight function being $w : e \longrightarrow S, e \in E$. The set S contains all possible trust values, usually from some minimum to some maximum value. The nodes of the graph correspond to the users, and the edges and edge weights correspond to the direct trust relations and the degree of trust associated with each relation. The set of *neighbors* of a user i, denoted N_i, consists of all nodes $j \in V$ such that a directed edge $e = (i, j)$ exists. We distinguish a *source* node s. The task of the algorithm that we present is to compute the source node's opinions (recommendation trust values) for every other node (user). That is, we want to come up with a single value in S for each user in the network. We denote s's recommendation trust value for user d by $t(s, d) \in S$, where $t : V \times V \longrightarrow S$.

Each of the algorithms that we present differs in the values and interpretation of edge weights (the set S), and the way the function $t(s, d)$ is computed from the graph and the weights. The unifying theme is the path interpretation that can be given to these computations. More specifically, we can define two operators that can be used to combine the available direct trust information. The first operator, which we will call *concatenation* operator and denote with the symbol $\otimes$, is used to combine trust values along a path from the source to the destination, as shown in Figure 5.1. More formally, consider a path p_1 from the source s to the destination d comprising of the edges $e_1 = (s, a_1), e_2 = (a_1, a_2), \ldots, e_k = (a_{k-1}, d)$. The recommendation trust value of s about d along the path p_1 is

$$t^{p_1} = w(e_1) \otimes w(e_2) \otimes \ldots \otimes w(e_k). \tag{5.1}$$

Note that it only makes sense to use the $\otimes$ operator for edge weights that are one after the other, i.e., form a directed path from the first to the last.

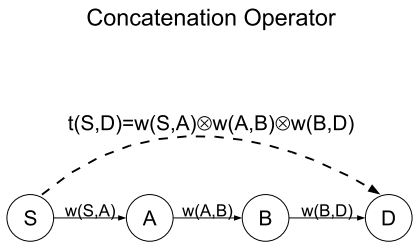

Figure 5.1: The concatenation operator $\otimes$ is used to combine opinions along a path.

The second operator, which we will call *summary* operator and denote with the symbol $\oplus$, is used to combine opinions computed along paths that start at the same node X, and end at the same node Y, i.e., paths that are, in a sense, parallel (see Figure 5.2). More formally, consider multiple paths $p_1, p_2, \ldots, p_n$ from the source s to the destination d with associated computed recommendation trust values $t^{p_1}(s,d), t^{p_2}(s,d), \ldots, t^{p_n}(s,d)$. The total recommendation trust is

$$t(s,d) = t^{p_1}(s,d) \oplus t^{p_2}(s,d) \oplus \ldots \oplus t^{p_n}(s,d). \tag{5.2}$$

Overall, we can write

$$t(s,d) = \bigoplus_{\text{path } p: s \rightsquigarrow d} t^p(s,d). \tag{5.3}$$

We now proceed to describe several trust computation algorithms within the framework that we have just built. For each algorithm, we will give the definition and interpretation of the weights, and the definition of the operators.

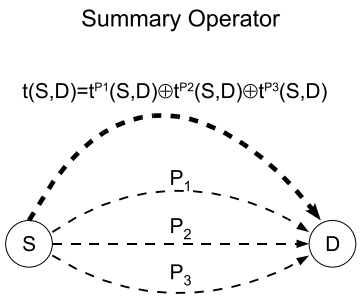

Figure 5.2: The summary operator $\oplus$ is used to combine opinions across paths.

5.2.1 Information Theoretic [57]

5.2.1.1 Entropy-Based

The weight is derived from a Bernoulli probability mass function, which, in effect, defines the probability p_{AB} that user B is trustworthy according to user A. The weight $w(A, B)$ is then computed as a function of the entropy of the Bernoulli distribution in the following way:

$$w(A,B) = \begin{cases} 1 - H(p_{AB}), & \text{for } 0.5 \leq p_{AB} \leq 1, \\ H(p_{AB}) - 1, & \text{for } 0 \leq p_{AB} \leq 0.5. \end{cases} \tag{5.4}$$

The entropy function is defined as $H(p) = -p \log_2 p - (1-p) \log_2 (1-p)$, and since $0 \leq p \leq 1$, we can see that $-1 \leq w(A, B) \leq 1$. So, in this case, the set S is $S = [-1, 1]$.

The concatenation operator is:

$$w(A,B) \otimes w(B,C) = w(A,B)w(B,C), \tag{5.5}$$

i.e., regular multiplication.

The summary operator is:

$$t^{p_1}(s,d) \oplus t^{p_2}(s,d) = \frac{w(e_1^{p_1})}{w(e_1^{p_1}) + w(e_1^{p_2})} t^{p_1}(s,d) + \frac{w(e_1^{p_2})}{w(e_1^{p_1}) + w(e_1^{p_2})} t^{p_2}(s,d), \tag{5.6}$$

i.e., a weighted sum, where the weight of each path is proportional to the trust value on the first edge of the path.

5.2.1.2 Probability-Based

The weight $w(A,B)$ in this case is a pair of numbers (p_{AB}, σ_{AB}). The number p_{AB} is the mean of a Beta probability distribution function, and σ_{AB} is the variance of this Beta distribution. It is interpreted as the confidence that user A has about the trust value p_{AB}, i.e., how certain A is that p_{AB} is an accurate estimate of the probability that B is trustworthy.

As an aside, a Beta pdf is often used in the literature to model how direct trust values appear from direct positive and negative experiences between two users. The connection is that if user A has a positive and b negative experiences with user B, the mean of the associated Beta pdf will be $\frac{a}{a+b}$ and the variance will be $\frac{ab}{(a+b)^2(a+b+1)}$. This will help make it easier to understand why the summary operator does what it does. However, we do not look into what alternatives exist to the generation of direct trust values. Our aim, as we have noted, is to compare alternatives to the computation of recommendation trust values.

The concatenation operator is:

$$w(A,B) \otimes w(B,C) = (p_{AB}, \sigma_{AB}) \otimes (p_{BC}, \sigma_{BC}) \tag{5.7}$$

$$= (p_{ABC}, \sigma_{ABC}), \tag{5.8}$$

where the two components are

$$p_{ABC} = p_{AB}p_{BC} + (1 - p_{AB})(1 - p_{BC}) \tag{5.9}$$

$$\begin{aligned} \sigma_{ABC} = p_{AB}\sigma_{BC} + \frac{1}{12}(1 - p_{AB}) \\ + p_{AB}(1 - p_{AB})(2p_{BC} - 1)^2 \end{aligned} \tag{5.10}$$

The summary operation is done through an intermediate transformation :

$$t^{p_1}(s,d) \oplus t^{p_2}(s,d) = (p_{sd}^{p_1}, \sigma_{sd}^{p_1}) \oplus (p_{sd}^{p_2}, \sigma_{sd}^{p_2}) \tag{5.11}$$

Each (p, σ) pair is transformed to an (a, b) pair. Then, the two pairs (a_1, b_1) and (a_2, b_2) are composed, and the result is transformed back to a (p, σ) pair.

$$(p, \sigma) \rightarrow (a, b) = \left(p(\tfrac{p(1-p)}{\sigma} - 1), (1 - p)(\tfrac{p(1-p)}{\sigma} - 1) \right) \tag{5.12}$$

$$(a_1, b_1) \oplus (a_2, b_2) = (a_1 + a_2 - 1, b_1 + b_2 - 1) \tag{5.13}$$

$$(a, b) \rightarrow (p, \sigma) = \left(\frac{a}{a+b}, \frac{ab}{(a+b)^2(a+b+1)} \right) \tag{5.14}$$

5.2.2 EigenTrust [30]

The weights in this case are real numbers between 0 and 1: $S = [0, 1]$. The weights are normalized on a user-per-user basis, i.e., for each user i, $\sum_{j \in N_i} w(i,j) = 1$.

The concatenation operator is:

$$w(A,B) \otimes w(B,C) = w(A,B)w(B,C), \tag{5.15}$$

i.e., regular multiplication.

The summary operator is:

$$t^{p_1}(s,d) \oplus t^{p_2}(s,d) = t^{p_1}(s,d) + t^{p_2}(s,d), \tag{5.16}$$

i.e., regular addition.

5.2.3 Probabilistic [39]

In this case the weights are treated exactly as probabilities: $S = [0,1]$. A weight $w(A,B)$ is interpreted to be the probability that the directed edge (A,B) exists. Then, the recommendation trust value of user s for user d is equal to the probability that there exists at least one directed path from s to d. We assume throughout that the probabilities on different edges are independent.

The concatenation operator is:

$$w(A,B) \otimes w(B,C) = w(A,B)w(B,C), \tag{5.17}$$

i.e., regular multiplication.

The summary operator is:

$$\begin{aligned} t^{p_1}(s,d) \oplus t^{p_2}(s,d) = t^{p_1}(s,d) + t^{p_2}(s,d) \\ -t^{p_1}(s,d)t^{p_2}(s,d), \end{aligned} \tag{5.18}$$

which is derived from the simple law of the probability of the union of two events: $P(A \cup B) = P(A) + P(B) - P(A \cap B) = P(A) + P(B) - P(A)P(B)$.

5.2.4 Multi-level [2]

The weights are discrete: $S = \{-1, 0, 1, 2, 3, 4\}$. The interpretation ranges from complete distrust (-1), to ignorance (0), to increasing levels of trust (1,2,3,4).

The concatenation operator is:

$$w(A, B) \otimes w(B, C) = \frac{w(A, B)}{4} \frac{w(B, C)}{4}, \tag{5.19}$$

i.e., multiplication of the values divided by 4. The division by 4 is presumably done to normalize the values to have a maximum equal to 1, but this is not explicitly stated in [2].

The summary operator is:

$$t^{p_1}(s, d) \oplus t^{p_2}(s, d) = \frac{1}{2} t^{p_1}(s, d) + \frac{1}{2} t^{p_2}(s, d), \tag{5.20}$$

i.e., averaging. This operator can be applied to multiple opinions at once, taking the average of all of them:

$$\begin{gathered} t^{p_1}(s, d) \oplus t^{p_2}(s, d) \oplus \ldots \oplus t^{p^n}(s, d) = \\ \frac{1}{n}(t^{p_1}(s, d) + t^{p_2}(s, d) + \ldots + t^{p_n}(s, d)). \end{gathered} \tag{5.21}$$

5.2.5 Subjective Logic [29]

The weights are ordered triplets of positive real numbers that sum to 1: $S = (b, d, u), b + d + u = 1, b, d, u \in [0, 1]$. These three numbers are called, respectively, *belief*, *disbelief*, and *uncertainty*.

The concatenation operator is:

$$\begin{aligned} w(A, B) \otimes w(B, C) &= (b_1, d_1, u_1) \otimes (b_2, d_2, u_2) \\ &= (b_1 b_2, b_1 d_2, d_1 + u_1 + b_1 u_2). \end{aligned} \tag{5.22}$$

The summary operator is:

$$\begin{aligned} t^{p_1}(s,d) \oplus t^{p_2}(s,d) &= (b_{sd}^{p_1}, d_{sd}^{p_1}, u_{sd}^{p_1}) \oplus (b_{sd}^{p_2}, d_{sd}^{p_2}, u_{sd}^{p_2}) \\ &= \left(\frac{b_{sd}^{p_1} u_{sd}^{p_2} + b_{sd}^{p_2} u_{sd}^{p_1}}{k}, \frac{d_{sd}^{p_1} u_{sd}^{p_2} + d_{sd}^{p_2} u_{sd}^{p_1}}{k}, \frac{u_{sd}^{p_1} u_{sd}^{p_2}}{k} \right), \end{aligned} \tag{5.23}$$

where $k = u_{sd}^{p_1} + u_{sd}^{p_2} - u_{sd}^{p_1} u_{sd}^{p_2}$.

5.2.6 Path-strength [34]

The weights are real numbers between 0 and 1: $S = [0, 1]$.

5.2.6.1 Strongest Path

The concatenation operator is the min operator:

$$w(A,B) \otimes w(B,C) = \min(w(A,B), w(B,C)). \tag{5.24}$$

The summary operator is the max operator:

$$t^{p_1}(s,d) \oplus t^{p_2}(s,d) = \max(t^{p_1}(s,d), t^{p_2}(s,d)). \tag{5.25}$$

5.2.6.2 Weighted Sum of Strongest Disjoint Paths

The concatenation operator is the min operator:

$$w(A,B) \otimes w(B,C) = \min(w(A,B), w(B,C)). \tag{5.26}$$

The summary operator is similar to the one in 5.2.1.1, a weighted sum of paths, where the weights are those of the first edges on each path. The difference is that now the paths are required to be disjoint.

5.2.7 Graph flows [35]

This and the next algorithm are based on arguments related to flows in graphs.

In this algorithm, the edge weights are viewed as capacities. A unit flow is sent out from the source s, and the trust value for the destination d is equal to the fraction of the flow that reaches the destination. Edge weights are again between 0 and 1: $S = [0, 1]$.

We can define the concatenation and summary operators as follows:

The concatenation operator is the min operator:

$$w(A, B) \otimes w(B, C) = \min(w(A, B), w(B, C)). \tag{5.27}$$

The summary operator is regular addition:

$$t^{p_1}(s, d) \oplus t^{p_2}(s, d) = t^{p_1}(s, d) + t^{p_2}(s, d). \tag{5.28}$$

5.2.8 Certificate Insurance [52]

In this algorithm, the weights are nonnegative real numbers: $S = [0, \infty)$. Again, the algorithm is a flow algorithm, just as the previous one. The same operators apply. However, the interpretation of the weights is different. Here, weights are expressed in monetary terms, in particular dollars. An edge (i, j) corresponds to a public key certificate that user i has issued for the public key of user j. The weight $w(i, j)$ is the amount of dollars that i agrees to pay to someone who believes that i's certificate is correct and later it turns out that it was not. This amount of dollars can be thought of as a kind of insurance.

5.3 Requirements for trust metrics

So far, we have listed several pairs of concatenation and summary operators. All of them are used to do the same calculation, that is, compute the recommendation trust value that a source node s should place on a destination node d. Based only on our intuition about this objective we can derive some conclusions about the desirable behavior of the operators. In this section we discuss some conditions that these operators should satisfy.

First of all, a user should not be able to unilaterally increase the source's recommendation trust value for the destination to a level higher than the source's trust value for the user himself. In other words, if s's trust in user i is $t(s,i) \in S$, then $t^{P_i}(s,d) \preceq t(s,i)$, where $\preceq$ is a partial order defined on S, and P_i is the set of paths from s to d that pass through i. The rationale is to avoid maliciously manipulated reports of trust values by users. A user's trust values about others cannot be trusted more than the user himself. At the most, a user can give the maximum trust value to everyone else, but even then, we do not want the source to increase its own trust values for everybody beyond some level.

On a related note, if s knows about d only through i, i.e., the path $s \rightsquigarrow i \rightsquigarrow d$ is the only path from s to d, then s cannot trust d more than how much i trusts d. Simply, there is no reason for s to be more optimistic than i's recommendation is. Continuing from the last paragraph, $t^{P_i}(s,d) \preceq t(i,d)$. Translating these considerations into a condition for the concatenation operator, we impose that trust should decrease along a path.

$$a \otimes b \preceq a, b, \quad a, b \in S. \tag{5.29}$$

We now come to the summary operator, which deals with aggregating recommendation trust values derived from different paths. Throughout the literature there is a prevailing notion that more independent paths are better than fewer [51]. More paths should in a sense be better than fewer paths, since more evidence is better than less evidence. It takes more malicious users to collaborate and subvert the trust computation algorithm, since at least one is needed on every path from the source to the destination.

But it is not the trust value that should increase; it is the confidence in the accuracy of the computed value, as long as, and to the extent that the trust values of different paths agree. In other words, if all recommendations agree that the destination is untrustworthy, then it stands to reason that the result of the summary operator should not increase the trustworthiness of the destination.

However, this conclusion is not always correct. What happens if some recommendations are positive and some are negative? Then two results are possible: One is that the confidence in the accuracy of the computed value should decrease, since the recommendations are in conflict. The other one is that the computed value should be the average of the recommendations (possibly weighted by the respective confidence values), while the confidence in its accuracy should increase.

Which one is more correct depends on the particular situation. If trust is ultimately assumed to be binary, i.e, a user is in reality either fully trustworthy or

fully untrustworthy, then the summary of conflicting opinions should decrease the confidence. This happens, for instance, when entities in the network are assumed to be divided in either friendly or enemy, with no gradation in between. On the other hand, trust can also be interpreted to be something that can legitimately be grey, as opposed to just black or white. It can be, for instance, the fraction of the time when a user has been seen to behave in a cooperative way, which is a quantity that can take any value between 0 and 1. In this case it is perfectly admissible to say that a user is trusted at a level of, say, 0.7. Therefore, two conflicting opinions could be reconciled by arguing that the two recommenders have seen different aspects of the behavior of the destination user. As a result, it would make sense to compute a summary value as the average of the two recommendations, and increase the total confidence value in the result.

So far, we have mentioned in passing a distinction between two concepts: trust and confidence. We defined trust to be an estimate of the behavior of a user, and confidence to be the accuracy of that estimate. However, not all the algorithms that we described incorporate the notion of confidence. As transpires from the discussion in this section, the usefulness of confidence is more apparent when there exist conflicting opinions. So, one situation that simplifies things is when conflicting opinions are not needed, or can be explicitly forbidden to exist. This saves us the trouble of having to deal with malicious users falsely accusing benign ones, but also prevents good users from notifying the network about potential misbehavior that they have noticed. If, for instance, trust values are used for access control decisions, then disallowing conflicting opinions amounts to disallowing revocation of privileges.

Whether this can be tolerated or not depends on the particular situation.

5.4 Algebraic properties of the operators

After going over the intuitive properties that we would like the operators to have, and the conditions under which we would like them to have those properties, we now proceed to the algebraic properties of these operators. The motivation for talking about algebraic properties is that they can be linked to issues with the numerical results that the computation returns. As a high level example to be elaborated on later, with certain operators it could happen that some edge weights are taken into account twice, which is clearly undesirable. Moreover, related to the algebraic properties are performance issues, as well as whether the computation can be done in a distributed manner or not.

First and foremost, both operators should be *closed* with respect to the set S, that is, if $a, b \in S$, then $a \otimes b \in S$, and $a \oplus b \in S$. The reason is that our admissible results are in the set S, and any value outside S has by definition no meaning in our computations. If S and $\otimes$ satisfy this property, then the pair $(S, \otimes)$ is called a *magma* or a *groupoid*. Similarly for the pair $(S, \oplus)$. Although this look like a fundamental property, some times it can be tricky to get right. For example, in one of the two proposed algorithms in [57], presented in Section 5.2.1.1, the summary operator is not closed for the set $S = [-1, 1]$. That summary operator is weighted averaging and it would create a problem in a situation where the following holds for

four users A, B, C, and D:

$$w(A,B) = 0.95 \qquad w(B,C) = 1 \tag{5.30}$$

$$w(A,D) = -0.9 \qquad w(D,C) = 1 \tag{5.31}$$

The algorithm would then compute

$$\begin{aligned} t(A,D) &= t^{ABC}(A,C) \oplus t^{ADC}(A,C) \\ &= w(A,B)w(B,C) \oplus w(A,D)w(D,C) \\ &= \tfrac{w(A,B)}{w(A,B)+w(A,D)} w(A,B)w(B,C) \\ &\quad + \tfrac{w(A,D)}{w(A,B)+w(A,D)} w(A,D)w(D,C) \\ &= \tfrac{0.95}{0.95-0.9} 0.95 \cdot 1 + \tfrac{-0.9}{0.95-0.9}(-0.9 \cdot 1) \\ &= 34.25 \notin S = [-1,1] \end{aligned} \tag{5.32}$$

A property for the summary operators, which is so natural, that is satisfied by all summary operators described above, is the *commutativity* property:

$$a \oplus b = b \oplus a, \forall a, b \in S \tag{5.33}$$

Remembering that a and b correspond to recommendation values derived from paths, we can see that it would not make sense to differentiate between, in effect, the names of the two paths when combining them. Commutativity makes the pair $(S, \oplus)$ a *commutative magma*.

We do not need commutativity for the $\otimes$ operator. This makes sense, since reversing the order of the edges of a path need not necessarily result in the same outcome. However, we note that some of the concatenation operators proposed are

indeed commutative (e.g. $\cdot$, min). This is not a problem, as long as the computation algorithm does not explicitly rely on the commutativity of this operator.

Another property of interest is associativity, and we would like both operators to be associative:

$$a \oplus (b \oplus c) = (a \oplus b) \oplus c \tag{5.34}$$

$$a \otimes (b \otimes c) = (a \otimes b) \otimes c, \forall a, b, c \in S \tag{5.35}$$

The reasoning is that the order in which the operator is applied should not matter. The justification in the case of the summary operator is that, if we have computed a recommendation trust value based on the currently available information, and then another path appears, we want to be able to just "add" together the information from the new path, and not do the whole computation from scratch. This pertains to the efficiency of the computation, especially when done over a network where information will arrive with different delays. Unfortunately, one intuitive summary operator – namely, averaging – is not associative in general:

$$\exists a, b, c \in S : \text{avg}(a, \text{avg}(b, c)) \neq \text{avg}(\text{avg}(a, b), c) \tag{5.36}$$

The averaging operator is intended to average over all available paths simultaneously, and not to include them one by one. However, in distributed environments, this leads to the problems just stated. One way to overcome this problem would be to count the number of paths already taken into account in the current result, so as to weight properly the current result and the new path value.

When it comes to the concatenation operator, associativity means that the final result should depend only on the order with which the edges appear in the path,

but not on the order with which we choose to do the calculations. Associativity is not satisfied by all the concatenation operators we have presented. For example, the operator presented in Section 5.2.1.2 is not associative. However, whether this is a significant problem or not depends on the actual computation algorithm and the way it is implemented distributedly.

As far as algebraic terminology goes, the pair $(S, \otimes)$ is now a *semigroup*, whereas the pair $(S, \oplus)$ is a *commutative semigroup*.

The last property we will consider is the *distributivity* (left and right) of $\otimes$ over $\oplus$:

$$\begin{aligned} a \otimes (b \oplus c) =& (a \otimes b) \oplus (a \otimes c) \\ (a \oplus b) \otimes c =& (a \otimes c) \oplus (b \otimes c) \end{aligned} \tag{5.37}$$

Distributivity is the most useful operation in terms of increasing the efficiency of computations. By inspection of (5.37), we see that the left sides need to compute two operations (one $\oplus$ in the parenthesis, and one $\otimes$ next). However, the right sides need three. This fact and the efficiency gains have been explored and discussed at length in the literature [3]. However, it seems to be the most difficult to satisfy.

We will just limit ourselves to discussing the flow-based metrics (presented in Sections 5.2.7 and 5.2.8) from the point of view of distributivity. The operators used there ($\otimes = \min, \oplus = +$) do not satisfy distributivity. For this reason, only if applied in a particular way will they return the correct (intended) result, which is the flow from the source to the destination. If all the paths from the source to the destination are edge-independent (share no common edges), then no particular way is needed. But if they are not independent, then the paths need to be decomposed

into successive segments comprising parallel (edge-independent) subpaths and common edges (a *series-parallel* decomposition). Then the summary operator will be applied to the edge-independent segments and then the concatenation operator will be applied to the successive segments. This could be repeated as needed. However, this cannot be done with all graphs, so other methods for computing flows should be used.

In the case of Subjective Logic (Sec. 5.2.5) distributivity is also not satisfied. It seems that where the series-parallel decomposition cannot be done, there is no way to do the computation, so some graphs can simply not be handled by that algorithm.

If the two operators satisfy all the properties that we have assigned to them so far, then the triplet $(S, \otimes, \oplus)$ is an algebraic structure called a *semiring*.

5.5 Conclusions

We have presented an algebraic framework that unifies many trust computation algorithms. By focusing on the algebraic properties of the algorithms under this framework, we are able to compare them in a much more rigorous way. We have shown links between the properties and considerations that can arise in practical implementations. As a result, we believe that a security practitioner can benefit from our exposition by adapting a particular metric to his own specifications, or even designing a new one.

Chapter 6

Semiring-Based Trust Computation Metrics

In this chapter, we present two novel trust metrics based on semirings. We also model PGP's web of trust computations as a semiring.

6.1 System Model

We view the trust computation problem as a generalized shortest path problem on a weighted directed graph $G(V, E)$ (*trust graph*). The vertices of the graph are the users/entities in the network. A weighted edge from vertex i to vertex j corresponds to the *opinion* that entity i, also referred to as the *issuer*, has about entity j, also referred to as the *target*. The weight function is $l(i, j) : V \times V \longrightarrow S$, where S is the opinion space.

Each opinion consists of two numbers: the *trust* value, and the *confidence* value. The former corresponds to the issuer's estimate of the target's trustworthiness. For example, a high trust value may mean that the target is one of the good guys, or that the target is able to give high quality location information, or that a digital certificate issued for the target's public key is believed to be correct. On the other hand, the confidence value corresponds to the accuracy of the trust value assignment. A high confidence value means that the target has passed a large number of tests that the issuer has set, or that the issuer has interacted with the target for a long

time, and no evidence for malicious behavior has appeared. Since opinions with a high confidence value are more useful in making trust decisions, the confidence value is also referred to as the *quality* of the opinion. The space of opinions can be visualized as a rectangle (ZERO_TRUST, MAX_TRUST)×(ZERO_CONF, MAX_CONF) in the Cartesian plane (Figure 6.1, for $S = [0, 1] \times [0, 1]$).

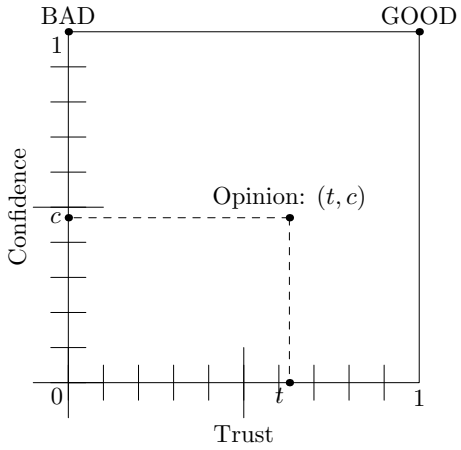

Figure 6.1: Opinion space

Both the trust and the confidence values are assigned by the issuer, in accordance with his own criteria. This means that a node that tends to sign public key certificates without too much consideration will often give high trust and high confidence values. The opposite holds true for a strict entity. When two such entities interact, it is important for the stricter entity to assign a low enough trust value to the less strict one. Otherwise, the less strict entity may lead the stricter one to undesirable trust decisions. This situation is easier to picture in the context of Certification Authorities and public key certification. In that context, a certification authority A will only give a high trust value to B, if B's policy for issuing certificates is at least as strict as A's and has the same durability characteristics [21].

In the framework described, two versions of the trust inference problem can be formalized. The first is finding the trust-confidence value that a source node A should assign to a destination node B, based on the intermediate nodes' trust-confidence values. Viewed as a generalized shortest path problem, it amounts to finding the generalized distance between nodes A and B. The second version is finding the most trusted path between nodes A and B. That is, find a sequence of nodes $\langle v_0 = A, v_1, \ldots, v_k = B \rangle : (v_i, v_{i+1}) \in E, 0 \leq i \leq k-1$ that has the highest aggregate trust value among all trust paths starting at A and ending at B.

Both problems are important: finding a target's trust value is needed before deciding whether to grant him access to one's files, or whether to disclose sensitive information, or what kind of orders he is allowed to give (in a military scenario, for instance). With this approach, a node will be able to rely on other nodes' past experiences and not just his own, which might be insufficient. The second problem is more relevant when it comes to actually communicating with a target node. The target node being trustworthy is one thing, but finding a trusted path of nodes is needed, so that traffic is routed through them. Note that in the usual shortest path problem in a graph finding the distance between two nodes, simultaneously finds the actual shortest path. In the trust case, we will usually utilize multiple trust paths to compute the trust distance from the source to the destination, since that will increase the evidence on which the source bases its final estimate. The first problem is addressed with what we call "Distance semiring" (Sec 6.3.2), and the second with the "Path semiring" (Sec 6.3.1).

The core of our approach is the two operators used to combine opinions: One

operator (denoted $\otimes$) combines opinions along a path, i.e. A's opinion for B is combined with B's opinion for C into one indirect opinion that A should have for C, based on B's recommendation. The other operator (denoted $\oplus$) combines opinions across paths, i.e. A's indirect opinion for X through path p_1 is combined with A's indirect opinion for X through path p_2 into one aggregate opinion. Then, these operators can be used in a general framework for solving path problems in graphs, provided they satisfy certain mathematical properties, i.e. form an algebraic structure called a semiring. More details on this general framework are in Appendix A. An existing trust computation algorithm (PGP [61]) is modeled as operating on a particular semirings. Note that our approach differs from PGP in that it allows the user to infer trust values for unknown users/keys. That is, not all trust values have to be directly assigned by the user making the computations. The operators are discussed in greater depth in section 6.3.

6.2 Semirings as a model for trust computations

In order to show the modeling power of this framework, we now model PGP's web of trust computations [61] as a semiring. Remember that PGP computes the validity of an alleged key-to-user binding, as seen from the point of view of a particular user, henceforth called the source. The input to the computation algorithm consists of three things: The source node, the graph of certificates issued by users for each other, and the trust values for each user as assigned by the source. Note that the validity of all key-to-user bindings has to be verified, since only certificates

signed by valid keys are taken into account, and any certificate may influence the validity of a key-to-user binding.

The validity of the key-to-user binding for user i will be deduced from the vector $d_i \in \mathbb{N}^k$, where k is the number of different trust levels defined by PGP. It seems that k is 4 ("unknown", "untrusted", "marginally trusted", "fully trusted"), but some include a fifth level : "ultimately trusted". Our analysis is independent of the exact value of k. The vector d_i will hold the number of valid certificates for user i that have been signed by users of each trust level. For example, $d_i = (0, 1, 2, 3)$ means that one "untrusted", two "marginally trusted", and three "fully trusted" users have issued certificates for user i's public key. In addition, all six of these certificates are signed by valid keys, i.e. keys for which the key-to-user binding has been verified.

In order to verify the actual validity of the binding, we will use the function $\texttt{val} : \mathbb{N}^k \rightarrow \mathbb{V}$, where $\mathbb{V}$ is the space of admissible results. For simplicity, we will be assuming that $\mathbb{V} =$ {"invalid", "valid"}, although values such as "marginally valid" have also been proposed. The output of `val` for a specific input is determined by thresholds such as: "A key-to-user binding is valid if at least two "marginally trusted" users have issued a certificate for it". These thresholds are incorporated in `val` and will be transparent to our analysis. Finally, for computation simplicity we will be assuming that $\mathbb{V} = \{0, 1\}$, where "invalid"$= 0$, and "valid"$= 1$.

The edge weights $w_{ij} \in \mathbb{N}^k, 1 \leq i, j \leq n$, where n is the number of users, correspond to the certificate from i about j's alleged public key. A weight can only have one of $k + 1$ possible values. Either it consists only of 0s, or of exactly $k - 1$

0s and one 1. An all-zero weight means that there is no certificate from i about j's key. An 1 in the position that corresponds to trust level t means that the source has assigned trust level t to i, and i has issued a certificate for j.

The $\otimes$ operator is defined as follows ($a, b \in \mathbb{N}^k$):

$$a \otimes b = \texttt{val}(a)b \in \mathbb{N}^k$$

The $\oplus$ operator is defined exactly as vector addition in $\mathbb{N}^k$.

Verification of the semiring properties

For $\otimes$, the absorbing element is $\textcircled{0} = (0, \ldots, 0) \in \mathbb{N}^k$, and the neutral element is $\textcircled{1} = \{x \in \mathbb{N}^k : \texttt{val}(x) = 1\}$. That is, all such vectors are mapped to $\textcircled{1}$; for our purposes, they are equivalent. It is trivial to prove that $\textcircled{0}$ is a neutral element for $\oplus$.

The $\otimes$ operator is associative:

$$a \otimes (b \otimes c) = a \otimes (\texttt{val}(b)c) = \texttt{val}(a)\texttt{val}(b)c$$

$$(a \otimes b) \otimes c = (\texttt{val}(a)b) \otimes c = \texttt{val}(\texttt{val}(a)b)c$$

and these two are equal because $\texttt{val}(\textcircled{0})$=0.

The $\oplus$ operator is commutative and associative, because it is vector addition.

The $\otimes$ operator distributes over $\oplus$:

$$a \otimes (b \oplus c) = \texttt{val}(a)(b + c)$$

$$(a \otimes b) \oplus (a \otimes c) = \texttt{val}(a)b + \texttt{val}(a)c$$

The computation algorithm below uses the above semiring to compute the validity or otherwise of all keys in the certificate graph G. The source node is s and the function w maps edges to edge weights.

PGP-SEMIRING-CALCULATION(G, w, s)

```
1  for i ← 1 to |V|
2        do d[i] ← ⓪
3  d[s] ← ①
4  S ← {s}
5  while S ≠ ∅
6        do u ← DEQUEUE(S)
7           for each v ∈ Neighbors[u] with val(d[v]) = 0
8                 do
9                    d[v] ← d[v] ⊕ (d[u] ⊗ w(u, v))
10                   if val(d[v]) = 1
11                      then ENQUEUE(S, v)
```

The computation starts at the source s, and progressively computes the validity of all keys reachable from s in the certificate graph. The queue S contains all valid keys for which the outgoing edges (certificates signed with these keys) have not been examined yet. When a key is extracted from S, its certificates to other keys are examined, and their d-vectors are updated. Only certificates to so-far-invalid keys are examined, since adding a certificate to the d-vector of a key already shown to be valid is redundant. If a so-far-invalid key obtains enough certificates to be-

come valid, it is added to the queue for future examination. Each key is enqueued at most once (when it becomes valid), and all keys in the queue are eventually dequeued. Ergo, the algorithm terminates. After termination, all valid keys have been discovered.

Note that if s is only interested in the validity of a particular key-to-user binding, then the algorithm can stop earlier: as soon as its validity is determined, or after all certificates for that key have been examined.

6.3 Trust Semirings

6.3.1 Path semiring

In our first semiring, the opinion space is $S = [0,1] \times [0,1]$ Our choice for the $\otimes$ and $\oplus$ operators is as follows (Figure 6.2):

$$(t_{ik}, c_{ik}) \otimes (t_{kj}, c_{kj}) = (t_{ik}t_{kj}, c_{ik}c_{kj}) \tag{6.1}$$

$$(t_{ij}^{p_1}, c_{ij}^{p_1}) \oplus (t_{ij}^{p_2}, c_{ij}^{p_2}) = \begin{cases} (t_{ij}^{p_1}, c_{ij}^{p_1}) & \text{if } c_{ij}^{p_1} > c_{ij}^{p_2} \\ (t_{ij}^{p_2}, c_{ij}^{p_2}) & \text{if } c_{ij}^{p_1} < c_{ij}^{p_2} \\ (t_{ij}^{*}, c_{ij}^{p_1}) & \text{if } c_{ij}^{p_1} = c_{ij}^{p_2} \end{cases}, \tag{6.2}$$

where $(t_{ij}^{p_1}, c_{ij}^{p_1})$ is the opinion that i has formed about j along the path p_1, and $t_{ij}^{*} = \max(t_{ij}^{p_1}, t_{ij}^{p_2})$.

Since both the trust and the confidence values are in the $[0,1]$ interval, they both decrease when aggregated along a path. When opinions are aggregated across paths, the one with the highest confidence prevails. If the two opinions have equal

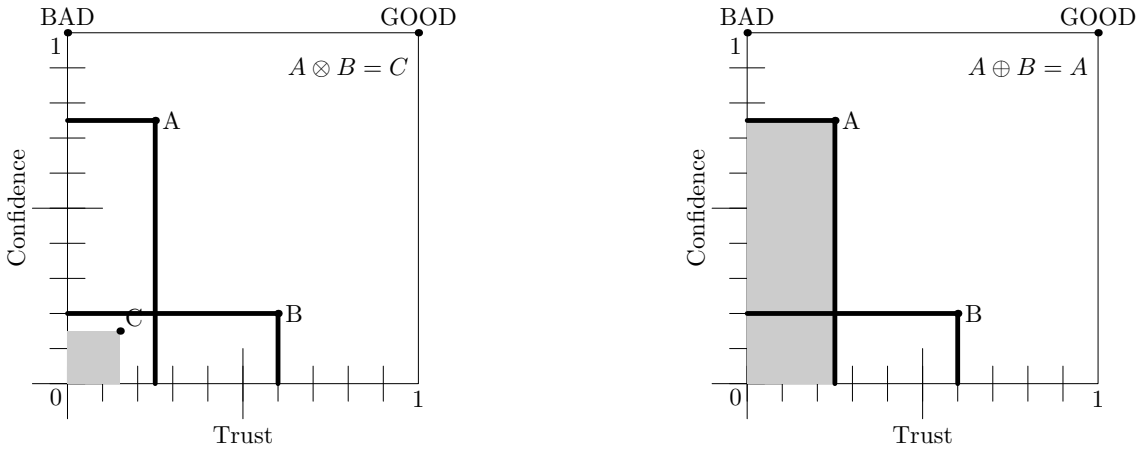

Figure 6.2: $\otimes$ and $\oplus$ operators for the Path semiring

confidences but different trust values, we pick the one with the highest trust value. We could have also picked the lowest trust value; the choice depends on the desired semantics of the application.

This semiring essentially computes the trust distance along the most confident trust path to the destination. An important feature is that this distance is computed along a single path, since the $\oplus$ operator picks exactly one path. Other paths are ignored, so not all available information is being taken into account. One of the advantages is that if the trust value turns out to be high, then a trusted path to the destination has also been discovered. Also, fewer messages are exchanged for information gathering.

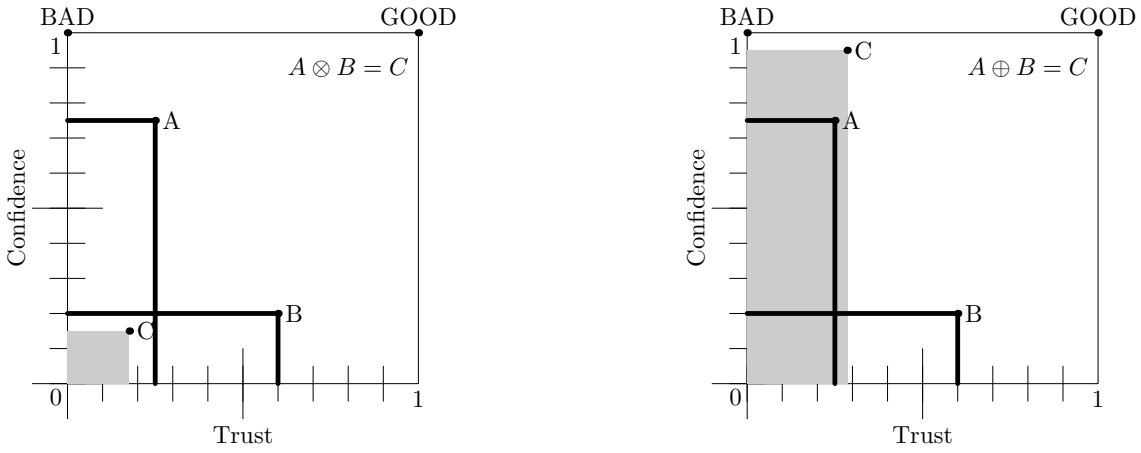

Figure 6.3: $\otimes$ and $\oplus$ operators for the Distance semiring

6.3.2 Distance semiring

Our second proposal, the distance semiring, is based on the *Expectation semiring* defined by Eisner in [20], and used for speech/language processing:

$$\begin{aligned}(a_1, b_1) \otimes (a_2, b_2) &= (a_1 b_2 + a_2 b_1, b_1 b_2) \\ (a_1, b_1) \oplus (a_2, b_2) &= (a_1 + a_2, b_1 + b_2)\end{aligned}$$

The opinion space is $S = [0, \infty] \times [0, 1]$. Before using this semiring, the pair (trust, confidence)=(t, c) is mapped to the weight $(c/t, c)$. The motivation for this mapping becomes clear when we describe its effect on the results of the operators. The binary operators are then applied to this weight, and the result is mapped back to a (trust, confidence) pair. For simplicity, we only show the final result without the intermediate mappings.

$$\begin{aligned}(t_{ik}, c_{ik}) \otimes (t_{kj}, c_{kj}) &\rightarrow \left(\frac{1}{\frac{1}{t_{ik}} + \frac{1}{t_{kj}}}, c_{ik} c_{kj} \right) \\ \left(t_{ij}^{p_1}, c_{ij}^{p_1}\right) \oplus \left(t_{ij}^{p_2}, c_{ij}^{p_2}\right) &\rightarrow \left(\frac{c_{ij}^{p_1} + c_{ij}^{p_2}}{\frac{c_{ij}^{p_1}}{t_{ij}^{p_1}} + \frac{c_{ij}^{p_2}}{t_{ij}^{p_2}}}, c_{ij}^{p_1} + c_{ij}^{p_2} \right)\end{aligned}$$

So, when aggregating along a path, both the trust and the confidence decrease. The component trust values are combined like parallel resistors. Recall that two resistors in parallel offer lower resistance than either of them in isolation. Also, a zero trust value in either opinion will result in a zero trust value in the resulting opinion (absorbing element), while a trust value equal to infinity will cause the corresponding opinion to disappear from the result (neutral element). On the other hand, the component confidence values are between 0 and 1, and they are multiplied, so the resulting confidence value is smaller than both.

When aggregating across paths, the total trust value is the weighted harmonic average of the components, with weights according to their confidence values. So, the result is between the two component values, but closer to the more confident one. Note, also, the behavior caused by extreme (zero or infinity) trust values: A zero trust value dominates the result (unless its corresponding confidence is zero); a trust value equal to infinity results in an increase in the trust value given by the other opinion. In order for the resulting trust value to be the maximum possible, both opinions have to assign the maximum. So, in general, we can say that this operator is conservative. A zero confidence value (neutral element) causes the corresponding opinion to disappear from the result.

6.3.3 Computation algorithm

The algorithm below, due to Mohri [40], computes the $\oplus$-sum of all path weights from a designated node s to all other nodes in the trust graph $G = (V, E)$.

Revisiting the illustrative example described in the Introduction, remember that entity A wanted to judge the validity of entity B's public key based on certificates signed by other entities in the network. Using the trust computation algorithm, A will compute those entities' trustworthiness. If they are trustworthy enough, then A will believe their certificates are true, and will accept B's key. If not, A will not accept it. Of course, the above assumes that A already knows the public keys of the entities that issued the certificates for B. If that is not the case, then the whole process will be repeated for the public key of each one of the unknown issuers.

GENERIC-SINGLE-SOURCE-SHORTEST-DISTANCE(G, s)

```
1  for i ← 1 to |V|
2        do d[i] ← r[i] ← ⓪
3  d[s] ← r[s] ← ①
4  S ← {s}
5  while S ≠ ∅
6        do q ← head(S)
7           DEQUEUE(S)
8           r' ← r[q]
9           r[q] ← ⓪
10          for each v ∈ Neighbors[q]
11                do if d[v] ≠ d[v] ⊕ (r' ⊗ w[(q, v)])
12                      then d[v] ← d[v] ⊕ (r' ⊗ w[(q, v)])
13                           r[v] ← r[v] ⊕ (r' ⊗ w[(q, v)])
14                           if v ∉ S
15                              then ENQUEUE(S, v)
16 d[s] ← ①
```

This is an extension to Dijkstra's algorithm [18]. S is a queue that contains the vertices to be examined next for their contribution to the shortest path weights. The vector $d[i], i \in V$ holds the current estimate of the shortest distance from s to i. The vector $r[i], i \in V$ holds the total weight added to $d[i]$ since the last time i was extracted from S. This is needed for non-idempotent semirings, such as the one proposed. Its computational complexity depends on the semiring used, and also

on the actual topology of the network. As the reader can see in [40], the crucial parameter of the topology is the number of paths from the source to the other nodes. So, the more sparse the network, the more efficient the algorithm. But, in any case, the algorithm can be executed in a distributedfashion just like OSPF [43] with local data exchanges only.

Our computation algorithm is based on Mohri's, but with three adjustments which are needed when considering the problem from the perspective of trust. Lines 11-13 of the algorithm will be referred to as "node q votes for node v".

First of all, some nodes may be prevented from voting. Only if a node's trust value exceeds a predefined trust threshold, is the node allowed to vote. This is motivated from the common sense observation that only good nodes should participate in the computation, and bad nodes should be barred. Note that there is no restriction on the corresponding confidence. This will initially lead to bad nodes being allowed to vote, but after some point they will be excluded since good nodes will acquire evidence for their maliciousness.

Second, no node is allowed to vote for the source (s). Since it is s that initiates the computation, it does not make sense to compute s's opinion for itself.

Third, no cyclic paths are taken into account. If that were the case, we would be allowing a node to influence the opinion about itself, which is undesirable. Unfortunately, there is no clear way to discard any single edge-opinion of the cycle. So, the approach taken is to discard any edges that would form a cycle if accepted. As a result, the order in which the voters are chosen in line 6 is important. We argue that it makes sense to choose the node for which the confidence is highest.

These adjustments introduce characteristics from the Path semiring into the Distance semiring. For example, the node with the maximum confidence gets to vote first. Moreover, some paths are pruned which means that fewer messages are exchanged, thus saving bandwidth, but also some of the existing information is not taken into account.

6.4 Conclusion

We have presented a scheme for evaluating trust evidence in Ad-Hoc networks. Our scheme is entirely based on information originating at the users of the network. No centralized infrastructure is required, although the presence of one can certainly be utilized. Also, users need not have personal, direct experience with every other user in the network in order to compute an opinion about them. They can base their opinion on second-hand evidence provided by intermediate nodes, thus benefitting from other nodes' experiences. Of course, we are taking into account the fact that second-hand (or third, or fourth...) evidence is not as valuable as direct experience. In this sense, our approach extends PGP, since PGP only uses directly assigned trust values.

At each round of computation, the source node computes opinions for all nodes. This means that information acquired at a single round can be stored and subsequently used for many trust decisions. If there is not enough evidence to determine an opinion, then no opinion is formed. So, when malicious nodes are present in the network they cannot fool the system into accepting a malicious node

as benevolent. A failsafe state exists that ensures graceful degradation as the number of adversaries increases.

Chapter 7

Attack Resistance of Trust Metrics

In this chapter, we address the issue of attack resistance of semiring trust metrics. In particular, we describe an algorithm that computes efficiently the effect of changing an arbitrary direct opinion on the computation of a source-destination indirect opinion. With this algorithm we can measure the effect of an attacker deleting an opinion, manipulating an opinion's value, inserting a completely bogus opinion, or even deleting all the opinions of a user by, e.g., destroying the network node operated by that user.

7.1 Introduction

Levien and Aiken, in [35], suggested a criterion for measuring the resistance of a trust metric to attackers. First, they distinguished between two types of attacks: node attacks, and edge attacks. Node attacks amount to a certain node being impersonated. So, the attacker can issue any number of arbitrary opinions (public key certificates in Levien's case) from the compromised node about any other node. Edge attacks are more constrained: Only one false opinion can be created per each attack. In other words, an attack of this type is equivalent to inserting a false edge in the trust graph. Obviously, a node attack is the more powerful of the two, since it permits the insertion of an arbitrary number of false edges.

The attack resistance of a metric can be gauged by the number of node or edge attacks that are needed before the metric can be manipulated beyond some threshold. For instance, it has been shown [52] that a single misbehaving entity (a 1-node attack) can cause the metric proposed in [9] to return an arbitrary result.

In this chapter, we will deal with single edge attacks on those semiring trust metrics where the aggregation operator $\oplus$ has the property that $a \oplus b = a$, or $a \oplus b = b$. For example, $\oplus$ could be max, or it could be the aggregation operator in our Path semiring (see Chapter 6), which returns the opinion with the maximum confidence value.

We extend the semiring trust computation framework with an algorithm that, given a source-destination node pair s and d, computes how much the indirect trust between s and d changes if an arbitrary direct opinion changes (either improves or deteriorates). From this, we can find which edge, if attacked, can cause the greatest change in the computed indirect trust. In the case of the Path semiring for trust, the same path that is used for the trust computation can be used for routing traffic from the source to the destination. So, a change in a direct opinion may cause the traffic to be redirected through a different path. We would then be interested to know how much the weight (direct opinion) of an arbitrary link can change without changing the optimal path between the source and destination. This is called *sensitivity analysis* of optimal paths. The bounds in the directions of improvement and deterioration that we get for each edge are called the *upper* and *lower* tolerance of that edge, respectively, and these are defined with respect to the source-destination path in question.

We would like to emphasize that this algorithm works for any semiring whose aggregation operator satisfies the property mentioned earlier ($a \oplus b = a$ or b). In particular, it works for semirings used for Quality of Service (QoS) routing, such as minimum delay, maximum capacity, and widest-shortest path routing [59].

Algorithms for upper and lower tolerances for shortest paths and for maximum capacity paths were given in [49] by reducing these problems to the same problem, called Minimum Cost Interval problem. In [26] an algorithm was given that computes how much the length of the shortest path changes, when an arbitrary link disappears. We use a generalized version of this algorithm when computing lower edge tolerances. An apparently equivalent problem, that computes how much a shortest path changes when a *node* disappears, was addressed in [45].

The practical significance of sensitivity analysis and of computing upper and lower tolerances is varied. From a security point of view, an attacker may want to cause the maximum trust manipulation by affecting a single opinion (edge attack). The manipulation could be aiming either to artificially improve the source's opinion for an untrustworthy target, or, conversely, to artificially deteriorate the source's opinion for a trustworthy one. It follows that a defender should pay extra attention to the opinion that could cause the greatest damage. Further, if a network administrator wants to achieve a trust increase between two nodes by improving (or adding) one trust opinion link, then the upper tolerances would be useful: they give insight to the effect that trust link improvements and additions have. As an example, consider what happens when two wireless network users are physically close, but have no previous interactions, and their indirect trust is insignificant. However,

they happen to be at strategic locations with regard to network connectivity by providing, e.g., a shortcut. Then, an administrator could intervene and provide the two users with an exogenous trust certificate that would immediately build trust between the two users, thus improving the network.

Computing tolerances is also relevant for judging the stability of routing paths, to the extent that routing is done through the most trusted path. If a link weight is close to its upper or lower tolerance, then it is more likely that a fluctuation in the link weight will cause the corresponding optimal path to change. This is undesirable, and a network administrator or designer would benefit from knowing it could happen.

The relevance of tolerances also extends to traffic redirection attacks, discussed in Section 7.3, which in wireless ad-hoc networks have been called wormhole attacks [27]. In this case, the upper tolerances of edges are giving the thresholds after which traffic will be redirected to the newly "improved" edges. An attacker may try to attract traffic to the links controlled by him. We can show that some edges can never be improved enough to attract traffic from a particular source-destination pair. Alternatively, the attacker may break certain links so as to force traffic to pass through compromised network elements (links and routers).

The rest of the chapter is organized as follows. Section 7.2 introduces two essential algebraic quantities that are at the heart of the problem, together with the algorithms that compute them. It then discusses how to get the tolerances from these quantities. In Section 7.3 we provide an example on how the traffic redirection attacks would work. The mathematical terminology is provided in Appendix A.

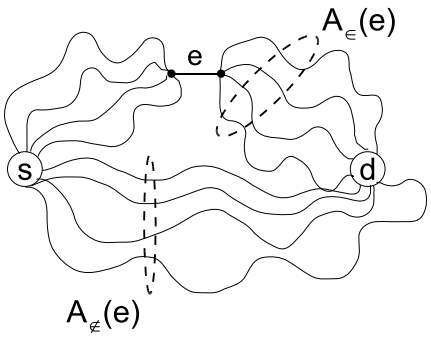

Figure 7.1: Partitioning of paths according to whether they use edge e or not.

7.2 Edge Sensitivities for the Algebraic Path Problem

We repeat that what we want to compute is how much the optimal $s \rightsquigarrow d$ path weight changes when the weight of an edge $e = (i, j)$ changes. Having fixed a source vertex s and a destination vertex d, we call A the set of all $s \rightsquigarrow d$ paths. We partition the set A into two subsets $A_{\in}(e), A_{\notin}(e)$ according to whether a path includes edge e or not (Figure 7.1):

$$A_{\in}(e) = \{p \in A | e \in p\} \tag{7.1}$$

$$A_{\notin}(e) = A \setminus A_{\in}(e) \tag{7.2}$$

By a slight abuse of notation, we use $w(X)$, where X is a set of paths with the same source and the same destination, to mean the $\oplus$-sum of the weights of the paths in X. So, we have that

$$w(A) = w(A_{\in}(e)) \oplus w(A_{\notin}(e)). \tag{7.3}$$

We will now proceed to compute the weight $w(A_{\in}(e))$, which we call *total e-path weight*, and $w(A_{\notin}(e))$, which we call *total non-e-path weight*.

7.2.1 Computation of total e-path weight

We write $w(A_\in(e))$ in the following way:

$$w(A_\in(e)) = \left[\left(\bigoplus_{s \overset{p}{\rightsquigarrow} i} w(p)\right) w(e) \left(\bigoplus_{j \overset{p}{\rightsquigarrow} d} w(p)\right)\right] \oplus \left[\left(\bigoplus_{s \overset{p}{\rightsquigarrow} j} w(p)\right) w(e) \left(\bigoplus_{i \overset{p}{\rightsquigarrow} d} w(p)\right)\right] \quad (7.4)$$

What we have done here is that we have split $A_\in(e)$ according to whether a path crosses edge $e = (i, j)$ in the direction from vertex i to vertex j or vice versa. Note that this is not necessarily a partition of $A_\in(e)$, since a path may cross edge e in both directions. However, adding more edges never improves a path (by property (A.4)), so the optimal path will never cross an edge twice, which in turn means that we can disregard all edge-repeating paths. As a result, we can assume without loss of generality that in (7.4) no path will appear inside both square brackets.

Assuming that the semiring R is commutative, we can factor out $w(e)$ in (7.4), and have

$$w(A_\in(e)) = w(e)w(B(e)) \quad (7.5)$$

where

$$w(B(e)) = \left(\bigoplus_{s \overset{p}{\rightsquigarrow} d, w(e) \leftarrow ①} w(p)\right) \quad (7.6)$$

which is more usefully written as

$$w(B(e)) = \left(\bigoplus_{s \overset{p}{\rightsquigarrow} i} w(p)\right) ① \left(\bigoplus_{j \overset{p}{\rightsquigarrow} d} w(p)\right) \oplus \left(\bigoplus_{s \overset{p}{\rightsquigarrow} j} w(p)\right) ① \left(\bigoplus_{i \overset{p}{\rightsquigarrow} d} w(p)\right) \quad (7.7)$$

So we have isolated the weight $w(e)$. We want to be able to compute, for any arbitrary new value of $w(e)$ given to us, the new optimal path weight efficiently for all edges e. To this end we compute the optimal-path tree rooted at s (the union of optimal paths from s to all other nodes), and the optimal-path tree rooted at d. These unions of paths are trees since we have assumed that the optimal path between a pair of nodes is unique. Denote by $d(s,i), d(j,d)$ the weight of the optimal path from s to i, and from j to d, respectively. Note that $d(j,d) = d(d,j)$, since the graph is undirected. Then, all we need to do for each edge is to compute

$$w(A_\in(e)) = w(e)\,(d(s,i)d(j,d) \oplus d(s,j)d(i,d))\,, \tag{7.8}$$

which is (7.5) combined with (7.7).

The computation of the shortest path trees can be done with a generalized version of Dijkstra's algorithm [18], which uses $\oplus$ instead of min, and $\otimes$ instead of $+$, shown in Figure 7.2. The generalized version works, as long as ① is the largest element of the semiring [53]. The complexity remains the same, i.e., $O(m + n\log n)$, when Fibonacci heaps are used for the priority queue needed in Dijkstra's algorithm. Obviously, we also need to assume that the semiring operations take constant time.

To summarize, in order to compute $w(A_\in)(e)$, we first compute the two optimal path trees rooted at s and d. Then, for each edge $e = (i,j)$ and weight $w(e)$, all we need to do is to look up the four values $d(s,i)$, $d(j,d)$,$d(s,j)$, $d(i,d)$, and substitute them into (7.8).

GENERALIZED-DIJKSTRA(G, s)

```
1  for i ← 1 to |V|
2        do d[i] ← ⓪
3  d[s] ← ①
4  S ← {s}
5  while S ≠ ∅
6        do q ← EXTRACTFROM(S)
7           for each v ∈ Neighbors[q]
8                 do if d[v] ≠ d[v] ⊕ (d[q] ⊗ w(q, v))
9                       then d[v] ← d[q] ⊗ w(q, v)
10                           if v ∉ S
11                              then ENQUEUE(S, v)
12 d[s] ← ①
```

Figure 7.2: Generalized version of Dijkstra's algorithm, where $\oplus$ replaces min and $\otimes$ replaces $+$.

7.2.2 Computation of total non-e-path weight

We turn to the computation of the quantity $w(A_{\notin}(e))$, which is the sum of the weights of all $s \rightsquigarrow d$ paths that do not go through edge e. Equivalently, it is the weight of the optimal such path.

$$w(A_{\notin}(e)) = \bigoplus_{s \overset{p}{\rightsquigarrow} d, w(e)=\mathbb{0}} w(p). \tag{7.9}$$

This equality holds, because $\mathbb{0}$ is absorbing for $\otimes$ so setting $w(e) = \mathbb{0}$ is equivalent to disregarding edge e for the purposes of calculating the optimal path. Any path that goes through that edge will have a weight equal to $\mathbb{0}$, so it will never be optimal.

Notice that for all $e \notin P^*$, we do not need to do any extra computations since $w(A_{\notin}(e)) = d(s, d)$. To compute $w(A_{\notin}(e))$ efficiently for the edges $e \in P^*$, we use the Generalized Edge Replacement algorithm described in Figure 7.4. The algorithm is due to Hershberger and Suri. We adapt the algorithm for semiring operators and provide a short explanation. For a more detailed explanation, the reader is referred to [26].

The main idea of the algorithm is to break up the summation in the way shown in (7.10) below, and make some simplifying assumptions that enable efficient computation of the quantities involved. However, one of these simplifying assumptions causes the algorithm not to find the optimal solution in all graphs. Hershberger and Suri stated that their algorithm works for undirected graphs but not for directed. However, we show here that it works in some class of directed graphs, too, and provide the properties that such directed graphs should satisfy.

$$\bigoplus_{s \overset{p}{\rightsquigarrow} d, w(e)=\textcircled{0}} w(p) = \bigoplus_{(u,v)\in E\setminus e} \left\{ \left(\bigoplus_{s \overset{p}{\rightsquigarrow} u, w(e)=\textcircled{0}} w(p) \right) \otimes w(u,v) \otimes \left(\bigoplus_{v \overset{p}{\rightsquigarrow} d, w(e)=\textcircled{0}} w(p) \right) \right\} \tag{7.10}$$

Define the property $\mathbf{P_1}(u)$ of a vertex $u \in V$ to be true when and only when the optimal path from s to u does not contain edge e. For the vertices u that satisfy $\mathbf{P_1}$,

$$\bigoplus_{s \overset{p}{\rightsquigarrow} u, w(e)=\textcircled{0}} w(p) = \bigoplus_{s \overset{p}{\rightsquigarrow} u} w(p). \tag{7.11}$$

Similarly, define the property $\mathbf{P_2}(v)$ of a vertex $v \in V$ to be true when and only when the optimal path from v to d does not contain edge e. For the vertices u that satisfy $\mathbf{P_2}$,

$$\bigoplus_{v \overset{p}{\rightsquigarrow} d, w(e)=\textcircled{0}} w(p) = \bigoplus_{v \overset{p}{\rightsquigarrow} d} w(p). \tag{7.12}$$

We like pairs of nodes (u, v) that have these two properties, because for them we do not need to recompute the optimal distances from s and to d. Since the corresponding optimal paths do not use edge e, it does not make a difference whether edge e is deleted or not. For these nodes, we can reuse the optimal-path trees computed earlier for the original graph.

The optimal path in the graph after the deletion of edge e (equivalently, after setting $w(e)$ to $\textcircled{0}$) has the following form: It starts at s, goes through nodes that satisfy $\mathbf{P_1}$ until node u, follows an edge $(u, v) \neq e$, and all remaining nodes from v to d satisfy property $\neg\mathbf{P_1}$. In other words, as soon as the optimal path leaves nodes that satisfy $\mathbf{P_1}$, it can never return. So, we do not need to do the external summation over all edges in $E \setminus e$, but only over the edges in the set $\{(u, v) | \mathbf{P_1}(u) \wedge \neg\mathbf{P_1}(v) \wedge (u, v) \neq$

$e\}$.

The algorithm limits the set $E \backslash e$, over which the external summation is taking place, to the set $\{(u,v)|\mathbf{P_1}(u) \wedge \neg\mathbf{P_1}(v) \wedge (u,v) \neq e\}$. This new set is, as will be seen below, easy to update when we move from considering one edge $e \in P^*$ to the next edge along the path P^*. Moreover, since the property $\mathbf{P_1}(u)$ holds, we can use (7.11) to simplify (7.10). So far, we have not made any restrictive assumptions. Whatever we have said can be done on all graphs.

In order to simplify further, the Generalized Edge Replacement algorithm assumes that $\mathbf{P_2}(\mathbf{v})$ also holds, so that (7.12) can also be used to simplify (7.10). This assumption limits the class of graphs for which the algorithm finds the optimal solution. The property that does hold is $\neg\mathbf{P_1}(\mathbf{v})$, but it is not always the case that $\neg\mathbf{P_1}(\mathbf{v}) \Rightarrow \mathbf{P_2}(\mathbf{v})$. So, the Generalized Edge Replacement algorithm is guaranteed to find the optimal solution only in cases when the following condition hods:

$$\forall v \in V, \{v \in V|\neg\mathbf{P_1}(v)\} \subseteq \{v \in V|\mathbf{P_2}(v)\}. \tag{7.13}$$

In words, this means that there cannot be any vertices v such that the shortest path from the source s to v has any common edges with the shortest path from v to the destination d. If we also allow the source and destination nodes to vary, condition (7.13) is equivalent to the nonexistence of a source s, destination d, and arbitrary node v, such that the optimal paths $s \rightsquigarrow d$, $s \rightsquigarrow v$, and $v \rightsquigarrow d$ have one or more edges in common. In particular, the algorithm works for undirected graphs, and for directed graphs without directed cycles. The original algorithm [26] is given

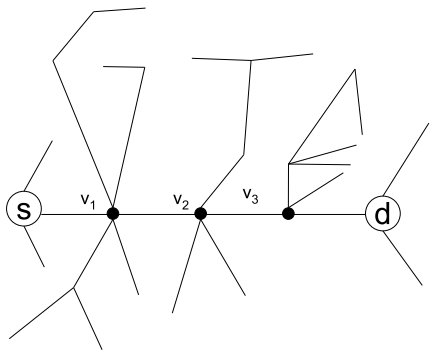

Figure 7.3: The shortest path tree is broken into a forest after removing all the edges of the $s \rightsquigarrow d$ shortest path.

only for undirected graphs, and is described below. The case of directed graphs and their relation to trust is discussed in Section 7.4 at more length.

Let the optimal path P^* consist of k nodes: $v_1 \equiv s, v_2, \ldots, v_k \equiv d$. If we delete all the edges of P^*, the optimal-path tree rooted at s becomes a forest. Exactly one node of $v_1 \equiv s, v_2, \ldots, v_k \equiv d$ belongs to each connected component of the forest, and we label each component by the subscript 1 through k of the corresponding node. For each edge $e \in E$, we consider the labels of the components that its two endpoints belong to. We define $left(e)$ to be the smallest of the two labels, and $right(e)$ the largest. For example, if there exists an edge e that goes between the component that s belongs to, and the component that d belongs to, then $left(e) = 1$, and $right(e) = k$. Note that it could happen that $left(e) = right(e)$.

The ingenuity of the algorithm is the way it keeps track of the candidate replacement edges when we move from considering the deletion of one edge of P^* to considering the deletion of the next. Namely, when moving from edge (v_{i-1}, v_i) to

(v_i, v_{i+1}), we no longer care about the edges whose "right" endpoint is v_i (since they can no longer be replacement edges), but we have to add to our list of candidates the edges whose "left" endpoint is v_i. These candidate edges are added to and removed from a priority queue, the structure of which makes it efficient to recover the edge that gives the $\preceq$-optimal weight among all $s \rightsquigarrow d$ paths that do not include a particular edge of P^*.

The running time of this algorithm is $O(m + n \log n)$, when Fibonacci heaps are used for the priority queue. Note that the optimal path trees rooted at s and at d need also to be computed before using this algorithm, but the overall asymptotic complexity is still $O(m + n \log n)$.

7.2.3 Computation of tolerances

To compute tolerances, we will consider again (7.3), which we repeat here for convenience.

$$w(A) = w(A_{\in}(e)) \oplus w(A_{\notin}(e)). \tag{7.14}$$

Substituting (7.8) into (7.3), and using the shorthand $w(B(e))$ we get that

$$w(A) = w(e)w(B(e)) \oplus w(A_{\notin}(e)). \tag{7.15}$$

We can now see the effect of changing the weight $w(e)$ on $w(A)$. In the equation above, only $w(e)$ changes, and we have four different cases depending on whether $w(A_{\in}(e)) = w(e)w(B(e)) \preceq w(A_{\notin}(e))$ or not, and also whether we increase $w(e)$ or decrease it. The two interesting cases appear, first, when $w(A_{\in}(e)) \preceq w(A_{\notin}(e))$ and $w(e)$ is increased (the wormhole case), and, second, when $w(A_{\in}(e)) \succeq w(A_{\notin}(e))$ and

GENERALIZED EDGE REPLACEMENT(G, s, d, P^*)

1 **for** $i \leftarrow 1$ **to** k $L[i] \leftarrow \emptyset$

2 **for** $i \leftarrow 1$ **to** k $R[i] \leftarrow \emptyset$

▷ Each L[i] and each R[i] is a set of edges

3 $Q \leftarrow$ NIL

▷ Q: priority queue for the candidate

replacement edges

4 **for** each $e \in E \setminus \{e \in E | e \in P^*\}$

5 **do if** $left(e) < right(e)$

Put e into $L[left(e)]$ and $R[right(e)]$.

▷ Ignore edges within the same

component.

6 **for** $i \leftarrow 1$ **to** $k - 1$

7 **do for** each $e = (u, v) \in L[i]$

8 **do** Put $(w(e), e)$ into Q, with wgt

$$w = d(s, u) \otimes w(u, v) \otimes d(v, d)$$

9 If $e \in R[i]$, remove (w, e) from Q

10 The $\preceq$-optimal weight in Q is $w(A_{\notin}(e))$.

Figure 7.4: For each edge e of the shortest path between nodes s and d, the generalized Hershberger-Suri algorithm computes the new optimal path (and associated weight) if edge e were removed from the graph.

$w(e)$ is decreased.

We will distinguish the cases when $e \in P^*$ and $e \notin P^*$. When $e \in P^*$, the situation is depicted in Figure 7.5. The following equivalences hold:

$$e \in P^* \Leftrightarrow P^* \in A_{\in}(e) \Leftrightarrow w(A_{\in}(e)) \succeq w(A_{\notin}(e))$$
$$\Leftrightarrow w(A(e)) = w(A_{\in}(e)) \quad (7.16)$$

If we increase the weight $w(e)$, then the optimal path weight will not decrease, and it may actually increase. In that case, the optimal path will not change.

If we decrease the weight $w(e)$ of an edge $e \in P^*$, then the optimal path weight will not increase. The optimality of P^* may change or not, depending on whether we decrease $w(e)$ below the lower tolerance of the edge or not. The lower tolerance of edge e is exactly equal to the new weight $w(e)^*$ that makes $w(A_{\in}(e))$ equal to $w(A_{\notin}(e))$.

In the case when $e \notin P^*$ (Figure 7.6), the lower tolerance of e is the $\preceq$-infimum of R, since no matter how much $w(e)$ is decreased, neither the optimal path nor the optimal path weight will change. Increasing the weight $w(e)$ will in general improve a previously suboptimal path. If it is improved beyond the weight of the previously optimal path (P^*), then the optimal path will change. This weight is the upper tolerance, which is again the value of $w(e)$ that makes $w(A_{\in}(e))$ equal to $w(A_{\notin}(e))$.

To summarize, $e \in P^*$ implies that $w(A) = w(P^*) = w(A_{\in}(e)) \succeq w(A_{\notin}(e))$. By changing $w(e)$, we change $w(A_{\in}(e)) = w(P^*)$, and the optimal path will stop being P^* if we decrease $w(e)$ enough to make $w(A_{\in}(e))$ decrease below $w(A_{\notin}(e))$. On the other hand, $e \notin P^*$ implies that $w(A) = w(P^*) = w(A_{\notin}(e)) \succeq w(A_{\in}(e))$.

The optimal path will change if we increase $w(e)$ enough to make $w(A_{\in}(e))$ increase beyond $w(A_{\notin}(e))$.

So, we see that the "interesting" tolerance in both cases is equal to

$$w(e)^* = w(A_{\notin}(e)) \oslash w(B(e)) \tag{7.17}$$

or, equivalently,

$$w(e)^* = w(A_{\notin}(e)) \oslash (d(s,i)d(j,d) \oplus d(s,j)d(i,d)), \tag{7.18}$$

where $\oslash$ is the inverse operator of $\otimes$, if it exists. If it does not exist, then we need to find the value of $w(e)$ that makes $w(e)w(B(e))$ equal to $w(A_{\notin}(e))$. In any case, for computing tolerances, these are the two important quantities to compute, and they need to be computed efficiently for all edges e. Computing the two optimal-path trees is enough for the efficient computation of $w(B(e))$, since for each e we only need four lookups $d(s,i), d(j,d), d(s,j), d(i,d)$. The Hershberger-Suri algorithm takes care of $w(A_{\notin}(e))$.

A final caveat is that the result of (7.17) has to be in the carrier set of the semiring to be admissible. This applies in particular to the wormhole scenario. Some edges can never be wormholes: those edges e for which it holds that

$$w(B(e)) \preceq w(A_{\notin}(e)). \tag{7.19}$$

7.3 Traffic Redirection Attacks

In this section we will see how knowing edge tolerances helps understand the effects of traffic redirection attacks on routing. Traffic redirection attacks aim to

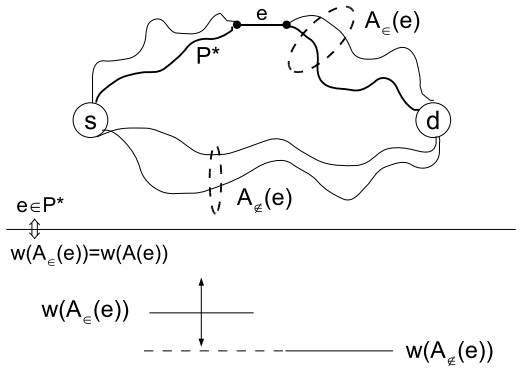

Figure 7.5: The edge e is in the optimal path P^*. If the weight $w(e)$ improves, the optimal path will remain P^*. If $w(e)$ deteriorates enough, P^* will stop being optimal. The lower tolerance of edge e is the lowest value of $w(e)$ for which P^* remains optimal.

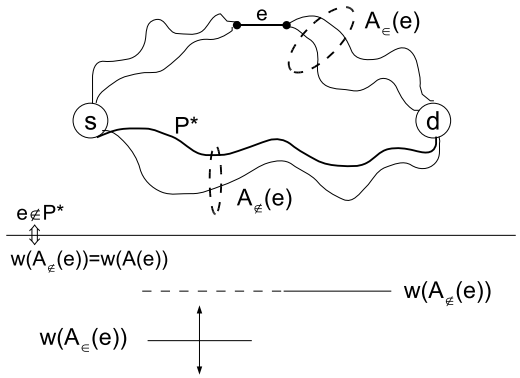

Figure 7.6: The edge e is not in the optimal path P^*. If the weight $w(e)$ improves enough, then P^* will stop being optimal. However, it is possible that some edges cannot be improved enough to create a path that is better than P^*. If $w(e)$ deteriorates, then nothing changes.

cause traffic to go through links or nodes of the attacker's choosing by making these links appear more trustworthy than the other candidate links. Their effect can be bad for various reasons:

- The redirected traffic could cause congestion to the part of the network it is redirected to.
- The redirected traffic will experience worse than optimal conditions (for instance, longer delay), assuming, of course, that before the attack it was being routed optimally.
- The attacker could eavesdrop on the traffic, modify it, or drop it.

There are two general mechanisms that can be used to achieve traffic redirection: traffic attraction and traffic repulsion. Traffic attraction occurs when a link is advertised to have much higher trust than it really has, thus luring traffic to go through it. Traffic repulsion occurs when the advertised trust of the link is much *worse* than in reality, thus causing traffic to avoid it.

Traffic attraction attacks have been known for a long time in the networking field [8], [48]. In wireless networks, the false advertisement of non-existent routes has been called the wormhole attack [27]. The effect is a traffic attraction attack.

The relevance of tolerances for traffic attraction attacks is the following: If the weight of a link is changed (advertised) to a value better than its upper tolerance, then the associated end-to-end traffic will be redirected to a path that crosses the changed link. Advertising non-existent links can be handled by finding upper tolerances for edges with initial weight ⓪. Depicted in Figure 7.7 are the initial weights

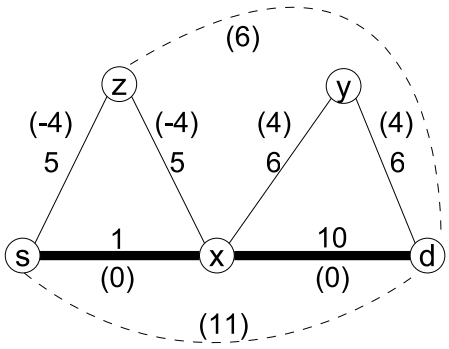

Figure 7.7: Traffic attraction attack: By artificially improving the opinion on a link, the attacker can lure traffic away from the optimal path. Shown next to each link are the initial weights and upper tolerances. Upper tolerances of two non-existent edges are also shown. Note that the negative values of tolerances are only shown for completeness purposes. Negative values for delays cannot be admitted. The optimal path is painted thick.

and upper tolerances of a simple topology in the case of delay. We can see that not all links can be used for an attraction attack. Only links that are "close enough" to the currently optimal end-to-end path in question are candidates.

In the case of traffic repulsion attacks, lower tolerances are relevant (Figure 7.8). They show how much a link's trust value needs to deteriorate before traffic is redirected. The new path can also be computed, which will show which new links and routers the new path goes through.

If we know the tolerances for all the edges in the network, we can see which edges, if artificially improved, destroyed, or degraded, will cause traffic to be redirected through which path. So, we can see the effects of the possible single-link

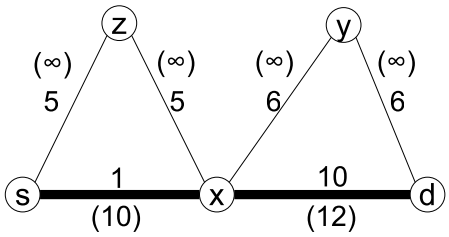

Figure 7.8: Traffic repulsion attack: By decreasing the trust on a link, the attacker can lure traffic away from the optimal path. Shown next to each link are the initial weights and lower tolerances. The optimal path is painted thick.

attacks, and plan accordingly. For example, one could check the trustworthiness of routers that could be used as replacements, or guard certain parts of a path whose replacements are judged undesirable for one reason or another. Note again that tolerances are computed with respect to specific source-destination pairs, so an edge will have many tolerances in general, and the analysis would have to be done for each pair separately, or, if this is impractical, for the "most important" ones.

7.4 Directed Graph Sensitivities

Directed graphs are more pertinent for trust, since opinions are directed. In directed graphs without directed cycles the Generalized Edge Replacement algorithm works optimally, since $\neg\mathbf{P_1}(v) \Rightarrow \mathbf{P_2}(v)$ for all $v \in V$, and for all edges $e \in P^*$. To see why this is so consider a node v such that for some edge e $\neg\mathbf{P_1}(v) \wedge \neg\mathbf{P_2}(v)$ is true. Then there would exist a directed cycle consisting of the subpath of the

$s \rightsquigarrow v$ shortest path that goes from e to v, and the subpath of the $v \rightsquigarrow d$ shortest path that goes from v to e. Since we assumed that the graph has no directed cycles, this is a contradiction.

We will now address whether limiting trust graphs to directed graphs without undirected cycles is excessively severe. In practice, we expect almost all graphs to have directed cycles. However, when computing indirect opinions from s to d, including cycles in the computation is uncomfortable from a trust point of view. First of all, cycles mean that a node can influence the opinion that is computed about itself, which is clearly undesirable. Secondly, cycles give rise to potential conflicts of interest in the following sense: two (or all) nodes on a cycle can agree to artificially increase their opinion for their next hop neighbor on the cycle, thus artificially increasing the computed opinions about themselves. In the same scenario, a node v can threaten another node u on the cycle that, if u does not increase his opinion about the next hop node (thus also increasing the computed opinion about v), then v will retaliate by decreasing his own opinion about his own next hop node, thus decreasing the computed opinion about u.

7.5 Conclusion

We provided an algebraic framework for computing the effect that arbitrary changes of link trust values have on optimal routing paths. Network designers and administrators are interested to know how the performance of their network can be affected, either in the positive or in the negative. We also showed that attackers can

cause malicious faults in an attempt to redirect traffic along compromised network elements.

Chapter 8

Practical Considerations

8.1 Taxonomy of Design Issues

In this section we are examining issues that should be considered by designers of trust metrics.

8.1.1 System Model

The most commonly used model is a labeled, directed graph. Nodes represent entities, and edges represent binary trust relations. These relations can be (for an edge $i \rightarrow j$): a public key certificate (issued by i for j's key), the likelihood that the corresponding public key certificate is valid, the trustworthiness of j as estimated by i, etc.

8.1.2 Centralized vs decentralized trust

By centralized trust we refer to the situation where a globally trusted party calculates trust values for every node in the system. All users of the system ask this trusted party to give them information about other users. The situation described has two important implications: First, every user depends on the trustworthiness of this single party, thus turning it into a single point of failure. Second, it is reasonable to assume that different users are expected to have different opinions about the same

target; this fact is suppressed here.

The decentralized version of the trust problem corresponds to each user being the "center of his own world". That is, users are responsible for calculating their own trust values for any target they want. This "bottom-up" approach is the one that has been most widely implemented and put into use, as a part of PGP [61] for public key certification.

Note that the distinction just mentioned refers to the semantics of trust. The actual algorithm used for the computation of trust is a separate issue: all data may be gathered at a single user, where the algorithm will be executed; or the computation may be done in a distributed fashion, throughout the network; or the algorithm may even be localized, in the sense that each node only interacts with his local neighborhood, without expecting any explicit cooperation from nodes further away.

8.1.3 Proactive vs reactive computation

This is an issue more closely related to the communication efficiency of the actual implementation. The same arguments as in routing algorithms apply: Proactive trust computation uses more bandwidth for maintaining the accuracy of the trust relationships. So, the trust decision can be reached without delay. On the other hand, reactive methods calculate trust values only when explicitly needed. The choice depends largely on the specific circumstances of the application and the network. For example, if local trust values change much more often than a trust decision needs

to be made, then a proactive computation is not favored: The bandwidth used to keep trust values up to date will be wasted, since most of the computed information will be obsolete before it is used.

8.1.4 Extensional vs intensional metrics

As mentioned by Maurer [39], one possible criterion to classify uncertainty methods is whether the uncertainty is dealt with *extensionally* or *intensionally*. In extensional systems, the uncertainty of a formula is computed as a function of the uncertainties of its subformulas. In intensional systems, uncertainty is attached to "state of affairs" or "possible worlds". In other words, we can either aggregate partial results in intermediate nodes (in-network computation), or we can collect all data (opinions and trust topology) at the initiator of the trust query and compute a function that depends on all details of the whole graph.

For example, the scheme proposed by Jøsang [29] is intensional, whereas our semiring model is extensional. As pointed out by Maurer [39], there seems to be a trade-off between computational efficiency and correctness. Extensional systems are more efficient, whereas intensional are more correct. For more on this, see the discussion on the distributivity property (Sec 5.4).

8.1.5 Negative and positive evidence (certificate revocation)

It is desirable to include both positive and negative evidence in the trust model. The model is then more accurate and flexible. It corresponds better to real-

life situations, where interactions between two parties can lead to either satisfaction or complaints. When a node is compromised (e.g. its private key is stolen) the public key certificates for this node should be revoked. So, revocation can be seen as a special case of negative trust evidence.

On the other hand, the introduction of negative evidence complicates the model. Specifically, an attacker can try to deface good nodes by issuing false negative evidence about them. If, as a countermeasure to that, issuing negative evidence is penalized, good nodes may refrain from reporting real malicious behavior for fear of being penalized.

8.1.6 What layer should trust be implemented in?

An important issue that is often glossed over is the layer at which the trust protocol will operate. That is, the services required by the protocol and the services it offers should be made clear, especially its relationship to other security components. As pointed out in [12], some secure routing protocols assume that security associations between protocol entities can be established with the use of a trust establishment algorithm, e.g. by discovering a public key certificate chain between two entities. However, in order to offer its services, the trust establishment algorithm may often assume that routing can be done in a secure way. This creates a circular dependency that should be broken if the system as a whole is to operate as expected.

8.2 Trust for routing

In this section we will discuss an implementation sketch for a trust protocol that could be used as a complement to aid a routing protocol. We will use parts of the research presented in this dissertation, and refer to them as appropriate.

The trust agent running at a node i collects information from the routing protocol about two things: the number of packets transmitted to each one-hop neighbor j for further forwarding (denoted by N_t^j), and how many of those packets were actually forwarded further by neighbor j (denoted by N_f^j). In particular, we assume that there is a mechanism to detect whether one-hop neighbors have forwarded a packet. In wireless networks, for example, this can be done with the Watchdog mechanism [38].

First of all, the values N_t^j and N_f^j will be translated to trust and confidence values for j. We follow the Beta distribution model, that is, we will use N_t^j and N_f^j to find the two parameters α and β needed to define a $\texttt{Beta}(\alpha, \beta)$ distribution. The parameter α will correspond to the number of successfully transmitted packets, therefore $\alpha = N_f^j$, while β will correspond to the number of dropped packets, so $\beta = N_t^j - N_f^j$. Trust will be the mean value of the Beta distribution, and confidence will increase as the variance of the distribution decreases. More precisely:

$$t^{ij} = \frac{\alpha}{\alpha + \beta} \tag{8.1}$$

$$c^{ij} = 1 - 12\,\mathrm{Var}\{\texttt{Beta}(\alpha, \beta)\} = 1 - 12\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{8.2}$$

The values α and β are both initialized to 1, and are increased by 1 for each

successfully forwarded packet, or dropped packet, respectively. So, initially, $t^{ij} = \frac{1}{2}$, and $c^{ij} = 0$ for all i, j. As the number of observations goes to infinity, the trust value will be following the fraction of packets successfully forwarded by neighbor j, while the confidence value will converge to 1, i.e., total confidence.

So far we described the direct trust generation process. The opinions generated will be exchanged with neighbors, similarly to distance-vector routing. That is, each node will have a vector of opinions, one opinion for each other node in the network, and these vectors will be exchanged with the neighbors. This exchange, as discussed earlier, can be done either proactively or reactively. If it is done reactively it will resemble more the controlled flooding paradigm, as done by on-demand routing protocols. Eventually, each node will compute end-to-end trust values using, for example, the path semiring, introduced in Chapter 6. Because we use semirings, we can compute opinions in a distributed way. The result of this trust computation phase is that each node will have a trust estimate about the paths to everybody else. What is more important for routing, the end-to-end trust estimate of i about a destination d will be associated with a next hop neighbor, through which the most trusted path passes.

We are now in a position to see how the generated and computed opinions will affect routing. Given a packet for destination d, node i will forward it to the next hop neighbor through which the most trusted path to d passes. In other words, we replace the routing metric of, e.g., hop count or delay with trust. This is an extreme case in which only trust matters when selecting routing paths. In general, we can use trust in combination with a more traditional metric, when the combination would

be tuned to fit the application's requirement (e.g. small hop count versus more trusted).

There is another way in which direct trust, in particular, can be taken into account in practice. For packets originating at node i, the next hop selection is done as in the previous paragraph. However, if a packet destined for d is coming from neighbor j, node i will check his opinion about neighbor j. He will only forward the packet if his opinion about j exceeds some threshold. This is following the protocol described in Chapter 3. In practice, this means that i will forward a packet coming from j if and only if j has forwarded "enough" packets transmitted to him by i. The threshold depends on i's E_i and N_i values (in Chapter 3 it is equivalent to $\frac{E_i}{N_i}$ per neighbor). A large threshold would conserve a user's energy at the expense of helping the network operate, while a user with a small one could risk the danger to exhaust his battery (or other resources, in case of non-wireless networks). Choosing a proper threshold would have to be empirically explored. It could even be the case that the threshold changes with time, e.g., increasing as the available energy decreases.

8.3 Practical Applications of Trust

In this section, we briefly survey three areas to which trust has been applied in one way or another, and point out the relevance of our work in each case. We have already shown (Ch. 6) how the Web-of-Trust concept of PGP could be modeled with our semiring-based indirect trust computation framework.

In [50], the notion of *data trust* is proposed for ephemeral networks, i.e., networks where only short-lived interactions happen between users. As a specific example, the authors are applying data trust ideas to vehicular networks. The novelty of data trust is that the focus is moved from computing trust on entities to computing the trustworthiness of the data reported by the entities. In particular, the authors are proposing that the trustworthiness of an event report is not only a function of the trustworthiness of the entity creating it, but also a function of, for instance, the proximity of the entity to the event being reported. In general, it is a function of the suitability of an entity to produce that report on that event. The reports are then relayed and aggregated by other entities, who create and transmit aggregate reports. Transitive trust could be applied to aggregate reports by making the trustworthiness of that report a function of the entity's suitability to aggregate. The point is that different entities may have different capabilities for aggregation (police cars versus private vehicles, for example).

Ebay [1] has a feedback system for its users to rate each other after a transaction. The rating can be either positive ($+1$), or negative (-1). A user's reputation is the sum of positive and negative feedback values he has received. However, there is nothing to prevent a malicious merchant from creating false accounts and fake transactions so as to give himself an artificially high rating. Such an attack would be immediately thwarted if transitive trust (indirect trust) computation were used. A user would only take into account trust information that is reachable from him through trust paths of direct interactions. This measure thwarts Sybil attacks [19] in general.

The Semantic Web is an attempt to make the information that is available on the Web more accessible to automated software agents. One of the issues surfacing is the trustworthiness of the information available, and how this trustworthiness can be assessed without human intervention. In [10], the authors propose a way to combine trust values as in the indirect trust computation we have described. The difference is that they compute the indirect trust for the *neighbors* of the destination, and use the computed values as weights when choosing among (or averaging over) the destination's neighbors' opinions about the destination. Since they are also using a generalized shortest path approach to compute indirect trust values, their scheme could benefit from our attack resistance evaluation of trust metrics (chapter 7).

Chapter A

Semirings and the Algebraic Path Problem

The algebra that we use is based on the algebraic structure called semirings. For a more complete survey of semirings and their applications, see Rote [53], and also Kschischang, Frey, Loeliger [33], and Aji, McEliece [3].

A.1 Definitions

A semiring is a triplet $(S, \oplus, \otimes)$, where S is a set (the *carrier set* of the semiring), and $\oplus, \otimes$ are two binary operations defined on S that satisfy certain properties. Intuitively, the properties correspond to those of the usual addition and multiplication (hence the symbols), but inverse or negative elements do not necessarily exist. Similar to the usual multiplication, we will be writing ab instead of $a \otimes b$. Also, $\otimes$ binds stronger than $\oplus$, so $ab \oplus c$ means, as expected, $(ab) \oplus c$.

- $\oplus$ is commutative, associative, with neutral element $\textcircled{0} \in S$:

$$a \oplus b = b \oplus a$$

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

$$a \oplus \textcircled{0} = a$$

- $\otimes$ is associative, with neutral element $\textcircled{1} \in S$, and absorbing element $\textcircled{0}$:

$$(ab)c = a(bc)$$

$$a\textcircled{1} = \textcircled{1}a = a$$

$$a\textcircled{0} = \textcircled{0}a = \textcircled{0}$$

- $\otimes$ distributes over $\oplus$:

$$a(b \oplus c) = ab \oplus ac$$

$$(b \oplus c)a = ba \oplus ca$$

If $\otimes$ is commutative ($ab = ba, \forall a, b \in S$), then the semiring is called *commutative*. If it holds that

$$ac = bc \Rightarrow a = b, \text{ for all } a, b \in S, c \in S \setminus \{\textcircled{0}\}, \tag{A.1}$$

then the semiring is called *right cancellative*. The dual property

$$ca = cb \Rightarrow a = b, \text{ for all } a, b \in S, c \in S \setminus \{\textcircled{0}\}, \tag{A.2}$$

is called *left cancellativity*, and if the semiring is both left and right cancellative, then it is simply called (multiplicatively) cancellative. Obviously, if the semiring is commutative, then the three notions of cancellativity are equivalent. The notion of multiplicative cancellativity will prove central to the relation between semirings and Sobrinho's algebra.

We define a total order on S by

$$a \preceq b \Leftrightarrow a \oplus b = b. \tag{A.3}$$

The reader can think of $\oplus$ as max for a concrete example. Throughout this chapter, $a \preceq b$ will intuitively be equivalent to saying that a is "worse" than b. A total order has certain properties, some of which are satisfied through the properties of $\oplus$, but some imply extra properties for $\oplus$. In particular:

- $\preceq$ is reflexive: $a \preceq a, \forall a \in S$. This means that the semiring has to be *idempotent*: $a \oplus a = a, \forall a \in S$.

- $\preceq$ is antisymmetric: if $a \preceq b$ and $b \preceq a$, then $a = b$. This is already satisfied because of the definition of $\preceq$ and the commutativity of $\oplus$.

- $\preceq$ is transitive: if $a \preceq b$ and $b \preceq c$, then $a \preceq c$. This is already satisfied because of the definition of $\preceq$ and the associativity of $\oplus$: $a \oplus c = a \oplus (b \oplus c) = (a \oplus b) \oplus c = b \oplus c = c$, where we have used, in sequence, $b \preceq c$, associativity of $\oplus$, $a \preceq b$, and again $b \preceq c$.

- For every $a, b \in S$, either $a \preceq b$, or $b \preceq a$. This property makes the order a *total* order as opposed to a *partial* order. It means that $a \oplus b$ is either a or b. From this property, idempotency follows as a corollary since $a \oplus a$ would necessarily be a.

Moreover, $a \oplus \textcircled{0} = a$ implies that $\textcircled{0} \preceq a$, that is, $\textcircled{0}$ is the $\preceq$-smallest element of the semiring.

Also, for all $a, b \in S$ we require that

$$a \otimes b \preceq a, b, \tag{A.4}$$

which implies that $a \oplus ab = a \Rightarrow a(\textcircled{1} \oplus b) = a$. If the semiring is multiplicatively cancellative, then we get that $\textcircled{1} \oplus b = \textcircled{1} \Leftrightarrow b \preceq \textcircled{1}$, that is, $\textcircled{1}$ is the $\preceq$-largest element of the semiring.

We will also use the symbol $\succeq$ with the obvious meaning

$$a \succeq b \Leftrightarrow b \preceq a. \tag{A.5}$$

When we use the words "increase" or "decrease", it will always be in the sense of the $\preceq$-order. For example, if $\oplus$ is min, then $a \preceq b$ means that $\min(a, b) = b \Leftrightarrow a \geq b$. So, in this case, "increasing" a weight in the $\preceq$-order means making it smaller.

A.2 The Algebraic Path Problem

Consider an undirected graph $G = (V, E)$, with $|V| = n$ vertices, and $|E| = m$ edges. A source vertex s and a destination vertex d are distinguished. The edges are weighted with a weight function $w : E \longrightarrow R$, where R is a semiring $(R, \oplus, \otimes)$. A path p is a sequence of nodes $(v_1, v_2, \ldots, v_k)$ such that $(v_i, v_{i+1}) \in E, 1 \leq i \leq k-1$. The weight of a path p is computed from the weights of its edges using the $\otimes$ operator.

$$w(p) = w(v_1, v_2) \otimes w(v_2, v_3) \otimes \ldots \otimes w(v_{k-1}, v_k) \tag{A.6}$$

The algebraic path problem is the computation of the $\oplus$-sum of the weights of all the paths p from s to d (see Figure A.1).

$$\bigoplus_{s \overset{p}{\rightsquigarrow} d} w(p) \tag{A.7}$$

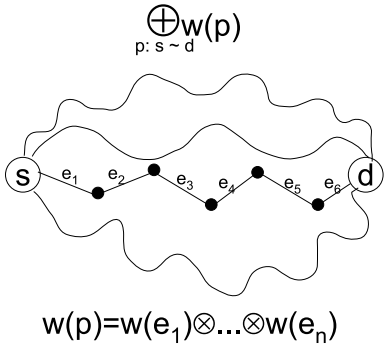

Figure A.1: The Algebraic Path Problem computes the $\oplus$-sum of the weights of all paths from a source s to a destination d. Each path weight is the $\otimes$-product of the weights of its edges.

Table A.1: Examples of semirings ($\bar{\mathcal{R}}_0^+ = \mathcal{R}_0^+ \cup \{\infty\} = [0, \infty]$)

S	{0,1}	$\bar{\mathcal{R}}_0^+$	$\bar{\mathcal{R}}_0^+$	$[0, 1]$	$\{(d, b) \in (\bar{\mathcal{R}}_0^+, \bar{\mathcal{R}}_0^+)\}$
$\oplus$	max	min	max	max	lex. min
$\otimes$	min	$+$	min	$\times$	$(d_1 + d_2, \min(b_1, b_2))$
⓪	0	∞	0	0	$(\infty, *)$
①	1	0	∞	1	$(0, \infty)$
Optimal path	Reachable	Shortest	Widest	Most reliable	Widest-shortest

For example, suppose that the semiring is the triplet $R = ([0, \infty], \min, +)$. Then, the algebraic path problem is equivalent to the shortest path problem with nonnegative weights. We assume that there is a unique optimal path, denoted by P^*, the weight of which is equal to the quantity in (A.7).

The problem of computing edge sensitivities with respect to an optimal subgraph of a graph appears when we consider how much the weight of an edge can change before the subgraph ceases to be optimal. For example, we can compute edge sensitivities for the minimum spanning tree of the graph. In this work, we compute sensitivities for the algebraic path. In particular, we compute how much the result of (A.7) changes when we vary the weight $w(e)$ of an edge $e = (i, j)$.

An associated concept is that of the upper and lower tolerances of an edge. The upper tolerance of an edge is the $\preceq$-largest weight it can take, while preserving the optimality of the path P^*. Conversely, the lower tolerance of an edge is the $\preceq$-smallest weight it can take, while preserving the optimality of the path P^*. It is expected that some tolerances will be trivial, since, for example, decreasing the weight of an edge that is not on the optimal path always preserves the optimality of the path.

Bibliography

[1] eBay – The World's Online Marketplace. http://www.ebay.com.

[2] A. Abdul-Rahman and S. Hailes. A distributed trust model. In *Proceedings of the 1997 New Security Paradigms Workshop*, pages 48–60, September 1997.

[3] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[4] E. Altman, A. Kherani, P. Michiardi, and R. Molva. Non-cooperative Forwarding in Ad Hoc Networks. Technical Report RR-5116, INRIA, 2004.

[5] R. J. Aumann and S. Hart, editors. *Handbook of Game Theory*. Elsevier Science Publishers B.V., 1992.

[6] R. J. Aumann and M. Maschler. Game theoretic aspects of gradual disarmament. In *Report to the U.S. Arms Control and Disarmament Agency ST-80*. Princeton: Mathematica, Inc., 1966.

[7] R. Axelrod and W. D. Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, March 2002.

[8] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communications Review*, 19(2):32–48, 1989.

[9] T. Beth, M. Borcherding, and B. Klein. Valuation of trust in open networks. In *Proceedings of the Third European Symposium on Research in Computer Security, ESORICS94*, pages 3–18, 1994.

[10] V. G. Bintzios, T. G. Papaioannou, and G. D. Stamoulis. An effective approach for accurate estimation of trust of distant information sources in the semantic web. In *Second International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing*, june 2006.

[11] A. Blanc, Y.-K. Liu, and A. Vahdat. Designing incentives for peer-to-peer routing. In *Proceedings of the IEEE Infocom Conference*, Miami, FL, March 2005.

[12] R. B. Bobba, L. Eschenauer, V. Gligor, and W. Arbaugh. Bootstrapping security associations for routing in mobile ad-hoc networks. In *Proceedings of IEEE Globecom 2003*, San Francisco, CA, December 2003.

[13] G. W. Brown. Iterative solution of games by fictitious play. In T. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 374–376. John Wiley and Sons, New York, 1951.

[14] S. Buchegger and J.-Y. L. Boudec. A robust reputation system for peer-to-peer and mobile ad hoc networks. In *Proceedings of P2PEcon 2004*, Harvard University, Cambridge MA, U.S.A., June 2004.

[15] L. Buttyan and J.-P. Hubaux. Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks. *ACM/Kluwer Mobile Networks and Applications*, 8(5), 2003.

[16] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations, rfc 2501, IETF, January 1999.

[17] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. In *Database Theory - ICDT 2003, 9th International Conference, Siena, Italy, January 8-10, 2003, Proceedings*, volume 2572 of *Lecture Notes in Computer Science*. Springer, 2002.

[18] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[19] J. R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, 2002.

[20] J. Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Philadelphia, July 2002.

[21] L. Eschenauer, V. D. Gligor, and J. Baras. On trust establishment in mobile ad-hoc networks. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *10th International Security Protocols Workshop, Cambridge, UK, April 2002*, volume 2845 of *Lecture Notes in Computer Science*, pages 47–66. Springer-Verlag, 2004.

[22] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and whitewashing in peer-to-peer systems. In *Proceedings of 3rd Annual Workshop on Economics and Information Security (WEIS04)*, May 2004.

[23] M. Félegyházi, J.-P. Hubaux, and L. Buttyán. Nash Equilibria of Packet Forwarding Strategies in Wireless Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 5(5):463–476, May 2006.

[24] E. J. Friedman and P. Resnick. The social cost of cheap pseudonyms. *Journal of Economics & Management Strategy*, 10(2):173–199, June 2001.

[25] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[26] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 252–259, Las Vegas, NV, October 2001.

[27] Y.-C. Hu, A. Perrig, and D. B. Johnson. Wormhole attacks in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2):370–380, February 2006.

[28] R. Ismail and A. Jøsang. The beta reputation system. In *Proceedings of the 15th Bled Conference on Electronic Commerce*, 2002.

[29] A. Jøsang. An algebra for assessing trust in certification chains. In *Proceedings of the Network and Distributed Systems Security (NDSS'99) Symposium*, San Diego, CA, 1999.

[30] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in p2p networks. In *WWW2003*, May 2003.

[31] M. Kearns, M. L. Littman, and S. Singh. Graphical models for game theory. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 253–260, 2001.

[32] D. Kreps and R. Wilson. Reputation and imperfect information. *Journal of Economic Theory*, 27:253–279, 1982.

[33] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498 – 519, February 2001.

[34] S. Lee, R. Sherwood, and B. Bhattacharjee. Cooperative peer groups in nice. In *Proceedings IEEE Infocom*, April 2003.

[35] R. Levien and A. Aiken. Attack-resistant trust metrics for public key certification. In *Proceedings of the 7th USENIX Security Symposium, San Antonio, Texas*, pages 229–242, January 1998.

[36] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining cooperation in multi-hop wireless networks. In *Proceedings of the 2nd Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[37] S. Marti, P. Ganesan, and H. Garcia-Molina. Sprout: P2P routing with social networks. Technical report, Stanford University, January 2004.

[38] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad-hoc networks. In *Proceedings of MOBICOM 2000*, pages 255–265, Boston, MA, 2000.

[39] U. Maurer. Modelling a public-key infrastructure. In E. Bertino, editor, *Proc. 1996 European Symposium on Research in Computer Security (ESORICS' 96)*, volume 1146 of *Lecture Notes in Computer Science*, pages 325–350. Springer-Verlag, 1996.

[40] M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[41] R. Morselli, J. Katz, and B. Bhattacharjee. A game-theoretic framework for analyzing trust-inference protocols. In *Second Workshop on the Economics of Peer-to-Peer Systems*, 2004.

[42] T. Moscibroda, S. Schmid, and R. Wattenhofer. When Selfish Meets Evil: Byzantine Players in a Virus Inoculation Game. In *25th Annual Symposium on Principles of Distributed Computing (PODC), Denver, Colorado, USA*, July 2006.

[43] J. Moy. OSPF version 2, RFC 2704, July 1991.

[44] R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1991.

[45] E. Nardelli, G. Proietti, and P. Widmayer. Finding the most vital node of a shortest path. *Theoretical Computer Science*, 296(1):167–177, 2003.

[46] J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, January 1950.

[47] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[48] R. Perlman. *Network layer protocols with Byzantine robustness*. PhD thesis, M.I.T., 1988.

[49] R. Ramaswamy, J. B. Orlin, and N. Chakravarti. Sensitivity analysis for shortest path problems and maximum capacity path problems in undirected graphs. *Mathematical Programming*, 102(2):355–369, 2005.

[50] M. Raya, P. Papadimitratos, V. D. Gligor, and J.-P. Hubaux. On Data-Centric Trust Establishment in Ephemeral Ad Hoc Networks. Technical report, LCA, EPFL, 2007.

[51] M. K. Reiter and S. G. Stubblebine. Resilient authentication using path independence. *IEEE Trans. Comput.*, 47(12):1351–1362, December 1998.

[52] M. K. Reiter and S. G. Stubblebine. Authentication metric analysis and design. *ACM Trans. Inf. Syst. Secur.*, 2(2):138–158, May 1999.

[53] G. Rote. Path problems in graphs. *Computing Supplementum*, 7:155–189, 1990.

[54] S. Singh, V. Soni, and M. Wellman. Computing approximate bayes-nash equilibria in tree-games of incomplete information. In *Proceedings of the 5th ACM conference on Electronic commerce*, pages 81–90. ACM Press, 2004.

[55] E. Sit and R. Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of the 1st International Workshop on Peer-To-Peer Systems, (IPTPS 2002)*, 2002.

[56] V. Srinivasan, P. Nuggehalli, C.-F. Chiasserini, and R. Rao. Cooperation in Wireless Ad Hoc Networks. In *Proc. of IEEE Infocom 2003*, San Francisco, March 30-April 3 2003.

[57] Y. Sun, W. Yu, H. Zhu, and K. J. R. Liu. A trust evaluation framework in distributed networks: Vulnerability analysis and defense against attacks. In *Proceedings IEEE Infocom*, 2006.

[58] G. Theodorakopoulos and J. S. Baras. On trust models and trust evaluation metrics for ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 24(2):318–328, February 2006.

[59] G. Theodorakopoulos and J. S. Baras. Sensitivity analysis of QoS routing paths. Submitted for publication, 2007.

[60] A. Urpi, M. Bonuccelli, and S. Giordano. Modelling Cooperation in Mobile Ad Hoc Networks: A Formal Description of Selfishness. In *Proc. of WiOpt 2003: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, INRIA Sophia-Antipolis, France, Mar. 2003.

[61] P. R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.