Unreliable Gossip

MSc Thesis *(Afstudeerscriptie)*

written by

Line van den Berg
(born September 28, 1992 in Neuchâtel, Switzerland)

under the supervision of **Prof Dr D. J. N. van Eijck** and **Drs T. Achourioti**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
January 12, 2018	Prof Dr R. M. de Wolf (Chair)
	Prof Dr D. J. N. van Eijck (Supervisor)
	Drs T. Achourioti (Supervisor)
	Prof Dr S. J. L. Smets
	Dr R. de Haan

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Voor opa, mijn grootste fan

“The leaks are real, but the news is fake”
– Donald Trump

Abstract

Dynamic communication is the field that describes the spread of information throughout a network of agents. More specifically, it assumes that every agent has a secret and investigates how agents should decide, based on their own knowledge about the network, what calls to make so that ultimately everyone knows all the secrets. In the dynamic setting, next to sharing secrets, agents additionally share phone numbers. This means that while the context spreads, the network grows. In this thesis, we adapt the framework of dynamic communication to account for unreliable agents. The motivation behind this is the question whether the assumption that *everybody is reliable* is realistic for any kind of practical application. This adaptation for unreliable agents puts an additional requirement on the executed communication protocols: to not only result in everyone knowing all the secrets (of the reliable agents), but also in everyone successfully identifying the unreliable agents. With this we provide an exploration in the field of dynamic communication with a possibility of error. We show that already by introducing a small amount of unreliability the known results dynamic communication cannot be extended. In particular, we prove that the Learn New Secrets protocol may not be sufficient on networks with only terminal *alternating bluffers*, agents that bluff about their own secret in every second call they are involved in—whereas this was true in the reliable case. Additionally, we show that unreliable agents that do not initiate communication are harder to identify than those that do. This has some seemingly paradoxical consequences for the kind of security measures taken against fake news. Namely, as we will prove, that individual actions as deleting or blocking the agents that are considered unreliable might actually have the opposite effect on the network: the unreliable agents may remain unidentified by the entire network. This exemplifies that security measures against the spreading of false information might come at the cost of the identification of the unreliable source. In conclusion, by putting unreliability in the spotlight, this thesis takes the research of dynamic communication in the direction of practical application and opens the doors for future research.

Acknowledgments

This thesis would not have been a thesis without help. First of all, I would like to hugely thank my supervisor Jan. Jan showed me that there is more to logic than solving puzzles. What started with generating Sudoku puzzles ended up in the research field of Gossip. Besides helping with the technicalities of this thesis, Jan always had time to discuss what starts today: life after graduation. Sharing my enthusiasm, Jan and I could talk ages about everything not related to this thesis, from the US presidential elections to the best ways to learn French. I hope after today I can still drop by for my weekly portion of life advice!

In addition, I would like to thank Dora very much for her involvement in this thesis. Dora taught me my baby-steps in logic at Amsterdam University College (AUC) and has made me hooked for logic ever since. Having Dora as my second supervisor brings back the good memories of writing my bachelor's thesis under her supervision. It is really cool to see how involved Dora is with her students and I hope, and am convinced that AUC will create more logicians in the near future.

Next, I would like to express my gratitude to Malvin for helping me with anything Haskell related and coping with all the *"help, my program crashed"* e-mails. In addition, I would like to thank Malvin for proofreading parts of my thesis and all his useful comments both on the code as the text—not to mention all the non-thesis related talks we had.

I want to thank the other members of the committee for their time reading this thesis and being present at the defense with hopefully interesting (yet, not too complicated) questions.

Then, a great thanks goes to my family and friends generally for bearing with me and for reminding me between all my climbing of a thesis that had to be written. For quite natural reasons (without me, no thesis), I would like to very much thank my parents for their unconditional support and trust in me. They have always motivated me to study what I like and follow my passion—even if that caused them to freeze their toes off just to see me climb one route on the World Championships Ice Climbing. And a special thanks goes to Wood for sharing huge amounts of stokedness to the moon and for together creating big, crazy adventures!

Last but not least, thanks to everyone who started reading my thesis and quit here. Thanks to everyone who found this funny.

Contents

Chapter 1

Introduction

Over the past 30 years, the internet has made great changes for technology and communication. An area where a very significant shift has occurred is news- (and more generally information) provision. Since nowadays digital media have become the major source of information, validity and integrity have become significant. With the growth of the internet comes the growth of fake news. Recently, there has been a lot of commotion about this topic, possibly having influenced the US presidential election results [2].

The challenge, of course, with fake news lies in its identification. In the beginning of March, Facebook announced a collaboration with NU.nl and the University of Leiden to tackle fake news in The Netherlands [19, 33]. Since then, it is possible for Facebook users to label news sources and articles as potentially "fake" and to have them examined on their reliability by a group of news-experts. Google has taken similar measures and it is expected that other tech-giants will follow. Even though this is a first welcome step in the battle against fake news, the procedure itself is highly inefficient. Ideally, computer algorithms would take care of this task much more efficiently.

In this thesis, we will consider a simplistic model of fake news, which can be thought of as communication errors or noise. We approach these errors and noise from the perspective of *dynamic communication*. Dynamic communication is the research field that describes the spreading of information in a network. More specifically, it investigates how agents should decide, based on their own knowledge about the network, on a successful *call sequence* to complete the network. In other words, how agents should decide on an order in which the agents call each other so that ultimately all the agents know the secret of each agent. ANY ("call any agent"), Call Me Once ("call each agent once") and Learn New Secret ("call only agents that you do not know the secret of") are examples of communication protocols that decide on such an order. We represent a dynamic communication network by a graph in which the nodes denote the agents and the connections the ability of the agents to contact each other. In the dynamic setting, agents can not only share their secrets while communicating, but also phone numbers of other agents. This means that the accessibility relation between the agents expands.

In our setting, we extend dynamic communication in order to deal with the possibility of error. Specifically, we adapt the current model to account for networks in which not all agents are necessarily reliable. The motivation behind this is the question whether the assumption that *everybody is reliable* is realistic for practical applications. We consider unreliable agents as agents that fail to truthfully share their own secret, the secrets of others or even the contact information of agents in the network. Failing to communicate contact information of other agents can also be viewed as manipulating the network by cutting connections. We call these options *agent types*. Next to a certain type, we also explore different properties of agents. Agents can for instance be *consistent* (a consistent agent remembers whom she has told what, i.e. she remembers who she has lied to), or *Trumpian* (a Trumpian agent deletes or blocks any agent that she consider unreliable). With this adaptation, we investigate what it means for a network to be complete and whether there are successful call sequences that follow a certain communication protocol. But now, next to learning all the secrets, this new setting creates the need for including the identification of the unreliable agents in the definition of completeness.

Within our framework, we have focused on a relatively simple unreliable agent: the *alternating bluffer*. This is an agent that bluffs about her own secret in every second call she is involved in. Investigating this agent provides a good first way of approaching the study of networks in which unreliability occurs unintentionally in the form of error or noise. This is because alternating bluffers follow a clear pattern and as a result their behavior is easily predicted. This way, the alternating bluffer allows for a shift from the idealized assumption that *everybody is reliable*, yet remains formally feasible, i.e. simple enough, to study.

We show that already by introducing this alternating bluffer, the known results about dynamic communication from [29, 30] fail to hold. In particular, we prove that with this small amount of error or noise we cannot continue classifying the LNS protocol by sun graphs. The failure of this exemplifies that it is going to be difficult to prove general results about dynamic communication with unreliability. Even further, it emphasizes the need for fully discarding the assumption that *everybody is reliable* for any kind of practical application as well as the need for new protocols to properly cope with unreliability.

Introducing unreliable agents into dynamic communication raises many interesting questions. For instance it raises questions about the kind of security measures that can be taken against the unreliable agents to ensure the reliability of information, yet preserve efficiency. We show that contrary to what one would expect, the individual actions of blocking or deleting agents that are considered unreliable may be insufficient from the perspective of identification. This means that even though the property of being Trumpian seems a natural response to unreliability, this may have the result that the unreliable agents can more easily remain undetected. Specifically, we prove that unreliable agents that do not actively spread false information are harder to identify than those that do. How this counter-intuitive consequence weighs against the reasons for preventing false information to spread is now on the agenda for future research.

Work done in this thesis shows that applications of this model reach beyond the identification of fake news. There is a close connection between dynamic communication and the study of epidemics [10]. In epidemics, we can use a variant of unreliable agents to model agents that are immune to the spreading disease or fail to transmit it. This opens up the possibility of applying this work to the field of herd immunity that investigates what percentage of the population has to be vaccinated in order to control an infectious disease. Other applications of this model are faulty sensors in a network of electronics, power grids [35], biological networks such as metabolic networks, food webs [36] and neural networks [35].

With this work, we provide an exploration in the work of dynamic communication with a possibility of error. This opens the doors for much needed future research in this area with a focus on practical applications.

1.1 Outline of Thesis

The structure of this thesis is as follows. In Chapter 2 we introduce the formal framework by van Ditmarsch et al. [29, 30] concerning dynamic communication. We motivate our framework with a working example and explain the use of the protocols ANY, CMO and LNS in completing networks. We then mention some of the known results.

Chapter 3 explores what kind of unreliable agents there are. We introduce a way to quantify information as true or false and explain different types and properties of agents that are used to classify unreliability.

We then combine unreliability with dynamic communication in Chapter 4. With a focus on the formally simple alternating bluffer, we extend the current framework of van Ditmarsch et al. [29, 30] by splitting the secret relation in three disjoint sets and adjusting the definitions accordingly. Additionally, we introduce a stronger version of completeness that includes the identification of unreliable agents.

In Chapter 5, we will conduct an exploration in the field of dynamic communication when we discard the assumption that *everybody is reliable.* We introduce reliable counter- and subgraphs that allow us to connect completeness on DCU graph with completeness on DC graphs. Then, with the help of the implementation introduced throughout this thesis, we will investigate the protocol LNS for our framework. We show that we cannot extend the results of the reliable case by [29] to the case of a possible error. More specifically, we prove that on DCU graphs that are sun graphs with only terminal unreliable agents, LNS may fail to identify the unreliable agents. Next, we restrict the protocol LNS in a way that only the reliable agents are allowed to initiate calls. We then prove that unreliable agents that are not allowed to initiate any form of communication are harder to identify than unreliable agents that do so. We end the chapter by discussing the results and by providing concrete questions for future research.

Chapter 6 describes some possible extensions of our framework. We introduce censorship actions that allow agents to delete or block the agents that they consider unreliable. Even though these actions seem suitable security measures against the spreading of false information, we show that these actions have some counter-intuitive consequences for the identification of unreliable agents. In addition, we explain a different attitude of the agents toward new information: the *see for yourself* attitude. Instead of considering a *presumption of innocence*, this attitude values first-hand over second-hand information. This provides a first way of approaching skepticism in dynamic communication networks. We also discuss how we can introduce a saboteur in the networks—an agent that is allowed to cut connections. Last but not least, we adapt our formalisms to account for broadcast calls, calls made to multiple agents at the same time.

With this, the thesis opens the doors for future research and a wide range of applications. Examples are security measures against fake news, herd immunity and "gossip about gossip" as an alternative to blockchain technology for cryptocurrencies. We discuss these directions for future research and applications in Chapter 7.

Chapter 8 concludes this thesis by highlighting our contribution to the current research and by discussing the important results of this thesis. We emphasize that this thesis provides a starting point for much needed further research.

1.2 Role of Implementation

Throughout this thesis, we implement our new framework of dynamic communication with unreliability in the programming language Haskell. This implementation is used to create an intuitive understanding of dynamic communication with unreliable agents and forms the basis for many ideas discussed. As such, the role of the implementation is to provide an easy and quick way of investigating properties and protocols and to test hypotheses concerning the framework. In particular, the implementation helps to find clean definitions, examples and counter-examples. Of course, there is a limitation to this approach. Namely that it becomes very time consuming and computationally difficult to cope with relatively large networks because of limited computational power. In Chapter 5, we will explain why the implementation can still provide solid ideas by the *Small Scope Hypothesis* [17].

The entire thesis, however, can be read without consulting the implementation parts. All the definitions, theorems and proofs stand and can be well understood by themselves. Therefore, on the one hand, the reader not familiar with the language of Haskell can skip these parts while reading the thesis. Readers, on the other hand, that are acquainted with the language will find that the implementation adds flavor and substance to the ideas introduced. Especially since the complete code can be accessed by the following link to the repository: `https://github.com/linevdberg/UnreliableGossip`. Here one can play around with the implementation and verify our results.

A motivation for literate programming can be found in [18] and a comprehensive introduction to Haskell for the means of logic can be accessed in [9].

Chapter 2

Dynamic Gossip

Dynamic gossip is the field that studies the spread of information, or more specifically secrets, throughout a network. Such a network is represented by a graph in which the nodes represent the agents and the connections between the nodes the ability of the agents to contact each other. When agents make use of this ability by means of a phone call, secrets are exchanged. The dynamic setting has the added feature that not only secrets are exchanged, but also phone numbers. This means that while the context spreads the network grows.

In this chapter we introduce Dynamic Communication (DC) graphs according to the work of Van Ditmarsch et al. [29]. It is assumed that every agent has a unique secret and that initially each agent knows only her own secret. Additionally, each agent has access to a set of phone numbers (also called *contact list*) at least including her own number. Calls in the network take place between two agents and during a call, all secrets and phone numbers known to the agents involved are exchanged–including the secrets and phone numbers they learned before the current call. In the DC graphs, knowledge is represented by directed binary relations: “knowing a phone number” is represented by N and “knowing a secret” by S. We will show that already from the definitions, it follows that $S \subseteq N$ for all the DC graphs we consider. Alternatively, we can think of N and S as functions in $A \to \mathcal{P}(A)$ where A is the set of all agents. Thus each call changes the way in which the graph is connected in an increasing manner. We say that a network is complete when the relation S is complete, i.e. when all agents know all the secrets.

Originally, dynamic communication has mostly been studied from a combinatorial and graph-theoretical perspective. For a survey of the combinatorial results, see [15]. One of the results concerns the number of calls needed to complete a network. In a network with $n > 3$ agents where each agent has the phone number of every other agent (i.e. N is complete), $2n-4$ calls are sufficient for everyone to learn all secrets.

Theorem 2.0.1. *In a DC Graph $G = (A, N, S)$ where A is the set of agents such that N is complete, $S = I_A$ and $|A| = n > 3$, at least $2n-4$ calls are*

$$\begin{tikzcd}
a \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] \arrow[dr, leftrightarrow] & b \arrow[d, leftrightarrow] \arrow[dl, leftrightarrow] \\
c \arrow[r, leftrightarrow] & d
\end{tikzcd}$$

Figure 2.1: A network with four agents in which the number relation N is complete. By Theorem 2.0.1 his network can be completed by the call sequence $ab; cd; ac; bd$.

sufficient to complete G.

Proof. We prove this by induction. The base case where $n = 4$, we can complete G by the call sequence $ab; cd; ac; bd$, which are indeed $2n - 4 = 2 \times 4 - 4 = 4$ calls. Now assume that for n agents, $2n - 4$ calls are sufficient to share all secrets. Let σ_n be a call sequence of length $2n - 4$ that completes the graph G with $|A| = n$. Then if we add an agent (i.e. we consider $n + 1$ agents), we add two calls between the new agent a_{n+1} and some existing agent a_i (with $i \in \{1, \ldots, n\}$): $\sigma_{n+1} = a_i a_{n+1}; \sigma_n; a_i a_{n+1}$. Now the secret of agent a_{n+1} will be spread throughout the network just like the secret of agent a_i, which will be spread to everyone by the call sequence σ_n. Thus everyone will learn the secret of agent a_{n+1}. Furthermore, agent a_{n+1} will learn all the secrets because agent a_i has learned all the secrets after executing $a_i a_{n+1}; \sigma_n$ and will share this to agent a_{n+1} in the final call $a_i a_{n+1}$. Thus $2n - 4 + 2 = 2n - 2 = 2(n + 1) - 4$ calls are sufficient. □

In [26] it is shown that less than $2n - 4$ calls is not possible to complete such networks. Papers like [14, 8] investigate the number of calls needed for graphs that are not complete. In these cases, naturally the minimum number of calls needed to distribute all secrets is larger.

Essential to Theorem 2.0.1 is the fact that there is some all-knowing scheduler who tells the agents whom they have to call and when. Even though this is very efficient, it is not very practical as such an outsider often does not exist in daily life. Only more recently, dynamic communication is studied from a more epistemic point of view, meaning that the agents base their decision on what call to make only on their own knowledge [4, 3, 5]. This means that such an all-knowing scheduler is not available and agents can follow certain protocols for their decision procedure.

Before we dive into the formal definitions as introduced in [29, 30], one may wonder whether such a network is relevant in an increasingly connected world. Nowadays it is relatively easy to exchange information with a group of agents at the same time via e-mail, WhatsApp, Facebook, et cetera. Another way is to send a text-message. These are examples of *one way calls* and *broadcast calls*. The applications of dynamic communication however reach far beyond communication between agents. For instance the spread of epidemics [10], electronic networks, food webs [36] and any other practical problem related to information gathering and sharing. In such networks agents do not necessarily have the ability to send information simultaneously to multiple agents. More about broadcast calls will be discussed in Section 6.4.

Even though dynamic communication clearly has a wide range of applications, it must be noted that there has not yet been found a sound and complete logic to describe it in a satisfying way. Investigating an appropriate axiomatization is beyond the scope of this thesis, but is desirable for future research in this field. In [34], a first approach is given to formalize and analyze dynamic gossip with the use of NetKAT, a sound and complete network programming language based on Kleene algebra with tests.

2.1 A Working Example

Consider a network of three agents a, b and c such that initially all the agents know only their own secret, agent a knows the phone number of agent b and agent b that of agent c. The first call to take place is the call ab. In this call, all the secrets and phone numbers known to a and b are shared. Thus agent b learns the phone number of agent a and agent a that of agent c. Next the call bc happens. This means that now not only agent b knows all the secrets, but agent c as well. Finally the call ac is made and the network is complete. A visualization is given in Example 2.1.1, where the dashed arrows denote the number relation and the solid arrows the secret relation as done in [29]. We assume that both relations are reflexive, and that the number relation is included in the secret relation, i.e. the dashed arrow is included in the solid arrow.

Example 2.1.1. *The development of a network with three agents where initially the number relation is not complete.*

$$\begin{tikzcd}
a \arrow[r, dashed, leftrightarrow] \arrow[rr, dashed, leftrightarrow, bend left] & b \arrow[r, dashed, leftrightarrow] & c & \overset{ab}{\rightarrow} & a \arrow[r, leftrightarrow] \arrow[rr, dashed, leftrightarrow, bend left] & b \arrow[r, dashed, leftrightarrow] & c & \overset{bc}{\rightarrow} & a \arrow[r, leftrightarrow] \arrow[rr, dashed, leftrightarrow, bend left] \arrow[rr, <-, bend left=50] & b \arrow[r, leftrightarrow] & c & \overset{ac}{\rightarrow} & a \arrow[r, leftrightarrow] \arrow[rr, leftrightarrow, bend left] & b \arrow[r, leftrightarrow] & c
\end{tikzcd}$$

As we will see later, this is an example of a valid execution of the Learn New Secrets protocol: a protocol that requires for a call ab to take place that agent a does not yet know (prior to the call) the secret of agent b.

2.2 Formalisms

Formally, a dynamic communication, or DC, graph is a triple $G = (A, N, S)$ consisting of a set of agents (A), an accessibility- or number-relation (N) and a secret-relation (S).

Definition 2.2.1. [Dynamic Communication Graph] *A* dynamic communication (DC) graph *is a triple $G = (A, N, S)$ where A is the set of agents in the* *network and* $I_A \subseteq N, S \subseteq A^2$ *where* N *and* S *denote the number and secret relation, respectively. An* initial dynamic communication graph *is a dynamic communication graph with* $S = I_A \subseteq N$*. Agent* x *is an* expert *if* $S_x = \{y \in A \mid xSy\} = A$*. Agent* x *is a* spider *if* $N_x = \{y \in A \mid xNy\} = A$*. Agent* x *is* terminal *iff* $N_x = \{x\}$*.*

We say that "agent a knows agent b's phone number" if and only if aNb and that "agent a knows agent b's secret" if and only if aSb. A network is considered to be complete when everyone knows all the secrets, i.e. whenever all agents are connected by the secret-relation. In other words, a network is complete when all agents are experts.

Definition 2.2.2. [Complete DC Graph] *A dynamic communication graph* $G = (A, N, S)$ *is* complete *if* S *is complete.*

Whenever a call ab is made in the network, agent a and b share their secrets, contact lists and other agents' secrets. If the network prior to the call is denoted by G, the resulting network is G^{ab}.

Definition 2.2.3. [Call] *Let* $G = (A, N, S)$*,* $x, y \in A$ *and* xNy*. The* call xy *maps* G *to* $G^{xy} = (A, N^{xy}, S^{xy})$ *defined by*

$$N_z^{xy} = \begin{cases} N_x \cup N_y & \text{if } z \in \{x, y\} \\ N_z & \text{otherwise} \end{cases} \tag{2.1}$$

$$S_z^{xy} = \begin{cases} S_x \cup S_y & \text{if } z \in \{x, y\} \\ S_z & \text{otherwise} \end{cases} \tag{2.2}$$

This clearly captures the fact that all information is exchanged when a call takes place. When dealing with multiple calls in a network, we need the definitions of call sequences and of induced DC graphs. Intuitively, an induced DC graph G^σ is the DC graph that is the result of applying a call sequence σ to G.

Definition 2.2.4. [Call sequence, Induced DC Graph] *A* call sequence σ *is a sequence of calls, with the following properties:*

- ϵ *is the empty sequence*
- $\sigma; \tau$ *is the concatenation of (finite) sequence* σ *and sequence* τ
- $\sigma \sqsubseteq \tau$ *denotes that* σ *is a prefix of* τ *(where* $\sigma \sqsubset \tau$ *means proper prefix)*
- $|\sigma|$ *is the length of a sequence*
- $\sigma[i]$ *is the* ith *call in* σ *(given* $|\sigma| = n \geq i$*)*
- $\sigma|i$ *is the prefix of* σ *consisting of the first* ith *calls (given* $|\sigma| = n \geq i$*)*
- *For* $x \in A$*,* σ_x *is the subsequence of calls in* σ *containing* x *defined as:* $\epsilon_x = \epsilon$*,* $(\sigma; uv)_x = \sigma_x; uv$ *if* $x = u$ *or* $x = v$ *and* $(\sigma; uv)_x = \sigma_x$ *otherwise*

We say that call xy is possible *given a dynamic communication graph $G = (A, N, S)$ if xNy. Call sequence ϵ is possible for any dynamic communication graph, and call sequence $xy; \sigma$ is possible for G if call xy is possible for G and sequence σ is possible for G^{xy}. If a call sequence σ is possible for a dynamic communication graph G, the* induced dynamic communication graph *G^{σ} is defined as: $G^{\epsilon} = G$, $G^{xy;\sigma} = (G^{xy})^{\sigma}$.*

It follows from the definitions that for any initial dynamic communication graph $G = (A, N, S)$ (i.e. $S = I_A$) and any call sequence σ, $S^{\sigma} \subseteq N^{\sigma}$. Thus in this setting, an agent can only know the secret of someone if she also knows her phone number. Of course, one may question whether this is a realistic feature.

Lemma 2.2.5. *For any initial dynamic communication graph $G = (A, N, S)$ and a call sequence σ, we have that $S^{\sigma} \subseteq N^{\sigma}$.*

Proof. We prove this by induction on σ. The base case follows by observing that $S^{\epsilon} = S = I_A \subseteq N = N^{\epsilon}$. For the induction step, assume that $S^{\sigma} \subseteq N^{\sigma}$ (IH) and let xy be a possible call in G^{σ}. Now suppose that $(a, b) \in S^{\sigma;xy}$. Then either $(a, b) \in S^{\sigma}$ or $(a, b) \in S^{\sigma;xy}$ but $(a, b) \notin S^{\sigma}$. In the first case $(a, b) \in N^{\sigma}$ and thus $(a, b) \in N^{\sigma;xy}$ follows directly from the induction hypothesis. In the second case, we assume without loss of generality that $a = x$. So $(x, b) \in S^{\sigma;xy}$ but $(x, b) \notin S^{\sigma}$. But then it must be that $(y, b) \in S^{\sigma}$ and by induction hypothesis, $(y, b) \in N^{\sigma}$. So, as all phone numbers and secrets are shared in the call xy, $(x, b) = (a, b) \in N^{\sigma;xy}$. □

2.3 Protocols

A protocol for dynamic communication is a procedure that describes how the agents in a network spread their secrets. More specifically, it describes how the agents should decide what calls to make. In this setting, we assume that these decisions are only based on the agent's own knowledge of phone numbers and secrets. However, there are more options. For instance whether the agents have knowledge about calls that they are not directly involved in. More specifically, this is the question whether communication in the network is asynchronous (agents are unaware of calls that they do not participate in), synchronous (agents know how many calls take place, but not who are making the calls) or observable (agents know exactly what calls take place, although they cannot learn the content of the call). In this thesis, we will assume that the networks are asynchronous. The other options are left for future analysis, see also [28].

This is precisely what we meant by the epistemic approach to dynamic communication: instead of an all-knowing scheduler deciding what calls should be made, the agents make this decision based on what they know about the network. In particular, distributed protocols are protocols in which the knowledge of the agents is explicitly modeled.

Definition 2.3.1. [Protocols] *A* protocol *is a procedure for agents to decide what call to make. Examples of protocols are the following:*

- ***Any Call:*** *(ANY) While $G = (A, N, S)$ is not complete, select any call $xy \in N$ such that $x \neq y$.*
- ***Call Me Once:*** *(CMO) While $G = (A, N, S)$ is not complete, select any call $xy \in N$ such that $x \neq y$ and there was no prior call xy or yx.*
- ***Learn New Secrets:*** *(LNS) While $G = (A, N, S)$ is not complete, select any call $xy \in N$ such that x does not yet know y's secret.*

In [29] more such protocols are discussed. For instance the Token (TOK) protocol in which only agents y that have not yet participated in any call or that were called in the previous call they were involved in (i.e. xy for some $x \in A$) are allowed to make a call, and the Spider (SPI) protocol in which this is the other way around: agents x, who did not yet participate in any call or in the previous call have made the call (i.e. xy for some $y \in A$), are only allowed to make a call.

Such protocols, denoted by P, can also be described by a so-called *protocol condition* $\pi(x, y)$ [29]. This is a first-order definable property for DCU graphs with a history of calls that has to be satisfied in order for a call xy to be permitted by the protocol P. We then say that the call xy is P-permitted. For the ANY protocol, $\pi(x, y) = \top$, for the CMO protocol, $\pi(x.y) = xy \notin \sigma_x \wedge yx \notin \sigma_y$, and for the LNS protocol, $\pi(x, y) = \neg S^\sigma xy$.

We can now define what it means for a protocol P to be successful on a DC graph $G = (A, N, S)$, where we let P_G be the set of all P-permitted call sequences on G [29].

Definition 2.3.2. [Success] *Let a DC graph $G = (A, N, S)$ and a protocol P be given. A finite call sequence $\sigma \in P_G$ is* successful *if all agents $a \in G^\sigma$ are experts, i.e. when S is complete. We say that a sequence σ is P-maximal on G if σ is P-permitted on G and there is no call P-permitted on G^σ.*

- *P is* strongly successful *on G if all P-maximal $\sigma \in P_G$ are successful.*
- *P is* weakly successful *on G if there is a $\sigma \in P_G$ that is successful.*
- *P is* unsuccessful *on G if there is no $\sigma \in P_G$ that is successful.*

In other words, P-maximality of σ means that no call xy can be added to σ such that $xy; \sigma$ is still P-permitted. We are now ready to state the dynamic communication problem.

Definition 2.3.3. [DC Problem] *Given a collection $\mathcal{G}$ of DC graphs and a protocol P, the* dynamic communication problem *is: Is P (strongly, weakly, un-) successful on $\mathcal{G}$?*

Given a certain class of networks, the DC problem asks the question what protocols result in a complete network. This is interesting because then we cannot only characterize networks by their properties, but also by what kind of protocols are successful.

It may be clear that on any network that consists of distinct clusters (i.e. groups of agents that are not connected to each other in any finite number of steps) no protocol can be successful. This is shown in [29].

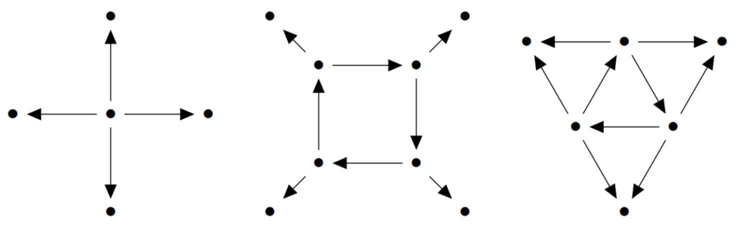


Figure 2.2: Some example sun graphs

Proposition 2.3.4. *If $G = (A, N, S)$ is not weakly connected, that is there are $x, y \in A$ such that there is no connection $xNz_1Nz_2 \dots z_nNy$ for some $n \in \mathbb{N}$ and $z_i \in A$, then for any possible call sequence σ, G^σ is not complete.*

This means that no protocol is weakly successful on networks that are not weakly connected, i.e. networks that are made of two or more distinct clusters of agents. Intuitively this makes sense because if two groups of agents are not connected to each other in any way, it is not possible for the agents of the one cluster to ever find out the secrets of the agents in the other cluster. We will call single agents that form such a cluster *strong isolated* agents and discuss these further in Section 3.3.

This raises the question what the minimum structural requirements are for protocols to be successful. We state some results from van Ditmarsch et al. [29].

Theorem 2.3.5. [Results from [29]] *We have the following results about protocols for DC graphs:*

- *Protocol ANY is weakly successful on an initial DC graph G iff G is weakly connected.*
- *Protocol CO is strongly successful on an initial DC graph G iff G is weakly connected.*
- *Protocol LNS is strongly successful on an initial DC graph G iff G is a sun graph.*

Where we say that a DC graph G is weakly connected iff for all agents $x, y \in A$ there is a undirected N-path between x and y. We say that a DC graph G is strongly connected iff for all agents $x, y \in A$ there is a N-path between x and y. An DC graph $G = (A, N, S)$ is a sun graph *('almost' strongly connected graph) iff N is strongly connected on $s(G)$, where $s(G)$ is the result of removing all terminal nodes (i.e. nodes x such that there is no y in G with xNy) from G.*

Thus the LNS protocol can be characterized by Definition 2.3.1, or by the class of networks $G = (A, N, S)$ such that G is a sun graph, etc. This means that we also have a semantical characterization of protocols, namely the class of networks on which the protocol terminates successfully. As a result, the protocols determine a partial order on the class of all dynamic communication networks. We have that LNS $\subset$ CO $\subseteq$ ANY [29].

2.4 Implementation

The implementation of the concepts discussed in this chapter can be found in [32]. This technical report explains the Haskell code that was used to create an intuitive understanding about dynamic communication graphs. In particular, with the implementation the computer was asked to generate certain dynamic communication graphs which were then tested for completeness.

We will not elaborate on the implementation by van Eijck and Gattinger [32], but it may be clear that it is the ground work for the implementation of our own framework in Chapter 4.

Chapter 3

Reliability of Agents

```
module AgentTypes where
```

In the previous chapter, we have discussed the framework of dynamic communication. In this framework it was assumed that both the agents and the communication between agents are reliable. This means that agents speak truthfully about their own secret and reliably share the information that they have about others, both secrets and phone numbers. In addition, there was no way that the communication between agents fails. That is, there is no way that not all information is shared in a call.

Even though this is indeed a good start to investigate dynamic communication, the assumption that everyone is reliable may not be very realistic for practical applications. There may be noise on the communication channels and agents may not follow the chosen protocol by keeping (a part of) the information they have to themselves or by manipulating the network. In fact, it may already be questioned how in the old framework agents decide on the protocol that will be executed without outside communication.

In order to broaden the context of dynamic communication, we introduce unreliability into the framework. We do this by adding a qualification to information: information is either true or false. With this, it is possible to let agents spread false information in the network. This opens the door to real-life applications, where unreliability can occur in terms of manipulation or noise.

In this chapter, we will discuss unreliability of agents and introduce agent types and properties that are needed to extend the framework of dynamic communication.

3.1 Unreliability

Before we can make unreliability explicit, we need to alter the definition of a secret. We then introduce a first- and second-order of lying.

3.1.1 Secrets

We extend the framework of dynamic communication by including a quantification of information: information can be true or false. This is made precise by considering secrets as values, namely bits. As such, agents can have a secret that is either 0 or 1. Lying about a secret then boils down to communicating a 0 whereas the secret is a 1, or vice versa.

An unreliable agent can now be uncovered whenever there is conflicting information about this agent in the network. More precisely, this means that the secret of the unreliable agent is considered to be both 0 and 1. Of course, this works in the case of first-order unreliable agents. With second-order unreliable agents, i.e. agents that lie about secrets of others, more information may need to be stored than just secrets. For example next to the gossip about secrets, we may need to require agents to gossip about gossip as well. This means that, in addition to secrets, agents store information about calls that have taken place and information that is shared in those calls. An application of this are swirlds hashgraphs, an alternative to blockchain for cryptocurrencies [6]. We will elaborate on this application in Section 7.2.4.

Naturally, one may wonder whether considering secrets as independent bits is applicable for applications where secrets can be any kind of information. This is not in line with the way false information is recognized in real life situations– namely by conflicting with other pieces of information. The reason why we have chosen for bits is its simplistic approach to modeling false information and its application for unreliability in a network of electronic sensors. We will elaborate more on this in Section 4.1.

3.1.2 First- and second-order lying

Naturally, there is only one way for agents to be reliable: agents do exactly what they are expected to do. However, it is clear that there are many ways to be unreliable. For instance, agents can speak untruthfully about their own secret, but also about the information they have about other agents' secrets. In this thesis, we will call lying about an agent's own secret *first-order lying* and lying about other agents' secrets *second-order lying.* Naturally, the higher the order of lying, the more complicated the network gets. Consider the following Examples 3.1.1 and 3.1.2.

Example 3.1.1. [First-order Unreliability] *Let G be a network of three agents a, b and c such that a and b are reliable and agent c is first-order unreliable. Now consider that the following calls take place: $bc; ac; ab$. Then if we consider agent c to be consistent in her lying (that is, she lies to both agents a and b), she will not be identified. If she, however, is not consistent, agents a and b may come to the conclusion that agent c is unreliable.*

Example 3.1.2. [Second-order Unreliability] *Consider the same situation as in Example 3.1.1, but now agent c is second-order unreliable. That is, agent c lies about the information she has about other agents' secrets. Now in the same call* *sequence $bc; ac; ab$, agent a will receive conflicting information about the secret of agent b and therefore may incorrectly conclude that agent b is unreliable.*

Note that in Example 3.1.2, the result is based on the fact that agent a only assumed first-order unreliability to take place.

To avoid the issue raised in Example 3.1.2 concerning a wrong accusation, we may need to require agents to store more information. For instance to keep track of what was said by whom. It is clear that a higher level of lying complicates the network to a great extent. We will elaborate more on this in Chapter 6.

Another way of being unreliable is not by lying about secrets, but by failing to communicate all the knowledge about phone numbers. This can happen by not giving another agent all the contacts in your phone book, but also by telling other agents incorrect phone numbers. In this way agents manipulate the network by cutting accessibility links between agents. As a result, the network may not only expand but also shrink. This puts the dynamic part of dynamic communication in a whole new perspective. We also call such agents *saboteurs* and we will learn more about them in Section 6.3.

3.2 Agent Types

In this section, we will make the ideas presented previously more precisely. First, we introduce the type of an agent to capture to what extent an agent acts unreliable.

Definition 3.2.1. [Agent Types] *We will consider the following agent types:*

- ***Reliable agent**: an agent that speaks the truth about her own secret and about her knowledge of other agents' secrets and phone numbers.*
- ***Bluffer**: an agent that lies about her own secret with a probability p but is reliable about all other knowledge.*
- ***Liar**: an agent that always lies about her own secret and about her knowledge of other agents' secrets, but does speak the truth about her knowledge of phone numbers.*
- ***Noisy agent**: an agent that lies with a probability p about her own secret and about her knowledge of other agents' secrets, but does speak the truth about her knowledge of phone numbers.*
- ***Saboteur**: an agent that is truthful about secrets but lies about phone numbers, i.e. she can remove connections between agents.*
- ***Leech**: an agent that only receives information, and does not share any herself.*
- ***Fact-checker**: an agent that, from the start, knows all the secrets of all agents and always speaks the truth.*

Where all non-reliable and non-fact checking agents are also collectively called unreliable agents*.*

It may be clear that there are many more options for agents to be unreliable, ranging from combinations of the above to agents that form coalitions to manipulate the reliable agents. Another realistic option is that the type of an agent may change when the network evolves. Consider for instance that an agent convinces another agent to change type, or that an agent, upon learning more about the network, decides herself to change type. For now, we will not consider these options and we will come back to them in Chapter 6.

Going back to our agent types, it is clear that it is negligible that liars lie about their own secret. This is because it is not possible for the other agents to find out that the liar is lying about her own secret by the consistency of the false information. If an agent for instance tells to everyone that her secret is a 1, whereas in fact it is a 0, there is no conflicting information in the network about her secret. Therefore, only the unreliable agent will know that she is lying.

In addition to the agent types described above, we consider a simplistic version of the bluffer: the alternating bluffer.

Haskell Code

```haskell
type Agent = Int

data Agenttype = Reliable | AlternatingBluffer | Bluffer | Liar |
                 NoisyAgent | Saboteur | StrongSaboteur | Leech |
                 FactChecker
  deriving (Eq,Ord,Show)
```

3.2.1 Alternating bluffer

The alternating bluffer is a version of the bluffer that lies in every second call she is involved in. As such, the alternating bluffer approaches the more general bluffer that lies with probability $p = 0.5$ while remaining easily predictable. Alternating bluffers namely follow a clear pattern. This makes the alternating bluffer an interesting agent in the study of unreliable agents by offering a formally precise and feasible framework. Furthermore, the alternating bluffer provides a good first approach to the more general bluffer. We have the following Definition 3.2.2.

Definition 3.2.2. [Alternating Bluffer] *An* alternating bluffer *is a particular kind of bluffer that lies about her own secret every second call. So the probability of lying in call number n, p_n is defined as follows:*

$$p_n = \begin{cases} 0 & \textit{if } n = 0 \\ 0 & \textit{if } p_{n-1} = 1 \\ 1 & \textit{otherwise} \end{cases} \tag{3.1}$$

It does not matter that by default the alternating bluffer speaks the truth in the first call because this has the same result as lying in the first call and flipping the secret of the alternating bluffer.

3.2.2 Saboteur

A saboteur is an agent that unreliable about phone numbers rather than secrets. With this agent, the dynamic component of dynamic communication takes a second form: instead of only enlarging the network with calls, the network may now also shrink. We can then raise the question whether there is a protocol that results in a complete network regardless of what connections are cut by the saboteur.

In [27], sabotage is approached by considering the nodes in the networks as objects and by introducing a cross-model modality referring to submodels of the network from which objects have been removed. A formula is then *possible under sabotage* at a node if it is true at that node in some submodel. In a similar way, a box-operator is defined for sabotage as true if and only if it is true at that node in all submodels.

It remains to specify what connections the saboteur is allowed to cut. There are many different options: cutting any connection, cutting only connections between agents in the contact list of the saboteur, cutting only connections targeted at a certain agent, etc. With the latter we mean that in targeting agent a, a saboteur can cut any connection ab such that $b \in N_a$. We will see in Section 6.3 that this option is very realistic in our setting that requires agents to act knowledge-based. That is, agents decide whom to call depending on the information available to them about the network, i.e. about phone numbers and secrets. As such, it is natural to require the saboteur to do so as well and limit the saboteur in cutting only connections targeted at the agent she is communicating with. In fact, the saboteur can then be considered as an agent lying about phone numbers.

3.2.3 Fact-checker

The idea for the fact-checker stems from the notion of an *oracle* in complexity theory: an oracle is an abstract machine or all-mighty source that is able to solve certain decision problems in a single operation and with very high efficiency. In our dynamic communication networks the fact-checker can be viewed as an agent with whom the agents can verify their information about secrets. Intuitively, we will consider a fact-checker with the limitation that she can only verify information. That is, if an agent a communicates with a fact-checker f, the fact-checker f may only provide the secrets of agents b such that a has prior (possibly wrong) knowledge about the secret of b. For else a single call to the fact-checker would result in an agent learning all the secrets of agents in the network and thus becoming an expert. In other words, agents cannot gather any new information but merely verify the information they have.

It is interesting that a fact-checker opens a wide range of possibilities for dynamic communication networks. As we will see in the next chapter, our framework of dynamic communication with the possibility of error will be based on the assumption that agents have a positive bias toward new information. That is, agents assume new information to be true until it is proved false. In Chapter 6, we will also explore other options for the attitude of agents toward new information. However, it may already be clear that with a fact-checker available agents can easily afford to being a little cautious. This is because agents can always verify their information with the fact-checker. We will see more examples where fact-checkers can make a big difference.

The question remains whether it is realistic to assume that fact-checkers exist. For one, agents need to know the identity of the fact-checker in order to make use of her because otherwise agents would not know that they are talking to the fact-checker instead of another type of agent. Additionally, a fact-checker seems more of an ideal abstraction than an applicable concept in real-life situations. Instead, we may require agents to rate other agents according to their reliability and knowledge and consider the highest rated agent as their "fact-checker". How this works in practice is left for future research.

There are other concerns with the fact-checker. For instance, we might require communication with the fact-checker to come at a certain cost, at least from an efficiency point of view. That can mean that in each call from an agent a to a fact-checker f, agent a can verify the secret of at most one agent $b \in S_a$. Intuitively, it makes sense that verifying information comes at a certain cost.

3.3 Agent Properties

Next to certain types, we also explore properties agents can exhibit. Agents can for instance be consistent, meaning that they "stick with their story"–i.e. agents remember to whom they have lied in the past and will make sure to lie to them again in future communication. Another option for agents is to delete or block the agents that they do not trust. This seems a natural response to finding out that another agent is unreliable. You simply won't talk to them again because you do not trust them! Remember that an agent a distrusts an agent b if she receives contrary information about the secret of agent b, i.e. she will learn that the secret of agent b is both 0 and 1. We will make this definition of distrust more precise in Section 4.2.

Definition 3.3.1. [Consistent Agents] *An agent a is called* consistent *if she is consistent in her communication. That is, whenever she lies to another agent b in the first call ab or ba, she will lie to b again in any future call ab or ba.*

It may be clear that reliable agents are always consistent. On the other hand, consistent unreliable agents thus remember to whom they have and to whom they have not spoken the truth. In this way, consistent unreliable agents can be considered as unreliable agents that do not wish to be discovered.

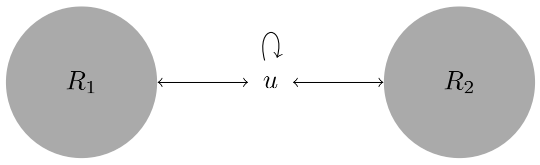
Definition 3.3.2. [Trumpian Agent] *An agent is called* Trumpian *if she does not communicate the phone numbers she has of agents that she distrusts, nor does she contact agents she distrust.*

Definition 3.3.3. [Strong Trumpian Agent] *An agent is called* strong Trumpian *if she is Trumpian and additionally blocks all agents that she distrusts, meaning they cannot contact her anymore either.*

Note that we will not apply the Definition of consistent, Trumpian and strong Trumpian to fact-checkers or leeches. In fact, by default the fact-checker and leech are consistent just like reliable agents.

It can be discussed whether the property of being (strong) Trumpian is a desirable property. In daily life, this property seems quite a natural response: if you find out that someone is unreliable, you might not want to talk to this person again. This is because the unreliability affects trust. However, there are situations in which this property can limit the amount of information an agent learns. This is because even though someone is unreliable, she may still have valuable information. Consider the following Example 3.3.4.

Example 3.3.4. *Consider the network G drawn below consisting of $n + m + 1$ agents such that agent u is first-order unreliable and all agents $r \in R_1$ and all agents $r' \in R_2$ are reliable, where $|R_1| = n$ and $|R_2| = m$. Note that the big gray circles indicate clusters. That is, for all agents $r \in R_1$ we have $R_1 \subseteq N_r$, and for all agents $r' \in R_2$ we have $R_2 \subseteq N_{r'}$. Additionally, we have that agent u has a connection with all agents $r \in R_1 \cup R_2$ (uNr and rNu). Now suppose that every reliable agent is strong Trumpian and has identified agent u as unreliable. That means that all the connections ur and ru for any $r \in R_1 \cup R_2$ will be cut. As a result, the reliable agents will remain* split. *In other words, agents $r \in R_1$ will never learn the number of agents $r' \in R_2$ and vice versa.*



So indeed, it might be that being (strong) Trumpian is undesirable. Yet, the problem is that if the agent type of the unreliable agent is unknown, agents can never be sure about the reliability of the information shared by the unreliable agent. As such, the response of deleting or blocking to unreliability is a simplified version of a change in attitude toward the unreliable agent. Blocking an agent is namely the same as not trusting any information of the agent. To prevent the issue of Example 3.3.4, we may introduce a version of deleting or blocking where the information about secrets of the deleted or blocked agents is not trusted, but the information about phone numbers is.

So far, we have considered properties and types of agents that only concern the agent herself. However, there are also properties that agents can have with respect to the network. Two of those are the properties *terminal* and *isolated.*

Definition 3.3.5. [Terminal Agent] *An agent a is called* terminal *if she initially does not have the phone number of any other agent, except herself. I.e.* $N_a = \{a\}$. *An agent a is called* isolated *if additionally no other agent has the phone number of her. I.e.* $N_a = \{a\}$ *and for all* $b \neq a$ *we have that* $a \notin N_b$.

It is clear that isolated agents are not very interesting as there is no way that they will be connected to the network again in the future. Also note that an initially terminal agent may not remain terminal in the future. This is because when a call sequence is executed, the terminal agent may receive a call and learn the numbers of some agents accordingly.

Haskell Code

In the code, Trumpian and strong Trumpian are referred to as *TrumpianDel* and *TrumpianBlock* as Trumpian agents delete the phone numbers of agents that they distrusts, and strong Trumpian agents block those agents.

```
data Agentproperty = None | Consistent | TrumpianDel | TrumpianBlock
  deriving (Eq,Ord,Show)

type Secret = Bool

type AgentInfo = (Agent, (Agenttype, Agentproperty), Secret)
```

3.4 Network Properties

Just like agents, networks can also have certain properties, ranging from graph-theoretical properties to properties more related to the specific setting of unreliable agents and information sharing. In Chapter 2, we have seen that our networks follow an epistemic approach so that the agents base their decision on what call to make on the information available to them, their own knowledge. Because of this epistemic approach, it is important to specify what information is available to the agents [28]. In dynamic communication as introduced in Chapter 2, agents are unaware of calls that they do not participate in. This means that the network is *asynchronous*. Other options are *synchronous* and *observable networks*.

Definition 3.4.1. [Asynchronous, Synchronous and Observable Networks] *We have the following options*

- ***Asynchronous Network****: agents are unaware of calls that they do not participate in*
- ***Synchronous Network****: agents know how many calls take place, but not who are making the calls*
- ***Observable Network****: agents know exactly which calls take place, although they cannot learn the content of the call*

It may be clear that in other types of networks, more information is available to agents. This then allows for different protocols. For instance protocols that take into account the calls that have already taken place in the observable setting. As a result, communication can become highly efficient.

Because we are interested in the general case of dynamic communication with the possibility of error, we will mainly focus on asynchronous networks. We will discuss more about these options in Section 4.1.

Chapter 4

Gossip with Unreliable Agents

```
module UnreliableGossip where

import AgentTypes
import Data.List
```

In this chapter, we adjust the framework introduced in Chapter 2 to account for unreliable agents in dynamic networks. In particular, we adjust dynamic communication to include alternating bluffers. We first explain the motivation for focussing on the alternating bluffer and our choices for other parameters. After that we will dive into the formal definitions of dynamic communication networks with unreliable agents. We will explain how the definition of a call now alters and allows the reliable agents to identify the unreliable agents. We will then explore what it means for a dynamic network with unreliable agents to be complete and for protocols to be successful.

4.1 Parameters for Unreliable Gossip

As we have seen in the previous chapters, we roughly have the following parameters when discussing dynamic communication with unreliable agents: agents, networks and protocols. In this thesis, we are interested in the application of dynamic communication to networks in which unreliability occurs in the form of unintended, random, memoryless error or noise. For instance networks of sensors that are communicating with each other. Such networks are designed in a way that errors do not occur very often, but neither are necessarily ruled out.

Assumptions:

- *Noise is minimized*

- *Noise occurs unintentionally*
- *Noise occurs randomly*
- *Noise is memoryless*

In this section we will briefly discuss the different options for agents, networks and protocols and motivate our choices. In Chapter 6 we will elaborate on alternative choices and provide extensions of our framework accordingly.

4.1.1 Agents

Until now, we have focused on agent types and agent properties. However, there are more possibilities when considering unreliable agents in dynamic networks. This is because information spreads and therefore agents can have a certain attitude when learning new information. Agents may be *naive*, meaning that they will consider new information to be reliable until proved the contrary, or rather *skeptical*, when they do not. When agents are naive, we also name this the *presumption of innocence* attitude. Typically, the attitude of an agent lays somewhere in between.

Another option is that agents have a preference for *first-hand* over *second-hand* information. First-hand information is information that agents obtain directly (i.e. the secret of an agent b that is obtained by calling agent b), whereas second-hand information is information that is obtained via another agent (i.e. the secret of agent b that is obtained from calling another agent c). In this setting, this preference is mainly interesting when applied to the set of agents that are considered to be unreliable. That is, should agents adopt a form of distrust toward the reliability of other agents? I.e. when an agent a considers agent b unreliable and then makes a call to agent c, should agent c, upon learning this information about the reliability of agent b, agree that agent b is unreliable?

These ideas are captured by the attitude of an agent.

Definition 4.1.1. [Attitude of Agents] *The* attitude *of agents describes how agents perceive new information. This attitude can range between agents being* naive *(agents belief all new information to be reliable until proved the contrary) and agents being extremely* skeptical *(not trusting any new information). Additionally, agents may have a certain preference for* first-hand *over* second-hand *information.*

In our setting unreliability occurs as noise. Therefore, we will focus on agents that are naive and that consider no difference between first-hand and second-hand information. This is because the amount of noise in the network is considered to be minimized. More on the justification for considering this attitude can be found in [13, 7] where different types of belief-revision are considered when false observations may occur—among which the naive attitude.

For the same reason, we will focus on agents that are only at times unreliable about their own secret. In particular, we focus on a simple version of such unreliability: the alternating bluffer. This agent lies about her own secret in every second call. As such, this agent provides a first way of approaching random, unintended noise while remaining formally feasible.

Within our assumption, agent properties are not very relevant. Even though some agents may be unreliable, they are not so intentionally and therefore are still "useful" in the way that they might still have valuable information about the network. For instance, unreliable agents may have valuable information about phone numbers or secrets that some of the other agents do not. For this same reason, (strong) Trumpian agents are not considered in our setting because there is no advantage to cutting agents out. Consistency is also not considered for unreliable agents, as noise typically does not depend on earlier noise. In other words, noise is memoryless. Note though that reliable agents are per definition consistent.

In Section 6.1 we will come back to these choices and consider (strong) Trumpian agents and a first way of allowing a little skepticism in the network.

Assumptions:

- *Unreliable agents are alternating bluffers*
- *Agents are naive, i.e. they consider no preference between first- and second-hand information and consider all new information reliable until the contrary is proved*
- *Unreliable agents are not consistent*
- *Agent do not have the property of being (strong) Trumpian*

4.1.2 Networks

In Chapter 3, we have seen that there are different options to what extent communication between agents can be observed by other agents. For simplicity, we will assume that communication in the networks in asynchronous. This means that agents are unaware of any calls that they are not involved in. Other options that we have already seen are synchronous and observable networks in which more information about the calls taking place in the network is available to the agents not involved in these calls.

Next to asynchronicity, we will consider communication to be costless. This is a very realistic assumption in today's world where information can spread at very low costs over for instance the internet. Before the rise of the internet however, this assumption would not be very feasible. Think of phone booths or writing letters. Even though this costs can now be neglected, there is still some cost to communication we have to consider: efficiency.

This assumption regarding the cost of communication may change when considering fact-checkers as then we may want to put a price on verifying information. This option is left for future research.

Assumptions:

- *Networks are asynchronous, i.e. agents cannot observe calls they are not involved in*
- *Communication is costless—unless from an efficiency point of view*

4.1.3 Protocols

in Chapter 2 we have seen the different protocols that were considered for dynamic communication: ANY, CMO and LNS. With unreliable agents, however, a different approach to protocols may be required. This is because unreliable agents may for instance have bad intentions and work together against the reliable agents. In such a setting, it seems natural that the unreliable agents and reliable agents follow a different protocol.

However, considering unreliability as noise or error, it seems likely to decide that unreliable agents follow the same protocol as reliable agents. This is because the unreliable agents do not know that they are unreliable (or else they could communicate that they are). Therefore, here we will assume that unreliable agents have the same intentions as reliable agents—finding out all the secrets—and therefore follow the same protocol.

Assumptions:

- *Reliable and unreliable agents follow the same protocol*

4.2 Formalisms

To adjust the framework of dynamic communication to account for unreliable agents, we specify the set of reliable agents, $R \subseteq A$, and assign a secret (a bit) to each agent with a function $f : A \to \{0, 1\}$. Additionally, we split the secret relation in three disjoint sets: X, Y, and Z, denoting which agents are considered reliable with a secret 1, with a secret 0 and which agents are considered to be unreliable, respectively.

Definition 4.2.1. [DCU Graph] *A dynamic communication graph involving unreliable agents, a* DCU Graph*, is a quintuple $G = (A, R, f, N, S)$ where A is a set of agents, $R \subseteq A$ is the set of reliable agents, $f : A \to \{0, 1\}$ is an initial distribution of secrets with $f(a) = t$ (meaning that the secret of a is t), $N \subseteq A^2$ is the network relation and $S : A \to \mathcal{P}A \times \mathcal{P}A \times \mathcal{P}A$ is a function assigning to each agent a a triple (X_a, Y_a, Z_a) where $X_a \subseteq A$ is the set of, according to a, reliable agents with a positive secret, $Y_a \subseteq A$ is the set of, according to a, reliable agents with a negative secret and Z_a is the set of unreliable agents, according to a. Note that we assume both N and S to be reflexive (every agents knows her own number and secret) and that all intersections of X_a, Y_a and Z_a are empty.*

Note that when all agents are reliable ($R = A$), a DCU graph is a DC graph with $S_a = X_a \cup Y_a$ for each $a \in A$. An initial DCU graph is a DCU graph such that for each agent a, a knows at least her own phone number and a only knows her own secret.

Definition 4.2.2. [Initial DCU Graph] *An* initial DCU Graph *is a DCU graph $G = (A, R, f, N, S)$ such that for all $a \in A$ we have that $\{a\} \subseteq N_a = \{b \mid aNb\} \subseteq A$ and $X_a \cup Y_a \cup Z_a = \{a\}$ I.e. $S = I_A$.*

Note that throughout this thesis, we will denote DCU graphs G by the quintuple (A, R, f, N, S) where for each $a \in A$ we have $S_a = X_a \cup Y_a \cup Z_a$. Henceforth, in a context where $S = I_A$ or $S_a = \{a\}$ is mentioned, we mean that the network is an initial network or that an agent knows only her own secret, respectively. Thus any initial DCU graph is of the form $G = (A, R, f, N, I_A)$. More generally, whenever we mention S_a, we use X_a, Y_a and Z_a for specifying the agents that are considered reliable with a secret 1, reliable with a secret 0 and unreliable by agent a, respectively.

Just like the case of DC graphs, we have that the secret relation is contained in the number relation. That is, for any call sequence σ, $(X \cup Y \cup Z)^\sigma \subseteq N^\sigma$.

Lemma 4.2.3. *For any DCU graph $G = (A, R, f, N, S)$ where all unreliable agents are alternating bluffers and any call sequence σ, we have that $S^\sigma = (X \cup Y \cup Z)^\sigma \subseteq N^\sigma$.*

Proof. We prove this by induction on σ. The base case follows by observing that $S^\epsilon = (X \cup Y \cup Z)^\epsilon = X \cup Y \cup Z = I_A \subseteq N = N^\epsilon$. For the induction step, assume that $S^\sigma = (X \cup Y \cup Z)^\sigma \subseteq N^\sigma$ (IH) and let xy be a possible call in G^σ. Now suppose that $(a, b) \in S^{\sigma;xy} = (X \cup Y \cup Z)^{\sigma;xy}$. Then either $(a, b) \in (X \cup Y \cup Z)^\sigma$ or $(a, b) \in (X \cup Y \cup Z)^{\sigma;xy}$ but $(a, b) \notin (X \cup Y \cup Z)^\sigma$. In the first case $(a, b) \in N^\sigma$ and hence $(a, b) \in N^{\sigma;xy}$ follow directly from the IH. In the second case, we can assume without loss of generality that $a = x$. So $(x, b) \in (X \cup Y \cup Z)^{\sigma;xy}$ but $(x, b) \notin (X \cup Y \cup Z)^\sigma$. But then it must be that $(y, b) \in (X \cup Y \cup Z)^\sigma$ and by the IH, $(y, b) \in N^\sigma$. So as all phone numbers and secrets are shared in the call xy, $(x, b) = (a, b) \in N^{\sigma;xy}$.

□

Of course, this proof depends on the fact that we only consider alternating bluffers in our framework. We will see that if a saboteur occurs in a DCU network G, Lemma 4.2.3 fails to hold. This is because the saboteur cuts the N relation while keeping the S relation intact.

Haskell Code

For the implementation, recall that the type AgentInfo stores the type, property and secret of an agent in a triple $(agent, (type, property), secret)$.

```haskell
type DCUagents  = [AgentInfo]
type DCUnumbers = [(Agent, Agent)]

type DCU = (DCUagents, DCUnumbers)
```

Thus, the type *DCU* describes the agents and the number-relation of the initial network.

```
type Item = (Agent, ([Agent], ([Agent], [Agent], [Agent]), Agenttype,
             Agentproperty))

type Table = [Item]
```

An element of the type Item looks like $(a, (N_a, (X_a, Y_a, Z_a), \textit{type}, \textit{property}))$. We initialize a Table given a DCU graph as follows.

```
initialize :: DCU -> Table
initialize ([]                                   , _       ) = []
initialize ((a, (aType, aProp), secret):rest, numbers) =
  sort $ (a, (aN, (aX, aY, []), aType, aProp)) : initialize (rest,
      numbers)
  where
    aN = sort $ check numbers
    check []           = []
    check ((b,c):rest2) = if b == a
                          then c : check rest2
                          else check rest2
    aX = [ a | secret     ]
    aY = [ a | not secret ]
```

4.2.1 Updating networks with calls

When a call from agent a to agent b takes place in the network, information is shared between them. This means that they both update their contact lists (N_a and N_b, respectively) and update their sets X, Y and Z (X_a, Y_a, Z_a and X_b, Y_b and Z_b, respectively). Whenever a conflict occurs (i.e. agent a considers the secret of an agent c to be a 0 whereas agent b considers it to be a 1, or vice versa), both agents a and b will decide that the agent subject to the conflict (agent c) is unreliable. In this way, agents can identify the unreliable agents.

Definition 4.2.4. [Call] *Let $G = (A, R, f, N, S)$ be a DCU Graph, $a, b \in A$ and Nab. The* call *ab maps G to $G^{ab} = (A, R, f, N^{ab}, S^{ab})$ defined by*

$$N_c^{ab} = \begin{cases} N_a \cup N_b & \textit{if } c \in \{a, b\} \\ N_c & \textit{otherwise} \end{cases} \tag{4.1}$$

$$S_c^{ab} = \begin{cases} (X', Y', Z') & \textit{if } c \in \{a, b\} \\ (X_c, Y_c, Z_c) & \textit{otherwise} \end{cases} \tag{4.2}$$

Where

$$X' = X_a \cup X_b \setminus \big((X_a \cap Y_b) \cup (X_b \cap Y_a) \cup (X_a \cap Z_b) \cup (X_b \cap Z_a)\big)$$

$$Y' = Y_a \cup Y_b \setminus \big((X_a \cap Y_b) \cup (X_b \cap Y_a) \cup (Y_a \cap Z_b) \cup (Y_b \cap Z_a)\big)$$

$$Z' = (Z_a \cup Z_b) \cup \big((Y_a \cap X_b) \cup (Y_b \cap X_a)\big)$$

We say that a call ab is possible in a network if agent a has the phone number of agent b. Naturally, we will only consider possible calls in our framework.

Definition 4.2.5. [Possible Call] *A call ab is* possible *in a network $G = (A, R, f, N, S)$ iff $(a, b) \in N$.*

In order to explain what happens if an agent a makes a call to an unreliable agent b that she distrusts (i.e. $b \in Z_a$), consider the following Example 4.2.6.

Example 4.2.6. *Let $G = (A, R, f, N, S)$ be a DCU graph with $A = \{a, b\}$, $R = \{a\}$, $f(a) = f(b) = 0$, $N = \{(a, b)\}^R$ where Q^R is the reflexive closure of the set Q and $S(a) = (\emptyset, \{a\}, \{b\})$, $S(b) = (\emptyset, \{b\}, \emptyset)$. After the call ab, G get mapped to $G^{ab} = (A, R, f, N^{ab}, S^{ab})$ where $N^{ab} = \{(a, b), (b, a)\}^R$ and $S^{ab}(a) = S^{ab}(b) = (\emptyset, \{a\}, \{b\})$. I.e. agent b learns that she herself is unreliable.*

$$\begin{tikzcd}
a \arrow[loop above] \arrow[r] & b \arrow[loop above]
\end{tikzcd}$$

From Example 4.2.6 it is clear that in such a situation, the unreliable agent b will find out that she is unreliable. One may question the desirability of this consequence of Definition 4.2.4. To see why it might be desirable, consider again a network of sensors in which some of the sensors are noisy. When sensors then learn that they are themselves unreliable, they may, as a consequence, stop sharing their observed information (their own secret). This can be a wanted property of such a network. But also for unreliable agents with bad intentions this can be an advantageous property. Such agents then namely find out that they have been "uncovered".

It may be clear that Definition 4.2.4 relies heavily on the attitude of agents to perceive new information as reliable until the contrary is proved: the *presumption of innocence* or *naive* attitude. Consider the following Example 4.2.7.

Example 4.2.7. *Let $G = (A, R, f, N, S)$ be a DCU graph with $A = \{a, b\}$, $R = \{a\}$, $f(a) = f(b) = 0$, $N = \{(a, b)\}^R$ where Q^R is the reflexive closure of the set Q and $S(a) = (\emptyset, \{a\}, \emptyset)$, $S(b) = (\emptyset, \{b\}, \emptyset)$. After the call ab, G get mapped to $G^{ab} = (A, R, f, N^{ab}, S^{ab})$ where $N^{ab} = \{(a, b), (b, a)\}^R$ and $S^{ab}(a) = S^{ab}(c) = (\emptyset, \{a, b\}, \emptyset)$. I.e. agent a considers agent b to be reliable.*

Agents may end up having wrong beliefs about other agents because of this naive attitude. Only another agent that has contradictory information about the secret of b can change this belief. In Section 6.2, we investigate different attitudes of agents toward new information. In particular, we look at the case in which agents do not simply adopt other agents' distrust. In other words, agents only use first hand information to decide who is unreliable.

Note that for certain agent types (leeches and fact-checkers), there are exceptions for updating the sets N_a, X_a, Y_a and Z_a. Leeches do namely not share any information, whereas fact-checkers only share partial information. How this changes Definition 4.2.4 is left for future research.

Haskell Code

Before we can write the Haskell code for Definition 4.2.4, we need to define functions for merging lists and checking if a call is possible.

```
-- merging lists
merge :: [Int] -> [Int] -> [Int]
merge xs     []     = xs
merge []     ys     = ys
merge (x:xs) (y:ys) =
  case compare x y of
    EQ -> x: merge xs     ys
    LT -> x: merge xs     (y:ys)
    GT -> y: merge (x:xs) ys
```

We only allow calls that are possible.

```
-- check if call is possible
possible :: Table -> (Agent,Agent) -> Bool
possible table (a, b) = a /= b && b `elem` aN where
  Just (aN, (_aX, _aY, _aZ), _aType, _aProp) = lookup a table
```

Now we can implement Definition 4.2.4. Recall that leeches do not share any information and that fact-checkers only share those secrets that are already part of X_a or Y_a for the information requesting agent a. Additionally, as fact-checkers do not initiate any call, they do not update their number relation. This means that they will not be able to initiate a call in the future, because they do not have any agent in their contact list.

```
-- update the Table with a call
call :: (Agent, Agent) -> Table -> [Table]
call (a,b) table = let
  Just (aN, (aX, aY, aZ), aType, aProp) = lookup a table
  Just (bN, (bX, bY, bZ), bType, bProp) = lookup b table

  -- Leeches do not share numbers
  -- Fact-checkers do not update their numbers
  -- (and therefore do also not share any)
  new_aN | aType == FactChecker = aN
         | bType == Leech       = aN
         | otherwise            = merge aN bN
  new_bN | bType == FactChecker = bN
         | aType == Leech       = bN
         | otherwise            = merge aN bN

  disagreementsX = (aX `intersect` bY) `union` (bX `intersect` aY)
                                       `union` (aX `intersect` bZ)
                                       `union` (bX `intersect` aZ)

  disagreementsY = (aX `intersect` bY) `union` (bX `intersect` aY)
                                       `union` (aY `intersect` bZ)
                                       `union` (bY `intersect` aZ)

  -- Fact-checkers do not tell which agents are unreliable
  -- (this could be an option as well)
  new_aX | bType == Leech       = aX
         | bType == FactChecker = aX `union` [b] \\ (aX `intersect` bZ)
         | otherwise            = merge aX bX \\ disagreementsX
  new_bX | aType == Leech       = bX
         | aType == FactChecker = bX `union` [a] \\ (bX `intersect` aZ)
         | otherwise            = merge aX bX \\ disagreementsX
```

```
    new_aY | bType == Leech       = aY
           | bType == FactChecker = aY \\ (aY `intersect` bZ)
           | otherwise            = merge aY bY \\ disagreementsY
    new_bY | aType == Leech       = bY
           | aType == FactChecker = bY \\ (bY `intersect` aZ)
           | otherwise            = merge aY bY \\ disagreementsY

    new_aZ | bType == Leech       = aZ
           | bType == FactChecker = aZ `union` (aX `intersect` bZ)
                                       `union` (bX `intersect` aZ)
           | otherwise            = merge aZ bZ `union` (aX `intersect` bY)
                                               `union` (bX `intersect` aY)
    new_bZ | aType == Leech       = bZ
           | bType == FactChecker = bZ `union` (aX `intersect` bY)
                                       `union` (bX `intersect` aY)
           | otherwise            = merge aZ bZ `union` (aX `intersect` bY)
                                               `union` (bX `intersect` aY)

    stripTable = table \\ [(a, (aN, (aX, aY, aZ), aType, aProp)),
                           (b, (bN, (bX, bY, bZ), bType, bProp))]
    newTable   = sort $ (a, (new_aN, (new_aX, new_aY, new_aZ), aType, aProp
        )) :
                       (b, (new_bN, (new_bX, new_bY, new_bZ), bType, bProp
                           )) :
                       stripTable
  in
    if possible table (a,b)
       then typeBehavior [(a,aType,aProp),(b,bType,bProp)] newTable (a,b)
       else error "forbidden call!"
```

Because by assumption not all agents need be reliable, this is not enough. We still have to check and update the behavior of the types and properties of the agents in the network. The code for this can be found in next section.

4.3 Alternating Bluffer

Recall that the alternating bluffer is an agent that only lies about own her secret in every second call she is involved in. We have chosen to focus on this particular agent because it is aa predictable version of the bluffer and a good approach to networks in which unreliability occurs in the form of error or noise.

Given the distribution of secrets, we need to configure the sets X_a and Y_a for each agent a. We specify this for reliable agents and alternating bluffers.

Definition 4.3.1. [Reliable Agent] *A* reliable agent *a is such that $a \in X_a$ if $f(a) = 1$ and $a \in Y_a$ if $f(a) = 0$.*

Definition 4.3.2. [Alternating Bluffer] *An* alternating bluffer *a is such that initially $a \in X_a$ if $f(a) = 1$ and $a \in Y_a$ if $f(a) = 0$. Then after each call σ involving agent a, we update X_a^σ and Y_a^σ to $(X_a^\sigma)'$ and $(Y_a^\sigma)'$ defined by*

$$(X_a^\sigma)' = \begin{cases} X_a^\sigma \cup \{a\} & \text{if } a \in Y_a \\ X_a^\sigma \setminus \{a\} & \text{if } a \in X_a \end{cases} \tag{4.3}$$

$$(Y_a^\sigma)' = \begin{cases} Y_a^\sigma \cup \{a\} & \text{if } a \in X_a \\ Y_a^\sigma \setminus \{a\} & \text{if } a \in Y_a \end{cases} \tag{4.4}$$

Recall that by default it does not matter that the alternating bluffer speaks truthfully in the first call. This would namely be the same situation as when the secret of the alternating bluffer is flipped and she lies in the first call.

Haskell Code

We can now finish the Haskell code for a call by letting the agents update their information with their types and properties.

```haskell
-- let's the agents behave according to their types and properties
typeBehavior :: [(Agent, Agenttype, Agentproperty)] -> Table -> (Agent,
     Agent) -> [Table]
typeBehavior []                      table _ = [table]
typeBehavior ((a,aType,aProp):rest) table (b,c)
  | aType == AlternatingBluffer && aProp == None
      = typeBehavior rest (alternate a table) (b,c)
  | aType == AlternatingBluffer && aProp == TrumpianDel
      = typeBehavior rest (alternate a (censorDelete a table)) (b,c)
  | aType == AlternatingBluffer && aProp == TrumpianBlock
      = typeBehavior rest (alternate a (censorBlock a table)) (b,c)
  | aType == Reliable           && aProp == None
      = typeBehavior rest table (b,c)
  | aType == Reliable           && aProp == TrumpianDel
      = typeBehavior rest (censorDelete a table) (b,c)
  | aType == Reliable           && aProp == TrumpianBlock
      = typeBehavior rest (censorBlock a table) (b,c)
  | aType == Saboteur           && aProp == None
      = concat [ typeBehavior rest (cut unluckyPair table) (b,c) |
                        unluckyPair <- filteredConnections table a (b,c)]
  | otherwise
      = typeBehavior rest table (b,c)
```

We define a function to capture Definition 4.3.2.

```haskell
-- a function for Alternating Bluffers
alternate :: Agent -> Table -> Table
alternate a table = sort $ (a, (aN, (newX, newY, aZ), aType, aProp)):
     newtable where
  Just (aN, (aX, aY, aZ), aType, aProp) = lookup a table
  newtable = table \\ [(a, (aN, (aX, aY, aZ), aType, aProp))]
  newX | a `elem` aX = aX \\ [a]
       | a `elem` aY = sort $ a:aX
       | otherwise   = aX

  newY | a `elem` aY = aY \\ [a]
       | a `elem` aX = sort $ a:aY
       | otherwise   = aY
```

4.4 Complete Networks

In this section, we explore what it means for dynamic networks with unreliable agents to be complete. In short, we adopt Definition 2.2.2 that captures completeness in the reliable case to a version that takes into account knowledge about the unreliable agents. We can distinguish between the following versions of completeness.

- All agents know all secrets of all reliable agents

- All reliable agents know all secrets of all reliable agents
- All agents know all secrets of all reliable agents and know the set of reliable agents
- All reliable agents know all secrets of all reliable agents and know the set of reliable agents

In the following, we will denote the first point as *completeness*, the third point as *strong completeness* and the second and fourth point as completeness and strong completeness *restricted to reliable agents*, respectively.

Definition 4.4.1. [Completeness] *A DCU graph $G = (A, R, f, N, S)$ is* complete *if for all $a \in A$ if $S(a) = (X_a, Y_a, Z_a)$ then $R \subseteq X_a \cup Y_a$ and $Z_a \subseteq A \setminus R$.*

Definition 4.4.2. [Completeness Restricted to Reliable Agents] *A DCU graph $G = (A, R, f, N, S)$ is* complete restricted to reliable agents *if for all $a \in R$ if $S(a) = (X_a, Y_a, Z_a)$ then $R \subseteq X_a \cup Y_a$ and $Z_a \subseteq A \setminus R$.*

We say that an agent a knows all the reliable secrets (i.e. the secrets of the reliable agents) if $R \subseteq X_a \cup Y_a$. Similarly, we say that an agent a can identify the set of unreliable agents if $Z_a = A \setminus R$.

In fact, in these definitions, $Z_a \subseteq A \setminus R$ is a direct consequence of $R \subseteq X_a \cup Y_a$ because by definition we have all intersections of X_a, Y_a and Z_a are empty. Thus we have that $(X_a \cup Y_a) \cap Z_a = (X_a \cap Z_a) \cup (Y_a \cap Z_a) = \emptyset \cup \emptyset = \emptyset$ and since $R \subseteq X_a \cup Y_a$, we have that $R \nsubseteq Z_a$ and hence $Z_a \subseteq A \setminus R$.

Note that we intuitively have that completeness implies completeness restricted to reliable agents. We will see that the same holds for strong completeness and successful identification. Then, by contra-position, if a network G is not (strong) complete restricted to reliable agents, it is also not (strong) complete.

We say that a DCU graph $G = (A, R, f, N, S)$ *successfully identifies* the unreliable agents if all agents know the set of unreliable agents.

Definition 4.4.3. [Successful Identification] *A DCU graph $G = (A, R, f, N, S)$* successfully identifies *the unreliable agents if for all $a \in A$ with $S(a) = (X_a, Y_a, Z_a)$, it holds that $Z_a = A \setminus R$.*

So a DCU graph successfully identifies the unreliable agents if every agent, including the unreliable agents themselves, identify the unreliable agents.

Definition 4.4.4. [Successful Identification Restricted to Reliable Agents] *A DCU graph $G = (A, R, f, N, S)$* successfully identifies restricted to reliable agents *the unreliable agents if for all $a \in R$ with $S(a) = (X_a, Y_a, Z_a)$, it holds that $Z_a = A \setminus R$.*

Strong completeness is now defined as follows.

Definition 4.4.5. [Strong Completeness] *A DCU graph $G = (A, R, f, N, S)$ is* strong complete *if for all $a \in A$ with $S(a) = (X_a, Y_a, Z_a)$, we have that $X_a \cup Y_a = R$ and $Z_a = A \setminus R$.*

Definition 4.4.6. [Strong Completeness Restricted to Reliable Agents] *A DCU graph $G = (A, R, f, N, S)$ is* strong complete restricted to reliable agents *if for all $a \in A$ with $S(a) = (X_a, Y_a, Z_a)$, we have that $X_a \cup Y_a = R$ and $Z_a = A \setminus R$.*

We have the following corollaries. The proofs are straightforward as $R \subseteq X_a \cup Y_a$ and $Z_a = A \setminus R$ imply that $X_a \cup Y_a = R$ because by definition all intersections of X_a, Y_a and Z_a are empty.

Corollary 4.4.7. *A DCU graph G is* strong complete *if G is complete and successfully identifies the unreliable agents.*

Corollary 4.4.8. *A DCU graph G is* strong complete restricted to reliable agents *if G is complete and successfully identifies the unreliable agents restricted to reliable agents.*

The justification for including successful identification in the definition of completeness arises from having the possibility of positive introspection for knowledge. In fact, positive introspection also requires agents to know the set of all agents A in the network. For else, it might be that there is an agent a with $R \subseteq X_a \cup Y_a$ and an agent $c \in A \setminus R$ such that $c \notin N_a$ and as a consequence $c \notin X_a \cup Y_a \cup Z_a$. I.e. agent a does not know about the presence of agent c in the network and may therefore incorrectly consider that she knows the secrets of all the reliable agents and can identify all unreliable agents, i.e. the network is strong complete. But of course, the network cannot be strong complete as agent a fails to successfully identify all the unreliable agents—in particular agent c.

When the set of agents A is known to the agents, this cannot occur. This is because agent a would then know the presence of agent c even though $c \notin N_a$. For strong completeness to be achieved, we need that $X_a \cup Y_a \cup Z_a = A$ and in particular $c \in X_a \cup Y_a \cup Z_a$. This illustrates the value of considering strong completeness over mere completeness.

Going back to the issue of positive introspection; even though in a strong complete network each agent can identify the unreliable agents, how can the agents be certain that these are all the unreliable agents? Per definition, agents assume that other agents are reliable until proved the contrary. Thus can an agent a be sure that indeed $X_a \cup Y_a = R$ and that there is no $c \in A \setminus R$ such that a considers c reliable ($c \in X_a \cup Y_a$)? The answer is no, not in our setting. Let alone that agents will know whether the other agents know that the network is strong complete. So even though strong completeness is a step toward positive introspection, this illustrates the big gap between strong completeness, positive introspection and even common knowledge. More about these issues can be found in [21], where the question is raised how many calls it takes to obtain higher-order shared knowledge of secrets. A correct and optimal protocol is then given to obtain such higher-order shared knowledge.

Even though it seems now that we should only be interested in strong completeness, there are still cases in which completeness should be considered. For instance when a saboteur occurs in the network. With the current framework, agents will never find out who the saboteur in the network is (because a saboteur does not cause contrary information in the network). Therefore strong completeness as defined above cannot be achieved. It is left for future research to adjust the definitions in a way that saboteurs can be uncovered—for instance by considering a network that is not asynchronous and in which agents can observe which connections are cut by the saboteur.

Another variation we consider are the definitions of completeness and strong completeness restricted to the reliable agents. On the one hand, for instance in the case of faulty sensors in a network, it seems desirable to achieve a situation in which the faulty sensors can identify themselves as unreliable. However, when we consider different intentions, completeness or strong completeness restricted to the reliable agents may already be sufficient.

We get the following hierarchy of completeness.

Proposition 4.4.9. *We have the following hierarchy of completeness for DCU graphs G:*
strong complete $\Rightarrow$ strong complete restricted to reliable agents $\Rightarrow$ complete restricted to reliable agents
strong complete $\Rightarrow$ complete $\Rightarrow$ complete restricted to reliable agents

Haskell Code

To be able to check for the different forms of completeness, we first need a function to map a Table to the set of all agents and to the set of reliable agents.

```haskell
-- maps the table to a list of its agents
agents :: Table -> [Agent]
agents = sort . map fst

-- find reliable agents from table
reliable :: Table -> [Agent]
reliable [] = []
reliable ((a, (_aN, (_aX, _aY, _aZ), aType, _aProp)):rest) =
  if aType == Reliable then a : reliable rest
    else reliable rest
```

We write functions to test the knowledge of the agents, defined as $X_a \cup Y_a$ for an agent a.

```haskell
-- map table to knowledge of each agent
-- where knowledge of an agent a is X_a union Y_a
knowledge :: Table -> [(Agent,[Agent])]
knowledge [] = []
knowledge ((a, (_aN, (aX, aY, _aZ), _aType, _aProp)):rest) =
  (a, merge aX aY) : knowledge rest
```

Now we can check for completeness and completeness restricted to reliable agents.

```haskell
-- table complete if all agents know all secrets of reliable agents
-- note: they may know more secrets
complete :: Table -> Bool
complete table = all (subsetAgs (reliable table)) aXYs where
  aXYs   = map snd (knowledge table)
  -- check if a list is a subset of another list
  subsetAgs :: [Agent] -> [Agent] -> Bool
  subsetAgs [] _ = True
  subsetAgs (a:as) list = a `elem` list && subsetAgs as list

-- table completeR if all reliable agents know all secrets of reliable
    agents
-- note: they may know more secrets
completeR :: Table -> Bool
completeR table = all (subsetAgs (reliable table)) aXYs where
  rTable = filter (\x -> fst x `elem` reliable table) table
  aXYs   = map snd (knowledge rTable)
  -- check if a list is a subset of another list
  subsetAgs :: [Agent] -> [Agent] -> Bool
  subsetAgs [] _ = True
  subsetAgs (a:as) list = a `elem` list && subsetAgs as list
```

To test for strong completeness, we have to find the unreliable agents in a network and check whether this corresponds to the distrusted set of agents of each agent.

```
-- find unreliable agents from table
unreliable :: Table -> [Agent]
unreliable [] = []
unreliable ((a, (_aN, (_aX, _aY, _aZ), aType, _aProp)):rest) =
  if aType /= Reliable then a : unreliable rest
    else unreliable rest

-- map table to list of distrusted agents
distrust :: Table -> [(Agent,[Agent])]
distrust [] = []
distrust ((a, (_aN, (_aX, _aY, aZ), _aType, _aProp)):rest) =
  (a, aZ) :distrust rest

-- table correctly identifies unreliable agents if all agents distrust
-- the unreliable agents
identifyUnreliable :: Table -> Bool
identifyUnreliable table = all (== unreliable table) aZs where
  aZs    = map snd (distrust table)

-- table correctly identifies unreliable agents restricted to reliable
    agents
-- if all reliable agents distrust the unreliable agents
identifyUnreliableR :: Table -> Bool
identifyUnreliableR table = all (== unreliable table) aZs where
  rTable = filter (\x -> fst x `elem` reliable table) table
  aZs    = map snd (distrust rTable)

-- test for strong completeness
strongComplete :: Table -> Bool
strongComplete table = complete table && identifyUnreliable table

strongCompleteR :: Table -> Bool
strongCompleteR table = completeR table && identifyUnreliableR table
```

4.5 Protocols

Here we continue on our discussion of protocols and implement them in our framework. We are interested only in protocols that are knowledge-based, meaning that an agent can decide for herself what call to make based on the information available to her. Next to the ANY, CMO and LNS protocols discussed in Chapter 2 we consider the LNS protocol restricted to reliable agents, LNSR.

The protocol LNSR says that a call ab can take place if the protocol condition for LNS is satisfied and additionally a is a reliable agent. For this protocol to take place, it may seem necessary for the unreliable agent to know that she is unreliable. Because else they would not know that they do not satisfy the protocol condition. That this is not the case can partly be justified by the fact that unreliable agents are considered as faulty agents and may therefore have lost their power to communicate. It may be questioned whether the unreliable agents would know that their ability to communicate is restricted. It may just be that from their point of view the communication is successful, whereas it is not from the point of view of the receiver. For example, for communicating sensors there may be some noise on the channels that restricts communication in one direction. So information can be received, but cannot be delivered.

Definition 4.5.1. [Protocol LNSR] *The protocol* LNSR *is as follows: while* $G = (A, R, f, N, S)$ *is not complete, select any call* $xy \in N$ *such that* $x \neq y$, x *is reliable and* x *does not yet know* y*'s secret. The protocol condition for LNSR is* $\pi(x, y) = x \in R \wedge \neg S^{\sigma}xy$.

A way around this would be to require agents to perform "check" calls. That is, after communicating the secrets, they communicate back on the information received. This relates closely to the Byzantine Generals Problem as introduced in [20], or the more simplified Two Generals Problem.

Definition 4.5.2. [Two Generals Problem] *Consider that there are two generals in the Byzantine army that need to decide whether or not to attack. If both generals attack or both general do not attack, this is fine. However, if one of the generals attacks alone, he will be defeated because he does not have enough forces to win by himself. The question is then raised how to coordinate such an attack. As this time is prior-radio, the generals need to send a messenger telling each other their decision. But then if the messenger is captured, the other general never receives the message. Thus, the receiving general needs to send a reply to let the sending general know he received the message. But what if the reply is lost? It is clear that the generals could spend eternity replying to each other without ever knowing for sure they are in agreement.*

In computing, the Two Generals Problem is a thought experiment meant to highlight the problems faced with coordinating a joint action (the *attack*) over an unreliable connection. In particular, it emphasizes the importance of common knowledge, defined between two agents a and b as "a knows φ and b knows φ and a knows that b knows that φ and b knows that a knows that φ and a knows that b knows that a knows φ ...". Even though by performing many check calls common knowledge can be approached, it can never be achieved. Thus in opting for a protocol that provides 100% security, this idea fails.

With the Two Generals Problem we touch upon the notion of security of communication. We will elaborate more on security protocols in Section 7.1.2.

It may be clear that unreliable agents complicate the whole situation. Consider for instance Example 2.1.1 in Chapter 2 with the following distribution of secrets and agents.

Example 4.5.3. *Let $G = (A, R, f, N, S)$ be an initial DCU graph where $A = \{a, b, c\}$, $R = \{a, b\}$, $f(a) = f(c) = 1$, $f(b) = 0$ and $N = \{(a, b), (b, c)\}^R$. We let c be an unreliable agent of the type alternating bluffer. Now executing the same LNS-permitted call sequence $\sigma = ab; bc; ac$ results in $G^\sigma = (A, R, f, N^\sigma, S^\sigma)$ where $N^\sigma = \{a, b \mid a, b \in A\}$, $S^\sigma_a = (\{a\}, \{b, c\}, \emptyset)$, $S^\sigma_b = (\{a, c\}, \{b\}, \emptyset)$, $S^\sigma_c = (\{a\}, \{b\}, \emptyset)$. I.e. both agent a and b consider agent c to be reliable, even though they have different information of her secret.*

$$\begin{tikzcd}
a \arrow[r] \arrow[loop above] & b \arrow[r] \arrow[loop above] & c \arrow[loop above]
\end{tikzcd}$$

From Example 4.5.3 it is clear that the protocols considered in Section 2.3 fail to correctly identify the unreliable agents in a network. Interestingly, the protocol would have resulted in a strong complete network if instead of agent c, agent b would be the unreliable agent. Because in that case, in the final call ac both agent a and c would have concluded that agent b is unreliable as they communicate different information about agent c's secret to each other.

Recall that we had different versions of success in Chapter 2. We adjust these for our framework.

Definition 4.5.4. [Success] *Let a DCU graph $G = (A, R, f, N, S)$ and a protocol P be given. A finite call sequence $\sigma \in P_G$ is* successful (restricted to reliable agents) in the (strong) complete sense *if G^σ is (strong) complete (restricted to reliable agents).*

- *P is* strongly successful (restricted to reliable agents) in the (strong) complete sense *on G if all P-maximal $\sigma \in P_G$ are (strong) successful (restricted to reliable agents).*

- *P is* weakly successful (restricted to reliable agents) in the (strong) complete sense *on G if there is a $\sigma \in P_G$ that is (strong) successful (restricted to reliable agents).*

- *P is* unsuccessful (restricted to reliable agents) in the (strong) complete sense *on G if there is no $\sigma \in P_G$ that is (strong) successful (restricted to reliable agents).*

Where a P-maximal sequence σ on G is a sequence σ that is P-permitted on G and that is infinite or such that no call is P-permitted on G^σ.

Haskell Code

```
data Protocol = ANY | CMO | LNS | LNSR
  deriving (Eq,Ord,Show)

-- a node is a table with a call history and a protocol
type Node = (Table,Seq,Protocol)

-- a type for calling sequences
type Seq = [(Agent,Agent)]

calls :: Seq -> Table -> [Table]
calls [] table = [table]
calls cs table = concatMap (calls (tail cs)) (call (head cs) table)
```

Before we can compute the call sequences that result in a (strong) complete (restricted to reliable agents) network, we have to compute which calls are permitted according to a certain protocol.

```
-- check whether a call sequence is permitted according to a Protocol
permitted :: Node -> (Agent, Agent) -> Bool
permitted (table, _    , ANY) (a,b) = possible table (a,b)
permitted (table, hist, CMO) (a,b) =
  possible table (a,b) && not ((a,b) `elem` hist || (b,a) `elem` hist)
permitted (table, _    , LNS) (a,b) =
  possible table (a,b) && b `notElem` merge (merge aX aY) aZ
  where
    Just (_aN, (aX, aY, aZ), _aType, _aProp) = lookup a table
permitted (table, hist, LNSR) (a,b) =
  permitted (table, hist, LNS) (a,b) && a `elem` reliable table

-- find all the factcheckers in a table
factcheckers :: Table -> [Agent]
factcheckers [] = []
factcheckers ((c, (_cN, (_cX, _cY, _cZ), cType, _cProp)):rest)
 | cType == FactChecker = c : factcheckers rest
 | otherwise            =     factcheckers rest
```

Now we compute all candidates for a next call.

```
-- compute all calls that are permitted in a table
candidates :: Node -> Seq
candidates (table, hist, prot) = let
  ags    = agents table
  pairs  = [(x,y) | x <- ags, y <- ags, x /= y ]
 in
  filter (permitted (table, hist, prot)) pairs
```

We use a depth-first search algorithm to find call sequences that complete the network.

```
-- a depth first search algorithm that proceeds by building a search tree
search :: (node -> [node])
       -> (node -> Bool) -> [node] -> [node]
search _        _     [] = []
search children goal (x:xs)
  | goal x    = x : search children goal xs
  | otherwise = search children goal (children x ++ xs)

verboseSearch :: Show node => (node -> [node])
       -> (node -> Bool) -> [node] -> IO ()
verboseSearch _        _     [] = return ()
verboseSearch children goal (x:xs)
  | goal x    = do
      putStrLn $ "found one: " ++ show x
      verboseSearch children goal xs
  | otherwise = do
      putStrLn $ "not done:" ++ show x
      putStrLn "children:"
      mapM_  print (children x)
      verboseSearch children goal (children x ++ xs)
```

Code for printing the history of a node.

```
-- print which calls have taken place in a Node
showHistOf :: Node -> IO()
showHistOf = print . snd' where
  snd' (_, x, _) = x
```

We distinguish between four ways of solving a graph:

- The graph is *complete*
- The graph is *strong complete* (i.e. the graph is complete and successfully identifies the unreliable agents)
- The graph is *complete restricted to reliable agents*
- The graph is *strong complete restricted to reliable agents*

```
-- four ways for solving a node:
-- complete, strong complete, complete restricted to reliable agents and
-- strong complete restricted to reliable agents
solved :: Node -> Bool
solved (table, _, _) = complete table

solvedStrong :: Node -> Bool
solvedStrong (table, _, _) = strongComplete table

solvedR :: Node -> Bool
solvedR (table, _, _) = completeR table

solvedStrongR :: Node -> Bool
solvedStrongR (table, _, _) = strongCompleteR table
```

Code for searching for a solution. We first want to extend the node with all possible calls.

```
extendNode :: Node -> [Node]
extendNode (table, hist, prot) =
  [(newTable, hist ++ [(x,y)], prot) | (x,y) <- candidates (table, hist,
                                                               prot)
                                                                    ,
                                       newTable <- call (x,y) table]
```

Using the search tree to find solutions.

```
-- similarly, we have four versions of solveNs, solveAndShow and
     solveShowHistOf
solveNs :: [Node] -> [Node]
solveNs = search extendNode solved

solveNsStrong :: [Node] -> [Node]
solveNsStrong = search extendNode solvedStrong
```

```
solveNsR :: [Node] -> [Node]
solveNsR = search extendNode solvedR

solveNsStrongR :: [Node] -> [Node]
solveNsStrongR = search extendNode solvedStrongR
```

Code for printing the call sequences that result in a complete network in one of the four ways. Note that this means that a protocol is *weakly successful* on a graph, see definition 2.3.2.

```
-- print calling sequences that (strongly) complete (restricted to
-- reliable agents) the network
solveAndShow :: Table -> Protocol -> IO()
solveAndShow t p = solveShowHistOf [(t, [], p)]

solveAndShowStrong :: Table -> Protocol -> IO()
solveAndShowStrong t p = solveShowHistOfStrong [(t, [], p)]

solveAndShowR :: Table -> Protocol -> IO()
solveAndShowR t p = solveShowHistOfR [(t, [], p)]

solveAndShowStrongR :: Table -> Protocol -> IO()
solveAndShowStrongR t p = solveShowHistOfStrongR [(t, [], p)]

-- print hisory
solveShowHistOf :: [Node] -> IO()
solveShowHistOf = sequence_ . fmap showHistOf . solveNs

solveShowHistOfStrong :: [Node] -> IO()
solveShowHistOfStrong = sequence_ . fmap showHistOf . solveNsStrong

solveShowHistOfR :: [Node] -> IO()
solveShowHistOfR = sequence_ . fmap showHistOf . solveNsR

solveShowHistOfStrongR :: [Node] -> IO()
solveShowHistOfStrongR = sequence_ . fmap showHistOf . solveNsStrongR
```

Code to merely check if a graph is solvable in one of the four ways.

```
-- check if a graph is solvable
solvable :: Table -> Protocol -> Bool
solvable t p = solveNs [(t, [], p)] /= []

solvableStrong :: Table -> Protocol -> Bool
solvableStrong t p = solveNsStrong [(t, [], p)] /= []

solvableR :: Table -> Protocol -> Bool
solvableR t p = solveNsR [(t, [], p)] /= []

solvableStrongR :: Table -> Protocol -> Bool
solvableStrongR t p = solveNsStrongR [(t, [], p)] /= []


strip :: Table -> [(Agent, ([Agent], ([Agent], [Agent], [Agent])))]
strip [] = []
strip ((a, (aN, (aX, aY, aZ), _aType, _aProp)):rest) =
  (a, (aN, (aX, aY, aZ))): strip rest
```

Chapter 5

Protocols for Unreliable Gossip: Results

```
module Results where

import AgentTypes
import UnreliableGossip
import Data.List
import Test.QuickCheck
```

In this chapter, we conduct an exploration into the field of dynamic communication when we discard the assumption that *everyone is reliable*. In particular, we will focus on the success of the Learn New Secrets (LNS) protocol on DCU graphs. We show that for real-life application, the use of this assumption is very debatable because already by adding a small amount of error, i.e. the alternating bluffer, the results that were found using this assumption do not longer hold.

We will discuss the results that were generated with the help of the implementation that we have introduced throughout this thesis. The role of the implementation in Haskell now becomes apparent: It is used to create an intuition about dynamic communication with unreliable agents. This is because the implementation allows us to easily and quickly investigate properties and test hypotheses concerning the framework.

Of course, there is a limitation to this approach. Namely that, because of limited computational power, it becomes very time consuming and computationally difficult to test large networks. However, by the *Small Scope Hypothesis* of [17], we will see how the implementation can still provide ideas for theorems and propositions. This hypothesis states that if a statement is invalid, it is very likely to have a small counter-example. As a result, we can create an intuition about DCU graphs by considering all the small cases.

We will then prove our main results. We show that we cannot extend the results of the reliable case by Van Ditmarsch et al. [29] to the case of unreliability. Already with a small amount of error or noise, the results that we have seen for

DC graphs break down. Specifically, we show that on DCU graphs that are sun graphs with terminal unreliable agents, LNS may fail to identify the unreliable agents. That means, there are LNS-permitted call sequences in which some of unreliable agents remain undetected.

As a next step, we will restrict LNS to the protocol LNSR that is just like LNS but does not allow the unreliable agents to initiate calls. That is, the unreliable agents can only receive calls in the network. We then prove that unreliable agents that are not allowed to initiate any form of communication are harder to identify than unreliable agents that do so. This is counter-intuitive: in terms of identification, unreliability that is not actively spread is worse than unreliability that is. This questions what kind of approach or security measure is desirable in the battle against false information. In particular, this raises the question how this result weighs against the reasons for preventing false information to spread. We will elaborate on this further in Section 6.1.

The structure of this chapter is as follows: first, we will show how we can test certain network properties with our implementation. The next step is to introduce reliable counter- and subgraph that allow us to connect completeness on DCU graphs with completeness on DC graphs. Then we will explain how the implementation can be used to generate empirical data. Lastly, we will prove and discuss our results and provide some further questions.

5.1 Testing Network Properties

In Section 3.4, we have already seen some properties of dynamic communication networks with unreliable agents. Here we elaborate on these properties and formalize some new ideas into a working Haskell code. We can then use this code to let the computer generate random networks with certain properties. As such, these properties act as assumptions on the network in order to test for certain hypotheses.

Recall the definition of a terminal agent: an agent a is terminal if and only if $N_a = \{a\}$. So this means that a is terminal if she does not have the phone number of any other agent. We are interested in the consequences of this property of being terminal for dynamic communication networks. In particular, we are interested in the case that all the unreliable agents in a network are initially terminal. This means that initially no unreliable agent can make a call. Of course, in our dynamic setting this property is dynamic and initial terminal agents may not remain terminal after executing some call sequence σ.

Still, we are wondering whether the fact that initially the unreliable agents are terminal has an effect on how well they can be identified. Intuitively it seems that a terminal agent has less power in the network: instead of actively initiating calls, a terminal agent can only passively receive them. However, as we will see in Section 5.4, there are situations in which the property of being terminal can strengthen the position of the unreliable agents in remaining uncovered.

We have the following Definition 5.1.1.

Definition 5.1.1. [Unreliable Terminal] *In a network $G = (A, R, f, N, S)$, the* unreliable agents are terminal *iff for all $a \in A \setminus R$, a is terminal, i.e. $N_a = \{a\}$.*

An operation on graphs that is related to the property of being terminal, is the operation of skinning a graph. Simply put, skinning a graph means that all the terminal nodes of a graph are deleted. Formally, we have the following Definition 5.1.2.

Definition 5.1.2. [Skinning a Graph] *Let $G = (A, R, f, N, S)$ be a DCU graph. Then $s(G)$ is the result of* skinning G, *i.e. removing all terminal nodes from G. That is, $s(G) = (B, R', f', N', S')$ with $B = A \setminus T_G$, $R' = R \setminus T_G$, $f' = f|_{T_G}$, $N' = N \cup B^2$ and $S' = S \cup B^2$, where T_G is the set of terminal nodes of G. I.e. $T_G = \{x \in A \mid \neg\exists y \in A \text{ s.t. } xNy\}$.*

Skinning a graph is not a closure operation. That means that performing the operation twice does not yield the same result as performing it only once. That means that there are graphs such that $s(s(G)) \neq s(G)$. Consider the following Example 5.1.3.

Example 5.1.3. *Let $G = (A, R, f, N, S)$ be an initial DCU graph with $A = R = \{a, b, c\}$ and $N = \{(a, b), (b, c)\}^R$. Then $s(G)$ is G without agent c, whereas $s(s(G))$ is G without agents b and c. See below.*

$$\begin{tikzcd}
a \arrow[r] \arrow[loop above] & b \arrow[r] \arrow[loop above] & c \arrow[loop above] \arrow[r, phantom, "\xrightarrow{s(G)}"] & a \arrow[r] \arrow[loop above] & b \arrow[loop above] \arrow[r, phantom, "\xrightarrow{s(G)}"] & a \arrow[loop above]
\end{tikzcd}$$

Recall that an agent a is *isolated* if in addition no other agent has her phone number, i.e. whenever $N_a = \{a\}$ and for each $b \in A$ such that $b \neq a$ we have that $\{a\} \nsubseteq N_b$. For our purposes, isolated agents are not interesting because they will never interfere with the network.

Haskell Code

Code for checking if unreliable agents in the network are terminal.

```
unreliableTerminal :: Table -> Bool
unreliableTerminal [] = True
unreliableTerminal ((a,(aN,(_aX,_aY,_aZ),aType,_aProp)):rest) =
  not (aType /= Reliable && aN /= [a]) && unreliableTerminal rest
```

Finding isolated agents in a network and deleting them.

```
-- find isolated agents in a network
isolatedAgents :: Table -> [Agent]
isolatedAgents table = loop table table [] where
  loop :: Table -> Table -> [Agent] -> [Agent]
  loop _ [] listOfAgents = listOfAgents
  loop t ((a,(aN,(aX,aY,aZ),aType,aProp)):rest) listOfAgents =
    if aN == [a] && nobodyLikesMe a othersTable
      then loop t rest (a:listOfAgents)
      else loop t rest listOfAgents where
        othersTable = t \\ [(a,(aN,(aX,aY,aZ),aType,aProp))]
    nobodyLikesMe :: Agent -> Table -> Bool
    nobodyLikesMe _ [] = True
    nobodyLikesMe c ((_,(bN,(_,_,_),_,_)):trest) =
      c `notElem` bN && nobodyLikesMe c trest

-- delete isolated agents from network
deleteIsolated :: Table -> Table
deleteIsolated t = dI t t where
  dI _ [] = []
  dI table ((a,(aN,(aX,aY,aZ),aType,aProp)):rest) =
    if a `elem` isolatedAgents table
      then dI table rest
      else (a,(aN,(aX,aY,aZ),aType,aProp)) : dI table rest
```

5.2 Reliable Counter- and Sub Graphs

Next to formalizing network properties, we are interested in comparing the case of dynamic communication with possible errors to the "reliable" case: the DC graphs. This is because there are some known results about the success of protocols for DC graphs [29]. Recall for instance that LNS on DC graphs is characterized by the set of initial DC graphs G that are sun graphs. The question is then raised whether such results can be extended when unreliability may occur.

In order to compare completeness of dynamic networks with unreliable agents (DCU graphs) to completeness in the reliable case (DC graphs), we define *reliable counter-graphs* and *reliable subgraphs*. The idea behind the first is to treat all the agents in the DCU graph as if they were reliable. This means that the structure of the network remains roughly the same but the agent types are changed. The reliable subgraph of G is the DCU graph G restricted to the reliable agents. Clearly, in this case the structure does change because all the unreliable agents a are deleted from the network and with it the connections aNb and bNa for some other agent b.

Definition 5.2.1. [Reliable Counter-graph G^*] *Let $G = (A, R, f, N, S)$ be a DCU Graph. Then we define its* reliable counter-graph *by $G^* = (A^*, N^*, S^*)$ with $A^* = A$, $N^* = N$ and $S^*_a = X_a \cup Y_a$.*

Definition 5.2.2. [Reliable Subgraph $G|_R$] *Let $G = (A, R, f, N, S)$ be a DCU Graph. Then we define its* reliable subgraph *by $G|_R = (A|_R, N|_R, S|_R)$ with $A|_R = R$, $N|_R = \{(a, b) \in N \mid a, b \in R\}$ and $(S|_R)_a = (X_a \cup Y_a)|_R = \{b \in X_a \cup Y_a \mid b \in R\}$.*

A sanity check tells us that indeed both G^* and $G|_R$ are DC graphs. Note that agents b that are considered unreliable (but may in fact be reliable) by an agent a in G (i.e. $b \in Z_a$) will not be reachable from a by the secret relation S^* in G^* nor by $S|_R$ in $G|_R$. This makes sense when b is indeed unreliable for the case of $G|_R$. However, when agent b is in fact reliable, this might be somewhat strange. In this thesis, as we only consider unreliability in a simple form of the alternating bluffer, we do not have to care about this as it can never happen that

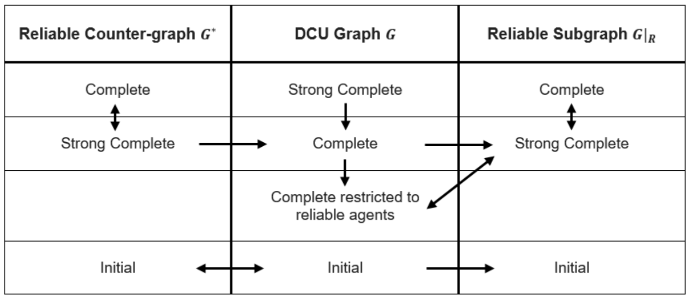


Table 5.1: Table of the relations between completeness

an agent is wrongly accused of being unreliable. This is because the only way for an agent to consider someone unreliable is when contrary information about the secret this agent is available. Yet, this can only happen when the agent herself spreads contrary information about her secret. Still, this may require a different description of G^* and $G|_R$ in the case that higher orders of lying are allowed ("lying about someone else's secret").

It is clear because G^* and $G|_R$ are DC graphs that the definitions of completeness and strong completeness coincide.

We have the following relations between G, its reliable counter-graph G^* and its reliable subgraph $G|_R$.

Proposition 5.2.3. *1) If G is an initial DCU graph, then G^* and $G|_R$ are initial DC graphs. Additionally, 2) if G^* is an initial DC graph, then G is an initial DCU graph. 3) If the reliable counter-graph G^* of DCU graph G is complete, then so is G. 4) A DCU graph G is complete restricted to reliable agents if and only if its reliable subgraph $G|_R$ is complete. Also 5) if a DCU graph G is complete, then so is $G|_R$.*

Proof. 1) and 2) follow from the fact Definitions 5.2.1 and 5.2.2 of reliable counter- and reliable subgraph: if G is an initial DCU graph, then for each $a \in A$ we have that $S^*_a = X_a \cup Y_a = \{a\}$ and for each $a \in R$ it is the case that $(S|_R)_a = (X_a \cup Y_a)|_R = \{a\}|_R = \{a\}$. Clearly the first of the two also holds in the other direction. 3) is a consequence of the fact that if G^* is complete, we have that for all $a \in A^*$ we have that $S^*_a = X_a \cup Y_a = A^* = A \supseteq R$. Hence G is complete. 4) and 5) follow from the fact that $G|_R$ is G without unreliable agents. If all the reliable agents r in G know the secrets of the reliable agents (i.e. $R \subseteq X_a \cup Y_a$), then clearly $G|_R$ is complete and the other way around. And if all agents know the secrets of the reliable agents, $G|_R$ is complete for similar reasons. □

For an overview of the results of Proposition 5.2.3, see Table 5.1.

Haskell Code

We write a code for changing a DCU graph into its reliable counter- or subgraph. Note that these will have the same type as the DCU graphs: Table.

```
-- turn a graph into a reliable counter-graph
reliableCounterGraph :: Table -> Table
reliableCounterGraph [] = []
reliableCounterGraph ((a, (aN, (aX, aY, _aZ), _aType, _aProp)):rest) =
  (a, (aN, (aX, aY, []), Reliable, None)) : reliableCounterGraph rest

-- turn a graph into a reliable subgraph
reliableSubGraph :: Table -> Table
reliableSubGraph table = rSG table where
  rSG [] = []
  rSG ((a, (aN, (aX, aY, _aZ), aType, aProp)):rest) =
    if aType == Reliable && aProp == None
      then (a, (new aN, (new aX, new aY, []), aType, aProp)) : rSG rest
      else rSG rest
      where
        new :: [Agent] -> [Agent]
        new = filter (\x -> x `elem` reliable table)
```

5.3 Generating Random Graphs

With the relevant properties and the relation to DC graphs defined, the use of literate programming now becomes relevant. Throughout this thesis, the implementation of our framework in Haskell is described. With this implementation, it is relatively easy to create an intuitive understanding about unreliability in dynamic communication networks. This is because it allows us to generate empirical data with the computational power of a computer. This role of the implementation is of great importance to the results that we will prove in Section 5.4 because they are based on ideas that resulted from testing random networks with a certain property against a hypothesis.

To generate random networks, we need code in Haskell that includes random generating and we need a package that allows us to do testing. The code can be found below. For the testing we use the "Test.QuickCheck" function.

During our testing phase, we have worked with random dynamic communication graphs with at most five agents. This limitation is because of practical reasons, as networks with more agents become extremely complicated for the computer to deal with. Yet, we can justify this with the *Small Scope Hypothesis* stating that if an assertion is invalid, it probably has a small counter-example [17]. In the field of software engineering this is used in the way that if no mistakes are encountered in small examples of the system, then the confidence increases that the system is also reliable for large examples. Thus, if we test all possible networks with at most five agents and certain properties against a hypothesis, we will either find a counter-example or the hypothesis most likely holds for all networks with these properties. With this, we can create an intuition about DCU networks.

Haskell Code

We generate an arbitrary set of agents.

```haskell
newtype ArbAgs = ArbAgs [Agent] deriving (Show,Eq,Ord)

instance Arbitrary ArbAgs where
  arbitrary = do
    -- change amount of agents in graphs
    n <- choose (3,5)
    return (ArbAgs [1..(n::Agent)])
```

```haskell
getrandomLength :: IO Int
getrandomLength = do
  (ArbAgs as) <- generate arbitrary
  return (length as)

powerset :: [a] -> [[a]]
powerset []     = [[]]
powerset (x:xs) = map (x:) pxs ++ pxs where pxs = powerset xs
```

Generate an arbitrary DCU graph. Choices: generate an initial graph, generate a graph without isolated agents (i.e. a graph that is weakly connected).

```haskell
-- generate arbitrary table
newtype ArbTable = ArbT Table deriving (Show,Eq,Ord)

instance Arbitrary ArbTable where
  arbitrary = do
    ArbAgs ags <- arbitrary
    t <- mapM (\i -> do
      iType <- elements [Reliable, AlternatingBluffer]
      n' <- sublistOf (ags \\ [i])
      let n = sort $ i : n'
      s' <- sublistOf (n \\ [i])
      let s = sort $ i : s'
      -- generate initial graphs, uncomment next line:
      return (i,(n,([i],[],[]),iType,None))) ags
      -- generate random graphs, uncomment next line:
      --return (i,(n, (s,[],[]), iType, None))) ags
    -- to have also isolated agents, uncomment next line:
    --return $ ArbT t
    -- only graphs without isolated agents (note: [] is now valid):
    return $ ArbT (deleteIsolated t)

unpack :: ArbTable -> Table
unpack (ArbT t) = t
```

Generate an arbitrary DCU graph where all the unreliable agents are terminal. We have the same choices as before.

```haskell
-- generate arbitrary table in which unreliable agents are terminal
newtype ArbTableUT = ArbTUT Table deriving (Show,Eq,Ord)

instance Arbitrary ArbTableUT where
  arbitrary = do
    ArbAgs ags <- arbitrary
    t <- mapM (\i -> do
      iType <- elements [Reliable, AlternatingBluffer]
      n' <- sublistOf (ags \\ [i])
      let n | iType == Reliable = sort $ i : n'
            | otherwise         = [i]
      s' <- sublistOf (n \\ [i])
      let s | iType == Reliable = sort $ i : s'
            | otherwise         = [i]
      -- generate initial graphs, uncomment next line:
      return (i,(n,([i],[],[]),iType,None))) ags
      -- generate random graphs, uncomment next line:
      --return (i,(n,(s,[],[]),iType,None))) ags
    -- to have also isolated agents, uncomment next line:
    -- return $ arBTUT t
    -- only graphs without isolated agents (note: [] is now valid):
    return $ ArbTUT (deleteIsolated t)
```

5.4 LNS with Unreliability

In this section, we introduce the results of our framework by a combination of empirical data and proofs. Throughout this section, the role of the implementation is to create an intuitive understanding of dynamic communication with unreliability. With this, it is easy to investigate properties for DCU networks and test whether certain hypotheses hold. This provides a first step toward proving general results. The results in this section, however, can be read and well-understood without the consultation of the implementation.

We show that one of the results on DC graphs, namely that LNS is strongly successful on initial DC graphs that are sun graphs (see Theorem 2.3.5), breaks down in the case that some agents are unreliable. That is, not all LNS-permitted call sequences result in a strong compete network. Specifically, we prove that already for sun graphs with only terminal alternating bluffers there are LNS-permitted call sequences that fail to identify the unreliable agents. As a result, it follows that LNS does not continue to characterize sun graphs and vice versa when a small amount of unreliability is introduced.

Next, we classify some graphs on which LNS is not strongly successful in the strong complete sense, but is successful in the complete sense. In particular, we show that on initial DCU graphs G such that N is weakly connected and there is a LNS-permitted call sequence that is successful on the reliable subgraph $G|_R$, there is an LNS-permitted sequence successful on G in the complete sense. That is, this call sequences causes all the agents to learn the secrets of the reliable agents. This opens the door for future research to the different types of completeness that we have in the case of unreliable agents. Specifically, this asks the questions what additional properties the protocol must have to achieve strong completeness over completeness.

We then further restrict the power of the unreliable agents by considering the protocol LNSR. This is the protocol LNS with the additional restriction that only reliable agents are allowed to initiate calls. We prove that even though intuitively this seems a restriction on the communicative power of the unreliable agents, LNSR in fact makes it harder for the reliable agents to identify the unreliable agents. This has some contradictory implications: it may be necessary for unreliable agents to initiate calls in order for them to be successfully identified. In other words, unreliability can be easier detected when it is spread more. In Section 6.1, we will further dive into the questions that are raised by this result.

We will explain our theorems and propositions with both empirical results and concrete examples, and highlight some questions that are raised. These remaining questions provide a starting point for future research.

5.4.1 Failure of LNS for DCU graphs

Just like the case of dynamic communication without unreliable agents, we are now also interested in classifying DCU networks by the protocols that are successful on them. But now, we are interested in their success in the sense of completeness and, even more so, in the sense of strong completeness. To answer this question, we first explore whether we can extend the known results of DC graphs by [29]. In particular, we explore whether we can extend the result about the class of DC networks on which any LNS-permitted call sequence results in completeness to a similar class of DCU graphs when considering strong completeness.

In this section, we show that the answer is no. Specifically, we prove that for any network that is a sun graph with only terminal alternating bluffers there are LNS-permitted call sequences that do not result in a strong complete network. This means that already with a small amount of unreliability, the known classification of LNS (see Theorem 2.3.5) fails to hold.

To prove this result, we will first prove the more general result that for any DCU graph $G = (A, R, f, N, S)$ with only terminal alternating bluffers and such that the reliable subgraph $G|_R$ of G is complete and no other secrets are known throughout the network, LNS is unsuccessful in the strong complete sense. This is formalized by Hypothesis 5.4.1.

Hypothesis 5.4.1. *For any DCU graph $G = (A, R, f, N, S)$ with only reliable agents and alternating bluffers, LNS is unsuccessful in the strong complete sense on G if all unreliable agents are terminal and the reliable subgraph $G|_R$ of G is complete.*

To get an intuition why Hypothesis 5.4.1 must hold, consider Example 5.4.2.

Example 5.4.2. *Let $G = (A, R, f, N, S)$ be as drawn below where $A = \{a, b, c\}$, $R = \{a, b\}$, $N = \{(a, b), (a, c), (b, a), (b, c)\}^R$, $X_a \cup Y_a = X_b \cup Y_b = \{a, b\}$ and $X_c \cup Y_c = \{c\}$. Let c be of the type alternating bluffer. So clearly we have that $s(G) = G|_R$ is complete. Now the only LNS-permitted call sequences on G are $\sigma_1 = ac; bc$ and $\sigma_2 = bc; ac$. In either way, this results in $X_a^{\sigma_1} \cup Y_a^{\sigma_1} = X_b^{\sigma_1} \cup Y_b^{\sigma_1} = \{a, b, c\}$. And in addition, $(X_a^{\sigma_1} \cap X_b^{\sigma_1}) \cup (Y_a^{\sigma_1} \cap Y_b^{\sigma_1}) = \{a, b\}$. In words, agent a and b both consider agent c reliable, but they do not agree about her secret.*

$$\begin{tikzcd}
a \arrow[loop above] \arrow[rr, leftrightarrow] \arrow[dr] & & b \arrow[loop above] \arrow[dl] \\
 & c \arrow[loop below] &
\end{tikzcd}$$

In Example 5.4.2 LNS fails to strongly complete the network because the reliable agents are not allowed to communicate with each other after they have been in contact with the unreliable agent. More generally, LNS restricts the reliable agents to call any of the other reliable agents. That means that after communicating with the unreliable agent, the reliable agents cannot verify the new information with each other. Hence, the unreliable agent will remain undetected.

More specifically, any LNS-permitted call sequence fails to identify the unreliable agent. This does not mean that LNS cannot be used to complete the network. Indeed, in Example 5.4.2, executing either σ_1 or σ_2 results in a complete network. This is because after executing one of these LNS-permitted call sequences, say σ_1, $R \subseteq X_c^{\sigma_1} \cup Y_c^{\sigma_1}$. Thus everyone knows the secrets of the reliable agents.

The rest of this section is structured as follows. First we will give a formal proof of Hypothesis 5.4.1 We do this by introducing the lemmas needed, proving the case of one unreliable agent and then by showing that Hypothesis 5.4.1 is actually a special case of a more general theorem. Then, we will discuss the consequences and corollaries of this result. In particular, we will show why we cannot extend the known result about the classification of LNS by sun graphs and vice versa. Then, we will provide a short discussion of the results.

Proof

Before we formalize and prove the idea presented in Hypothesis 5.4.1, we need Lemma 5.4.3. Lemma 5.4.3 states that $s(G) \subseteq G|_R$ for some DCU graph G if and only if all the unreliable agents are terminal. Clearly this makes sense because if removing all terminal agents results in a network with only reliable agents, then the unreliable agents must be terminal. For the reverse direction, if all the unreliable agents are terminal, then by removing all the terminal agents we end up with a network consisting of only reliable agents. Note that it might still be that there are reliable agents that are terminal and therefore it is not the case that $s(G) = G|_R$ in the right to left direction.

Lemma 5.4.3. *Let G be a DCU graph. We have that $s(G) \subseteq G|_R$ iff all unreliable agents are terminal agents.*

Proof. For the left to right direction, let G be a DCU graph and suppose that $s(G) \subseteq G|_R$ and suppose, toward a contradiction, that there is an unreliable agent $a \in A \setminus R$ such that a is not terminal. Thus there is another agent $b \in A$

such that aNb. But then $a \in s(G)$ and hence $s(G) \nsubseteq G|_R$. For the reverse direction, suppose that all the unreliable agents are terminal. Then clearly, by performing the skinning operations, all the unreliable agents will be deleted. Hence $s(G) \subseteq G|_R$. $\square$

A sanity check tells us that trivially Lemma 5.4.3 also holds in the case that there are no unreliable agents. That is, if $G = G|_R$. This follows from the fact that $s(G) \subseteq G$ for any graph G.

Notice that now for any DCU graph G, we have that $s(G) = G|_R$ implies that $X_a \cup Y_a = \{a\}$ for each unreliable agent a. That is, the unreliable agents do not have any prior knowledge of the secrets of other agents.

Lemma 5.4.4. *Let $G = (A, R, f, N, S)$ be a DCU graph. Then if $s(G) \subseteq G|_R$ we have that $X_a \cup Y_a = \{a\}$ for each $a \in A \setminus R$.*

Proof. This follows directly from Lemma 5.4.3 and the fact that $X^\sigma \cup Y^\sigma \subseteq N^\sigma$ for each call sequence σ. Because for any terminal agent a, $N_a = \{a\} \supseteq X_a \cup Y_a \supseteq \{a\}$. $\square$

We now justify Hypothesis 5.4.1 with exhaustive search over small domains. We have tested Hypothesis 5.4.1 with maximal five agents, no isolated agents and only terminal alternating bluffers. As we do not find any counter-example in this situation, the Small Scope Hypothesis suggests that there are also no larger counter-examples. It then follows that our Hypothesis 5.4.1 will most likely hold for all DCU graphs with these properties.

```
*UnreliableGossip AgentTypes Examples Gossip Results Sabotage
     UnreliableGossip Test.QuickCheck>
quickCheck (\(ArbTUT t) -> (complete (reliableSubGraph t) && t /= [] ==>
     not (solvableStrong t LNS)))
+++ OK, passed 100 tests.
```

In order to get an intuition for proving Hypothesis 5.4.1, we first prove the case of one alternating bluffer. I.e. we will show that if G is as in Hypothesis 5.4.1 and $|A \setminus R| = 1$, the unreliable agent will remain unidentified. This is because the only call sequence that is LNS-permitted on such G is of the form $a_1u; a_2u; \ldots; a_nu$ where $R = \{a_1, \ldots, a_n\}$, $A = R \cup \{u\}$ and $u \in N_{a_i}$ for each $i \in \{1, \ldots, n\}$.

Proposition 5.4.5. *LNS is unsuccessful (in strong complete sense) on DCU graphs $G = (A, R, f, N, S)$ such that N is weakly connected, there is one unreliable agent u (i.e. $|A \setminus R| = |\{u\}| = 1$) that is of the type alternating bluffer and $s(G) \subseteq G|_R$ is complete (that is, $(X_a \cup Y_a)|_R = R$ for each $a \in R$).*

Proof. Let $G = (A, R, f, N, S)$ be as above. We know by Lemma 5.4.3 that the unreliable agent u is terminal and that the reliable subgraph $G|_R$ is complete. Thus, for each $a \in R$ we have that $X_a \cup Y_a = R$. And since $N_u = \{u\} \supseteq X_u \cup Y_u \supseteq \{u\}$, it follows that $X_u \cup Y_u = \{u\}$. Now the only LNS-permitted call on G is au where $a \in R$ and $u \in N_a$. After this call, we have that $N_a^{au} = N_u^{au} = N_a$ and $X_a^{au} \cup Y_a^{au} = (X_a \cup Y_a) \cup \{u\} = R \cup \{u\} = A = X_u^{au} \cup Y_u^{au}$. The rest of the relations stay the same. So the next LNS-permitted call on G^{au} must be again of the form bu where $b \in R$ and $u \in N_b$. We can repeat this until every reliable agent with a connection to the unreliable agent has made a call. Then, no call is LNS-permitted and hence LNS fails to identify u as the unreliable agent. $\square$

It now seems that if an unreliable agent is "added" to a complete DC graph (i.e. $A = R$), that is $A^+ = A \cup \{u\}$, $S^+ = S \cup I_{\{u\}}$ and $N^+ = N \cup N'$ where $N' \subseteq \{(a, u), (u, a) \mid a \in A\}$, she will remain uncovered by the protocol LNS. The consequence of this is that unreliable agents can freely penetrate a complete network without being identified. In fact, it is not even necessary that the added unreliable agent is terminal. We merely need that there is no prior knowledge about her secret in the network. Consider the following example.

Example 5.4.6. *Let $G = (A, R, f, N, S)$ be such that $A = \{a, b, c\}$, $R = \{a, b\}$, $G|_R$ is complete, c of the type alternating bluffer and $N_c = \{a\}$ and $N_a = N_b = \{a, b\}$. Then there is no LNS-permitted all sequence that identifies c as the unreliable agent.*

Intuitively, this holds because agents a and b do not have any prior knowledge about the secret of agent c. As a result, they cannot encounter contrary information about the secret because LNS only allows them to be in contact with the unreliable agent once. We formalize this idea for multiple unreliable agents as follows. We will then show that Hypothesis 5.4.1 is a specific instance of Theorem 5.4.7.

Theorem 5.4.7. *LNS is unsuccessful (in strong complete sense) on DCU graphs $G = (A, R, f, N, S)$ such that N is weakly connected, all unreliable agents $a \in A \setminus R$ are of the type alternating bluffer, $G|_R$ is complete and the reliable agents do not have any prior knowledge about the secrets of the unreliable agents, i.e. for each $r \in R$ we have that $X_r \cup Y_r = R$.*

We prove this by observing that the only LNS-permitted calls ab on G are those calls where both agents are unreliable or at least one of them is. I.e. the call ab is allowed if a is unreliable or b is unreliable. So in particular this means that the reliable agents can only be involved in calls with unreliable agents.

Proof. Let $G = (A, R, f, N, S)$ be as described above and suppose, toward a contradiction, that LNS is successful (in the strong complete sense) on G. So that means that there is an LNS-permitted call sequence σ such that for each $a \in A$, both $X_a^\sigma \cup Y_a^\sigma = R$ and $Z_a^\sigma = A \setminus R$. Furthermore, by assumption and because knowledge about the secrets of reliable agents cannot be lost, $R \subseteq X_r^\tau \cup Y_r^\tau$ for each $\tau \sqsubseteq \sigma$ and each $r \in R$. This also means that at any stage, the only calls that are permitted by LNS are calls ab such that at least one of the agents a and b is unreliable. Let $\{a_1, \dots, a_n\} = A \setminus R$ and choose some $r \in R$. So it must be that $Z_r^\sigma = \{a_1, \dots, a_n\}$. By the definition of a call, this means that $\forall i \in \{1, \dots, n\}\ \exists \tau \sqsubseteq \sigma$ such that $\sigma = \tau; rb; \tau'$ or $\sigma = \tau; br; \tau'$ for some $b \in N_r^\tau \setminus (X_r^\tau \cup Y_r^\tau) \subseteq \{a_1, \dots, a_n\}$, and $a_i \in X_r^\tau \cap Y_b^\tau$ or $a_i \in Y_r^\tau \cap X_b^\tau$. In particular, LNS requires that $b \neq a_i$. Now let a^* be the last unreliable agent that r identifies. That is, there is a $\tau^* \sqsubseteq \sigma$ such that $\sigma = \tau^*; rb; \tau'$ (or $\sigma = \tau^*; br; \tau'$) for some agent $b \in N_r^{\tau^*} \setminus (X_r^{\tau^*} \cup Y_r^{\tau^*})$, $a^* \in X_r^{\tau^*} \cap Y_b^{\tau^*}$ or $a^* \in Y_r^{\tau^*} \cap X_b^{\tau^*}$ and $Z_r^{\tau^*} \subseteq \{a_1, \ldots, a_n\} \setminus \{a^*\}$. As a^* is the last agent to be identified, $Z^{\tau^*;rb} = \{a_1, \ldots, a_n\}$ and $\tau'_r = \epsilon$. That is, agent r does not perform anymore calls after the call rb. Let's now see who should be the agent b that is communicating to agent r. We already know that $b \neq a^*$. Thus it must be that $b = a_j$ for some $a_j \in \{a_1, \ldots, a_n\}$. But then by LNS, agent r cannot have prior knowledge about the secret of agent $b = a_j$, i.e. $a_j \notin Z_r^{\tau^*}$. As a result, after the call rb it is the case that $a_j \in X_r^{\tau^*;rb} \cup Y_r^{\tau^*;rb}$. And since agent r will not be involved in any more calls, agent a_j remains unidentified from the point of view of r. This is a contradiction. Hence, LNS is not successful on G. □

Note that the proof actually is a stronger result than the one stated in Theorem 5.4.7: namely that no LNS-permitted call sequence can result in a network that is strong complete restricted to the reliable agents. Clearly then, LNS is also not successful with regards to the whole network. Note that the result also holds for the general bluffer, regardless of the probability p with which she lies about her own secret. In fact, in case of the bluffer it might be even harder to identify all the unreliable agents because there is a greater chance that less contrary information about the secrets of the unreliable agents is available.

We are now ready to prove Hypothesis 5.4.1.

Theorem 5.4.8. [Hypothesis 5.4.1] *LNS is unsuccessful (in strong complete sense) on DCU graphs $G = (A, R, f, N, S)$ such that N is weakly connected, all unreliable agents $a \in A \setminus R$ are of the type alternating bluffer, $s(G) \subseteq G|_R$ is complete (that is, $(X_a \cup Y_a)|_R = R$ for each $a \in A|_R = R$) and no other secrets are known throughout the network (i.e. for each $a \in R$, $X_a \cup Y_a = (X_a \cup Y_a)|_R = R$).*

The proof is now straightforward because the graphs G for which Theorem 5.4.8 applies are a specific case of the graphs G described in Theorem 5.4.7.

Proof. Let $G = (A, R, f, N, S)$ be as above. So for each $a \in R$ we have that $(X_a \cup Y_a) = (X_a \cup Y_a)|_R = R$ and by Lemma 5.4.4 we have that for all $a \in A \setminus R$, $X_a \cup Y_a = \{a\}$. We can now apply Theorem 5.4.7 to find that indeed LNS is unsuccessful on G in the strong complete sense. □

Consequences and corollaries

In this section, we will explain how Theorem 5.4.8 leads to the conclusion that we cannot continue to classify LNS by sun graphs for the case of strong completeness when unreliable agents are introduced. More generally, we prove that as a consequence of Theorem 5.4.8 any LNS-permitted call sequence σ fails to identify the unreliable agents on an initial DCU network G where all the unreliable agents are terminal if the network reaches a state (through executing some $\tau \sqsubseteq \sigma$) in which its reliable subgraph $s(G^\tau) = G^\tau|_R$ is complete. In other words, on such networks σ does not result in strong completeness if the reliable agents learn the secrets of the reliable agents before contacting the unreliable agents.

This conclusion was already illustrated in Example 4.2.7 where we considered a DCU network $G = (\{a, b\}, \{a\}, f, \{(a, b)\}^R, I_A)$ of two agents with one terminal unreliable agent. Trivially then $G|_R$ was complete and the only LNS-permitted call was ab—a call that does not lead to agent a identifying agent b as unreliable. Formally, we have the following result.

Corollary 5.4.9. *Let G be a DCU graph, all unreliable agents of the type alternating bluffer and σ a LNS-permitted call sequence. σ is unsuccessful on G in the strong complete sense if there is a τ such that $\tau \sqsubseteq \sigma$ and $s(G^\tau) \subseteq (G^\tau)|_R$ is complete.*

Proof. This follows directly from Theorem 5.4.8 by observing that G^τ fits the requirements described. □

Thus Corollary 5.4.9 describes a class of networks on which LNS is not strongly successful in the strong complete sense. That is, there are LNS-permitted call sequences that do not result in strong completeness—a situation in which all aents know the secrets of the reliable agents and can identify the unreliable agents. Recall now the results by [29] about the classification of DC graphs, Theorem 2.3.5: LNS is strongly successful on an initial DC graph G iff G is a sun graph. We now show that this classification fails when there is at least one terminal alternating bluffer in the network.

Corollary 5.4.10. *LNS is not strongly successful in the strong complete sense on initial DCU graphs $G = (A, R, f, N, S)$ that are sun graphs such that all unreliable agents are of the type alternating bluffer and are terminal in G.*

Proof. Let $G = (A, R, f, N, S)$ be as described. Since we know that G is a sun graph and all the unreliable agents are terminal, $G|_R$ must be a sun graph as well. No by Theorem 2.3.5, any LNS-permitted call sequence τ with only calls between reliable agents will complete the reliable subgraph $G|_R$ of G. That is, $(G|_R)^\tau = G^\tau|_R$ is complete. Additionally, since τ only consists of calls between reliable agents, the unreliable agents remain terminal in G^τ. But now for any extension of τ $\sigma = \tau; \upsilon$, by Corollary 5.4.9 σ is unsuccessful on G in the strong complete sense. □

Notice that this result merely claims that not all LNS-permitted call sequences are successful in the strong complete sense on initial DCU graphs G that are sun graphs and where all unreliable agents are terminal alternating bluffers. In fact, there might still be some other LNS-permitted call sequences that do result in a strong complete network. Consider the following Example 5.4.11.

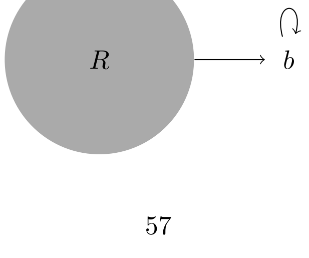
Example 5.4.11. *Let $G = (A, R, f, N, S)$ be an initial DCU graph such that $A = \{a, b, c\}$, $R = \{a, b\}$, $N = \{(a, b), (b, a), (a, c), (b, c)\}^R$ and $S = I_A$. Furthermore, let c be of the type alternating bluffer. See also the figure below. Now the call sequence $\sigma = ac; bc; ab$ results in a strong complete network.*

$$\begin{tikzcd}
a \arrow[loop above] \arrow[rr, leftrightarrow] \arrow[dr] & & b \arrow[loop above] \arrow[dl] \\
& c \arrow[loop below] &
\end{tikzcd}$$

However, the impact of Corollary 5.4.10 is that the known results about the classification of DC graphs by LNS fail to apply for DCU graphs when we consider strong completeness. Because clearly for any DCU graph G described by Corollary 5.4.10, LNS is strongly successful on the reliable countergraph G^* [29]. That is, any LNS-permitted call sequence on such graphs G^* results in a complete network. As a result, Corollary 5.4.10 implies that already with a small bit of unreliability (namely at least one terminal alternating bluffer), we cannot extend the results of the reliable case to the case of unreliability. This is very relevant because many applications of this framework in fact do have to deal with small portions of unreliability in the form of error or noise. Thus we learn that already a small noise or error can completely change the situation. So also the other results on DC networks in [29] would have to be re-examined for their validity on DCU networks.

What in fact happens in the proofs of Theorem 5.4.8 and Theorem 5.4.7 is that that there is at least one unreliable agent b for which the protocol LNS partitions the group of reliable agents into three disjoint sets $R^{f(b)=0}$, $R^{f(b)=1}$ and $R^{b\notin R}$ that consider the secret of agent b to be 0 and 1 and consider agent b unreliable, respectively, such that $R^{b\notin R} \neq R$. This is another way of saying that agent b will not be successfully identified by the whole network. Consider the following Example 5.4.12.

Example 5.4.12. *Let $G = (A, R, f, N, S)$ be a DCU graph with $A = \{a_1, a_2, \ldots, a_n, b\}$ for some $n \in \mathcal{N}$, $A \setminus R = \{b\}$, $s(G) = G|_R$ is complete and b is of the type alternating bluffer with $f(b) = 1$. Furthermore consider that for each $a_i \in R$, $a_i N b$. The remaining structure of N follows from the fact that $G|_R$ is complete and $S \subseteq N$. See also the figure below, where the big gray cluster denotes that both N and S are strongly connected. By LNS now, the only permitted calls are of the form $a_i b$ or $b a_i$. Consider, without loss of generality, the LNS-permitted call sequence $\sigma = a_1 b; a_2 b; \ldots; a_n b$. This means that each agent a_i with i odd will update $X_i^\sigma = X_i \cup \{b\}$ and the agents a_i with i even will update $Y_i^\sigma = Y_i \cup \{b\}$. Then $R = \{a_1, a_2, \ldots, a_n\} = \{a_i \mid i \in \{1, \ldots, n\}$ is even$\} \cup \{a_i \mid i \in \{1, \ldots, n\}$ is odd$\} = R^{f(b)=0} \cup R^{f(b)=1}$.*



Note that we considered σ without loss of generality because LNS requires any call to take place to be between a reliable agent and the unreliable agent b. This has the result that the reliable agents cannot verify the secret of the unreliable agent b and therefore b will remain undetected. In fact, in the case of only one unreliable agent as in Example 5.4.12, the unreliable agent will remain *completely undetected*. That is, $R = R^{f(b)=0} \cup R^{f(b)=1}$.

The question can now be asked whether the situation changes when not all the reliable agents have the phone number of the unreliable agent in Example 5.4.12. toThen, by LNS, the unreliable agent b will call all the reliable agents that do not have the phone number of b. So no agent will remain ignorant about the secret of agent b, yet no agent will identify b as unreliable.

The situation changes slightly when there are multiple unreliable agents in the network. This is because the unreliable agents will now learn the phone number of each other via the reliable agents. Hence by LNS a call between the unreliable agents will take place. But then, the reliable agents can verify their information with the unreliable agents. Consider the following Example 5.4.13.

Example 5.4.13. *Let $G = (A, R, f, N, S)$ be a DCU graph with $A = \{a, b, c, d\}$, $R = \{a, b\}$, $N = \{(a, b), (a, c), (a, d), (b, a), (b, c), (b, d)\}^R$ and $S = \{(a, b), (b, a)\}^R$. See also the figure below. Now consider the following LNS-permitted call sequence $\sigma = ac; bd; ad; bc$. Then $X^\sigma_a \cup Y^\sigma_a = \{a, b, d\}$, $Z^\sigma_a = \{c\}$ and $X^\sigma_b \cup Y^\sigma_b = \{a, b, c\}$, $Z^\sigma_b = \{d\}$.*

$$\begin{tikzcd}
c \arrow[loop above] & d \arrow[loop above] \\
a \arrow[u] \arrow[ur] \arrow[r, leftrightarrow] \arrow[loop below] & b \arrow[u] \arrow[ul] \arrow[loop below]
\end{tikzcd}$$

So even though in Example 5.4.13 neither unreliable agent will remain completely undetected, neither unreliable agent will be completely detected either. We have that $R^{c \notin R} \neq R$ and $R^{d \notin R} \neq R$. This is interesting because it shows that as long as the unreliable agents do not initiate communication, they can remain completely undetected by the network. One of the next steps is exactly this, considering the protocol LNSR that restricts the unreliable agents from initiating calls.

Discussion

In this section, we have showed that the current classification of DC networks by the strong success of LNS fails to apply to DCU networks. That is, we have proved that there are sun graphs with alternating bluffers and LNS-permitted call sequences on these graphs such that executing them does not result in a strong complete network. Thus, already by adding a small amount of unreliability, the original results of [29] about dynamic communication break down.

This exactly emphasizes the relevance of our framework in providing a different approach to modeling information spread. We have namely shown that by going away from the idealized assumption that *everybody is reliable*, the situation changes radically and the current classification of networks and protocols fails.

5.4.2 Success of LNS

We have seen in the previous section that the known classification of networks already fails when a small amount of unreliability is introduced. In particular, we have shown on what kind of DCU graphs LNS is not strongly successful in the strong complete sense. An LNS-permitted call sequence σ does not strongly complete a DCU graph G if a state is reached by executing some $\tau \sqsubseteq \sigma$ in which $s(G^\tau) = (G^\tau)|_R$ is complete. In other words, if a state is reached in which the situation of the unreliable agents has not changed ($N_b^\tau = N_b$ and $S_b^\tau = S_b$ for each $b \in A \setminus R$) but the reliable agents form a complete network.

We are now interested in finding out on what kind of DCU graphs there are LNS-permitted call sequences that result in a complete network. This brings us to Theorem 5.4.14.

Theorem 5.4.14. *LNS is successful (in complete sense) on initial DCU graphs $G = (A, R, f, N, S)$ such that N is weakly connected and LNS is successful (in the complete sense) on $G|_R$.*

Intuitively, Theorem 5.4.14 holds because if some agents know all secrets of the reliable agents, they can communicate these secrets to agents that do not yet know all these secrets. The proof of Theorem 5.4.14 is based on the idea that any LNS-permitted call sequence σ that completes $G|_R$ can be extended to an LNS-permitted call sequence that completes G itself as long as the number relation N is weakly connected. Namely, for each unreliable agent b we point out one agent a with $b \in N_a$ that, after σ has been executed, calls agent b to update her. This call is allowed because σ was a call sequence on $G|_R$ and therefore no calls of the form ab with $a \in R$ and $b \in A \setminus R$ have yet taken place.

Proof

The proof of Theorem 5.4.14 is based on the fact that we can extend any call sequence σ that is LNS-permitted and completes $G|_R$ to a LNS-permitted call sequence σ^+ that completes G. Note that it is crucial to this proof that N is weakly connected.

Proof. Let $G = (A, R, f, N, S)$ be an initial DCU graph where N is weakly connected and $B = \{b_1, \ldots, b_n\} = A \setminus R$ is an enumeration of the unreliable agents. Furthermore, let σ be an LNS-permitted call sequence that completes

$G|_R$. Then we extend σ to an LNS-permitted call sequence σ^+ that completes G in steps:

$$\sigma_0^+ = \sigma$$

$$\sigma_1^+ = \begin{cases} \sigma_0^+; ab_1 & \text{if } \exists a \in A \text{ s.t. } R \subseteq S_a,\, b_1 \in N_a \text{ and } b_1 \notin S_a \\ \sigma_0^+ & \text{otherwise} \end{cases}$$

$$\cdots$$

$$\sigma_n^+ = \begin{cases} \sigma_{n-1}^+; ab_n & \text{if } \exists a \in A \text{ s.t. } R \subseteq S_a,\, b_n \in N_a \text{ and } b_n \notin S_a \\ \sigma_{n-1}^+ & \text{otherwise} \end{cases}$$

After n rounds, we do the same thing for the set $B' \subset B$ of unreliable agents b that are not yet involved in σ_n^+. We continue this until we reach an empty set $B^{'''\cdots}$. σ^+ is now defined as the call sequence we ended up with. Clearly, σ^+ is LNS-permitted because no prior call between reliable and unreliable agents has yet been made in σ. Furthermore, σ^+ call sequence that completes G. It remains to show that this procedure terminates. But in fact, this already follows from the fact that N is weakly connected and that any LNS-permitted call sequence on a finite network is finite.

□

Discussion

To some extent, Theorem 5.4.8 seems to bite with Corollary 5.4.9. This is because by combining these results, we find that if a DCU graph G reaches a state in which $s(G^\sigma) = G^\sigma|_R$ is complete but G^σ is not for some LNS-permitted call sequence σ, then we can extend σ to an LNS-permitted call sequence σ^+ that completes G, however, any extension σ' of σ will fail to strongly complete G. So in particular σ^+ will complete G but fail to strongly complete G. In other words, in such a situation it is possible for everyone to learn the secrets of the reliable agents, but it is not possible to successfully identify all the unreliable agents.

Thus we have found a class of DCU graphs on which LNS can still do the job in terms of completeness, but LNS fails in terms of strong completeness. Note though that these are not initial DCU graphs. This raises the question what extra properties the networks or protocols need to possess to ensure strong completeness given completeness. For now it is clear that the current properties do not suffice.

5.4.3 Restricting LNS to LNSR

Thus, we are interested in adding extra properties to the network or to protocols. In particular, we are interested in ways to restrict the power of the unreliable agents. In Section 4.5, the protocol LNSR was introduced as an adjustment of the LNS protocol. The protocol LNSR is just like LNS with the restriction that unreliable agents are not allowed to initiate calls.

Intuitively, we introduced the protocol LNSR to limit the power of the unreliable agents. Here we show that from the perspective of identification the opposite is true. That is, we show that by executing the protocol LNSR instead of LNS, the chances increase that the unreliable agents remain uncovered. Specifically, we prove that whenever LNSR is successful on a DCU graph G, then so is LNS itself.

Theorem 5.4.15. *If LNSR is (strongly) successful in the (strong) complete sense on a DCU graph G, then so is LNS.*

This implies that LNSR $\subseteq$ LNS, i.e. the class of DCU graphs on which LNSR is (strongly) successful in the (strong) complete sense is a subset of the class of DCU graphs on which LNS is. In particular, we will show that LNSR $\neq$ LNS and hence LNSR $\subset$ LNS. Hence restricting the unreliable agents in their communicative power, i.e. not allowing them to initiate any call, shrinks the class of graphs that can be completed.

Proof

The proof of Theorem 5.4.15 is straightforward given that any LNSR-permitted call sequence is also LNS-permitted.

Proof. [Theorem 5.4.15] We have that any LNSR-permitted call sequence σ is also LNS-permitted. So if LNSR is successful in the (strong) complete sense on a DCU graph G, there is an LNS-permitted call sequence σ such that G^σ is (strong) complete. But then by definition, because σ is also LNS-permitted, LNS is also successful on G in the (strong) complete sense. $\square$

The reverse direction is not true. There are DCU graphs G on which LNS is successful in the (strong) complete sense, but LNSR is not. There are even DCU graphs G in which with only terminal unreliable agents for which this holds. This implies that LNSR $\neq$ LNS. Consider Examples 5.4.16 and 5.4.17.

Example 5.4.16. *Consider the initial DCU network $G = (A, R, f, N, S)$ drawn below, where $A = \{a, b, c, d\}$, $R = \{a, b, d\}$ and $N = \{(a, c), (d, b), (d, c)\}^R$. We have that all the unreliable agents are terminal (as $A \setminus R = \{c\} = N_c$). By trying out all possible call sequences, we find that LNS is successful (in the complete sense) on G, but LNSR is not.*

$$\begin{tikzcd}
a \arrow[loop above] \arrow[d] & b \arrow[loop above] \\
c \arrow[loop below] & d \arrow[l] \arrow[u] \arrow[loop below]
\end{tikzcd}$$

```
*UnreliableGossip AgentTypes Gossip Results UnreliableGossip Test.
    QuickCheck>
solvable [(1,([1,3],  ([1],[],[]),Reliable,          None)),
          (2,([2],    ([2],[],[]),Reliable,          None)),
          (3,([3],    ([3],[],[]),AlternatingBluffer,None)),
          (4,([2,3,4],([4],[],[]),Reliable,          None)))] LNS
True

*UnreliableGossip AgentTypes Gossip Results UnreliableGossip Test.
    QuickCheck>
solvable [(1,([1,3],  ([1],[],[]),Reliable,          None)),
          (2,([2],    ([2],[],[]),Reliable,          None)),
          (3,([3],    ([3],[],[]),AlternatingBluffer,None)),
          (4,([2,3,4],([4],[],[]),Reliable,          None)))] LNSR
False
```

Example 5.4.17. *Consider the initial DCU graph $G = (A, R, f, N, S)$ drawn below, where $A = \{a, b, c, d\}$, $R = \{b, c, d\}$ and $N = \{(b, d), (c, a), (c, b), (d, b), (d, c)\}^R$. We have that all the unreliable agents are terminal (as $A \setminus R = \{a\} = N_a$). By trying out all possible call sequences, we find that LNS is successful in the strong complete sense on G, but LNSR is not. That is, the unreliable agents can be identified when the protocol LNS is executed, but they will remain uncovered when they are limited in their communication.*

$$\begin{tikzcd}
a \arrow[loop above] & b \arrow[loop above] \arrow[d, leftrightarrow] \\
c \arrow[u] \arrow[ur] \arrow[loop below] & d \arrow[l] \arrow[loop below]
\end{tikzcd}$$

```
*UnreliableGossip AgentTypes Gossip Results UnreliableGossip Test.
    QuickCheck>
solvableStrong [(1,([1],    ([1],[],[]),AlternatingBluffer,None)),
                (2,([2,4],  ([2],[],[]),Reliable,          None)),
                (3,([1,2,3],([3],[],[]),Reliable,          None)),
                (4,([2,4],  ([4],[],[]),Reliable,          None))] LNS
True

*UnreliableGossip AgentTypes Gossip Results UnreliableGossip Test.
    QuickCheck>
solvableStrong [(1,([1],    ([1],[],[]),AlternatingBluffer,None)),
                (2,([2,4],  ([2],[],[]),Reliable,          None)),
                (3,([1,2,3],([3],[],[]),Reliable,          None)),
                (4,([2,4],  ([4],[],[]),Reliable,          None))] LNSR
False

*UnreliableGossip AgentTypes Gossip Results UnreliableGossip Test.
    QuickCheck>
solvable [(1,([1],    ([1],[],[]),AlternatingBluffer,None)),
          (2,([2,4],  ([2],[],[]),Reliable,          None)),
          (3,([1,2,3],([3],[],[]),Reliable,          None)),
          (4,([2,4],  ([4],[],[]),Reliable,          None))] LNSR
False
```

Note that in Example 5.4.17, the network can still result in a complete network by executing the protocol LNSR. So indeed LNSR fails in the identification of the unreliable agents.

Consequences and corollaries

In Example 5.4.16, even though the unreliable agents are terminal and are not permitted by LNS to initiate any call in the initial network, they are still required to initiate calls in a later stage of the network in order to obtain strong completeness. In particular, this is the case because initially the network is split into two groups of reliable agents that can only be connected through the unreliable agent. Agent a can only be in touch with agent b and agent d via the alternating bluffer c.

This result is counterintuitive because we introduced LNSR to limit the power of the unreliable agents. But in fact, this can be understood if we think of restricting the communicative power of the unreliable agents as a way of limiting the amount of (contrary) information in the network about the secret of the unreliable agents. If we have an unreliable agent $u \in A \setminus R$ and another agent $a \in R$ such that $a \in N_u$ but $u \notin N_a$, there will be no communication between them when we execute LNSR.

Yet, it is very interesting that we need the unreliable agents to initiate communication in order to make it easier for the reliable agents to identify them. This raises interesting questions for real-life applications. For instance, it follows from Theorem 5.4.15 that a fake news article will be easier uncovered if this article is actively shared. Or a faulty sensor will be easier identified if its communicative power is not lost due to the error.

As a result of Theorem 5.4.15 and Examples 5.4.16 and 5.4.17, we find that the set of graphs on which LNSR is (strongly) successful in the (strong) complete sense is a proper subset of those graphs on which LNS is. In other words, we have that LNSR $\subset$ LNS. Now, because LNSR is an example of a protocol in which the agents follow different protocols, we can generalize this result to the following proposition.

Conjecture 5.4.18. *Consider all DCU graphs with strictly more than one agent. Let $\mathfrak{F}_P$ be the set of DCU graphs that is solvable (in the (strong) complete sense) by protocol P and $\mathfrak{F}_{P'}$ those that are solvable by protocol P'. Then $\mathfrak{F}_P \cap \mathfrak{F}_{P'}$ is a proper subset of the set $\mathfrak{F}$ of solvable DCU graphs G in which the agents follow a combination of protocols P and P' (i.e. A is partitioned into disjoint sets A_P and $A_{P'}$ such that $A_P \cup A_{P'} = A$ and for each agent $a \in A_P$ we have that a follows protocol P, and for each $a \in A_{P'}$, a follows P').*

Note that $\mathfrak{F}_P \cap \mathfrak{F}_{P'} \subseteq \mathfrak{F}$ because we can imagine that some of the agents follow a protocol P whereas some other agents follow the Do Not Call (DNC) protocol, i.e. the protocol that prevents agents from initiating calls. Clearly for DCU graphs with $|A| > 1$ we have $\mathfrak{F}_{DNC} = \emptyset$[1]. And as such, we have that

[1]Trivially, whenever we have an initial DCU graph with only one agent, it is solvable by DNC.

$\mathfrak{F}_P \cap \mathfrak{F}_{DNC} = \mathfrak{F}_P \cap \emptyset = \emptyset$. Therefore trivially $\mathfrak{F}_P \cap \mathfrak{F}_{P'} \subseteq \mathfrak{F}$. An example of Conjecture 5.4.18 is the combination of CMO and LNS. We know that LNS is strongly successful (i.e. any LNS-permitted call sequence is successful) on DC graphs G that are sun graphs [29] and that LNS $\subset$ CMO. Therefore we get the following Conjecture 5.4.19.

Conjecture 5.4.19. *The combination of protocols LNS and CMO is strongly successful on DC graphs G that are sun graphs.*

Studying combinations of protocols is interesting for real-life applications. There are many networks with no predefined rules for communication. In fact, we may already wonder how agents in an initial network decide on a protocol to execute without outside information—because no communication has yet taken place. In such situations, the chance that agents follow different protocols is very likely. Consider your own network of friends and family. It would be quite an exception if everyone in the network acts according to the same communication protocol.

Discussion

We have shown that any classification of DCU graphs by LNSR is a proper subset of the same classification by LNS. So there are networks G on which the unreliable agents can be identified when they are allowed to initiate calls, but they will remain uncovered when their communication is restricted. This result is very counter-intuitive: unreliability can more easily be detected when it is actively spread. In a sense, this shows that from a point of view of identification, false information that is not actively spread is worse than false information that is.

An implication of this result is that it might be the wrong strategy to cut off or block unreliable agents from the network. Because in that case it might be harder for the rest of the network to identify the unreliable agents as unreliable. One may now wonder whether this is a problem. For unreliable agents that are cut-off from the network cannot harm the network by spreading false information—so why then is it a problem that they are not identified? Clearly there is a more philosophical debate behind this. Yet, it may already be clear that when not all unreliable agents are identified, the agents that are still in touch with the unreliable agent are more likely to believe her.

This raises interesting questions for the applications of our framework, for instance identifying fake news or faulty sensors in a network of electronics. What are now for instance the good measures to take in the battle against fake news? Because from a practical point of view, letting fake news spread actively may not be the desirable solution. Further research is required to investigate these issues and to answer the question how this counter-intuitive consequence weighs against the reasons for preventing false information to spread. We will elaborate on this in Section 6.1.

5.5 The Need for New Protocols: Further Questions

We have conducted an exploration in the field of dynamic communication with a possibility of error. The motivation behind this was the question whether the assumption that *everybody is reliable* is realistic for any kind of practical application. With the introduction of the alternating bluffer, we have stepped away from this idealized assumption while still creating a formally precise framework.

In this chapter we have shown that the known results about the classification of dynamic communication networks fail to apply when a small amount of error is introduced. Specifically, for sun graphs where all unreliable agents are terminal alternating bluffers, not all LNS-permitted call sequences result in strong completeness. As such, the known correspondence between LNS and sun graphs is not applicable and cannot be extended to the case of unreliability. In particular, if there is only one terminal alternating bluffer, she will remain completely undetected by the network—that is, nobody will consider her to be unreliable. This is because the protocol LNS makes it very hard for agents to verify information as each agent can only be contacted once.

The question is now, with alternating bluffers, to what class of DCU graphs does the protocol LNS correspond and how can we adjust LNS to still classify sun graphs. For the latter, an example protocol would be an extension of LNS in which agents are allowed to verify information. In order to realize this, we will need to further specify the secret relation to account for information that is known and information that is known and verified. Clearly, the more verification is required, the more secure the network.

Additionally, we have shown that unreliability can be easier detected by the agents in the network if it can be initiated. That is, we have provided examples of networks in which the alternating bluffers cannot be identified by the protocol LNSR, but can be identified by LNS. This is because alternating bluffers that initiate communication are more on the radar—and therefore more contrary information about their secret can be encountered. This seems paradoxical because there are many reasons for preventing false information to spread in a network. Still, from an identification point of view, unreliable agents that do not initiate calls are harder to identify because they create less havoc among the reliable agents.

A natural next step is now to question whether it is desirable for a network to allow false information to be actively spread—even though this might help in identifying the unreliable sources. As a first response, the answer would be guessed no. Allowing false information to spread is not desirable. However, when not all unreliable agents are identified, even though the unreliable agents do not initiate calls, the agents in contact with the unreliable agents are more likely to consider them reliable. So even though it seems that preventing false information to be actively spread would cause less harm, the harm it does cause is more critical—because the agents learning their secret are less likely to find out that the source of these secrets is actually unreliable. Even more, it would be interesting to investigate whether this depends on the amount of unreliability present in the network and in what form.

Research Questions:

- *We have tried running the old protocol LNS on the old class of sun graphs and failed to find a precise match. The question now is for what kind of DCU graphs is LNS still strongly complete in the sense that everybody learns the secrets of the reliable agents and can identify the unreliable agents? And what kind of adaptions can we make to LNS so that it can continue to classify sun graphs? Do we need to change the data structure in order to obtain this?*
- *We have concluded that LNSR $\subset$ LNS in terms of the classes of graphs on which these protocols are successful. What more can we say about the difference between alternating bluffers that initiate calls, and alternating bluffers that do not? In addition, how can we decide, from a philosophical point of view, what is worse: false information to be actively spread or not identifying unreliable sources? Does this depend on the amount of unreliable agents?*

Of course, when considering different types of agents, different questions arise. In the next chapter, we will explore some of these.

Chapter 6

Extensions of the Framework

By introducing unreliability into the field of dynamic communication we have opened the doors to real-life applications where unreliability is more of a rule than an exception. Until now, we have focused on a relatively simple version of unreliability to extend the available framework of gossip: the alternating bluffer. The reason for this was that this agent provides a good first way of approaching applications in which unreliability occurs unintentionally in the form of error or noise. Good examples of such applications are networks of electronics that communicate over electronic channels. These systems are designed in a way that the error rate is small, yet neither are errors ruled out.

Here, we discuss a few concrete ways of extending our framework. For instance by considering a different attitude of agents (the *see for yourself* attitude) or by including *broadcast calls* or *censorship actions* in the network. These adaptations can be sorted in four groups, that represent what parts of the current framework are adjusted. For each of these groups, we first give a few examples.

- **Type and property of agents**: lying with a certain probability about different quantities (phone number, secrets, etc) and on different levels (about one own's information or about that of others)
- **Attitude of agents**: including initial uncertainty about new information (being skeptic) or a different perception toward first- and second-hand information
- **Network properties**: synchronicity of the network (to what extent calls are observed by agents not directly involved in the call) of whether broadcast calls are allowed
- **Protocols**: ANY, CMO, LNS extended with security measures against unreliable agents

It may be clear that to include higher levels of lying (to lie about someone else's secret), we need to require the agents to store more information in order to be able to identify the unreliable agents. This means, we need to adjust the data structure. We have already seen a problem with this in Chapter 3 when we considered Example 3.1.2. In Example 3.1.2, the unreliable agent is a *second-order* liar, meaning that she lies about secrets of other agents. In order to prevent this, we may want to require the agents to additionally keep track of what was said by whom.

In this chapter, we will not focus on higher levels of lying. Instead, we will elaborate more on four specific extension of our framework: censorship actions, the *see for yourself* attitude, sabotage and broadcast calls. In each section, we will end with one or more research questions that arise from the extensions. We will elaborate more on protocols with security measures in Chapter 7.

6.1 Censorship Actions

In Section 5.4 we have seen that unreliable agents that do not initiate calls are harder to identify than unreliable agents that do so. Here, we make this result more precise by showing that the property of being (strong) Trumpian is not a desirable security measure from the perspective of identification.

6.1.1 Formalisms

Recall the definition of (strong) Trumpian agents in Chapter 3. Strong Trumpian or Trumpian agents are agents with the property that if they consider someone unreliable, they will remove or block her from their contact list. To realize (strong) Trumpian agents in our framework, we need different actions to calling: censorship actions. We introduce two actions, one for deleting agents and one for blocking agents.

Definition 6.1.1. [Censorship: Delete] *Let $G = (A, R, f, N, S)$ be a DCU graph and let $a \in A$. The censorship action* delete λ_a *maps G to $G^{\lambda_a} = (A, R, f, N^{\lambda_a}, S)$ defined by*

$$N_c^{\lambda_a} = \begin{cases} N_c & \text{if } c \neq a \\ \{b \in N_c \mid b \notin Z_c\} & \text{if } c = a \end{cases} \tag{6.1}$$

The censorship action *delete* does exactly what we want: it lets agents cut links with those that they do not trust. Note that this does not only mean that the agents performing the action will not be able to contact the agents she considers unreliable but that they can also not give the phone numbers of the deleted agents to others in future calls. This makes sense naturally: once you have decided an agent is not to be trusted, you will not pass along their contact information to others that you do trust. It follows that a reliable agent a is Trumpian if she performs the action λ_a after each call ab or ba.

Definition 6.1.2. [Trumpian Reliable Agent] *A reliable agent a is* Trumpian *if she performs the censorship action λ_a after each call ab or ba with any other agent b.*

A stronger censorship action is the action in which the agents that are considered unreliable are not merely deleted, but blocked. In this situation, the agent performing the censorship action is unable to contact the agents she considers unreliable but additionally this is also true the other way around. Thus the blocked agent can also not contact the agent performing the action anymore.

Definition 6.1.3. [Censorship: Block] *Let $G = (A, R, f, N, S)$ be a DCU graph and let $a \in A$. The censorship action* block *μ_a maps G to $G^{\mu_a} = (A, R, f, N^{\mu_a}, S)$ defined by*

$$N_c^{\mu_a} = \begin{cases} N_c & \text{if } c \neq a \text{ and } c \notin Z_a \\ N_c \setminus \{a\} & \text{if } c \neq a \text{ and } c \in Z_a \\ \{b \in N_c \mid b \notin Z_c\} & \text{if } c = a \end{cases} \tag{6.2}$$

For each a in A, we will use λ_a for the censorship action *delete* and μ_a for censorship action *block*. Clearly now, a reliable agent a is (strong) Trumpian if she performs the action λ_a (μ_a) after each call ab or ba.

Definition 6.1.4. [Strong Trumpian Reliable Agent] *A reliable agent a is* Strong Trumpian *if she performs the censorship action μ_a after each call ab or ba with any other agent b.*

Note that in the block action, the blocked agents are also not able to communicate the phone number of the agent performing the action in future calls. It may be argued that this is not a desirable consequence. For phone numbers in real-life situations, it is namely not true that blocking an agent is equivalent to deleting your own phone number from that agent's contact list. Enabling blocked agents to still communicate these phone numbers comes at the cost of a partition of the accessibility relation N. As such, we will need to change the data structure. We will not elaborate on this option further and assume that blocked agents are not able to communicate the phone numbers of the agents that blocked them.

Haskell Code

```haskell
-- censorship action for Trumpian agents
censorDelete :: Agent -> Table -> Table
censorDelete a table = sort $ (a, (newN, (aX, aY, aZ), aType, aProp)):
    newTable
  where
    Just (aN, (aX, aY, aZ), aType, aProp) = lookup a table
    newN = aN \\ aZ
    newTable = table \\ [(a, (aN, (aX, aY, aZ), aType, aProp))]
```

```haskell
-- cencorship action for strong Trumpian agents
censorBlock :: Agent -> Table -> Table
censorBlock a table = sort $ (a, (newN, (aX, aY, aZ), aType, aProp)):
    newTable
  where
    Just (aN, (aX, aY, aZ), aType, aProp) = lookup a table
    newN = aN \\ aZ
    stripTable = table \\ [(a, (aN, (aX, aY, aZ), aType, aProp))]
    newTable   = adjust a aZ stripTable
    -- function to remove a's number from all agents in aZ
    adjust :: Agent -> [Agent] -> Table -> Table
    adjust _    _    []                                          = []
    adjust ags agsZ ((b, (bN, (bX, bY, bZ), bType, bProp)):rest) =
      if b `elem` agsZ
        then (b, (bN \\ [ags], (bX, bY, bZ), bType, bProp)):
             adjust ags agsZ rest
        else (b, (bN, (bX, bY, bZ), bType, bProp))         :
             adjust ags agsZ rest
```

6.1.2 Isolating completeness

With these censorship actions, we can define new protocols that aim at a different completeness result: instead of identifying the unreliable agents in the network, the aim is now to isolate the unreliable agents—i.e. create a situation in which the unreliable agents are not able to communicate with the reliable agents. We first define what it means for a network to (strongly) isolate the unreliable agents.

Definition 6.1.5. [Isolate Unreliable Agents] *A DCU graph $G = (A, R, f, N, S)$* isolates *the unreliable agents if for all $b \in R$ and for all $a \in A \setminus R$ we have that $a \notin N_b$.*

Definition 6.1.6. [Strongly Isolate Unreliable Agents] *A DCU graph $G = (A, R, f, N, S)$* strongly isolates *the unreliable agents if for all $b \in R$ and for all $a \in A \setminus R$ we have that $a \notin N_b$ and $b \notin N_a$.*

At first hand, it may seem that these notions of isolating the unreliable agents is a stronger version of identifying the unreliable agents. Because in order to isolate an unreliable agent, she needs to be uncovered first. However, this is precisely why it is not a stronger version: she does not necessarily need to be uncovered by all the agents in order to be isolated from the network. In other words, an unreliable agent can be (strongly) isolated by the network if she is identified by only some of the agents in the network. However, this does not hold for identification: to be successfully identified by the network as unreliable, she will need to be identified by each individual. Consider the following Example 6.1.7.

Example 6.1.7. *Consider the initial network $G = (A, R, f, N, S)$ with $A = \{a, b\}$, $R = \{a\}$, $f(a) = f(b) = 0$ and $N = \{(b, a)\}^R$, It is clear that the network isolates the unreliable agents (because $b \notin N_a$) but fails to identify them (because it is an initial network).*

$$\begin{tikzcd} a \arrow[loop above] & b \arrow[l] \arrow[loop above] \end{tikzcd}$$

We now show that in a network in which all the unreliable agents are identified and the reliable agents are (strong) Trumpian, the unreliable agents must already be (strongly) isolated. The proof follows directly from the definition of (strong) Trumpian agents.

Proposition 6.1.8. *If $G = (A, R, f, N, S)$ is a DCU graph in which all the unreliable agents are identified and all agents are (strong) Trumpian, then the unreliable agents are (strongly) isolated in G.*

Proof. If all the unreliable agents are identified in a network $G = (A, R, f, N, S)$, this means that for each $a \in A$ we have that $Z_a = A \setminus R$. Now, by the definition of (strong) Trumpian agents, this requires that each agent $a \in A$ has deleted (or blocked) all $b \in Z_a = A \setminus R$. Therefore, all unreliable agents are (strong) isolated in G. $\square$

We can now define different kinds of completeness that include the isolating of unreliable agents.

Definition 6.1.9. [Isolating Completeness] *A DCU graph $G = (A, R, f, N, S)$ is said to be* isolating complete *if G is complete restricted to the reliable agents and G isolates the unreliable agents.*

Definition 6.1.10. [Strong Isolating Completeness] *A DCU graph $G = (A, R, f, N, S)$ is said to be* strong isolating complete *if G is complete restricted to the reliable agents and G strongly isolates the unreliable agents.*

Note that it does not make sense to consider (strong) isolating completeness for the whole set of agents because by isolating the unreliable agents, the chances of the unreliable agents finding out all the reliable secrets approaches zero in the limit. Definition 6.1.10 of strong isolating completeness is very natural because in this situation the networks has reached a state in which the unreliable agents cannot harm the network. As a consequence of Proposition 6.1.8, we now have the following Theorem 6.1.11 about success on DCU graphs.

Theorem 6.1.11. *If $G = (A, R, f, N, S)$ is an initial DCU graph such that all $a \in R$ are (strong) Trumpian and LNS is (strongly) successful on G in the strong complete sense, then LNS is (strongly) successful on G in the (strong) isolating completeness sense.*

Proof. Follows directly from Proposition 6.1.8. $\square$

Haskell code

```
numbersTable :: Table -> [(Agent,[Agent])]
numbersTable [] = []
numbersTable ((a,(aN,(_aX,_aY,_aZ),_aType,_aProp)):rest) =
  (a,aN) : numbersTable rest

unreliableIsolated :: Table -> Bool
unreliableIsolated t = all (intersectionEmpty t) aNs where
  rTable = filter (\x -> fst x `elem` reliable t) t
  aNs = map snd (numbersTable rTable)
  intersectionEmpty :: Table -> [Agent] -> Bool
  intersectionEmpty table list
      | null $ unreliable table `intersect` list = True
      | otherwise                                = False


unreliableStrongIsolated :: Table -> Bool
unreliableStrongIsolated t = unreliableIsolated t &&
                             all (intersectionEmpty t) aNs where
  uTable = filter (\x -> fst x `elem` unreliable t) t
  aNs = map snd (numbersTable uTable)
  intersectionEmpty :: Table -> [Agent] -> Bool
  intersectionEmpty table list
      | null $ reliable table `intersect` list = True
      | otherwise                              = False
```

```
-- new definitions of completeness
isoComplete :: Table -> Bool
isoComplete table = completeR table && unreliableIsolated table

strongIsoComplete :: Table -> Bool
strongIsoComplete table = completeR table && unreliableStrongIsolated
    table
```

```
-- check whether a node is solved
solvedIso :: Node -> Bool
solvedIso (table, _, _) = isoComplete table

solvedStrongIso :: Node -> Bool
solvedStrongIso (table, _, _) = strongIsoComplete table

-- use the search tree to find solutions
solveNsIso :: [Node] -> [Node]
solveNsIso = search extendNode solvedIso

solveNsStrongIso :: [Node] -> [Node]
solveNsStrongIso = search extendNode solvedStrongIso

-- print calling sequences that complete the network
-- with the definitions of completeness above
solveAndShowIso :: Table -> Protocol -> IO ()
solveAndShowIso t p = solveShowHistOfIso [(t, [], p)]

solveAndShowStrongIso :: Table -> Protocol -> IO ()
solveAndShowStrongIso t p = solveShowHistOfStrongIso [(t, [], p)]

-- print history
solveShowHistOfIso :: [ Node ] -> IO ()
solveShowHistOfIso = sequence_ . fmap showHistOf . solveNsIso

solveShowHistOfStrongIso :: [ Node ] -> IO ()
solveShowHistOfStrongIso = sequence_ . fmap showHistOf . solveNsStrongIso

-- check if a graph is solvable
solvableIso :: Table -> Protocol -> Bool
solvableIso t p = solveNsIso [(t, [], p)] /= []

solvableStrongIso :: Table -> Protocol -> Bool
solvableStrongIso t p = solveNsStrongIso [(t, [], p)] /= []
```

6.1.3 Discussion

Even though a strong isolating complete network seems a very nice situation in which the network has reached a state in which the unreliable agents cannot harm the network, this comes at a certain cost. Unreliable agents may namely still have useful information for the reliable agents. Consider the following Example 6.1.12.

Example 6.1.12. *Let $G = (A, R, f, N, S)$ be the network drawn below where $R = A\setminus\{b\}$ and the agents $a \in A\setminus\{b, c\}$ form a cluster (so for each $a \in A\setminus\{b, c\}$, we have that $X_a \cup Y_a = A \setminus \{b, c\}$) that consider agent b unreliable, i.e. $Z_a = \{b\}$. Furthermore, we assume that $c \notin N_a$ and $a \notin N_c$ for each a in the cluster.*

$$\begin{tikzcd}
A \setminus \{b,c\} \arrow[r] & b \arrow[r] \arrow[loop above] & c \arrow[loop above]
\end{tikzcd}$$

In Example 6.1.12, if all reliable agents are (strong) Trumpian, the connections ab (and ba) with $a \in A \setminus \{b, c\}$ will be cut. In the case that the reliable agents are strong Trumpian, this has the consequence that also agent c will be isolated from the reliable cluster $A \setminus \{b, c\}$. Hence agent c will never become part of the network, though she is in fact reliable. Note that since we have not specified X_c, Y_c and Z_c, it is not necessarily the case that the network is strong complete nor (strongly) isolating complete.

This downside can only be partly justified by the fact that reliable agents in our framework may never know when the discovered unreliable agents speak the truth. In this line of arguing, we can say that there is no reason to not (strongly) isolate the unreliable agents because their information can never be trusted entirely. From this perspective, censorship actions may seem a natural response to unreliability.

However, Example 6.1.7 and Theorem 5.4.15 do provide an argument against censorship actions as security measures to deal with unreliable agents. These namely show that unreliable agents that do not actively initiate calls in the network are harder to identify than those unreliable agents that do. Because from the perspective of identification, censorship actions may have the contrary result: by isolating the unreliable agents, they will not be able to actively initiate calls and therefore they are harder to identify. This also follows already from the fact that in order for a network to isolate an unreliable agent, she does not have to be considered unreliable by all the agents. Consider for example the situation in which the unreliable agents are terminal in an initial network—then they are clearly isolated but not identified. Yet, to be successfully identified, you will have to be identified by each individual.

Another question about censorship actions as security measures against unreliable against is raised from a practical point of view. This is because there are many reasons why letting false information spread freely throughout the network is not a satisfying solution. It then depends on what counts more: preventing the false information to spread or identifying the unreliable sources. At first, it may seem that preventing the spread of false information weighs much more than the identification of unreliable sources. But now consider that in an attempt to strongly isolate the unreliable agents, some reliable agent is still in contact with the unreliable agent. If we then assume that she has no prior knowledge about the secret of this unreliable agent, she will consider her reliable. And because the aim is now not to identify the unreliable agents, but isolate them, she may never find out about the unreliability of this agent. In other words, she may continue to consider the information learned from this agent to be truthful for ever. This is where not successfully identifying the unreliable sources can still be harmful for the network.

The relevance of our results for this discussion is now that seemingly choosing either one will come at the cost of the other. This means that when one chooses for the prevention of the spreading of false information, this might actually result in making it easier for the unreliable sources to remain undetected. As such, intuitive security measures that prevent false information to spread may have rigorous counter-intuitive consequences for the identification of the unreliable sources.

Of course, with different protocols the game is changed. Further research is required to shed light on these issues in connection to network security.

Research Question:

- *Censorship actions may prevent the spread of false information, but have the result that unreliable sources are harder to detect. How do now these counter-intuitive consequences weigh against the reasons for preventing false information to spread? Is it also true for other security measures that we need to choose between prevention and identifying?*

6.2 See For Yourself

In this section, we will shift from different properties to different attitudes of agents. Until now, our framework was based on the assumption that agents consider new information *true until it is proved false*, also called the *presumption of innocence* or *naive* approach. This was based on the fact that we considered dynamic communication with unreliability for practical applications in which unreliability occurs unintentionally in the form of error or noise. In this section we drop this assumption and research different attitudes. In particular, we are interested in the case that the agents are skeptic toward second-hand information. This means that agents keep the same naive attitude toward new information involving secrets or phone numbers, but are skeptic toward distrust in agents. For example, if agent a communicates with agent b and agent b tells agent a that she considers agent c unreliable, agent a will not simply adopt this distrust in agent c. This can be viewed as agent a wanting to find out for herself whether agent c is unreliable. For this reason, this attitude is called the *see for yourself* attitude.

6.2.1 Formalisms

Formally, this new attitude requires a change of Definition 4.2.4. Namely a change in the way that the set Z_a of an agent a is updated. Specifically, the clause for Z' will change to the following for agents a and b:

$$Z'_a = Z_a \cup \big((Y_a \cap X_b) \cup (Y_b \cap X_a)\big)$$

$$Z'_b = Z_b \cup \big((Y_a \cap X_b) \cup (Y_b \cap X_a)\big)$$

This is indeed what we want, both agents a and b do not adopt the distrust of the other agent. We then have that $Z_a \cup Z_b$ will not necessarily be a subset of Z'_a or Z'_b.

It is clear that with this new attitude, it will likely take longer for the agents to identify the unreliable agents in a network. This is because every agent wants to determine for herself whether someone is unreliable. With higher levels of lying, however, this attitude may as well be beneficial. Consider the following Example 6.2.1.

Example 6.2.1. *Let $G = (A, R, f, N, S)$ be a DCU network with $A = \{a, b, c\}$, $R = \{a, c\}$, $f(a) = f(c) = 1$, $f(b) = 0$, $N = \{(a, b), (b, c)\}^R$ and $S_a = (\{a\}, \emptyset, \emptyset)$, $S_b = (\emptyset, \{b\}, \{c\})$ and $S_c = (\{c\}, \emptyset, \emptyset)$. See also the figure below. Note that the reflexive arrows are omitted.*

$$\begin{tikzcd}
a \arrow[loop above] \arrow[r] & b \arrow[loop above] \arrow[r] & c \arrow[loop above]
\end{tikzcd}$$

In Example 6.2.1, after the call ab, with the old attitude agent a would incorrectly learn that agent c is unreliable. With the new attitude, however, agent a would not accept the information that c is unreliable and as a next step agent a could call agent c to learn her secret. If agent a then does not encounter contrary information about agent c's secret, she will, in this case correctly, continue to consider her reliable.

Haskell Code

```
callSFY :: (Agent, Agent) -> Table -> [Table]
callSFY (a,b) table = let
  Just (aN, (aX, aY, aZ), aType, aProp) = lookup a table
  Just (bN, (bX, bY, bZ), bType, bProp) = lookup b table
  newN = merge aN bN
  disagreementsX = (aX `intersect` bY) `union` (bX `intersect` aY) `union
      ` (aX `intersect` bZ) `union` (bX `intersect` aZ)
  disagreementsY = (aX `intersect` bY) `union` (bX `intersect` aY) `union
      ` (aY `intersect` bZ) `union` (bY `intersect` aZ)
  newX = merge aX bX \\ disagreementsX
  newY = merge aY bY \\ disagreementsY
  new_aZ = aZ `union` (aX `intersect` bY) `union` (bX `intersect` aY)
  new_bZ = bZ `union` (aX `intersect` bY) `union` (bX `intersect` aY)
  stripTable = table \\ [(a, (aN, (aX, aY, aZ), aType, aProp)), (b, (bN,
      (bX, bY, bZ), bType, bProp))]
  addedTable = sort $ (a, (newN, (newX, newY, new_aZ), aType, aProp)):(b,
       (newN, (newX, newY, new_bZ), bType, bProp)):stripTable
 in
   if possible table (a,b)
      then typeBehavior [(a,aType,aProp),(b,bType,bProp)] addedTable (a,b
          )
      else error "forbidden call!"
```

```
callsSFY :: Seq -> Table -> [Table]
callsSFY [] table = [table]
callsSFY cs table = concatMap (callsSFY (tail cs)) (callSFY (head cs)
    table)

extendNodeSFY :: Node -> [Node]
extendNodeSFY (table, hist, prot) = [(newTable, hist ++ [(x,y)], prot) |
    (x,y) <- candidates (table, hist, prot), newTable <- callSFY (x,y)
    table]

solveNsSFYR :: [Node] -> [Node]
solveNsSFYR = search extendNodeSFY solvedR

solveAndShowSFYR :: Table -> Protocol -> IO()
solveAndShowSFYR t p = solveShowHistOfSFYR [(t, [], p)]

solveShowHistOfSFYR :: [Node] -> IO()
solveShowHistOfSFYR = sequence_ . fmap showHistOf . solveNsSFYR
```

6.2.2 Discussion

With the *see for yourself* attitude we provide a first approach to exploring skepticism in dynamic communication networks with unreliability. The next step would be to include a stronger skepticism of agents toward new information. For instance to regard new information as “potentially reliable” instead of reliable. In order to realize this, it is expected that the data structure needs to be altered so that the information that agents know can be rated according to how reliable they consider it. In addition, we may be interested in considering the attitude of agents to be dependent on the provider of the information or for attitudes to be a dynamic quantity. That is, agents could adopt their attitude toward other agents depending on what information they possess. We will explore these options further in Chapter 7.

Next to approaching skepticism in dynamic communication with the *see for yourself* attitude, this attitude is also a first adaptation of our framework to deal with higher levels of lying. This is because, as we have seen, if an unreliable agent lies about the reliability of another agent, this attitude may help in uncovering this lie.

Both the *see for yourself* attitude and the *presumption of innocence* attitude that we considered before rely on the fact that there are no fact-checkers in the network. Intuitively, with fact-checkers the game changes. Because then being skeptic toward an agent could be translated to checking the information you have about that agent with the fact-checker. To prevent that all information is verified with the fact-checker, it would be natural to require calls to a fact-checker to be costly. Of course, from an efficiency point of view, it already is.

In conclusion, the *see for yourself* attitude is a first start in exploring different attitudes of agents. More research is needed to explore what other attitudes are possible and are sufficient in dealing with different types of agents (i.e. uncovering).

Research Questions:

- *We have altered the definitions of DCU graphs to go from a naive point of view to the* see for yourself *attitude. What are the next steps in changing the data structure to allow for more skepticism? For instance to consider all new information reliable to a certain extend.*

- *How do the different attitudes work when considering different agent types, for instance the fact-checker? It is expected that agents can adopt a stronger skepticism when they are allowed to verify information with an all-knowing source.*

6.3 Sabotage

In the previous chapters, we have focused on unreliability concerning secrets. The alternating bluffer was an agent lying about her own secret in every second call. Here we briefly explore a different form of unreliability, unreliability about phone numbers: the saboteur. Saboteurs are agents that have the ability to cut connections between agents—this can be considered as lying about your contact list.

In this primary approach, we consider for simplicity that after each call the saboteur is involved in (i.e. sa or as for a saboteur s) the saboteur can only perform a single *cut* action targeted at the agent that she has been calling with (agent a). Limiting the power of the saboteur in this way intuitively makes sense: a saboteur does not have more information about the network than the information available to her, and she needs to be in contact with someone in order to "lie" about a phone number. So this also means that after the call sa or as, a saboteur s can only cut a connection ab such that either $b \in N_s$ or $b \in N_a$. In words, she can only lie about a phone number in her own contact list or in the contact list of the targeted agent.

Why we allow the latter is because if the call sa or as takes place and agent a shares her contact list N_a with the saboteur s, s can claim that some $b \in N_a$ changed phone numbers. Trivially, saboteurs can only cut non-reflexive, existing connections between agents. In fact, this is a special case of the censorship action delete: instead of only being able to delete her own phone number from the contact list of a, the phone number of any agent $b \in N_a^{sa}$ can be deleted by the saboteur.

Note that our framework does yet have the tools to identify saboteurs. This is because saboteurs do not introduce any contrary information in the network. An adaption of our framework may be to let saboteurs really lie about phone numbers instead of deleting them. Then this would indeed cause contrary information, but now about phone numbers. Further research into this option is required for exploratory purposes.

With the saboteur, we are not only interested in the call sequences that complete a network, bur rather in those sequences that do this *regardless* of the connections that are cut by the saboteurs. We get the following Definition 6.3.1.

Definition 6.3.1. [Saboteur Game] *Given an initial DCU graph $G = (A, R, f, N, S)$ with $R \subsetneq A$, at least one agent $a \in A \setminus R$ of the type saboteur and a protocol P, the* saboteur game *is as follows: if there is any P-maximal sequence σ such that G^σ is not complete, the saboteurs win. Otherwise the reliable agents win. In the latter case we also say that the* saboteur cannot harm *the network.*

We now asks ourselves what properties a network needs to possess in order to unable the saboteur from doing harm to the network—i.e. the network can be completed no matter what cut actions the saboteurs perform. Limiting the saboteurs by requiring them to be terminal is at least not going to work. Consider Example 6.3.2.

Example 6.3.2. *Let $G = (A, R, f, N, S)$ be an initial DCU network with $A = \{a, b, c\}$, $R = \{a, b\}$ and c of the type saboteur. Furthermore, let $N = \{(a, b), (a, c)\}^R$. Then the call sequence $\sigma = ac; cb$ results in a stuck network if c cuts the link ab in the first call and the link ba in the second call. The result of this is $G^\sigma = (A, R, f, N^\sigma, S^\sigma)$ where $N^\sigma_a = S^\sigma_a = \{a, c\}$, $N^\sigma_b = \{b, c\}$, $S^\sigma_b = \{a, b, c\}$ and $N^\sigma_c = S^\sigma_c = \{a, b, c\}$.*

In Example 6.3.2, there is only a single terminal saboteur in the initial network. Yet, as the example shows, she is able to sabotage the network by cutting the link between the reliable agents. This means that agent a will not learn the secret of agent b. Note that further restricting the saboteur to not initiate any call can result the same way if ac is the first call to be made. Because in that case, G^{ac} would be stuck. It is not the case that there is no successful call sequence on G. If the call ab would take place before ac, all the agents would know the secrets of the reliable agents. However, because agents make knowledge-based decisions and the agents in Example 6.3.2 do not have any prior knowledge about the network or in particular about the agent types, agent a would have no preference to perform the call ab over the call ac. Therefore the network is not safe from the attack of a saboteur and the saboteur wins the saboteur game.

It follows from Example 6.3.2 that in case of the saboteur, we do not necessarily have that $S^\sigma \subseteq N^\sigma$ which was true for DC graphs and DCU graphs with alternating bluffers. This is obvious because the saboteur cuts the N-relation without harming the S-relation between agents.

Haskell Code

We need a code for cutting connections and finding the connections that a saboteur is allowed to cut. Remember that after a call *sa* or *as*, the saboteur *s* can cut any connection aNb such that aNb and $a \neq b$.

```haskell
-- cut link between two agents
cut :: (Agent, Agent) -> Table -> Table
cut (a,b) table = sort $ (a, (new_aN, (aX, aY, aZ), aType, aProp)):
                         (b, (new_bN, (bX, bY, bZ), bType, bProp)):
                         newTable
  where
    Just (aN, (aX, aY, aZ), aType, aProp) = lookup a table
    Just (bN, (bX, bY, bZ), bType, bProp) = lookup b table
    new_aN = aN \\ [b]
    new_bN = bN \\ [a]
    newTable = table \\ [(a, (aN, (aX, aY, aZ), aType, aProp)),
                         (b, (bN, (bX, bY, bZ), bType, bProp))]

-- finds all the connections in a table
connections :: Table -> [(Agent, Agent)]
connections []                                              = []
connections ((a, (aN, (_aX, _aY, _aZ), _aType, _aProp)):rest) =
  sort $ makelist a aN ++ connections rest
  where
    makelist :: Agent -> [Agent] -> [(Agent, Agent)]
    makelist _ []      = []
    makelist c (b:cNs) = (c,b): makelist c cNs

-- finds all the non-reflexive connections in a table
filteredConnections :: Table -> Agent -> (Agent, Agent) ->
                           [(Agent, Agent)]
-- a is the saboteur and bc is the call taking place
filteredConnections table a (b,c) =
  filter (\x -> fst x /= a && (fst x == b || fst x == c)) newTable
  where
    newTable = filter (uncurry (/=)) (connections table)
```

We define a new version of Table and Node that keep track of the connections that were cut.

```haskell
type SabTable = (Table, [(Agent, (Agent, Agent))])

-- SabNode keeps track of which agent cut which links
type SabNode = (SabTable,Seq,Protocol)

-- new version of call
callSab :: (Agent, Agent) -> SabTable -> [SabTable]
callSab (a,b) (table, sabhist) = let
  Just (aN, (aX, aY, aZ), aType, aProp) = lookup a table
  Just (bN, (bX, bY, bZ), bType, bProp) = lookup b table

  -- Leeches do not share numbers
  -- FactCheckers do not update their numbers
  -- (and therefore do also not share any)
  new_aN | aType == FactChecker = aN
         | bType == Leech       = aN
         | otherwise            = merge aN bN
  new_bN | bType == FactChecker = bN
         | aType == Leech       = bN
         | otherwise            = merge aN bN

  disagreementsX = (aX `intersect` bY) `union` (bX `intersect` aY)
                                       `union` (aX `intersect` bZ)
                                       `union` (bX `intersect` aZ)
  disagreementsY = (aX `intersect` bY) `union` (bX `intersect` aY)
                                        `union` (aY `intersect` bZ)
                                        `union` (bY `intersect` aZ)

  -- FactCheckers do not tell which agents are unreliable
  -- (this could be an option as well)
  new_aX | bType == Leech       = aX
         | bType == FactChecker = aX `union` [b] \\ (aX `intersect` bZ)
         | otherwise            = merge aX bX \\ disagreementsX
  new_bX | aType == Leech       = bX
         | aType == FactChecker = bX `union` [a] \\ (bX `intersect` aZ)
         | otherwise            = merge aX bX \\ disagreementsX

  new_aY | bType == Leech       = aY
         | bType == FactChecker = aY \\ (aY `intersect` bZ)
         | otherwise            = merge aY bY \\ disagreementsY
  new_bY | aType == Leech       = bY
         | aType == FactChecker = bY \\ (bY `intersect` aZ)
         | otherwise            = merge aY bY \\ disagreementsY

  new_aZ | bType == Leech       = aZ
         | bType == FactChecker = aZ `union` (aX `intersect` bZ)
                                    `union` (bX `intersect` aZ)
         | otherwise            = merge aZ bZ `union` (aX `intersect` bY)
                                             `union` (bX `intersect` aY)
  new_bZ | aType == Leech       = bZ
         | bType == FactChecker = bZ `union` (aX `intersect` bY)
                                    `union` (bX `intersect` aY)
         | otherwise            = merge aZ bZ `union` (aX `intersect` bY)
                                             `union` (bX `intersect` aY)

  stripTable = table \\ [(a, (aN, (aX, aY, aZ), aType, aProp)),
                         (b, (bN, (bX, bY, bZ), bType, bProp))]
  newTable   = sort $
                (a, (new_aN, (new_aX, new_aY, new_aZ), aType, aProp)) :
                (b, (new_bN, (new_bX, new_bY, new_bZ), bType, bProp)) :
                stripTable
 in
   if possible table (a,b)
      then typeBehaviorSab [(a,aType,aProp),(b,bType,bProp)]
                           (newTable, sabhist) (a,b)
      else error "forbidden call!"
```

We adjust the code for typeBehavior.

```
typeBehaviorSab :: [(Agent, Agenttype, Agentproperty)] -> SabTable
                           -> (Agent, Agent) -> [SabTable]
typeBehaviorSab [] (table, sabhist) _ = [(table, sabhist)]
typeBehaviorSab ((a,aType,aProp):rest) (table, sabhist) (b,c)
  | aType == AlternatingBluffer && aProp == None
      = typeBehaviorSab rest (alternate a table,sabhist) (b,c)
  | aType == AlternatingBluffer && aProp == TrumpianDel
      = typeBehaviorSab rest
               (alternate a (censorDelete a table), sabhist) (b,c)
  | aType == AlternatingBluffer && aProp == TrumpianBlock
      = typeBehaviorSab rest
               (alternate a (censorBlock a table), sabhist) (b,c)
  | aType == Reliable           && aProp == None
      = typeBehaviorSab rest (table, sabhist) (b,c)
  | aType == Reliable           && aProp == TrumpianDel
      = typeBehaviorSab rest (censorDelete a table, sabhist) (b,c)
  | aType == Reliable           && aProp == TrumpianBlock
      = typeBehaviorSab rest (censorBlock a table, sabhist) (b,c)
  | aType == Saboteur           && aProp == None
      = concat [ typeBehaviorSab rest (cut unluckyPair table,
                         sabhist ++ [(a, unluckyPair)]) (b,c)   |
                 unluckyPair <- filteredConnections table a (b,c)]
  | otherwise
      = typeBehaviorSab rest (table, sabhist) (b,c)


callsSab :: Seq -> SabTable -> [SabTable]
callsSab [] sabtable = [sabtable]
callsSab cs sabtable = concatMap (callsSab (tail cs))
                                 (callSab (head cs) sabtable)
```

We define what it means for a node to be stuck.

```
stuck :: SabNode -> Bool
stuck ((table,_), hist, prot) = not (complete table) &&
                                null (candidates (table, hist, prot))

extendSabNode :: SabNode -> [SabNode]
extendSabNode ((table, sabhist), hist, prot) =
  [(newTable, hist ++ [(x,y)], prot) | (x,y) <- candidates (table, hist,
      prot),
                                       newTable <- callSab (x,y) (table,
                                           sabhist)]

saboteurGame :: [SabNode] -> [SabNode]
saboteurGame = search extendSabNode stuck

saboteurGameShow :: Table -> Protocol -> IO()
saboteurGameShow t p = saboteurGameShowHistOf [((t, []), [], p)]

saboteurGameShowHistOf :: [SabNode] -> IO()
saboteurGameShowHistOf = sequence_ . fmap showSabHistOf . saboteurGame

showSabHistOf :: SabNode -> IO()
showSabHistOf = print . sabHistOf where
  sabHistOf ((_,sabhist),hist,_) = (hist, sabhist)
```

Find out if at an initial DCU graph and a protocol P, it holds that the saboteurs cannot harm.

```
sabFindHistOf :: SabNode -> (Seq,[(Agent,(Agent,Agent))])
sabFindHistOf ((_,sabhist),hist,_) = (hist,sabhist)

sabGameFind :: SabTable -> Protocol -> [(Seq,[(Agent,(Agent,Agent))])]
sabGameFind t p = solveFindSabHistOf [(t, [], p)]

solveFindSabHistOf :: [SabNode] -> [(Seq,[(Agent,(Agent,Agent))])]
solveFindSabHistOf = fmap sabFindHistOf . saboteurGame

filterHistSabHist :: [(Seq,[(Agent,(Agent,Agent))])] -> [(Seq,[(Agent,(
     Agent,Agent))])]
filterHistSabHist [] = []
filterHistSabHist ((hist,sabhist):rest)
  | null sabhist = filterHistSabHist rest
  | otherwise    = (hist,sabhist) : filterHistSabHist rest

sabCannotHarm :: Table -> Protocol -> Bool
sabCannotHarm t p = null $ filterHistSabHist $ sabGameFind (t,[]) p
```

6.3.1 Discussion

So far, our framework has taken a step toward practical applications by dropping the assumption that *everybody is reliable*. With the introduction of the saboteur, we take another step from the reliable case by additionally invalidating the condition that $N^\sigma \subseteq S^\sigma$ for any call sequence σ. As such, we come to a more general framework for modeling unreliability in dynamic communication: Agents can now not only be unreliable about secrets, but also about phone numbers.

The question now is how to deal with saboteurs. That is, how can we equip the reliable agents in order to uncover saboteurs? We have already suggested that considering saboteurs to lie about phone numbers rather than deleting them might open doors for the reliable agents to uncover the saboteurs. For this it is needed that the data structure is adjusted accordingly by splitting the number relation just like we did in the case of secrets.

In conclusion, with the saboteur we have considered another approach in stepping away from the current framework of dynamic communication without unreliable agents. Further research is required to specify these ideas and come with concrete ways to adjust the data structure so that saboteurs can be uncovered.

Research Questions:

- *With what kind of properties of the network of with what protocols can the saboteurs not do any harm—in the sense that regardless of what connections are cut, the call sequences result in a (strong) complete network?*
- *How can the framework of dynamic communication be altered so that agents are equipped with tools to uncover saboteurs? Does considering numbers of agents as specific values do this trick?*

6.4 Broadcast Calls

The last extension of our framework that we discuss is to include the option for broadcast calls. Broadcast calls are calls that are made by a single agent to multiple agents at the same time. Such calls can be thought of as sending a message to a group of agents via e-mail or more recent platforms such as WhatsApp or Facebook. In this way, a broadcast call is a *public announcement* to a group of agents. With this extension, we come closer to modeling the real-life dissemination of information and connect the current framework to the logic of public announcements. More on broadcast calls in networks can be found in [15].

6.4.1 Formalisms

Naturally, in a broadcast call the agent that initiates the call shares information but does not receive any information back from the agents contacted. We have the following Definition 6.4.1, adopted from the definition of a call.

Definition 6.4.1. [Broadcast Call] *Let $G = (A, R, f, N, S)$ be a DCU Graph, $a, b_1, b_2, \ldots, b_n \in A$ with $n \in \mathbb{N}$ and Nab_i for each $i \in \{1, \ldots, n\}$. The* broadcast call aB *with* $B = \{b_1, \dots, b_n\}$ *maps* G *to* $G^{aB} = (A, R, f, N^{aB}, S^{aB})$ *defined by*

$$N_c^{aB} = \begin{cases} N_a \cup N_{b_i} & \textit{if } c = b_i \textit{ for some } i \in \{1, ..n\} \\ N_c & \textit{otherwise} \end{cases} \tag{6.3}$$

$$S_c^{aB} = \begin{cases} (X'_i, Y'_i, Z'_i) & \textit{if } c = b_i \textit{ for some } i \in \{1, ..n\} \\ (X_c, Y_c, Z_c) & \textit{otherwise} \end{cases} \tag{6.4}$$

Where

$$X'_i = X_a \cup X_{b_i} \setminus \big((X_a \cap Y_{b_i}) \cup (X_{b_i} \cap Y_a) \cup (X_a \cap Z_{b_i}) \cup (X_{b_i} \cap Z_a)\big)$$

$$Y'_i = Y_a \cup Y_{b_i} \setminus \big((X_a \cap Y_{b_i}) \cup (X_{b_i} \cap Y_a) \cup (Y_a \cap Z_{b_i}) \cup (Y_{b_i} \cap Z_a)\big)$$

$$Z'_i = (Z_a \cup Z_{b_i}) \cup \big((Y_a \cap X_{b_i}) \cup (Y_{b_i} \cap X_a)\big)$$

Haskell Code

We adjust the code for a call to account for broadcast calls.

```haskell
-- update the Table with a broadcast call
broadcastCall :: (Agent, [Agent]) -> Table -> [Table]
broadcastCall (_,[]) table = [table]
broadcastCall (a,(b:brest)) table = let
  Just (aN, (aX, aY, aZ), aType, aProp) = lookup a table
  Just (bN, (bX, bY, bZ), bType, bProp) = lookup b table

  -- Leeches do not share numbers
  -- Fact-checkers do not update their numbers
  -- (and therefore do also not share any)
  new_bN | bType == FactChecker = bN
         | aType == Leech       = bN
         | otherwise            = merge aN bN

  disagreementsX = (aX `intersect` bY) `union` (bX `intersect` aY)
                                       `union` (aX `intersect` bZ)
                                       `union` (bX `intersect` aZ)

  disagreementsY = (aX `intersect` bY) `union` (bX `intersect` aY)
                                       `union` (aY `intersect` bZ)
                                       `union` (bY `intersect` aZ)

  -- Fact-checkers do not tell which agents are unreliable
  -- (this could be an option as well)
  new_bX | aType == Leech       = bX
         | aType == FactChecker = bX `union` [a] \\ (bX `intersect` aZ)
         | otherwise            = merge aX bX \\ disagreementsX

  new_bY | aType == Leech       = bY
         | aType == FactChecker = bY \\ (bY `intersect` aZ)
         | otherwise            = merge aY bY \\ disagreementsY

  new_bZ | aType == Leech       = bZ
         | bType == FactChecker = bZ `union` (aX `intersect` bY)
                                     `union` (bX `intersect` aY)
         | otherwise            = merge aZ bZ `union` (aX `intersect` bY)
                                              `union` (bX `intersect` aY)

  stripTable = table \\ [(b, (bN, (bX, bY, bZ), bType, bProp))]
  newTable   = broadcastCall (a,brest)
```

```
                        (sort $ (b, (new_bN , (new_bX , new_bY , new_bZ),
                             bType , bProp)) :
                        stripTable)
 in
   if possible table (a,b)
      then typeBehavior [(a,aType,aProp),(b,bType,bProp)] (head newTable)
            (a,b)
      else error "forbidden call!"

-- multiple calls
broadcastCalls :: [(Agent,[Agent])] -> Table -> [Table]
broadcastCalls [] table = [table]
broadcastCalls cs table = concatMap (broadcastCalls (tail cs)) (
     broadcastCall (head cs) table)
```

6.4.2 Discussion

We have already discussed that broadcast calls in dynamic communication connect our framework to public announcements in Dynamic Epistemic Logic (DEL). Such public announcement impose on the structure that the possible worlds in which the public announcement is false will be deleted after the public announcement has been announced.

In our situation, however, we have to consider the possibility of a false public announcement. Such a false public announcement is also called a *public lie*. Research on public lies and how to recover from them is done in [31, 1].

Clearly, this connection is interesting because we might gain valuable insights about dynamic communication with broadcast calls from studying public announcements and public lies. In particular, the ideas about recovering from public lies are interesting in deciding how agents should act upon learning that a public announcement was in fact a public lie.

Research Question:

- *What can we learn from public announcements and public lies when considering broadcast calls? Are the ideas on how to recover from a public lie relevant for our framework?*

6.5 Conclusions

In our new framework, we have stepped away from the assumption that *everybody is reliable* and looked at the consequences of this for dynamic communication networks. In particular, we have adjusted the current framework of dynamic communication to account for the alternating bluffer. In this chapter, we have considered concrete ways of extending our framework. Next to dropping this idealized assumption, we have explored a different aim of the network (isolating the unreliable agents), a different attitude that takes into account that not all new information is reliable (the *see for yourself* attitude), a step away from the assumption that $S^\sigma \subseteq N^\sigma$ (the saboteur) and another option for communication (broadcast calls).

Further research is needed to continue formalizing these ideas and to consider these ideas in combination with different protocols and different agent types.

Chapter 7

Future Research and Applications

We have discussed a few concrete extensions of our framework in the previous chapter. Here we will introduce more drastic directions for future research and show how our framework opens new doors in the field of dynamic communication with unreliable agents.

We explore possible research questions related to security protocols and a different approach to the parameters on which our framework is based. For instance by treating the parameters as dynamic quantities of the network instead of static. It may be clear that these directions for future research relate closely to real-life situations.

In addition, we will highlight some applications of this work, ranging from electronic networks to cryptocurrencies and herd immunity. This wide range of applications illustrates the importance of research done in this field. Already by including a small amount of unreliability into dynamic communication (in the form of the alternating bluffer), the results that are known about DC graphs fail to hold. As such, for any kind of practical application in which unreliability may occur in the form of error or noise, the assumption that *everybody is reliable* should be questioned and discarded. As a result, the relevance of this thesis goes far beyond the framework of dynamic communication with unreliable agents.

In the following we will elaborate on directions for future research and discuss a wide range of applications of our work.

7.1 Future Research

Here, we will dive into some possibilities for future research into dynamic communication with unreliability. A great interest for future research lies in the investigation of security protocols. These are protocols that prevent unreliability from remaining unidentified—i.e. help to uncover the unreliable agents. This is in particular of interest from a knowledge-based perspective: what security measures can the reliable agents individually take based on their knowledge of the network?

Other options for future research are to treat the parameters as dynamic quantities instead of static, or to require agents to share more information in communication than only secrets and phone numbers. For instance about their own agent type or the protocol they are following. Both these options raise the question: does this make it easier to uncover the unreliable agents?

Yet, first we will explore agent and network rates, a measure for the amount of unreliability in an agent's contact list and in the network, respectively. We will explain how these definitions may help to further investigate agent types and properties.

7.1.1 Agent and network rates

In order to measure unreliability, we introduce the notions *rate of an agent* and *network rate*. These notions put a number on the amount of unreliability presence in the direct surrounding of an agent (namely in her contact list) and in the network as a whole. The idea for these notions stems from the philosophical question concerning unreliable connections: what happens when an agent only has unreliable connections, i.e. when all agents in her contact list are unreliable. We say that in such a situation, an agent is *doomed*. It is obvious that the property of being doomed is not much a wanted property.

Definition 7.1.1. [Doomed Agent] *An agent a is said to be* doomed *if for all agents b such that $b \in N_a$, $b \notin R$. In other words, agent a is doomed if she only has connections to agents that are unreliable. An agent a is is said to be* initially doomed *if she is doomed in the initial network (before any calls are made). An agent is said to be* strongly doomed *if additionally all the agents that know her phone number are also unreliable. I.e. for all b such that $a \in N_b$, $b \notin R$.*

Even though this indeed gives a measure of unreliability, in our dynamic setting this property is dynamic rather than static. This is because we consider communication dynamically by letting agents expand their connections. Thus that means that the property doomed is not a fatal state of an agent. In fact, there are situations in which an initially doomed agent might still successfully identify the unreliable agents when the network evolves. This is because even though all the agents in the initial contact list of the doomed agent are unreliable, she might learn the phone numbers of other reliable agents. As such, a doomed agent may not remain doomed.

However, we can think of cases where the unreliable agents wish to harm the network by making it harder for the doomed agent to "escape" this situation. This is for instance the case if the unreliable agents do not share the phone numbers of reliable agents, or the phone numbers of other agents in general. With other types of agents the situation may change. For instance, the presence of a fact-checker in the network can change the game in the opposite way. A fact-checker may namely facilitate such an escape because the doomed agent may now verify her connections with the fact-checker.

The definition of a doomed agent is a particular case of what we can call the *rate of an agent*. This is a number between 0 and 1 that gives an indication of the degree to which the connections of an agent are reliable. A doomed agent is the particular case in which the rate r_a of an agent a is equal to 0.

Definition 7.1.2. [Rate of an Agent] *The* rate of an agent*, r_a, is defined as:*

$$r_a = \frac{\textit{number of reliable agents in agent a's contacts}}{\textit{number of agents in agent a's contacts}}$$

In a more mathematical notation, this becomes

$$r_a = \frac{\sum_{|N_a \cap R|} b}{|N_a|}$$

Where $N_a = \{b \in A \mid aNb\}$ and $R \subseteq A$ is the set of reliable agents in the network.

Note that the rate for an agent a only takes into account agents b such that aNb, but not agents c such that cNa. Therefore, even though the rate naturally is a measure of how "good" the agent's position is in the network, not all information about the position in the network can be inferred from r_a. Though r_a does give a measure of the reliability of information that agent a can actively obtain—namely via her own connections.

There is a natural way of extending the rate of an agent to apply to the network as a whole.

Definition 7.1.3. [Rate of a Network] *The* rate of a network*, r_G, is defined as:*

$$r_G = \frac{\textit{number of reliable connections in G}}{\textit{number of all connections in G}}$$

Where a connection aNb is reliable iff $a, b \in R$. Mathematically speaking, we have the following definition

$$r_G = \frac{\sum_{a \in R} |N_a \cap R|}{\sum_{a \in A} |N_a|}$$

This is not the same as the fraction of reliable agents that are present in the network, $\frac{|R|}{|A|}$. Consider the following Examples 7.1.4 and 7.1.5.

Example 7.1.4. *Let $G = (A, R, f, N, S)$ be an initial DCU graph such that $A = \{a, b, c\}$, $R = \{a, c\}$, f is some distribution of secrets and $N = \{(a, b)\}^R$. In other words, agent c is terminal. Then $r_G = \frac{2}{4} = \frac{1}{2}$ and $\frac{|R|}{|A|} = \frac{2}{3}$.*

$$\begin{tikzcd}
a \arrow[loop above] \arrow[r] & b \arrow[loop above] & c \arrow[loop above]
\end{tikzcd}$$

Example 7.1.5. *Let $G = (A, R, f, N, S)$ be an initial DCU graph such that $A = \{a, b, c\}$, $R = \{a, c\}$, f is some distribution of secrets and $N = \{(a, b), (b, c)\}^R$. Then $r_G = \frac{2}{5}$ and $\frac{|R|}{|A|} = \frac{2}{3}$.*

$$\begin{tikzcd}
a \arrow[loop above] \arrow[r] & b \arrow[loop above] \arrow[r] & c \arrow[loop above]
\end{tikzcd}$$

Even though in these examples $\frac{|R|}{|A|}$ is equivalent, r_G is not. This shows the importance of reliable connections over the amount of unreliability in a network.

It may be clear that in a network with a rate closer to 1 the chance of the network becoming complete is greater than in a network with a rate closer to 0. This is not necessarily true for the rate of an agent because this does not include the type of the agents that can contact agent a. It might as well be that there is an unreliable agent b that is not in agent a's contact list whereas agent a is in the contact list of agent b. Even worse, this might be the case for all such agents b.

The rate of an agent is a good start for investigating the position of an agent in the network. However, we need another measure for the degree of control agents have over the network. Intuitively, agents with a more central position in the network have considerably more opportunities to influence the information flow. This is because information flows primarily over the shortest paths between agents [11]. In graph theory, this measure is captured by the notion of *betweenness centrality* of a node. Freeman [11] gave the first formal definition of betweenness centrality of a node v as a measure of the amount of shortest paths between any two nodes in the network that pass through node v relative to all shortest paths between these two nodes. A high betweenness centrality score now indicates that an agent lies on a substantial fraction of shortest paths connecting other agents. In [11] it is proved that the maximum value for the betweenness centrality is achieved only by the central node in a star graph. This is because in a star graph every path between two nodes passes through the central node. Of course in our framework the situation may change when the network evolves because of the dynamic component.

Connecting to this work, the rate and betweenness centrality of an agent may provide useful information about the way in which the network evolves. Additionally, it would be interesting to compare how the rate changes in a network G with a fixed call sequence σ with different agent types. Such comparisons may provide insights into the differences between the different unreliable agents.

Research Questions:

- *What insights can agent- and network rates and the betweenness centrality provide us about the different kinds of unreliability and protocols considered? Is this indeed a valid way of measuring unreliability in a network?*
- *Are there thresholds for the network rate that ensure (strong) completeness? That is, what is the minimum amount of reliability connections compared*

to unreliable connections we need in order to identify the unreliable agents? How does this depend on different agent types?

7.1.2 Security protocols

We have developed an intuition about what happens to a dynamic communication network when unreliability is introduced. In addition, we have explored the protocol LNSR and the property of being (strong) Trumpian. Seemingly paradoxical, within this exploration we came to the conclusion that this property is not suitable as an active measure against the unreliable agents from the perspective of identification. The reason for this is two-fold: on the one hand, because there are situations in which all the unreliable agents are isolated, yet remain unidentified, on the other hand because unreliable agents that do not actively initiate calls are harder to identify than those that do. We therefore need to consider different measures to ensure the reliability of information and identification of unreliable sources.

In this section, we connect our work to the field of information theory and particularly to the field that studies error-correcting codes. Research into this connection can help develop security protocols for dynamic communication networks.

Channels

There is a clear analogy between our framework and the field that studies errors in electronic channels. Such a channel can be viewed as a network with two parties, a and b, such that there is a one-way connection from a to b, aNb. Even though these channels are not considered to be dynamic, they are well-studied—in particular the errors that might occur on them. This is because in such channels errors do in fact occur that are due to the failure of parts of the set-up.

Formally, a channel is a tuple consisting of an input set $\mathcal{X}$, an output set $\mathcal{Y}$ and a probability distribution $P_{Y|X}$ that describes the output $y \in Y$ in terms of the input $x \in X$.

Definition 7.1.6. [Discrete Channel] *A* (discrete) channel *is a tuple* $(\mathcal{X}; P_{Y|X}; \mathcal{Y})$ *such that* $\mathcal{X}$ *and* $\mathcal{Y}$ *are finite sets and* $P_{Y|X} : \mathcal{Y} \to [0,1]$ *is a probability distribution.* $\mathcal{X}$ *represents the set of possible inputs,* $\mathcal{Y}$ *the set of possible outputs, and* $P_{Y|X}(y|x)$ *is the probability of receiving output* $y \in \mathcal{Y}$ *given input* $x \in \mathcal{X}$.

Note that if we are given a distribution P_X for the input set $\mathcal{X}$ and the conditional distribution $P_{Y|X}$, we can find the distribution P_Y for the output set $\mathcal{Y}$ by marginalization:

$$P_Y(y) = \sum_{x \in \mathcal{X}} P_{Y|X}(y|x) \cdot P_X(x)$$

A well-known example of such a noisy channel is the *binary symmetric channel* (BSC). In the BSC, a message bit $m \in [0,1]$ is sent from the sender a

to the receiver b but is received *"flipped"* with a certain probability f. Flipped means that if a sends the message 0, b will receive the message 1 and vice versa. This means that b will either receive the correct message m with probability $1 - f$, or will receive the flipped message $1 - m$ with probability f.

Definition 7.1.7. [Binary Symmetric Channel (BSC)] *A* binary symmetric channel *with parameter $f \in [0, 1/2]$ is a discrete channel defined by $\mathcal{X} = \mathcal{Y} = \{0, 1\}$ and*

$$P_{Y|X}(0|0) = P_{Y|X}(1|1) = 1 - f$$
$$P_{Y|X}(0|1) = P_{Y|X}(1|0) = f$$

Note that the binary symmetric channel is memory-less, meaning that the probability distribution of the output only depends on the current input. We say that a channel is *deterministic* if for all x in $\mathcal{X}$ there exists a y in $\mathcal{Y}$ such that $P_{Y|X}(y|x) = 1$ and a channel is *lossless* if for all y in $\mathcal{Y}$ there exists a unique x in $\mathcal{X}$ such that $P_{Y|X}(y|x) > 0$. So in a deterministic channel, the output is completely determined by the input and in a lossless channel the input is completely determined by the output. A *noiseless* channel is a channel that is both deterministic and lossless. Recall that the binary symmetric channel is neither deterministic nor noiseless.

Error-correcting codes

To equip against errors in such channels one can either change the set-up or introduce what is called an error-correcting code. Consider for example the binary symmetric channel with an error probability of $f = 0.25$ and consider that a sends b the message 000. Now the chance that b will receive a message consisting of at least two bits that are 1 is $3(1 - f)f^2 + f^3 = 0.03125 = \frac{1}{32}$. In other words, this chance is rather small. This is an example of the n-bit repetition code R_n, namely R_3. Formally, a code is a procedure for encoding and decoding a message.

Definition 7.1.8. [Code] *Let $M, n \in \mathbb{N}$. An (M, n)-code for the channel $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$ consists of*

- *An index set $[M] = \{1, \ldots, M\}$ representing the set of possible messages*
- *An injective encoding function $enc : [M] \to \mathcal{X}^n$, where n denotes the number of channel uses for a single message*
- *A decoding function $dec : \mathcal{Y}^n \to [M]$*

Where the set of all codewords, $\{enc(1), \ldots, enc(M)\}$ is called the codebook.

An example of this is the repetition code. In the n-bit repetition code, the sender a sends the message m precisely n times. The receiver b then decodes the message by means of majority voting.

Definition 7.1.9. [Repetition Code] *The n-bit* repetition code R_n *works as follows: a single message bit m is encoded by simply repeating the bit n times (i.e. $enc(m) = mm \dots m$). Decoding is done by the majority vote (i.e. $dec(y) = MAJ(y_1, \dots, y_n)$) which is 1 if and only if (strictly) more than half of the bits in y are 1s. Usually repetition codes require n to be odd to avoid ties in the decoding.*

In the example in the beginning of this section where $f = 0.25$, $m = 0$ and the repetition code R_3 is applied, agent b will successfully decode the message if at most one bit was flipped by the channel. It may be clear that such codes have a certain trade-off between preserving correctness and efficiency. The above example illustrates this: a small error (namely if one out of three bits are flipped) can be corrected at the cost of sending two extra bits. The number of bits of information that are transmitted per channel use is captured by the rate of a code.

Definition 7.1.10. [Rate of a Code] *The* rate *of an (M, n)-code is defined as*

$$R := \frac{\log M}{n}$$

The n-bit repetition code R_n is a $(2, n)$ code with a rate of $1/n$. I.e. one bit of information can be communicated at the cost of n channel uses. Therefore the rate of a code captures the cost of preserving correctness, where correctness is captured by the average or maximal probability of error during the decoding.

Definition 7.1.11. [Probability of Error] *Given an (M, n) code for a channel $(\mathcal{X}, P_{Y|X}, \mathcal{Y})$, the* probability of error *λ_m is the probability that the decoded output is not equal to the input message m. Formally,*

$$\lambda_m := P\left[dec(Y^n) \neq m \mid X^n = enc(m)\right]$$

The maximal probability of error *is defined as $\lambda := \max_{m \in [M]} \lambda_m$ and similarly the* average probability of error *is defined as $p_e := \frac{1}{M} \sum_{m=1}^{M} \lambda_m$.*

We have already calculated the probability of error for the message $m = 0$ for the repetition code R_3 to be $3(1-f)f^2 + f^3$. More generally, the n-bit repetition code R_n is a $(2, n)$ code with an average/maximal probability of error of

$$\sum_{k=(n+1)/2}^{n} \binom{n}{k} f^k (1-f)^{n-k}$$

when used on a binary symmetric channel with bit flip probability f. There are other error-correcting codes that perform better in terms of the amount of channel uses while preserving the probability of error. An example is the $(2^4, 7)$ Hamming code.

Security protocols for DCU networks

Because of the result in this thesis that the property (strong) Trumpian is not a suitable measure from the perspective of identification in the battle against false information, a logical next step is to investigate how the notions of information theory introduced here are related to communication protocols. Although this may depend on the type of unreliability occurring in the network, we are interested in how ideas about security can be translated to dynamic networks with unreliability. In particular, this is of interest from a knowledge-based perspective: what security measures can the reliable agents individually take based on their knowledge of the network? That is, what measures can the agents take based only on the information they possess about the network—the information they have about phone numbers and secrets of other agents.

First, however, we need to explain why the identification of unreliable agents is important and why therefore deleting and blocking unreliable agents is not the best approach against unreliable sources. It is obvious that whenever an unreliable agent is not identified by someone, she will consider the information by the unreliable agent truthful. Then, if she does not learn about the unreliability of that agent, she will end up having wrong beliefs about the secret of that agent—or, when considering different agent types, worse. It may now be clear why we are interested in security measures aiming at the identification of unreliability.

In the search for such security measures, clearly the knowledge-based aspect of DCU networks complicates the setting. Consider the n-bit *repetition code* R_n discussed before in which a message m is send n times over a channel. A version of this error-correcting code on DCU networks is for instance to execute a successful (in the complete sense) call sequence multiple times. This may indeed help identify the unreliable agents. Because if now one of the reliable agents identifies an unreliable agent after one round of the call sequence, clearly after executing the call sequence twice, everybody will learn that this agent is unreliable.

Conjecture 7.1.12. *Let $G = (A, R, f, N, S)$ be an initial DCU graph, $A \setminus R \neq \emptyset$ and σ a successful call sequence in the complete sense. That is G^σ is complete. Then the unreliable agents b that are identified by some agents $a \neq b$ in G^σ (i.e. there is at least one agent $a \in R$ such that $b \in Z_a$ and $a \neq b$), will be identified by all the agents in $G^{\sigma;\sigma}$. As a consequence, if all the unreliable agents are identified by some agent in G^σ, then $G^{\sigma;\sigma}$ is strong complete.*

An idea to prove Conjecture 7.1.12 is by observing that a call sequence σ is successful in the complete sense on an initial DCU G if everyone knows the secrets of the reliable agents. That is, for each $a \in A$ and each $b \in R$ it must be that $b \in S_a$. So when σ is executed twice, it needs to be that the other information available to agent b (i.e. X_b, Y_b and Z_b) will also be spread throughout the network. Thus when all unreliable agents are identified by some reliable agent, this information will spread throughout the network and become available to all the agents. As a result, the network becomes strong complete.

The approach of the n-bit repetition code seems successful in identifying the unreliable agents. However, it raises the immediate question how executing a call sequence more than once can be practically implemented into our framework. Agents act knowledge-based and therefore executing a call sequence more than once requires the agents having knowledge about the order of calls in the first execution of σ and about the moment this first execution is finished. Thus this is not a very feasible option, even when more information about the calls taking place in the network can be observed and is stored by the agents.

An easier version of such a security protocol is the direct application of the repetition code by immediately repeating each call ab taking place in the network n times: $ab; ab; \ldots ; ab$, or an extension of an available protocol (for example ANY, CMO or LNS) such that whenever an agent a concludes that agent b is unreliable (i.e. $b \in Z_a$), agent a calls each $c \in N_a \setminus Z_a$ to update c on her findings. More specifically, whenever in a call ab we have that $Z_a^{ab} \neq Z_a$, agent a will make calls to agents $c \in N_a^{ab} \setminus Z_a^{ab}$ with the information $Z_a^{ab} \setminus Z_a$. We call LNSCU the extension of LNS in this way. Note that she does not need to communicate Z_a because by the protocol LNSCU, she will already have done so. Clearly with this extension, the knowledge of unreliable agents spreads quickly. However, this approach is not very efficient because agents in our setting do not keep track of what they have told whom. Therefore, there will be a great amount of redundant calls, nor is it clear that this procedure ends. Namely when agent a updates agent c about the unreliability of agent b, then agent c will continue spreading this message and because she has not tracked that she heard this from agent a, she will call agent a again with this information, etc. This is a great cost in terms of efficiency.

The direct application of R_n might be more successful. This means that each individual call is immediately repeated n times and a valid call sequence may look like $ab; ab; \ldots ; ab; cd; cd; \ldots ; cd; \ldots$. With alternating bluffers this approach works for any $n \geq 2$. However, when the unreliable agents are consistent this idea fails—naturally also R_n fails in this case.

Another option for security protocols occurs when we involve a fact-checker in our network. Recall that a fact-checker is an agent that functions the way an *oracle* does in complexity theory, namely providing true information only. As such, we can view the fact-checker as an all-mighty source of information. We have limited the fact-checker in a way that agents can only verify information with the fact-checker and not learn any new information. In addition, we might be interested in a stronger limitation that in each call with the fact-checker the secret of only one agent can be verified. This is in line with the fact that communication, and in particular communication with the fact-checker, might be costly—at least from an efficiency-perspective. In fact, analogously to information theory (i.e. the rate of a channel), there is always a choice to make to what extent security is valued at the cost of efficiency.

With the fact-checker, we can for instance design a protocol that allows the agents to verify the secret of every other agent exactly once. Such a protocol requires the secret relation to be specified into $(X_a, X'_a, Y_a, Y'_a, Z_a)$ where X_a, Y_a and Z_a are as before and $X'_a \subseteq X_a$ and $Y'_a \subseteq Y_a$ are the sets of agents that agent a has verified with the fact-checker that have the secret 1 and 0, respectively. An example is the Call Me Once Check Me Once (CMOCMO) protocol. A similar extension of the protocols ANY, LNS and LNSR can be made.

Definition 7.1.13. [Protocol Call Me Once Check Me Once (CMOCMO)] *The protocol CMOCMO states that whenever $G = (A, R, f, N, S)$ is not complete, we can select any call xy such that $xy \in N$, $x \neq y$ and there was no prior call xy or any call xf where f is a fact-checker and x verifies the secret of an agent $y \in (X_a \setminus X'_a) \cup (Y_a \setminus Y'_a)$.*

It may be questioned whether it is realistic to assume such a fact-checker. We have already touched upon this discussion in Section 3.2.1. And indeed, this assumption seems more of an ideal abstraction that can only be approached in real-life situations. More likely is that agents rate other agents according to their reliability and then verify their information with the agent that they consider most reliable and knowledge-rich. This is what we also do in real-life situations by for instance verifying news with a newspaper that we consider to be very truthful.

The main research question now is to investigate what kind of security protocol finds a good balance between efficiency and security, and whether the situation with a fact-checker in the network can be approached by letting agents classify the other agents according to their reliability. Such protocols are very desirable for many applications.

Additionally, we are interested to know whether the results about censorship actions also apply to different security measures. Namely, if also these security measures impose a choice between preventing the spread of false information and the identification of unreliable sources.

Research Questions:

- *We have given two extensions of the current protocols (LNSCU and CMOCMO) that aim to ensure the identification of unreliable agents. The question now is whether these protocol extensions are sufficient and whether there are more efficient protocols to be discovered.*

- *Censorship actions exemplify that there is a choice between preventing false information to be spread and identifying unreliable sources. Is this true for other security measures as well? Or can we find a best-of-both-worlds solution?*

7.1.3 A different approach to parameters

The framework introduced in this thesis to model dynamic communication with unreliable agents was based on some predefined parameters. We for instance assumed that the network is asynchronous, meaning that agents do not observe calls that are being made in which they are not involved, and we explored two different attitudes of agents: the *presumption of innocence*– and the *see for yourself* attitude. We also say that agents with this first attitude are *naive*.

Additionally, we assumed that in calls only information regarding phone numbers and secrets is shared.

In this section we will approach the parameters differently. We will explore the idea that the parameters are treated as dynamic quantities rather than static or the idea to require agents to share more information in calls.

Dynamic parameters

Treating the parameters as dynamic quantities is motivated by the fact that it may be unrealistic to assume that the parameters do not change while the network evolves. In our framework, agents have a certain type and property and follow a certain, not necessarily the same, protocol. Until now, we have treated these characteristics of agents as static, but what happens if agents change their behavior based on the information available about the network? For instance, if agents find out that all their connections are unreliable, they might decide to start following a different protocol or take on certain properties. In particular, she might decide that she will change her behavior toward a *subset* of the agents in the network—namely toward those agents that she considers unreliable.

These are two ideas: parameters such as agent types, properties or protocols are dynamic so that they can change during the execution of a call sequence, and agents behave differently toward different agents. That is, an agent a in a network $G = (A, R, f, N, S)$ may be described by the way she acts upon another agent b. That means that for each agent $a \in A$, a can be described by a set of types $\{t_a^b\}$, properties $\{prop_a^b\}$ and protocols $\{P_a^b\}$ where t_a^b, $prop_a^b$ and P_a^b denote the type, property and protocol agent a acts out toward agent b

By treating the parameters as dynamic, real-life situations can be better approached. This is because it is quite natural for agents to change their behavior once unreliability is detected. Note though that also here, changing your behavior by taking on the property of being (strong) Trumpian may not be the desirable approach to deal with unreliability with regards to the network as a whole.

In order to account for dynamic parameters, we need to change the data structure accordingly so that each agent is described by the triple $(\{t_a^b\}, \{prop_a^b\}, \{P_a^b\})$. It then remains to define how agents change their behavior, with options ranging from doing so depending on your own information or randomly by means of a coin toss.

Communicated parameters

Another approach to the parameters of dynamic communication is by requiring more information to be shared in calls. In addition to sharing phone numbers and secrets, we can require agents to share agent types, properties and protocols. This allows agents to calculate individually which calls are likely to take place based on their knowledge about the network. If in addition we let agents communicate the calls they have been involved in in the past and the calls that they heard to have taken place, we create a situation in which the agents "gossip about gossip". That is, the subject to the gossip is the gossip itself. Intuitively, this will make it much easier for agents to observe unreliable agents. This is because when all information about calls and the information shared in these calls is shared, each agent can individually determine the sets Z_a of agents that are considered unreliable by an agent a. We will elaborate on "gossip about gossip" in Section 7.2.4 where we discuss the application of this idea as an alternative to blockchain technology.

Requiring agents to share more information is a very realistic requirement. In communication, we might namely want more information than just phone numbers and secrets in order to decide whether another agent is reliable. In particular, we could let agents decide themselves what information they want to share.

Generally, we expect that storing more information about the network will make it harder for agents to be unreliable without being detected. Therefore this approach is also interesting to investigate from a security point of view.

Research Questions:

- *Are there easy adaptations of our framework so that the parameters can be treated dynamically or more information is required to be shared?*
- *In our framework we consider that agents only share information about phone numbers and secrets. The question is now whether including more information to be shared makes it harder for the unreliable agents to remain uncovered. Or do then more sophisticated unreliable agents emerge?*

7.2 Applications

In this section, we explore some applications of our framework. These range from unreliability in the form of error or noise in electronic network, to identifying fake news and "gossip about gossip" as an alternative to blockchain technology.

7.2.1 Noise in electronic networks

The framework in this thesis was designed for dynamic communication networks in which unreliability occurs in the form of error or noise. An obvious direct application of this are electronic failures in systems of electronics. In such systems, noise occurs as an unwanted random disturbance. Think for example of a network of sensors (light sensors, movement sensors, temperature sensors, etc.) that have to decide together on a joint task (turning the light on or off). If now an error occurs on the communication channel or one of the sensors in the network is faulty, the question can be raised whether the other sensors can, by pure means of communication, identify this error and act accordingly. For instance by ignoring the information of the faulty sensor in the decision making for the task.

In this setting it is clear what the benefits are from studying dynamic communication where the assumption that *everybody is reliable* is dropped.

Namely to investigate whether unreliability can be detected without the inference of a user.

Here the result that unreliable sources can be easier uncovered when they initiate communication is relevant. This is because it shows that faulty sensors that in addition to sending wrong signals have lost their power to send signals to other sensors are more critical than just faulty sensors that send wrong information.

7.2.2 Identifying fake news

Another application of this framework is the algorithmic approach to the identification of fake news. With the rise of technology and the increasing influence of the internet over the past 30 years, a major shift has occurred in the area of news and information provision. It is nowadays easier than ever to obtain and spread information. The downside of this is that it is just as easy to obtain and spread *false* information. As a result, validity and integrity of information and sources have become of major significance.

Recently, there has been a great discussion about this topic, possibly having influenced the US presidential election results [2]. The problem with fake news is two-fold: the ease of generating and spreading fake news, and the difficulties of identifying the news as fake. To tackle the latter, social media are starting to work together with newspapers and news websites to check the validity of information and sources [19, 33]. For instance, Facebook users now have the option of labeling news sources and articles as potentially "fake" to have them examined on their reliability by a group of news-experts. Similar measures are taken by Google and it is a matter of time until other tech-giants will follow. In our framework, this corresponds to introducing a fact-checker in the network.

However, these actions do not come close in solving the problem because of their inefficiency and dependency on human resources. Ideally, computer algorithms would take care of the task of identifying and possibly warding off fake news.

Our framework provides a starting point for such algorithms that is based on using communication as a means of security. This can for instance be realized by considering news platforms as agents with their articles as the information available (like secrets) and their sources as agents they communicate with.

In this thesis, we have shown that, seemingly paradoxical, unreliable agents that do not initiate communication are harder to identify than those that do. This means that false information that is actively spread throughout the network can be easier detected. Additionally, we have considered the censorship actions delete and block that correspond to (strong) Trumpian agents as a means of security. With this we have shown that by isolating the unreliable agents, it becomes harder to successfully identify them. This result raises questions for the kind of security measures taken because there are many reasons why letting unreliable sources spread their word is not suitable in the battle against fake news.

Still, there are also disadvantages to failing to identify the unreliable sources. This is because when information is learned from an unreliable source without being able to identify this source as unreliable, this information will be considered true. It is a philosophical question how this counter-intuitive result weights against the reasons for preventing false information to spread. Our contribution to this question is that it might be hard to achieve both prevention and identification—at least with the current framework. Further research is much needed to provide a suitable answer to this question.

7.2.3 Epidemiology and herd immunity

There is also a close connection between dynamic communication and the field of epidemiology. The spreading of information in a network namely provides a good estimate for the propagation of an epidemic throughout groups of individuals [10, 22, 23] . Because of this connection graph theoretical properties have already been studied with regards to their effect on the spread of an epidemic. For instance how graph theoretical properties of a network determine the persistence of an epidemic by either impeding or facilitating its spread and maintenance [12]. This provides conceptual results about the transmission characteristics of infectious diseases.

A great interest in the study of epidemiology lies in investigating to what extent a population needs to be vaccinated to contain infectious diseases. This is also known as *herd immunity.* Herd immunity is an indirect form of protection against infectious diseases. It occurs when a large percentage of the population is immune to the disease so that the introduction of one infective person into the population does not cause an invasion. For different diseases the immune fraction of the population has been identified [16].

In this setting, our ideas of unreliability can be implemented in such networks to further investigate herd immunity. Yet, this does require a change of our data structure: instead of agents having an individual secret, we now want to consider a joint secret that represents the disease. Unreliability can now occur in the form of not being able to spread the secret, i.e. an agent is immune.

So why is this different from the current graph theoretical approach to epidemiology and herd immunity? Because our framework considers dynamic communication. That is, the agents one can come in contact with may expand. In addition to this providing a realistic approach to spreading diseases, this allows for more aspects to be studied. For instance, this allows to investigate the specific effects of isolating the infected agents.

7.2.4 Gossip about gossip

In Section 7.1.3, we have already touched on the idea of “gossip about gossip”. This means that the information shared in the network is the gossip itself. So in addition to secrets and phone numbers, agents share the calls in which they and other agents have been involved.

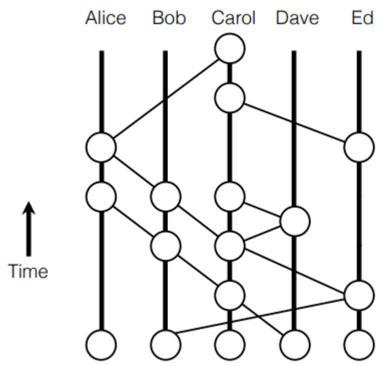


Figure 7.1: Gossip history represented by a directed graph where each member is one column of vertices. When Alice receives gossip from Bob (telling her everything he knows), that event is represented by a vertex in Alice's column with two edges going downward to the immediate-preceding gossip events by Alice and Bob. Figure taken from [6].

Recently, this idea has been studied in combination with hashgraphs as an alternative to the role of blockchain technology in cryptocurrencies as Bitcoin [6, 25][1]. A *hashgraph* is a directed acyclic graph where each vertex contains the hashes of the events below it. To illustrate this, see Figure 7.1.

Because the subject of the gossip is now the gossip itself, a virtual voting system can be used to reach consensus. For instance to reach consensus about the set of unreliable agents. This is because each member of the network can decide for herself how another agent is supposed to vote in round n, given the information available in round $n-1$. Specifically, each member of the network can compute the set Z_a of agents considered unreliable by agent a, for each agent a.

As a result, it becomes harder to cheat because the voting is done virtually on everyone else's computer. This is where gossip about gossip in combination with hashgraphs provides an advantage over blockchain technology. In contrast to blockchain, it is proved in [6] that hashgraphs do have Byzantine fault tolerance. In the proof it is assumed that at most one third of the network is cheating—i.e. are unreliable. In [24] is it shown that this is the best threshold we can get for the fraction of unreliable agents. That this framework has byzantine fault tolerance means that eventually all the reliable members will know the exact ordering of, say, the first 100 transactions and in particular, they will reach a state in which they know that they know it. So agents actually know when it is guaranteed that consensus has been achieved. Compare this to the case of Bitcoin. With Bitcoin the probability of reaching consensus grows after each confirmation and as a result participants can decide that after a certain amount

[1]Note that these articles are not peer reviewed and based on ideas of the company "Swirlds". See also their website: `http://www.swirlds.com/`.

of confirmation they are "sure enough"—even though mathematically speaking they never are. It may be clear that Byzantine fault tolerance is an desirable property for such systems.

Another advantage of gossip about gossip over blockchain technology is that no proof of work is required. In Bitcoin, a lot of computational power, and thus time and energy, is required to mine a block even though there is no assurance that this block is chosen for the extension of the chain. It might namely be that another block is mined around the same time and the community chooses for this other block, disregarding yours. This is a waste and additionally provides uncertainty to participants about transactions they want to make. In hashgraph, whatever transaction a participant wishes to make, as soon as the participant starts to gossip it, it will definitely become part of the permanent record. Even though a participant may not immediately know what position the transaction will have in time, she will be guaranteed that the transaction will be part of the history [6].

The connection between these ideas and dynamic gossip is clear because also here we are interested in communication protocols that are efficient, yet secure. Changing the data structure of our framework to allow for the spreading of the gossip itself by means of gossiping is required to analyze to what extent studying DCU networks can help for this application.

Chapter 8

Conclusion

In this thesis, we have extended the work on dynamic communication done in [29, 30] to account for unreliability. The motivation behind this extension is the question whether the assumption that *everybody is reliable* is realistic for any kind of practical application, where error and noise might, and does, occasionally occur.

To account for this unreliability, we have broadened the current framework by introducing a quantification of information as either true or false. This is formalized by considering the shared information (secrets) as bits. An unreliable agent now is an agent that does not act the same way reliable agents do: with options ranging from lying about one's own secret to manipulating the network by cutting connections between agents. Including this quantification of information creates the need for a different kind of completeness. Next to learning all the secrets (of the reliable agents), we are now also interested in identifying the unreliable agents. This happens whenever agents receive conflicting information about this agent's secret.

Within our framework, we have focused on a relatively simple type of unreliable agent: the alternating bluffer. This is an agent that lies about her own secret in every second call she is involved in. Investigating this agent provides a good first way of approaching the study of networks in which unreliability occurs unintentionally in the form of error or noise—while still remaining formally precise and feasible.

We showed that already by introducing this agent we cannot simply extend the results about the success of LNS on DC graphs. In fact, already with this small amount of error or noise, the results fail. Specifically, we proved that LNS is not strongly successful on DCU graphs that are sun graphs where all the unreliable agents were terminal alternating bluffers. This was one of the known results for networks in which everyone is reliable. The failure of extending this result to the case of terminal alternating bluffers exemplifies that it is going to be difficult to prove general results about DCU graphs. Yet, more importantly, this result emphasizes the need for discarding the assumption on which this result is based (i.e. that *everybody is reliable*) for any kind of practical application.

Next, we explored an additional restriction on the unreliable agents: the protocol LNS that only allows reliable agents to initiate a call. We proved that unreliable agents that do not initiate the spread false information in the network are harder to identify than unreliable agents that do so. This seems paradoxical because these unreliable agents create less havoc in the network. Yet, it turns out that this is precisely why they are harder to identify: there is less spread of false, and thus contrary, information. This result has counter-intuitive consequences for, among others, the kind of security measures that should be taken in the battle against fake news. Namely that individually blocking or cutting out the agents that are considered unreliable might have the contrary effect: the rest of the network might then not learn about their unreliability. And indeed, this is confirmed by our result that there are networks in which the unreliable agents are isolated by means of the security measure that all the reliable agents have the strong Trumpian property (i.e. each reliable agent acts out the censorship action delete), but in which the unreliable agents are not successfully identified—even though the reverse direction does hold. From a practical point of view, this raises interesting questions because there are many reasons why letting fake information spread may not be a desirable solution. Further research is required to shed light on these issues.

By putting unreliability in the spotlight, this thesis has taken the research on dynamic communication in the direction of practical applications. These range from the identification of fake news to noise in electronic networks and herd immunity. In addition, we have explored recent investigations to using "gossip about gossip" as an alternative for blockchain technology in cryptocurrencies. Our framework might help provide further insights into preventing unreliable agents attacking such a system.

In conclusion, our framework takes a turn from the idealized assumption that *everybody is reliable* and shows that already with a small adaptation such as the alternating bluffer, the known result about dynamic communication fail to hold. In addition, we have shown that security measures against the spreading of false information may come at a cost concerning the identification of the unreliable source. With this we have concluded an exploration in the field of dynamic communication with unreliability and have offered a starting point for much needed future research.

Bibliography

[1] Thomas Ågotnes, Hans van Ditmarsch, and Yanjing Wang. True lies. *Synthese*, pages 1–35, 2016.

[2] Hunt Allcott and Matthew Gentzkow. Social media and fake news in the 2016 election. Technical report, National Bureau of Economic Research, 2017.

[3] Krzysztof R Apt, Davide Grossi, and Wiebe van der Hoek. Epistemic protocols for distributed gossiping. *arXiv preprint arXiv:1606.07516*, 2016.

[4] Maduka Attamah, Hans Van Ditmarsch, Davide Grossi, and Wiebe van der Hoek. Knowledge and gossip. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pages 21–26. IOS Press, 2014.

[5] Maduka Attamah, Hans van Ditmarsch, Davide Grossi, and Wiebe van der Hoek. The pleasure of gossip. In *Rohit Parikh on Logic, Language and Society*, pages 145–163. Springer, 2017.

[6] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. 2016.

[7] Alexandru Baltag, Nina Gierasimczuk, and Sonja Smets. Belief revision as a truth-tracking process. In *Proceedings of the 13th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 187–190. ACM, 2011.

[8] Thomas J. Watson IBM Research Center. Research Division and MC Golumbic. *The general gossip problem.* 1974.

[9] Kees Doets and Jan van Eijck. The haskell road to logic, math and programming, 2004.

[10] Patrick T Eugster, Rachid Guerraoui, A-M Kermarrec, and Laurent Massoulié. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, 2004.

[11] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[12] Ayalvadi Ganesh, Laurent Massoulié, and Don Towsley. The effect of network topology on the spread of epidemics. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 2, pages 1455–1466. IEEE, 2005.

[13] Nina Gierasimczuk. *Knowing one's limits: logical analysis of inductive inference*. Institute for Logic, Language and Computation, 2010.

[14] Frank Harary and Allen J Schwenk. The communication problem on graphs and digraphs. *Journal of the Franklin Institute*, 297(6):491–495, 1974.

[15] Sandra M Hedetniemi, Stephen T Hedetniemi, and Arthur L Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[16] Herbert W Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.

[17] Daniel Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[18] Donald Ervin Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[19] Jeroen Kraan and Floris Poort. Facebook start ook in Nederland met aanpak nepnieuws. `http://www.nu.nl/internet/4506750/facebook-start-in-nederland-met-aanpak-nepnieuws.html`, 2017.

[20] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[21] Faustine Maffre. *Ignorance is bliss: observability-based dynamic epistemic logics and their applications*. PhD thesis, Université Paul Sabatier-Toulouse III, 2016.

[22] Mark EJ Newman. Spread of epidemic disease on networks. *Physical review E*, 66(1):016128, 2002.

[23] Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Physical review letters*, 86(14):3200, 2001.

[24] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[25] SAMMANTICS. A new approach to consensus: Swirlds hashgraph. `http://sammantics.com/blog/2016/7/27/hashgraph-consensus`, 2016.

[26] Robert Tijdeman. On a telephone problem. *Nieuw Archief voor Wiskunde*, 3(19):188–192, 1971.

[27] Johan van Benthem. An essay on sabotage and obstruction. In *Mechanizing Mathematical Reasoning*, pages 268–276. Springer, 2005.

[28] Hans van Ditmarsch, Davide Grossi, Andreas Herzig, Wiebe van der Hoek, and Louwe B Kuijer. Parameters for epistemic gossip problems. *Proc. LOFT 2016*, 2016.

[29] Hans van Ditmarsch, Jan van Eijck, Pere Pardo, Rahim Ramezanian, and François Schwarzentruber. Dynamic gossip. *arXiv preprint arXiv:1511.00867*, 2015.

[30] Hans van Ditmarsch, Jan van Eijck, Pere Pardo, Rahim Ramezanian, François Schwarzentruber, and Liber Amicorum Alberti. Gossip in dynamic networks.

[31] Hans van Ditmarsch, Jan van Eijck, Floor Sietsma, and Yanjing Wang. On the logic of lying. *Games, actions and social software*, 7010:41–72, 2012.

[32] Jan van Eijck and Malvin Gattinger. Gossip. Technical report, CWI, Amsterdam, 2015.

[33] Laurens Verhagen. Facebook pakt nepnieuws nu ook in Nederland aan. `http://www.volkskrant.nl/tech/facebook-pakt-nepnieuws-nu-ook-in-nederland-aan~a4469279/`, 2017.

[34] Jana Wagemaker. Gossip in NetKAT. Master's thesis, University of Amsterdam, The Netherlands, 2017.

[35] Duncan J Watts and Steven H Strogatz. Collective dynamics of'small-world'networks. *nature*, 393(6684):440, 1998.

[36] Richard J Williams and Neo D Martinez. Simple rules yield complex food webs. *Nature*, 404(6774):180, 2000.