Distributed Computation on Unreliable Radio Channels

by

Calvin Newport

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

Author .....
Department of Electrical Engineering and Computer Science
August 7, 2009

Certified by .....
Nancy Lynch
NEC Professor of Software Science and Engineering
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Distributed Computation on Unreliable Radio Channels

by

Calvin Newport

Submitted to the Department of Electrical Engineering and Computer Science
on August 7, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

An important topic in wireless networking is the development of reliable algorithms for environments suffering from *adversarial interference*. This term captures any type of channel disruption outside the control of the algorithm designer—from contention with unrelated devices to malicious jamming. In this thesis, we provide four contributions toward a comprehensive theoretical treatment of this topic.

First, we detail a formal modeling framework. This framework is general enough to describe almost any radio network studied to date in the theory literature. It can also precisely capture the often subtle details of adversarial behavior. In addition, we prove a pair of composition results that allow a *layered* strategy for designing radio network algorithms The results can be used to combine an algorithm designed for a powerful channel with an implementation of this channel on a less powerful channel.

Next, we formalize adversarial interference with the definition of the *t-disrupted channel*. We then define the more powerful (t, b, p)-*feedback channel*, and provide both a randomized and deterministic implementation of the latter using the former. To emphasize the utility of this layered approach, we provide solutions to the *set agreement*, *gossip*, and *reliable broadcast* problems using the powerful feedback channel. Combined with the implementation algorithms and composition results, this automatically generates solutions to these problems for the less powerful, but more realistic, t-disrupted channel.

Finally, we define a variant of the modeling framework that captures the attributes of an *ad hoc* network, including asynchronous starts and the lack of advance knowledge of participating devices. Within this new framework, we solve the *wireless synchronization problem* on a t-disrupted channel. This problem requires devices to agree on a common round numbering scheme. We conclude by discussing how to use such a solution to adapt algorithms designed for the original model to work in the ad hoc variant.

Thesis Supervisor: Nancy Lynch
Title: NEC Professor of Software Science and Engineering

Acknowledgments

First and foremost, I thank my advisor, **Nancy Lynch**, not only for her tireless work on this thesis, but for her longterm efforts, spanning my five years at MIT, to help mold me in my emergent role as an academic researcher. From her, I learned the value of precision, clear communication of ideas, and taking the extra time required to push work from *good enough* to *excellent.* I also thank both **Rachid Guerraoui** and **Sam Madden** for sitting on my thesis committee and providing invaluable feedback on how to polish this work.

My thanks to Rachid, however, goes beyond his efforts on my committee. Since his visit to MIT during the academic year of 2005 to 2006, he has been become one of my main collaborators and mentors. From him I learned the importance of identifying problems that are *interesting*, and then taking the time to convey this interestingness to the reader. In terms of graduate student collaborators, I must extend similar appreciation to **Seth Gilbert**, with whom I have worked closely on many of the results inspiring this thesis, as well as on quite a few other unrelated papers. To this day, I maintain a *Seth projects* list on my computer that grows ever longer with potential future collaborations.

I have also benefitted greatly from my interactions with the various postdocs that have passed through the theory of distributed systems group, especially **Gregory Chockler**, **Murat Demirbas**, **Ling Cheung**, and **Fabian Kuhn**, all of whom with I have had the honor of coauthoring papers. I must, of course, also thank **Shlomi Dolev** and **Darek Kowlaski**, who were important collaborators on many of the t-disrupted channel papers that helped motivate this thesis.

Finally, I conclude by thanking my most important collaborator of all, my wife Julie, who tolerated a half-decade of my semi-coherent rants on radio networks and disruptive adversaries, and gave me the support necessary to see the ideas through to something complete.

Contents

List of Figures

Chapter 1

Introduction

The increasing importance of wireless networking is beyond dispute. Long-range cellular links bring data packets to billions of mobile phones. Medium-range technologies, such as the 802.11 family of protocols [1], introduced high-speed wireless networking to millions of homes and offices. And short-range protocols, such as Bluetooth [11] and Zigbee [5], eliminate the need for wires in connecting nearby devices. To name just a few examples. David Leeper, the chief technical officer for wireless technologies at Intel, described this trend as follows:

> Economic forces and physical laws are driving the growth of a new wireless infrastructure that will become as ubiquitous as lighting and power infrastructure are today [66].

With any rapidly-growing technology trend, however, comes new issues to be addressed. In this thesis, we carve out and tackle one of these issues from among the many facing our community: the design and analysis of reliable protocols in unreliable radio network settings.

The Trouble with Open Airwaves. In a 2007 paper [43], a research team led by Ramakrishna Gummadi asked a simple question: *What is the impact of RF interference on the increasingly crowded unlicensed bands of the radio spectrum?* They setup an 802.11 access point and connected a laptop client. They then introduced several common sources of radio interference, including a Zigbee node, cordless phone, and two different types of malicious jammers. They found that even "relatively small amounts" of RF interference can result in "substantial performance problems for commodity 802.11 NICs." They were able, for example, to disrupt a link with a signal 1000 times weaker than the 802.11 signal, and shut down a network running multiple access points on multiple channels, using only a single interferer. Changes to standard operational parameters, such as CCA threshold, bit rate, and packet size, could not eliminate these effects.

This result underscores an important reality: much of the future of wireless computing will unfold in the unlicensed bands of the radio spectrum. Furthermore, these bands are increasingly crowded and vulnerable. Designers of protocols that use the unlicensed bands must take into account a variety of interference sources, including:

- Selfish devices that use the band without consideration of other protocols.
- Malicious devices that actively try to prevent communication.
- Electromagnetic interference from unrelated sources (e.g., military radar, microwaves, and medical equipment).

For simplicity, we will refer to this diversity of disruption with the generic term *adversarial interference*, because its behavior is unpredictable and falls outside of the control of the individual protocol designer.

The Gap Between Systems and Theory. The systems community has been attacking adversarial interference from multiple angles, including *hardware solutions* (e.g., more resilient, spread spectrum-style signal encoding [3, 78]), *regulatory solutions* (e.g., additional rules for the proper use of the relevant bands [76]), and *systematic solutions* based on new approaches to wireless networking (e.g., cognitive radios [69, 64, 62, 63, 4]).

There has also been recent interest in the problem of jamming at the link layer and above; c.f., [13, 71, 48, 44], which analyze specific link and network layer protocols, highlighting semantic vulnerabilities that can be leveraged to gain increased jamming efficiency. As mentioned, improved spread spectrum techniques, such as FHSS and DSSS, are designed with unpredictable interference in mind [3, 78]. Frequency-hopping schemes are also used to circumvent such disruption [80, 81, 11]. And in the sensor network community, a growing body of work addresses the detection (and, perhaps, subsequent destruction) of jamming devices [83, 84, 67].

The theoretical distributed algorithms community, by contrast, lags in their study of this increasingly important topic. As detailed in Section 1.1, although theoreticians have studied distributed algorithms for radio networks since the 1970's, the first papers to address *adversarial* behavior in these networks did not appear until 2004. The community currently lacks the theoretical models that allow a formal study of what can and cannot be solved in a radio network model suffering from adversarial interference. And as the history of distributed algorithms has taught us, the mathematical realities of such questions have a way of defying our intuition (e.g., the FLP result which proved the impossibility of consensus in an asynchronous system with one failure [34]).

Reducing the Gap. In this thesis, we aim to reduce this gap between theory and practice. A detailed description of its contributions can be found in Section 1.2. Here we provide only a brief overview of its results.

First, we describe a formal modeling framework for the study of distributed algorithms in radio networks. This framework allows for a precise, probabilistic, automaton-based description of radio channels and algorithm behavior. It includes formal notions of *problems*, *solving problems*, and *implementing one channel using another channel*. It also includes a pair of composition results that facilitate the combination of multiple simple components toward the goal of solving complex problems.

Using this framework, we formalize the *t-disrupted radio channel*, which captures a single-hop radio network suffering from the the type of adversarial interference described above. We first introduced this model in 2007 [31], and it has since become a popular topic of study [32, 81, 40, 30, 68, 80, 74]. The t-disrupted channel provides the devices access to a collection of $\mathcal{F} > 0$ separate communication frequencies, up to $t < \mathcal{F}$ of which can be disrupted in each round by an abstract adversary. This adversary does not necessarily model a literal adversarial device. Instead, it incarnates the diversity of possible interference sources experienced in an open network environment. To the best of our knowledge, this is the first formal channel model to capture the unique traits of this increasingly relevant setting.

As evidenced by the complexity of the algorithms featured in our previous studies of t-disrupted radio channels [31, 32, 40, 30], solving problems in this model can prove difficult. With this in mind, we introduce the more powerful (t, b, p)-*feedback channel*. This channel behaves like a t-disrupted channel enhanced to provide *feedback* to the processes about what was received on *all* frequencies during the current round. (Specifically, it guarantees that with probability p, at least $n - b$ devices receive the feedback.) We then describe two implementations—one randomized and one deterministic—of a (t, b, p)-feedback channel using a t-disrupted channel.

With the feedback channel defined and implemented, we next describe solutions to the set agreement, gossip, and reliable broadcast problems using this channel. When these algorithms are composed with our feedback channel implementation algorithms, we automatically generate complementary solutions for the t-disrupted channel. These examples highlight the utility of our layered approach to algorithm design in difficult radio network settings.

We conclude by introducing a variant of our modeling framework that captures the unique properties of *ad hoc* radio networks. We then define and provide a solution to the *wireless synchronization problem* in such an ad hoc network with a t-disrupted channel. This problem requires an unknown number of processes, activated (potentially) in different rounds, to agree on a global round numbering. We discuss how these solutions can be used to adapt algorithms designed for a non-ad hoc network (e.g., our feedback channel implementation algorithms) to work in this ad hoc setting.

1.1 Related Work

In this section we summarize the existing theoretical work on radio networks with adversarial behavior. In Section 1.5, we present a extended review of related work in the radio network field.

1.1.1 Early Results Concerning Adversarial Behavior

As mentioned, the theoretical distributed algorithm community has only recently devoted attention to the development of bounds for wireless networks with adversarial behavior. The first work in this direction was the 2004 paper of Koo [54], which studies Byzantine-resilient reliable broadcast. Koo assumed the devices are positioned on an

infinite grid. He further assumed that at most some bounded fraction, t, of any device's neighbors might suffer from Byzantine faults—allowing them to deviate from the protocol. The faulty devices, however, are restricted to broadcasting on a TDMA schedule that prevents collisions. It is also assumed that source addresses cannot be spoofed.

Koo proved that reliable broadcast—the dissemination of a message from a distinguished source device to all other devices—is possible only for $t < \frac{1}{2}R(2R+1)$, where R is the transmission radius of devices on the grid (using the L_{inf} distance metric). In a 2005 paper, Bhandari and Vaidya [10] proved Koo's bound tight by exhibiting a matching algorithm. In 2006, Koo, Bhandari, Katz, and Vaidya [55] extended the model to allow for a bounded number of collisions and spoofed addresses. Assuming these bounds are known, they exhibited an upper bound that is less efficient but tolerates the same fraction of corrupted neighbors as in the original model.

Others have also considered models with probabilistic message corruption. Drabkin et al. [33] allowed Byzantine devices to interfere with communication; each honest message, however, is successfully delivered with some non-zero probability, and authentication is available via public-key cryptography. Pelc and Peleg [77] also assumed that messages may be corrupted (or lost) with some non-zero probability.

In [39, 42], we considered a single-hop model with an external adversary. In contrast to previous work, we did not constrain adversarial behavior—allowing both the jamming and overwriting of messages. This is the first work, to our knowledge, to consider an unrestricted adversary in a wireless context. Such an adversary, of course, can prevent any non-trivial coordination by simply broadcasting continually—jamming the channel.

We turned our attention, therefore, to the *efficiency* of the adversary, producing tight bounds on the *jamming gain*—a term taken from the systems community that we use to describe the minimum possible ratio of honest broadcasts to adversarial broadcasts that can prevent termination. Specifically, we showed a gain of 2 is fundamental for reliable broadcast, leader election, k-selection, and consensus. The implication: regardless of the protocol, the adversary can always prevent termination while expending at most half the number of broadcasts as honest players.

In recent work, Awerbuch et al. [7] considered a similar problem. They assumed an adversary that can jam a single-hop wireless network for a $(1-\epsilon)$ fraction of the rounds. They produced an efficient MAC protocol that achieves a constant throughput on the non-jammed steps, even when only loose bounds on ϵ and the number of devices are known.

1.1.2 The t-Disrupted Radio Channel

The t-disrupted radio channel highlighted in this thesis was first introduced by Dolev, Gilbert, Guerraoui and Newport [31]. In this paper, we studied the gossip problem. In a break from previous models, we replaced *temporal* restrictions on the adversary with *spatial* restrictions. Specifically, we allowed the adversary to cause disruption in *every* round. We assumed, however, that devices have access to $\mathcal{F} > 1$ frequencies, and the adversary can participate on only up to $t < \mathcal{F}$ frequencies per round. If the adversary

broadcasts on a frequency concurrently with an honest device, both messages are lost due to collision. This adversary does not necessarily model a literal malicious device (though it could); it is used an abstraction that incarnates the diversity of unpredictable interference sources that plague crowded radio networks.

In [31], we focused on deterministic oblivious gossip algorithms, and produce a tight bound of $\Theta(\frac{n}{\epsilon\mathcal{F}^2})$ rounds for disseminating $(1-\epsilon)n$ rumors for $t=1$. For $t>1$, we described an oblivious upper bound that requires $O(\frac{ne^{t+1}}{\mathcal{F}\epsilon^t})$ rounds, and proved that for the worst case of $\epsilon = t/n$, any deterministic oblivious solution requires $\binom{n}{t+1}/\binom{\mathcal{F}}{t+1}$ rounds. It remains an open question to generate lower bounds for larger ϵ values; i.e., smaller numbers of rumors needing to be disseminated.

In [32], we returned to this model but now allow for randomization. We generalized gossip with a problem we call *Authenticated Message Exchange* (AME), which attempts to disseminate messages according to an arbitrary "exchange graph"— a graph with one node for each device and edges describing the messages to be exchanged. We presented f-AME, a "fast" solution to AME that requires $\Theta(|E|t^2 \log n)$ rounds, where $|E|$ describes the edge set of the exchange graph. Initializing this protocol with a complete graph, we can solve gossip for $\epsilon = t/n$ in $\Theta(n^2t^2 \log n)$ rounds—a significant improvement over the exponential bounds of [31].

In [40] we closed our study of gossip by analyzing the open case of adaptive deterministic solutions. We showed that for $\mathcal{F} = \Omega(t^2)$, there exists a linear-time solution for gossip with $\epsilon = t/n$ This is a significant improvement over the exponential bounds for the oblivious case and, surprisingly, an improvement over the general randomized solution of [32]. (The randomized solution is optimized for the worst-case $t = \mathcal{F} - 1$, for smaller t there exist simpler randomized algorithms that match this linear bound.) For larger t, however, the performance of the adaptive deterministic solutions degrade to exponential, opening a major gap with the randomized solution.

The strategies used in our feedback channel emulation algorithms of Part II, and the agreement, gossip, and reliable broadcast solutions of Part III, are inspired by the techniques deployed in these gossip papers.

Recently, other researchers have begun to study the t-disrupted setting. Meier et al. [68] proved bounds for the minimum time required for two devices to communicate on an undisrupted frequency. (Their setting differed slightly from ours in that they assumed that the processes did not know t in advance.) Strasser et al. [81, 80] studied the t-disrupted channel from a more practical perspective, proposing efficient schemes for establishing shared secrets and reassembling large messages divided into small packets for transmissions. Jin et al. [74] also studied practical realizations of the setting—with a focus on the issues introduced by real radios.

1.1.3 Other Results

In [30], we modified our model to allow processes to be activated in different rounds. We also eliminated the assumption that processes know the number of participants in advance. In this more difficult *ad hoc* setting, we solve the wireless synchronization problem, which requires processes to agree on a global round numbering.

We presented three main results in this paper. The first is a lower bound of

$$\Omega\left(\frac{\log^2 n}{(\mathcal{F}-t)\log\log(n)}+\frac{\mathcal{F}t}{\mathcal{F}-t}\log n\right)$$

rounds for some process to learn the global number.

The second result is an upper bound, which we call the Trapdoor Protocol, that almost matches the lower bound with a running time of

$$O\left(\frac{\mathcal{F}}{\mathcal{F}-t}\log^2 n+\frac{\mathcal{F}t}{\mathcal{F}-t}\log n\right)$$

rounds.

The third result is an *adaptive* algorithm we call the Good Samaritan Protocol, that terminates in

$$O\left(t'\log^3 n\right)$$

rounds in *good* executions; that is, when the devices begin executing at the same time, and there are never more than t' frequencies disrupted in any given round, for some $t' < t$. In all executions, even those that are not good, it terminates in $O\left(\mathcal{F}\log^3 n\right)$ rounds.

In Part IV, we present a formalization of the ad hoc model studied in the paper. We also present a detailed description and proof of the Trapdoor Protocol.

Our experience with these results emphasized the difficulty of describing a communication model that includes adversarial behavior. For example, a subtle question that arose in our analysis of gossip was the dependencies between the adversarial and algorithmic decisions. For example, if a process chooses a frequency at random in round r, does the channel adversary know this information *before* selecting frequencies to disrupt? Motivated by these subtle questions, in [73] we detailed the formal radio network modeling framework also presented in Part I of this thesis. As mentioned earlier in this introduction, the framework includes an automaton-based description of radio channels and algorithm behavior, formal notions of problems, solving problems, and implementing one channel using another channel, and a collection of useful composition results.

1.2 Contributions

As described in the introduction, this thesis provides four main contributions to the theoretical study of distributed algorithms for unreliable radio channels: (1) a formal modeling framework; (2) the implementation of the powerful (t, b, p)-feedback channel using a realistic t-disrupted radio channel; (3) the solving of set agreement, gossip, and reliable broadcast using a feedback channel (which, when coupled with our composition results, produce solutions for the t-disrupted channel as well); and (4) the ad hoc variant of our model, and a solution to the wireless synchronization problem that can be used to mitigate the difficulties of this setting.

In the following four sections, we describe each of these contributions in detail.

1.2.1 A Formal Modeling Framework for Radio Networks

The vast majority of existing theory concerning radio networks relies on informal English descriptions of the communication model (e.g., "If two or more processes broadcast at the same time then..."). This lack of formal rigor can generate subtle errors. For example, the original BGI paper [8] claimed a $\Omega(n)$ lower bound for multihop broadcast. It was subsequently discovered that due to a small ambiguity in how they described the collision behavior (whether or not a message *might* be received from among several that collide at a receiver), the bound is actually logarithmic [9, 58]. In our work on consensus [21], for another example, subtleties in how the model treated transmitters receiving their own messages—a detail often omitted in informal model descriptions—induced a non-trivial impact on the achievable lower bounds. And as mentioned in Section 1.1, in studies of adversarial behavior in a radio setting [10, 55, 42, 31, 32, 7, 41], informal descriptions of the adversary's power prove perilous, as correctness often rests on nuanced distinctions of the adversary's knowledge and capabilities. For example, when the adversary chooses a frequency to disrupt in a given time slot, what is the exact dependence between this choice and the honest processes' randomization?

Informal model descriptions also prevent comparison between different results. Given two such descriptions, it is often difficult to infer whether one model is strictly stronger than the other or if the pair is incomparable. And without an agreed-upon definition of what it means to implement one channel with another, algorithm designers are denied the ability to build upon existing results to avoid having to solve problems from scratch in every model variant.

The issues outlined above motivate the need for a radio network modeling framework that allows:

- precise descriptions of the channel model being used;
- a mathematical definition of a problem and solving a problem;
- a mathematical notion of implementing one channel using another (perhaps presented as a special case of a problem); and
- composition theorems to combine algorithms designed for powerful channels to work with implementations of these powerful channels using weaker channels.

In this thesis, we present a modeling framework that accomplishes these goals. In more detail, this framework focuses on the *system*, which includes three components: the channel, the algorithm, and the user (see Figure **??**). We briefly describe each:

The Channel

The channel is a probabilistic automaton that receives messages and frequency assignments as input from the algorithm and returns the corresponding received messages. Using this formalism, we can easily provide an automaton definition that captures the standard radio channel properties seen in the existing literature; for example, a

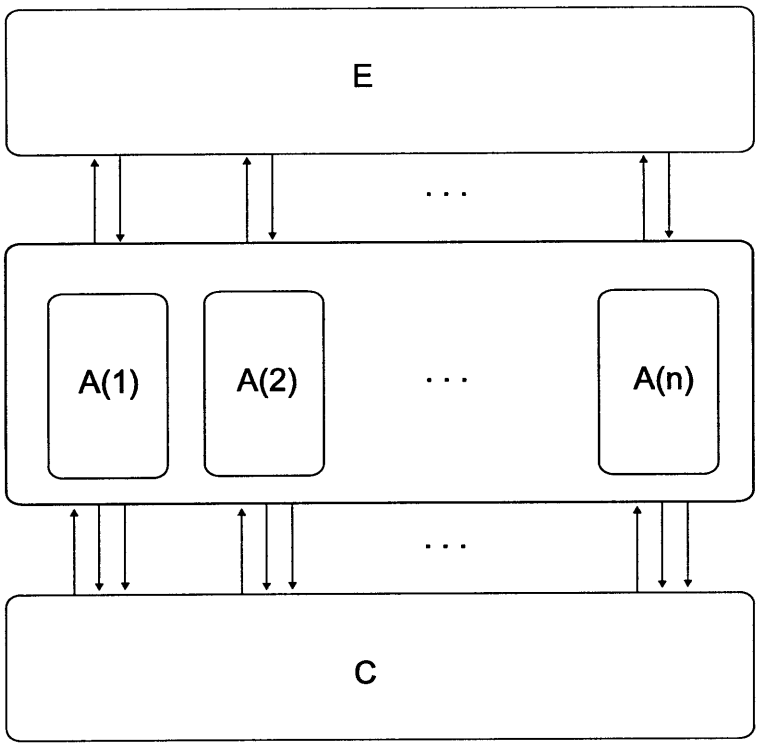

Figure 1-1: A system including environment E, algorithm A (which consists of n processes, $A(1)$,...,$A(n)$), and channel C.

channel that respects integrity, no duplication and throws out all messages if two or more are broadcast concurrently. Such definitions can be extended to include collision detection information of varying strength; for example, as in [21]. Because the channel automaton is probabilistic, we can also capture probabilistic guarantees; for example, on whether messages are corrupted or lost due to collision. Finally, as demonstrated in the systems examined in this thesis, an arbitrary automaton-based definition allows the modeling of powerful adversaries that have access to complete send and receive histories.

The Algorithm

The algorithm is a mapping from the process identifiers to *processes*, which are probabilistic automaton that describe process behavior. Each process automaton is defined by its state set, a transition function, a message generation function and an output generation function. The message generation function generates the message to be passed to the channel in a given round. The output generation function generates any output values to be passed up to the environment. The transition function evolves the state based on messages and environment inputs received during that round. We also describe a useful pseudocode convention that allows for compact, intuitive, and, most importantly, *unambiguous* process definitions.

The Environment

The environment is a probabilistic automaton that sends and receives I/O values from each of the processes described by the algorithm. We use the environment formalism to precisely describe the interaction of an algorithm with the outside world. This allows for formal problem definitions that can encompass concerns of well-formedness and timing-dependencies—that is, we can define both problems that require tight correspondence with the rounds number and those that are agnostic to how many rounds are used.

Our modeling framework also includes the following additional useful definitions and theorems:

Problems

We formalize the concept of a *problem* as a mapping from environments to sets of I/O trace probability functions (i.e., a function from traces to probabilities).[1] We say an algorithm $\mathcal{A}$ solves a problem P with a channel $\mathcal{C}$, only if for every environment $\mathcal{E}$, the trace probability function generated by the system $(\mathcal{C}, \mathcal{A}, \mathcal{E})$ is in $P(\mathcal{E})$.

The mapping-based definition allows us to capture well-formedness properties that an environment must satisfy. For example, we can define a problem to map environments that do *not* satisfy a given well-formedness property to the set of *all* trace functions; in effect saying that if the environment is misbehaved then all algorithms solve the problem with that environment. The mapping to a *set* of distributions allows non-determinism in the problem definitions (notice: the channel, process, and environment automata are probabilistic but *not* non-deterministic), and the use of trace probability functions instead of singleton traces allows us to capture probabilistic behavior in our problems.

Channel Implementation

Our modeling framework allows us to capture a formal notion of channel implementation. That is, given a channel automaton $\mathcal{C}$, we show how to transform it into a problem $P_{\mathcal{C}}$ that captures precisely what it means for an algorithm to emulate the behavior of $\mathcal{C}$ (in this emulation, messages are passed back and forth through the algorithm's I/O interface). This ability to implement channels using other channels is a crucial issue in radio networks. Indeed, from a theoretical perspective, this is the problem solved by the wireless link layer—it takes a poorly behaved "realistic" channel and tries to implement, from the point of view of the network layer and above, a channel that is better behaved. As discussed in the upcoming sections, this link layer approach is the strategy we adopt to tame channels suffering from adversarial interference; we transform them into a better behaved channel before attempting to solve hard problems.

[1] These functions are not distributions, as their domain includes finite traces and extensions of these same traces.

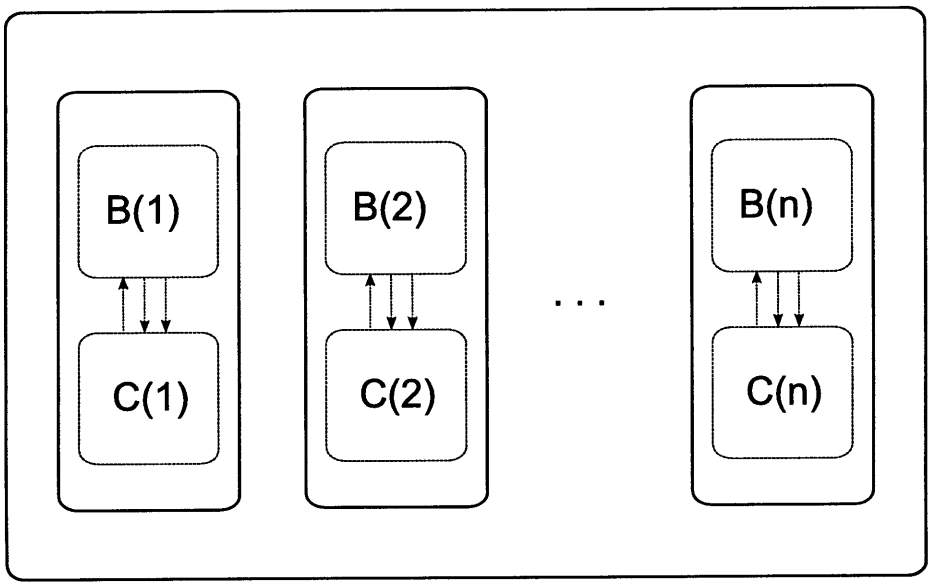

Figure 1-2: The composition algorithm $\mathcal{A}(B, C)$, where B is an algorithm that solves a problem, and C is an algorithm that emulates a channel.

Among other questions this result makes tractable is the comparison of different communication channel properties. If, for example, you could show that channel $\mathcal{C}$ can implement $\mathcal{C}'$, but $\mathcal{C}'$ cannot implement $\mathcal{C}$, you could claim that $\mathcal{C}'$ is *weaker* than $\mathcal{C}$. Such meaningful comparisons are impossible without a formal definition of channels and channel implementation.

Composition

Our modeling framework also includes two channel composition theorems. The first proves that if an algorithm solves a problem P with some channel $\mathcal{C}$, and another algorithm implements channel $\mathcal{C}$ using channel $\mathcal{C}'$, then we can construct a *composition algorithm* that solves P on $\mathcal{C}'$ (see Figure 1-2). The second theorem shows how to compose a channel implementation algorithm $\mathcal{A}$ with a channel $\mathcal{C}$ to generate a new *composition channel* $\mathcal{C}'$ (see Figure 1-3). We prove that $\mathcal{A}$ using $\mathcal{C}$ implements $\mathcal{C}'$. This result is useful for proving properties about a channel implementation algorithm such as $\mathcal{A}$.

These theorems, of course, are a crucial addendum to our formal notion of channel implementation. The purpose of implementing a better-behaved channel using a more difficult low-level channel is to allow the algorithm designers to program for the former. These composition theorems formalize this intuition, providing the necessary theoretical link between the algorithms, the high-level channel implementations, and the low-level channel below.

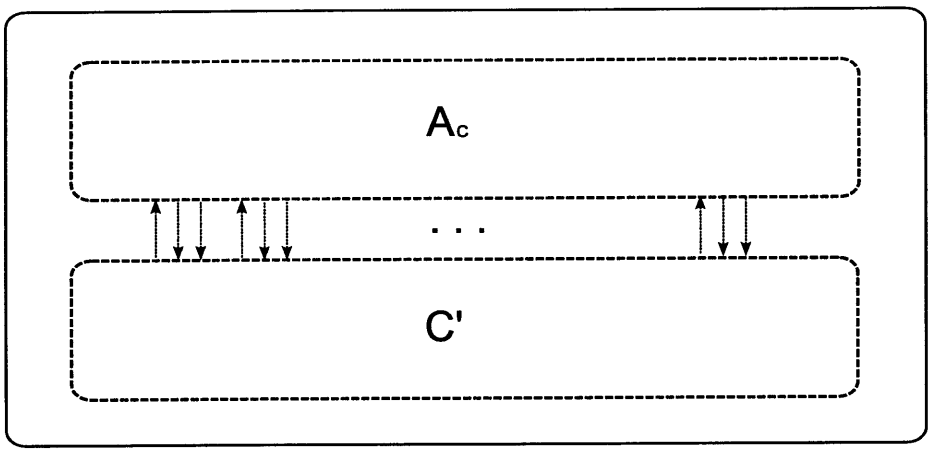

Figure 1-3: The composition channel $\mathcal{C}(A_C, C')$, where A_C is a channel implementation algorithm, and C' is a channel.

1.2.2 Feedback Channel Implementations

The main realistic channel studied in this thesis is the *t-disrupted channel*. Recall, this channel captures a single-hop radio network suffering from adversarial interference. It is comprised of $\mathcal{F}$ separate communication frequencies. During each round, each device chooses one frequency on which to participate. An abstract interference adversary can disrupt up to $t < \mathcal{F}$ of these frequencies per round. Much of our existing work in this model focused on the gossip problem [31, 32, 40], producing efficient algorithms that are, unfortunately, complicated to both describe and prove correct. We attack such complexities—which seem intrinsic to the t-disrupted setting—by implementing a better-behaved high-level channel on the t-disrupted channel. We then design algorithms to work on this more tractable high-level option.

Specifically, we define the (t, b, p)-*feedback channel* as the appropriate high-level channel to consider. This channel behaves like a t-disrupted channel that provides feedback to the devices regarding what was received on *all* $\mathcal{F}$ frequencies. We describe both a deterministic and a randomized implementation of this feedback channel using a low-level t-disrupted channel.

Below we define the feedback channel and summarize the main ideas behind our randomized and deterministic implementations.

The (t, b, p)-Feedback Channel

The (t, b, p)-feedback channel behaves like a t-disrupted channel with the following exception: some devices receive on all $\mathcal{F}$ frequencies simultaneously while others simply receive $\perp$. In contrast, on a t-disrupted channel, a process can participate on only one frequency per round.

Formally, we say that with probability greater than or equal to p, at least $n - b$ of the n devices succeed in receiving on all channels simultaneously. For example, a $(t, 0, 1)$-feedback channel is a t-disrupted channel in which all devices receive on all frequencies during all rounds.

To motivate this definition, we note that a key strategy for tolerating an interference adversary, is to communicate on more frequencies than can be simultaneously disrupted; this idea, for example, forms the core of our gossip solutions [31, 32, 40]. The difficulty of this approach, however, is coordinating the actions of the honest devices—that is, keeping a consistent view of who avoided a disrupted frequency, and who did not, in the previous rounds. Without this information, it is hard to schedule future broadcast rounds without having the honest devices contend among themselves. The feedback of the feedback channel eliminates this challenge, allowing the algorithm designer to focus on the concerns specific to the problem at hand.

The Randomized Implementation

We present the $RFC_{\epsilon,t}$ algorithm, which is a randomized implementation of a $(t, 0, 1 - \frac{1}{n^{\epsilon}})$-feedback channel that requires

$$\Theta\left(\frac{\mathcal{F}^2\epsilon}{\mathcal{F}-t}\log n\right)$$

real rounds for each emulated round of the feedback channel. This algorithm uses a listener/dissemination strategy. In each emulated round of the feedback channel, a single distinguished *listener* device is stationed to receive on each frequency. After the scheduled broadcasts complete, the devices choose frequencies at random and the listeners broadcast what they received during the first round with a constant probability. We prove that this period of random frequency hopping is long enough to disseminate information from the listeners to all the devices in the system, with high probability.

The Deterministic Implementation

We also present the $DFC_{t,M}$ algorithm, which is a deterministic implementation of a $(t, t, 1)$-feedback channel that requires

$$\Theta\left(\mathcal{F}^2|M|\right)$$

real rounds for each emulated round of the feedback channel, where $|M|$ is the size of a novel combinatorial object called a *multiselector* (these are formally defined and bounded in Chapter 2, and informally described below). The value of $|M|$ varies, depending on both the size of t compared to $\mathcal{F}$, and whether the proof of the multiselector's size is existential or constructive. The smallest value for $|M|$ considered in this thesis, which corresponds to $t \leq \sqrt{\mathcal{F}}$ and an existential proof, is $O(t\log(n/t))$, giving $DFC_{t,M}$ a runtime of:

$$\Theta\left(\mathcal{F}^2 t\log(n/t)\right)$$

The largest value for $|M|$ considered in this thesis, which corresponds to $t > \sqrt{\mathcal{F}}$ and a constructive proof, is $O(t^t\log^t(n))$, giving $DFC_{t,M}$ a runtime of:

$$\Theta\left(\mathcal{F}^2 t^t \log^t(n)\right)$$

This algorithm uses a similar listener/dissemination strategy. It cannot, however, rely on the random hopping approach of the randomized solution—in this case we assume no random information. To circumvent this difficulty we recruit *many* listeners for each frequency. To disseminate feedback for a given frequency f, we position a listener from f on each frequency, and have them continually broadcast what they heard on f. We will then have the other devices receive on a shifting pattern of frequencies in an attempt to receive at least one of these equivalent information messages.

This leaves us with the problem of determining an efficient receiving schedule for the non-listener devices. Here we make use of multiselectors. A non-trivial generalization of the *selector* [28, 49, 59]—a structure used in the design of single-frequency radio network algorithms—the multiselector is a sequence of mappings of devices to frequencies, first defined and constructed in [40].

A multiselector guarantees that for all subsets of a certain size, there exists a mapping that splits the subset elements onto distinct frequencies. By listening according to an appropriately parameterized multiselector, we ensure that *most* devices receive the information (i.e., evade disruption at least once).

We repeat the above procedure of having a group of listeners broadcast on unique frequencies, while the other processes receive according to a multiselector, for all of the listener groups, allowing most devices to acquire the needed feedback knowledge. (This description elides some of the subtle issues tackled by the DFC algorithm—such as handling listeners devices have a message to broadcast.)

1.2.3 Set Agreement, Gossip, and Reliable Broadcast Solutions

To prove the utility of the (t, b, p)-feedback channel we present algorithms that use this channel to solve set agreement, gossip, and reliable broadcast—three fundamental building blocks for distributed communication and coordination.

Consider, for example, the simple solution to k-set agreement, using a feedback channel, described in Figure 1-4. (For an explanation of this pseudocode format, see Section 7.2.) Whereas this problem would be difficult to solve on a t-disrupted channel, the feedback provided by our the feedback channel enables a simple solution.

In this example algorithm, $k+1 > t$ devices transmit on different frequencies. All devices then decide on the smallest value received, if they succeed in hearing from all frequencies, and otherwise they decide their own value. With probability p', at most k' do not receive on all frequencies—the rest decide the same minimum value derived from their common feedback. Provided that $k' < k$, with probability p', no more than k values will be decided.

After providing a formal proof of the set agreement problem, and the correctness of our solution, we proceed to describe solutions to both (γ, δ)-gossip and (c, p)-reliable broadcast. The (γ, δ)-gossip problem is a generalization of all-to-all gossip

Local state for Process i:
  $m_i \in V \cup \perp$, initially $\perp$.
  $f_i \in [\mathcal{F}]$, initially 1.

For round $r = 1$:
  $I_i \leftarrow INPUT_i()$
  if $(i \leq t+1)$ then
    $m_i \leftarrow I_i[p]$
    $f_i \leftarrow i$
  $BCAST_i(f_i, m_i)$
  $N_i \leftarrow RECV_i()$
  if $(N_i = \perp)$ then
    $O_i[p] \leftarrow I_i[p]$
  else
    $O_i[p] \leftarrow min(N_i)$
  $OUTPUT_i(O_i)$

Figure 1-4: Solving k-set agreement, with probability at least p_{agree}, using a (t, k', p')-feedback channel, with $k' < k$, $p' \geq p_{agree}$: code presented for process i.

that requires devices to disseminate at least $(1-\gamma)n$ of the n rumors to at least $(1-\delta)n$ devices. And the (c,p)-reliable broadcast problem is a generalization of standard reliable broadcast that requires, with probability p, for messages to disseminate to at least $n - c$ of the receivers.

Our solutions for gossip and reliable broadcast are more involved than the toy example of set agreement. But in both cases, the power of the feedback channel significantly reduces their potential complexity. By combining these results with our feedback channel implementation algorithms and our composition theorems, we generate solutions that work on with a t-disrupted channel.

1.2.4 Ad Hoc Networks and Wireless Synchronization

The main model we consider in this thesis assumes that all devices start during the same round. It also assumes that n, the total number of participating devices, is known in advance. These assumptions simplify the development and verification of algorithms. In real deployments, however, these assumption do not necessarily hold. This is especially true for *ad hoc* networks.

In the final part of our thesis, we loosen our model in two ways:

1. Devices do not necessarily start during the same round.
2. Device know only a loose upper-bound on the total number of devices that will eventually participate.

We call this the *ad hoc radio network model.* In this model, we tackle the *wireless synchronization* problem which requires devices to agree on a global round numbering. In more detail, this problem requires all active devices to output a value from $\mathbb{N}_\perp =$

$\{\perp\} \cup \mathbb{N}$ in each round. Once a device outputs a value from $\mathbb{N}_\perp$, it must never again output $\perp$. We require that devices output values from $\mathbb{N}$ in increasing order, with each value exactly 1 larger than its predecessor. Finally, we require that all non-$\perp$ values output during a given round must be the same and that all devices eventually stop outputting $\perp$. At a high-level, this problem requires devices to agree on a round number.

We describe and prove correct a solution to the wireless synchronization problem. This algorithm runs on a t-disrupted channel in an ad hoc setting, and guarantees that every device synchronizes within

$$O\left(\frac{\mathcal{F}}{\mathcal{F}-t}\log^2 n + \frac{\mathcal{F}t}{\mathcal{F}-t}\log n\right)$$

rounds of being activated. We then discuss a strategy for using such a solution to adapt non ad hoc algorithms to this model. At a high-level, the strategy can be summarized as follows:

1. Divide up rounds into *epochs* of length x, for some constant x. Let each new epoch start at the rounds in $E = \{r \mid r \ mod \ x = 0\}$.

2. Once a process is started and synchronizes to the current round number (i.e., stops outputting $\perp$) it considers itself a "participant."

3. At each round in E, all participants run a simple, fixed-round *announcing protocol* to disseminate the set of participants. A simple announcing protocol might have participants select frequencies at random and then broadcast their id with a reasonable probability.

4. During the rounds after the announcing protocol and before the next epoch begins, all participants who participated in the full announcing protocol can emulate the original radio network model, starting from round 1.

We claim this strategy could potentially be formalized to allow us to run a fixed-length emulation of a synchronous start-up model every x rounds. During this emulation we could execute, for example, our feedback channel implementation and arbitrary protocols—e.g., broadcast, consensus, gossip—on top of the feedback channel.

1.3 Overview of Thesis Chapters

To simplify navigation of this thesis, we provide below a roadmap to the chapters that follow. In Section 1.4 we describe several alternate paths through these chapters for readers who are more interested in the algorithms of the later parts, and less interested in the formal modeling details presented in the early parts.

Chapter 1: Introduction. This chapter contains the introduction that you are currently reading.

Chapter 2: Mathematical Preliminaries. This chapter contains the basic mathematical notation, definitions, and theorems that are used throughout the thesis.

Part I: Channel Models

Chapter 3: Model. This chapter contains the basic model definitions, including the specifications of an environment, algorithm, channel, execution, and trace.

Chapter 4: Problems. This chapter contains the formal definition of a problem and what it means to solve a problem. It also specifies *delay tolerance*, an important problem constraint, and details what it means for a channel to implement another channel.

Chapter 5. Composition. This chapter contains our two main composition results. The first concerns the composition of a problem-solving algorithm with a channel implementation algorithm to produce a new algorithm. The second concerns the composition of a channel implementation algorithm with a channel to produce a new channel.

Chapter 6. Channels This chapter contains formal definitions of the t-disrupted and (t, b, p)-feedback channel properties studied throughout the rest of the thesis. It also includes definitions of the partial collision and total collision properties, both of which are used frequently in existing work on radio networks.

Part II: Algorithms for Implementing a Feedback Channel

Chapter 7: Preliminaries. This chapter contains some notation that will prove useful throughout Part II. It also defines the pseudocode template that we use to efficiently and precisely describe our process definitions for the remainder of the thesis.

Chapter 8: The $RFC_{\epsilon,t}$ Randomized Channel Algorithm. This chapter contains the description and correctness proof for our randomized implementation of a feedback channel.

Chapter 9: The $DFC_{t,M}$ Deterministic Channel Algorithm. This chapter contains the description and correctness proof for our deterministic implementation of a feedback channel.

Part III: Algorithms for Solving Problems Using a Feedback Channel

Chapter 10: Preliminaries. This chapter contains some notation and definitions that will prove useful throughout Part III.

Chapter 11: k-Set Agreement This chapter contains the description and correctness proof of the $SetAgree_t$ algorithm, which solves set agreement using a feedback channel.

Chapter 12: (γ, δ)-Gossip This chapter contains the description and correctness proof of the $FastGossip_{\gamma,t,b}$ algorithm, which solves (γ, δ)-gossip using a feedback channel.

Chapter 13: (c, p)-Reliable Broadcast This chapter contains the description and correctness proof of the $RBcast_{t,p,k}$ algorithm, which solves (c, p)-gossip using a feedback channel.

Part IV: Ad Hoc Networks

Chapter 14: Preliminaries. This chapter contains some notation and definitions that will prove useful throughout Part IV.

Chapter 15: The Ad Hoc Radio Network Model. This chapter contains the formal definition of the *ad hoc* variant of our radio network model.

Chapter 16: The Wireless Synchronization Problem This chapter contains the description and correctness proof of the $Trapdoor_t$ algorithm, which solves the wireless synchronization problem using a t-disrupted channel. It also discusses the use of such a solution to overcome the difficulties of an ad hoc setting.

1.4 Two Alternative Paths Through this Thesis

For the reader more interested in the algorithms of Parts II, III, and IV, and less interested in the modeling details of Part I, we present two alternative paths through this thesis.

Path 1: No Formal Model Details. The first path bypasses most of Part I to focuses on the basic ideas behind the algorithms of the later parts. A reader interested in this path should start by reading the mathematical preliminaries in Chapter 2, and the informal descriptions of the *t-disrupted* and *(t, b, p)-feedback* channels in Section 6.2. The reader can then proceed to the algorithms presented in Parts II and III. For each of these algorithms, the problem overview, pseudocode and algorithm overview should be easily understandable without knowledge of the formal model. The formal problem definitions and correctness proofs, however, will be difficult to follow without this knowledge. It is advised, therefore, that the reader following this path should skip these definitions and proofs. The reader should then proceed to Part IV. The summary included at the beginning of this part provides a high level overview of the ad hoc variant of our modeling framework that is presented in

Chapter 14. The reader should read the overview but then skip the details of Chapter 14 and continue on to the *wireless synchronization problem* solution presented in Chapter 15. As with the preceding algorithms, the reader should read the problem overview, pseudocode, and algorithm description, but skip the formal problem definition and algorithm correctness proofs.

Path 2: Only Essential Formal Model Details. The second path is designed for the reader who wants to understand the *correctness* proofs for the algorithms, but is willing to trust that the *composition* results are correct. This reader should read the thesis chapters in order, skipping Chapter 5. Though avoiding only a single chapter might not seem to save much complexity, the reader following this path should take comfort in the knowledge that Chapter 5 is arguably the most complex of those included in this thesis.

1.5 Additional Related Work

In Section 1.1 we surveyed the related work most directly connected to the results of this thesis. In this section, we adopt a wider perspective, and survey the full history of the theoretical study of radio networks. We begin with the earliest radio data network protocols of the 1970's and end with the 2004 paper that opens Section 1.1.

1.5.1 Early Work on Radio Networks

Early interest in radio data networks dates to 1970 with the development of the ALOHA protocol [2]. Originally conceived for communication between the Hawaiian Islands, ALOHA had devices send data packets as they arrived. If they fail to receive an acknowledgment from the destination they can attempt a retransmission after a random interval. Roberts [79] later refined the protocol to include communication slots. The resulting slotted scheme can be modeled and analyzed as a Markov chain. As Kaplan [50] established, the resulting chain is non-ergodic—as the arrival rate increases, the system descends into instability. It was shown, however, that for any arrival rate $\lambda \leq 1/e$ it is possible to dynamically adjust the packet broadcast probability to ensure stability [45].

This model was generalized as a single-hop multiaccess channel with collisions and transmitter collision detection. In the late 1970's and into the 1980's, research on these channels turned to more advanced algorithms that could overcome the arrival rate limitation of the ALOHA-style random back-off solutions, namely *splitting algorithms*, which increased maximum λ values to as high as .587; c.f., [14, 46, 82]. This class of solution uses a more advanced back-off strategy in which the set of collided messages is repeatedly divided until it reaches a size that succeeds.

Kleinrock and Tobagi [52] introduced the concept of carrier sensing—allowing devices to detect channel use *before* attempting a broadcast. The simple slotted ALOHA style protocols could be modified to use this new capability and subsequently perform competitively with the more complicated splitting algorithms.

See Gallager's 1985 paper on multiaccess channels for a more technical overview of these early results [36].

1.5.2 The Modern Radio Network Model

The work on radio channels that dominated the 1970's and 1980's focused on an idealized single-hop model with transmitter collision detection. Researchers assumed all devices were generating messages at a fixed rate, and they pursued the goal of achieving stable throughput. Much of the modern theoretical analysis of radio networks differs from this model into two key points. First, the transmitter collision detection is removed in favor of either no detection or the basic ability for a receiver to distinguish between *silence* and *collision*. Second, the model of packets arriving at all devices at a regular rate is now often replaced with a single *source* node trying to communicate a single message to the entire network. This is commonly dubbed the *reliable broadcast* problem.

The first study to capture this modern model was the work of Bar-Yehuda, Goldreich, and Itai. Their seminal paper on broadcast in a radio network [8] modeled a multi-hop network in which there was no transmitter or receiver collision detection. Devices are assumed to only know their label and the label of their neighbors. They describe a randomized algorithm with expected runtime of $O(D \log n + \log n^2)$ rounds, and demonstrate a class of diameter 2 network graphs on which deterministic solutions require $\Omega(n)$ rounds.

The original BGI paper claimed their lower bound held for a model in which colliding broadcasts always lead to collisions—causing the receivers to hear silence. In a subsequent errata notice [9], the authors refined the linear lower bound to hold only for a more pessimistic model in which one of the colliding broadcasts might be received. Kowalski and Pelc [58] later demonstrated that in the original collision model, there exists an $O(\log n)$ deterministic broadcast algorithm for this same class of small diameter networks. They also describe a sub-linear algorithm for any network with $o(\log \log n)$ diameter, and show the existence of diameter 4 networks that require $\Omega(n^{1/4})$ rounds for a deterministic solution. Coupled with the randomized algorithm of BGI, this result establishes the exponential gap between determinism and randomization originally proposed in the earlier work. In [56, 29] the randomized algorithm of [8] is improved to $O(D \log n/D + \log^2 n)$. Most subsequent work assumes this stronger condition that two or more concurrent broadcasts always yields a collision.

The early 1990's also found interest applied toward centralized deterministic solutions to broadcast. In [17] a $O(D \log^2 n)$ solution was proposed, which was later improved in [35] to $O(D + \log^5 n)$. Alon et al. [6] established a lower bound of $\Omega(\log^2 n)$.

Another popular variation was to study distributed algorithms with even *less* information than in [8, 58, 56, 29]. Under the assumption that devices know only their own labels (and not the number or identity of their neighbors), Chlebus et al. described a deterministic upper bound that runs in time $O(n)$. This was later proven optimal [57]. In terms of randomized lower bounds in this setting, Kushelevitz and

Mansour proved $\Omega(D \log n/D)$ to be necessary. Considering that the $\Omega(\log^2 n)$ bound of [6] also holds for randomized algorithms, this implies that the $O(D \log n/D + \log^2 n)$ solution of [56, 29] is optimal.

The above citations still capture only a fraction of the endless variations under which the broadcast problem has been studied. Other constraints thrown into the modeling mix include: a bound Δ on the maximum degree of the network graph; receiver-side collision detection; geometric constraints on the node deployment (e.g., unit disk graphs); and the ability of devices to broadcast *spontaneously*—that is, before receiving the broadcast message. Each unique subset of constraints yields its own research lineage. A more extended summary of these results can be found in the related work section of [58].

1.5.3 Beyond Broadcast

Though broadcast is the standard problem studied in the radio model, it is not the only problem. Consider, for example, the *wake-up* problem [18, 24, 37], which has devices in a radio network activated according to an arbitrary (perhaps adversarial) schedule, and tries to minimize the time between the first activation and the first non-collided transmission.

Other problems that have received considerable attention in this radio model include gossip [25, 38], leader election [70], and consensus [72, 23, 21]. The latter papers weaken the radio model by allowing for more arbitrary behavior when collisions occur. It also augments devices with variable-strength receiver-side collision detectors.

1.5.4 Tolerating Failures in Radio Networks

The introduction of device failures in the radio network model begins in the late 1990's—surprisingly late considering the long-term focus on fault-tolerance in the classical distributed computing literature.

Crash Failures. A 1997 paper by Pagani and Rossi [75] studies radio broadcast resilient to transient send and receive failures. The next year, Kushelevitz and Mansour [65] considered a radio channel in which each bit of a message is flipped with a given probability.

The first appearance of classic device failures appeared in the work of Kranakis et al. [60, 61] which considers broadcast on mesh networks with up to t devices permentantly failed (i.e., they begin the execution in a failed state). For the oblivious case they prove a tight bound of $\Theta(D + t)$ rounds and for the adaptive case $\Theta(D + \log min\{R, t\})$, where R is the transmission radius in the mesh and t a bound on failures. Five years later, Clementi et al. [27] noted that any algorithm designed for an unknown network (i.e., a model in which each device knows only its own label) will also work in the permenant failure model of Kranakis (e.g., [57]).

In [27], Clementi et al. considered crash faults that can occur at any point during the execution.

They proved a lower bound of $\Omega(Dn)$ rounds for oblivious broadcast, and show that it holds for the adaptive case if $D = \Theta(\sqrt{n})$. The surprising implication is that a simple *round-robin* approach is optimal for dealing with crash failures. If the maximum in-degree Δ is $o(n/\log n)$, however, they can improve their upper bound to $O(D \cdot min\{n, \Delta \log n\})$.

In single-hop radio networks, crash failures were first studied in the context of the classic *Do-All* problem. A pair of papers [19, 26] in 2001 and 2002, investigate optimal solutions in this model.

Adversarial Failures. This brings us, finally, to the work we reviewed in section 1.1, which we describe as the direct motivation for the work in this thesis. See Section 1.1 for details.

Chapter 2

Mathematical Preliminaries

Here we define some notation, constants, concepts and facts used throughout the thesis.

2.1 Notation and Constants

We begin with the following useful notation:

- We use $[i]$, for any positive integer i, to denote the integer set $\{1, ..., i\}$.
- We use S^i, for some non-empty value set S and positive integer i, to denote all vectors of length i consisting only of values from S.
- We use *i-vector*, for positive integer i, to refer to a vector of size i.
- We use the notation $P(S)$, for non-empty set S, to describe the powerset of S.
- We use the '$*$' symbol within a function call to indicate a wildcard character—that is, it represents *for all inputs from the corresponding domain set.*
- Given a set S, we used the notation $S_\perp$ to describe the set $S \cup \{\perp\}$.

We continue by fixing the following constants:

- Let $\mathcal{M}$, $\mathcal{R}$, $\mathcal{I}$, and $\mathcal{O}$ be four non-empty value sets that do not include the value $\perp$.
  (These represent the possible sent messages, received messages, environment inputs, and environment outputs, respectively. Notice, we use a different alphabet for sent and received messages, as the latter might include additional symbols unique to the act of receiving; e.g., collision indicators or partial information about what was detected on the channel.)
- Let n and $\mathcal{F}$ be integers greater than 0.
  (They describe the number of processes and available communication frequencies, respectively.)

We also introduce the following additional notation, which is defined with respect to these constants:

- We call a vector from $(\mathcal{I}_\perp)^n$ an *input assignment.*
- We call a vector from $(\mathcal{O}_\perp)^n$ an *output assignment.*
- We call an input or output assignment *empty* if and only if it equals $\perp^n$.
- We call a vector from $(\mathcal{M}_\perp)^n$ a *send assignment.*
- We call a vector from $[\mathcal{F}]^n$ a *frequency assignment.*
- We call a vector from $(\mathcal{R}_\perp)^n$ a *receive assignment.*

2.2 Probability Facts

The following theorems will be used in several locations throughout the thesis where probabilistic analysis is required.

Theorem 2.2.1 (Useful Probability Facts) *The following two facts hold:*

1. *For any probability* $p > 0 : (1-p) < e^{-p}$.
2. *For any probability* $p \leq 1/2 : (1-p) \geq (1/4)^p$.

Proof. We begin with the first fact. We know from Bernoulli that:

$$\lim_{n\to\infty}\left((1-\frac{1}{n})^n\right) = 1/e$$

For $x, 0 < x \leq 1$, we can rewrite the equation as an inequality:

$$(1-x)^{\frac{1}{x}} < 1/e$$

Given some probability $p > 0$, we can use the above inequality to prove our fact:

$$\begin{aligned}(1-p) &= \left((1-p)^{\frac{1}{p}}\right)^p \\ &< (1/e)^p \\ &= e^{-p}\end{aligned}$$

The second fact follows from a simple argument concerning the shape of the two relevant graphs. Notably, between the points $p = 0$ and $p = 1/2$, the curve for $(1/4)^p$ is concave—bounded from above by $(1-p)$. □

Theorem 2.2.2 (Union Bound) *For arbitrary events* $\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_k$,

$$Pr[\bigcup_{i=1}^{k} \mathcal{E}_i] \leq \sum_{i=1}^{k} Pr[\mathcal{E}_i]$$

2.3 Multiselectors

The $DFC_{t,M}$ protocol, presented in Chapter 9 makes use of a combinatorial object called a *multiselector*. Multiselectors generalize the the *selector* family of combinatorial objects [53, 12]. Whereas selectors are sometimes used to solve problems in single frequency radio networks, we use *multi*selectors for to solve problems in *multiple* frequency networks. These objects were first introduced in [40]. The definitions and theorems provided below all come from this original paper.

Definition 2.3.1 *An (n,c,k)-multiselector, where $n \geq c \geq k \geq 1$, is a sequence of functions $M_1, M_2, \ldots, M_m$, from $[n] \rightarrow [c]$, such that:*

> *For every subset $S \subseteq [n]$, where $|S| = k$, there exists some $\ell \in [m]$ such that M_ℓ maps each element in S to a unique value in $[c]$, that is, $\forall i, j \in S$, $i \neq j$: $M_\ell(i) \neq M_\ell(j)$.*

The next theorems establishes the existence of (n, c, k)-multiselectors and bounds their size. The proofs for the first two theorems are non-constructive.

Theorem 2.3.2 (From [40]) *For every $n \geq c \geq k$, there exists an (n, c, k)-multiselector of size:*

$$\begin{aligned} c = k &: \frac{ke^c}{\sqrt{2\pi c}} \ln \frac{en}{k} \\ c/2 < k < c &: ke^k \ln \frac{en}{k} \\ k \leq c/2 &: k2^{2k^2/c} \ln \frac{en}{k} \end{aligned}$$

For the case where $c >> k$, we can prove the existence of more efficient multiselectors.

Theorem 2.3.3 (From [40]) *For every $n \geq c \geq k^2$, there exists an (n, c, k)-multiselector of size $O(k \log(n/k))$.*

In [40] we also show how to explicitly construct multiselectors. The resulting sizes, however, are larger than those produced by the existential proofs. In the following two theorems, the notation "we can construct a (n, c, k)-multiselector" indicates that we specify an algorithm, in the accompanying proofs, that produces an (n, c, k)-multiselector as output.

Theorem 2.3.4 (From [40]) *For every n, c, k, $n \geq c \geq k$, we can construct a (n, c, k)-multiselector of size $O(k^k \log^k n)$.*

As with the existential proofs, a large c relative to k provides a more efficient construction.

Theorem 2.3.5 (From [40]) *For every $n > c > k^2$, we can construct a (n, c, k)-multiselector of size $O(k^6 \log^5 n)$.*

In the presentation of the $FastGossip_t$ algorithm, in Chapter 12.2, we make use of a generalization of the multiselector, called, for obvious reasons: a *generalized multiselector*. As with the regular multiselectors from above, this structure was first introduced in [40], which provided the following definition:

Definition 2.3.6 (From [40]) *A generalized (n,c,k,r)-multi-selector, where $n \geq c \geq k \geq 1$ and $n \geq r$, is a sequence of functions $M_1, M_2, \ldots, M_m$ from $[n] \rightarrow [0, c]$ such that:*

> *For every subset $S \subseteq [n]$ where $|S| = r$, for every subset $S' \subseteq S$ where $|S'| = k$, there exists some $\ell \in \{1, \ldots, m\}$ such that (1) M_ℓ maps each element in S' to a unique value in $\{1, \ldots, c\}$, and (2) M_ℓ maps each element in $S \setminus S'$ to 0.*

In [40], we proved the following about the size of these objects:

Theorem 2.3.7 (From [40]) *For every $n \geq r \geq c \geq k$ where $n \geq 2r$, there exists (n, c, k, r)-multi-selectors of size $O\left(r\frac{(c+1)^r e^k}{k^k} \log(en/r)\right)$ or $O\left(r\frac{(c+1)^r}{(c-k)^k} \log(en/r)\right)$.*

Part I

Channel Models

In Part I we describe our radio network modeling framework. In Chapter 3, we define the basic components of our model and their execution. In Chapter 4, we define a problem and what it means to solve a problem. We also introduce a special type of problem that corresponds to implementing a channel. We continue, in Chapter 5, by proving our two composition theorems. The first composes a problem-solving algorithm and a channel emulation algorithm to generate a new algorithm. The second composes a channel emulation algorithm and a channel to generate a new channel. We conclude the part with Chapter 6, which formalizes the channel properties studied in the remainder of this thesis.

Chapter 3

Model

We model n distributed probabilistic processes that operate in synchronized time slots and communicate on a shared radio channel comprised of $\mathcal{F}$ independent communication frequencies. We assume that all processes start during the same round, and each has a unique id in the range 1 to n, where the value of n is known. Later, in Part IV, we describe how to relax these assumptions.

The processes communicate with each other using the radio channel. We assume that each process chooses a single frequency on which to send or receive messages during each round. The processes can also interact with the environment through input and output ports. In each round, each process can receive a single input value from the environment and return a single output value.

3.1 Systems

We formalize our setting with a collection probabilistic automata. Specifically, we use one such automaton to model each process (and label the collection of all n processes an *algorithm*), another automata models the *channel*, and another models the *environment*. The combination of an algorithm, channel, and environment defines a *system* (see Figure 3-1). We define each component below. In the following, we use the notation *finite support*, with respect to a discrete probability distribution over some sample space S, to indicate that the distribution assigns non-zero probability to only a finite subset of elements from S.

Definition 3.1.1 (Channel) *A* channel *is an automaton $\mathcal{C}$ consisting of the following components:*

- **cstates**$_{\mathcal{C}}$*, a potentially infinite set of* states.
- **cstart**$_{\mathcal{C}}$*, a state from states*$_{\mathcal{C}}$ *known as the* start state.
- **crand**$_{\mathcal{C}}$*, for each state $s \in cstates_{\mathcal{C}}$, a discrete probability distribution with finite support over cstates*$_{\mathcal{C}}$*.*
  (This distribution captures the probabilistic behavior of the automaton.)

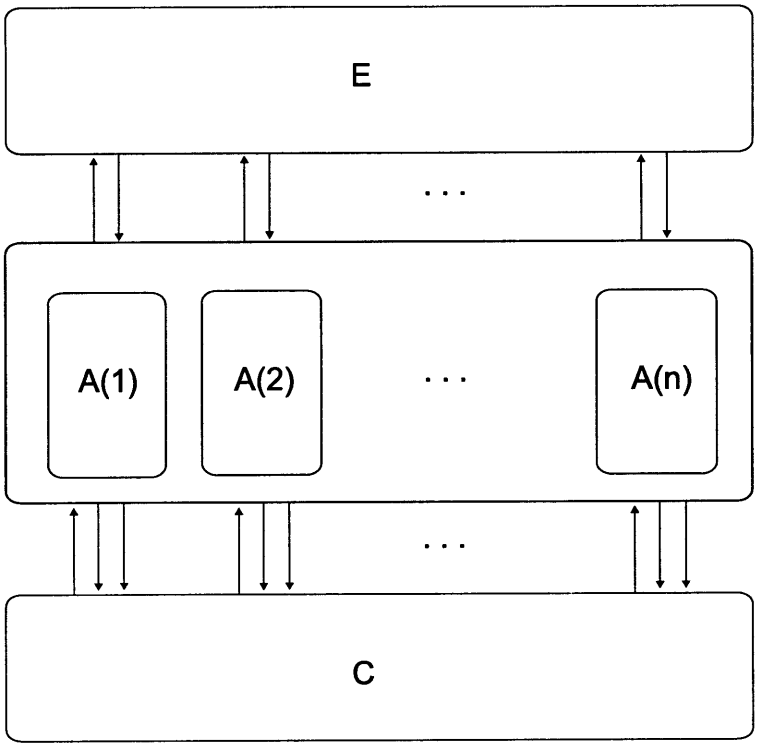

Figure 3-1: A system including environment E, algorithm A (which consists of n processes, $A(1)$,...,$A(n)$), and channel C. The arrows connecting the environment and processes indicate that the environment passes inputs to the processes and the processes return outputs in exchange. The arrows between the processes and the channel capture the broadcast behavior: the processes pass a message and frequency to the channel which returns a received message.

- **crecv**$_{\mathcal{C}}$, *a message set generation function that maps* cstates$_{\mathcal{C}} \times (\mathcal{M}_{\perp})^n \times [\mathcal{F}]^n$ *to* $(\mathcal{R}_{\perp})^n$.
  (Given the message sent—or $\perp$ *if the process is receiving—and frequency used by each process, the channel returns the message received—or* $\perp$ *if no messages is received—by each process. Notice that this function is deterministic; probabilistic behavior is capture by* crand$_{\mathcal{C}}$*.)*

- **ctrans**$_{\mathcal{C}}$, *a transition function that maps* cstates$_{\mathcal{C}} \times (\mathcal{M}_{\perp})^n \times [\mathcal{F}]^n$ *to* cstates$_{\mathcal{C}}$.
  (The state transition function transforms the current channel state based on the messages sent and frequencies chosen by the processes during this round. Notice that as with crecv$_{\mathcal{C}}$ *this function is deterministic; probabilistic behavior is captured by* crand$_{\mathcal{C}}$*.)*

Because we model a channel as an automaton, we can capture a wide variety of possible channel behavior. It might, for example, describe a simple single-hop radio channel with fixed deterministic receive rules. On the other hand, it could also encode a complex multihop topology and a sophisticated (perhaps probabilistic) physical layer model. It can even describe adversaries with precise power constraints, such as those included in the t-disrupted and (t, b, p)-feedback channel properties of Chapter 6.

We continue with the definition of an environment.

Definition 3.1.2 (Environment) *A environment is an automaton* $\mathcal{E}$ *consisting of the following components:*

- **estates**$_{\mathcal{E}}$, *a potentially infinite set of* states.

- **estart**$_{\mathcal{E}}$, *a state from* estates$_{\mathcal{E}}$ *known as the* start state.

- **erand**$_{\mathcal{E}}$, *for each state* $s \in$ estates$_{\mathcal{E}}$, *a discrete probability distribution with finite support over* estates$_{\mathcal{E}}$.
  (This distribution captures the probabilistic behavior of the automaton.)

- **ein**$_{\mathcal{E}}$, *an* input generation function *that maps* estates$_{\mathcal{E}}$ *to* $(\mathcal{I}_{\perp})^n$.
  (This function generates the input the environment will pass to each process in the current round. The $\perp$ *placeholder represents no input. Notice that this function is deterministic; all probabilistic behavior is captured by* erand$_{\mathcal{E}}$*.)*

- **etrans**$_{\mathcal{E}}$, *a transition function that maps* estates$_{\mathcal{E}} \times \mathcal{O}_{\perp}$ *to* estates$_{\mathcal{E}}$.
  (The transition function transforms the current state based on the current state and the outputs generated by the processes during the round. The $\perp$ *placeholder represents no output. Notice that this function, as with* ein$_{\mathcal{E}}$*, is deterministic; all probabilistic behavior is captured by* erand$_{\mathcal{E}}$*.)*

Environments model the interaction of the outside world with the processes. It allows us to capture well-formedness conditions on the inputs provided to our processes in terms of constraints on environment definitions. We continue with our final automata type, the process.

Definition 3.1.3 (Process) *A* process *is an automaton $\mathcal{P}$ consisting of the following components:*

- **states**$_{\mathcal{P}}$, *a potentially infinite set of* states.
- **start**$_{\mathcal{P}}$, *a state from states*$_{\mathcal{P}}$ *known as the* start state.
- **rand**$_{\mathcal{P}}$, *for each state $s \in$ states*$_{\mathcal{P}}$, *is a discrete probability distribution with finite support over states*$_{\mathcal{P}}$. *(This distribution captures the probabilistic behavior of the automaton.)*
- **msg**$_{\mathcal{P}}$, *a* message generation function *that maps states*$_{\mathcal{P}} \times \mathcal{I}_{\perp}$ *to* $\mathcal{M}_{\perp}$. *(Given the current process state and input from the environment—or $\perp$ if no input—this process generates a message to send—or $\perp$ if it plans on receiving. Notice that this function is deterministic; all the probabilistic behavior is captured by rand*$_{\mathcal{P}}$.*)*
- **freq**$_{\mathcal{P}}$, *a* frequency selection function *that maps states*$_{\mathcal{P}} \times \mathcal{I}_{\perp}$ *to* $[\mathcal{F}]$. *(This function is defined the same as msg*$_{\mathcal{P}}$, *except it generates a frequency to participate on instead of a message to send. Notice that this function, as with msg*$_{\mathcal{P}}$, *is deterministic; all the probabilistic behavior is captured by rand*$_{\mathcal{P}}$. *)*
- **out**$_{\mathcal{P}}$, *an* output generation function *that maps states*$_{\mathcal{P}} \times \mathcal{I}_{\perp} \times \mathcal{R}_{\perp}$ *to* $\mathcal{O}_{\perp}$. *(Given the current process state, input from the environment, and message received, it generates an output—or $\perp$ if no output—to return to the environment. Notice that this function, as with the two above, is deterministic; all the probabilistic behavior is captured by rand*$_{\mathcal{P}}$.*)*
- **trans**$_{\mathcal{P}}$, *a state transition function mapping states*$_{\mathcal{P}} \times \mathcal{R}_{\perp} \times \mathcal{I}_{\perp}$ *to states*$_{\mathcal{P}}$. *(This state transition function transforms the current state based on the input from the environment and the received message. Notice that this function, as with the three above, is deterministic; all the probabilistic behavior is captured by rand*$_{\mathcal{P}}$.*)*

We combine the processes into an algorithm.

Definition 3.1.4 (Algorithm) *An algorithm $\mathcal{A}$ is a mapping from $[n]$ to processes.*

As a useful shorthand, we sometimes refer to a process $\mathcal{A}(i)$ of an algorithm $\mathcal{A}$, for $i \in [n]$, simply as *process i*, or just i, if the *process* as implied by the context.

We can now pull together the pieces.

Definition 3.1.5 (System) *A system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$, consists of an environment $\mathcal{E}$, an algorithm $\mathcal{A}$, and a channel $\mathcal{C}$.*

Before continuing with the definition of an execution. we demonstrate the utility of our formalism by defining several standard algorithm constraints.

Definition 3.1.6 (Deterministic Process) *We say a process $\mathcal{P}$ is deterministic if and only if $\forall s, s' \in states_{\mathcal{P}}, s \neq s' : rand_{\mathcal{P}}(s)(s) = 1$, and $rand_{\mathcal{P}}(s)(s') = 0$.*

That is, a deterministic process is defined such that its probabilistic state transformation is the identity function. We can now combine deterministic processes to yield a deterministic algorithm.

Definition 3.1.7 (Deterministic Algorithm) *We say an algorithm $\mathcal{A}$ is deterministic if and only if for all $i \in [n]$, $\mathcal{A}[i]$ is a deterministic process.*

Finally, we capture the special case of algorithms that do not assume unique identifiers (or asymmetric knowledge).

Definition 3.1.8 (Uniform Algorithm) *We say an algorithm $\mathcal{A}$ is uniform if and only if it maps all values from $[n]$ to the same process.*

3.2 Executions

We now define an execution of a system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$.

Definition 3.2.1 (Execution) *An* execution *of a system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$ is a (potentially infinite) sequence*

$$S_0, C_0, E_0, R_1^S, R_1^C, R_1^E, I_1, M_1, F_1, N_1, O_1, S_1, C_1, E_1, ...$$

where for all $r \geq 0$, S_r and R_r^S are n-vectors, where for each $i \in [n]$, $S_r[i]$, $R_r^S[i] \in \mathcal{A}(i)$, C_r and R_r^C are in $cstates_{\mathcal{C}}$, E_r and R_r^E are in $estates_{\mathcal{E}}$, M_r is in $(\mathcal{M}_\perp)^n$, F_r is in $[\mathcal{F}]^n$, N_r is in $(\mathcal{R}_\perp)^n$, I_r is in $(\mathcal{I}_\perp)^n$, and O_r is in $(\mathcal{O}_\perp)^n$. We assume the following constraints:

1. *If finite, the sequence ends with an environment state E_r, for some $r \geq 0$.*
2. *$\forall i \in [n] : S_0[i] = start_{\mathcal{A}(i)}$.*
3. *$C_0 = cstart_{\mathcal{C}}$.*
4. *$E_0 = estart_{\mathcal{E}}$.*
5. *For every round $r > 0$:*
   (a) *$\forall i \in [n] : R_r^S[i]$ is selected according to distribution $rand_{\mathcal{A}(i)}(S_{r-1}[i])$.*

(b) R_r^C *is selected according to* $crand_{\mathcal{C}}(C_{r-1})$.

(c) R_r^E *is selected according to* $erand_{\mathcal{E}}(E_{r-1})$.

(d) $I_r = ein_{\mathcal{E}}(R_r^E)$.

(e) $\forall i \in [n] : M_r[i] = msg_{\mathcal{A}(i)}(R_r^S[i], I_r[i])$ *and* $F_r[i] = freq_{\mathcal{A}(i)}(R_r^S[i], I_r[i])$.

(f) $N_r = crecv_{\mathcal{C}}(R_r^C, M_r, F_r)$.

(g) $\forall i \in [n] : O_r[i] = out_{\mathcal{A}(i)}(R_r^S[i], I_r[i], N_r[i])$.

(h) $\forall i \in [n] : S_r[i] = trans_{\mathcal{A}(i)}(R_r^S[i], N_r[i], I_r[i])$.

(i) $C_r = ctrans_{\mathcal{C}}(R_r^C, M_r, F_r)$.

(j) $E_r = etrans_{\mathcal{E}}(R_r^E, O_r)$.

Our execution definition has each process, the channel, and the environment start in their well-defined start states (conditions 1–3). It then models each round $r > 0$ unfolding as follows. First, the processes, environment, and channel transform their states (probabilistically) according to their corresponding *rand* distribution (4.*a*–4.*c*). Next, the environment generates inputs to pass to the processes by applying *ein* to its current state (4.*d*). Next, the processes each generate a message to send (or $\perp$ if they plan on receiving) and a frequency on which to do this sending or receiving, by applying *msg* and *freq* to their current states (4.*e*). Next, the channel returns the received messages to the processes by applying *crecv* to the channel state and the messages and frequencies selected by the processes (4.*f*). Next, the processes generate output values to pass back to the environment by applying *out* to their state, the environment input, and the received messages (4.*g*). Finally, all automata transition to new states by applying their transition functions to all relevant inputs from the round (4.*h*–4.*j*).

Much of our later analysis concerns *finite executions.* Keep in mind, by condition 1, a finite execution must end with an environment state assignment, E_r, for $r \geq 0$. That is, it contains no partial rounds.

Give a finite execution of a system, we define a function Q that returns the probability of that execution being generated by the system. This function will provide the foundation for definitions that concern the probability of various behaviors in our model.

Definition 3.2.2 (Q) *For every system* $(\mathcal{E}, \mathcal{A}, \mathcal{C})$*, and every finite execution* α *of this system,* $Q(\mathcal{E}, \mathcal{A}, \mathcal{C}, \alpha)$ *describes the probability that* $(\mathcal{E}, \mathcal{A}, \mathcal{C})$ *generates* α*. That is,* $Q(\mathcal{E}, \mathcal{A}, \mathcal{C}, \alpha)$ *is the product of the probabilities of probabilistic state transformations in* α *as described by* $rand_{\mathcal{A}}$*,* $crand_{\mathcal{C}}$*, and* $erand_{\mathcal{E}}$.

3.3 Traces

We define a trace as a sequence of input and output vectors. This captures the interaction of an algorithm with the environment. As is standard in the modeling of distributed systems, we will later define problems in terms of allowable traces and their respective probabilities.

Definition 3.3.1 (Trace) *A trace t is a (potentially infinite) sequence of vectors from $(\mathcal{I}_\perp)^n \cup (\mathcal{O}_\perp)^n$. Let T be the set of all traces.*

To make use of traces, we need a formal notion of the probability of a system generating a fixed trace. We tackle this challenge with the introduction of a *trace probability function*, which is a function that maps finite traces to probabilities.

In this section we define two useful trace probability functions that correspond to systems. They return the probability that the corresponding system generate a given trace passed as input. Before describing these functions, however, we need the following three helper definitions, which are useful for extracting traces from executions.

- The function **io** maps an execution to the subsequence consisting only of the $(\mathcal{I}_\perp)^n$ and $(\mathcal{O}_\perp)^n$ vectors.
- The function **cio** maps an execution α to $io(\alpha)$ with all $\perp^n$ vectors removed. The 'c' in *cio* indicates the word *clean*, as the function *cleans* empty vectors from a finite trace. Notice, both input and output value vectors can equal $\perp^n$, as both $\mathcal{I}_\perp$ and $\mathcal{O}_\perp$ include $\perp$; *cio*, however, makes no distinction between input and output when cleaning out these vectors.
- The predicate **term** returns *true* for a finite execution α, if and only if the output vector in the final round of α does not equal $\perp^n$.

We now define our two trace probability functions: D and D_{tf}. The difference between D and D_{tf} is that the latter ignores *empty vectors*—that is, input or output vectors consisting only of $\perp$. (The letters tf abbreviate "time free," as it ignores the amount of time elapses between meaningful inputs and outputs.)

We use *term* in conjunction with *cio* in the definition of D_{tf}, to prevent a prefix and an extension of the prefix (with only $\perp^n$ inputs and outputs in the extension) from both being included in the sum for a given trace. (The helper function *cio* would map both the prefix and its extended version to the same trace.)

Definition 3.3.2 (D & D_{tf}) *For every system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$, and every finite trace β, we define the trace probability functions D and D_{tf} as follows:*

- $D(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) = \sum_{\alpha | io(\alpha)=\beta} Q(\mathcal{E}, \mathcal{A}, \mathcal{C}, \alpha)$.
- $D_{tf}(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) = \sum_{\alpha | term(\alpha) \wedge cio(\alpha)=\beta} Q(\mathcal{E}, \mathcal{A}, \mathcal{C}, \alpha)$.

Given a system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$ and a finite trace β, $D(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta)$ returns the probability that this system generates this trace, while $D_{tf}(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta)$ returns the probability that the system generates a trace that differs from β only in the inclusion of extra $\perp^n$ vectors.

Notice, these functions are *not* distributions. If you sum them over all traces for a given system you can generate a value much greater than 1. This follows because the domain of the functions are *finite* traces, meaning that such a sum would include traces and their own extensions. Both functions *are* distributions, however, when considered over all traces of some fixed length.

Chapter 4

Problems

In this chapter we provide a formal definition of a *problem*, in terms of traces, and then provide two definitions of *solving* a problem—one that considers the rounds between "meaningful" inputs and outputs (i.e., not $\perp^n$) and one that does not.

4.1 Problems, Solving, and Time-Free Solving

We begin by defining a problem as a function from environments to sets of trace probability functions. At a high-level, this function describes for each environment (that is, each source of inputs to the algorithms), what behavior would be considered *solving* the problem with respect to that environment. This approach simplifies the handling of *well-formedness*, as, for example, one can define a problem to say: *if an environment does not behave as the algorithm expects, then map to all possible trace probability functions.*

Definition 4.1.1 (E) *Let E be the set of all possible environments.*

Definition 4.1.2 (Problem) *A problem P is a mapping from E to sets of trace probability functions.*

We can now specify two notions of solving a problem: *solves* and *time-free solves.* The former considers $\perp^n$ vectors while the latter ignores them.

Definition 4.1.3 (Solves & Time-Free Solves) *We say algorithm $\mathcal{A}$* solves *problem P using channel $\mathcal{C}$ if and only if:*

- $\forall \mathcal{E} \in E, \exists F \in P(\mathcal{E}), \forall \beta \in T : D(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) = F(\beta)$.
  (Or, equivalently: $\forall \mathcal{E} \in E : \lambda\beta\ D(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) \in P(\mathcal{E})$.*)*

We say $\mathcal{A}$ time-free *solves P using $\mathcal{C}$ if and only if:*

- $\forall \mathcal{E} \in E, \exists F \in P(\mathcal{E}), \forall \beta \in T : D_{tf}(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) = F(\beta)$.
  (Or, equivalently: $\forall \mathcal{E} \in E : \lambda\beta\ D_{tf}(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) \in P(\mathcal{E})$.*)*

Both definitions capture the idea that for all environments $\mathcal{E}$, the trace probability function $D(\mathcal{E}, \mathcal{A}, \mathcal{C}, *)$ for the fist case, and $D_{tf}(\mathcal{E}, \mathcal{A}, \mathcal{C}, *)$ for the second case, must be in the set $P(\mathcal{E})$.

4.2 Delay Tolerant Problems

In the presentation of the composition algorithm, in Section 5.1 of Chapter 5, we restrict our attention to environments that are indifferent to delays. That is, they behave the same regardless of how many rounds of empty ($\perp^n$) outputs are generated by the algorithm between the more informative, non-$\perp^n$, outputs. We can capture this behavior with the following automaton constraints.

Definition 4.2.1 *We say an environment $\mathcal{E}$ is* delay tolerant *if and only if for every state $s \in estates_{\mathcal{E}}$ and $\hat{s} = etrans_{\mathcal{E}}(s, \perp^n)$, the following conditions hold:*

1. $ein_{\mathcal{E}}(\hat{s}) = \perp^n$.
   (If the environment is in a marked *version of state s—i.e., the state, $\hat{s}$, generated when the environment is in some state s, and then receives output $\perp^n$ from the algorithm—the environment always returns $\perp^n$ as input.)*

2. $erand_{\mathcal{E}}(\hat{s})(\hat{s}) = 1$.
   (If the environment is in a marked *state, the environment stays in that state during the probabilistic state transformation at the beginning of each round.)*

3. $etrans_{\mathcal{E}}(\hat{s}, \perp^n) = \hat{s}$.
   (If the environment is in a marked *state and receives $\perp^n$ from the algorithm, it stays in the same state when the transition function is applied at the end of the round.)*

4. *For every non-empty output assignment O, $etrans_{\mathcal{E}}(\hat{s}, O) = etrans_{\mathcal{E}}(s, O)$.*
   (If the environment is in a marked *version of state s, and then receives an output assignment $O \neq \perp^n$, then it transitions as if it were in state s receiving this output—in effect, ignoring the rounds spent marked.)*

A delay tolerant environment behaves in a special manner when it receives output $\perp^n$ from the algorithm. Assume it is in some state s when this output is received. The environment transitions to a special *marked* version of the state, which we denote as $\hat{s}$. It then cycles on this state until it receives a non-$\perp^n$ output from the algorithm. At this point it transitions as if it were in the unmarked state s—effectively ignoring the rounds in which it was receiving empty outputs. That is, whether it was in the marked state for 1 round or 1000 rounds, when it receives a non-$\perp^n$ output, it transitions as it would had it never received the interceding empty inputs.

We use this definition of a delay tolerant environment to define a delay tolerant problem:

Definition 4.2.2 (Delay Tolerant Problem) *We say a problem P is* delay tolerant *if and only if for every non-delay tolerant environment $\mathcal{E}$, $P(E)$ returns the set containing every trace probability function.*

As mentioned, we use these definitions in our presentation of the composition algorithm in Section 5.1. The composition algorithm composes an algorithm that solves a *delay tolerant* problem, with an algorithm that emulates a channel, to form a new algorithm. This composed algorithm simulates an execution of the problem-solving algorithm and the channel emulation algorithm, using the latter to handle the messages from the former. In this simulation, however, the problem-solving algorithm may have to be *paused* while the channel emulation algorithm generates a set of received messages. From the perspective of the environment sharing a system with this composed algorithm, this introduces extra delays where only $\perp^n$ inputs are being generated. Because the problem is delay tolerant, however, we are able to prove, in Theorem 5.1.7, that these added delays do not affect the correctness of the problem-solving algorithm's output.

4.3 Implementing Channels

In this section we define a precise notion of implementing a channel with another channel as a special case of solving a problem. Channel implementation is a crucial behavior in the study of radio networks, as it decomposes the solving of problems into building a better behaved channel, and then solving the problem on this better channel. In real world deployments, for example, this decomposition is captured in the layers of the network stack—the MAC layer being an implementation of a powerful channel with a less powerful channel. When coupled with the composition theorems of Chapter 5, the implementation definitions below allow a formal treatment of this basic behavior.

We begin with some useful notation. Specifically, if an algorithm is going to implement a channel, it needs some way of being passed the messages to be sent, and then some way of returning the corresponding received messages. We use the input and output interface of the algorithm to accomplish this goal. The notation below defines a set of input values, which we call *send-encoded*, that can be used to input the messages to be sent to a channel emulation algorithm. It also defines a set of output values, which we call *receive-encoded*, that can be used to output the corresponding received messages.

- We say an input value $v \in \mathcal{I}_\perp$ is *send-encoded* if and only if $v \in (send \times \mathcal{M}_\perp \times \mathcal{F})$. Note, in the above, *send* is a literal.
- We say an input assignment is send-encoded if and only if all input values in the assignment are send-encoded.
- We say an output value $v \in \mathcal{O}_\perp$ is *receive-encoded* if and only if $v \in (recv \times \mathcal{R}_\perp)$. Note, in the above, *recv* is a literal.
- We say an output assignment is receive-encoded if and only if all output values in the assignment are send-encoded.

We now continue with three components—the channel environment, the channel algorithm, and the channel identity algorithm. The channel environment and the channel identity algorithm will be used below, in Definition 4.3.4, which defines the channel problem corresponding to a given channel. Though the channel algorithm is not necessary for this definition (it is first used in the definitions of Chapter 5), we define it here as it is the counterpart to the channel environment, which *is* used in Definition 4.3.4.

Definition 4.3.1 (Channel Environment) *An environment $\mathcal{E}$ is a channel environment if and only if it satisfies the following conditions:*

1. *It is delay tolerant.*
2. *It generates only send-encoded and empty input assignments.*
3. *It generates a send-encoded input assignment in the first round and in every round $r > 1$ such that it received a receive-encoded output vector in $r - 1$. In every other round it generates an empty input assignment.*

This constraint requires the environment to pass down messages to send as inputs and then wait for the corresponding received messages, encoded as algorithm outputs, before continuing by passing down the next batch messages to send.

The natural counterpart to a channel environment is a channel algorithm, which behaves symmetrically.

Definition 4.3.2 (Channel Algorithm) *We say an algorithm $\mathcal{A}$ is a channel algorithm if and only if it satisfies the following conditions:*

1. *It generates only receive-encoded and empty output assignments.*
2. *It never generates two consecutive receive-encoded output assignments without a send-encoded input in between.*
3. *Given a send-encoded input, it eventually generates a receive-encoded output.*

The channel algorithm responds to each send-encoded input with a receive-encoded output. When paired with a channel environment, the resulting system will generate an alternating sequence of send-encoded inputs and receive-encoded outputs, with an arbitrary number of intervening $\perp^n$ outputs between them.

We continue with a special algorithm, called the *channel identity algorithm*, that simply passes messages back and forth between a channel environment and a channel.

Definition 4.3.3 ($\mathcal{A}^I$) *Each process $\mathcal{A}^I(i)$, $i \in [n]$, of the channel identity algorithm $\mathcal{A}^I$, behaves as follows. If $\mathcal{A}^I(i)$ receives a send-enabled input, $(send, m, f)$, it sends message m on frequency f during that round and generates output $(recv, m')$, where m' is the message it receives in this same round. Otherwise it receives on frequency 1 and generates output $\perp$.*

By combining a channel environment $\mathcal{E}$ with the channel identity algorithm $\mathcal{A}^I$ and a channel $\mathcal{C}$, we get a system that, abstractly speaking, connects $\mathcal{E}$ directly to $\mathcal{C}$. We use observation to define what it means to implement $\mathcal{C}$. Roughly speaking, an algorithm implements $\mathcal{C}$ if its corresponding time-free probability trace function, D_{tf}, behaves the same as the time-free probability function corresponding to $\mathcal{A}^I$ and $\mathcal{C}$.

Definition 4.3.4 (Channel Problem) *For a given channel $\mathcal{C}$ we define the corresponding (channel) problem $P_{\mathcal{C}}$ as follows: $\forall \mathcal{E} \in E$, if $\mathcal{E}$ is a channel environment, then $P_{\mathcal{C}}(\mathcal{E}) = \{F\}$, where, $\forall \beta \in T : F(\beta) = D_{tf}(\mathcal{E}, \mathcal{A}^I, \mathcal{C}, \beta)$. If $\mathcal{E}$ is not a channel environment, then $P_{\mathcal{C}}(\mathcal{E})$ is the set containing every trace probability function.*

The notion of implementing a channel is reduced to solving the corresponding channel problem.

Definition 4.3.5 (Implements) *We say an algorithm $\mathcal{A}$ implements a channel $\mathcal{C}$ using channel $\mathcal{C}'$ only if $\mathcal{A}$ time-free solves $P_{\mathcal{C}}$ using $\mathcal{C}'$.*

Chapter 5

Composition

In this chapter, we prove two useful composition results. The first divides the task of solving a complex problem on a weak channel into first implementing a strong channel using a weak channel, and then solving the problem on the strong channel. The second result simplifies the proofs from Part II that require us to show that the channels implemented by our emulation algorithms, RFC and DFC, satisfy the (t, b, p)-feedback channel property presented in Chapter 6.

5.1 The Composition Algorithm

Assume we have an algorithm $\mathcal{A}_P$ that time-free solves a delay tolerant problem P using channel $\mathcal{C}$, and an algorithm $\mathcal{A}_\mathcal{C}$ that implements channel $\mathcal{C}$ using some other channel $\mathcal{C}'$. In this section we describe how to construct algorithm $\mathcal{A}(\mathcal{A}_P, \mathcal{A}_\mathcal{C})$ that combines $\mathcal{A}_P$ and $\mathcal{A}_\mathcal{C}$. We then prove that this *composition algorithm* solves P using $\mathcal{C}'$.

This result frees algorithm designers from the responsibility of manually adapting their algorithms to work with implemented channels (which introduce unpredictable delays between messages being sent and received). The composition algorithm, and its accompanying theorem, can be viewed as a general strategy for this adaption.

We then generalize this result to a sequence of channel implementation algorithms that start with some weak channel and end with the strong channel needed by P. In the following, the delay tolerance property is crucial for proving that the composition still solves P—the implementation of $\mathcal{C}$ using $\mathcal{C}'$ may introduce a large number of extra rounds in which $\mathcal{A}$ remains idle.

5.1.1 Composition Algorithm Definition

Below we provide a formal definition of our composition algorithm. Though the details are voluminous, the intuition is straightforward. At a high-level, the composition algorithm $\mathcal{A}(\mathcal{A}_P, \mathcal{A}_C)$ calculates the messages generated by $\mathcal{A}_P$ for the current round of $\mathcal{A}_P$ being emulated. It then *pauses* $\mathcal{A}_P$ and executes $\mathcal{A}_\mathcal{C}$ to emulate the messages being sent on $\mathcal{C}$. This may require many rounds (during which the environment is

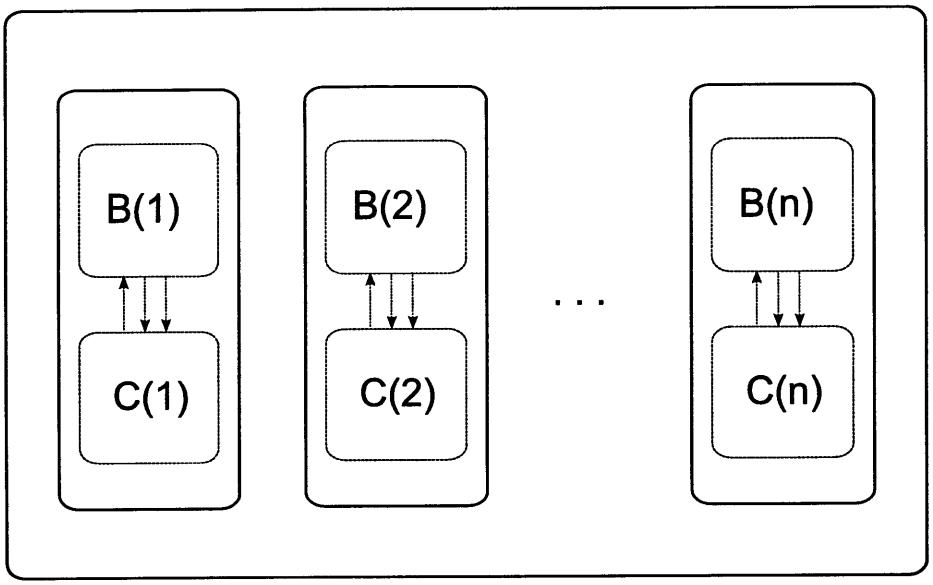

Figure 5-1: The composition algorithm $\mathcal{A}(B, C)$, where B is an algorithm that solves a delay tolerant problem, and C is a channel algorithm that emulates a channel. The outer rectangle denotes the composition algorithm. Each of the inner rectangles is a process of the composition algorithm. Each of these processes, in turn, internally simulates running B and C, which is denoted by the labelled dashed boxes within the processes.

receiving only $\perp^n$ from the composed algorithm—necessitating its delay tolerance property). When $\mathcal{A}_C$ finishes computing the received messages, we *unpause* $\mathcal{A}_P$, and then finish the emulated round by passing these messages to the algorithm. The only tricky point in this construction is that when we pause $\mathcal{A}_P$ we need to also store a copy of its input, as we will need this later to complete the simulated round once we unpause. Specifically, the transition function applied at the end of the round requires this input as a parameter. See Figure 5-1 for a diagram of the composition algorithm.

The formal definition follows.

Definition 5.1.1 (The Composition Algorithm: $\mathcal{A}(\mathcal{A}, \mathcal{A}_C)$) *Let $\mathcal{A}_P$ be an algorithm and $\mathcal{A}_C$ be a channel algorithm.*

Fix any $i \in [n]$. To simplify notation, let $A = \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C)(i)$, $B = \mathcal{A}_P(i)$, and $C = \mathcal{A}_C(i)$. We define process A as follows:

- $\mathbf{states_A} \in states_B \times states_C \times \{active, paused\} \times \mathcal{I}_\perp$.
  Given such a state $s \in states_A$, we use the notation $s.prob$ to refer to the $states_B$ component, $s.chan$ to refer to the $states_C$ component, $s.status$ to refer to the $\{active, paused\}$ component, and $s.input$ to refer to the $\mathcal{I}_\perp$ component.

  The following two helper function simplify the remaining definitions of process

components:

$$siminput(s \in states_A, in \in \mathcal{I}_\perp) = \begin{cases} \perp & \text{if } s.status = paused, \\ (send, m, f) & \text{else,} \end{cases}$$

where $m = msg_B(s.prob, in)$ *and* $f = freq_B(s.prob, in)$.
(This helper function determines the input that should be passed to the C component of the composition process A. If the B component has a message to send, then the function returns this message as a send-encoded input, otherwise it returns $\perp$*.)*

$$simrec(s \in states_A, in \in \mathcal{I}_\perp, m \in \mathcal{R}_\perp) = \begin{cases} \perp & \text{if } o = \perp, \\ m' & \text{if } o = (recv, m'), \end{cases}$$

where $o = out_C(s.chan, siminput(s, in), m)$.
(This helper function determines the message, as generated by the C component of A, that should be received by the B component. If the C component does not have a received message to return—i.e., it is still determining this message—the helper function evaluates to $\perp$*.)*

- $\mathbf{start_A} = (start_B, start_C, active, \perp)$.
  (The B and C components start in their start states, the B component is initially active, and the input component contains only $\perp$*.)*

- $\mathbf{msg_A(s, in)} = msg_C(s.chan, siminput(s, in))$.
  (Process A sends the message generated by the C component.)

- $\mathbf{freq_A(s, in)} = freq_C(s.chan, siminput(s, in))$.
  (As with msg_A*, process A also the frequency generated by the C component.)*

- *The* out_A *function is defined as follows:*

$$\mathbf{out_A(s, in, m)} = \begin{cases} \perp & \text{if } m' = \perp, \\ out_B(s.prob, s.input, m') & \text{if } m' \neq \perp,\ s.status = paused, \\ out_B(s.prob, in, m') & \text{if } m' \neq \perp,\ s.status = active, \end{cases}$$

where $m' = simrec(s, in, m)$.
(The process A will generate $\perp$ *as output unless it is in a round in which the C component is returning a message to the B component—as indicated by simrec. This event indicates that A should* unpause *the B component and complete its simulated round by generating its output with the* out_B *function. It is possible, however, that the C component responds with a message by simrec in the* same *round that it was passed a send-encoded input by siminput. In this case, there is no time to pause the B component—i.e., set status to paused—so its output is generated slightly differently. Namely, in this fast case it is not necessary to*

retrieve the input from the s.input component, as there was no time to store it in that component; the out_B function can simply be passed the input from the current round.)

- *The $rand_A$ distribution is defined as follows:*

$$\mathbf{rand_A(s)(s')} = \begin{cases} p_C & \text{if } s.status = s'.status = paused, s.input = s'.input, \\ & s.prob = s'.prob, \\ p_B \cdot p_C & \text{if } s.status = s'.status = active, s.input = s'.input, \\ 0 & \text{else} \end{cases}$$

where $p_C = rand_C(s.chan)(s'.chan)$ and $p_B = rand_B(s.prob)(s'.prob)$.
(If the B component is paused, then the probability is determined entirely by the probability of the transformation of the C component in s to s'. By contrast, if the B component is active, then we multiple the probabilities of both the B and C component transformations.)

- *Let $\mathbf{trans_A(s, m, in)} = s'$.*
As in our definition of out_A, let $m' = simrec(s, in, m)$. We define the components of state s' below:

  - *$s'.chan = trans_C(s.chan, m, siminput(s, in))$.*
    (The C component transitions according to its transition function being passed: its state, the received message for the round, and the simulated input for the round.)
  - *$s'.prob$ is defined as follows:*

$$s'.prob = \begin{cases} trans_B(s.prob, m', s.input) & \text{if } m' \neq \perp, s.status = paused, \\ trans_B(s.prob, m', in) & \text{if } m' \neq \perp, s.status = active, \\ s.prob & \text{else.} \end{cases}$$

    (The B component transformation depends on whether or not the C component has a receive message to return. If it does not, then the B component remains paused and therefore stays the same. If the C component does have a message to return, it transitions according to its transitions function being passed: the message from C, and the appropriate input. As in the definition of out_A, the input returned depends on whether or not there was time to store it in the input component.)
  - *$s'.input$ is defined as follows:*

$$s'.input = \begin{cases} in & \text{if } s.status = active \\ s.input & \text{else.} \end{cases}$$

(If the B component is active then we store the input from the current round in the input component. Otherwise, we keep the same value in input.)

- *$s'.status$ is defined as follows:*

$$s'status = \begin{cases} active & \text{if } m' \neq \perp, \\ paused & \text{else.} \end{cases}$$

(If the C component has a message to return to the B component, then we can unpause the B component by setting the status component to active. Otherwise we set it to paused.)

In this definition, the process A simulates passing the messages from B through the channel protocol C, pausing B, and storing the relevant input in *s.input*, while waiting for C to calculate the message received for each given message sent.

5.1.2 Composition Algorithm Theorems

We now prove that this composition works (i.e., solves P on $\mathcal{C}'$). Our strategy uses channel-free executions: executions with the channel states removed. We define two functions for extracting these executions. The first, *simpleReduce*, removes the channel states from an execution. The second, *compReduce*, is defined for an execution of the composition algorithm. Given such an execution, it extracts the channel-free execution described by the states of the environment and those in the *prob* component of the composition algorithm states.

Notice, the *comp* in *compReduce* captures the fact that the execution is being extracted from a *comp*osition, though fans of symmetry could also assume it captures the more *complex* nature of the systems with composition.

Definition 5.1.2 (Channel-Free Execution) *We define a sequence α to be a channel-free execution of an environment $\mathcal{E}$ and algorithm $\mathcal{A}$ if and only if there exists an execution α', of a system including $\mathcal{E}$ and $\mathcal{A}$, such that α is generated by removing the channel state assignments from α'.*

Definition 5.1.3 (simpleReduce) *Let $\mathcal{E}$ be a delay tolerant environment, $\mathcal{A}$ be an algorithm, and $\mathcal{C}$ a channel. Let α be an execution of the system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$. Then simpleReduce$(\alpha)$ returns the channel-free execution of $\mathcal{E}$ and $\mathcal{A}$ that is generated by removing the channel state assignments from α.*

Before defining *compReduce*, we introduce a helper function that simplifies discussions of executions that contain the composition algorithm.

Definition 5.1.4 (Emulated Round) *Let $\mathcal{E}$ be an environment, $\mathcal{A}$ be an algorithm, $\mathcal{A}_{\mathcal{C}}$ be a channel algorithm, and $\mathcal{C}'$ a channel. Let α be an execution of the*

system $(\mathcal{E}, \mathcal{A}(\mathcal{A}, \mathcal{A}_C), C')$. *The emulated rounds of* α *are the collections of consecutive real rounds that capture the emulation of a single communication round by* $\mathcal{A}_C$. *Each collection begins with a real round in which siminput returns a send-encoded input for all processes, and ends with the next round in which simrec returns a message at every process, where siminput and simrec are defined in the definition of the composition algorithm (Definition 5.1.1).*

Definition 5.1.5 (compReduce) *Let* $\mathcal{E}$ *be a delay tolerant environment,* $\mathcal{A}_P$ *be an algorithm,* $\mathcal{A}_C$ *be a channel algorithm, and* C' *a channel. Let* α' *be an execution of the system* $(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), C')$. *Then* $compReduce(\alpha')$ *returns the channel-free execution of* $\mathcal{E}$ *and* $\mathcal{A}_P$ *defined as follows:*

1. *Divide* α' *into* emulated rounds.
2. *If* α' *is a finite execution and ends with a partial emulated round, then* $compReduce(\alpha') = null$, *where null is a special marker indicating bad input to the function.*
3. *Else,* $compReduce(\alpha') = \alpha$, *where* α *is a channel-free execution of* $\mathcal{E}$ *and* $\mathcal{A}_P$, *constructed as follows. For each* emulated *round* r *of* α', *add a corresponding* single *round* r *to* α, *where we define the relevant assignments—*$R_r^S, R_r^E, I_r, M_r, F_r, O_r, S_r, E_r$*—of this round as follows:*
   (a) $\forall i \in [n] : R_r^S[i] = S[i].prob$, *where* S *is the first state assignment of the composition algorithm in the first real round of emulated round* r *from* α'. [1]
   (b) $\forall i \in [n] : F_r[i]$ *equals the* $[\mathcal{F}]$ *component, and* $M_r[i]$ *equals the* $\mathcal{M}_\perp$ *component, of* $siminput(S[i], in_i)$, *where* S *is described as in the previous item, and* in_i *is the input received by process* i *at the beginning of the first real round of emulated round* r *from* α'.
   (c) $\forall i \in [n] : N_r[i]$ *equals the* $\mathcal{R}_\perp$ *component of* $simrec(S'[i], in_i, m_i)$, *where* S' *is the first algorithm state assignment,* in_i *is the input received by process* i, *and* m_i *is the message received by* i, *in the last real round of emulated round* r *from* α'.
   (d) I_r *equals the input assignment from the first real round of emulated round* r *from* α'.
   (e) O_r *equals the output assignment from the last real round of emulated round* r *from* α'.
   (f) $\forall i \in [n] : S_r[i] = S''[i].prob$, *where* S'' *is the last algorithm state from the last real round of emulated round* r *from* α'.

[1] Recall, in each round of an execution, there are two state assignments for algorithms, channels, and environments. The first is chosen according to the relevant distribution defined for the final state assignment of the previous round, and the second is the result of applying the transition function to the appropriate assignments in the round.

(g) R_r^E *equals the environment state in the first real round of emulated round* r *from* α'.

(h) E_r *equals the last environment state from the last real round of emulated round* r *of* α'.

At a high-level, the above definition first extracts from α': the environment states, input and output assignments, and the $\mathcal{A}_P$ states encoded in the *prob* component of the composition algorithm. It then cuts out all but the first and last state of the emulated round (which could be the same state if the channel behavior required only a single round to emulate). Finally, it adds the send, frequency, and receive assignments that were used in the emulated round. The resulting channel-free execution captures the behavior of $\mathcal{A}_P$ and $\mathcal{E}$ executing in a system that appears to also include channel $\mathcal{C}$.

We continue with a helper lemma that proves that the execution of $\mathcal{A}_P$ emulated in an execution of a composition algorithm that includes $\mathcal{A}_P$, with the same probability as $\mathcal{A}_P$ running by itself.

Lemma 5.1.6 *Let* $\mathcal{E}$ *be a delay tolerant environment,* $\mathcal{A}_P$ *be an algorithm, and* $\mathcal{A}_C$ *be a channel algorithm that implements* $\mathcal{C}$ *with* $\mathcal{C}'$. *Let* α *be a channel-free execution of* $\mathcal{E}$ *and* $\mathcal{A}_P$. *It follows:*

$$\sum_{\alpha' | simpleReduce(\alpha')=\alpha} Q(\mathcal{E}, \mathcal{A}_P, \mathcal{C}, \alpha') = \sum_{\alpha'' | compReduce(\alpha'')=\alpha} Q(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}', \alpha'')$$

Proof. To simplify notation, let S'_s be the set that contains every execution α' of $(\mathcal{E}, \mathcal{A}_P, \mathcal{C})$ such that $simpleReduce(\alpha') = \alpha$, and let S'_c be the set that contains every execution α'' of $(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}')$ such that $compReduce(\alpha'') = \alpha$. (The inclusion of a prime symbol, $'$, in this notation, is to ensure forward compatibility with the notation from the theorem that follows.)

In the proof that follows, we determine the probability of various executions that result from applying Q to the execution and its system. Recall that $Q(\mathcal{E}, \mathcal{A}, \mathcal{C}, \alpha)$, for a system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$ and execution α of the system, simply multiplies the probabilities of the state transformations that occur at the beginning of each round for each of the automata in the system (the environment, channel, and n processes).

We begin by establishing several facts about S'_s. For every $\alpha' \in S'_s$:

1. The sequence of states of $\mathcal{E}$ in α' is the same as in α. This follows from the definition of *simpleReduce*.

2. The sequence of states of $\mathcal{A}_P$ in α' is the same as in α. This also follows from the definition of *simpleReduce*.

3. It follows from observation 1 that the product of probabilities of environment state transformations is the same in α and α'. Call this product p_E.

4. It follows from observation 2 that the product of probabilities of algorithm state transformations is the same in α and α'. Call this product p_A.

We can use these observations to reformulate the first sum from the lemma statement in a more useful form. From observations 1 and 2, we know that the prefixes in S_s differ only in their channel states. The reason why multiple executions might reduce to the same channel-free execution, by *simpleReduce*, is that it is possible that different channel states might produce the same received messages as in α, given the sent messages and frequencies of α.

With this in mind, we rewrite the first sum from the lemma as:

$$p_E p_A \sum_{\alpha' \in S'_s} p_C(\alpha'),$$

where $p_C(\alpha')$ returns the product of the probabilities of the channel state transformations in α'. By the definition of S'_s, we can also describe $\sum_{\alpha' \in S'_s} p_C(\alpha')$ as follows:

> The probability that $\mathcal{C}$ generates the receive assignments in α, given the message and frequency assignments from α as input.

With this in mind, let $\mathcal{E}_\alpha$ be the simple channel environment that passes down the sequence of send-encoded inputs that match the message and frequency assignments in α. Let β be the trace the encodes the message, frequency, and received message assignments in α as alternating send-encoded inputs and receive-encoded outputs. We can now formalize our above observation as follows:

$$\sum_{\alpha' \in S'_s} p_C(\alpha') = D_{tf}(\mathcal{E}_\alpha, \mathcal{A}^I, \mathcal{C}, \beta)$$

We now establish several related facts about S'_c. For every $\alpha'' \in S'_c$:

5. The sequence of states of $\mathcal{E}$ in α can be generated by taking the sequence of these states from α'', and then potentially removing some of the marked states. (Recall, "marked" is from the definition of delay-tolerant. These extra marked states in α'' correspond to the rounds in which the composition algorithm paused $\mathcal{A}_P$ while running $\mathcal{A}_\mathcal{C}$ on $\mathcal{C}'$ to emulate $\mathcal{C}$.) This follows from the definition of *compReduce*.

6. The sequence of states of $\mathcal{A}_P$ in α can be generated by taking these states encoded in the the *prob* component of the composition algorithm states in α'', and then removing those from a composition algorithm state with *status* = *paused*. This also follows from the definition of *compReduce*.

7. It follows from observation 5, and the fact that a marked state transforms to itself with probability 1, that the product of the environment state transformations probabilities in α'' equal p_E, as in α.

8. To calculate the product of the algorithm state transformation probabilities in α'', we should first review the definition of the state distribution for the composition algorithm. Recall that for such a state, there are two cases. In the first case, *status* = *paused*. Here, the probability of transformation to a new state is determined only by the *chan* component. In the second case, *status* = *active*. Here, the transformation probability is the product of the transformation probabilities of both the *prob* and *chan* components.
   It follows from this fact, and observation 6, that the product of the probabilities of the algorithm state transformations in α'' equal p_A times the product of the probabilities of the $\mathcal{A}_C$ state transformations encoded in the *chan* components.

As when we considered S'_s, we can use these observations to reformulate the second sum from the lemma statement. Specifically, we rewrite it as:

$$p_E p_A \sum_{\alpha'' \in S'_c} p_{C,C'}(\alpha''),$$

where $p_{C,C'}(\alpha'')$ is the product of the state transformation probabilities of both C' and the *chan* component of the composition algorithm. By the definition of S'_c and the composition algorithm, we can also describe $\sum_{\alpha'' \in S'_c} p_{C,C'}(\alpha'')$ as follows:

> The probability that $\mathcal{A}_C$ running on C' outputs the receive assignments in α, given the corresponding message and frequency assignments passed as send-encoded inputs to $\mathcal{A}_C$.

Let $\mathcal{E}_\alpha$ and β be described as above. We can now formalize our above observation as follows:

$$\sum_{\alpha'' \in S'_c} p_{C,C'}(\alpha'') = D_{tf}(\mathcal{E}_\alpha, \mathcal{A}_C, C', \beta)$$

By assumption, $\mathcal{A}_C$ *implements* C with C'. If we unwind the definition of implements (Definition 4.3.5), it follows that $\mathcal{A}_C$ solves the *channel problem* P_C using C'. If we then unwind the definition of this channel problem (Definition 4.3.4), it follows that:

$$D_{tf}(\mathcal{E}_\alpha, \mathcal{A}_C, C', \beta) = D_{tf}(\mathcal{E}_\alpha, \mathcal{A}^I, C, \beta)$$

We combine this equality with our rewriting of the p_C and $p_{C,C'}$ sums from above, to conclude:

$$p_E p_A \sum_{\alpha' \in S'_s} p_C(\alpha') = p_E p_A \sum_{\alpha'' \in S'_c} p_{C,C'}(\alpha'').$$

Because these two terms were defined to be equivalent to the two sums from the lemma statement, we have shown the desired equality. □

We can now prove our main theorem and then a corollary that generalizes the result to a chain of implementation algorithms.

Theorem 5.1.7 (Algorithm Composition) *Let $\mathcal{A}_P$ be an algorithm that time-free solves delay tolerant problem P using channel $\mathcal{C}$. Let $\mathcal{A}_C$ be an algorithm that implements channel $\mathcal{C}$ using channel $\mathcal{C}'$. It follows that the composition algorithm $\mathcal{A}(\mathcal{A}_P, \mathcal{A}_C)$ time-free solves P using $\mathcal{C}'$.*

Proof. By unwinding the definition of *time-free solves*, we rewrite our task as follows:

$$\forall \mathcal{E} \in E, \exists F \in P(\mathcal{E}), \forall \beta \in T : D_{tf}(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}', \beta) = F(\beta).$$

Or, equivalently:

$$\forall \mathcal{E} \in E \ [\lambda \beta \ D_{tf}(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}', \beta) \in P(\mathcal{E})].$$

Fix some $\mathcal{E}$. Assume $\mathcal{E}$ is delay tolerant (if it is not, then $P(\mathcal{E})$ describes every trace probability function, and we are done). Define trace probability function F such that $\forall \beta \in T : F(\beta) = D_{tf}(\mathcal{E}, \mathcal{A}_P, \mathcal{C}, \beta)$. By assumption $F \in P(\mathcal{E})$. It is sufficient, therefore, to show that $\forall \beta \in T : D_{tf}(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}', \beta) = F(\beta) = D_{tf}(\mathcal{E}, \mathcal{A}_P, \mathcal{C}, \beta)$. Fix some β. Below we prove the equivalence. We begin, however, with the following helper definitions:

- Let $ccp(\beta)$ be the set of every channel-free execution α of $\mathcal{E}$ and $\mathcal{A}_P$ such that $term(\alpha) = true$ and $cio(\alpha) = \beta$.[2]
- Let $S_s(\beta)$, for trace β, describe the set of executions included in the sum that defines $D_{tf}(\mathcal{E}, \mathcal{A}_P, \mathcal{C}, \beta)$, and $S_c(\beta)$ describe the set of executions included in the sum that defines $D_{tf}(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}', \beta)$. (The s and c subscripts denote *simple* and *complex*, respectively.) Notice, for an execution to be included in S_c it cannot end in the middle of an emulated round, as this execution would not satisfy *term*.
- Let $S'_s(\alpha)$, for channel-free execution α of $\mathcal{E}$ and $\mathcal{A}_P$, be the set of every execution α' of $(\mathcal{E}, \mathcal{A}_P, \mathcal{C})$ such that $simpleReduce(\alpha') = \alpha$. Let $S'_c(\alpha)$ be the set of every execution α'' of $(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}')$ such that $compReduce(\alpha'') = \alpha$. Notice, for a execution α'' to be included in S'_c, it cannot end in the middle of an emulated round, as this execution would cause *compReduce* to return *null*.

We continue with a series of four claims that establish that $\{S'_s(\alpha) : \alpha \in ccp(\beta)\}$ and $\{S'_c(\alpha) : \alpha \in ccp(\beta)\}$ partition $S_s(\beta)$ and $S_c(\beta)$, respectively.

Claim 1: $\bigcup_{\alpha \in ccp(\beta)} S'_s(\alpha) = S_s(\beta)$.
We must show two directions of inclusion. First, given some $\alpha' \in S_s(\beta)$, we know $\alpha = simpleReduce(\alpha') \in ccp(\beta)$, thus $\alpha' \in S'_s(\alpha)$. To show the other direction,

[2]This requires some abuse of notation as *cio* and *term* are defined for executions, not channel-free executions. These extensions, however, follow naturally, as both *cio* and *term* are defined only in terms of the input and output assignments of the executions, and these assignments are present in channel-free executions as well as in standard execution executions.

we note that given some $\alpha' \in S'_s(\alpha)$, for some $\alpha \in ccp(\beta)$, $simpleReduce(\alpha') = \alpha$. Because α generates β by *cio* and satisfies *term*, the same holds for α', so $\alpha' \in S_s(\beta)$.

Claim 2: $\bigcup_{\alpha \in ccp(\beta)} S'_c(\alpha) = S_c(\beta)$.
As above, we must show two directions of inclusion. First, given some $\alpha'' \in S_c(\beta)$, we know $\alpha = compReduce(\alpha'') \in ccp(\beta)$, thus $\alpha'' \in S'_c(\alpha)$. To show the other direction, we note that given some $\alpha'' \in S'_c(\alpha)$, for some $\alpha \in ccp(\beta)$, $compReduce(\alpha'') = \alpha$. We know α generates β by *cio* and satisfies *term*. It follows that α'' ends with the same final non-empty output as α, so it satisfies *term*. We also know that *compReduce* removes only empty inputs and outputs, so α'' also maps to β by *cio*. Therefore, $\alpha'' \in S_c(\beta)$.

Claim 3: $\forall \alpha_1, \alpha_2 \in ccp(\beta), \alpha_1 \neq \alpha_2 : S'_s(\alpha_1) \cap S'_s(\alpha_2) = \emptyset$.
Assume for contradiction that some α' is in the intersection. It follows that $simpleReduce(\alpha')$ equals both α_1 and α_2. Because *simpleReduce* returns a single channel-free execution, and $\alpha_1 \neq \alpha_2$, this is impossible.

Claim 4: $\forall \alpha_1, \alpha_2 \in ccp(\beta), \alpha_1 \neq \alpha_2 : S'_c(\alpha_1) \cap S'_c(\alpha_2) = \emptyset$.
Follows from the same argument as claim 3 with *compReduce* substituted for *simpleReduce*.

The following two claims are a direct consequence of the partitioning proved above and the definition of D_{tf}:

Claim 5: $\sum_{\alpha \in ccp(\beta)} \sum_{\alpha' \in S'_s(\alpha)} Q(\mathcal{E}, \mathcal{A}_P, \mathcal{C}, \alpha') = D_{tf}(\mathcal{E}, \mathcal{A}_P, \mathcal{C}, \beta)$.

Claim 6: $\sum_{\alpha \in ccp(\beta)} \sum_{\alpha' \in S'_c(\alpha)} Q(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}', \alpha') = D_{tf}(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}', \beta)$.

We conclude by combining claims 5 and 6 with Lemma 5.1.6 to prove that:

$$D_{tf}(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_C), \mathcal{C}', \beta) = D_{tf}(\mathcal{E}, \mathcal{A}_P, \mathcal{C}, \beta),$$

as needed. □

Corollary 5.1.8 (Generalized Algorithm Composition) *Let $\mathcal{A}_{1,2}$, ..., $\mathcal{A}_{j-1,j}$, $j > 2$, be a sequence of algorithms such that each $\mathcal{A}_{i-1,i}$, $1 < i \leq j$, implements channel $\mathcal{C}_{i-1}$ using channel $\mathcal{C}_i$. Let $\mathcal{A}_{P,1}$ be an algorithm that time-free solves delay tolerant problem P using channel $\mathcal{C}_1$.It follows that there exists an algorithm that time-free solves P using $\mathcal{C}_j$.*

Proof. Given an algorithm $\mathcal{A}_{P,i}$ that time-free solves P with channel C_i, $1 \leq i < j$, we can apply Theorem 5.1.7 to prove that $\mathcal{A}_{P,i+1} = \mathcal{A}(\mathcal{A}_{P,i}, A_{i,i+1})$ time-free solves P with channel $\mathcal{C}_{i+1}$. We begin with $\mathcal{A}_{P,1}$, and apply Theorem 5.1.7 $j-1$ times to arrive at algorithm $\mathcal{A}_{P,j}$ that time-free solves P using $\mathcal{C}_j$. □

5.2 The Composition Channel

Given a channel implementation algorithm $\mathcal{A}$ and a channel $\mathcal{C}'$, we define the channel $\mathcal{C}(\mathcal{A}, \mathcal{C}')$. This *composition channel* encodes a local emulation of $\mathcal{A}$ and $\mathcal{C}'$ into its probabilistic state transitions. We formalize this notion by proving that $\mathcal{A}$ implements $\mathcal{C}(\mathcal{A}, \mathcal{C}')$ using $\mathcal{C}'$. To understand the utility of this result, assume you have a channel implementation algorithm $\mathcal{A}$ and you want to prove that $\mathcal{A}$ using $\mathcal{C}'$ implements a channel that satisfies some useful automaton property. (As demonstrated in the next section, it is often easier to talk about all channels that satisfy a property than to talk about a specific channel.) You could apply our composition channel result to establish that $\mathcal{A}$ implements $\mathcal{C}(\mathcal{A}, \mathcal{C}')$ using $\mathcal{C}'$. This reduces the task to showing that $\mathcal{C}(\mathcal{A}, \mathcal{C}')$ satisfies the relevant automaton properties.

5.2.1 Composition Channel Definition

At a high-level, the composition channel $\mathcal{C}(\mathcal{A}, \mathcal{C}')$, when passed a message and frequency assignment, *emulates* $\mathcal{A}$ using $\mathcal{C}'$ being passed these messages and frequencies as input and then returning the emulated output from $\mathcal{A}$ as the received messages. This emulation is encoded into the *crand* probabilistic state transition of $\mathcal{C}(\mathcal{A}, \mathcal{C}')$. To accomplish this feat, we have define two types of states: *simple* and *complex*. The composition channel starts in a simple state. The *crand* distribution always returns complex states, and the *ctrans* transition function always returns simple states, so we alternate between the two. The simple state contains a component *pre* that encodes the history of the emulation of $\mathcal{A}$ and $\mathcal{C}'$ used by $\mathcal{C}(\mathcal{A}, \mathcal{C}')$ so far. The complex state also encodes this history in *pre*, in addition it encodes the next randomized state transitions of $\mathcal{A}$ and $\mathcal{C}'$ in a component named *ext*, and it stores a table, encoded in a component named *oext*, that stores for each possible pair of message and frequency assignments, an emulated execution that extends *ext* with those messages and frequencies arriving as input, and ending when $\mathcal{A}$ generates the corresponding received messages. The *crecv* function, given a message and frequency assignment and complex state, can look up the appropriate row in *oext* and return the received messages described in the final output of this extension. This approach of simulating execution extensions for all possible messages in advance is necessitated by the fact that the randomized state transition occurs before the channel receives the messages being sent in that round. See Figure 5-2 for a diagram of the composition channel.

Below we provide a collection of helper definitions which we then use in the formal definition of the composition channel.

Definition 5.2.1 (*toinput*) *Let the function toinput map pairs from* $(\mathcal{M}_\perp)^n \times [\mathcal{F}]^n$ *to the corresponding send-encoded input assignment describing these messages and frequencies.*

Definition 5.2.2 (Environment-Free Execution) *We define a sequence of assignments* α *to be an environment-free execution of a channel algorithm* $\mathcal{A}$ *and channel* $\mathcal{C}$*, if and only if there exists an execution* α'*, of a system including* $\mathcal{A}$*,* $\mathcal{C}$*, and*

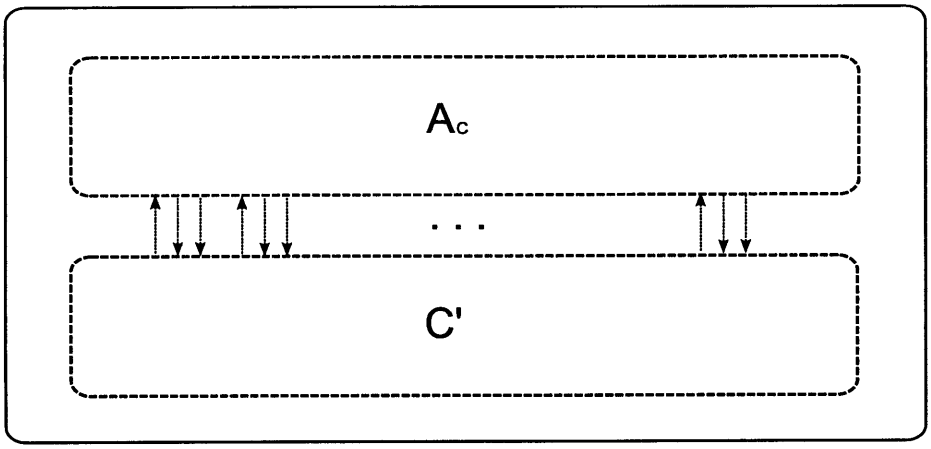

Figure 5-2: The composition channel $\mathcal{C}(A_C, C')$, where A_C is a channel algorithm and C' is a channel. The outer rectangle denotes the composition channel. The A_C and C' dashed rectangles inside the algorithm capture the fact that the composition channel internally simulates running A_C on C'.

a channel environment, such that α is generated by removing the environment state assignments from α'. If α is finite then the final output assignment in α must be receive-encoded.

Definition 5.2.3 (State Extension) *Let α be a finite environment-free execution of some channel algorithm $\mathcal{A}$ and channel $\mathcal{C}$. We define a state extension of α to be α extended by any R^S, R^C, where $\forall i \in [n] : R^S[i] \in states_{\mathcal{A}(i)}$, $R^C \in cstates_{\mathcal{C}}$, and the final process and channel state assignments of α can transform to R^S and R^C with non-0 probability by $rand_{\mathcal{A}}$ and $crand_{\mathcal{C}}$, respectively.*

In other words, we extend the execution α by the next states of the channel and algorithm. We continue, next, with a longer extension.

Definition 5.2.4 (I-Output Extension) *Let α be a finite environment-free execution of some channel algorithm $\mathcal{A}$ and channel $\mathcal{C}$. Let α' be a state extension of α. We define an I-output extension of α', for some $I \in (\mathcal{I}_\perp)^n$, to be any extension of α' that has input I in the first round of the extension, an empty input assignment (i.e., $\perp^n$) in every subsequent round, and that ends in the first round with a receive-encoded output assignment, thus forming a new environment-free execution.*

In other words, we extend our state extension with a particular input, I, after which we run it with empty inputs until it returns a receive-encoded output assignment.

We can now provide the formal definition of the composition channel:

Definition 5.2.5 (The Composition Channel: $\mathcal{C}(\mathcal{A}, C')$) *Let $\mathcal{A}$ be a channel algorithm and C' be a channel. To simplify notation, let $C = \mathcal{C}(\mathcal{A}, C')$. We define the composition channel C as follows:*

1. $\mathbf{cstates_C} = simpleStates_C \cup complexStates_C$,
   where $simpleStates_C$ *and* $complexStates_C$ *are defined as follows:*

   - *The set* $simpleStates_C$ *(of* simple states*) consists of one state for every finite environment-free execution of* $\mathcal{A}$ *and* C'.
   - *The set* $complexStates_C$ *(of* complex states*) consists of one state for every combination of: a finite environment-free execution* α *of* $\mathcal{A}$ *and* C'*; a state extension* α' *of* α*; and a table with one row for every pair* $(M \in (\mathcal{M}_\perp)^n, F \in [\mathcal{F}]^n)$*, such that this row contains an* $(toinput(M,F))$*-output extension of* α'.

   Notation: *For any simple state* $s \in simpleStates_C$*, we use the notation* $s.pre$ *to describe the environment-free execution corresponding to* s*. For any complex state* $c \in complexStates_C$*, we use* $c.pre$ *to describe the environment-free execution,* $c.ext$ *to describe the state extension, and* $c.oext(M,F)$ *to describe the* $toinput(M,F)$*-output extension, corresponding to* c.

2. $\mathbf{cstart_C} = s_0 \in simpleStates_C$,
   where $s_0.pre$ *describes the* 0*-round environment-free execution of* $\mathcal{A}$ *and* C'*. (That is, it consists of only the start state assignment for* $\mathcal{A}$ *and start channel assignment for* C'*.)*

3. $\mathbf{crand_C}(\mathbf{s} \in \mathbf{simpleStates_C})(\mathbf{q} \in \mathbf{complexStates_C})$
   is defined as follows:

   - *If* $q.pre \neq s.pre$*, then* $crand_C(s)(q) = 0$.
   - *Else,* $crand_C(s)(q)$ *equals the product of the probability, for each row* $q.oext(M,F)$ *in* $q.oext$*, that* $q.pre$ *extends to* $q.oext(M,F)$*, given input* $toinput(M,F)$.

   By contrast, $crand_C(s' \in cstates_C)(s \in simpleStates_C) = 0$*. (That is,* $crand_C$ *assign probability mass to complex states only.)*

4. $\mathbf{ctrans_C}(\mathbf{q} \in \mathbf{complexStates_C}, \mathbf{M}, \mathbf{F}) = s \in simpleStates_C$,
   where $s.pre = q.oext(M,F)$.

   Notice that we do not need to define $ctrans_C$ *for a simple state, as our definition of* $crand_C$ *prevents the function from ever being applied to such a state.*

5. $\mathbf{crecv_C}(\mathbf{q} \in \mathbf{complexStates_C}, \mathbf{M}, \mathbf{F}) = N$,
   where N *is contains the messages from the receive-encoded output of the final round of* $q.oext(M,F)$.

   As with $ctrans_C$*, we do not need to define* $crecv_C$ *for a simple state, as our definition of* $crand_C$ *prevents the function from ever being applied to such a state.*

5.2.2 Composition Channel Theorems

To prove that $\mathcal{A}$ implements $\mathcal{C}(\mathcal{A}, \mathcal{C}')$ on $\mathcal{C}'$, we begin with a collection of helper definitions and lemmas.

Definition 5.2.6 (*ex*) *Let α be an execution of system $(\mathcal{E}, \mathcal{A}^I, \mathcal{C}(\mathcal{A}, \mathcal{C}'))$, where $\mathcal{E}$ is a channel environment, $\mathcal{A}$ is a channel algorithm, and $\mathcal{C}'$ is a channel. We define $ex(\alpha)$ to return the execution α', of system $(\mathcal{E}, \mathcal{A}, \mathcal{C}')$, defined as follows:*

1. *We construct α' in increasing order of rounds. Start by setting round 0 to contain the starts states of $\mathcal{E}$, $\mathcal{A}$, and $\mathcal{C}'$.*
2. *We then proceed, in increasing order, through each round $r > 0$ of α, expanding α' as follows:*
   (a) *Add the sequence of algorithm and channel states that results from taking $R_r^C.oext(M_r, F_r)$ and then removing the prefix $R_r^C.pre$, where R_r^C is the random channel state in round r of α, and M_r and F_r are the message and frequency assignments from this round, respectively.*
   (b) *Add R_r^E as the random environment state in the first round of this extension, where R_r^E is the random environment state in round r of α.*
   (c) *Add E_r as the second environment state in the last round of the extension, where E_r is the second environment state in round r of α (i.e., the result of applying the environment transition function to R_r^E.)*
   (d) *Add $\hat{R}_r^E$ as the environment state for all other positions and rounds in the extension, where $\hat{R}_r^E$ is the* marked *version of state R_r^E, where "marked" is defined in the definition of delay tolerant. (Recall, $\mathcal{E}$ is a channel environment which implies that it is delay tolerant.)*

That is, we extract the simulated execution of $\mathcal{A}$ on $\mathcal{C}'$ encoded in the composition channel in α, and then add in the states of $\mathcal{E}$ from α, using marked states to fill in the environment state gaps for the new rounds added by the extraction.

The following definition uses *ex* to capture a key property about the relationship between systems with channel algorithms and systems with those algorithms encoded in a composition channel.

Definition 5.2.7 (*comp*) *Let $\mathcal{E}$ be a channel environment, $\mathcal{A}$ be a channel algorithm, $\mathcal{C}'$ be a channel, and α' be an execution of the system $(\mathcal{E}, \mathcal{A}, \mathcal{C}')$. Let X be the set of all executions of $(\mathcal{E}, \mathcal{A}^I, \mathcal{C}(\mathcal{A}, \mathcal{C}'))$. Then $comp(\alpha') = \{\alpha : \alpha \in X, ex(\alpha) = \alpha'\}$.*

It might seem surprising that multiple executions of $(\mathcal{E}, \mathcal{A}^I, \mathcal{C}(\mathcal{A}, \mathcal{C}'))$, can expand to α' by *ex*, as *ex* is deterministic—it simply extracts an environment-free executions encoded in the $\mathcal{C}(\mathcal{A}, \mathcal{C}')$ state, and then adds environment states in a fixed manner. The explanation, however, concerns the *unused* rows of the *oext* table in the states of $\mathcal{C}(\mathcal{A}, \mathcal{C}')$. For every execution α of $(\mathcal{E}, \mathcal{A}^I, \mathcal{C}(\mathcal{A}, \mathcal{C}'))$ that generates α' by *ex*, the rows

in the *oext* table that correspond to the messages and frequencies in α', are the same. These are the rows that ex uses to construct its expanded execution. The other rows, however, which are not used, can be different, as they are discarded by ex.

We proceed by proving an important lemma about the probabilities associated with states that share a given row entry. Let s be a simple state. Let X be the set of complex states that are compatible with s (that is, their *pre* components match $s.pre$), and all have the same output extension α'' in a fixed *oext* row. The lemma says that the probability that s transitions to a state in X, by the composition channel state distribution $crand(s)$, equals the probability of $s.pre$ extending to α''.

Lemma 5.2.8 *Let $\mathcal{A}$ be a channel algorithm, $\mathcal{C}'$ be a channel, M be a message assignment, F be a frequency assignment, α be a finite environment-free execution of $\mathcal{A}$ and $\mathcal{C}'$, α' be a state extension of α, α'' be a $toinput(M,F)$-output extension of α', s be the simple state of $\mathcal{C}(\mathcal{A},\mathcal{C}')$ with $s.pre = \alpha$, and:*

$$X = \{s' \in cstates_{\mathcal{C}(\mathcal{A},\mathcal{C}')} : s' \text{ is complex}, \ s'.pre = \alpha, s'.oext(M,F) = \alpha''\}.$$

It follows:

$$\sum_{s' \in X} crand_{\mathcal{C}(\mathcal{A},\mathcal{C}')}(s)(s') = Pr[\alpha''|\alpha],$$

where $Pr[\alpha''|\alpha]$ is the probability that $\mathcal{A}$ using $\mathcal{C}'$ extend α to α'', given the input assignments in α''.

Proof. Let p_{ext} describe the probability that $\mathcal{A}$ using $\mathcal{C}'$ extends α to α'. (That is, p_{ext} is the product of the probabilities assigned to algorithm and channel states added to α in α'. Recall, these probabilities are determined by the state distributions corresponding to the final algorithm and channel states in α.)

Also recall that every complex state has an *oext* table with one row for every possible message and frequency assignment pair. Label the non-(M,F) rows in the *oext* table with incrementing integers, $j = 1, ...$; i.e., there is one *row index* j for every row except the (M,F) row.

For every such row j, number the possible I_j-output extensions of α' with incrementing integers, $k = 1, ...$, where I_j is the message and frequency assignment pair corresponding to row j. And let $p_{j,k}$, for some row index j and output extension index k, equal the probability that $\mathcal{A}$ and $\mathcal{C}$ produce extension k of α', given input I_j. (As with p_{ext}, this probability is the product of the probabilities of the algorithm and channel state transformations in the extension).

Because we do not have a j defined for row (M,F), we separately fix p_{fix} to describe the probability that $\mathcal{A}$ using $\mathcal{C}$ extends α' to α'', given the input $toinput(M,F)$

For any fixed j, we know:

$$\sum_k p_{j,k} = 1. \tag{5.1}$$

This follows from the constraint that $\mathcal{A}$ is a channel algorithm. Such an algorithm, when passed a send-encoded input, must eventually return a receive-encoded output in every extension.

To simplify our use of the $p_{j,k}$ notation, let Y be a set of vectors that have one position for every row index, j. Let each entry contain an extension index, k, for that row. Fix Y such that it contains every such vector. Each $\overline{v} \in Y$, therefore, describes a unique configuration for the non-(M, F) rows in the *oext* table, and Y contains a vector for every such unique configuration.

Using our new notation, and the definition of *crand* for a composition channel, we can rewrite our sum from the lemma statement as follows:

$$\sum_{s' \in X} crand_{C(\mathcal{A},C')}(s)(s') = p_{ext}p_{fix} \sum_{\overline{v} \in Y} \prod_{j} p_{j,\overline{v}[j]}.$$

To understand the right-hand side of this equation, recall that $crand_{C(\mathcal{A},C')}(s)(s')$, for any $s' \in X$, returns the product of the probabilities, for each row in $s'.oext$, that $\mathcal{A}$ and C' extend α to the extension in that row, given the corresponding input. Since every extension of α in $s'.oext$ starts with α', we pull out from the product, the probability, p_{ext} of α'. And because every state in s' has the same extension in the (M, F) row, we can pull the probability of that extension, p_{fix}, from the product as well.

To simplify this sum, we note that the sum of products, $\sum_{\overline{v} \in Y} \prod_j p_{j,\overline{v}[j]}$, consists of exactly one term of the form $p_{1,*}p_{2,*}....$, for each unique combination of extension indices. Applying some basic algebra, we can therefore rewrite this sum of products as the following product of sums:

$$\sum_{\overline{v} \in Y} \prod_{j} p_{j,\overline{v}[j]} = (p_{1,1} + p_{1,2} + ...)(p_{2,1} + p_{2,2} + ...)...$$

We apply Equation 4.1 from above, to reduce this to $(1)(1)... = 1$. It follows that:

$$\sum_{s' \in X} crand_{C(\mathcal{A},C')}(s)(s') = p_{ext}p_{fix}.$$

Finally, we note that $p_{ext}p_{fix}$ matches our definition of $Pr[\alpha''|\alpha]$ from the lemma statement, completing the proof. □

This lemma is used in the proof of the following result, as well as in our proof for the *RFC* channel algorithm in Chapter 8.

Lemma 5.2.9 *Let α' be an execution of a system $(\mathcal{E}, \mathcal{A}, C')$, where $\mathcal{E}$ is a channel environment, $\mathcal{A}$ is a channel algorithm, C' is a channel, and the final output in α' is receive-encoded. It follows:*

$$Q(\mathcal{E}, \mathcal{A}, C', \alpha') = \sum_{\alpha \in comp(\alpha')} Q(\mathcal{E}, \mathcal{A}^I, C(\mathcal{A}, C'), \alpha)$$

Proof. Divide α' into execution fragments, each beginning with a round with a send-encoded input and ending with the round containing the corresponding receive-encoded output. We call these fragments, in this context, *emulated rounds.* (This is

similar to how we defined the term with respect to a channel algorithm execution in the composition algorithm section.) By our assumption that α' ends with a receive-encoded output, no part of α' falls outside of an emulated round.

Fix some emulated round e_r of α'. Let I be the send-encoded input passed down by the environment during e_r. Let M and F be the message and frequency assignments encoded in I (i.e., $I = toinput(M, F)$).

To simplify notation, in the following we use $\mathcal{C}$ as shorthand for the composition channel $\mathcal{C}(\mathcal{A}, \mathcal{C}')$. Let s be the simple state of $\mathcal{C}$ that encodes in $s.pre$ the environment-free execution obtained by removing the environment state assignments from the execution of α' through emulated round $e_r - 1$. Let X be the set containing every complex state q such that: $q.pre = s.pre$ and $q.oext(M, F)$ extends $q.pre$ as described by e_r. (There is exactly one such extension for $q.oext(M, F)$. There can be multiple complex states including this extension, however, because the entries can vary in the other rows of $oext$.) Let p describe the probability that $\mathcal{A}$ using $\mathcal{C}'$ extends $s.pre$ to $q.oext(M, F)$, given input I.

We can apply Lemma 5.2.8 for $\mathcal{A}$, $\mathcal{C}'$, M, F, $\alpha = s.pre$, s, $\alpha'' = q.oext(M, F)$, and X, to directly prove the following claim:

Claim: $\sum_{q \in X} crand_{\mathcal{C}}(s)(q) = p$

We now apply this claim, which concerns only a single emulated round, to prove the lemma, which concerns the entire execution α'.

We use induction on the emulated round number of α'. Let R be the total number of emulated rounds in α'. Let $\alpha'[r]$, $0 \leq r \leq R$, describe the prefix of α' through emulated round r. Notice, because we assumed that α' ends with a receive-encoded output: $\alpha'[R] = \alpha'$. Our hypothesis for any emulated round $r \leq R$ states:

$$Q(\mathcal{E}, \mathcal{A}, \mathcal{C}', \alpha'[r]) = \sum_{\alpha \in comp(\alpha'[r])} Q(\mathcal{E}, \mathcal{A}^I, \mathcal{C}, \alpha)$$

We now prove our inductive step. Assume our hypothesis holds for some $r < R$. Every execution in $comp(\alpha'[r])$ concludes with the same simple channel state s_r, where $s_r.pre$ describes the environment-free execution generated by removing the environment assignment states from $\alpha'[r]$.

We know the probability that $\mathcal{E}$ passes down $I = toinput(M, F)$ to begin the next emulated round of α' is the same as the probability that it passes down I in round $r + 1$ of any of the executions in $comp(\alpha'[r])$. This follows from the delay-tolerance of $\mathcal{E}$, which has it behave the same upon receiving a given receive-encoded output, regardless of the pattern or preceding empty outputs.

Finally, by applying the above claim, we determine that given a execution that ends in s_r, the probability that it transform by $crand_{\mathcal{C}}$ to a state q, such that $q.oext(M, F) = \alpha'[r + 1]$, equals the probability that $\alpha'[r]$ transforms to $\alpha'[r + 1]$, given input I. This combines to prove the inductive step.

We conclude the proof by noting that the base case follows from the fact that the probability of $\alpha[0]$ and $comp(\alpha[0])$ is 1 in both systems. □

Theorem 5.2.10 (Channel Composition) *Let $\mathcal{A}$ be a channel algorithm and $\mathcal{C}'$ be a channel. It follows that $\mathcal{A}$ implements $\mathcal{C}(\mathcal{A},\mathcal{C}')$ using $\mathcal{C}'$.*

Proof. By unwinding the definition of *implements*, we can rewrite the theorem statement as follows: for every channel environment $\mathcal{E}$ and trace $\beta \in T$:

$$D_{tf}(\mathcal{E},\mathcal{A},\mathcal{C}',\beta) = D_{tf}(\mathcal{E},A^I,\mathcal{C}(\mathcal{A},\mathcal{C}'),\beta)$$

Fix one such channel environment $\mathcal{E}$. To prove our above equality, it is sufficient to show that for every $\beta \in T$, the two trace probability functions return the same probability. We first introduce some simplifying notation: $S_{comp} = (\mathcal{E},A^I,\mathcal{C}(\mathcal{A},\mathcal{C}'))$, and $S = (\mathcal{E},\mathcal{A},\mathcal{C}')$. We now rewrite our equality regarding D^{tf} in terms of Q:

$$\forall\beta \in T: \quad \sum_{\alpha'|term(\alpha')\wedge cio(\alpha')=\beta} Q(S,\alpha') \quad = \quad \sum_{\alpha|term(\alpha)\wedge cio(\alpha)=\beta} Q(S_{comp},\alpha)$$

For simplicity, we will call the $Q(S,*)$ sum the *first sum* and the $Q(S_{comp},*)$ sum the *second sum.* We restrict our attention to traces that end with a non-empty output, as any other trace would generate 0 for both sums. Fix one such trace β. For this fixed β, consider each α' included in the first sum. (By the definition of *term*, each such α' must also end with a non-empty output.) By Lemma 5.2.9, we know:

$$Q(S,\alpha') \quad = \quad \sum_{\alpha\in comp(\alpha')} Q(S_{comp},\alpha)$$

Recall that $\alpha \in comp(\alpha') \Rightarrow cio(\alpha') = cio(\alpha)$ and $term(\alpha) = true$, so each execution in our *comp* set is included in the second sum.

We next note that for every pair of executions α'_1 and α'_2 of S, such that $\alpha'_1 \neq \alpha'_2$: $comp(\alpha'_1) \cap comp(\alpha'_2) = \emptyset$. In other words, each execution included from S is associated with a *disjoint* set of matching executions from S_{comp}. To see why, assume for contradiction that there exists some $\alpha \in comp(\alpha'_1) \cap comp(\alpha'_2)$. It follows that $ex(\alpha)$ equals both α'_1 and α'_2. However, because ex is deterministic, and $\alpha'_1 \neq \alpha'_2$, this is impossible.

It follows that for each α' included in the first sum there is a collection of executions included in the second sum that add the same probability mass. Furthermore, none of these collections overlap.

To prove that the probability mass is exactly equal, we are left only to argue that every execution included in the second sum is covered by one of these *comp* sets. Let α be a execution included in the second sum. We know that $cio(\alpha) = \beta$ and $term(\alpha) = true$, therefore the same holds of $ex(\alpha)$ which implies that α is covered by $comp(ex(\alpha))$. □

We conclude with a theorem that helps apply properties proved for a channel algorithm, to the emulation of the algorithm encoded in a composition channel. First, however, we need a simple helper definition:

Definition 5.2.11 (Reachable) *We say an environment-free execution is* reachable *if and only if the product of the algorithm and channel transformation probabilities is non-zero. This is, it encodes probabilistic transformations of these states that could occur in an execution with probability greater than* 0.

We use this definition in the following theorem.

Theorem 5.2.12 *Let $\mathcal{A}$ be channel algorithm, $\mathcal{C}'$ be a channel, and s be a state of the composition channel $\mathcal{C}(\mathcal{A}, \mathcal{C}')$. Let α be an environment-free execution encoded either in s.pre or in an entry of the table s.oext (this latter case requires that s is complex).*
If α is reachable, then there exists a channel environment $\mathcal{E}$ such that we can add states from $\mathcal{E}$ to α to generate a valid execution of $(\mathcal{E}, \mathcal{A}, \mathcal{C}')$.

Proof. We choose $\mathcal{E}$ to be a simple channel environment that passes down the sequence of input assignments in α, cycling on marked states while waiting for the next non-empty output in α. □

Chapter 6

Channels

In this chapter, we tackle the challenge of defining channels. We provide definitions for four types of channels. Two we study in the remainder of this thesis, and two are formalizations of popular models used in the existing literature.

6.1 Preliminaries

In our modeling framework, an algorithm designer can construct an algorithm to work for a specific channel automaton, $\mathcal{C}$. This designer could then prove correctness for any system that includes $\mathcal{C}$. Because our model does not include non-determinism, however, it is sometimes useful to instead prove that an algorithm works with *any* channel from a set of channels that all satisfy some constraints. We call such sets *channel properties*, which we define as follows:

Definition 6.1.1 (Channel Property) *A channel property X is a set of channels.*

We also introduce the corresponding notion of satisfying a property:

Definition 6.1.2 (Satisfy a Channel Property) *We say a channel $\mathcal{C}$ satisfies channel property X if and only if $\mathcal{C} \in X$.*

One might expect a channel property to be defined with respect to traces describing send and receive behavior. As we will demonstrate with the properties defined in this chapter, however, it is often more useful to define a property with respect to a set of restrictions on the components of the channel automaton.

Specifically, in this chapter, we define four useful channel properties. We begin with a pair of *basic* properties that capture some common assumptions about the radio models used in the existing theory literature. Though we do not use these channel properties in the chapters that follow, we present them here to highlight the flexibility of our framework and the ease with which it can formalize existing work. We then proceed with a pair of *advanced* properties that will become the focus of the remainder of this thesis.

All of four of our channel properties assume $\mathcal{F} > 0$ communication frequencies. In each round, each process can tune into a single frequency and then decide to either

broadcast or receive. This is motivated by the real world observation that most radio networks are comprised of multiple frequencies that the devices can switch between. The *802.11* standard [1], for example, allows radios to switch between around a dozen frequencies carved out of the unlicensed 2.4 Ghz band, while bluetooth radios [11] can switch between close to 75 frequencies, carved out of this same band.

Most radio models in the existing theoretical literature fix $\mathcal{F} = 1$. In recent work, however, we have been among a growing numbers of researchers who argue that taking advantage of the *multiple* frequencies that are actually available in practice can increase performance and help solve otherwise unsolvable problems; c.f., [31, 43, 32, 40, 30].

In addition, the channel properties defined in this thesis all capture single-hop radio channels. In Section 6.4, however, we describe how they might be extended to a multi-hop setting.

We begin, below, by providing informal descriptions of all four radio channel properties considered in this chapter. Once the informal descriptions are established, we proceed with the formal definitions.

6.2 Summary of Channel Properties

This chapter considers two *basic* channel descriptions—the partial collision and total collision properties–and two *advanced* descriptions—the t-disrupted and (t, b, p)-feedback properties. We summarize the behavior of all four below. In Sections 6.3, 6.4, and 6.5, we provide their formal definitions.

The Partial Collision Property. Channels that satisfy the partial collision property match common expectations regarding a radio channel, namely:

1. At most one message can be received per round.
2. Senders receive their own message.
3. All receivers on the same frequency receive the same thing in a given round (i.e., either they all detect silence, indicated by $\perp$, or receive the same message).
4. If m is the only message sent on a frequency in a given round, then all receivers on that frequency receive m.
5. If multiple messages are sent on a frequency, the receivers on that frequency might all receive nothing, due to collision, or they might all receive one of the messages.

The term *partial* in the property name is motivated by condition 5, which specifies that collisions *can* occur, but are not guaranteed. Therefore, the model is partially collision-prone. (The usage of the term *partial* in the context of collisions was first introduced in [20]).

This property matches the assumptions of the channel used in the broadcast lower bound of BGI [8], which demonstrated networks of diameter 3 that required a linear number of rounds to disseminate a message from a distinguished source.

This paper stood alone in its inclusion of condition 5 from above—most papers that followed this seminal work replaced this condition with an insistence that 2 or more messages sent simultaneously *must* cause a collision. More recently, however, we have argued [20, 22, 72, 21] that the partial collision model is more realistic, as in practice, radios can *capture* and successfully decode one message from among many simultaneous transmissions, and, in general, introducing more non-deterministic behaviors in a model strengthens upper bound results.

We capture this stronger collision assumption with the next property:

The Total Collision Property. The total collision property shares conditions 1–4 with the partial collision property, but replaces condition 5 with the following:

5. If multiple messages are sent on a frequency, the receivers on that frequency all receive nothing, due to collision.

In [58], the authors reconsidered the lower bound of [8] in the context of a channel that satisfied the total collision property. They showed that in this new case, the bound could be exponentially improved from linear to logarithmic, which demonstrates the somewhat counterintuitive reality that insisting on message loss makes the model more powerful. (The proof [58] takes advantage of this constraint to emulate a type of reliable collision detection that aided their efficient results.)

As mentioned, most existing studies of radio networks assume some variant of either the partial or total collision properties (typically with $\mathcal{F} = 1$); c.f., [17, 6, 35, 37, 25, 70, 56, 57, 29, 18, 24].

In addition to the common pair of properties described above, we also define two novel properties. These advanced properties differ from the basic properties above in that they allow for unpredictable (and perhaps even adversarial) interference in the network.

The t-Disrupted Property. We first introduced the t-disrupted radio channel in [31], and then expanded on it in [32, 40, 30]. This property is similar to the partial collision property, but it includes the additional condition that up to t frequencies per round can be *disrupted*, which causes all receivers on those frequencies to receive nothing. We assume t is a known constant less than $\mathcal{F}$.

This disruption *could* model the behavior of a literal adversarial device trying to jam the network and prevent distributed coordination and communication. As described in the introduction, however, it is probably more useful as a convienent way to model the wide diversity of unpredictable sources of radio network disruption—from unrelated protocols using the same bandwidth, to unexpected interference from devices such as microwaves, radars, and lighting.

An important assumption when describing such adversarial behavior is the relevant dependencies. In our case, we must specify the dependence on the channels'

choice of frequencies to disrupt and the frequencies chosen for use by the algorithm, in a given round. We require that the choice of frequencies to disrupt in some round r of an execution must be made without advanced knowledge of the processes' broadcast behavior during round r. The choice can, however, depend on the definition of the processes and the history of the broadcast behavior through round $r-1$. If the processes are deterministic, this is enough information for the channel to calculate the upcoming behavior.

For example, if a process selects a frequency at random on which to participate, the channel cannot know in advance which frequency that channel chose. This gives the process at least a $\frac{\mathcal{F}-t}{\mathcal{F}}$ probability of selecting a non-disrupted frequency. If, on the other hand, we allowed the channel to know the broadcast behavior before selecting frequencies to disrupt, it could always disrupt this process's frequency, regardless of its random choice. (As you will encounter in the formal definitions below, part of the utility of our model is its ability to precisely such notions of dependence.)

Solving problems on a t-disrupted channel is complicated, as indicated by the length of the algorithms and proofs in [31, 32, 40, 30]. With this in mind, in this thesis we also define the following more powerful channel that overcomes much of the complexities of the t-disrupted property. This property is not meant to correspond to channel behavior observed in the real world. Instead, it captures behavior we later implement with the channel emulation algorithms of Part II, to provide a more attractive programming layer for algorithm designers:

The (t, b, p)-Feedback Channel. The (t, b, p)-feedback channel, for $0 \leq t < \mathcal{F}$, $0 \leq b \leq n$, $0 < p \leq 1$, behaves like a t-disrupted channel that has been enhanced such that *most* processes, be them senders or receivers, receive *feedback* about what happened on *all* the frequencies. Specifically, in each round there are two possible values that can be received: $\perp$ or a *feedback* vector, R, containing a value from $\mathcal{R}_\perp$ for each of the $\mathcal{F}$ frequencies. For each $f \in [\mathcal{F}]$, $R[f]$ describes what a receiver on frequency f would have received had the processes that broadcast in this round been broadcasting on a t-disrupted channel.

A perfect feedback channel would guarantee that all processes receive feedback in all rounds, with probability 1. In practice, however, it can be difficult to achieve such guarantees. This motivates the inclusion of the b and p terms, which are interpreted as: the channel guarantees that with probability p, at least $n-b$ processes receive the feedback.

It is important to emphasize that the only two possible outcomes for a process is to receive the common feedback vector or $\perp$. The low probability event is *not* receiving a malformed vector, it describes, instead, the case where more than b processes receive $\perp$ instead of feedback.

As mentioned, the (t, b, p)-feedback channel *does not* correspond to an actual radio scenario. Its definition is introduced for the first time in this thesis. Our goal in Part II is to design algorithms that *implement* this idealized channel using the realistic (but difficult to program) t-disrupted channel. Later, in Part III, we design solutions to common problems using the feedback channel. These solutions demonstrate how the

ability to gain feedback simplifies algorithm design in a setting with unpredictable frequency disruption. By applying our composition results of the previous chapters, these solutions are then automatically adopted to work in the more inhospitable t-disrupted setting.

We chose the (t, b, p)-feedback channel as the appropriate high-level channel to implement with a t-disrupted channel because our experience working with the latter taught us that the most solutions in this setting require processes to disseminate information about who has and has not succeeded in broadcasting on a non-disrupted frequency. This information is often then used to schedule the next group of broadcasters. The feedback channel handles the details of this dissemination, allowing the algorithm designer to focus on the issues unique to the problem being solved. That being said, other high-level channels might also prove appropriate targets for implementation. Consider, for example, a p-reliable channel, which delivers messages to all processes with probability p. Indeed, we formalized a definition of such a channel in [73], and described a simple algorithm for implementing this channel using a t-disrupted channel. We leave the definition and implementation of alternative high-level channels as interesting future work.

With these informal definitions concluded we can proceed with the formal versions. All four property definitions rely on a useful formalism known as a receive function, which we define in the next section before moving on to the property definitions.

6.3 Receive Functions

A receive function maps a domain including messages sent, frequencies used, and perhaps some other parameters, to a range describing sets of possible messages received. These functions capture the relationship between different sending scenarios and possibilities for the corresponding receives. We use these functions to simplify the definition of the channel properties that follow.

We fix the following set, defined for any $t \in \{0, ..., \mathcal{F} - 1\}$, which we will use here and in Section 6.5:

$$F_t = \{s \subset [\mathcal{F}] : |s| \leq t\}.$$

We also define a pair of predicates that aid the receive function definitions:

Definition 6.3.1 (*solobcast*, *collide*) *Let* **solobcast**$(\mathbf{M}, \mathbf{F}, \mathbf{f}, \mathbf{i})$, *for a message assignment M, frequency assignment F, frequency f, and process i, return true if and only if i is the only process broadcasting on frequency f in M and F.*
Let **collide**$(\mathbf{M}, \mathbf{F}, \mathbf{f}, \mathbf{i}, \mathbf{j})$, *for a message assignment M, frequency assignment F, frequency f, and processes i and j, return true if and only if $i \neq j$ and both processes broadcast on f in M and F.*

The following receive function definition captures behavior fundamental to many radio channels.

Definition 6.3.2 (f_{bbc}) *The* basic broadcast channel *receive function, f_{bbc}, is defined such that for every send assignment M and frequency assignment F, $f_{bbc}(M, F)$ consists of every receive assignment N that satisfies the following:*

1. $\forall i \in [n]$: $N[i] \in \mathcal{M}_\perp$.
   (Each process receives no more than one message.)

2. $\forall i \in [n] : N[i] \in \mathcal{M} \Rightarrow \exists j \in [n]$: $M[j] = m \wedge F[i] = F[j]$.
   (If a process receives a message m on frequency f, then some process broadcast m on f.)

3. $\forall i \in [n]$: $M[i] \in \mathcal{M} \Rightarrow N[i] = M[i]$.
   (Each broadcaster receivers its own message.)

4. $\forall i, j \in [n] : F[i] = F[j] \wedge M[i] = M[j] = \perp \Rightarrow N[i] = N[j]$.
   (All processes that receive on the same frequency receive the same thing.)

The above receive function captures some of the basic behaviors common to our four informal property descriptions above (notably, conditions 1–3 from the partial collision property, which are also shared by the other three). It does not, however, provide a sufficient number of properties to define a useful channel. Notably, it does not specify the circumstances under which messages are received or not received. We answer this question with the following receive function, which matches condition 3 from our partial collision property description:

Definition 6.3.3 (f_{if}) *The* interference freedom *receive function, f_{if}, is defined such that for every send assignment M and frequency assignment F, $f_{if}(M, F)$ consists of every receive assignment N that satisfies the following:*

$$\forall f \in [\mathcal{F}], \forall i, j \in [n] : solobcast(M, F, f, i) \wedge F[j] = f \Rightarrow N[j] = M[i]$$

(If a single process broadcasts on a given frequency, then all processes receiving on that frequency receive its message.)

For a given M and F, the set of receive assignments $f_{bbc}(M, F) \cap f_{if}(M, F)$ describe the legal receives for a channel satisfying the partial collision property. Indeed, in the formal definition of this property below, our task is reduced to requiring that the channel receive function return a receive assignment from this set. To capture the more stringent collision requirements of the total collision property, we introduce the following additional receive function:

Definition 6.3.4 (f_{tc}) *The* total collision *receive function f_{tc} is defined such that for every send assignment M and frequency assignment F, $f_{tc}(M, F)$ consists of every receive assignment N that satisfies the following:*

$$\forall i, j, k \in [n], \forall f \in [\mathcal{F}] : collide(M, F, f, i, j) \wedge F[k] = f \wedge M[k] = \perp \Rightarrow N[k] = \perp$$

(If two or more processes broadcast on the same frequency, then each receiver on that frequency receives $\perp$.)

Notice that this definition assumes no collision detection—a receiver cannot distinguish no broadcasts on a frequency from multiple broadcasts. To include collision detection, however, it is sufficient to replace the final $\perp$ in the above definition with a special collision notification marker

The final receive function we define captures the idea that up to t frequencies might be disrupted—preventing communication.

Definition 6.3.5 (f_{dis}^t) *The t-disrupted receive function, f_{dis}^t, $0 \leq t < \mathcal{F}$, is defined such that for every send assignment M, frequency assignment F, and set $B \in F_t$, $f_{dis}^t(M, F, B)$ consists of every receive assignment N that satisfies the following:*

1. $\forall i \in [n] : M[i] = \perp \wedge F[i] \in B \Rightarrow N[i] = \perp$.
   (Every process receiving on a frequency in B receives $\perp$.)

2. $\forall f \in [\mathcal{F}] \setminus B, \forall i, j \in [n] : solobcast(M, F, f, i) \wedge F[j] = f \Rightarrow N[j] = M[i]$.
   (A variation of the interference freedom receive function that is modified to apply only to frequencies not *in B.)*

With our receive functions defined, we can now tackle channel properties.

6.4 Two Basic Channel Properties

We define the first two properties informally described at the opening of this chapter. As mentioned earlier, most of the difficult work of restricting allowable behavior was handled by our receive functions, simplifying these two property definitions to connecting the channel automaton behavior to the results specified by the appropriate collection receive functions from above.

Definition 6.4.1 (Partial Collision Channel Property) *The* partial collision channel *property is the set consisting of every channel $\mathcal{C}$ such that for every channel state $s \in cstates_{\mathcal{C}}$, message assignment M, and frequency assignment F: $crecv_{\mathcal{C}}(s, M, F) \in f_{bbc}(M, F) \cap f_{if}(M, F)$.*

Definition 6.4.2 (Total Collision Channel Property) *The* total collision channel *property is the set consisting of every channel $\mathcal{C}$ such that for every channel state $s \in cstates_{\mathcal{C}}$, message assignment M, and frequency assignment F: $crecv_{\mathcal{C}}(s, M, F) \in f_{bbc}(M, F) \cap f_{if}(M, F) \cap f_{tc}(M, F)$.*

Notice, these definitions are not overly complex or difficult to follow, but they are also mathematically precise, as contrasted with the more informal, English descriptions from existing work.

Multihop Variants. Both of these basic channel properties describe single hop networks. More work is required to extend them to multiple hops. One approach for this extension would be to include the multihop topology—i.e., a graph—as an input to the receive functions. The resulting channel properties could then be stated in a format similar to the following:

> The *multihop total collision channel* property contains every channel $\mathcal{C}$ where there exists a multihop topology $G = (V, E)$, such that for every channel state $s \in cstates_{\mathcal{C}}$, message assignment M, and frequency assignment F: $crecv_{\mathcal{C}}(s, M, F) \in f_{bbc}(M, F, G) \cap f_{if}(M, F, G) \cap f_{tc}(M, F, G)$.

This thesis, however, focuses on single hop networks as the appropriate setting for establishing the basic theory concerning the type of adversarial channels we study. We leave the formal definition of multihop variants of our channel properties as future work.

We continue, in the next section, with our two advanced channel properties.

6.5 Two Advanced Channel Properties

The previous properties describe a *closed* network model in which algorithms operate without outside interference. The only disruption in such a network is caused by the processes themselves. The remainder of this thesis, by contrast, concerns an *open* network model in which algorithms most cope with unpredictable outside interference. We define two properties that capture this behavior below. We begin, however, with some final helper definitions.

The first helper definition is similar to a receive function. We omitted it from the receive functions section, however, because it returns $\mathcal{F}$-vectors instead of receive assignments (which are n-vectors). This function is used to describe the possible valid feedback vectors returned to processes by a feedback channel.

Definition 6.5.1 (*recvAll*) *The function* $recvAll : (\mathcal{M}_\perp)^n \times [\mathcal{F}]^n \times F_t \to (\mathcal{M}_\perp)^{\mathcal{F}}$, *is defined such that for every message assignment* M, *frequency assignment* F, *and set* $B \in F_t$, $recvAll(M, F, B)$ *consists of every* $\mathcal{F}$*-vector* R , *where there exists a set* $B' \subseteq B$ *such that the following conditions hold:*

For every $f \in [\mathcal{F}]$*:*

1. *If* $f \in B'$, *then* $R[f] = \perp$.
   (If f *is in the blocked set* B' , *then* $R[f] = \perp$.*)*
2. *If* $\nexists i \in [n]$ *such that* $M[i] \neq \perp$ *and* $F[i] = f$, *then* $R[f] = \perp$.
   (If no process broadcasts on f, *then* $R[f] = \perp$.*)*
3. *If* $f \notin B'$ *and* $\exists i, j \in [n]$ *such that* $collide(M, F, f, i, j) = true$, *then either* $R[f] = \perp$ *or* $R[f] = m$, *where* m *is a message broadcast on* f *by some process.*
   (If f *is not in the blocked set* B', *and at least two processes broadcast on* f, *then* $R[f]$ either *contains* $\perp$ *or one of the broadcast messages.)*

4. *If $f \notin B'$, and $\exists i \in [n]$ such that $solobcast(M, F, f, i) = true$, then $R[f] = M[i]$. (If f is not in the blocked set B', and only a single process i broadcasts on f, then $R[f]$ equals the message sent by i.)*

The *recvAll* function returns a description of what a receiver would have received on each frequency had the described broadcast behavior occurred on a channel that returned receive assignments in the intersection of the basic broadcast, and t-disrupted receive functions. Notice, unlike the t-disrupted channel property, total collisions are not enforced. In condition 3 from above, a collision causes processes to receive *either* $\perp$ or one of the colliding messages. This extra flexibility simplifies the implementation of feedback channels in Part II.

The set B passed to *recvAll* describes the frequencies that *can* be disrupted, Because the function includes feedback vectors for every subset of B, it is possible that fewer frequencies will be disrupted.

The following two helper definitions will also aid the description of the properties that follow.

Definition 6.5.2 (Receivable, Transformable) *Fix some channel $\mathcal{C}$. We say that a state $s \in cstates_{\mathcal{C}}$ is* ***receivable*** *if and only if there exists a state $s' \in cstates_{\mathcal{C}}$ such that $crand_{\mathcal{C}}(s')(s) > 0$. In other words, it is a state that can be generated by the randomized transformation of the channel state.*
Similarly, we say that a state s is ***transformable*** *if and only if there exists a state s' such that $crand_{\mathcal{C}}(s)(s') > 0$. In other words, it is a state that can be transformed by crand into another state with non-zero probability.*

Definition 6.5.3 (*bcount*) *For any message assignment M, $bcount(M) = |\{i \in [n] : M[i] \neq \perp\}|$. That is, it returns the number of broadcasters in M.*

We can now provide the formal description of the t-disrupted channel property.

Definition 6.5.4 (t-Disrupted Channel Property) *The t-disrupted channel property, $0 \leq t < \mathcal{F}$, is the set consisting of every channel $\mathcal{C}$ such that there exists a mapping $fblocked_{\mathcal{C}} : cstates_{\mathcal{C}} \rightarrow F_t$, such that for every channel state $s \in cstates_{\mathcal{C}}$, message assignment M, and frequency assignment F, the following is satisfied:*

$$crecv_{\mathcal{C}}(s, M, F) \in f_{bbc}(M, F) \cap f_{tc}(M, F) \cap f^t_{dis}(M, F, B),$$

where $B = fblocked_{\mathcal{C}}(s)$.

The definition of a t-disrupted channel is similar to the that of the total collision channel (Definition 6.4.2). The only difference is that we replaced the f_{if} receive function from the total collision property with the f^t_{dis} receive function. Recall, the f_{if} function requires that if a single process broadcasts on a given frequency, then all receivers on that frequency receive its message (Definition 6.3.3). As described in the definition of f^t_{dis} (Definition 6.3.5), the only difference between f_{if} and f^t_{dis} is that the latter *disrupts* the frequencies in the set $B \in F_t$, which is passed as one

of the function parameters. In this context, *disrupts* indicates that receivers on the frequency receive $\perp$, even if there is only a single broadcaster.

With this in mind, $crecv_{\mathcal{C}}(s, M, F) \in f_{bbc}(M, F) \cap f_{tc}(M, F) \cap f^{t}_{dis}(M, F, B)$ consists of receive assignments that describe a total collision channel with the frequencies in B disrupted. The final piece to this definition is the selection of the set B. This selection is specified by the final requirement of the definition: $B = fblocked_{\mathcal{C}}(s)$. In other words, the set of blocked frequencies is determined by the receivable channel state in the round. Because this state is generated independently of the message and frequency assignments for the round, this enforces an independence between the choice of frequencies to disrupt and the algorithm's broadcast behavior.

We continue with the formal definition of the (t, b, p)-feedback channel property. This property features some subtleties, so we carefully walk through its two conditions in the text that follows the formal definition.

Definition 6.5.5 ((t, b, p)-Feedback Channel Property) *The (t, b, p)-feedback channel property, $0 \leq t < \mathcal{F}$, $0 \leq b \leq n$, $0 < p \leq 1$, is the set consisting of every channel $\mathcal{C}$ such that there exist two mappings $pblocked_{\mathcal{C}} : cstates_{\mathcal{C}} \times (\mathcal{M}_{\perp})^n \times [\mathcal{F}]^n \rightarrow P([n])$, and $fblocked_{\mathcal{C}} : cstates_{\mathcal{C}} \rightarrow F_t$, that satisfy the following:*
Let M be a message assignment with $bcount(M) \leq \mathcal{F}$, and let F be a frequency assignment. Then:

1. *For every transformable state s_1, the probability that the distribution $crand_{\mathcal{C}}(s_1)$ returns a state s_2 where $|pblocked_{\mathcal{C}}(s_2, M, F)| \leq b$, is at least p.*

2. *For every receivable state s, there exists an $R \in recvAll(M, F, fblocked_{\mathcal{C}}(s))$, where, letting $N = crecv_{\mathcal{C}}(s, M, F)$, we have, for every $i \in [n]$:*

$$N[i] = \begin{cases} \perp & \text{if } i \in pblocked_{\mathcal{C}}(s, M, F), \\ R & \text{else.} \end{cases}$$

At a high-level, this channel behaves like a t-disrupted channel enhanced such that some processes receive on all frequencies simultaneously, while the others simply receive $\perp$. It guarantees that with probability p, no more than b processes receive $\perp$.

We now consider the details of the definition. First, notice that for every channel $\mathcal{C}$ in the property set, we require the existence of two mappings: $pblocked_{\mathcal{C}}$ and $fblocked_{\mathcal{C}}$. The first will be used to determine the processes that receive $\perp$, given a channel state, message assignment, and frequency assignment. The second is used in the same manner as in the definition of the t-disrupted channel property (Definition 6.5.4). That is, it determines the disrupted frequencies, given a channel state.

With these functions defined, we continue with the two conditions. For both conditions, we fix a message assignment M, and frequency assignment F, in advance. We require that $bcount(M) \leq \mathcal{F}$. The conditions that follow, therefore, do not necessarily apply to message assignments with more than $\mathcal{F}$ broadcasters. As discussed in the description of the *DFC* feedback channel implementation algorithm (Chapter 9), it proves difficult to implement a feedback channel if one must satisfy these conditions

for any number of broadcasters. This follows from the algorithm's requirement that it can recruit some non-broadcasting processes to *listen* on each frequency.

The first condition specifies that given any transformable channel state s_1, the distribution $crand_{\mathcal{C}}(s_1)$ assigns at least probability p to states s_2, where

$$|pblocked_{\mathcal{C}}(s_2, M, F)| \leq b.$$

That is, regardless of the channel state and messages being sent, with probability p no more than b processes will be included in the corresponding $pblocked_{\mathcal{C}}$ set.

The second condition restricts the receive assignment $N = crecv_{\mathcal{C}}(s, M, F)$, where s is a receivable channel state. First it identifies a feedback vector

$$R \in recvAll(M, F, fblocked_{\mathcal{C}}(s)).$$

Recall from the definition of *recvAll* (Definition 6.5.1) that this R describes what could be received on each frequency of a t-disrupted channel, given broadcast behavior M and F, and some subset of the frequencies in $fblocked_{\mathcal{C}}(s)$ being disrupted. The condition concludes by specifying that each process in the $pblocked_{\mathcal{C}}(s, M, F)$ set receives $\perp$ (i.e., is *blocked*), while the rest receive the feedback vector R.

In using this property definition in the proof of our algorithms in Part III, we rely heavily on two elements, in particular. First, the fact that with probability p, at least $n - b$ receive the feedback, *regardless of the channel state or broadcast behavior.* This simplifies the analysis in these proofs, as we have a fixed probability of getting *enough* feedback, for *every* round—regardless of the algorithm's behavior.

The second useful element of this definition is that the disrupted frequencies are determined only by the receivable state. Because this state is generated independently of the message and frequency assignments for a round, it allows the use of random frequency hopping as an effective method for avoiding disrupted frequencies (see, for example, the correctness proof for our reliable broadcast algorithm: Theorem 13.3.1).

Part II

Algorithms for Implementing a Feedback Channel

In Part II we provide a randomized and a deterministic implementation of a (t, b, p)-feedback channel using a t-disrupted channel. Specifically, in Chapter 8 we describe $RFC_{\epsilon,t}$, a randomized algorithm that implements a $(t, 0, 1 - \frac{1}{n^{\epsilon}})$-feedback channel, where ϵ is any constant of size at least 1. For the deterministic case, we describe, in Chapter 9, the algorithm $DFC_{t,M}$, which implements a $(t, t, 1)$-feedback channel. In the latter case, we trade the deterministic guarantee of success (i.e., $p = 1$) for an increased number of potentially blocked processes (t in this case, as compared to 0 for the randomized case).

Chapter 7

Preliminaries

In this preliminary chapter, we define the notation and pseudocode template used in Part II.

7.1 Notation

The following definitions are used in Chapters 8 and 9:

- We use the terms *t-disrupted channel* and (t, b, p)*-feedback channel* to refer to any channel that satisfy the t-disrupted channel property and (t, b, p)-feedback channel property, respectively.
- The term *emulated round* describes all the rounds that transpire between the channel environment passing down a send-encoded input, and the channel algorithm returning the corresponding receive-encoded output.
- The term *real round*, by contrasts, disambiguates the underlying system rounds from the emulated rounds.

7.2 Pseudocode Template

To simplify the description of algorithms, we introduce a pseudocode template.

Specifically, for algorithm $\mathcal{A}$, we describe process $\mathcal{A}(i)$, for every $i \in [n]$, using the format described in Figure 7-1. Informally, the process first retrieves random bits, then receives its input vector, then chooses a frequency and broadcast message (or $\perp$ if it wants to receive), then broadcasts, then receives its received messages from the round, and finally generates an output.

Formally, the template captures this behavior with the following functions:

1. $RAND_i(\kappa \in \mathbb{N})$ returns a string of κ random bits selected uniformly at random.
2. $INPUT_i()$ returns an input value from $\mathcal{I}_\perp$.

Constants for $\mathcal{A}(i)$:
(Constants and values)

Local state that is implicitly included in all algorithm descriptions:
$bit_i \in \{0,1\}^*$, initially 0.
$I_i \in \mathcal{I}_\perp$, initially $\perp$.
$N_i \in \mathcal{R}_\perp$.

Local state for $\mathcal{A}(i)$:
(Local state variables defined and initialized)

For all rounds $r > 0$:
(Local state transformations)
$bit_i \leftarrow RAND_i(\kappa \in \mathbb{N})$
(Local state transformations)
$I_i \leftarrow INPUT_i()$
(Local state transformations)
$BCAST_i(f_i \in [\mathcal{F}], m_i \in \mathcal{M}_\perp)$
$N_i \leftarrow RECV_i()$
(Local state transformations)
$OUTPUT_i(O_i \in \mathcal{O}_\perp)$
(Local state transformations)

Figure 7-1: Pseudocode template for process $\mathcal{A}(i)$.

3. $BCAST_i(f_i \in [\mathcal{F}], m_i \in \mathcal{M}_\perp)$ broadcasts message m_i on frequency f_i. (If $\perp$ is passed instead of a message, this indicates that the process should receive.)

4. $RECV_i()$ returns the message set—or $\perp$ to represent no messages—received during this round.

5. $OUTPUT_i(O_i \in \mathcal{O}_\perp)$ outputs the value O_i.

We also note the following useful additions to our template.

7.2.1 Implicit State

In the pseudocode description provided in Figure 7-1, we describe a collection of state variables labelled "Local state that is implicitly included in all algorithm descriptions." Because these state variables are used in *all* algorithms, we follow the convention that it is unnecessary to reproduce their variable names, types, and initial values in every pseudocode description. We define these variables only once: in Figure 7-1. These definitions are implicit in all subsequent pseudocode descriptions found later in this thesis.

7.2.2 Subroutines

Sometimes when presenting an algorithm it proves useful to separate out certain sections of code. The convention we follow allows the algorithm designer to describe a *subroutine* in a separate figure. Wherever the subroutine title is included in the main algorithm description, we replace the title with the body of the subroutine code. Subroutines can be nested within other subroutines. In all cases, we simply replace a subroutine title with its code body to generate a single pseudocode description.

This simple convention prevents a subroutine from referencing itself; i.e., no recursion. It also sidesteps the notion of multiple scopes within the same process. Subroutines of this type are sometimes also called *macros*.

7.2.3 Undefined Rounds

For simplicity, sometimes we preface process pseudocode with "For all rounds $r \leq k$", for some $k \geq 0$. In this circumstance, it assumed that for all rounds after round k, processes broadcast nothing and output $\perp$.

Chapter 8

The $RFC_{\epsilon,t}$ Randomized Channel Algorithm

In this chapter, we provide a randomized implemention of a (t, b, p)-feedback channel using a t-disrupted channel. Specifically, we describe $RFC_{\epsilon,t}$, a randomized implementation of a $(t, 0, 1 - \frac{1}{n^\epsilon})$-feedback channel using a t-disrupted channel. The main code for $RFC_{\epsilon,t}$ is presented in Figure 8-1; it references subroutines defined in Figures 8-2, 8-3, and 8-4.

The algorithm requires

$$\Theta\left(\frac{\mathcal{F}^2\epsilon}{\mathcal{F}-t}\log n\right)$$

real rounds for each emulated round. Below we provide a high-level summary of the algorithm, including definitions for a relevant helper function and notation. We then describe the pseudocode in detail.

Assumptions on the Size of n. The $RFC_{\epsilon,t}$ algorithm assumes that $n \geq \mathcal{F}$. It could easily be adapted to smaller n, however, as long as $n > t$. This adaptation restricts the processes to use only the first n frequencies.

8.1 Algorithm Overview

The algorithm cycles, indefinitely, through four phases: *waiting*, *broadcast*, *feedback*, and *endfeedback*, and then back to *waiting*, and so on. Each full cycle through these phases corresponds to one emulated round. Below, we provide a high-level description of the algorithm's operation. In the following, the algorithm designates the first $\mathcal{F}$ processes as listeners.

1. At the beginning of an emulated round, the algorithm is in the *waiting* phase. When it receives a send-encoded input from the environment, its transitions into the one-round *broadcast* phase. During this round, each listener receives on the frequency that matches its id, regardless of the message and frequency

passed as input from the channel environment. A non-listener process passed a non-$\perp$ message from the channel environment, broadcasts this message on the frequency also passed from the environment. A non-listener that is passed message $\perp$ can receive on any frequency.

2. After this *broadcast* phase, the algorithm transitions into the *feedback* phase, during which each listener attempts to inform the rest of the processes about what it learned during the *broadcast* phase. Specifically, in each round of the *feedback* phase, each process chooses a frequency at random. Each non-listener process always receives on its chosen frequency. Each listener process receives with probability $1/2$, otherwise it broadcasts a set containing what it received on its assigned frequency during the first round, and what message and frequency pair it was passed by the environment during the first round (if the message is non-$\perp$.) We choose a sufficiently large number of rounds in the feedback phase to ensure that *all* processes hear from *all* $\mathcal{F}$ listeners with probability at least $1 - \frac{1}{n^c}$. This allows the processes to agree on an appropriate receive-encoded output.

3. After the *feedback* phase concludes, the algorithm transitions to the *endfeedback* phase, during which the processes use the information from the preceding phase to generate their receive-encoded outputs. They then return to the *waiting* phase in preparation for starting the next emulated round.

8.2 Helper Function and Notation

Before proceeding with a more detailed discussion of the pseudocode, we define the helper function and notation it uses.

We begin with the helper function *clean*, used in the *PrepareOutput* subroutine. It is defined as follows:

- $clean : P([\mathcal{F}] \times \mathcal{M}_\perp \times [\mathcal{F}]) \rightarrow \{\perp\} \cup (\mathcal{M}_\perp)^{\mathcal{F}}$.
  To define $clean(S)$ we consider two cases for input S:

  - **Case 1**: $\exists f \in [\mathcal{F}], \nexists (*, *, f) \in S$.
    In this case: $clean(S) = \perp$.
    (If the set does not include a message from each listener, then *clean* returns $\perp$.)
  - **Case 2**: $\forall f \in [\mathcal{F}], \exists (*, *, f) \in S$.
    In this case: $clean(S) = R \in (\mathcal{M}_\perp)^{\mathcal{F}}$, where we define R as follows.
    $\forall f \in [\mathcal{F}]$:
    - If S contains exactly one element of the form $(f, *, *)$, then $R[f] = m$, where m is the message (from $\mathcal{M}_\perp$) in this tuple.
    - Else, $R[f] = \perp$.

(If the set contains information from each listener, the function calculates the received value for each frequency, returning m if and only if m is the only value included for a given frequency, and returning $\perp$ otherwise. That is, $R[f] = \perp$ if either: (a) no listener reported receiving a message on f, or (b) two or more messages were sent on f.)

In other words, given a set of 3-tuples of the form (message, frequency, and listener id)—i.e., the information generated during the *feedback phase*—this function calculates an an output value the matches the definition of a feedback channel (i.e., it returns either $\perp$ or a feedback vector).

We also define the following useful notation, used in the *PrepareMessage* subroutine.

- Given an input value $I \in (\mathcal{M}_\perp \times [\mathcal{F}])$, we use the notation $I.msg$ and $I.freq$ to refer to the first and second element of the pair, respectively.
- If $I \notin (\mathcal{M}_\perp \times [\mathcal{F}])$, then $I.msg = I.freq = \perp$.

8.3 Pseudocode Explanation

The pseudocode that defines the process is contained in the three subroutines: *PrepareMessage*, *ProcessMessage*, and *PrepareOutput*. The main body of the code calls these subroutines in between the standard functions common to all algorithms that use our pseudocode template. In the sections that follow, we describe all three subroutines in detail.

8.3.1 The *PrepareMessage* Subroutine

The purpose of this subroutine (presented in Figure 8-2) is to determine the broadcast behavior for the current round (that is, the frequency on which to participate and the message, if any, to send). It also handles the transition from the *waiting* phase to the *broadcast* phase.

The subroutine handles two cases. The first case is when $status_i = waiting$ and $I_i \neq \perp$, which indicates that process i has just received a new send-encoded input from the environment. In this case, it first sets $status_i \leftarrow broadcast$, changing the process status to *broadcast*, as it is about to broadcast the message received from the environment. It then extracts the message component of the input into m_i ($m_i \leftarrow I_i.msg$), and the frequency component into f_i ($f_i \leftarrow I_i.freq$). If the process is not a listener (i.e., if $i > \mathcal{F}$), then it is done with the subroutine. Otherwise, i *is* a listener ($i \leq \mathcal{F}$), and then the process temporarily stashes a copy of m_i and f_i in $tempM_i$ and $tempF_i$ ($tempM_i \leftarrow m_i$ and $tempF_i \leftarrow f_i$), and then resets m_i and f_i to describe receiving on frequency i ($m_i \leftarrow \perp$ and $f_i \leftarrow i$).

The second case for this subroutine is when $status_i = feedback$. (Notice, there is no case for $status_i = broadcast$ or $status_i = endfeedback$, as, by the control logic of the process, this subroutine can never called with these values for $status_i$). If

process i is a listener and the final bit of the random bits in *bit* equals 1 (if $i \leq \mathcal{F}$ and $bit_i[\lceil \lg \mathcal{F} \rceil + 1] = 1$), then it sets m_i to its *feedback messages set*, fbm_i ($m_i \leftarrow fbm_i$), otherwise it sets to m_i to $\perp$ to indicate it should receive ($m_i \leftarrow \perp$). Then, regardless of whether i is a listener or not, the process sets f_i to the random frequency encoded in bit_i ($f_i \leftarrow bit_i[1...\lg \mathcal{F}]$). We can summarize the behavior in this case as follows: non-listeners always receive; a listener, by contrast, receives with probability 1/2, and broadcasts its feedback messages set with probability 1/2.

After the call to the *PrepareMessage* subroutine completes, the process proceeds, in the main body of its pseudocode description, to broadcast according to f_i and m_i ($BCAST_i(f_i, m_i)$). It then stores its receive input in N_i ($N_i \leftarrow RECV_i()$) and calls *ProcessMessage* to process these values.

8.3.2 The *ProcessMessage* Subroutine

The purpose of this subroutine (presented in Figure 8-3) is to process the received messages. It also handles the transition from the *broadcast* phase to the *feedback* phase, and from the *feedback* phase to the *endfeedback* phase.

The subroutine handles two cases. The first case is when $status_i = broadcast$. Here, the process first resets $recvSet_i$ to $\emptyset$ to prepare for the feedback phase ahead—a phase during which it will attempt to fill this set with information about what the send-encoded input encoded for every frequency. If the process is not a listener (i.e., if $i > \mathcal{F}$), it sets $status_i \leftarrow feedback$ and ends the subroutine. If the process *is* a listener ($i \leq \mathcal{F}$), it begins by initializing its feedback messages set, fbm_i, to contain only (f_i, N_i, i). The first two values in this 3-tuple indicate that message N_i was received on frequency f_i. (Notice, by the definition of *PrepareMessage*, described above, $f_i = i$ at this point.) The third value indicates that i is the source of this information.

If the listener was passed a message by the environment in this round ($tempM_i \neq \perp$), it will add this information to fbm_i as well ($fbm_i \leftarrow fbm_i \cup (tempF_i, tempM_i, i)$). In this way, listener i describes in its fbm set, not only what it received while listening on i during the broadcast-phase round, but also what it *would* have sent if it had not been listening. As with the non-listener, its final action in the subroutine is to set its $status_i$ to $feedback$.

This brings us to the second case for this subroutine, which is when $status_i$ equals $feedback$.

Here, the process first checks whether its receive set, N_i, is empty ($N_i = \perp$). If it is *not* empty, it adds N_i to its $recvSet_i$ set ($recvSet_i \leftarrow recvSet_i \cup N_i$). Regardless of whether N_i is empty, the process will then increment its *feedback round counter*, $fbcounter$ ($fbround \leftarrow fbround+1$). If the counter is larger than $FBMAX$, then the feedback phase is done, which the process indicates by setting $status_i \leftarrow endfeedback$.

8.3.3 The *PrepareOutput* Subroutine

The purpose of this subroutine (presented in Figure 8-4) is to prepare the process output. If the process is in the *endfeedback* phase it generates a receive-encoded

Constants for $RFC_{\epsilon,t}(i)$:
$FBMAX = \lceil (\mathcal{F}^2 4(\epsilon+2)\ln n)/(\mathcal{F}-t) \rceil$.

Local state for $RFC_{\epsilon,p}(i)$:
$m_i, tempM_i \in \mathcal{M}_\perp \cup P([\mathcal{F}] \times \mathcal{M}_\perp \times [freq])$, initially $\perp$.
$f_i, tempF_i \in [\mathcal{F}]$, initially 1.
$status_i \in \{waiting, broadcast, feedback, endfeedback\}$, initially *waiting*.
$fbLeader_i \in [\mathcal{F}]$, initially 1.
$fbRound_i \in \mathbb{N}$, initially 0.
$recvSet_i, fbm_i \in P([\mathcal{F}] \times \mathcal{M}_\perp \times [\mathcal{F}])$, initially $\emptyset$.

For all rounds $r > 0$:
$bit_i \leftarrow RAND_i(\lceil \lg \mathcal{F} \rceil + 1)$
$I_i \leftarrow INPUT_i()$
PrepareMessage()$_i$
$BCAST_i(f_i, m_i)$
$N_i \leftarrow RECV_i()$
ProcessMessage()$_i$
PrepareOutput()$_i$
$OUTPUT_i(O_i)$

Figure 8-1: Implementation $RFC_{\epsilon,t}$: a $(t, 0, 1 - \frac{1}{n^\epsilon})$-feedback channel using a t-disrupted channel. Code presented is for process $RFC_{\epsilon,t}(i)$.

output. Otherwise, it outputs $\perp$. The subroutine also handles the transition from the *endfeedback* phase back to the *waiting* phase, in preparation for a new emulated round.

This subroutine always begins by having the process set the output value, O_i, to $\perp$. If $status_i$ does not equal $endfeedback$, then the subroutine is done. If, however, $status_i$ does equal $endfeedback$, then the feedback phase has concluded in this round and the process prepares its receive-encoded output to return to the environment. First, however, it resets $status_i \leftarrow waiting$, $m_i \leftarrow \perp$, $f_i \leftarrow 1$, and $fbRound \leftarrow 0$, so it is ready to start a new emulated round in the next real round. It then sets $O_i = (recv, clean(recvSet_i))$. Recall, a receive-encoded output must be of the form $(recv, m)$, where m is the received message. The *clean* helper function, as described in Section 8.2, will return $\perp$ if the $recvSet_i$ does not include information from all $\mathcal{F}$ listeners, and otherwise returns a feedback vector describing the information contained in the set.

After this subroutine returns, the process, in the main body of the pseudocode, outputs O_i ($OUTPUT_i(O_i)$), to conclude the round.

8.4 Proof of Correctness for $RFC_{\epsilon,t}$

We continue by stating the main theorem.

Theorem 8.4.1 *For any $t \in \{0, ..., \mathcal{F}-1\}$, $\epsilon \in \mathbb{N}^+$, and t-disrupted channel $\mathcal{C}$, there exists a $(t, 0, 1 - \frac{1}{n^\epsilon})$-feedback channel $\mathcal{C}'$ such that $RFC_{\epsilon,t}$ implements $\mathcal{C}'$ using $\mathcal{C}$.*

PrepareMessage()$_i$:
  if ($status_i = waiting$) and ($I_i \neq \perp$) then
    $status_i \leftarrow broadcast$
    $m_i \leftarrow I_i.msg$
    $f_i \leftarrow I_i.freq$
    if ($i \leq \mathcal{F}$) then
      $tempM_i \leftarrow m_i$
      $tempF_i \leftarrow f_i$
      $m_i \leftarrow \perp$
      $f_i \leftarrow i$
  else if ($status_i = feedback$) then
    if ($i \leq \mathcal{F}$) and $bit_i[\lceil \lg \mathcal{F} \rceil + 1] = 1$ then
      $m_i \leftarrow fbm_i$
    else
      $m_i \leftarrow \perp$
    $f_i \leftarrow bit_i[1...\lceil \lg \mathcal{F} \rceil]$

Figure 8-2: PrepareMessage()$_i$ subroutine for $RFC_{\epsilon,t}(i)$.

ProcessMessage()$_i$:
  if ($status_i = broadcast$) then
    $recvSet_i \leftarrow \emptyset$
    if ($i \leq \mathcal{F}$) then
      $fbm_i \leftarrow \{(f_i, N_i, i)\}$
      if ($tempM_i \neq \perp$) then
        $fbm_i \leftarrow fbm_i \cup \{(tempF_i, tempM_i, i)\}$
    $status_i \leftarrow feedback$
  else if ($status_i = feedback$) then
    if $N_i \neq \perp$ then
      $recvSet_i \leftarrow recvSet_i \cup N_i$
    $fbRound_i \leftarrow fbRound_i + 1$
    if ($fbRound_i > FBMAX$) then
      $status_i \leftarrow endfeedback$

Figure 8-3: ProcessMessage()$_i$ subroutine for $RFC_{\epsilon,t}(i)$.

PrepareOutput()$_i$:
  $O_i = \perp$
  if ($status_i = endfeedback$) then
    $status_i \leftarrow waiting$
    $m_i \leftarrow \perp$
    $f_i \leftarrow 1$
    $fbRound \leftarrow 0$
    $O_i = (recv, clean(recvSet_i))$

Figure 8-4: PrepareOutput()$_i$ subroutine for $RFC_{\epsilon,t}(i)$.

For the the proof, fix a specific t, ϵ, and $\mathcal{C}$ that satisfy the constraints of the theorem statement. Let $\epsilon' = \epsilon + 2$.

We will use the expression *feedback phase* to indicate the rounds in an execution of $RFC_{\epsilon,t}$ in which the processes start with $status = feedback$. Similarly, we will use *broadcast phase* to indicate the rounds in which the processes start with $status = waiting$ and then transform to $status = broadcast$ during the call to *PrepareMessage*.

We prove four helper lemmas before tackling the proof of the main theorem. The first two, Lemmas 8.4.2 and 8.4.3, are used by the third, Lemma 8.4.4, which bounds the probability that *no* process outputs $\perp$ at the end of the emulated round. Lemma 8.4.5, the final helper lemma, proves that the all processes that do not output $\perp$, output the same, well-formatted feedback vector.

We start with Lemma 8.4.2, which bounds the probability that a process receives a message from a listener during a feedback phase round. In this lemma, as well as in Lemma 8.4.3, we use the notation *fixed fbm_j value from (listener) process $RFC_{\epsilon,t}(j)$*, to refer to the value of fbm_j fixed by process j during the *broadcast* phase of the current emulated round. Recall from the *ProcessMessage* subroutine that once this value is fixed, it is not changed again until the *broadcast* phase of the next emulated round.

Lemma 8.4.2 *Fix some $i \in [n]$, $j \in [\mathcal{F}]$, channel environment $\mathcal{E}$, and $r-1$ round execution α of $(\mathcal{E}, RFC_{\epsilon,t}, \mathcal{C})$, such that round r is a feedback-phase round.*
The probability that process $RFC_{\epsilon,t}(i)$ receives a message containing the fixed fbm_j value from (listener) process $RFC_{\epsilon,t}(j)$ during a one-round extension of α, is at least $p_{i,j}^{srecv} = \frac{\mathcal{F}-t}{4\mathcal{F}^2}$.

Proof. We consider two cases for i and j. In the first case, $i = j$. If this process broadcasts in r, it will receive fbm_j (by the definition of the t-disrupted property, broadcasters receive their own messages). By the definition of the algorithm, every listener process (i.e., processes 1 to $\mathcal{F}$), during every feedback phase round, broadcasts with probability $1/2 \geq \frac{\mathcal{F}-t}{4\mathcal{F}^2}$, regardless of the history of the execution to this point. So we are done.

The second case is $i \neq j$. At the start of round r, the channel $\mathcal{C}$ probabilistically selects a new state R_r^C by distribution $crand_{\mathcal{C}}(C_{r-1})$ (where C_{r-1} is its final state in α). By the definition of the t-disrupted property, it disrupts the frequencies in $S = fblocked_{\mathcal{C}}(R_r^C)$. The definition also provides that $|S| \leq t$.

During this same round, every process selects a frequency uniformly and at random. The non-listeners receive. The listeners broadcast with probability 1/2. For each process, these random choices are made independently of the history of the execution to this point.

Consider the frequency f_i selected by i. It is important to recall here that the random transformations of the channel state and the process states, in this round, all occur *independently*. They can depend on α (i.e., the execution history up to this round), but they cannot depend on the random selections made by the other entities during the same round. Combining this reminder with the fact that process i selects

its frequency in r uniformly and independent of α, the probability that $f_i \notin S$ is $(\mathcal{F} - |fblocked_C(R_r^C)|)/\mathcal{F} \geq (\mathcal{F} - t)/\mathcal{F}$.

In a similar way, we consider the frequency f_j selected by j, and note that the probability that $f_i = f_j$ is $1/\mathcal{F}$. Combine these two probabilities with the probability of $1/2$ that j broadcasts, and it follows that the probability that i and j land on an undisrupted frequency in round r, and j broadcasts during this round, is at least:

$$\frac{\mathcal{F} - t}{2\mathcal{F}^2}. \tag{8.1}$$

We now calculate the probability that no other listener broadcasts on $f = f_i = f_j$ during this round. We call this probability $p_{j,f}^{solo}$, and we bound it as follows:

$$\begin{aligned}
p_{j,f}^{solo} &\geq \prod_{k \in [\mathcal{F}] \setminus j} (1 - \frac{1}{2\mathcal{F}}) && (8.2)\\
&\geq \prod_{k \in [\mathcal{F}] \setminus j} (1/4)^{\frac{1}{2\mathcal{F}}} && (8.3)\\
&= (1/4)^{\frac{\mathcal{F}-1}{2\mathcal{F}}} && (8.4)\\
&> (1/4)^{\frac{1}{2}} && (8.5)\\
&= (1/2) && (8.6)
\end{aligned}$$

To calculate step (2), we note that every listener chooses frequency f, and decides to broadcast with probability exactly $\frac{1}{2\mathcal{F}}$. Step (3) makes use of Theorem 2.2.1. We now combine (1) and (6) to obtain the following result: the probability that i receives fbm_j from listener j during round r, is at least $p_{i,j}^{srecv} = \frac{\mathcal{F}-t}{4\mathcal{F}^2}$. □

We next look at the probability that a process i hears from a listener j *at least once* during all $FBMAX$ rounds of a feedback phase.

Lemma 8.4.3 *Fix some $i \in [n]$, $j \in [\mathcal{F}]$, channel environment $\mathcal{E}$, and $r-1$ round execution α of $(\mathcal{E}, RFC_{\epsilon,t}, \mathcal{C})$, such that round r is the first round of a feedback phase. The probability that process $RFC_{\epsilon,t}(i)$ receives a message containing the fixed fbm_j value from (listener) process $RFC_{\epsilon,t}(j)$ during at least one of the rounds of an $FBMAX$-round extension of α, is at least $p_{i,j}^{precv} = 1 - n^{-\epsilon'}$.*

Proof. The feedback phase lasts $FBMAX$ rounds (where $FBMAX$ is defined in the constants for the pseudocode for the algorithm). We can extend α one round at a time through the feedback phase. At each round we can apply Lemma 8.4.2, as it holds regardless of the execution history up to that round. It follows that in each of these rounds, process i receives fbm_j with probability at least $p_{i,j}^{srecv} = \frac{\mathcal{F}-t}{4\mathcal{F}^2}$. Conversely, i fails to receive fbm_j in *all* $FBMAX$ rounds, with probability no more than:

$$(1 - p_{i,j}^{srecv})^{FBMAX}. \tag{8.7}$$

By applying Theorem 2.2.1, and noting that $FBMAX = \epsilon' 1/p_{i,j}^{srecv} \ln n$, we can simplify (7) as follows:

$$\begin{aligned}
(1 - p_{i,j}^{srecv})^{FBMAX} &< (e^{-p_{i,j}^{Srecv}})^{FBMAX} && (8.8)\\
&= e^{FBMAX \cdot -p_{i,j}^{srecv}} && (8.9)\\
&= e^{-\epsilon' \ln n} && (8.10)\\
&= n^{-\epsilon'} && (8.11)
\end{aligned}$$

This yields the needed probability. □

The next lemma bounds the probability that *all* processes hear from *all* listeners.

Lemma 8.4.4 *Fix some channel environment $\mathcal{E}$ and $r-1$ round execution α of $(\mathcal{E}, RFC_{\epsilon,t}, \mathcal{C})$, such that round r is the first round of a feedback phase.*
The probability that no process outputs $(recv, \perp)$ at the final round of an $FBMAX$-round extension of α, is at least $1 - (1/n^{\epsilon})$.

Proof. Fix some $i \in [n]$ and $j \in [\mathcal{F}]$. By Lemma 8.4.3, we know that the probability that i fails to receive fbm_j from j during this phase starting at round r, is no more than $n^{-epsilon'}$. Applying a union bound (Theorem 2.2.2), we conclude that i fails to hear from at least one of the $\mathcal{F}$ listeners, with probability no greater than $\frac{\mathcal{F}}{n^{\epsilon'}} \leq n^{-\epsilon'-1}$.

We apply the union bound a second time to calculate that the probability that at least one process fails to hear from at least one listeners, is no more than $n^{-\epsilon'-2} = n^{-\epsilon}$

It follows that with probability at least $1 - (1/n^{\epsilon})$, all processes hear from all listeners during this feedback phase. Finally, by the definition of the algorithm, a process that hears from all listeners during a feedback phase, does not output $\perp$, as needed by the lemma statement. □

We proved that with high probability no process outputs $\perp$. Our final helper lemma constrains this non-$\perp$ output. Unlike the previous lemmas, this one does not concern probabilities—it is a straightforward statement concerning the construction of the receive-encoded values output by $RFC_{\epsilon,t}$.

Lemma 8.4.5 *Fix some channel environment $\mathcal{E}$ and a $r+FBMAX-1$ round execution α of $(\mathcal{E}, RFC_{\epsilon,t}, \mathcal{C})$, such that the final $FBMAX$ rounds of α describe a feedback phase. Let R_{r-1}^{C} be the channel state selected from distribution $crand_{\mathcal{C}}(C_{r-2})$ during round $r-1$. Let M and F be the message and frequency assignments, respectively, encoded in the send-encoded input I_{r-1} of this round. And let L_f describe the frequencies on which exactly one listener was scheduled to broadcast, by M and F.*
There exists a vector $R \in recvAll(M, F, fblocked_{\mathcal{C}}(R_{r-1}^{C}) \setminus L_f))$, such that every process outputs either $(recv, \perp)$ or $(recv, R)$ at the end of the feedback phase starting in round r.

Proof. The relevant output values are generated by the call $clean(S)$ at the end of the *PrepareOutput* subroutine (Figure 8-4), so, in this proof, $R = clean(S)$—leaving us to argue about the values returned by this helper function.

By the definition of *clean*, a process outputs $(recv, \perp)$ at the end of a feedback phase *unless* it received the fbm set from every listener, during the feedback phase. Specifically, we defined *clean* to return $\perp$ if it does not have a 3-tuple of the form $(*, *, i)$, for every $i \in [\mathcal{F}]$. (Recall that this final element in these 3-tuples indicates the source of the information.)

By the definition of the algorithm, the listeners set their fbm set at the end of the broadcast phase round (see the code in *ProcessMessage* that follows the *else if* $(status_i = broadcast)$ conditional), and then perform no further modifications during the subsequent feedback round (see the code following the *if* $(status_i = feedback)$ conditional, in the same subroutine). It follows that all processes that heard from all listeners, pass the same set S to *clean* at the end of the emulated round. We are left, then, to argue that $clean(S) \in recvAll(M, F, fblocked_{\mathcal{C}}(R^C_{r-1}) \setminus L_f)$.

If no listener was scheduled to broadcast in M and F (and therefore $L_f = \emptyset$), this clearly holds. There was one listener on each frequency of a t-disrupted channel during the broadcast phase, and all processes hear from all $\mathcal{F}$ listeners regarding what they received during the broadcast phase round. The *clean* function will return the needed $\mathcal{F}$-vector that matches *recvAll*'s output format.

If, however, a single listener was scheduled to broadcast on a frequency in $fblocked_{\mathcal{C}}(R^C_{r-1})$, the issue becomes more complicated. In this case, the listener *assigned* to this frequency (by the algorithm) will receive $\perp$—because the frequency is disrupted. The listener *scheduled* to broadcast on this frequency (by the channel environment), however, includes its message in its fbm set (the $fbm_i \leftarrow fbm_i \cup \{(tempF_i, tempM_i, i)\}$ line from *ProcessMessage*) and then disseminates this set to all processes during the subsequent feedback phase. It will be the only value associated with this frequency and therefore returned in the $clean(S)$ vector—even though the frequency was disrupted. We anticipate this possibility by subtracting, in our lemma statement, these frequencies (i.e., L_f) from $fblocked_{\mathcal{C}}(R^C_{r-1})$, to generate the blocked set we pass to *recvAll*.

A similar subtlety arises when two non-listeners broadcast on a frequency, and one listener is scheduled to broadcast on the same frequency. The listener assigned to that frequency receives $\perp$ due to collision. The scheduled listener's message, however, will be disseminated and included in the $clean(S)$ set. We handle this case by noting that we defined *recvAll* to allow for a message to be received from among multiple messages broadcast simultaneously (Definition 6.5.1). □

We conclude by deploying our helper lemmas, along with our composition channel theorem from earlier, to prove our main theorem.

Proof (Theorem 8.4.1). By Theorem 5.2.10, we know that $RFC_{\epsilon,t}$ implements channel $\mathcal{C}(RFC_{\epsilon,t}, \mathcal{C})$ using $\mathcal{C}$. For simplicity, we introduce the shorthand notation $\mathcal{C}' = \mathcal{C}(RFC_{\epsilon,t}, \mathcal{C})$. Our goal is to prove that $\mathcal{C}'$ is a $(t, 0, 1 - \frac{1}{n^\epsilon})$-feedback channel. To do so, we must prove that $\mathcal{C}'$ satisfies the conditions of Definition 6.5.5.

This definition requires the existence of the two mappings, $fblocked_{\mathcal{C}'}$ and $pblocked_{\mathcal{C}'}$, that satisfy the definition's two conditions. Below, we address the two conditions in turn. We define $fblocked_{\mathcal{C}'}$ and $pblocked_{\mathcal{C}'}$ as needed by the conditions.

Proving Condition (1) of Feedback Channel Definition. We begin with condition (1), which states:

> Let M be a message assignment with $bcount(M) \leq \mathcal{F}$, and let F be a frequency assignment. Then, for every transformable state s_1, the probability that the distribution $crand_{\mathcal{C}'}(s_1)$ returns a state s_2 where $|pblocked_{\mathcal{C}'}(s_2, M, F)| \leq b$, is at least p.

In our case, $b = 0$ and $p = 1 - (1/n^{\epsilon})$. We need to define $pblocked_{\mathcal{C}'}$ such that the above is satisfied for these parameters.

Recall from Definition 5.2.5 that the composition channel has two types of states: *simple* and *complex*. Because the state s_1 is transformable, we know it is simple.

Therefore, the distribution $crand_{\mathcal{C}'}(s_1)$ assigns non-zero probability only to complex states. With this is in mind, we define $pblocked_{\mathcal{C}'}$ only for complex states.

Specifically, given such a state, s_2, and assignments M and F that match the conditions' preconditions, define $pblocked_{\mathcal{C}'}(s_2, M, F)$ to return the set of processes that output $(recv, \perp)$ at the end of the last emulated round encoded in $s_2.oext(M, F)$.

Our second step is to show that this definition of $pblocked_{\mathcal{C}'}$ matches the constraints of the condition. Fix some s_1, M, and F that satisfy the preconditions of the condition. Let $\alpha = s_1.pre$. Let S be the set of all complex states assigned a non-zero probability by $crand_{\mathcal{C}'}(s_1)$, and let $S' \subset S$ be the subset of states from S such that the last emulated round in their $oext(M, F)$ entry has *no* process output $(recv, \perp)$. Note, by our definition of $pblocked_{\mathcal{C}'}$, for every such state $s' \in S'$: $pblocked_{\mathcal{C}'}(s', M, F) = 0$.

We need to lower bound the probability mass in $crand_{\mathcal{C}'}(s_1)$ assigned to states in S'. To do so, we first partition S' by the values of the $oext(M, F)$ rows. Let $exts(S')$ be the set consisting of every unique environment-free execution contained in a $oext(M, F)$ row of a state in S'. For each $\alpha' \in exts(S')$, we define partition component $X_{\alpha'}$ of S' as follows:

$$X_{\alpha'} = \{s' \in S' : s'.oext(M, F) = \alpha'\}$$

We apply Lemma 5.2.8 with respect to α and α' to determine:

$$\sum_{s' \in X_{\alpha'}} crand_{\mathcal{C}'}(s_1)(s') = Pr[\alpha'|\alpha],$$

where $Pr[\alpha'|\alpha]$ is the probability that $RFC_{\epsilon,t}$, using channel $\mathcal{C}$, extends α to α', given the input assignments in α'.

We now rewrite the sum of the probability assigned to the states in S' as a double sum, and then substitute the above result to simplify:

$$\begin{aligned}\sum_{s' \in S'} crand_{\mathcal{C}'}(s_1)(s') &= \sum_{\alpha' \in exts(S')} \sum_{s' \in X_{\alpha'}} crand_{\mathcal{C}'}(s_1)(s') \\ &= \sum_{\alpha' \in exts(S')} Pr[\alpha'|\alpha] \\ &= p_{fdbk}\end{aligned}$$

where p_{fdbk} is the probability that $RFC_{\epsilon,t}$, using channel $\mathcal{C}$, extends α to *some* emulated round that has no process output $(recv, \perp)$, given the input $toinput(M, F)$.

This is a statement about the behavior of $RFC_{\epsilon,t}$ running on $\mathcal{C}$, which allows us to deploy our Lemma 8.4.4 from earlier in this chapter. First, however, we must transform the environment-free execution α (from $s_1.pre$) into an execution. We deploy Theorem 5.2.12 to this end. This tells us that there exists a channel environment, and a way to add states from this channel environment to $s_1.pre$, that provides a valid execution. We can now apply Lemma 8.4.4 to determine that the probability that $RFC_{\epsilon,t}$, using $\mathcal{C}$, extends this execution to an emulated round with no $(recv, \perp)$ outputs, is at least $1 - (1/n^{\epsilon})$. It follows that $p_{fdbk} = 1 - (1/n^{\epsilon})$, as well.

We have shown, therefore, that with probability at least $1 - (1/n^{\epsilon})$, s_1 transforms to a state $s_2 \in S'$. For every such $s_2 \in S'$: $pblocked_{\mathcal{C}'}(s_2, M, F) = 0$, as needed.

Proving Condition (2) of Feedback Channel Definition. We continue with condition (2), which states:

> Let M be a message assignment with $bcount(M) \leq \mathcal{F}$, and let F be a frequency assignment. Then, for every receivable state s, there exists an $R \in recvAll(M, F, fblocked_{\mathcal{C}'}(s))$, where, letting $N = crecv_{\mathcal{C}}(s, M, F)$, we have, for every $i \in [n]$:
>
> $$N[i] = \begin{cases} \perp & \text{if } i \in pblocked_{\mathcal{C}'}(s, M, F) \\ R & \text{else} \end{cases}$$

We already defined $pblocked_{\mathcal{C}'}$, so we are left to define $fblocked_{\mathcal{C}'}$ and show that the two functions, when combined, satisfy the above property.

Our first step is to define $fblocked_{\mathcal{C}'}$. Because s is receivable, we know it is complex. With this in mind, let $s_{\mathcal{C}}$ be the last state of $\mathcal{C}$ encoded in $s.ext$. (Recall from Definition 5.2.5), $s.pre$ is a prefix that contains no partial rounds. While $s.ext$, by contrast, extends $s.pre$ by a single channel and algorithm state. That is, it takes the states at the end of the prefix in $s.pre$, and transforms them probabilistically to their new state by applying the matching distributions.)

We define:

$$fblocked_{\mathcal{C}'}(s) = fblocked_{\mathcal{C}}(s_{\mathcal{C}})$$

In other words, $fblocked_{\mathcal{C}'}(s)$ evaluates to the frequencies disrupted by t-disrupted channel, $\mathcal{C}$, in the emulations of RFC on $\mathcal{C}$, captured in state s of the composition

channel $\mathcal{C}'$. This is where it proves important that we included an *ext* field in addition to the *oext* table in our definition of complex composition channel state. It requires that every emulated round in the *oext* table start with the same state $s_{\mathcal{C}}$ of the channel $\mathcal{C}$.

Our second step is to show that our definitions match the constraints of the condition. Fix some s, M, and F that satisfy the preconditions of the condition. Above, we defined $pblocked_{\mathcal{C}'}(s, M, F)$ to return the processes that output $(recv, \perp)$ in the last emulated round in $s.oext(M, F)$. We are left to show that the processes not in this set all output the same $R \in recvAll(M, F, fblocked_{\mathcal{C}'}(s))$.

Lemma 8.4.5 aids us in this effort. As in our proof of condition (1), however, we must first take the relevant environment-free execution encoded in s and transform it into a valid execution for which the lemma applies.

We apply Theorem 5.2.12 to the execution in $s.oext(M, F)$. This produces a matching execution of a system with $RFC_{\epsilon,t}$, $\mathcal{C}$, and a channel environment. This execution has the same states of $RFC_{\epsilon,t}$ and $\mathcal{C}$, and the same input and output assignments, as in $s.oext(M, F)$. We can apply Lemma 8.4.5 to the last emulated round in this execution. It follows that the processes that do not output $\perp$ output the same $R \in recv(M, F, fblocked_{\mathcal{C}}(s') \setminus L_f)$, where s' is the first receivable channel state in this emulated round, and L_f is the subset of the frequencies defined in the statement of Lemma 8.4.5.

The same holds, therefore, for the output at the end of $s.oext(M, F)$.

By definition, $s_{\mathcal{C}} = s'$. It follows that $fblocked_{\mathcal{C}}(s') \setminus L_f$ is a subset of $fblocked_{\mathcal{C}}(s_{\mathcal{C}}) = fblocked_{\mathcal{C}'}(s)$. Therefore, any $R \in recv(M, F, fblocked_{\mathcal{C}}(s') \setminus L_f)$ is also in $recvAll(M, F, fblocked_{\mathcal{C}'}(s))$, as the definition of *recvAll* defines valid feedback vectors for all subsets of the blocked set passed as its third parameter.[1]

We have shown, therefore, that every process not in $pblocked_{\mathcal{C}'}(s, M, F)$ returns the same $R \in recvAll(M, F, fblocked_{\mathcal{C}'}(s))$, as needed. □

8.5 Time Complexity of $RFC_{\epsilon,t}$

We now bound the time complexity of the emulated rounds generated by $RFC_{\epsilon,t}$.

Theorem 8.5.1 *Fix some $t \in \{0, ..., \mathcal{F} - 1\}$, $\epsilon \in \mathbb{N}^+$, t-disrupted channel $\mathcal{C}$, and channel environment $\mathcal{E}$. Let α be an execution of $(\mathcal{E}, RFC_{\epsilon,t}, \mathcal{C})$. Every emulated round in α requires*

$$\lceil (\mathcal{F}^2 4(\epsilon + 2) \ln n)/(\mathcal{F} - t) \rceil + 1 =$$
$$\Theta\left(\frac{\mathcal{F}^2 \epsilon}{\mathcal{F} - t} \log n\right),$$

real rounds to complete.

[1]Formally, this is captured in its wording that given such a parameter B, it returns every R, such that there exists a $B' \subseteq B$, where the properties that follow hold with respect to B'.

Proof. The result follows directly from the algorithm definition. □

Chapter 9

The $DFC_{t,M}$ Deterministic Channel Algorithm

In this chapter, we provide a deterministic implementation of a (t, b, p)-feedback channel using a t-disrupted channel. Specifically, we describe $DFC_{t,M}$, for $t \in \{0, ..., \mathcal{F}-1\}$ and $(n, \mathcal{F}, t+1)$-multiselector M (recall, a *multiselector* is a combinatorial object defined in Section 2.3). The main code for $DFC_{t,M}$ is presented in Figure 9-1; it references subroutines defined in Figures 9-2, 9-3, and 9-4.

The algorithm requires

$$\Theta\left(\mathcal{F}^2|M|\right),$$

real rounds for each emulated round, where $|M|$ is the size of the multiselector, M. This size varies from a maximum of $O\left(\mathcal{F}^{\mathcal{F}} \log^{\mathcal{F}} n\right)$ to a minimum of $O\left(t \log\left(n/t\right)\right)$, depending on both the size of t compared to $\mathcal{F}$, and whether we consider an existential or constructive proof of the multiselector's size. (See Chapter 2 for more details on these objects and their sizes.)

Assumption on the Size of n. The $DFC_{t,M}$ algorithm assumes that $n = \Omega(\mathcal{F}^3)$. It is possible to implement this feedback channel with smaller n, but this increases complexity—both in terms of time and details of the algorithm—without adding significant new insight to the problem.

9.1 Algorithm Overview

The algorithm cycles, indefinitely, through five phases: *waiting*, *broadcast*, *feedback*, *combine*, and *output*, and then back to *waiting*, and so on. Each full cycle through these phases corresponds to one emulated round. Below, we provide a high-level description of the algorithm's operation. In the following, the algorithm designates the first $\mathcal{F}^3 + \mathcal{F}^2$ processes as listeners. Similarly, it also designates the first $\mathcal{F}^2 t + \mathcal{F}$ processes as combiners.

At a high-level, its operation can be summarized as follows:

1. At the beginning of an emulated round, the algorithm is in the *waiting* phase. When it receives a send-encoded input from the environment, it transitions into the one round *broadcast* phase. During this round, the algorithm divides the listeners into $\mathcal{F}$ groups of size $\mathcal{F}^2 + \mathcal{F}$, and assigns each group to a frequency. Each listener receives on the frequency assigned to its group, with the following exception: if a listener has been scheduled to broadcast by its input, it abandons its role as listener and follows its scheduled behavior. Non-listeners always behave as described in their input. (Notice, this differs from the *RFC* algorithm, which had listeners ignore the behavior in their input during this first round, and stick to their assigned frequencies. This worked for *RFC* because we designated only one listener per frequency, so that listener could add what it was *supposed* to broadcast to the set of values it heard. In *DFC*, by contrast, we assign multiple listeners to each frequency. It is important for the correctness of the algorithm that *every* listener on the same frequency is disseminating the *same* set of values during the feedback phase. Different processes might hear from different listeners. These processes, however, must receive the same information from their respective listener so they can later generate the same feedback vector, as required by the definition of the feedback channel property.)

2. After this *broadcast* phase, the algorithm transitions into the *feedback* phase, during which the listeners attempt to inform the rest of the processes about what they learned during the *broadcast* phase. Specifically, each frequency is assigned $\mathcal{F}^2 + \mathcal{F}$ listeners. The algorithm divides listeners these into $\mathcal{F} + 1$ non-overlapping groups of size $\mathcal{F}$. We call these *listener groups*. Each of the $\mathcal{F}^2 + \mathcal{F}$ listener groups in the system gets its own *feedback epoch* to disseminate its values. During this epoch, the algorithm assigns one group member to each frequency to broadcast while all other processes receive according to a $(n, \mathcal{F}, t + 1)$-multiselector. That is, in each round of the epoch, each of process applies the multiselector function corresponding to that round to to its id, and then receives on the frequency returned by the function. For example, during the r^{th} round of a feedback epoch, a process i that is not in the listener group for this epoch, receives on frequency $M_r(i)$, where M_r is the r^{th} function of M.

3. After the *feedback* phase concludes, the algorithm transitions to the *combine* phase. The algorithm divides the combiners into $\mathcal{F}t+1$ non-overlapping *combine groups* of size $\mathcal{F}$. The algorithm assigns each combine group its own *combine epoch* to disseminate the values it knows. The processes follow the same procedure as in the feedback epochs—one member of the combine group is assigned to each frequency to broadcast while the other processes receive according to M.

4. After the *combine* phase concludes, the algorithm transitions to the *output* phase, during which the processes use the information from the preceding phases to generate their receive-encoded outputs. They then return to the *waiting* phase in preparation for starting the next emulated round.

The are several key insights behind the correctness of this algorithm. First, note that up to $\mathcal{F}$ of the listeners on a given frequency might abandon their listening responsibilities to broadcast. The definition of the feedback channel property (Definition 6.5.5), requires that $bcount(M) \leq \mathcal{F}$, where M is the message assignment for the round, and $bcount(M)$ returns the number of non-$\perp$ entries in M. It follows that we do not need to address the case of more than $\mathcal{F}$ broadcasters.

Split the listeners from a frequency into $\mathcal{F}+1$ groups of size $\mathcal{F}$. It follows that at least one of these groups is comprised *only* of listeners that actually listened. Let us identify one such *behaved* listener group for each frequency, and consider only the feedback epochs associated with these $\mathcal{F}$ behaved groups.

In each of these select epochs, we can show that the behaved listeners propagate what they heard in the initial broadcast round to all but at most t of the other processes. To see why, recall that during such an epoch, each listener broadcasts on a unique frequency while the receivers receive for $|M|$ rounds, choosing their frequency according to the corresponding function of M. By the definition of an $(n, \mathcal{F}, t+1)$-multiselector, each group of $t+1$ processes is assigned disjoint frequencies by at least one function in M. It follows that no group of $t+1$ can be disrupted in every round of a feedback epoch—in at least one round, they receive on disjoint frequencies, and only t can be disrupted in that round.

After the feedback phase is over, up to $\mathcal{F}t$ processes may have failed to hear from some frequency's listeners (t per behaved feedback epoch). In the combine phase, we have $\mathcal{F}t+1$ non-intersecting groups of size $\mathcal{F}$ disseminate all values they know. At least one of these groups must be made up of $\mathcal{F}$ processes that know everything (recall, no more than $\mathcal{F}t$ processes are missing knowledge). By the same argument applied to the feedback phase, all but at most t processes will subsequently hear from a member of this group, leaving at least $n-t$ with full knowledge of what happened during the initial broadcast round. These knowledgeable processes can calculate appropriate output, while the remaining processes can consider themselves *blocked*—and output $\perp$.

9.2 Helper Functions and Notation

Before proceeding with a more detailed discussion of the pseudocode, we define the helper functions and notation it uses.

We begin with the definitions for the following helper functions:

For all $i \in [n]$:

- $lfreq(i \in [\mathcal{F}^3 + \mathcal{F}^2]) = \lceil \frac{i}{\mathcal{F}^2+\mathcal{F}} \rceil$.
  (For larger values of i, $lfreq(i) = 1$.)
  This helper function returns the frequency on which i should listen if i is a listener.

- $fgroup(i \in [\mathcal{F}^3 + \mathcal{F}^2]) = \lceil \frac{i}{\mathcal{F}} \rceil$.
  (For larger values of i, $fgroup(i) = 1$.)
  This helper function returns the listener group of i if i is a listener.

- $ffreq(i \in [\mathcal{F}^3 + \mathcal{F}^2]) = i - \mathcal{F}(fgroup(i) - 1)$.
  (For larger values of i, $ffreq(i) = 1$.)
  This helper function returns the frequency i should broadcast on during the feedback epoch associated with its frequency group. It guarantees that all $\mathcal{F}$ processes from a given frequency group will be returned unique frequencies.

- $cgroup(i \in [\mathcal{F}^2 t + \mathcal{F}]) = \lceil \frac{i}{\mathcal{F}} \rceil$.
  (For larger values of i, $cgroup(i) = 1$.)
  This helper function returns the combine group of i if i is a combine process.

- $cfreq(i \in [\mathcal{F}^2 t + \mathcal{F}]) = i - \mathcal{F}(cgroup(i) - 1)$.
  (For larger values of i, $cfreq(i) = 1$.)
  This helper function returns the frequency i should broadcast on during the combine epoch associated with its combine group. It guarantees that all $\mathcal{F}$ processes from a given combine group will be returned unique frequencies.

- $merge(N_i, vals_i)$: merges $vals_i$ with the $vals$ vector (if any) contained in N_i.
  (That is, if the received message, N_i, contains a $vals$ vector that includes a value in position k, such that $vals_i[k] = \perp$, $merge$ assigns $vals_i[k] = vals[k]$.

- $vclean(vals_i)$: if $vals_i$ does not contain $undefined$ in any position it returns $vals_i$, otherwise it returns $\perp$.

We also use the notation $I.msg$ and $I.freq$, for every input $I \in \mathcal{I}_\perp$, defined as in Section 8.2.

9.3 Pseudocode Explanation

As with $RFC_{\epsilon,t}$, the pseudocode that defines the process is contained in the three subroutines: *PrepareMessage*, *ProcessMessage*, and *PrepareOutput*. The main body of the code, calls these subroutines in between the standard functions common to all algorithms that use our pseudocode template. In the sections that follow, we describe all three subroutines in detail.

9.3.1 The *PrepareMessage* Subroutine

The purpose of this subroutine (presented in Figure 9-2) is to determine the broadcast behavior for the current round. It also handles the transition from the *waiting* phase to the *broadcast* phase.

This subroutine handles three cases. The first case is when $phase_i = waiting$ and $I_i \neq \perp$, which indicates that process i has just received a new send-encoded input from the environment. In this case, it first sets $phase_i = broadcast$, changing the process phase to *broadcast*. As in the RFC channel algorithm, this indicates that the process is about to broadcast the message received from the environment. It next extracts the message component of the input into m_i ($m_i \leftarrow I_i.msg$), and the frequency component into f_i ($f_i \leftarrow I_i.freq$). If the process i is not a listener

(i.e., if $i > \mathcal{F}^3 + \mathcal{F}^2$), or if it *is* a listener but was passed a message to broadcast in its input ($m_i \neq \perp$), then it is done with the subroutine, and it will proceed to broadcast as instructed in its input. Otherwise, if the process is a listener that was not passed a message in its input, it sets its receiving frequency to that specified by $lfreq$ ($f_i \leftarrow lfreq(i)$). (Recall, by the definition $lfreq$, the first $\mathcal{F}^3+\mathcal{F}^2$ processes are partitioned into $\mathcal{F}$ groups of size $\mathcal{F}^2 + \mathcal{F}$. The $lfreq(i)$ function returns the number of the group to which i belongs.

The second case for this subroutine is when $phase_i = feedback$. If process i is a listener ($i \leq \mathcal{F}^3 + \mathcal{F}^2$) and the variable $currepoch_i$ equals the feedback group to which i belongs ($currepoch_i = fgroup(i)$), then i will set its broadcast message to its $vals_i$ vector ($m_i \leftarrow vals_i$), and its frequency to the frequency returned by $ffreq(i)$ ($f_i \leftarrow ffreq(i)$). (Recall, $fgroup(i)$ partitions the listeners into groups of size $\mathcal{F}$ and then returns the number of the group to which i belongs, and $ffreq$ assigns a unique frequency to all listeners assigned to the same frequency group by $fgroup$.)

If the process is not a listener, or the process *is* a listener but $currepoch_i$ does not equal its frequency group, then it receives on frequency $M_{ernd_i}(i)$, where M_{ernd_i} is the $ernd_i^{th}$ function of the multiselector M. (Recall, each function of the multiselector M maps processes to values from $[\mathcal{F}]$.)

The third case for this subroutine is when $phase_i = combine$. It is symmetric to the second case with the exception that we replace $fgroup$ with $cgroup$, and $ffreq$ with $cfreq$. For completeness, however, we still detail its operation.

If process i is a combine process ($i \leq \mathcal{F}^2 t + \mathcal{F}$), and the variable $currepoch_i$ equals the combine group to which i belongs ($currepoch_i = cgroup(i)$), then i will set its broadcast message to its $vals_i$ vector ($m_i \leftarrow vals_i$), and its frequency to the frequency returned by $cfreq(i)$ ($f_i \leftarrow cfreq(i)$). (Recall, $cgroup(i)$ partitions the combine processes into groups of size $\mathcal{F}$, and then returns the number of the group to which i belongs, and $cfreq$ assigns a unique frequency to all combine processes assigned to the same combine group by $cgroup$.)

If the process is not a combine process, or the process *is* a combine process but $currepoch_i$ does not equal its combine group, then it receives on frequency $M_{ernd_i}(i)$, where M_{ernd_i} is the $ernd_i^{th}$ function of the multiselector M. (Recall, each function of the multiselector M maps processes to values from $[\mathcal{F}]$.)

After the call to the *PrepareMessage* subroutine completes, the process proceeds, in the main body of its pseudocode description, to broadcast according to f_i and m_i ($BCAST_i(f_i, m_i)$). It then stores its receive input in N_i ($N_i \leftarrow RECV_i()$) and calls $ProcessMessage$ to process these values.

9.3.2 The *ProcessMessage* Subroutine

The purpose of this subroutine (presented in Figure 9-3) is to process the received messages. It also handles the transition from the *broadcast* phase to the *feedback* phase, from the *feedback* phase to the *combine* phase, and the *combine* phase to the *output* phase.

This subroutine, as with *PrepareMessage*, handles three cases. We start with the case where $phase_i = broadcast$. Here, the process i starts by initializing its

$vals_i$ vector to $undefined^{\mathcal{F}}$. If the process is a listener that actually listened during this round (i.e., if $i \leq \mathcal{F}^3 + \mathcal{F}^2$ and $m_i = \perp$), then it sets position $lfreq(i)$ of $vals_i$ to the value it received listening on frequency $lfreq(i)$ ($vals_i[lfreq(i)] = N_i$). Regardless of whether or not i is a listener, it concludes the subroutine by initializing $currepoch_i \leftarrow 1$ and $erand_i \leftarrow 1$, and then setting $phase_i \leftarrow feedback$. It is now ready to begin the *feedback* phase.

The second case handled by this subroutine is when $phase_i = feedback$. Here, the first action of process i is to merge the information it received in this *feedback* phase round with its $vals_i$ vector ($merge(N_i, vals_i)$). It then increments $ernd_i$, which indicates the current *epoch round*. (Recall from the overview that each feedback group gets its own epoch, consisting of $|M|$ rounds, to attempt to disseminate what the group members received during the *broadcast* phase round.) If $ernd_i > |M|$, then the process advances the epoch by setting $currepoch_i \leftarrow currepoch_i + 1$, and then resets $ernd_i \leftarrow 1$. If this newly incremented $currepoch_i$ is too large ($currepoch_i > \mathcal{F}^2 + \mathcal{F}$), then it concludes the *feedback* phase and can advance to the *combine* phase. The process accomplish this by setting $phase_i \leftarrow combine$ and reseting $currepoch_i \leftarrow 1$. (We will use the same pair of variables, $currepoch_i$ and $ernd_i$, to track progress through the combine epochs.)

The third case handled by this subroutine is when $phase_i = combine$. Here, as with the feedback case, the first action of process i is to merge the information it received in this *combine* phase round with its $vals_i$ vector ($merge(N_i, vals_i)$). It then increments $ernd_i$. The logic that follows is symmetric to the logic for the *feedback* phase case, with the exception that the *combine* phase consists of only $\mathcal{F}t+1$ epochs, not $\mathcal{F}^2 + \mathcal{F}$.

Specifically, the process checks to see if $ernd_i > |M|$ after the increment. If so, it increments $currepoch_i$ and resets $ernd_i \leftarrow 1$. It then checks to see if its newly increment $currepoch_i$ is too large ($currepoch_i > \mathcal{F}t+1$). If so, the process is ready to proceed to the *output* phase, which it indicates by setting $phase_i = output$. Notice, there is no need to reset $currepoch_i$ back to 1 as this will be handled by the *broadcast* phase case from above, before the process next needs to use this variable.

9.3.3 The *PrepareOutput* Subroutine

The purpose of this subroutine (presented in Figure 9-4) is to prepare the process output. If the process is in the *output* phase it generates a receive-encoded output. Otherwise, it outputs $\perp$. The subroutine also handles the transition from the *output* phase back to the *waiting* phase, in preparation for a new emulated round.

This subroutine handles two cases. The first case is when $phase_i = output$. Here, the process resets $phase_i \leftarrow waiting$, $m_i \leftarrow \perp$, and $f_i \leftarrow 1$, so it will be ready to start a new emulated round in the next real round. It concludes by setting $O_i \leftarrow (recv, vclean(vals_i))$: the receive-encoded output generated by the current emulated round. If $phase_i \neq output$, by contrast, the process simply sets $O_i \leftarrow \perp$.

After this subroutine returns, the process, in the main body of the pseudocode, outputs O_i ($OUTPUT_i(O_i)$), to conclude the round.

Local state for $DFC_{t,M}(i)$:
  $m_i \in \mathcal{M}_\perp \cup (\mathcal{M}_\perp \cup \{undefined\})^{\mathcal{F}}$, initially $\perp$.
  $f_i \in [\mathcal{F}]$, initially 1.
  $phase_i \in \{waiting, broadcast, feedback, combine, output\}$, initially $waiting$.
  $currepoch_i$, $ernd_i \in \mathbb{N}$, initially 0.
  $vals_i \in (\mathcal{M}_\perp \cup \{undefined\})^{\mathcal{F}}$, initially $undefined^{\mathcal{F}}$.

For all rounds $r > 0$:
  $I_i \leftarrow INPUT_i()$
  PrepareMessage()$_i$
  $BCAST_i(f_i, m_i)$
  $N_i \leftarrow RECV_i()$
  ProcessMessage()$_i$
  PrepareOutput()$_i$
  $OUTPUT_i(O_i)$

Figure 9-1: Implementation $DFC_{t,M}$: a $(t, t, 1)$-feedback channel using a t-disrupted channel. Code presented is for process $DFC_{t,M}(i)$.

PrepareMessage()$_i$:
  if $(phase_i = waiting)$ and $(I_i \neq \perp)$ then
    $phase_i \leftarrow broadcast$
    $m_i \leftarrow I_i.msg$
    $f_i \leftarrow I_i.freq$
    if $(i \leq \mathcal{F}^3 + \mathcal{F}^2)$ and $(m_i = \perp)$ then
      $f_i \leftarrow lfreq(i)$
  else if $(phase_i = feedback)$ then
    if $(i \leq \mathcal{F}^3 + \mathcal{F}^2)$ and $(currepoch_i = fgroup(i))$ then
      $m_i \leftarrow vals_i$
      $f_i \leftarrow ffreq(i)$
    else
      $m_i \leftarrow \perp$
      $f_i \leftarrow M_{ernd_i}(i)$
  else if $(phase_i = combine)$ then
    if $(i \leq \mathcal{F}^2 t + \mathcal{F})$ and $(currepoch_i = cgroup(i))$ then
      $m_i \leftarrow vals_i$
      $f_i \leftarrow cfreq(i)$
    else
      $m_i \leftarrow \perp$
      $f_i \leftarrow M_{ernd_i}(i)$

Figure 9-2: PrepareMessage()$_i$ subroutine for $DFC_{t,M}(i)$.

```
ProcessMessage()_i:
  if (phase_i = broadcast) then
    vals_i ← undefined^F
    if (i ≤ F^3 + F^2) and (m_i = ⊥) then
      vals_i[lfreq(i)] ← N_i
    currepoch_i ← 1
    ernd_i ← 1
    phase_i ← feedback
  else if (phase_i = feedback) then
    merge(N_i, vals_i)
    ernd_i ← ernd_i + 1
    if (ernd_i > |M|) then
      currepoch_i ← currepoch_i + 1
      ernd_i ← 1
      if (currepoch_i > F^2 + F) then
        phase_i ← combine
        currepoch_i ← 1
  else if (phase_i = combine) then
    merge(N_i, vals_i)
    ernd_i ← ernd_i + 1
    if (ernd_i > |M|) then
      currepoch_i ← currepoch_i + 1
      ernd_i ← 1
      if (currepoch_i > Ft + 1) then
        phase_i ← output
```

Figure 9-3: ProcessMessage()$_i$ subroutine for $DFC_{t,M}(i)$.

```
PrepareOutput()_i:
  if (phase_i = output) then
    phase_i ← waiting
    m_i ← ⊥
    f_i ← 1
    O_i ← (recv, vclean(vals_i))
  else
    O_i ← ⊥
```

Figure 9-4: PrepareOutput()$_i$ subroutine for $DFC_{t,M}(i)$.

9.4 Proof of Correctness for $DFC_{t,M}$

We first describe the formal theorem statement, then prove helper lemmas that constrain the behavior of $DFC_{t,M}$ for a given emulated round. We conclude by returning to prove the main theorem.

We need to show that for any t, $(n, \mathcal{F}, t+1)$-multiselector M, and t-disrupted channel, $DFC_{t,M}$ implements a $(t, t, 1)$-feedback channel using this channel. We formalize this as follows:

Theorem 9.4.1 *For any $t \in \{0, ..., \mathcal{F} - 1\}$, $(n, \mathcal{F}, t + 1)$-multiselector M, and t-disrupted channel $\mathcal{C}$, there exists a $(t, t, 1)$-feedback channel $\mathcal{C}'$ such that $DFC_{t,M}$ implements $\mathcal{C}'$ using $\mathcal{C}$.*

For the remainder of the proof, fix a specific t, M, and $\mathcal{C}$ that satisfy the constraints of the theorem statement.

Because we do not deal with probabilities in the following proofs, we can fix an arbitrary execution in advance, and have our lemmas apply to an emulated round in this arbitrary execution. Specifically, fix an execution α from a system including a channel environment, $DFC_{t,M}$, and $\mathcal{C}$. Assume it contains no partial emulated rounds. (That is, it ends with the final round of an emulated round.) For simplicity, in the following we call an emulated round *good*, if and only if no more than $\mathcal{F}$ processes are assigned to broadcast in the message assignment encoded in the input at the beginning of the emulated round. We can restrict our attention to good rounds as our definition of a feedback channel places no restrictions on non-good rounds (the definition properties are only required to hold if $bcount(M) \leq \mathcal{F}$, where $bcount(M)$ is the number of processes in message assignment M that are assigned a non-$\perp$ value.) Therefore, assume the final emulated round in α is good. This is the round our lemmas will concern.

We begin with a lemma that constrains the final output in a good emulated round.

Lemma 9.4.2 *At most t processes output $(recv, \perp)$ at the end of the final emulated round in α.*

Proof. We can divide this emulated round into three phases: broadcast, feedback, combine. We assign each real round of the emulated round to the phase name corresponding to the state of its *phase* variable when it calls *ProcessMessage()*.

We begin by considering the single round of the broadcast phase—the first real round of the emulated round. The first $\mathcal{F}^3 + \mathcal{F}^2$ processes are designated *listeners*. We assign $\mathcal{F}^2 + \mathcal{F}$ processes to each frequency by $lfreq$, and divide the processes on each frequency into $\mathcal{F} + 1$ groups by $fgroup$.

Listeners that receive a message to broadcast in the send-encoded input that initiated the emulated round will ignore their listener duties. Because the emulated round is *good*, however, this applies to no more than $\mathcal{F}$ listeners. Because each frequency divides its listeners into $\mathcal{F}+1$ non-overlapping groups, it follows that every frequency has at least one group composed of $\mathcal{F}$ listeners that actually listened on that frequency. For simplicity, we call such listener groups *behaved*.

We continue with the feedback phase rounds. Each of the $\mathcal{F}^2+\mathcal{F}$ frequency groups gets $|M|$ rounds to disseminate what they received during the broadcast phase round. Consider the $|M|$ rounds dedicated to a behaved group from frequency 1. During each of these M rounds, each of the listeners from the group broadcasts their *vals* vector on a unique frequency, while the other processes listen to the channel assigned to them by the matching function of the multiselector, M.

We claim that no more than t processes can fail to hear from a listener during in every one of these $|M|$ rounds. Assume for contradiction that there exists a set S, consisting of $t+1$ processes, that fail to hear from a listener throughout all of these rounds. Recall that M is a $(n, \mathcal{F}, t+1)$-multiselector. By the definition of such a multiselector, there is some round from among these $|M|$ in which all $t+1$ processes from S are receiving on a unique channel. Because we have a single broadcaster on each channel, the only way to prevent a process from receiving a listener message is for that frequency to be disrupted. No more than t frequencies can be disrupted per round, however, so at least one process from S must receive a message from a listener in this round: providing a contradiction to our assumption on the size of S.

We extend this same argument for a behaved listener group from each of the other frequencies. It follows that for each frequency, all but at most t processes learn the message received on that frequency during the broadcast phase. When the feedback phase concludes, therefore, we have no more than $\mathcal{F}t$ processes—t for each frequency—that do not know *everything*. The remaining processes have a *vals* vector with a value in every position. Call such vectors *complete*.

The purpose of the combine phase is to reduce this number of processes without complete vectors from $\mathcal{F}t$ to t. The algorithm dedicates $|M|$ rounds to each of the $\mathcal{F}t+1$ mutually disjoint combine groups defined by *cgroup*. Each group contains $\mathcal{F}$ processes. They disseminate on unique frequencies while the other processes listen according to M—just as in the feedback phase.

At least one of these combine groups is composed entirely of processes with complete *vals* vectors (as there are $\mathcal{F}t+1$ groups and no more than $\mathcal{F}t$ non-complete vectors). By the same argument deployed for the feedback phase, during the $|M|$ rounds assigned to this combine group, all but at most t processes receive this complete *vals* vector and therefore make their own vector complete.

At the end of the emulated round, only processes without complete *vals* vectors output $(recv, \perp)$. As we just proved, this describes no more than t processes, as needed. □

We constrained the number of $\perp$ outputs at the end of an emulated round. We now prove that the non-$\perp$ outputs all equal the same feedback vector.

Lemma 9.4.3 *Let s be the channel state generated by applying $crand_C$ in the single broadcast phase round of the final emulated round in α. Let M and F be the message assignment and frequency assignment, respectively, encoded in the process inputs at the beginning of the emulated round. It follows that there exists an $R \in recvAll(M, F, fblocked_C(s)))$, such that every process outputs either $(recv, \perp)$ or $(recv, R)$ at the end of the emulated round.*

Proof. The output at the end of the emulated round is determined by each process i passing its *vals* vector to *vclean*. If the vector is not complete (i.e., still contains "undefined" in at least one location), *vclean* returns $\perp$, and the process outputs $(recv, \perp)$. If the vector is complete, *vclean* returns the vector $vals_i$, and the process outputs $(recv, vals_i)$.

We first argue that all processes with a complete vector have the same vector. To prove this claim, we describe the two ways that a position in a local *vals* vector can be changed from "undefined" to a message.

1. During the *ProcessMessage()* sub-routine of the broadcast phase round, listeners can directly set the position corresponding to the frequency on which they listened. They set the value to the value from $\mathcal{R}_\perp$ returned from the channel during this round. In addition, note that because the definition of a t-disrupted channel includes the basic broadcast receive function, we know that all listeners who receive on the same frequency will receive the same thing. Therefore, the listeners on a given frequency who modify their *vals* vector at this position will all set it to the same value.

2. The application of the *merge* function during the feedback and combine phases can also modify the *vals* vector. This function, however, only combines a process's *vals* vector with a *vals* vector received from another process. It does not introduce new values.

We can conclude from our above two points that any two processes that complete the emulated round with a complete *vals* vector, have the same vector. Furthermore, this vector describes exactly what was received on each of the frequencies during the broadcast phase round. During this round, messages were broadcast according to M and F, and frequencies were blocked according to $fblocked_{\mathcal{C}}(s)$. With this in mind, the common complete vector clearly satisfies the constraints of set $recvAll(M, F, fblocked_{\mathcal{C}}(s)))$. □

Having established our key lemmas about emulated rounds, we return to the proof for our main theorem.

Proof (Theorem 9.4.1). By Theorem 5.2.10 we know that $DFC_{t,M}$ implements channel $\mathcal{C}(DFC_{t,M}, \mathcal{C})$ using $\mathcal{C}$. For simplicity, we introduce the shorthand notation $\mathcal{C}' = \mathcal{C}(DFC_t, \mathcal{C})$. Our goal is to prove that $\mathcal{C}'$ is a $(t, t, 1)$-feedback channel. To do so, we must prove that $\mathcal{C}'$ satisfies the properties of Definition 6.5.5.

This definition requires the existence of the two mappings, $pblocked_{\mathcal{C}'}$ and $fblocked_{\mathcal{C}'}$, that satisfy definition's two conditions. As in our proof for RFC, we address these two conditions in order, defining the mappings as needed.

Proving Condition (1) of the Feedback Channel Definition. We begin with condition (1), which states:

> Let M be a message assignment with $bcount(M) \leq \mathcal{F}$, and let F be a frequency assignment. Then, for every transformable state s_1, the probability that the distribution $crand_{\mathcal{C}'}(s_1)$ returns a state s_2 where $|pblocked_{\mathcal{C}'}(s_2, M, F)| \leq b$, is at least p.

In our case, $b = t$ and $p = 1$. We need to define $pblocked_{\mathcal{C}'}$ such that the above is satisfied for these parameters.

Recall from Definition 5.2.5 that the composition channel has two types of states: *simple* and *complex*. Because the state s_1 from the condition is transformable, we know it is simple. Therefore, $crand_{\mathcal{C}'}(s_1)$ returns only complex states. With this in mind, we only need to define $pblocked_{\mathcal{C}'}$ for complex states. Specifically, given such a state, s_2, and assignments M and F, define $pblocked_{\mathcal{C}'}(s_2, M, F)$ to return the processes that output $(recv, \perp)$ at the end of $s_2.oext(M, F)$. (Notice, this is the same definition as used in the RFC proof.)

Our second step is to show that this definition of $pblocked_{\mathcal{C}'}$ matches the constraints of the condition. Fix some s_1, M, and F that satisfy the preconditions of the condition. Let s_2 be a complex state assigned a non-zero probability by $crand_{\mathcal{C}'}(s_1)$.

As in our RFC proof, we apply Theorem 5.2.12 to $s_2.oext(M, F)$, to derive a corresponding execution of a system with $DFC_{t,M}$, $\mathcal{C}$, and a channel environment, that has the same states of $DFC_{t,M}$ and $\mathcal{C}$, and the same input and output assignments, as in $s_2.oext(M, F)$.

By Lemma 9.4.2, we know that at most t processes output $(recv, \perp)$ in the final emulated round of this execution. The same holds, therefore, for the outputs generated at the end of $s_2.oext(M, F)$.

We have shown, therefore, that for *all* states s_2, such that s_1 transforms to s_2 with non-zero probability, $pblocked_{\mathcal{C}'}(s_2, M, F)$ contains no more than t processes, as needed.

Proving Condition (2) of the Feedback Channel Definition. We continue with condition (2), which states:

> Let M be a message assignment with $bcount(M) \leq \mathcal{F}$, and let F be a frequency assignment. Then, for every receivable state s, there exists an $R \in recvAll(M, F, fblocked_{\mathcal{C}'}(s))$, where, letting $N = crecv_{\mathcal{C}}(s, M, F)$, we have, for every $i \in [n]$:
>
> $$N[i] = \begin{cases} \perp & \text{if } i \in pblocked_{\mathcal{C}'}(s, M, F) \\ R & \text{else} \end{cases}$$

We already defined $pblocked_{\mathcal{C}'}$, so we are left to define $fblocked_{\mathcal{C}'}$ and show that the two functions, when combined, satisfy the above condition.

Our first step is to define $fblocked_{\mathcal{C}'}$. Because s is receivable, we know it is complex. With this in mind, as in the definition of this function from RFC, let s_C be the the channel state added to $s.pre$ in $s.ext$.

We then define:

$$fblocked_{\mathcal{C}'}(s) = fblocked_{\mathcal{C}}(s_{\mathcal{C}}).$$

Our second step is to show that our definitions match the constraints of the condition. Fix some s, M, and F that satisfy the preconditions of the condition. Above, we defined $pblocked_{\mathcal{C}'}(s, M, F)$ to return the processes that output $(recv, \perp)$ in the last emulated round in $s.oext(M, F)$. We are left to show that the processes *not* in this set output the same $R \in recvAll(M, F, fblocked_{\mathcal{C}'}(s))$.

Once again, we apply Theorem 5.2.12 to $s.oext(M, F)$ to derive an execution of a system with $DFC_{t,M}$, $\mathcal{C}$, and a channel environment, that has the same states of $DFC_{t,M}$ and $\mathcal{C}$, and the same input and output assignments, as in $s.oext(M, F)$. We apply Lemma 9.4.3 to this prefix, which tells us that the processes that do not output $(recv, \perp)$, will output $(recv, R)$, where $R \in recvAll(M, F, fblocked_{\mathcal{C}}(s_{\mathcal{C}})))$. The same, therefore, holds for the outputs at the end of $s.oext(M, F)$. Because we defined $fblocked_{\mathcal{C}}(s_{\mathcal{C}}) = fblocked_{\mathcal{C}'}(s)$, it follows that $R \in recvAll(M, F, fblocked_{\mathcal{C}'}(s)))$.

We have shown, therefore, that every process not in $pblocked_{\mathcal{C}'}(s, M, F)$ returns the same $R \in recvAll(M, F, fblocked_{\mathcal{C}'}(s))$, as needed. □

9.5 Time Complexity of $DFC_{t,M}$

We conclude by bounding the time complexity of $DFC_{t,M}$. We first define the running time in terms of $|M|$, and then use our theorems concerning multiselector constructions to provide precise bounds in terms of n, $\mathcal{F}$, and t.

Theorem 9.5.1 *Fix some $t \in \{0, ..., \mathcal{F}-1\}$, $(n, \mathcal{F}, t+1)$-multiselector M, t-disrupted channel $\mathcal{C}$, and channel environment $\mathcal{E}$. Let α be an execution of $(\mathcal{E}, DFC_{t,M}, \mathcal{C})$. Every emulated round in α requires no more than*

$$\Theta\left(\mathcal{F}^2|M|\right)$$

rounds to complete.

Proof. Each emulated round of $DFC_{t,M}$ consists of 1 broadcast phase round, $\mathcal{F}^2 + \mathcal{F}$ feedback phase epochs, each consisting of $|M|$ rounds, and $\mathcal{F}t + 1$ combine phase epochs, each also consisting of $|M|$ rounds. The $\mathcal{F}^2|M|$ term dominates in the asymptotic definition. □

We can reduce this to concrete times in terms of n, $\mathcal{F}$, and t, as follows. As in Chapter 2, we use the notation "we can construct an $(n, \mathcal{F}, t+1)$-multiselector" to indicate that we present in [40] an algorithm that outputs such a multiselector, given the parameters n, $\mathcal{F}$, and $t+1$ as input.

Theorem 9.5.2 *Fix some $t \in \{0, ..., \mathcal{F}-1\}$. The following hold:*

1. *There exists an $(n, \mathcal{F}, t+1)$-multiselector M, such that for any channel environment $\mathcal{E}$, t-disrupted channel $\mathcal{C}$, and execution α of $(\mathcal{E}, DFC_{t,M}, \mathcal{C})$, every emulated round in α requires no more than*

$$O\left(\mathcal{F}^{2.5} e^{\mathcal{F}} \log\left(\frac{n}{\mathcal{F}}\right)\right)$$

real rounds to complete.

2. *We can construct an $(n, \mathcal{F}, t+1)$-multiselector M, such that for any channel environment $\mathcal{E}$, t-disrupted channel $\mathcal{C}$, and execution α of $(\mathcal{E}, DFC_{t,M}, \mathcal{C})$, every emulated round in α requires no more than*

$$O\left(\mathcal{F}^{\mathcal{F}+2} \log^{\mathcal{F}} n\right)$$

real rounds to complete.

Fix some $t \in \{0, ..., \lfloor\sqrt{\mathcal{F}}\rfloor\}$. The following hold:

3. *There exists an $(n, \mathcal{F}, t+1)$-multiselector M, such that for any channel environment $\mathcal{E}$, t-disrupted channel $\mathcal{C}$, and execution α of $(\mathcal{E}, DFC_{t,M}, \mathcal{C})$, every emulated round in α requires no more than*

$$O\left(\mathcal{F}^{2.5} \log\left(\frac{n}{t}\right)\right)$$

real rounds to complete.

4. *We can construct an $(n, \mathcal{F}, t+1)$-multiselector M, such that for any channel environment $\mathcal{E}$, t-disrupted channel $\mathcal{C}$, and execution α of $(\mathcal{E}, DFC_{t,M}, \mathcal{C})$, every emulated round in α requires no more than*

$$O\left(\mathcal{F}^{8} \log^{5} n\right)$$

real rounds to complete.

Proof. We define the appropriate multiselectors in Theorems 2.3.2, 2.3.4, 2.3.3, and 2.3.5, respectively, of Chapter 2, and then use them as parameters to DFC. The needed running times follow from combining the multiselector sizes provided by these theorems and the DFC runtime provided by Theorem 9.5.1. □

Part III

Algorithms for Solving Problems Using a Feedback Channel

In Part III we solve a collection of useful radio network problems. We design our algorithms to work with the feedback channel—harnessing its power to provide concise and intuitive solutions. By then applying the composition algorithm theorem (Theorem 5.1.7), we automatically derive solutions for the more realistic, yet also more difficult, t-disrupted channel—thus demonstrating the promise of our composition-based modeling framework. Specifically, we study k-set agreement (Chapter 11), gossip (Chapter 12), and reliable broadcast (Chapter 13). The acts of agreeing on a set of values, exchanging a set of initial values, and reliably communicating a collection of messages, respectively, provide a potent algorithmic toolkit for any radio network setting.

To emphasize the benefit provided by the feedback channel, consider the solution to gossip described in Chapter 12, and compare it to the solution from [40], a paper in which we solved gossip directly on a t-disrupted channel. The bulk of this 12-page paper was dedicated to the presentation of the algorithm (presented in four different subroutines), explanations of the complicated code, and proof sketches that could provide only intuition regarding correctness. To emphasize this complexity, we have reproduced, in Figures 9-5, 9-6, 9-7, and 9-8, the pseudocode of the algorithm described in [40]. Specifically, this pseudocode describes a deterministic solution for gossip that runs on a t-disrupted channel and assumes the easy case where n and $\mathcal{F}$ are (very) large compared to t.[1] In Chapter 12, by contrast, we are able to present a gossip algorithm (Figure 12-1) that is much simpler than the algorithm described in Figures 9-5, 9-6, 9-7, and 9-8. This increased simplicity is further emphasized by the observation that the pseudcode from [40] is more compact than the pseudocode used in this thesis; that is, [40] often combines multiple steps into a single line of informal pseudocode, and it does not include state type declarations.

We are also able to provide a detailed explanation of our feedback channel-based gossip algorithm behavior in only one page. A proof sketch, of the type acceptable for an extended abstract, might have required an additional half page at most. Instead, we provide a formal proof that still requires only three pages. The implication is that the use of the feedback channel significantly simplified the challenge of solving this problem.

Before continuing, we also note that numerous *other* problems that are useful in a radio network setting would have their solutions simplified by a feedback channel. To name just a few: maintaining replicated state, leader election, data aggregation, and mutual exclusion (i.e., to allocate a scare resource). All of these problems would be aided by the consistency granted by common feedback being provided to all (or most) processes. The three problems we study in this part are meant simply as case studies.

[1]The pseudocode template used in [40] differs slightly from the template used in this thesis, but we trust the reader will be able to understand the basic ideas.

```
Gossip()_i
  L ← a partition of the set {1, ..., F^2} into F sets of size F.
  for e = 1 to |E| do
    knowledgeable ← Epoch(L, knowledgeable, E[e])_i
  L ← a partition of the set {F^2 + 1, ..., 2F^2} into F sets of size F.
  for e = 1 to |E| do
    knowledgeable ← Epoch(L, knowledgeable, E[e])_i
  Special-Epoch(knowledgeable)_i
```

Figure 9-5: Pseudocode for main body of the deterministic gossip algorithm presented in [40]. The subroutines it calls can be found in Figures 9-6 and 9-7. This algorithm was designed to work directly on a t-disrupted channel. It is presented here to emphasize its complexity as compared to our solution from Chapter 12 (see Figure 12-1) that uses a feedback channel.

```
Epoch(L, knowledgable, rnds)_i
  S ← ∅
  if knowledgable = true then
    let S be the set of processes that are not in L and not completed.
    Partition S into ⌈|S|/F⌉ sets of size F.
  for r = 1 to rnds do
    if (knowledgable = true) and (r ≤ ⌈|S|/F⌉) then
      if ∃k ∈ {1, ..., F} : i = S[r][k] then
        schedule i to transmit on channel k.
      if ∃k ∈ {1, ..., F} : i ∈ L[k] then
        schedule i to receive on channel k.
  knowledgable ← Disseminate(L[1], ..., L[c])_i
  return knowledgable
```

Figure 9-6: Pseudocode for the *Epoch* subroutine used by the gossip algorithm from [40] that is reproduced in Figure 9-5. The definition of the *Disseminate* subroutine, used by *Epoch*, can be found in Figure 9-8.

```
Special-Epoch(knowledgeable)_i
  let M be an (n, F, 5t)-multiselector.
  special ← false
  if (knowledgeable = false) or (i has not completed) then
    special ← true
  if knowledgeable = true then
    L ← set of F^2(5t + 1) smallest processes that have completed in a previous epoch.
    Partition L into (5t + 1) sets L_1, ..., L_{t+1} of size F^2.
    Partition each L_k into F sets L_k[1], ..., L_k[F] of size F.
  for s = 1 to 5t + 1 do
    for r = 1 to |M| do
      if special = true then
        schedule i to transmit on channel M_r(i)
      if ∃k : i ∈ L_s[k] then
        schedule i receive on channel k.
    Disseminate(L_s[1], ..., L_s[F])_i
```

Figure 9-7: Pseudocode for the *Special-Epoch* subroutine used by the gossip algorithm from [40] that is reproduced in Figure 9-5.

```
Disseminate(L[1], ..., L[F])_i
  let M be a (n, F, t + 1)-multiselector.
  knowledgeable ← true
  for k = 1 to F do
    for each round r = 1 to |M| do
      if ∃j ∈ {1, ..., F} : i = L[k][j] then
        schedule i to transmit on frequency j.
      if i ∉ L[k] then
        schedule i to receive on frequency M_r(i).
    if i does not receive a message in any of the |M| rounds then
      knowledgeable ← false.
  L' ← an arbitrary subset of {1, ..., n} of size F(Ft + 1).
  Partition L' into Ft + 1 sets L'[1], ..., L'[Ft + 1] of size F.
  for each s = 1 to Ft + 1 do
    for each r = 1 to |M| do
      if ∃j ∈ {1, ..., F} : i = L'[s][j] then
        schedule i to transmit on frequency j.
      if i ∉ L'[s] then
        schedule i to receive on frequency M_r(i).
    if i receives a message in any of the |M| rounds from a process with knowledgeable = true then
      knowledgeable ← true
  return knowledgeable
```

Figure 9-8: Pseudocode for the *Disseminate* subroutine that is called by the *Epoch* subroutine presented in Figure 9-6 and the *Special-Epoch* subroutine presented in Figure 9-7.

Chapter 10

Preliminaries

In this preliminary chapter we provide some definitions used by all three problems studied in Part III.

10.1 The Value Set V

For all of Part III, fix V to be a non-empty set of values. We use V to describe the input values used by the three problems that follow.

10.2 The Input Environment

We define a simple environment that passes down a particular vector from V.

Definition 10.2.1 ($\overline{v}$-input environment) *For each $\overline{v} \in V^n$, we define the unique $\overline{v}$-input environment $\mathcal{E}$ as follows:*

1. $estates_{\mathcal{E}} = \{s_0, s_1\}$, *and* $estart_{\mathcal{E}} = s_0$.
2. $erand_{\mathcal{E}}(s_i)(s_i) = 1$, *and* $erand_{\mathcal{E}}(s_i)(s_{1-i}) = 0$, *for* $i \in \{0, 1\}$.
3. $ein_{\mathcal{E}}(s_0) = \overline{v}$, *and* $ein_{\mathcal{E}}(s_1) = \perp^n$.
4. $etrans_{\mathcal{E}}(s_0, *) = s_1$, *and* $etrans_{\mathcal{E}}(s_1, *) = s_1$.

In other words, the $\overline{v}$-input environment returns $\overline{v}$ as input in the first round, then outputs $\perp^n$ for all subsequent rounds. We note that this environment is delay tolerant (Definition 4.2.1):

Fact 10.2.2 *For all $\overline{v} \in V^n$, the $\overline{v}$-input environment is delay tolerant.*

Proof. The definition of delay tolerance (Definition 4.2.1) requires the existence of a state $\hat{s} = trans(s, \perp^n)$, for each state s of the environment, that satisfies a collection of four conditions. For the $\overline{v}$-input environment, these conditions are trivially satisfied for $\hat{s}_0 = \hat{s}_1 = s_1$. □

The $\overline{v}$-input environment is appropriate for use with one-shot, decision-style problems in which initial values are passed to the processes only at the beginning of an execution, and the processes then eventually generate some output. The three problems studied in Part III happen to be of this form, but environments could be defined for almost any type of conceivable problem.

Chapter 11

k-Set Agreement

In this chapter, we examine a fundamental problem, k-set agreement [16], that yields a straightforward solution. It acts, therefore, as a warm-up: highlighting the pieces of our modeling framework in action—from problem definitions to our composition theorems—without the distraction of complex algorithmic logic. The problems in the subsequent sections prove more complex.

Informally speaking, the k-set agreement problem, for integer $k \in [n]$, initializes each process with a value from V. Each process must then eventually output a *decision value* from among these initial values, such that there are no more than k different values output. In the formal problem definition given below (Definition 11.1.2), we strengthen the problem slightly by requiring processes to output their decision values during the same round. We also introduce a success probability, to accommodate randomized solutions.

11.1 Problem Definition

Before formalizing the k-set agreement problem, we first introduce a helper definition.

Definition 11.1.1 ($DS(\overline{v}, k)$) *Given an n-vector $\overline{v} \in V^n$ and integer $k \in [n]$, we define the the decision set $DS(\overline{v}, k)$ to consist of every finite trace $\overline{v}, \overline{v}'$ where $\overline{v}' \in V^n$ satisfies the following:*

1. *$\overline{v}'$ contains no more than k distinct values.*
2. *Every value contained in $\overline{v}'$ is also contained in $\overline{v}$.*

We now formalize the problem.

Definition 11.1.2 ($SETA_{k,p}$) *The problem $SETA_{k,p}$, for $k \in [n]$ and probability p, is defined as follows for every environment $\mathcal{E}$:*

1. *If there exists $\overline{v} \in V^n$ such that $\mathcal{E}$ is the $\overline{v}$-input environment, then $SETA_{k,p}(\mathcal{E})$ is the set consisting of every trace probability function F, such that:*

$$\sum_{\beta \in DS(\overline{v},k)} F(\beta) \geq p.$$

2. *Else, $SETA_{k,p}(\mathcal{E})$ is the set consisting of every trace probability function.*

In other words, if the environment $\mathcal{E}$ is the $\overline{v}$-input environment for some $\overline{v}$ (i.e., behaves as it is supposed to for set agreement), the problem $SETA_{k,p}(\mathcal{E})$ includes every probability function that assigns at least probability p to the set of traces that satisfy the desired agreement properties. Notice that we do not restrict how these functions divide this probability mass among the traces. Instead, we include one function for every possible such division. On the other hand, if $\mathcal{E}$ is not the $\overline{v}$-input environment, then we include every trace probability function in $SETA_{k,p}(\mathcal{E})$. The effect is that if the environment does not behave properly, then the problem is trivially solved.

Combining the definition of $SETA$ with Fact 10.2.2 directly implies the following:

Fact 11.1.3 *The $SETA_{k,p}$ problem is delay tolerant.*

We now turn our attention to solving this problem.

11.2 The $SetAgree_t$ Algorithm

The basic strategy for solving agreement with a feedback channel proceeds as follows. First, schedule more processes to broadcast than the number of frequencies that can be disrupted. Second, apply some deterministic function to the resulting feedback set to determine the decision value. For example, we might have every process return the smallest value from this common set, by some fixed ordering.

Complications arise, however, when some problems fail to receive the feedback. To accommodate this possibility, we have each such process decide *its own* initial value. So long as the number of such processes is strictly less than k, we still solve k-set agreement. If we want to solve the problem with at least some probability p, we are fine if the probability parameter in our feedback channel is at least p.

We capture this strategy with the $SetAgree_t$ algorithm presented in Figure 11-1. In this algorithm, we define $minval(N_i)$, where N_i is the receive value from a feedback channel, and $N_i \neq \perp$, to return the smallest value included in this feedback vector, by some fixed ordering. As we prove below, the algorithm solves k-set agreement with probability p, using a (t, b, p')-feedback channel, provided that $p' \geq p$ and $b < k$.

11.3 Correctness of the $SetAgree_t$ Algorithm

Theorem 11.3.1 *Fix some $k, b \in [n]$, $k > b$, $t \in \{0, ..., \mathcal{F} - 1\}$, and probabilities p, p', $p' \geq p$. Let $\mathcal{C}$ be a (t, b, p')-feedback channel. Then $SetAgree_t$ time-free solves problem $SETA_{k,p}$ using $\mathcal{C}$.*

Local state for $SetAgree_t(i)$:
  $m_i \in V \cup \{\perp\}$, initially $\perp$.
  $f_i \in [\mathcal{F}]$, initially 1.

For round $r = 1$:
  $I_i \leftarrow INPUT_i()$
  if $(i \leq t+1)$ then
    $m_i \leftarrow I_i$
    $f_i \leftarrow i$
  $BCAST_i(f_i, m_i)$
  $N_i \leftarrow RECV_i()$
  if $(N_i = \perp)$ then
    $O_i \leftarrow I_i$
  else
    $O_i \leftarrow minval\{N_i\}$
  $OUTPUT_i(O_i)$

Figure 11-1: The $SetAgree_t$ protocol. The pseudocode presented is for process $SetAgree_t(i)$. (Note: V is a fixed non-empty value set, and *minval* returns the smallest value contained in a feedback vector, by some fixed ordering.)

Proof. Let $\mathcal{E}$ be a $\overline{v}$-input environment for some n-vector $\overline{v} \in V^n$. Informally speaking, we know that when we execute $SetAgree_t$ with $\mathcal{E}$ and $\mathcal{C}$, the processes that receive the feedback all output the same value—the minimum value, as determined by *minval*, returned in the feedback vector. The remaining blocked processes each output their own initial value. With probability $p' > p$ no more than $b < k$ are blocked. It follows that with this same probability no more than k values are output, satisfying the problem definition.

To formalize this argument in terms of our feedback channel definition, we construct a one round execution of system $(\mathcal{E}, SetAgree_t, \mathcal{C})$, where $\mathcal{E}$ is the arbitrary input environment fixed above. By condition 1 of the feedback channel property definition (Definition 6.5.5), we know that with probability at least p', the initial state of $\mathcal{C}$, C_0, transforms to a state R_1^C, such that $pblocked_{\mathcal{C}}(R_1^C) \leq b$. (Recall, $pblocked_{\mathcal{C}}$ is a mapping, required to exist by Definition 6.5.5, that describes the processes that receive $\perp$ from feedback channel.) Fix a one-round execution of $(\mathcal{E}, SetAgree_t, \mathcal{C})$ for which this property holds. Call this execution α. We continue by arguing about the process outputs in this same round of α. We will show that the resulting trace is in the decision set $DS(\overline{v}, k)$.

Let $B = fblocked_{\mathcal{C}}(R_1^C)$. (Recall, $fblocked_{\mathcal{C}}$ is a mapping, required to exist by Definition 6.5.5, that describes the disrupted frequencies in the round.) Let N be the receive assignment returned by $\mathcal{C}$ during this round. By the second condition of the feedback channel property, we know that there exists a $R \in recvAll(M, F, B)$ such that at least $n - |pblocked_{\mathcal{C}}(R_1^C)| \geq n - b$ processes receive R in the first round of α.

By the definition of *recvAll* (Definition 6.5.1), this feedback vector, R, contains at least one of the $t+1$ initial values broadcast in the first round (only the values broadcast on the frequencies in B are lost, and $|B| \leq t$). It follows that every process

that receives R, outputs the same initial value, as returned by *minval*. The other processes output their own initial values. The total number of initial values output in α, therefore, is no more than $b + 1 \leq k$. By the definition of the algorithm, all processes output $\perp$ in all subsequent rounds in all extensions of α. Therefore, the trace generated by α, once the $\perp^n$ vectors are removed, is in $DS(\overline{v}, k)$.

We now combine the pieces. We first argued that with probability $p' \geq p$, our system generates an execution such that the in the first round, no more than b processes receive $\perp$ from the feedback channel. We then argued that in any such execution, α, the resulting trace, once the $\perp^n$ vectors are removed, is in $DS(\overline{v}, k)$. It follows:

$$\sum_{\beta \in DS(\overline{v},k)} D_{tf}(\mathcal{E}, SetAgree_t, \mathcal{C}, \beta) \geq p.$$

Therefore, the D_{tf} trace probability function defined for our system is in $SETA_{k,p}$. We conclude that $SetAgree_t$ time-free solves $SETA_{k,p}$ with $\mathcal{C}$. □

11.4 Composing with Feedback Channel Implementations

We use the composition algorithm and our feedback channel implementations to extend $SetAgree_t$ to work with a t-disrupted channel.

Theorem 11.4.1 *Fix some $t \in \{0, ..., \mathcal{F} - 1\}$, t-disrupted channel $\mathcal{C}$, $(n, \mathcal{F}, t + 1)$-multiselector M, and integer $\epsilon \geq 1$. Let $p = 1 - (1/n^\epsilon)$. Then:*

1. *The algorithm $\mathcal{A}(SetAgree_t, RFC_{\epsilon,t})$ time-free solves $SETA_{1,p}$ using $\mathcal{C}$.*
2. *The algorithm $\mathcal{A}(SetAgree_t, DFC_{t,M})$ time-free solves $SETA_{t,1}$ using $\mathcal{C}$.*

Proof. We begin with result 1. By Theorem 8.4.1, the algorithm $RFC_{\epsilon,t}$ implements a $(t, 0, 1 - (1/n^\epsilon))$-feedback channel using $\mathcal{C}$. By Theorem 11.3.1, $SetAgree_t$ time-free solves $SETA_{1,p}$ using this channel. By Fact 11.1.3 we know $SETA_{1,p}$ is delay tolerant. Therefore we can apply Theorem 5.1.7 to conclude that $\mathcal{A}(SetAgree_t, RFC_{\epsilon,t})$ time-free solves $SETA_{1,p}$ using $\mathcal{C}$. Result 2 follows from a symmetric argument. □

This is the first time in this thesis that we deploy our composition algorithm theorem (Theorem 5.1.7). At first glance, the use of this theorem with our channel implementation algorithms from Part II might seem odd. The composition algorithm theorem proof argues about algorithms that *implement* channels, a definition stated in terms of the probabilities of message traces given a channel and a channel environment. The channel implementation algorithm proofs, and the k-set agreement algorithm proofs from above, by contrast, make use of the properties of a channel automaton. The explanation for this apparent incongruity is that the automaton

properties used to define a feedback channel, are, in reality, simply a convienent way to specify the probability of message traces generated by a channel for a given source of messages.

It also important to notice that two results from the above theorem—one for RFC and one for DFC—are not directly comparable. Combining $SetAgree_t$ with RFC allows us to solve 1-consensus (a strong problem), but with only a probabilistic correctness guarantee. Combining $SetAgree_t$ with DFC, by contrast, only solves $(t+1)$-consensus, but it does so with probability 1. We can imagine different scenarios in which each would be the most appropriate choice. For example, if the correctness of the agreement was the foundation for a mission critical behavior, we might tolerate an increased value of k in return for the guarantee of correctness. Whereas, in other contexts, we might tolerate some (small) probability of a correctness violation in exchange for agreeing on a single value.

Time Complexity. The $SetAgree_t$ algorithm requires only a single round to generate its outputs. Its time complexity when composed with RFC and DFC, therefore, matches the time complexity of these algorithms emulating a single round of the feedback channel. We know from Theorem 8.5.1 that each emulated round of RFC requires

$$\Theta\left(\frac{\mathcal{F}^2(\epsilon+1)\log n}{\mathcal{F}-t}\right)$$

real rounds. We know from Theorem 9.5.1 that each emulated round of DFC requires

$$\Theta\left(\mathcal{F}^2|M|\right)$$

real rounds. The size of multiselector M depends on the size of $\mathcal{F}$ relevant to t. Here, we consider only the worst case where $\mathcal{F} < t^2$ (the other case follows in the same manner). We can apply Theorem 9.5.2, in this case, to derive that there *exists* a multiselector M such that our composition requires

$$\Theta\left(\mathcal{F}^{2.5}e^{\mathcal{F}}\log\left(\frac{n}{\mathcal{F}}\right)\right)$$

real rounds to complete, and we can *construct* a multiselector such that our composition requires

$$\Theta\left(\mathcal{F}^{\mathcal{F}+2}\log^{\mathcal{F}} n\right)$$

real rounds.

Chapter 12

(γ, δ)-Gossip

In this chapter, we continue with the gossip problem [47]. The classic definition of this problem initializes each process with a rumor. The goal is then for *all* processes to learn *all* rumors. To accommodate the difficulty of a radio network with frequency disruptions, we generalize the problem to a variant we call (γ, δ)-gossip (first introduced in [31]). This variant requires that at least $\lceil (1-\delta)n \rceil$ processes receive at least $\lceil (1-\gamma)n \rceil$ rumors. (Note: $(0,0)$-gossip matches the classic definition cited above.) To maximize the utility of such a solution—and simplify the problem description—we require that all processes that output a set of values output the *same* set, and that this output occurs during the *same* round.

12.1 Problem Definition

Before formalizing the (γ, δ)-gossip problem, we first introduce some helper definitions.

Definition 12.1.1 ((n,γ)-multiset) *We define a (n,γ)-multiset, for real number γ, $0 \leq \gamma \leq 1$, and integer $n > 0$, to be any multiset of values from V of size s, $\lceil (1-\gamma)n \rceil \leq s \leq n$. We say an (n,γ)-multiset is* **compatible** *with an n-vector from V^n if and only if for every value v that appears k times in the multiset, the same value appears in at least k positions in the n-vector.*

Definition 12.1.2 ($GS(\overline{v}, \gamma, \delta)$) *Given an n-vector $\overline{v} \in V^n$, and two real numbers, γ, δ, $0 \leq \gamma, \delta \leq 1$, we define the gossip set $GS(\overline{v}, \gamma, \delta)$ to consist of every finite trace $\overline{v}, \overline{v}'$ where $\overline{v}'$ is an n-vector that satisfies the following:*

1. *There exists an (n,γ)-multiset S, compatible with $\overline{v}$, such that every position in $\overline{v}'$ contains either S or $\perp$. That is, $\forall i \in [n] : \overline{v}'[i] \in \{S, \perp\}$.*
2. *No more than $\lfloor \delta n \rfloor$ positions in $\overline{v}'$ contain $\perp$. That is, $|\{i \in [n] : \overline{v}'[i] = \perp\}| \leq \lfloor \delta n \rfloor$.*

Each finite trace in $GS(\overline{v}, \gamma, \delta)$, describes a solution to (γ, δ)-gossip for rumors $\overline{v}$. We formalize this notion below.

Definition 12.1.3 ($GOS_{\gamma,\delta,p}$) *The problem $GOS_{\gamma,\delta,p}$, for real numbers, γ, δ, $0 \leq \gamma, \delta \leq 1$, and probability p, is defined as follows for every environment $\mathcal{E}$:*

1. *If there exists a $\overline{v} \in V^n$ such that $\mathcal{E}$ is the $\overline{v}$-input environment, then $GOS_{\gamma,\delta,p}(\mathcal{E})$ is the set consisting of every trace probability function F, such that:*

$$\sum_{\beta \in GS(\overline{v},\gamma,\delta)} F(\beta) \geq p.$$

2. *Else, $GOS_{\gamma,\delta,p}(\mathcal{E})$ is the set consisting of every trace probability function.*

This problem definition shares its structure with $SETA_{k,p}$ (Definition 11.1.2). Also as with $SETA_{k,p}$, combining the definition of GOS with Fact 10.2.2 directly implies the following:

Fact 12.1.4 *The $GOS_{\gamma,\delta,p}$ problem is delay tolerant.*

We now turn our attention to solving this problem. We begin by describing an efficient solution that works with a (t, b, p)-feedback channel, where $\mathcal{F} > t + b$. To understand why we require this constraint on $\mathcal{F}$, imagine that the processes use the feedback from the previous round to schedule a unique broadcaster on each frequency for the next round. Up to b processes might not have the feedback and therefore not schedule themselves to broadcast as they otherwise should. An additional t processes might schedule themselves, but broadcast on disrupted frequencies. If $\mathcal{F} > t + b$, it follows that there is *still* at least one process that schedules itself, broadcasts, and is not disrupted—therefore ensuring the amount of disseminated information grows.

After describing this solution we discuss a modification that makes use of a generalized form of the multiselector object from Chapter 2. This generalized solution works for any $t, b < \mathcal{F}$, but at the cost of increased time complexity.

12.2 The $FastGossip_{\gamma,t,b}$ Algorithm

The $FastGossip_{\gamma,t,b}$ algorithm, presented in Figure 12-1, provides an efficient solution to gossip on a (t, b, p)-feedback channel, where $n, \mathcal{F} > t + b$. Specifically, using this channel, the algorithm solves the $GOS_{\gamma,\frac{t}{n},p^{GOSMAX}}$ problem in $GOSMAX$ rounds, where $GOSMAX = \lceil \frac{(1-\gamma)n}{\mathcal{F}-t-b} \rceil$, as defined in the pseudocode.

Helper Functions. Before proceeding with the pseudocode description, we first define the following helper functions that are used in the pseudocode to streamline its presentation:

- $bcastsched(S_i \subseteq [n], vals_i \in (V \cup \{undefined\})^n, i \in [n])$: The function behaves differently depending on the size of S_i. We divide the behavior into two cases:

Constants for $FastGossip_{\gamma,t,b}(i)$:
  $GOSMAX = \lceil \frac{(1-\gamma)n}{\mathcal{F}-t-b} \rceil$

Local state for $FastGossip_{\gamma,t,b}(i)$:
  $vals_i \in (V \cup \{undefined\})^n$, initially $undefined^n$.
  $knowledgeable_i \in \{true, false\}$, initially $true$.
  $S_i \subseteq [n]$, initially $[n]$.
  $m_i \in V \cup \{\perp\} \cup (V \cup undefined)^n$, initially $\perp$.
  $f_i \in [\mathcal{F}]$, initially 1.

For all rounds $r \leq GOSMAX$:
  $I_i \leftarrow INPUT_i()$
  if $(r = 1)$ then
    $vals_i[i] \leftarrow I_i$
  if $(knowledgeable_i = true)$ then
    $m_i \leftarrow bcastsched(S_i, vals_i, i)$
    $f_i \leftarrow freqsched(S_i, i)$
  else
    $m_i \leftarrow \perp$
  $BCAST_i(f_i, m_i)$
  $N_i \leftarrow RECV_i()$
  if $(N_i = \perp)$ then
    $knowledgeable_i \leftarrow false$
  else
    $knowledgeable_i \leftarrow true$
    $update(S_i, vals_i, N_i)$
  if $(r = GOSMAX$ and $knowledgeable_i = true)$ then
    $OUTPUT_i(clean(vals_i, S_i))$
  else
    $OUTPUT_i(\perp)$

Figure 12-1: The $FastGossip_{\gamma,t,b}$ protocol. If executed with a (t, b, p)-feedback channel, where $\mathcal{F} > t + b$, and passed $\gamma \geq \frac{t+b}{n}$, it solves $GOS_{\gamma,\frac{b}{n},pGOSMAX}$. The code presented is for process $FastGossip_{\gamma,t,b}(i)$. (Note: V is a fixed, non-empty value set.)

1. $|S_i| > t + b$: if i is one of the $min\{\mathcal{F}, |S_i|\}$ smallest values in S_i, then it returns $vals_i$, otherwise it returns $\perp$.
2. $|S_i| \leq t + b$: if $i \leq t + b + 1$ then it returns $vals_i$, otherwise it returns $\perp$.

- $freqsched(S_i \subseteq [n], i \in [n])$: If $bcastsched(S_i, vals_i, i)$ would return $vals_i$ (for any arbitrary $vals_i$), then the function returns $ord(S_i, i)$ (that is, i's position in ascending order in S_i), otherwise it returns 1.

- $update(S_i \subseteq [n], vals_i \in (V \cup \{undefined\})^n, N_i \in \mathcal{R}_\perp)$: The function updates both S_i and $vals_i$ according to a receive assignment N_i generated by a feedback channel.

  - *To update* $\mathbf{vals_i}$ it applies the following logic: If there exists a process index j such that $vals_i[j] = undefined$, and there is a $vals$ vector in N_i that includes a value for j, the function sets $vals_i[j]$ to this value.

  - *To update* $\mathbf{S_i}$ it sets $S_i = [n] \setminus procs(N_i)$, where $procs(N_i)$ returns every process j such that a value for j is included in a $vals$ vector in N_i.

- $clean(vals_i \in (V \cup \{undefined\})^n, S_i \subseteq [n])$: The function returns a version of $vals_i$ modified such that for every $j \in S_i$, $vals_i[j] = bot$.

Algorithm Description. At a high level, the algorithm works as follows. The set S_i describes the processes that have not yet *disseminated* their value—that is, had their value returned in a feedback vector by the feedback channel. The boolean variable $knowledgeable_i$ describes whether process i has the latest information for S_i. (If the process has the latest information, $knowledgeable_i = true$, otherwise it is $false$.) During each round, the $\mathcal{F}$ smallest values in S_i are supposed to broadcast on unique frequencies. Unknowledgeable processes, however, do not have the latest information for S_i, so do not know whether they are supposed to broadcast. Any process that receives the feedback vector from the channel—instead of $\perp$—gains enough information to update its S_i and become knowledgeable.

In more detail, during the first round, each process i initializes its position in its $vals$ vector to the initial value it received as input ($vals_i[i] \leftarrow I_i$). The process's decision of whether or not to broadcast depends on the variable $knowledgeable_i$. (This variable is initially set to $true$, but its value can change as the execution proceeds.) If $knowledgeable_i = true$, then process i sets its broadcast message to $bcastsched(S_i, vals_i, i)$ and its frequency to $freqsched(S_i, i)$. These two helper functions, as defined above, return, $\perp$ and 1, respectively, to all but the $min\{|S_i|, \mathcal{F}\}$ smallest processes. These small processes, by contrast, get their $vals$ vector and a unique frequency on which to broadcast. (The behavior is slightly different if $|S_i| < t + b$, in which case the functions return the $vals$ vectors to the $t + b + 1$ smallest processes. This is needed to ensure that there remains at least one unblocked $vals$ vector in every round after S_i becomes less than $t + b$.)

After broadcasting according to m_i and f_i, and then receiving N_i from the feedback channel, process i does the following. If $N_i = \perp$—i.e., it did not receive the feedback from the channel—then process i sets $knowledgeable_i \leftarrow false$, as it does not know which processes succeeded in disseminating their values in this round. By contrast, if i *does* receive the feedback vector, it sets $knowledgable_i \leftarrow true$ and then calls the helper function *update* with S_i, $vals_i$, and N_i. As defined above, this helper function updates S_i by removing the processes that succeeded in disseminating in this round, and then adds any new values learned in the feedback to the $vals_i$ vector.

Finally, the process determines its output for the round. If $r \neq GOSMAX$, or $r = GOSMAX$ but $knowledgeable_i = false$, it outputs $\perp$. By contrast, if $r = GOSMAX$ and $knowledgeable_i = true$, it returns the $clean(vals_i, S_i)$. As defined above, *clean* sets, for every $j \in S_i$, $vals_i[j] = \perp$. (This is needed to ensure that every *vals* vector output by a process is the same. Without this call to *clean*, it is possible that a process i originally set $vals_i[i] = I_i$, but then never succeeded in disseminating this information to the other processes. If i did not reset $vals_i[i] = \perp$ with *clean*, it would be the only process with a value in that position in its output.)

12.3 Correctness of the $FastGossip_{\gamma,t,b}$ Algorithm

We now present the main correctness theorem.

Theorem 12.3.1 *Fix a (t, b, p)-feedback channel $\mathcal{C}$, for some $t \in \{0, ..., \mathcal{F} - 1\}$, $b \in \{0, ..., \mathcal{F} - t - 1\}$, and probability p. Fix some real values γ and δ, where $\frac{t+b}{n} \leq \gamma \leq 1$ and $\frac{b}{n} \leq \delta \leq 1$. Let $p' \leq p^{\lceil \frac{(1-\gamma)n}{\mathcal{F}-t-b} \rceil}$. It follows that $FastGossip_{\gamma,t,b}$ time-free solves $GOS_{\gamma,\delta,p'}$ using $\mathcal{C}$.*

Proof. Let $\mathcal{E}$ be a $\overline{v}$-input environment for some n-vector $\overline{v} \in V^n$. In this proof we demonstrate that with probability at least p', the system $(\mathcal{E}, FastGossip_{\gamma,t,b}, \mathcal{C})$ generates a trace such that when we remove the $\perp^n$ vectors, the trace is in $GS(\overline{v}, \gamma, \delta)$. This is sufficient to prove the theorem.

By the first condition of the feedback channel property definition (Definition 6.5.5), we know that for any round r of an execution of $(\mathcal{E}, FastGossip_{\gamma,t,b}, \mathcal{C})$, with probability at least p, the transformable channel state, C_r, at the beginning of round r, will transform to a receivable state, R_r^C, where $pblocked_{\mathcal{C}}(R_r^C) \leq b$. (Recall that $pblocked_{\mathcal{C}}$ is a mapping, required to exist by the feedback channel property definition, that determines the processes that receive $\perp$ in a round.)

Therefore, with probability at least $p^{GOSMAX} = p^{\lceil \frac{(1-\gamma)n}{\mathcal{F}-t-b} \rceil}$, a $GOSMAX$-round execution of the system $(\mathcal{E}, FastGossip_{\gamma,t,b}, \mathcal{C})$, will satisfy this property in every round. As we continue with the proof, we restrict our attention to one such execution, α.

We prove two claims about α that build to our needed final result that the trace generated by α, once the $\perp^n$ vectors are removed, is in the gossip set $GS(\overline{v}, \gamma, \delta)$.

> *Claim 1:* For each round $r \in [GOSMAX]$ of α, no more than b processes start the round with $knowledgeable = false$. For any two processes $i, j \in [n]$ that

start r with their *knowledgeable* variable equal to *true*, it follows that $S_i = S_j$ at this point.

We prove this claim by induction. The base case ($r = 1$) follows from the fact that every process i starts the first round with $knowledgeable_i = true$ and $S_i = [n]$. To prove the inductive step, assume the claim holds for some $r \in [GOSMAX - 1]$. By the definition of α, at least $n - b$ processes will receive the feedback vector in this round and subsequently set $knowledgeable \leftarrow true$. They will then start round $r + 1$ with this same value, satisfying the first part of the claim. To prove the second part, consider two processes i and j that both receive the common feedback vector R in r. (Recall, by the definition of a feedback channel property, every process that receives the feedback receives the same information.) Both processes will set $S_i = S_j = [n] \setminus procs(R)$, in their call to *update* in r, satisfying the second part of the claim.

We continue our next claim. In the following, we use the notation S^r, $r \in [GOSMAX]$, to describe the common S set calculated at the end of round r in α, by the processes that received feedback in this round.

Claim 2: For every round $r \in [GOSMAX - 1]$ in α, $S^{r+1} \subseteq S^r$. Furthermore, if $|S^r| > t + b$ then $|S^r| - |S^{r+1}| \geq min\{\mathcal{F}, |S^r|\} - b - t$.

Fix some $r \in [GOSMAX - 1]$. By Claim 1, we know that at the beginning of round $r + 1$ at least $n - b$ processes are knowledgeable and have the same set S^r. These processes call *bcastsched* and $freqsched$ with this common S^r. We consider two cases for the values returned.

First, if $|S^r| > t + b$, there exist $x = min\{\mathcal{F}, |S|\}$ processes that have their *vals* vector and a unique frequency returned by these functions—if they call them (unknowledgeable processes do not call these functions). At least $x - b$ of these processes are knowledgeable, and therefore do call the functions and receive their assignments.

Consider the feedback vector returned by the feedback channel in this round. To do so, let R^C_{r+1} be the receivable state of the feedback channel in round $r + 1$. Let $B = fblocked_C(R^C_{r+1})$. (Recall that $fblocked_C$ is a mapping, required by the definition of the feedback channel property, that describes the disrupted frequencies in this round.) And let M and F be the message and frequency assignments for this round. By the second condition of the feedback channel property, and the definition of α, we know that there exists a feedback vector $R \in recvAll(M, F, B)$, such that at least $n - |pblocked_C(R^C_{r+1})| \geq n - b$ processes receive R. By the definition of *recvAll*, this feedback vector R contains at least one of the $x - b \geq 1$ *vals* vectors broadcast in this round (only the values broadcast on the frequencies in B will be lost, $|B| \leq t$, and $x - b > t$).

By definition, $S^{r+1} = [n] \setminus procs(R)$. We want to show that $S^{r+1} \subseteq S^r$, as stated in the claim. Assume for contradiction that S^{r+1} contains a value j that is not in S^r—thus invalidating a subset relationship. By definition, if $j \notin S^r$, its value was described in the feedback during round r. The knowledgeable processes that broadcast in $r+1$,

however, received this feedback and therefore included j's value in their *vals* vectors. Because at least one such processes has its *vals* vector included in the $r+1$ feedback (by our above argument), j will not be included in S^{r+1}, a contradiction. It follows that $S^{r+1} \subseteq S_r$.

Furthermore, we know that each of the $x - b - t \geq 1$ knowledgeable processes that have their *vals* vectors included in R, were in S^r (otherwise they could not be scheduled to broadcast in $r+1$). They are not, however, included in S^{r+1}, as we just defined them to have their values in R. This proves the additional property that $|S^r| - |S^{r+1}| \geq x - b - t$.

The second case to consider for Claim 2 is when $|S^r| \leq t + b$. By the definition of our helper functions, for this size of S^r, the first $t + b + 1$ processes are scheduled to broadcast on unique frequencies. By the same argument as above, at least $t+b+1-b = t+1$ of these processes are knowledgeable at the start of $r+1$, and therefore actually broadcast as assigned. At least $t + 1 - t = 1$ of these processes avoid a disrupted frequency, and subsequently has its *vals* vector returned in the feedback. This *vals* vector includes every value from the feedback of round r, therefore S_{r+1} cannot include any process that was not included in S^r. It follows that $S^{r+1} \subseteq S^r$, as needed for this case.

Having proved our two claims, we can return to the theorem statement. Consider the last round $r = GOSMAX$ in α. Up to b (but no more) of these processes might fail to receive the feedback in this round. These processes output $\perp$.

Every other process i, outputs $clean(vals_i, S_i)$. By our argument for claim 1, we know that these knowledgeable processes pass the same set, S^{r+1}, to *clean*. To bound the size of this set, we deploy claim 2 and the value of $GOSMAX$. Specifically, we show that $|S^{r+1}| \leq \gamma n \geq t + b$. Claim 2 establishes that if $|S^r| > t + b$, then $|S^r| - |S^{r+1}| \geq min\{\mathcal{F}, |S^r|\} - b - t$. We note that the S^r set can have a size in $\{t + b + 1, ..., \mathcal{F} - 1\}$ for only one round. During this one round, by our above argument, all but at most $t+b$ will have their values included in the feedback, bringing the size down to $t + b$. And once the size gets to $t + b$ or below, we are done (i.e., the size is less than or equal to γn for any allowable γ value).

It follows that for $GOSMAX = \lceil \frac{(1-\gamma)n}{\mathcal{F}-t-b} \rceil$ rounds, we remove $\mathcal{F} - t - b$ values in each round. The exception is the final round, in the case where $(1 - \gamma)n$ does not divide evenly by $\mathcal{F} - t - b$, which has the size of S in $\{t+b+1, ..., \mathcal{F}-1\}$. In this final round, however, we remove enough to get the size of S to $t + b$ or below, as needed.

We next note that by the definition of *clean*, we know that processes that output in the final round set the entry in *vals* for processes in S^{r+1} to $\perp$.

We are left to show that no knowledgeable process i can have an entry j in $vals_i$, such that $j \notin S^{r+1}$ and $vals_i[j] = \perp$. Assume for contradiction that this occurs. Because $vals_i[j] = \perp$, we know no value for j was included in the feedback in this final round (otherwise, i's call to *update* would have updated $vals_i[j]$ to a non-$\perp$ value). If j's value is not in this feedback, however, then S^{r+1} must include j—a contradiction.

It follows that all *vals* vectors are the same, and include $|S^{r+1}| \leq \gamma n$ $\perp$ values. Furthermore, at most $b \leq n\delta$ processes output $\perp$. And by the definition of the

algorithm, every process outputs $\perp$ in every subsequent round of every extension of α. Therefore the resulting trace of α, once the $\perp^n$ vectors are removed, is in $GS(\overline{v}, \gamma, \delta)$.

We now combine the pieces of our proof. We first argued that with probability $p^{GOSMAX} \geq p'$, our system generated an execution such that in each of the first $GOSMAX$ rounds, no more than b processes receive $\perp$ from the feedback channel. We then argued that in any such execution, α, the resulting trace, once the $\perp^n$ vectors are removed, is in $GS(\overline{v}, \gamma, \delta)$. It follows:

$$\sum_{\beta \in GS(\overline{v},\gamma,\delta)} D_{tf}(\mathcal{E}, FastGossip_{\gamma,t,b}, \mathcal{C}, \beta) \geq p'.$$

Therefore, the D_{tf} trace probability function defined for our system is in $GOS_{\gamma,\delta,p'}$. We conclude that $FastGossip_{\gamma,t,b}$ time-free solves $GOS_{\gamma,\delta,p'}$ using $\mathcal{C}$. □

12.4 Composing with Feedback Channel Implementations

We use the composition algorithm and our feedback channel implementations to construct the following gossip solutions for a t-disrupted channel. The proofs below are more complex than the corresponding proofs for the set agreement algorithm. This is due to the fact that the gossip problem has more parameters to argue about.

Theorem 12.4.1 *Fix some $t \in \{0, ..., \mathcal{F} - 1\}$ and t-disrupted channel $\mathcal{C}$. The algorithm $\mathcal{A}(FastGossip_{\gamma,t,0}, RFC_{\epsilon+1,t})$ time-free solves $GOS_{\gamma,\delta,p'}$ using $\mathcal{C}$, for any γ, δ, p' where $\frac{t}{n} \leq \gamma \leq 1$, $0 \leq \delta \leq 1$, and $p' \leq 1 - (1/n^{\epsilon})$, for some constant $\epsilon \geq 1$.*

Proof. Fix some constants $t, \gamma, \delta, \epsilon, p'$ and channel $\mathcal{C}$ that satisfy the constraints of the theorem statement. By Theorem 8.4.1, the algorithm $RFC_{\epsilon+1,t}$ implements a $(t, 0, 1 - (1/n^{\epsilon+1}))$-feedback channel using $\mathcal{C}$. Before we deploy our algorithm composition theorem, we must first show that $FastGossip_{\gamma,t,0}$ time-free solves $GOS_{\gamma,\delta,p'}$ using any feedback channel with these same parameters.

We know by Theorem 12.3.1 that given a (t, b, p)-feedback channel $\mathcal{C}'$, where $\mathcal{F} > t + b$, $FastGossip_{\gamma,t,b}$ time-free solves $GOS_{\gamma,\delta,p'}$, for any $\frac{t+b}{n} \leq \gamma \leq 1$, $\frac{b}{n} \leq \delta \leq 1$, and $p' \leq p^{GOSMAX}$, using $\mathcal{C}'$.

In our above formulation, $b = 0$, so $t + b = t < \mathcal{F}$. In addition, we fixed $\frac{t+b}{n} = \frac{t}{n} \leq \gamma \leq 1$ and $\frac{b}{n} = 0 \leq \delta \leq 1$. It follows that our definitions of t, b, γ and δ satisfy what is needed to apply Theorem 12.3.1. We are left to show that p' in our above formulation is no more than $(1 - (1/n^{\epsilon+1}))^{GOSMAX}$. Let $p = 1 - (1/n^{\epsilon+1})$. We know the probability of $GOSMAX$ successes (each occurring with probability p) is the same as $1 - p_{fail}$, where p_{fail} is the probability of at least one failure. We know each failure occurs with probability $1 - p = (1/n^{\epsilon+1})$. By a union bound over $GOSMAX$ rounds, p_{fail} is no larger than $(1/n^{\epsilon})$. Putting the pieces together, we bound $(1 - (1/n^{\epsilon+1}))^{GOSMAX} \geq 1 - (1/n^{\epsilon}) = p'$, as needed.

We have established that $FastGossip_{\gamma,t,b}$ time-free solves $GOS_{\gamma,\delta,p'}$ for a feedback channel with the parameters provided by $RFC_{\epsilon+1,t}$ (and the constraints of the theorem statement). This observation, plus the fact that *FastAgree* is delay tolerant (Fact 12.1.4), when combined with our composition algorithm theorem (Theorem 5.1.7), proves the theorem. □

We now make a similar argument using our deterministic implementation of a feedback channel.

Theorem 12.4.2 *Fix some $t \in \{0, ..., \lfloor \mathcal{F}/2 \rfloor - 1\}$, t-disrupted channel $\mathcal{C}$, and $(n, \mathcal{F}, t+1)$-multiselector M. The algorithm $\mathcal{A}(FastGossip_{\gamma,t,t}, DFC_{t,M})$ time-free solves $GOS_{\gamma,\delta,p'}$ using $\mathcal{C}$, for any γ, δ, p' where $\frac{2t}{n} \leq \gamma \leq 1$, $\frac{t}{n} \leq \delta \leq 1$, and $0 \leq p' \leq 1$.*

Proof. Fix some constants t, γ, δ and p', multiselector M, and channel $\mathcal{C}$, that satisfy the constraints of the theorem statement. By Theorem 9.4.1, the algorithm $DFC_{t,M}$ implements a $(t, t, 1)$-feedback channel using $\mathcal{C}$. As in our preceding result, before we deploy our algorithm composition theorem, we must first show that $FastGossip_{\gamma,t,t}$ time-free solves $GOS_{\gamma,\delta,p'}$ using any feedback channel with these same parameters.

We know by Theorem 12.3.1 that given a (t, b, p)-feedback channel $\mathcal{C}'$, where $\mathcal{F} > t + b$, $FastGossip_{\gamma,t,b}$ time-free solves $GOS_{\gamma,\delta,p'}$, for any $\frac{t+b}{n} \leq \gamma \leq 1$ $\frac{b}{n} \leq \delta \leq 1$, and $p' \leq p^{GOSMAX}$, using $\mathcal{C}'$.

In our above formulation, $b = t$. We fixed t to be less than $\lfloor \mathcal{F}/2 \rfloor$, so it follows that $t + b < \mathcal{F}$. We also fixed $\frac{t+b}{n} = \frac{2t}{n} \leq \gamma \leq 1$ and $\frac{b}{n} = \frac{t}{n} \leq \delta \leq 1$.

It follows that our definitions of t, b, γ and δ satisfy what is needed to apply Theorem 12.3.1. We are left to show that p' in our above formulation is no more than p^{GOSMAX}, where p is the probability parameter from our feedback channel. Fortunately, as we established above, $p = 1$, and we set $p' \leq 1$, so clearly $p' \leq p^{GOSMAX} = 1$ as needed.

We have established that $FastGossip_{\gamma,t,b}$ time-free solves $GOS_{\gamma,\delta,p'}$ for a feedback channel with the parameters provided by $DFC_{t,M}$ (and the constraints of the theorem statement). This observation, plus the fact that *FastAgree* is delay tolerant (Fact 12.1.4), when combined with our composition algorithm theorem (Theorem 5.1.7), proves the theorem. □

Time Complexity. We continue with the time complexity. We know that the $FastGossip_{\gamma,t,b}$ algorithm requires $GOSMAX = \lceil \frac{(1-\gamma)n}{\mathcal{F}-t-b} \rceil$ rounds to generate output. The composition $\mathcal{A}(FastGossip_{\gamma,t,0}, RFC_{\epsilon+1,t})$, therefore, requires $GOSMAX$ emulated rounds of the feedback channel. We know from Theorem 8.5.1 that each emulated round requires

$$\Theta\left(\frac{\mathcal{F}^2(\epsilon+1)\log n}{\mathcal{F}-t}\right)$$

real rounds. It follows that the composition requires

$$\Theta\left(\frac{\mathcal{F}^2(\epsilon+1)(1-\gamma)n\log n}{(\mathcal{F}-t)^2}\right)$$

rounds. By comparison, the randomized gossip solution presented in [32], which was constructed to run directly on a t-disrupted channel, requires

$$\Theta\left(nt^2\log n\right)$$

rounds to disseminate $n-t$ values (i.e., $\gamma=\frac{t}{n}$.) This time complexity nearly matches the complexity of *FastGossip* composed with *RFC* for this same γ value—indicating that the layered approach adds minimal extra overhead in this case.

The composition $\mathcal{A}(FastGossip_{\gamma,t,t}, DFC_{t,M})$ also requires $GOSMAX$ emulated rounds of the feedback channel. We know from Theorem 9.5.1 that each emulated such round requires

$$\Theta\left(\mathcal{F}^2|M|\right)$$

real rounds. The size of multiselector M depends on the size of $\mathcal{F}$ relevant to t. We first consider the worst case where $\mathcal{F}<t^2$. We can apply Theorem 9.5.2 to derive that there *exists* a multiselector M such that our composition requires

$$\Theta\left(\frac{\mathcal{F}^{2.5}e^{\mathcal{F}}(1-\gamma)n\log\frac{n}{\mathcal{F}}}{\mathcal{F}-2t}\right)$$

rounds, and we can *construct* a multiselector such that our composition requires

$$\Theta\left(\frac{\mathcal{F}^{\mathcal{F}+2}(1-\gamma)n\log^{\mathcal{F}}n}{\mathcal{F}-2t}\right)$$

rounds. For the better case of $\mathcal{F}\geq t^2$, we can apply Theorem 9.5.2 to derive that there exists a multiselector M such that our composition requires

$$\Theta\left(\frac{\mathcal{F}^2t(1-\gamma)n}{\mathcal{F}-2t}\log\left(n/t\right)\right)$$

rounds. We can compare this final result to the $O(n)$ time complexity achieved in [40] for solving gossip on a t-disrupted channel with $\gamma=\frac{t}{n}$ and $\mathcal{F}\geq t^2$. Our time complexity is roughly a factor of $t^3\log\left(n/t\right)$ slower than the algorithm in [40]. The t^3 factor is avoided in [40] because this algorithm uses a faster feedback strategy that takes advantage of the assumption that $\mathcal{F}\geq t^2$. Our DFC algorithm could potentially be modified to behave differently for this case of large $\mathcal{F}$, and perhaps improve its efficiency. For simplicity of presentation, however, we it designed to work the same for all t and $\mathcal{F}$.

The multiplicative log factor from our bound is transformed into an additive factor in [40] by delaying feedback for multiple rounds while multiple groups of processes, that have not yet disseminated their values, broadcast. The feedback is then aggregated and returned all at once. In terms of *FastGossip*, this would be equivalent to partitioning S into groups of size $\mathcal{F}$, then letting each of these groups broadcast before receiving the aggregate feedback that spans these rounds. To accommodate this behavior we would have to slightly generalize the definition of the feedback channel to allow a specification of a *feedback schedule* (i.e., a description of which rounds should return aggregate feedback).

12.5 Generalizing the $FastGossip_{\gamma,t,b}$ Algorithm

A key constraint of the $FastGossip_{\gamma,t,b}$ algorithm is that $\mathcal{F} > t+b$. As explained, this requirement allows us to make progress even if up to $t+b$ of $\mathcal{F}$ potential concurrently broadcasting processes fail—due to frequency disruptions or lack of global knowledge from missing the previous feedback.

It is also easy to see that for $\mathcal{F} \leq t$, gossip is impossible for any non-trivial values of γ and δ, as the channel can disrupt *every* frequency in *every* round, preventing any communication between processes.

We are left, therefore, to consider the intermediate case where $t < \mathcal{F} \leq t+b$. In this section we *discuss* a generalization of *FastGossip* that works under these difficult constraints. We stop short of actually formalizing this generalization, as the details add little beyond the intuition.

Summary of the Generalization. At the core of the *FastGossip* algorithm was the strategy of scheduling at least $t+b+1$ processes to broadcast concurrently in every round. Up to t of these scheduled processes might be disrupted by the channel. An additional b might fail to broadcast because they lack the global knowledge needed to make the scheduling decision. This leaves at least 1 process to broadcast successfully.

To explain our generalized strategy, assume the worse case of $\mathcal{F} = t-1$. For this value of $\mathcal{F}$ and any $b > 0$, the strategy described above will fail, as the combination of disruption and lack of knowledge can prevent *any* broadcast from succeeding.

We can alleviate this problem, however, by using a *generalized* multiselector (as defined Chapter 2). At the beginning of a round r, the up to b processes that failed to receive feedback in round $r-1$ know that they failed to receive feedback. They do not know, however, whether or not they should be broadcasting, and, if so, on what frequency.

Imagine that we extend the broadcasting to include one round for each function from a generalized $(n, \mathcal{F}, \mathcal{F}, t+b)$-multiselector M^G. (To prove the existence of such a generalized multiselector using Theorem 2.3.7, this will require that $n \geq 2(t+b)$.) During each round of this new *broadcast phase*, each process that is either (a) knowledgeable and scheduled to broadcast; or (b) is unknowledgeable, uses the corresponding function of M_G to determine on what frequency to broadcast in that round. If the function maps the process to 0, it does not broadcast.

Let S be the set of processes that satisfy (a) and (b) from above. This set is no larger than $t + b$. Let S' be the $\mathcal{F}$ processes that *should* broadcast in this phase according to the helper functions. By the definition of the generalized multiselector, there will be a round during the phase in which only these processes broadcast, and they do so on unique channels. Because no more than t can choose a disrupted frequency, some process will succeed in having its value disseminated.

In this sense, the multiselector compensates for a *bounded* lack of knowledge. That is, in each round, there is a known upper bound on the number of unknowledgeable nodes.

The limit to this approach, of course, is the added time complexity. The size constraints of Theorem 2.3.7 are onerous: the resulting runtime will be exponential in some function of t and b. Whether more efficient generalized multiselectors exist, and whether there exist reasonably compact protocols for constructing them, remain interesting open questions.

Chapter 13

(c, p)-Reliable Broadcast

We conclude Part III with a study of the reliable broadcast problem [15]. The standard definition for this problem requires every process to eventually receive every broadcast. (In our model, this might be captured by an environment passing down a broadcast message to a single process as input and then, eventually, requiring every process to return that message as output.)

On a (t, b, p)-feedback channel, where $b > 0$, we are hindered by the reality that up to b processes might be permentantly blocked from receiving messages from the feedback channel. Presumably, this would prevent them from learning about the reliable broadcast messages of other processes. With this in mind, we generalize the problem to require that at least $n - c$ processes receive each message, for some parameter $c \geq b$. To accommodate the reality of a randomized feedback channel implementation, we also introduce a success probability p.

For simplicity, we focus on a *one-shot variant* of the problem that provides some subset of processes a message to broadcast in the first round, and then, in some subsequent round, every process simultaneously outputs the complete set of messages they received.[1]

In more detail, the (c, p)-reliable broadcast problem, for integer $c \geq 0$, and probability p, assumes an environment that passes message values from V, to some subset of the processes, in the first round. (Without loss of generality, we can assume it gives each process a unique message.) In some round $r \geq 1$, every process outputs the set of messages they have received. We require that these sets include only messages that were actually passed to processes in the first round. We also require that with probability p: for every such message m, at least $n - c$ sets include m. For $c = 0$ and $p = 1$, this defines classic reliable broadcast.

Comparing Reliable Broadcast to Gossip. There are obvious similarities between the (c, p)-reliable broadcast problem and the (γ, δ)-gossip problem of the pre-

[1]The algorithm described in Figure 13-1 would likely work for a reasonably-defined *online variant* of the problem (i.e., one in which processes could keep receiving messages as input, but perhaps would have to send an acknowledgement token to the environment before a new message could be passed down). Such a variant, however, is more complicated to define formally. That is, the environment constraints would be complicated to specify.

vious chapter. Conceptually, where they differ, however, is in the process input. For the gossip problem, *every* process receives an initial value; a sufficient fraction of the processes then need to output a set containing a sufficient fraction of these values. In the reliable broadcast problem, by contrast, some *arbitrary subset* of the processes receive an initial value. They then need to disseminate *all* of these values to a sufficient fraction of the processes. It is true that a solution to reliable broadcast could *almost* be used to solve $(0, \frac{c}{n})$-gossip (the "almost" refers to the fact that gossip requires every non-$\perp$ output set to be the same, whereas reliable broadcast does not, so long as each message is included in enough output sets). For larger parameter values, however, a reliable broadcast solution might not be very efficient, as the set of processes receiving values is *a priori* unknown—preventing a solution from using the type of fast deterministic strategy showcased by the *FastGossip* algorithm in Chapter 12. (This fast solution depends on the guarantee that *every* process receives a value.) Practically speaking, however, an important argument for the inclusion of both problems is that our gossip solution is deterministic, whereas our reliable broadcast solution is randomized. We wanted to include a randomized algorithm in Part III to demonstrate the randomized analysis of an algorithm using a feedback channel.

13.1 Problem Definition.

Before formalizing the (c, p)-reliable broadcast problem, we first introduce the following helper definition.

Definition 13.1.1 ($BS(\overline{v}, c)$) *Given an n-vector $\overline{v} \in (V_\perp)^n$, such that no value $v \in V$ appears in two or more positions in the vector, and an integer $c \in [n]$, we define the broadcast set $BS(\overline{v}, c)$ to consist of every finite trace $\overline{v}, \overline{v}'$ where $\overline{v}'$ is an n-vector that satisfies the following:*

1. *Let $vals(\overline{v})$ be the set of non-$\perp$ values in $\overline{v}$. For every $i \in [n]$: $\overline{v}'[i] = S_i \subseteq vals(\overline{v})$.*
2. *For every $v \in vals(\overline{v})$, there exists at least $n - c$ positions j such that $v \in S_j$.*

We now formalize our problem definition in the same manner as for set agreement (Definition 11.1.2) and gossip (Definition12.1.3).

Definition 13.1.2 ($RBCAST_{c,p}$) *The problem $RBCAST_{c,p}$, for $c \in [n]$ and probability p, is defined as follows for every environment $\mathcal{E}$:*

1. *If there exists a $\overline{v} \in (V \cup \{\perp\})^n$ such that there exists no $v \in V$ in two or more positions in $\overline{v}$, and $\mathcal{E}$ is the $\overline{v}$-input environment, then $RBCAST_{c,p}(\mathcal{E})$ is the set consisting of every trace probability function F, such that:*

$$\sum_{\beta \in BS(\overline{v},c)} F(\beta) \geq p.$$

2. *Else, $RBCAST_{c,p}(\mathcal{E})$ is the set consisting of every trace probability function.*

Combining the definition of $RBCAST$ with Fact 10.2.2 directly implies the following:

Fact 13.1.3 *The $RBCAST_{c,p}$ problem is delay tolerant.*

We now turn our attention to solving this problem. At the core of our solution is the same basic randomized strategy used in the RFC algorithm. That is, processes select frequencies, and decide whether to broadcast, at random.

13.2 The $RBcast_{t,p,k}$ Algorithm

The $RBcast_{t,p,k}$ algorithm, described in Figure 13-1, time-free solves the $RBCAST_{c,p'}$ problem using any (t, b, p)-feedback channel, where $c \geq b$ and $p' \leq 1-(1/n^k)$. The algorithm *terminates* (that is, has processes output their message set) in $O\left((1/p)kn\mathcal{F}\log n\right)$ rounds.

Helper Function. Before proceeding to the pseudocode description, we first define the following helper function that is used in the pseudocode to streamline its presentation:

- $extract(N_i \in \mathcal{R}_\perp)$: Given a receive value N_i from the feedback channel, such that $N_i \neq \perp$, this helper function returns every message from V in N_i. If $N_i = \perp$, $extract(N_i)$ returns $\perp$.

Algorithm Description. At a high level, the algorithm works as follows. Assume process i is passed message $m \neq \perp$ as input in the first round. For each of the next $RBMAX$ rounds, this process chooses a frequency at random and broadcasts its message with probability $1/n$.

If process i is passed $\perp$ as its first-round input, then it simply receives during each of these $RBMAX$ rounds. In both cases, process i extracts any values from the feedback vectors it receives, and adds them to $outs_i$. After $RBMAX$ rounds, process i outputs $outs_i$. Two things are important to note. First, the algorithm takes advantage of the power of the feedback channel to help it disseminate messages. If process i has a message, for example, it is sufficient that it broadcasts alone once on a non-disrupted frequency—the feedback channel will disseminate this value to other processes in the feedback vector for this round. The second important point is the choice of $RBMAX$. In the proof that follow, we show that $RBMAX$ is sufficiently large to ensure that processes with messages to send succeed in broadcasting alone on a non-disrupted frequency, with sufficiently high probability.

In more detail, process i initializes $outs_i$ to $\emptyset$ and msg_i and m_i to $\perp$. For each round $r \in [RBMAX]$, if it receives a non-$\perp$ input it sets msg_i to this input. Notice,

Constants for $RBcast_{t,p,k}(i)$:
  $RBMAX = \lceil \frac{(k+1)\mathcal{F}^2 n \ln n}{(\mathcal{F}-t)(\mathcal{F}-1)p} \rceil$.

Local state for $RBcast_{t,p,k}(i)$:
  $out_i \in P(V)$, initially $\emptyset$.
  $msg_i, m_i \in V \cup \{\perp\}$, initially $\perp$.
  $f_i \in [\mathcal{F}]$, initially 1.

For all round $r \leq RBMAX$:
  $bit_i \leftarrow RAND_i(\lceil \lg \mathcal{F} \rceil + \lceil \lg n \rceil)$
  $I_i \leftarrow INPUT_i()$
  if $(I_i \neq \perp)$ then
    $msg_i \leftarrow I_i$
  $f_i \leftarrow bit_i[1...\lceil \lg \mathcal{F} \rceil]$
  if $(bit_i[\lceil \lg \mathcal{F} \rceil + 1...\lceil \lg n \rceil] = 0)$ then
    $m_i \leftarrow msg_i$
  else
    $m_i \leftarrow \perp$
  $BCAST_i(f_i, m_i)$
  $N_i \leftarrow RECV_i()$
  $out_i \leftarrow out_i \cup extract(N_i)$
  if $(r = RBMAX)$ then
    $OUTPUT_i(outs_i)$
  else
    $OUTPUT_i(\perp)$

Figure 13-1: The $RBcast_{t,p,k}$ protocol. The code presented is for process $RBcast_{t,p,k}(i)$.

if the algorithm is run with an input environment, this can only happen during the first round. It simplifies the pseudocode, however, to avoid isolating this case.

The process then selects a frequency at random ($f_i \leftarrow bit_i[1...\lceil \lg \mathcal{F} \rceil]$), and with probability $1/n$ (*if* $bit_i[\lceil \lg \mathcal{F} \rceil + 1...\lceil \lg n \rceil] = 0$) it sets m_i to msg_i, and otherwise sets $m_i \leftarrow \perp$. It then broadcasts m_i on f_i. It follows that if process i was passed an message, and therefore msg_i contains this message, it broadcasts this message on a random frequency with probability $1/n$, and otherwise receives. If process i was not passed a message, then regardless of the outcome of this probabilistic decision, it receives.

Finally, the process extracts any received values and adds them to out_i ($out_i \leftarrow out_i \cup extract(N_i)$). At the end of round $RBMAX$, the process outputs $outs_i$. In all other rounds, it outputs $\perp$.

13.3 The Correctness of the $RBcast_{t,p,k}$ Algorithm

We now present the main correctness theorem.

Theorem 13.3.1 *Fix a (t, b, p)-feedback channel $\mathcal{C}$, for some $t \in \{0, ..., \mathcal{F} - 1\}$, $b \in \{0, ..., \mathcal{F} - 1\}$, and probability p. Fix integers $c \geq b$ and $k > 0$, and probability $p' \leq 1 - (1/n^k)$. It follows that $RBcast_{t,p,k}$ time-free solves $RBCAST_{c,p'}$ using $\mathcal{C}$.*

Proof. Fix some $\overline{v}$-input environment $\mathcal{E}$, such that $\overline{v}$ does not include the same $v \in V$ in two or more positions. Fix an $r - 1$ round execution α of the system ($\mathcal{E}$, $RBcast_{t,p,k}$, $\mathcal{C}$). Fix some process i that was passed a message m as input in the first round (i.e, $\overline{v}[i] = m$) of α.

Let us first consider what happens in round r, given this arbitrary prefix α. Let C_r be the channel state of $\mathcal{C}$ at the beginning of round r, and let R_r^C be the state chosen according to the distribution $crand_C(C_r)$ in this round.

By the definition of a feedback channel (Definition 6.5.5), there exists a mapping $fblocked_C$, such that $B = fblocked_C(R_r^C)$ describes the frequencies that are *disrupted* in this round. Process i chooses a frequency f on which to participate in r at random. Formally, this occurs in the probabilistic transformation of the process state in this round, which is independent of the probabilistic transformation of the channel state. It follows that the probability that $f \neq B$ is at least $\frac{\mathcal{F}-|B|}{\mathcal{F}} \geq \frac{\mathcal{F}-t}{\mathcal{F}}$.

By the definition of the algorithm, process i broadcasts m on f in this round, with probability $1/n$. We combine these two probabilities to conclude that process i broadcasts on an undisrupted frequency in r with probability at least $\frac{\mathcal{F}-t}{n\mathcal{F}}$. Call this *Claim* 1. We will refer to it again soon.

This does not guarantee, however, that the message sent by process i is included in the feedback vector returned by the feedback channel in this round. It is possible that another process broadcasts concurrently on f, generating a collision. For any fixed $j \neq i$ that also received a message as input, process j also chooses to broadcast on f in r with probability $\frac{1}{n\mathcal{F}}$.

By a union bound, the probability that at least one process broadcasts concurrently with i is no more than $\frac{n-1}{n\mathcal{F}} < \frac{1}{\mathcal{F}}$. Thus, the probability that *no* process collides with process i on f in r is at least $(1 - \frac{1}{\mathcal{F}}) = \frac{\mathcal{F}-1}{\mathcal{F}}$. Call this *Claim* 2.

We now combine *Claim* 1 and *Claim* 2 to conclude that process i broadcasts alone on a non-disrupted frequency f in round r with probability at least

$$p_{solo} = \frac{(\mathcal{F} - t)(\mathcal{F} - 1)}{\mathcal{F}^2 n}.$$

We now turn our attention to calculating the probability that this message, broadcast alone on an undisrupted frequency, makes it to at least $n-b$ processes. We return, then, to the definition of the feedback channel property. By this definition, there exists a mapping $pblocked_C$, such that $B' = pblocked_C(R_r^C)$ describes the processes that fail to receive the feedback in this round. By the condition 1 of the feedback channel property definition, with probability at least p, $|B'| \leq b$.

(Specifically, given any transformable state, including, C_r, the probability that it transforms to a state R_r^C encoding no more than b blocked processes, is at least p.) Therefore, with probability at least $p_{solo} \cdot p$, process i disseminates m to at least $n - b$ processes in round r.

This probability was calculated for an arbitrary execution α. We can therefore construct a new $RBMAX$-round execution, round by round, applying this result for each round, to determine the probability that i succeeds in disseminating its value to at least $n - b$ processes, at least once.

Notice that we can rewrite

$$RBMAX = \frac{k+1}{p_{solo} \cdot p} \ln n.$$

With this in mind, we bound the probability that i fails to disseminate m in all $RBMAX$ rounds as follows:

$$\begin{aligned} (1 - p_{solo} \cdot p)^{RBMAX} &< e^{-p_{solo} \cdot p \cdot RBMAX} \\ &= e^{-(k+1)\ln n} \\ &= (1/n^{k+1}) \end{aligned}$$

The first step above uses the probability facts from Chapter 2. To extend this argument to all processes with messages, we note that there are up to n such processes. By a union bound, at least one of them fails to disseminate its message during these rounds with probability no greater than $(1/n^k)$. Therefore, with probability at least $(1 - (1/n^k))$, every process that receives a message as input at the beginning of an execution of this system, disseminates its message to at least $n - b \geq n - c$ processes. After $RBMAX$ rounds, every process outputs the full set of received messages, and then outputs $\perp$ in every subsequent round of every extension of the execution. We have shown that with probability at least $p' = (1 - (1/n^k))$, system $(\mathcal{E}, RBcast_{t,p,k}, \mathcal{C})$ generates an execution, α, such that the trace generated by α, once the $\perp^n$ vectors are removed, is in $BS(\overline{v}, c)$. It follows:

$$\sum_{\beta \in BS(\overline{v},c)} D_{tf}(\mathcal{E}, RBcast_{t,p,k}, \mathcal{C}, \beta) \geq p'.$$

Therefore, the D_{tf} trace probability function defined for our system is in $RBCAST_{c,p'}$. We conclude that $RBcast_{t,p,k}$ solves $RBCAST_{c,p'}$ with $\mathcal{C}$. □

Notice that the structure of this proof differs from the structure of the proofs for our set agreement (Theorem 11.3.1) and gossip (Theorem 12.3.1) algorithms. This follows because the algorithm considered here is randomized, while the previous two algorithms are deterministic. To prove the that the deterministic algorithms solve the problem, we split the proof into two parts. The first part bounds the probability that the feedback channel behaves as needed, and the second part proves that if the channel behaves we generate the needed trace. The proof for the randomized reliable broadcast protocol, by contrast, bounds the probability of a given round doing something *good*, where *good* captures both the behavior of the feedback channel *and* the algorithm—because they are both probabilistic, there is no need to handle the behaviors differently—and then applies this argument round by round, to calculate the probability that enough rounds are *good*. Notice that the definition of the feedback channel property (Definition 6.5.5) simplified this decomposition of the feedback probabilities and undisrupted broadcast probabilities. This follows from the definition's constraint that the feedback probability hold regardless of the algorithm's behavior.

13.4 Composing with Feedback Channel Implementations

We use the composition algorithm and our feedback channel implementations to construct the following reliable broadcast solutions for a t-disrupted channel.

Theorem 13.4.1 *Fix some $t \in \{0, ..., \mathcal{F} - 1\}$ and t-disrupted channel $\mathcal{C}$. The algorithm $\mathcal{A}(RBcast_{t,p,k}, RFC_{\epsilon,t})$ time-free solves $RBCAST_{c,p'}$ using $\mathcal{C}$, for any p,k,ϵ, c, and p' where $\epsilon \geq 1$, $p = 1 - (1/n^{\epsilon})$, $c \geq 0$, $p' \leq 1 - (1/n^{k})$.*

Proof. Fix some constants t, p, k, ϵ, c, p' and channel $\mathcal{C}$ that satisfy the constraints of the theorem statement. By Theorem 8.4.1, the algorithm $RFC_{\epsilon,t}$ implements a $(t, 0, 1 - (1/n^{\epsilon}))$-feedback channel using $\mathcal{C}$. Before we deploy our algorithm composition theorem, we must first show that $RBcast_{t,p,k}$ time-free solves $RBCAST_{c,p'}$ using any feedback channel with these same parameters.

We know by Theorem 13.3.1 that given a (t, b, p)-feedback channel $\mathcal{C}'$, $RBcast_{t,p,k}$ time-free solves $RBCAST_{c,p'}$, if $c \geq b$ and $p' = (1 - (1/n^{k}))$. In our case, $b = 0$ and we define p' as required.

Combining the observation, and the fact that $RBcast_{t,p,k}$ is delay-tolerant (Fact 13.1.3), with our composition theorem (Theorem 5.1.7), proves the theorem statement. □

Theorem 13.4.2 *Fix some $t \in \{0, ..., \mathcal{F}-1\}$, t-disrupted channel $\mathcal{C}$, and $(n, \mathcal{F}, t+1)$-multiselector M. The algorithm $\mathcal{A}(RBcast_{t,1,k}, DFC_{t,M})$ time-free solves $RBCAST_{c,p}$ using $\mathcal{C}$, for any k,c, and p where $c \geq t$ and $p \leq 1 - (1/n^k)$.*

Proof. The proof follows the same structure as Theorem 13.4.1, with the exception that $b = t$ in the feedback channel implemented by $DFC_{t,M}$. □

Time Complexity. The $RBcast_{t,p,k}$ algorithm requires $RBMAX = \lceil \frac{(k+1)\mathcal{F}^2 n \ln n}{(\mathcal{F}-t)(\mathcal{F}-1)p} \rceil = O((1/p)kn\mathcal{F}\log n)$ rounds. The composition $\mathcal{A}(RBcast_{t,p,k}, RFC_{\epsilon,t})$, therefore, requires $RBMAX$ emulated rounds of the feedback channel. We know from Theorem 8.5.1 that each emulated round requires

$$\Theta\left(\frac{\mathcal{F}^2 \epsilon \log n}{\mathcal{F}-t}\right)$$

real rounds. It follows that the composition requires

$$O\left(\frac{k \cdot \epsilon \cdot n\mathcal{F}^3 \log^2 n}{p(\mathcal{F}-t)}\right)$$

real rounds. The composition $\mathcal{A}(RBcast_{t,p,k}, DFC_{t,M})$ also requires $RBMAX$ emulated rounds of the feedback channel. We from Theorem 9.5.1 that each of these emulated rounds require

$$\Theta\left(\mathcal{F}^2|M|\right)$$

real rounds. The size of multiselector M depends on the relationship of $\mathcal{F}$ to t. For the worst case where $\mathcal{F} < t^2$, we can apply Theorem 9.5.2 to derive that there *exists* a multiselector M such that our composition requires

$$O\left(\frac{k \cdot n \cdot e^{\mathcal{F}} \mathcal{F}^{3.5} \log^2 n}{p}\right)$$

real rounds, and we can *construct* a multiselector such that our composition requires

$$O\left(\frac{k \cdot n\mathcal{F}^{\mathcal{F}+3} \log^{\mathcal{F}+1} n}{p}\right)$$

real rounds.

Part IV

Ad Hoc Networks

In Part IV we address a useful variant of the model described in Chapter 3. This original model includes strong assumptions, namely: all devices start during the same round and they (potentially) know the total number of devices participating in the system. In some radio network environments, however, these assumptions prove unrealistic. Consider, for example, *ad hoc networks*. Devices in this setting typically come together in an unpredictable fashion—perhaps, for example, the PDAs of conference attendees in the same room attempt to coordinate. In this context we should not assume that the devices start during the same round. We should also not assume that the exact number of devices that will eventually participate is known in advance—though a reasonable upper bound on this number *might* be available; for example, in the PDA scenario, it is unlikely that more than, say, 500 such devices will gather in the same room.

In Chapter 14, we define a modified version of our modeling framework from Chapter 3 that captures these attributes of an ad hoc setting. We call this variant the *ad hoc radio network model*. It allows devices to start during different rounds and does not guarantee that all n devices will participate. (In this sense, the model parameter n becomes an upper bound on the total potential number of participants.)

In Chapter 15, we define the *wireless synchronization problem*, which requires the devices in ad hoc radio network to eventually agree on a common round numbering. We argue that a solution to this problem can be used to adapt *any algorithm* written for our original modeling framework from Chapter 3 to run in our ad hoc variant. This is, solving this problem can facilitate a powerful translation of results between models. With this in mind, we present the $Trapdoor_t$ algorithm, first described in [30], which solves the wireless synchronization algorithm and comes close to matching the best known lower bound. We then discuss a collection of strategies for accomplishing our above-mentioned goal of using such a solution to adapt algorithms from the non-ad hoc framework to operate in the ad hoc variant presented in Chapter 14.

Chapter 14

The Ad Hoc Radio Network Model

To define the *ad hoc radio network model* we introduce a collection of modifications to the radio network model presented in Chapter 3. In the following, we assume that the input alphabet, $\mathcal{I}$, includes a special token, *wake*.

14.1 Definitions

We first define a type of process that we call an *ad hoc process*. These processes are initialized in a *quiet* state, and then remain in this state until they receive a special *wake* input from the environment. We then introduce *ad hoc* variants of algorithms and executions that build upon these modified processes.

Definition 14.1.1 (Ad Hoc Process) *An ad hoc process $\mathcal{P}$ is defined the same as in Definition 3.1.3, with the addition of the following condition:*

1. *There exists a special* quiet *state* $\mathbf{quiet}_{\mathcal{P}} \neq \mathbf{start}_{\mathcal{P}}$ *such that:*

   (a) $rand_{\mathcal{P}}(quiet_{\mathcal{P}})(quiet_{\mathcal{P}}) = 1$.

   (b) $msg_{\mathcal{P}}(quiet_{\mathcal{P}}, I \in \mathcal{I}_{\perp}) = \perp$.

   (c) $out_{\mathcal{P}}(quiet_{\mathcal{P}}, I \in \mathcal{I}_{\perp}, m \in \mathcal{R}_{\perp}) = \perp$.

   (d) $trans_{\mathcal{P}}(quiet_{\mathcal{P}}, m \in \mathcal{R}_{\perp}, I \in \mathcal{I}_{\perp} \setminus \{wake\}) = quiet_{\mathcal{P}}$.

   (e) $trans_{\mathcal{P}}(quiet_{\mathcal{P}}, m \in \mathcal{R}_{\perp}, wake) = start_{\mathcal{P}}$.

In other words, an ad hoc process loops in a special *quiet* state until it receives a *wake* input, at which point it transitions to its start state.

The definition of an ad hoc algorithm incorporates this process definition:

Definition 14.1.2 (Ad Hoc Algorithm) *An ad hoc algorithm $\mathcal{A}$ is a mapping from $[n]$ to ad hoc processes.*

These modified definitions of processes and algorithms require a corresponding modified definition of an execution:

Definition 14.1.3 (Ad Hoc Execution) *An ad hoc execution α of a system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$, where $\mathcal{A}$ is an ad hoc algorithm, is defined the same as Definition 3.2.1 of Chapter 3, with the exception that the first condition is replaced with the following:*

1. $\forall i \in [n] : S_0[i] = quiet_{\mathcal{A}(i)}$.

The rest of the definitions from Chapters 3 and 4 remain the same with the exception that we add the modifier *ad hoc* in front of any reference to a *process*, *algorithm*, or *execution*.

14.2 Pseudocode for Ad Hoc Processes

To describe an ad hoc process, we modify the pseudocode template of Section 7.2. Specifically, in this template, which is defined for a process i, we replace the global round counter, r, with a *local round counter*, r_i. The pseudocode template captures what occurs *after* the process receives a *wake* input. The local round counter starts at 1. These counters, therefore, differ between processes that receive *wake* during different rounds. (We emphasize the local nature of this counter by replacing "For rounds r..." in the template with the text "For local round r_i...").

14.3 Wake Environments

Before continuing, we note the obvious: ad hoc executions are interesting only if the environment wakes up some processes. With this in mind, we introduce the following general constraint for an environment that will prove useful for almost any problem definition in the ad hoc model.

Definition 14.3.1 (P-Wake Environment) *We say an environment $\mathcal{E}$ is a P-wake environment, for $P \subseteq [n]$, if and only if it in every execution of a system including $\mathcal{E}$, the environment satisfies the following:*

1. *Processes in P receive the input wake exactly once.*
2. *Processes not in P never receive the input wake.*

In other words, the P-wake environment will wake up the processes in P, and let the other processes remain in their quiescent state. As you might expect, when we define problems for the ad hoc radio network model we will typically require non-trivial behavior only for environments that are P-wake environments, for some $P \subseteq [n]$.

14.4 Fixing a Model

For the remainder of Part IV, we restrict our attention to the ad hoc radio network model defined above. To simplify notation, we leave out the *ad hoc* notation preceding the relevant terms (process, algorithm, etc.). In the following, its presence is implied.

Chapter 15

The Wireless Synchronization Problem

The wireless synchronization problem, which was first described in [30], requires that the processes activated by a wake environment eventually agree on a global round numbering. Informally, a solution to the problem must satisfy the following five requirements, with respect to some probability p. (In the following, we use the term *active*, with respect to a given round $r > 1$, and a process i, to indicate that process i received input *wake* before round r.)

1. *Validity:* In every round, every active process outputs a value in $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$. If a process outputs a number, then we consider that to be its labelling of the round number; if it outputs $\perp$, then it has not yet determined a round number.

2. *Synch Commit:* Once a process outputs a non-$\perp$ value (in $\mathbb{N}$), it never again outputs $\perp$.

3. *Correctness:* The round number increments in each round: if a process outputs i in round r, then it outputs $i + 1$ in round $r + 1$.

4. *Agreement:* With probability p: in every round, all non-$\perp$ outputs are the same.

5. *Liveness:* Eventually, every active process stops outputting $\perp$.

The *synch commit* property ensures that each process knows when it has successfully synchronized. Once synchronization has been achieved, the round number continues to increment (as per *correctness*). These guarantees ensure that a synchronization routine can safely be used as a building block for a protocol that depends on the round numbers.

15.1 Problem Definition

We continue by formalizing these properties with a precise definition that matches the form of a problem in our framework. In the proof for Theorem 15.4.1, however,

we refer to the informal description of the properties, not the formal mathematical version below, to simplify its presentation.

We begin, as with the other problems studied in this thesis, with a helper definition that captures correct traces.

Definition 15.1.1 ($SS(P)$) *The synch set $SS(P)$, for some $P \subseteq [n]$, contains every infinite trace $T_1, T_2, ..., T_i \in \mathcal{I}_\perp \cup \mathcal{O}_\perp$, that satisfies the following conditions:*

1. *For every positive integer k: $T_k \neq \perp^n$.*

2. *For every positive integer k, $T_k \in (\{\perp\} \cup \mathbb{N})^n$ or $T_k \in \{\perp, wake\}^n$.*
   (The first set describes the possible output assignments—devices either output $\perp$ or a round number—and the second set describes the possible input assignments—the environments we will consider generate only wake and $\perp$ as inputs.)

3. *For every $j \notin P$, and positive integer k: $T_k[j] = \perp$.*
   (Processes not in P do nothing.)

4. *For every $i \in P$, there exists exactly one positive integer, k_i, such that $T_{k_i}[i] = wake$.*
   (The environment wakes up each process in P once.)

5. *Let $S = \{k_j : j \in P\}$. For every $i \in P$, there exists an integer k_i^s, $k_i^s > k_i$, $k_i^s \notin S$, and an integer $r_i > 0$, such that the following hold:*

   (a) *For every integer k, $k \geq k_i^s$, $k \notin S$: $T_k[i] = r_i + m$, where $m = |\{r : k_i^s \leq r \leq k, r \notin S\}|$.*
   (In round k_i^s, process i begins outputting an incrementing sequence of integers, starting with r_i. We are careful, in our definition of m, to not count trace vectors that include wake, as such vectors are associated with environment inputs, not process outputs.)

   (b) *For every integer k, $k < k_i^s$, $k \neq k_i$: $T_k[i] = \perp$.*
   (Before a process begins outputting round numbers, it outputs nothing. To handle trace vectors corresponding to input assignments, we note that only T_{k_i} would contain a non-$\perp$ value for i, a case we handle by restricting $k \neq k_i$.)

6. *For every positive integer k, if there exist $i, j \in P$ such that $T_k[i], T_k[j] \in \mathbb{N}$, then $T_k[i] = T_k[j]$.*
   (All processes that output round numbers in the same trace vector, output the same values.)

Condition 2 of the definition above captures *validity* from the informal definition. Conditions 5.*a* and 5.*b* combine to capture both *synch commit* and *correctness*. Condition 4 captures *liveness* and condition 6 captures *agreement*. The remaining

conditions, 1 and 3, are required to eliminate some malformed traces that otherwise satisfy the above conditions. Specifically, condition 1 eliminates traces with $\perp^n$ vectors, and condition 3 eliminates traces with inputs to and outputs from processes not in P.

We now constrain the environments for this problem.

Definition 15.1.2 (P-synch environment) *We say an environment $\mathcal{E}$ is a P-synch environment, for some $P \subseteq [n]$, if and only if it satisfies the following conditions:*

1. *$\mathcal{E}$ is a P-wake environment.*
2. *$\mathcal{E}$ never outputs any value other than $\perp$ and wake.*

In other words, a P-synch environment eventually wakes up the processes in P, and that is all it does. We can now pull together these pieces into a formal problem definition. The structure below is the same structure used for the problems studied in Part III: the problem, $SYNCH_p$, includes any trace function that assigns *enough* probability mass (i.e., at least p) to traces in SS.

Definition 15.1.3 ($SYNCH_p$) *The problem $SYNCH_p$, for probability p, is defined as follows for every environment $\mathcal{E}$:*

1. *If there exists a $P \subseteq [n]$ such that $\mathcal{E}$ is a P-synch environment, then $SYNCH_p(\mathcal{E})$ is the set consisting of every trace probability function F, such that:*

$$\sum_{\beta \in SS(P)} F(\beta) \geq p.$$

2. *Else, $SYNCH_p(\mathcal{E})$ is the set consisting of every trace probability function.*

Useful Notation. In the following discussion of the wireless synchronization problem, we use the notation *synchronize*, with respect to a process, round, and an execution, to indicate that the process outputs its first non-$\perp$ value in that round of the execution.

As mentioned in the introduction to this chapter, we also use the notation *active* with respect to a process, round $r > 1$, and execution, to indicate that the process received input *wake* in some round $r' < r$ of the execution. We also sometimes use the term with respect to only a round and an execution. In this case, it refers to every process that is active in that round. And sometimes we use it with respect to only an execution. Here, it describes every process that becomes active at some point in the execution.

15.2 Lower Bound

In [30] we prove a lower bound on the time required to synchronize every active processes in a solution to the wireless synchronization problem. The bound holds for a subset of algorithms that we call *regular*. An algorithm, $\mathcal{A}$, is regular if and only if there exists a fixed sequence of pairs $(F_1, b_1), (F_2, b_2)...$, where each F_k is a probability distribution over frequencies and each b_k is a probability, such that for each active process $\mathcal{A}(i)$ and local round r_i (i.e., the r^{th} round after $\mathcal{A}(i)$ receives a *wake* input), if $\mathcal{A}(i)$ has not received a message through $r-1$, it chooses its frequency and whether or not to broadcast according to F_r and b_r, respectively. In other words, all active processes behave in a uniform manner until they receive *some* information—at which point their behavior can deviate in an arbitrary fashion.

We can restate this existing bound in the notation of our formal model, as follows:

Theorem 15.2.1 (From [30]) *Fix some $t \in \{0, ..., \mathcal{F}-1\}$. Let $p = 1-(1/n)$ and $\mathcal{A}$ be a regular algorithm that solves $SYNCH_p$ with any t-disrupted channel. There exists a t-disrupted channel $\mathcal{C}$, P-synch environment $\mathcal{E}$, $P \subseteq [n], |P| > 0$, and an execution of $(\mathcal{E}, \mathcal{A}, \mathcal{C})$, in which some process in P requires:*

$$\Omega\left(\frac{\log^2 n}{(\mathcal{F}-t)\log\log(n)} + \frac{\mathcal{F}t}{\mathcal{F}-t}\log n\right)$$

rounds to synchronize after becoming active.

15.3 The $Trapdoor_t$ Algorithm

In this section, we describe the regular algorithm $Trapdoor_t$. We first presented this algorithm, and sketches of its correctness proof, in [30]. Here, we provide pseudocode (missing from [30]) and fill in the proof details. Specifically, we show that it solves $SYNCH_p$ with any t-disrupted channel $\mathcal{C}$. Furthermore, it guarantees that every active process synchronizes within

$$O\left(\frac{\mathcal{F}}{\mathcal{F}-t}\log^2 n + \frac{\mathcal{F}t}{\mathcal{F}-t}\log n\right)$$

rounds of becoming active. This comes close to matching our lower bound from above. Notably it captures both the $\log^2 n$ and $\frac{\mathcal{F}t}{\mathcal{F}-t}$ terms from the lower bound. It is only off in the first term, which is $\mathcal{F}\log\log n$ times larger in the upper bound as compared to the lower bound.

Helper Function. Before proceeding to the pseudocode description, we first define the following helper function that is used in the pseudocode to streamline its presentation. The function is called *trand*, which stands for *trapdoor randomness*. Given a round number r, and process status variable *status*, it calculates the appropriate broadcast probability for a process with the status described in *status* during round r, and returns the corresponding number of bits that the process should request from

Constant	Value
$\mathcal{F}'$	$min\{2t, \mathcal{F}\}$
p_{ko}	$\frac{\mathcal{F}'-t}{12\mathcal{F}'}\left(\frac{1}{4}\right)^6$
ℓ'	$\frac{6}{p_{ko}} \ln 2n$
ℓ_E	$17\ell'$
p_{fko}	$\frac{\mathcal{F}'-t}{4(\mathcal{F}')^2}\left(\frac{1}{4}\right)^6$
ℓ_E^+	$max\{\frac{3}{p_{fko}} \ln n,\ \ell_E\}$

Figure 15-1: Constants used in the definition of $Trapdoor_t$.

Epoch #	1	2	...	$\lg n - 1$	$\lg n$
Start Round	1	$\ell_E + 1$	...	$(\lg n - 2)\ell_E + 1$	$(\lg n - 1)\ell_E + 1$
End Round	ℓ_E	$2\ell_E$	...	$(\lg n - 1)\ell_E$	$(\lg n - 1)\ell_E + \ell_E^+$
Epoch Length	ℓ_E	ℓ_E	...	ℓ_E	ℓ_E^+
Asymptotic Length	$\Theta\left(\frac{\mathcal{F}'}{\mathcal{F}'-t}\log n\right)$	$\Theta\left(\frac{\mathcal{F}'}{\mathcal{F}'-t}\log n\right)$	...	$\Theta\left(\frac{\mathcal{F}'}{\mathcal{F}'-t}\log n\right)$	$\Theta\left(\frac{(\mathcal{F}')^2}{\mathcal{F}'-t}\log n\right)$
Broadcast Prob.	$1/n$	$2/n$	...	$1/4$	$1/2$

Figure 15-2: Epoch lengths and contender broadcast probabilities for $Trapdoor_t$. Note: $\mathcal{F}' = min\{2t, \mathcal{F}\}$, ℓ_E and ℓ_E^+ are defined in Figure 15-1.

RAND. That is, if the broadcast probability is p, it returns $\lg(1/p)$. The process can then request this many bits from *RAND* and subsequently broadcast if and only if they all evaluate to 0 (an event that occurs with probability p).:

- $trand(r \in \mathbb{N}, status \in \{contender, leader, knockedout\})$
  We consider two cases based on the value of *status*:
  - **Case 1:** $status = contender$.
    - If $r \leq T = (\lg(n) - 1)\ell_E + \ell_E^+$, where ℓ_E and ℓ_E^+ are defined as in Figure 15-1, then let p be the probability from the *Broadcast Prob.* row of Figure 15-2 from the column that includes r in the interval defined by its *Start Round* and *End Round* values. The function then returns $\lceil \lg(1/p) \rceil$.
    - Else, the function returns 1.
  - **Case 2:** $status = leader$ **or** $status = knockedout$.
    The function returns 1.

Constants for $Trapdoor_t(i)$:
(see Figure 15-1)

Local state for $Trapdoor_t(i)$:
$p_i \in \mathbb{N}$, initially 1.
$rnd_i \in \mathbb{N} \cup \{\perp\}$, initially $\perp$.
$status_i \in \{contender, leader, knockedout\}$, initially $contender$.
$m_i \in \{\perp\} \cup (leader \times \mathbb{N}) \cup (contender \times \mathbb{N} \times i)$, initially $\perp$.
$f_i \in [\mathcal{F}]$, initially 1.

For local rounds $r_i > 0$:
  if $(rnd_i \neq \perp)$ then
    $rnd_i \leftarrow rnd_i + 1$
  $p_i \leftarrow trand(r_i, status_i)$
  $bit_i \leftarrow RAND_i(p_i + \lceil \lg \mathcal{F}' \rceil)$
  if $(status_i = contender)$ then
    if $(bit_i[1...p_i] = 0)$ then
      $m_i \leftarrow (contender, r_i, i)$
    else
      $m_i \leftarrow \perp$
  else if $(status_i = leader)$ then
    if $(bit_i[1...p_i] = 0)$ then
      $m_i \leftarrow (leader, r_i)$
    else
      $m_i \leftarrow \perp$
  else if $(status_i = knockedout)$ then
    $m_i \leftarrow \perp$
  $f_i \leftarrow bit_i[p_i + 1...p_i + \lceil \lg \mathcal{F}' \rceil]$
  $BCAST_i(f_i, m_i)$
  $N_i \leftarrow RECV_i()$
  if $(status_i = contender)$ and $N_i = (contender, r_j, j) > (contender, r_i, i)$ then
    $status_i \leftarrow knockedout$
  if $(status_i \neq leader)$ and $N_i = (leader, r_j)$ then
    $status_i \leftarrow knockedout$
    $rnd_i \leftarrow r_j$
  if $(status_i = contender)$ and $(r = (\lg(n) - 1)\ell_E + \ell_E^+)$ then
    $status_i \leftarrow leader$
    $rnd_i \leftarrow r_i$
  $OUTPUT_i(rnd_i)$

Figure 15-3: The $Trapdoor_t$ protocol. The pseudocode presented is for process $Trapdoor_t(i)$. (Note: the inequality $(contender, r_j, j) > (contender, r_i, i)$ is defined in terms of lexicographic order—it evaluates to true if and only if $r_j > r_i$ or $(r_j = r_i$ and $j > i)$.)

Algorithm Description. At a high level, the $Trapdoor_t$ algorithm, presented in Figure 15-3, works as follows. Fix $\mathcal{F}' = min\{\mathcal{F}, 2t\}$. The processes make use of only the first $\mathcal{F}'$ frequencies. (When $\mathcal{F} > 2t$, using more than $2t$ frequencies does not help because the increased probability of avoiding disruption with a randomly selected frequency is balanced out by the decreased probability of two processes finding each other.)

When a process becomes active, it starts as a *contender*. Each contender proceeds through $\lg n$ *epochs*. (For simplicity of notation, assume n is a power of 2.) Each of the first $\lg(n) - 1$ epochs is of length $\ell_E = \Theta\left(\frac{\mathcal{F}'}{\mathcal{F}'-t} \log n\right)$ rounds. The final epoch is of length $\ell_E^+ = \Theta\left(\frac{(\mathcal{F}')^2}{\mathcal{F}'-t} \log n\right)$ rounds. (The precise definitions of these constants are provided in Figure 15-1).

At the beginning of round r of epoch e, every contender chooses a frequency f uniformly at random from $[1, \ldots, \mathcal{F}']$. It then broadcasts a "contender" message on frequency f with probability $\frac{2^e}{2n}$ (see Figure 15-2). The message is labelled with the contender's *timestamp*: a pair (r_a, uid), where r_a is the number of rounds the contender has been active, and *uid* is a unique identifier. Otherwise, it listens on frequency f. If a contender receives a message from another contender, and the sender of that message has a larger timestamp (by lexicographic order), then the receiver is *knocked out* (i.e., the trapdoor opens beneath its feet). A process that is knocked out continues to listen on a channel (chosen randomly from $[1, \ldots, \mathcal{F}']$) in every round. If a contender completes all $\lg n$ epochs without being knocked out, then it becomes a leader.

As soon as a contender becomes a leader, it declares that the current round is equivalent to its local round counter, and begins to output an incrementing round number in every subsequent round. From that point onwards, in every round, it chooses a channel at random (from $[1, \ldots, \mathcal{F}']$) and sends a message containing the current round number, with probability 1/2. Any active non-leader process that receives a message from a leader will be knocked out (if it is not already) and adopt the global round number in the message. It will then start outputting rounds.

In more detail, at the beginning of each round, process i increments its global round counter ($rnd_i \leftarrow rnd_i + 1$), if it does not contain $\perp$. It then calls $trand(r_i, status_i)$, which returns the number of bits corresponding to the probability with which process i should broadcast in local round r_i, given $status_i$. It sets p_i equal to this value. The process proceeds by calling $RAND$, asking for p_i bits to help it make its broadcast decision, plus an additional $\lg \mathcal{F}'$ to help it select a random frequency ($bit_i \leftarrow RAND_i(p_i + \lceil \lg \mathcal{F}' \rceil)$).

Process i then decides which message, if any, to broadcast. There are three cases, depending on the value of $status_i$. If process i is a leader and the first p_i bits returned by $RAND$ equal 0 (an event that happens with probability $1/2^{p_i}$), then it sets m_i to the message $(leader, r_i)$. That is, a message that encodes its local round counter as its proposal for the global round counter. Otherwise it sets $m_i \leftarrow \perp$.

If process i is a contender, it follows a similar logic with respect to its p_i bits. This time, however, if the bits equal 0 it sets its message to $(contender, r_i, i)$. That is, a message announcing that it is a contender, and including a timestamp, (r_i, i),

describing how many rounds i has been active and its id.

(The crucial observation for this case is the selection criteria for p_i. As described in the definition of the helper function *trand*, a contender progresses through $\lg n$ epochs. The first $\lg(n) - 1$ consist of ℓ_E rounds, and the final epoch consists of ℓ_E^+ rounds. During the first epoch, the process uses a broadcast probability of $1/n$. This doubles in each successive epoch until it reaches $1/2$. This behavior is captured in Figure 15-2. The insight behind this doubling is for the processes to adapt their broadcast probability to the *a priori* unknown number of processes active in the system.)

Finally, if process i is knocked out, it sets $m_i \leftarrow \perp$. A knocked out process never broadcasts. Instead, it waits to learn the global round number from a leader, if it has not already.

After m_i is initialized, process i sets f_i to a random frequency ($f_i \leftarrow bit_i[p_i+1...p_i+\lceil \lg \mathcal{F}' \rceil]$), and then broadcasts ($BCAST_i(f_i, m_i)$) and receives ($N_i \leftarrow RECV_i()$).

If process i is a contender in this round and it receives a contender message from a process with a larger timestamp (i.e., if ($status_i = contender$) and $N_i = (contender, r_j, j) > (contender, r_i, i)$, where $>$ is with respect to the lexicographic order), then it changes its status to *knockedout*. If process i is not a leader and it receives a leader message $(leader, r_j)$, it sets its status to *knockedout* and its global round counter to r_j ($rnd_i \leftarrow r_j$). And if process i is a contender, and it made it through all $\lg n$ epochs ($r = (\lg(n) - 1)\ell_E + \ell_E^+$), then it sets its own status to leader and its global round counter to its local round counter.

The final action of process i in the round is to output its global round counter ($OUTPUT_i(rnd_i)$).

15.4 Correctness of the $Trapdoor_t$ Algorithm

We now present the main correctness theorem.

Theorem 15.4.1 *For any $t \in \{0, ..., \mathcal{F} - 1\}$, t-disrupted channel $\mathcal{C}$, and probability $p \leq 1-(1/n)$, the algorithm $Trapdoor_t$ solves $SYNCH_p$ using $\mathcal{C}$. Every active process synchronizes within*

$$O\left(\frac{\mathcal{F}}{\mathcal{F}-t}\log^2 n + \frac{\mathcal{F}t}{\mathcal{F}-t}\log n\right)$$

rounds after being activated.

For the remainder of this section, fix a specific t, $\mathcal{C}$, and p that satisfy the constraints of the theorem. Let $\mathcal{F}' = min\{2t, \mathcal{F}\}$. Assume $\mathcal{F}' \geq 2$, $n \geq 4$, $n \geq \mathcal{F}'$, and n is a power of 2.

Proof Notation. To aid the proofs that follow, we introduce a few additional pieces of notation, all of which are defined with respect to a $r-1$ round execution, $r > 1$, of a system that includes $Trapdoor_t$, $\mathcal{C}$, and a P-synch environment, for some non-empty $P \subseteq [n]$.

- Let p_i^r be the probability that process i broadcasts in round r.
- Let $W(r) = \sum_{i \in P} p_i^r$. We sometimes refer to $W(r)$ as the *broadcast weight* for round r.
- Let $S(r)$ be the set of processes that begin round r active.
- Let $S(r,i) \subseteq S(r)$, $i \in S(r)$, be the set of active processes that start r as either: (a) a leader; or (b) a contender with a timestamp (i.e., the pair consisting of the process's local round counter and id) larger than the timestamp of process i.
- Let $W(r,i) = \sum_{j \in S(r,i)} p_j^r$.

We continue with a helper lemma that proves, given some $r-1$ round execution, that if $W(r) = \Theta(\mathcal{F}')$, and for some process $i \in S(r)$, a constant fraction of this weight is in $W(r,i)$, then the probability that i is knocked out in a one round extension of this execution is bounded as: $\Omega\left((\mathcal{F}' - t)/(\mathcal{F}')\right)$.

Put another way, if i chooses a non-disrupted frequency, which occurs with probability $\Omega\left((\mathcal{F}' - t)/(\mathcal{F}')\right)$, it has a constant probability of being knocked out. This captures the intuition that when $W(r,i) = \Theta(\mathcal{F}')$, a constant amount of probability mass capable of knocking out i is expected on each frequency.

Lemma 15.4.2 *Fix some $P \subseteq [n]$, P-synch environment $\mathcal{E}$, process $i \in P$, positive constants $c, c' \geq 1$, and $r-1$ round execution, α, of $(\mathcal{E}, Trapdoor_t, \mathcal{C})$, such that process i starts round r as a contender, $c\mathcal{F}' \leq W(r) \leq 3c\mathcal{F}'$, and $W(r,i) \geq W(r)/c'$. The probability that process i is knocked out in a one-round extension of α is at least:*

$$\frac{\mathcal{F}' - t}{2\mathcal{F}'} \left(\frac{c}{c'}\right) \left(\frac{1}{4}\right)^{3c}.$$

Proof. We first extend α by the probabilistic state transformations that occur at the beginning of round r, generating: R_r^E, R_r^S, and R_r^C. Let f_i be the frequency chosen by i in round r (as encoded in R_r^S). Let B_r be the frequencies disrupted by $\mathcal{C}$ in r as returned by $fblocked_{\mathcal{C}}(R_r^C)$. Recall that $fblocked_{\mathcal{C}}$ is the mapping, describing the disrupted frequencies in the round, required to exist for $\mathcal{C}$ by the definition of the t-disrupted channel property (Definition 6.5.4). Because f_i is generated independent of the other probabilistic transformations made during r, it follows that $f \notin B_r$ with probability $(\mathcal{F}' - |B_r|)/\mathcal{F}' \geq (\mathcal{F}' - t)/\mathcal{F}'$.

Now consider process i's decision to receive or broadcast., By the definition of the algorithm, process i decides to *receive* with probability $1 - p_i^r \geq 1/2$. As with its frequency choice, this decision is generated independent of the other probabilistic transformations in r. We can combine this with the above probability to conclude that process i *receives* on a *non-disrupted* frequency, with probability at least: $\frac{\mathcal{F}'-t}{2\mathcal{F}'}$.

We next bound the probability that exactly one process broadcasts on f_i, and this process is from $S(r,i)$—thus knocking out i, if f_i is undisrupted. Keeping in mind, as mentioned above, that each active process chooses it frequency and makes it broadcast

decision independent from the execution history and other probabilistic decisions in the round, we can bound this probability of a single broadcaster as follows:

$$\sum_{j \in S(r,i)} \left(\frac{p_j^r}{\mathcal{F}'} \prod_{k \in P, k \neq i,j} \left(1 - \frac{p_k^r}{\mathcal{F}'}\right)\right) .$$

The first term within the sum captures the probability that a process j from $S(r,i)$ chooses and broadcasts on f_i. The second term (i.e., the $\prod$ term) captures the probability that none of the *other* processes in P do the same (which would cause a collision).

Next, we simplify:

$$\begin{aligned}
\sum_{j \in S(r,i)} \left(\frac{p_j^r}{\mathcal{F}'} \prod_{k \in P, k \neq i,j} \left(1 - \frac{p_k^r}{\mathcal{F}'}\right)\right) &\geq \sum_{j \in S(r,i)} \left(\frac{p_j^r}{\mathcal{F}'} \prod_{k \in P, k \neq i,j} (\frac{1}{4})^{\frac{p_k^r}{\mathcal{F}'}}\right) && (15.1)\\
&= \sum_{j \in S(r,i)} \left(\frac{p_j^r}{\mathcal{F}'} (\frac{1}{4})^{\sum_{k \in P, k \neq i,j} \frac{p_k^r}{\mathcal{F}'}}\right) && (15.2)\\
&\geq \sum_{j \in S(r,i)} \left(\frac{p_j^r}{\mathcal{F}'} (\frac{1}{4})^{3c}\right) && (15.3)\\
&\geq (\frac{c}{c'})(\frac{1}{4})^{3c} && (15.4)
\end{aligned}$$

The substitution in 15.1 follows from the probability facts presented in Chapter 2. Steps 15.3 and 15.4 substitute bounds derived from the contstaints on $W(r)$ and $W(r,i)$, provided in the lemma statement.

The needed result follows from the combination of our term bounding the probability of i listening on a non-disrupted frequency in r, and our term bounding the probability that a process from $S(i,r)$ broadcasts alone on this same frequency. □

We now apply Lemma 15.4.2 to derive a bound on the probability that the total weight *never* gets too high. The key idea in the following is that once the broadcast weight reaches $\Theta(\mathcal{F}')$, the probability of being knocked out becomes sufficiently large to bring the broadcast sum back down due by knock outs (recall, when a process is knocked out, its contribution to the broadcast sum reduces to 0, reducing the overall broadcast weight). (In this sense, the probability mass in our system behaves like a self-regulating feedback circuit: when it grows too large, it reduces itself.)

Lemma 15.4.3 *Fix some non-empty $P \subseteq [n]$ and P-synch environment $\mathcal{E}$. The probability that $(\mathcal{E}, Trapdoor_t, \mathcal{C})$ generates an execution α that includes a round r such that:*

- *there is no more than one leader at the beginning of r, and*
- $W(r) \geq 6\mathcal{F}'$,

is less than or equal to $(1/n^2)$.

Proof. Our strategy is to construct an execution of $(\mathcal{E}, Trapdoor_t, \mathcal{C})$, round by round, until at least one of the following two properties holds:

1. Two or more processes become leader.

2. Every process in P is either a leader or knocked out.

Assume we build a finite execution α, such that at least one of these properties holds for the final round of α. Also assume that in every round r of α, $W(r) < 6\mathcal{F}'$. It follows that all extensions of α will satisfy the constraints of the lemma statement (leaders remain leaders and knocked out processes remain knocked out). To prove our lemma, therefore, it is sufficient to prove that the probability of the system generating a finite execution like α is at least $1 - (1/n^2)$. With our overall strategy defined, we can now proceed with the details.

We begin by constructing an execution of $(\mathcal{E}, Trapdoor_t, \mathcal{C})$, round by round. We stop our construction at the first round r such that either one of the above two conditions holds, or the following new condition holds:

3. r is the first round such that $W(r-1) \leq 2\mathcal{F}'$ and $W(r) > 2\mathcal{F}'$

(Condition 2 will eventually hold in every execution of the system. But it is possible that one of the other two conditions will become true first.)

Call this execution we are generating, α. Assume one of the above conditions is first satisfied for round r. (Therefore, α is an $r-1$ round finite execution.) By the definition our conditions and r, we know that for every $r' \leq r-1$: $W(r') < 6\mathcal{F}'$ and there is at most one leader.

Let us now consider what condition round r of α satisfies. If it is condition 1 or 2, we are done. As we argued, all extension of α will satisfy our lemma statement. We assume, therefore, that it satisfies condition 3. That is, r is the first round such that $W(r-1) \leq 2\mathcal{F}'$ and $W(r) > 2\mathcal{F}'$.

So far we have worked with an arbitrary extension. Going forward, we consider the probabilities of the different ℓ_E-round extensions of α, where ℓ_E is the epoch length (Figure 15-1). Let us start with a simple observation about *every* ℓ_E-round extension of α: Between $r-1$ and $r+\ell_E$ of any such extension, the broadcast weight of processes in $S(r-1)$ can at most double (this doubling of broadcast probability occurs, for each contender, only once every ℓ_E rounds), and at most n new processes can be made active, each adding an initial weight of $1/n$ to the total broadcast weight. It follows:

$$W(r+\ell_E) \leq 2W(r-1) + 1 < 6\mathcal{F}'. \tag{15.5}$$

We will use this fact in several places in the remainder of the proof.

Our next step is to choose a process i that is a contender at the beginning of round r of α, such that $W(r,i) \geq (7/8)W(r)$. That is, at least seventh-eighths of the

broadcast weight in this round can knock out i. (Because we assume there is at most one leader, the total weight is at least $2\mathcal{F}' \geq 4$, and no process contributes more than $1/2$ to this sum, we know there exists a process that satisfies these constraints.)

Consider the possible extensions of α. Assume we get to a round r' such that: (a) we did not reduce the weight below $2\mathcal{F}'$ between r and r'; (b) there is no more than one leader at the beginning of r'; and (c) $W(r') \geq (3/4)W(r)$. We argue that we can apply Lemma 15.4.2 with respect to this extension, round r', process i, and constants: $c = 2$, $c' = 12$.

This argument requires that we justify our choice of constants. Recall, by the lemma statement, c and c' must satisfy the following:

$$c\mathcal{F}' \leq W(r') \leq 3c\mathcal{F}' \text{ and } W(r', i) \geq W(r')/c'$$

By the same reasoning used to derive equation 15.5 from above, we know:

$$W(r') \leq 2W(r) + 1 \leq 3W(r), \tag{15.6}$$

which bounds $W(r')$ from above. To bound $W(r', i)$, we note that in round r, no more than $(1/8)$ of the broadcast weight in this round was contributed by processes not in $S(r, i)$ (i.e., processes that cannot knock out i.) Since round r, these processes could have at most doubled their weight. In addition, less than n new processes could have been activated, each contributing less than 1 to the broadcast weight. It follows that the broadcast weight in r' that *cannot* knock out i is less than $(2/8)W(r) + 1 \leq (1/2)W(r)$. Because we assumed that $W(r') \geq (3/4)W(r)$, we derive the following bound on $W(r', i)$:

$$W(r', i) \geq (1/4)W(r) \tag{15.7}$$

We know our choice of c is correct. To see that c' is sufficiently large, we note that we maximize c' in the equation $W(r', i) \geq W(r')/c'$, when we minimize $W(r', i)$ and maximize $W(r')$. Using the minimum and maximum values given by bounds 15.7 and 15.6, respectively, we get $c' \leq 12$, as needed. We are safe, therefore, to apply Lemma 15.4.2 with these values, and conclude that i is knocked out with probability at least:

$$p_{ko} = \frac{\mathcal{F}' - t}{2\mathcal{F}'} \left(\frac{1}{6}\right) \left(\frac{1}{4}\right)^6. \tag{15.8}$$

With these observations, we can now construct an ℓ' round extension of α (where ℓ', as defined in Figure 15-1, equals $\frac{1}{p_{ko}} 6 \ln 2n < \ell_E$), round by round. At the start of each round r' in this process, either something *good* happens—we elect two or more leaders or the broadcast weight will be low $W(r') \leq (3/4)W(r)$—or there exists some contender i that gets knocked out with probability p_{ko}. The probability, therefore, that our two good conditions do not hold *and* i does not get knocked out for an entire ℓ'-round extension of α, is no greater than:

$$
\begin{aligned}
(1-p_{ko})^{\ell'} &< e^{-p_{ko}\ell'} && (15.9)\\
&= e^{-6\ln 2n} && (15.10)\\
&= 1/(64n^6) && (15.11)
\end{aligned}
$$

We can extend this argument to consider *every* process i, such that $W(r,i) \geq (7/8)W(r)$ in α. To dispense with dependency issues, we apply a union bound, which tells us that the probability that the two good conditions do not hold and at least one process i such that $W(r,i) \geq (7/8)W(r)$ does not get knocked out, in our extension, is no greater than $1/(64n^5)$.

Flipping this around, we can say that with probability *at least* $1 - 1/(64n^5)$, an ℓ'-round extension of α elects two leaders, or arrives at a round r' such that at least $(1/8)$ of the broadcast weight from r (i.e., $(1/8)W(r)$) has been knocked out. This final point combines the outcome of the two distinct cases: (a) every process i with $W(r,i) \geq (7/8)W(r)$ getting knocked out; and (b) the failure of the good condition that the weight was greater than $(3/4)W(r)$.

This is the crucial observation that we will now use in a series of union bounds to generate out final result. Notice that we defined $\ell_E \geq 17\ell'$. Applying a union bound to our above probability, and an extension of ℓ_E rounds of α, we conclude that with probability at least $1-17/(64n^5) > 1-1/n^5$, during this extension either we elect two or more leaders or have $(17/8)W(r)$ weight knocked out. Notice, $(17/8)W(r) > 4\mathcal{F}'$, and $4\mathcal{F}'$ was the maximum amount of broadcast weight added in this interval (by 15.5). It follows that if this event occurs, there exists a round in the interval where the broadcast weight falls below $2\mathcal{F}'$.

If this occurs in an extension, we can jump to this round, and start over from here, extending until we next arrive at a round that, like α, satisfies the one of the conditions, 1—3, from earlier in the proof. At this point, we then repeat our argument, which says that with probability at least $1 - 1/n^5$, an extension of this new execution will either elect two leaders or will arrive at a new round such that the weight falls back below $2\mathcal{F}'$. And so on.

We are left to ask how many such applications of this $1-1/n^5$ argument might we have to make in a given extension until we are guaranteed to have satisfied at either conditions 1 or 2 from above. In the worst case, each of the $|P| \leq n$ processes that are eventually activated has a full $T = (\lg(n) - 1)\ell_E + \ell_E^+$ rounds of being a contender. So there are at most $nT < n^3$ applications of this argument until we are guaranteed that our desired conditions hold. By a final union bound, our argument fails to hold at least once in an extension with probability no worse than $n^3/n^5 = 1/n^2$, as needed. □

We conclude with the proof of the main theorem. The key argument in this proof concerns agreement. Our strategy is to bound the probability of their ever being more than one leader. This argument relies on Lemma 15.4.3 to prove that the total broadcast weight in the system remains sufficiently low. Keeping the total weight low helps us increase the probability that the first leader knocks out other processes. By

contrast, if the broadcast weight in the system got too high, the leader might fail to knock out other processes because its messages are lost to collisions.

Proof (of Theorem 15.4.1). To simplify the presentation of this proof, we prove the properties of the problem as described at the beginning of this chapter. The reader will have to trust that the formal mathematical version of the problem (Definition 15.1.3) is equivalent to this informal description.

Fix some non-empty $P \subseteq [n]$ and P-synch environment $\mathcal{E}$. We first note that in any execution of $(\mathcal{E}, Trapdoor_t, \mathcal{C})$, properties 1–3 of the problem follow directly from the definition of the protocol.

To establish property 4, *agreement*, we first note that a process i does not synchronize to a global round number (i.e., set rnd_i to a value in $\mathbb{N}$) output a round number without first receiving a message from a leader. It is sufficient, therefore, to show that with probability p, at most one process becomes leader in an execution of $(\mathcal{E}, Trapdoor_r, \mathcal{C})$.

First, we restrict ourselves to the executions of the system that satisfy the constraints of Lemma 15.4.3. Recall, this lemma tells us that an execution satisfies these constraints with probability at least $1 - (1/n^2)$.

Second, we construct such an execution of the system, round by round, until the first round in which a process receives a *wake* input. Call this prefix α. Let i be the process with the largest id of the processes that received this input in this round. By definition of the algorithm, i will always have the largest timestamp in the system—therefore, no contender can knock out i. It follows that in *any* extension of α, i will become leader after completing its final epoch.

Consider the other active processes in decreasing order of their timestamps. Let j be the process among these with the next highest timestamp behind i. Let α' be an extension of α up to the first round of j's final epoch. Call this round r. There is at most 1 leader in the system at this round (i.e., i). By our assumption that this execution satisfies Lemma 15.4.3, we also know that $W(r) \leq 6\mathcal{F}'$.

Process i sends and process j receives in this round with probability (1/4). (Process i is either leader or in its final epoch during r. Either way, its broadcast probability is 1/2.) By our standard claims on independence of probabilistic choices (as argued in Lemma 15.4.2), we note that that i broadcasts to j on a non-disrupted frequency f in this frequency with probability at least $(\mathcal{F}' - t)/(4(\mathcal{F}')^2)$.

We next bound the probability that no *other* contender broadcasts on f in r (which would cause a collision, and thus prevent a knock out), to be at least:

$$p_{solo} = \prod_{k \in P, k \neq i,j} \left(1 - \frac{p_k^r}{\mathcal{F}'}\right) \tag{15.12}$$

$$\geq (1/4)^6 \tag{15.13}$$

(As in Lemma 15.4.2, we used the probability facts from Chapter 2, and our assumption that $W(r) \leq 6\mathcal{F}'$. This is the crucial calculation where it is important that $W(r)$ is not too large.)

It follows that with probability at least

$$p_{fko} = \frac{p_{solo}(\mathcal{F}' - t)}{(4(\mathcal{F}')^2)},$$

process i knocks out process j in r.

We can extend this argument to all ℓ_E^+ rounds of j's final epoch. Process i will fail to knockout j in each such round, with probability no greater than $(1 - p_{fko})^{\ell_E^+}$. We note that $\ell_E^+ \geq \frac{3}{p_{fko}} \ln n$. This allows us to simplify as follows (once again, in the first step, deploying our useful probability facts):

$$\begin{aligned}(1 - p_{fko})^{\ell_E^+} &< e^{-p_{fko}\ell_E^+} \\ &= e^{-3 \ln n} \\ &= 1/n^3\end{aligned}$$

We apply a union bound over *all* potential non-j leaders, in decreasing order of their timestamps, to derive that the probability that at least one of them fails to get knocked out by i, is no greater than $1/n^2$. We then deploy yet another union bound to prove that either this fails to happen, or the conditions of Lemma 15.4.3 fails to hold, with probability no greater than $2/n^2$.

We next consider property 5, *liveness*. We apply the same style of argument used to calculate the agreement probability. This time, however, we consider the probability that each active process j receives a message from process i in the first ℓ_E^+ rounds *after* it completes its final epoch. During these rounds, i will definitely be a leader, as by assumption it was the first process to activate. Therefore, if a process j receives a message from i during these rounds, it will synchronize, if it has not already.

Let $T = (\lg n - 1)\ell_E + 2\ell_E^+$ (i.e., the running time of the algorithm plus an extra ℓ_E^+ rounds). We apply a final union bound to determine that the probability that agreement fails, *or* liveness fails to occur by round T, is no greater than $(2/n^2 + 2/n^2) = 4/n^2 \leq 1/n$. (The final reduction follows from our assumption that $n > 3$, made earlier in the chapter.)

We are almost done. Our last task is to show that

$$T = O\left(\frac{\mathcal{F}}{\mathcal{F} - t} \log^2 n + \frac{\mathcal{F}t}{\mathcal{F} - t} \log n\right),$$

the running time from the theorem statement. Fortunately, this follows easily from the definitions of ℓ_E and ℓ_E^+, and the observation that: $\frac{\mathcal{F}'}{\mathcal{F}'-t} = \Theta(\frac{\mathcal{F}}{\mathcal{F}-t})$ and $\frac{(\mathcal{F}')^2}{\mathcal{F}'-t} = \Theta(\frac{\mathcal{F}t}{\mathcal{F}-t})$. □

A Note on Constants. Before continuing, we note that the constant factors in our definitions of ℓ_E and ℓ_E^+ are large—perhaps too large for practical application. A more sophisticated probabilistic analysis, however, might significantly reduce these constants. We tolerate these larger constants to help keep the proof intuition clear.

15.5 Using Synchronization to Overcome the Difficulties of Ad Hoc Networks

A solution to the wireless synchronization problem can help mitigate the difficulties introduced by the ad hoc radio network model, namely: the activation of processes in different rounds and lack of advance knowledge of the active processes. Below, we describe a general strategy for using a wireless synchronization solution to adapt algorithms constructed for the original model of Chapter 3 (e.g., the channel emulation algorithms of Part II) to the ad hoc variant presented in Chapter 14

Overview of Strategy. At a high level, our strategy works as follows. If processes agree on a global round number then they can divide the rounds into repeating *epochs* of length x, where x is some large constant to be defined later. We specify that a new epoch starts at every round r such that $r \ mod \ x = 0$.

We then divide each epoch into two *sub-epochs*, in the same manner. The first is of length x_1 and the second of length x_2, where $x_1 + x_2 = x$. The first sub-epoch is used by the processes to run a *set membership algorithm*—an algorithm that generates agreement on the current set of *participating* processes. The second sub-epoch is then used by the participating processes to run algorithms designed for our original model. They treat the first round of this sub-epoch as round 1 for the original model algorithm. They can also use the knowledge of the set of participating processes, S, to assign the participants ids 1 through $|S|$, as expected by the original model. The algorithm run here must either terminate within the x_2 rounds alloted to it in the current epoch, or, perhaps, be able to be *paused* between epochs.[1] Notice, the channel emulation algorithms of Part II, and the problem-solving algorithms of Part III, would probably be of the former type, as they have bounded finite runtimes.

To realize this strategy, we need a set membership algorithm and a way to integrate the round number synchronization into the epoch structure. We address both issues below.

The Set Membership Algorithm. A simple set agreement algorithm has each newly activated process (i.e., each process that is not yet in the set of participants) select a frequency at random. With probability $1/n$, it broadcasts. Otherwise, it receives. For a sufficiently large sub-epoch length, x_1, it can be guaranteed that this process will be heard by all other processes with high probability, and therefore it will be added to all participant sets.

Agreeing on a Global Round Number. The other key component to our strategy is how newly activated processes learn the global round number. We describe three different approaches to accomplishing this goal.

[1]This second property—the ability to be paused—would also require that the algorithm have a graceful way of handling the arrival of new participants that might be added to the membership set between epochs. We leave the formalization of these algorithm properties as interesting future work.

The first approach is for newly activated processes to run a wireless synchronization solution immediately following their activation. Once they become *synchronized* to the global round counter, they can execute the appropriate sub-epoch algorithms. If the solution, like $Trapdoor_t$, is leader-based, the leader will have to continue to participate in the synchronization algorithm even after it synchronizes.

Their are two disadvantages to this approach. The first is that the broadcasts from the wireless synchronization solution can disrupt the other communication in the epoch. The second is the possibility that some processes, such as the leader in $Trapdoor_t$, has to continually broadcast, which might disrupt its participation in the algorithm being run in the second sub-epoch.

With this in mind, a better approach is to divide the frequencies into two groups. The first group could be used by processes to run the wireless synchronization protocol after being activated. The second group can be used for the epoch algorithms. This eliminates the issue of the synchronization solution disrupting the epoch algorithms. It does require, however, that $\mathcal{F}$ is sufficiently large for this division. That is, $\mathcal{F} \geq 2t + 2$. Furthermore, for leader-based synchronization solutions, we still have the problem of a leader not being able to fully participate in the epoch algorithms.

The third approach is to add a third sub-epoch. During this new sub-epoch, the process with the smallest id in the membership set announces the global round number on randomly selected frequencies. When a process is first activated, it enters a *listening phase* during which it receives on random frequencies for x rounds. This phase will overlap an entire third sub-epoch. If there are participating processes already in the system, and the third sub-epoch is of sufficient length, this newly activated process will learn the global round number during one of these rounds—preventing it from needing to execute a disruptive wireless synchronization solution.

If a newly activated process does not receive a round number, only then does it execute the synchronization solution to safely agree on a round. If the solution is leader-based, the leader can eventually abandon its duty, trusting the third sub-epoch to synchronize any further new arrivals. This option minimizes the potential of disruption from the synchronization solution at the cost of a longer epoch.

Chapter 16

Conclusion

Within the broad context of wireless networking, an increasingly important niche is the study of reliable algorithms for settings suffering from what we call *adversarial interference*. This term captures any type of channel disruption *outside the control* of the algorithm designer, including: contention with unrelated devices using overlapping sections of the radio spectrum, electromagnetic noise from non-networked devices—lighting, radar, etc.—and malicious jamming that takes advantage of the open nature of the radio medium. Though the systems community has made important progress in mitigating some of the effects of this interference (e.g., through the development of more resilient signal encoding), the problem is far from solved. That is, the theory community cannot escape their obligation to grapple with this pernicious behavior in the design and proof of reliable radio network algorithms. Furthermore, it can be argued that with the appropriate tools for addressing these problems, the theory community can aid their systems brethren in this battle against unpredictable disruption.

16.1 Contributions

This thesis aids the theory community in achieving the above-stated goals. Specifically, it contains the following four important contributions.

16.1.1 Radio Network Modeling Framework

We described a formal modeling framework for the study of distributed algorithms in radio networks. This framework allows for a precise, probabilistic, automaton-based description of radio channels and algorithm behavior. It is general enough to capture almost any radio model studied to date in the theory literature. At the same time, it is particularly well-suited to specifying the often subtle details of adversarial behavior.

This framework also includes formal notions of *problems*, *solving problems*, and *implementing one channel using another channel*. These formalisms are used by a pair of composition results that enable a *layered* approach to algorithm design. The first result can be used to combine an algorithm that solves a problem P using a powerful

channel $\mathcal{C}_1$, with an algorithm that implements $\mathcal{C}_1$ using a less powerful channel $\mathcal{C}_2$, to produce an algorithm that solves P using $\mathcal{C}_2$. The second result combines a channel implementation algorithm with a channel to produce a new channel. Later in the thesis, this result facilitates the design of channel implementation algorithms.

16.1.2 Channel Definitions and Implementations

Using our modeling framework, we formalized the intuitive notion of adversarial interference with the precise definition of the *t-disrupted* radio channel property. A t-disrupted channel provides the devices access to a collection of $\mathcal{F} > 0$ separate communication frequencies, up to $t < \mathcal{F}$ of which can be disrupted in each round by an abstract *interference adversary*. This adversary does not necessarily model a literal adversarial device. Instead, it incarnates the diversity of possible interference sources experienced in an open radio network environment.

We then introduced the more powerful (t, b, p)-*feedback* channel property. Such a channel behaves like a t-disrupted channel enhanced to provide feedback to the devices about what was received on *all* frequencies during the current round. Specifically, it guarantees that with probability p, at least $n - b$ devices receive the feedback. We argued that obtaining such feedback is essential to solving problems in a setting with adversarial interference. This channel, therefore, provides just enough information to allow algorithm designers to turn their focus onto the specifics of the problem they are solving.

We then described two implementations of a (t, b, p)-feedback channel using a t-disrupted channel: the *RFC* and *DFC* algorithms. The former is randomized and the latter is deterministic.

16.1.3 Solving Problems Using a Feedback Channel

To demonstrate the power of a layered approach to algorithm design, we described intuitive solutions to three common problems using a feedback channel. Specifically, we solved *set agreement*, *gossip*, and *reliable broadcast*. When these algorithms are composed with our feedback channel implementation algorithms, we automatically generate complementary solutions for the t-disrupted channel. We argue that devising solutions from scratch on the t-disrupted channel would have introduced a significant amount of extra complexity and repeated effort—validating the practical utility of our layered strategy.

16.1.4 The Ad Hoc Variant

Our final contribution was the introduction of a variant of our modeling framework that better captures the properties of an *ad hoc* radio network. Specifically, we modified our framework so that only an arbitrary subset of the processes are activated (by the environment), and these activations might occur during different rounds.

With this modification defined, we then provided a solution to the *wireless synchronization problem* using a t-disrupted channel in the ad hoc setting. This problem

requires an unknown number of processes to agree on a global round numbering. We discussed how such a solution can be used to adapt algorithms designed for a non-ad hoc network (e.g., our feedback channel implementation algorithms) to work in this ad hoc setting.

16.2 Future Work

Though this thesis solidifies a strong theoretical treatment of adversarial interference in radio networks, much interesting future work on the topic remains. With respect to the modeling framework, for example, it would be useful to devise a more general algorithm composition result that allows the combination of arbitrary algorithms (not just *channel* algorithms with *problem-solving* algorithms). Consider, for example, the ability to combine an algorithm that implements a replicated state machine with an algorithm that solves mutual exclusion using a replicated state machine. Such general composition is useful for the development and verification of more complex algorithms.

In addition, though the bulk of the existing theory work on radio networks assumes synchronous time slots, producing an asynchronous (or partially asynchronous) version of our modeling framework remains an interesting project. Presumably, one might use an existing general modeling language, such as TIOA [51], for the this endeavor, thus avoiding the need to tackle, from scratch, the complex modeling issues created by non-synchronous environments (e.g., reconciling *probabilistic* behavior with the *non-deterministic* scheduling of asynchronous events).

With respect to the channel definitions, there remains work to be done in the investigation of other useful channel properties. The t-disrupted channel property, for example, is simple and captures a wide variety of settings, but it does not allow for spoofed message. It can also be argued that it is *too* powerful: a more tractable property might make the decision of which frequencies to disrupt oblivious; i.e., independent of the behavior in the current execution.[1] Numerous similar variations are worth exploring, from providing more power to the algorithm (e.g., collision/disruption detection), to capturing more detailed disruption effects (e.g., probabilistic flipping of individual bits in a message).

Similarly, the (t, b, p)-feedback channel property is just one suggestion from among many powerful channel types that could potentially simplify the development of algorithms. For example, maybe a single-frequency reliable broadcast channel is also an appropriate high-level channel to implement. It is also clear that for both cases—low-level disrupted channels and high-level well-behaved channels—multihop definitions are needed.

With respect to the channel implementation algorithms, it remains an open question whether more efficient solutions are possible. Another interesting open question is whether the restriction on the number of broadcasters is necessary for a determin-

[1]This latter property could easily be captured by modifying Definition 6.5.4 such that $fblocked_C$, applied to the channel start state, returns an infinite sequence of blocked sets, $B_1 \in F_t, B_2 \in F_t, ...$, that describe what frequencies will be blocked in *every* round of the execution.

istic implementation of a feedback channel. (Recall, the feedback channel property described in Definition 6.5.5 requires that $bcount(M) \leq \mathcal{F}$, where M is the message assignment passed to the channel, and $bcount(M)$ returns the number of broadcasters in M.)

With respect to the algorithms that use a feedback channel—the set agreement, gossip, and reliable broadcast solutions—these were meant as case studies that demonstrate the components of the framework in action. They are not, by any means, the *only* useful problems that can be solved in this setting. Of the numerous possible problems worth addressing in a radio network environment, some of the most obvious include: multihop broadcast and point-to-point routing, maintaining replicated state, leader election, and using leaders to construct network-wide structures such as connected backbones.

Finally, with respect to the ad hoc variant of the model, it remains pressing future work to formalize the proposed use of a wireless synchronization solution to adapt algorithms from the original model to work in the ad hoc variant. Specifically, a precise specification of the repeating epoch strategy is required, as is a theorem statement that captures its exact guarantees.

Bibliography

[1] IEEE 802.11. Wireless LAN MAC and physical layer specifications, June 1999.

[2] N. Abramson. The Aloha system - Another approach for computer communications. *The Proceedings of the Fall Joint Computer Conference*, 37:281–285, 1970.

[3] David Adamy. *A First Course in Electronic Warfare.* Artech House, 2001.

[4] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran, and Shantidev Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey. *Computer Networks*, 50(13):2127–2159, 2006.

[5] Z.B. Alliance. Zigbee specification. *ZigBee Document 053474r06*, 1, 2005.

[6] Noga Alon, Amotz Bar-Noy, Nathan Linial, and David Peleg. A lower bound for radio broadcast. *Journal of Computer and System Sciences*, 43(2):290–298, October 1992.

[7] B. Awerbuch, A. Richa, and C. Scheideler. A jamming-resistant mac protocol for single-hop wireless networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2008.

[8] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. *Journal of Computer and System Sciences*, 45(1):104–126, 1992.

[9] R. Bar-Yehuda, O. Goldreich, and A. Itai. Errata regarding "on the time complexity of broadcast in radio networks: An exponential gap between determinism and randomization". http:www.wisdom.weizmann.ac.ilmathusersodedp_bgi.html, 2002.

[10] Vartika Bhandari and Nitin H. Vaidya. On reliable broadcast in a radio network. In *Proceedings of the International Symposium on Principles of Distributed Computing*, pages 138–147, 2005.

[11] Bluetooth Consortium. *Bluetooth Specification Version 2.1*, July 2007.

[12] Annalisa De Bonis, Leszek Gasieniec, and Ugo Vaccaro. Optimal two-stage algorithms for group testing problems. *SIAM Journal on Computing*, 34(5):1253–1270, 2005.

[13] T. X. Brown, J. E. James, and Amita Sethi. Jamming and sensing of encrypted wireless ad hoc networks. Technical Report CU-CS-1005-06, UC Boulder, 2006.

[14] J. I. Capetanakis. "the multiple access broadcast channel: Protocol and capacity considerations. *IEEE Transactions on Information Theory*, IT-25:505–515, Sept. 1979.

[15] J.M. Chang and N.F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer systems*, 2(3):251–273, 1984.

[16] S. Chaudhuri. Agreement is harder than consensus: Set consensus problems in totally asynchronous systems. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 1990.

[17] I. Chlamtac and O. Weinstein. The wave expansion approach to broakodcasting in multihop radio networks. *IEEE Transactions on Communications*, 39:426–433, 1991.

[18] B. Chlebus and D. Kowalski. A better wake-up in radio networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, pages 266–274, 2004.

[19] Bogdan S. Chlebus, Dariusz R. Kowalski, and Andrzej Lingas. The do-all problem in broadcast networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, pages 117–127, 2001.

[20] G. Chockler, M. Demirbas, S. Gilbert, N. Lynch, C. Newport, and T. Nolte. Reconciling the theory and practice of unreliable wireless broadcast. In *International Workshop on Assurance in Distributed Systems and Networks (ADSN)*, June 2005.

[21] Gregory Chockler, Murat Demirbas, Seth Gilbert, Nancy Lynch, Calvin Newport, and Tina Nolte. Consensus and collision detectors in radio networks. *Distributed Computing*, 21:55–84, 2008.

[22] Gregory Chockler, Murat Demirbas, Seth Gilbert, and Calvin Newport. A middleware framework for robust applications in wireless ad hoc networks. In *Proceedings of the 43rd Allerton Conference on Communication, Control, and Computing*, 2005.

[23] Gregory Chockler, Murat Demirbas, Seth Gilbert, Calvin Newport, and Tina Nolte. Consensus and collision detectors in wireless ad hoc networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, pages 197–206, New York, NY, USA, 2005. ACM Press.

[24] M. Chrobak, L. Gasieniec, and D. Kowalski. The wake-up problem in multi-hop radio networks. In *Symposium on Discrete Algorithms (SODA)*, 2004.

[25] M. Chrobak, L. Gasieniec, and W. Rytter. broadcasting and gossiping in radio networks. *Journal of ALgorithms*, 43:177–189, 2002.

[26] A. Clementi, A. Monti, and R. Silvestri. Optimal f-reliable protocols for the do-all problem on single-hop wireless networks. In *Algorithms and Computation*, pages 320–331, 2002.

[27] A. Clementi, A. Monti, and R. Silvestri. Round robin is optimal for fault-tolerant broadcasting on wireless networks. *Journal of Parallel and Distributed Computing*, 64(1):89–96, 2004.

[28] Andrea E. F. Clementi, Angelo Monti, and Riccardo Silvestri. Selective families, superimposed codes, and broadcasting on unknown radio networks. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 709–718, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[29] Artur Czumaj and Wojciech Rytter. Broadcasting algorithms in radio networks with unknown topology. In *Proc. of FOCS*, October 2003.

[30] Shlomi Dolev, Seth Gilbert, Rachid Guerraoui, Fabian Kuhn, and Calvin Newport. The wireless synchronization problem. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2009.

[31] Shlomi Dolev, Seth Gilbert, Rachid Guerraoui, and Calvin Newport. Gossiping in a multi-channel radio network: An oblivious approach to coping with malicious interference. In *Proceedings of the International Symposium on Distributed Computing*, 2007.

[32] Shlomi Dolev, Seth Gilbert, Rachid Guerraoui, and Calvin Newport. Secure communication over radio channels. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2008.

[33] Vadim Drabkin, Roy Friedman, and Marc Segal. Efficient byzantine broadcast in wireless ad hoc networks. In *Dependable Systems and Networks*, pages 160–169, 2005.

[34] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[35] I. Gaber and Y. Mansour. Broadcast in radio networks. In *In proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[36] R. Gallager. A perspective on multiaccess channels. *IEEE Trans. Information Theory*, IT-31:124–142, 1985.

[37] L. Gasieniec, A. Pelc, and D. Peleg. wakeup problem in synchronous broadcast systems. *SIAM Journal on Discrete Mathematics*, 14:207–222, 2001.

[38] L. Gasieniec, T. Radzik, and Q. Xin. Faster deterministic gossiping in directed ad-hoc radio networks. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory*, 2004.

[39] S. Gilbert, R. Guerraoui, and C. Newport. Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. *Theoretical Computer Science*, 410(6-7):546–569, 2009.

[40] Seth Gilbert, Rachid Guerraoui, Darek Kowalski, and Calvin Newport. Interference-resilient information exchange. In *Proceedings of the Conference on Computer Communication*, 2009.

[41] Seth Gilbert, Rachid Guerraoui, Dariusz Kowalski, and Calvin Newport. Interference-resilient information exchange. In *To Be Published.*, 2009.

[42] Seth Gilbert, Rachid Guerraoui, and Calvin Newport. Of malicious motes and suspicious sensors: On the efficiency of malicious interference in wireless networks. In *Proceedings of the International Conference on Principles of Distributed Systems*, December 2006.

[43] R. Gummadi, D. Wetherall, B. Greenstein, and S. Seshan. Understanding and mitigating the impact of rf interference on 802.11 networks. In *SIGCOMM*, pages 385–396. ACM New York, NY, USA, 2007.

[44] V. Gupta, S. Krishnamurthy, and S. Faloutsos. Denial of service attacks at the mac layer in wireless ad hoc networks. In *Military Communications Conference*, 2002.

[45] B. Hajek and T. van Loon. Decentralized dynamic control of a multiaccess broadcast channel. *IEEE Transactions on Automation and Control*, AC-27:559–569, 1979.

[46] J. F. Hayes. An adaptive technique for local distribution. *IEEE Transactions on Communication*, COM-26(8):1178–1186, August 1978.

[47] S.M Hedetniemi, S. T. Hedetniemi, and A.L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4), 1988.

[48] Y.C. Hu and A. Perrig. A survey of secure wireless ad hoc routing. *IEEE Security and Privacy Magazine*, 02(3):28–39, 2004.

[49] Piotr Indyk. Explicit constructions of selectors and related combinatorial structures, with applications. In *Proc. of SODA*, 2002.

[50] M. Kaplan. A sufficient condition for non-ergodicity of a markov chain. *IEEE Transactions on Information Theory*, IT-25:470–471, July, 1979.

[51] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata.* Synthesis Lectures on Computer Science. Morgan Claypool Publishers, 2006. Also MIT-LCS-TR-917a.

[52] L. Kleinrock and F.A. Tobagi. Packet switching in radio channels. *IEEE Transactions on Communications*, COM-23:1400–1416, 1975.

[53] J. Komlos and A.G. Greenberg. An asymptotically fast non-adaptive algorithm for conflict resolution in multiple access channels. *IEEE Trans. Inf. Theory*, March 1985.

[54] C-Y. Koo. Broadcast in radio networks tolerating byzantine adversarial behavior. In *Proceedings of the International Symposium on Principles of Distributed Computing*, pages 275–282, 2004.

[55] Chiu-Yuen Koo, Vartika Bhandari, Jonathan Katz, and Nitin H. Vaidya. Reliable broadcast in radio networks: The bounded collision case. In *Proceedings of the International Symposium on Principles of Distributed Computing*, 2006.

[56] D. Kowalski and A. Pelc. Broadcasting in undirected ad hoc radio networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, pages 73–82, 2003.

[57] D. Kowalski and A. Pelc. Time of radio broadcasting: Adaptiveness vs. obliviousness and randomization vs. determinism. In *In Proceedings of the 10th Colloquium on Structural Information and Communication Complexity*, 2003.

[58] D. Kowalski and A. Pelc. Time of deterministic broadcasting in radio networks with local knowledge. *SIAM Journal on Computing*, 33(4):870–891, 2004.

[59] Dariusz R. Kowalski. On selection problem in radio networks. In *Proceedings of the International Symposium on Principles of Distributed Computing*, pages 158–166, New York, NY, USA, 2005. ACM Press.

[60] E. Kranakis, D. Krizanc, and A. Pelc. Fault-tolerant broadcasting in radio networks. In *Proceedings of the Annual European Symposium on Algorithms*, pages 283–294, 1998.

[61] Evangelos Kranakis, Danny Krizanc, and Andrzej Pelc. Fault-tolerant broadcasting in radio networks. *Journal of Algorithms*, 39(1):47–67, April 2001.

[62] S. Krishnamurthy, R. Chandrasekaran, Neeraj Mittal, and S. Venkatesan. Brief announcement: Synchronous distributed algorithms for node discovery and configuration in multi-channel cognitive radio networks. In *Proc. of DISC*, October 2006.

[63] S. Krishnamurthy, R. Chandrasekaran, S. Venkatesan, and N. Mittal. Algorithms for node discovery and configuration in cognitive radio networks. Technical report, Tecnical Report UTDCS-23-06 (UT Dallas), May 2006.

[64] S. Krishnamurthy, M. Thoppian, S. Kuppa, R Chanrasekaran, S. Venkatesan, N. Mittal, and R. Prakash. Time-efficient layer-2 auto-configuration for cognitive radios. In *Proc. of PDCS*, November 2005.

[65] E. Kushelevitz and Y. Mansour. Computation in noisy radio networks. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[66] David Leeper. A long-term view of short-term wireless. *Computer*, 34(6):39–44, 2001.

[67] M. Li, I. Koutsopoulos, and R. Poovendran. Optimal jamming attacks and network defense policies in wireless sensor networks. In *Proceedings of the Conference on Computer Communication*, 2007.

[68] Dominic Meier, Yvonne Anne Pignolet, Stefan Schmid, and Roger Wattenhofer. Speed dating despite jammers. In *The Proceedings of the International Conference on Distributed Computing in Sensor Systems*, 2009.

[69] J. Mitola. *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio.* PhD thesis, Royal Institute of Technology, Sweden, 2000.

[70] Koji Nakano and Stephan Olariu. A survey on leader election protocols for radio networks. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks*, page 71. IEEE Computer Society, 2002.

[71] R. Negi and A. Perrig. Jamming analysis of mac protocols. Technical report, Carnegie Mellon University, 2003.

[72] Calvin Newport. Consensus and collision detectors in wireless ad hoc networks. Master's thesis, MIT, 2006.

[73] Calvin Newport and Nancy Lynch. Modeling radio networks. In *The International Conference on Concurrency Theory*, 2009.

[74] Guevara Noubir, Tao Jin, and Bishal Thapa. Zero pre-shared secret key establishment in the presence of jammers. In *Proceedings International Symposium on Mobile Ad Hoc Networking and Computing*, 2009.

[75] E. Pagani and G. Rossi. Reliable broadcast in mobile multihop packet networks. In *Proceedings of the International Conference on Mobile Computing and Networking*, pages 34–42, 1997.

[76] JM Peha. Approaches to spectrum sharing. *IEEE Communications Magazine*, 43(2):10–12, 2005.

[77] Andrzej Pelc and David Peleg. Broadcasting with locally bounded byzantine faults. *Information Processing Letters*, 93(3):109–115, 2005.

[78] Richard Poisel. *Modern Communications Jamming Principles and Techniques.* Artech House, 2004.

[79] L. G. Roberts. Aloha packet system with and without slots and capture. In *ASS Note 8.* Advanced Research Projects Agency, Network Information Center, Stanford Research Institute, 1972.

[80] Mario Strasser, Christina Pöpper, and Srdjan Capkun. Efficient uncoordinated fhss anti-jamming communication. In *Proceedings International Symposium on Mobile Ad Hoc Networking and Computing*, 2009.

[81] Mario Strasser, Christina Pöpper, Srdjan Capkun, and Mario Cagalj. Jamming-resistant key establishment using uncoordinated frequency hopping. In *The IEEE Symposium on Security and Privacy*, 2008.

[82] B. S. Tsybakov and V. A. Mikhailov. Free synchronous packet access in a broadcast channel with feedback. *Prob. Inf. Transmission*, 14(4):1178–1186, April 1978. Translated from Russin original in Prob. Peredach. Inform., Oct.-Dec. 1977.

[83] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *Computer*, 35(10):54–62, 2002.

[84] W. Xu, W. Trappe, Y. Zhang, and T. Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings International Symposium on Mobile Ad Hoc Networking and Computing*, 2005.