Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2009-023 June 4, 2009

Modeling Radio Networks

Calvin Newport and Nancy Lynch

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Modeling Radio Networks*

Calvin Newport and Nancy Lynch

MIT CSAIL, Cambridge, MA
{cnewport,lynch}@csail.mit.edu

Abstract. We describe a modeling framework and collection of foundational composition results for the study of probabilistic distributed algorithms in synchronous radio networks. Existing results in this setting rely on informal descriptions of the channel behavior and therefore lack easy comparability and are prone to error caused by definition subtleties. Our framework rectifies these issues by providing: (1) a method to precisely describe a radio channel as a probabilistic automaton; (2) a mathematical notion of implementing one channel using another channel, allowing for direct comparisons of channel strengths and a natural decomposition of problems into implementing a more powerful channel and solving the problem on the powerful channel; (3) a mathematical definition of a problem and solving a problem; (4) a pair of composition results that simplify the tasks of proving properties about channel implementation algorithms and combining problems with channel implementations. Our goal is to produce a model streamlined for the needs of the radio network algorithms community.

1 Introduction

In this paper we describe a modeling framework, including a collection of foundational composition results, for the study and comparison of distributed algorithms in synchronous radio networks. In the two decades that followed the deployment of *AlohaNet* [1]—the first radio data network—theoreticians invested serious effort in the study of distributed algorithms in this setting; c.f., [2–4]. This early research focused on the stability of ALOHA-style MAC layers under varying packet arrival rates. In a seminal 1992 paper, Bar-Yehuda, Goldreich, and Itai (BGI) [5] ushered in the modern era of radio network analysis by introducing a synchronous multihop model and a more general class of problems, such as reliable broadcast. Variants of this model have been studied extensively in the intervening years; c.f., [6–8]. Numerous workshops and conferences are now dedicated exclusively to radio network algorithms–e.g., POMC, ADHOCNETS—and all major distributed algorithms conference have sessions dedicated to the topic. In short, distributed algorithms for radio networks is an important and well-established field.

* This work has been support in part by Cisco-Lehman CUNY A New MAC-Layer Paradigm for Mobile Ad-Hoc Networks, AFOSR Award Number FA9550-08-1-0159, NSF Award Number CCF-0726514, and NSF Award Number CNS-0715397.

The vast majority of existing theory concerning radio networks, however, 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 [5] claimed a $\Omega(n)$ lower bound for multihop broadcast in small diameter graphs. 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]. In our work on consensus [10], 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 so on. We also note that informal model descriptions prevent comparability 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 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 resolve problems in every model variant.

In this paper we describe a modeling framework that addresses these issues. Specifically, we use probabilistic automata to describe executions of distributed algorithms in a *synchronous* radio network.[1] (We were faced with the decision of whether to build a custom framework or use an existing formalism for modeling probabilistic distributed algorithms, such as [11–13]. We opted for the custom approach as we focus on the restricted case of synchronous executions of a fixed set of components. We do not the need the full power of general models which, among other things, must reconcile the nondeterminism of asynchrony with the probabilistic behavior of the system components.)

In our framework: The radio network is described by a channel automaton; the algorithm is described by a collection of n process automata; and the environment—which interacts with the processes through input and output ports—is described by its own automaton. In addition to the basic system model, we present a rigorous definition of a problem and solving a problem, and cast the task of implementing one channel with another as a special case of solving a problem. We then describe two foundational composition results. The first shows how to compose an algorithm that solves a problem P using channel C_1 with an algorithm that implements $\mathcal{C}_1$ using channel $\mathcal{C}_2$. We prove the resulting composition solves P using $\mathcal{C}_2$. (The result is also generalized to work with a chain of channel implementation algorithms.) The second composition result shows how to compose a channel implementation algorithm $\mathcal{A}$ with a channel $\mathcal{C}$ to generate a new channel $\mathcal{C}'$. 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 $\mathcal{A}$. We conclude with a case study that demonstrates the framework and the composition theorems in action.

---

[1] We restrict our attention to synchronous settings as the vast majority of existing theoretical results for radio networks share this assumption. A more general *asynchronous* model remains important future work.

2 Model

We model n processes that operate in synchronized time slots and communicate on a radio network comprised of $\mathcal{F}$ independent communication frequencies. The processes can also receive inputs from and send outputs to an environment. We formalize this setting with automata definitions. Specifically, we use a probabilistic automaton for each of the n processes (which combine to form an *algorithm*), another to model the *environment*, and another to model the communication *channel*. A system is described by an algorithm, environment, and channel.

For any positive integer $x > 1$ we use the notation $[x]$ to refer to the integer set $\{1, ..., x\}$, and use S^x, for some set S, to describe all x-vectors with elements from S. Let $\mathcal{M}$, $\mathcal{R}$, $\mathcal{I}$, and $\mathcal{O}$ be four non-empty value sets that do not include the special placeholder value $\perp$. We use the notation $\mathcal{M}_\perp$, $\mathcal{R}_\perp$, $\mathcal{I}_\perp$, and $\mathcal{O}_\perp$ to describe the union of each of these sets with $\{\perp\}$. Finally, we fix n and $\mathcal{F}$ to be positive integers. They describe the number of processes and frequencies, respectively.

2.1 Systems

The primary object in our model is the *system*, which consists of an *environment* automaton, a *channel* automaton, and n process automata that combine to define an *algorithm*. We define each component below:

Definition 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 probability distribution over $cstates_\mathcal{C}$. (This distribution captures the probabilistic nature of the automaton. Both the environment and process definitions include similar distributions.)*
- *$crecv_\mathcal{C}$, a message set generation function that maps $cstates_\mathcal{C} \times \mathcal{M}_\perp^n \times [\mathcal{F}]^n$ to $\mathcal{R}_\perp^n$.*
- *$ctrans_\mathcal{C}$, a transition function that maps $cstates_\mathcal{C} \times \mathcal{M}_\perp^n \times [\mathcal{F}]^n$ to $cstates_\mathcal{C}$.*

Because we model a channel as an arbitrary automaton, we can capture a wide variety of possible channel behavior—from simple deterministic receive rules to complex, probabilistic multihop propagation.

We continue by the defining the elements of a system: an environment, process, and algorithm. We then define a system execution.

Definition 2 (Environment). *A environment is some 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 probability distribution over* $estates_{\mathcal{E}}$.
- $ein_{\mathcal{E}}$, *an* input generation function *that maps* $estates_{\mathcal{E}}$ *to* $\mathcal{I}_{\perp}^{n}$.
- $etrans_{\mathcal{E}}$, *a transition function that maps* $estates_{\mathcal{E}} \times \mathcal{O}_{\perp}$ *to* $estates_{\mathcal{E}}$.

Definition 3 (Process). *A* process *is some automaton* $\mathcal{P}$ *consisting of the following components:*

- $states_{\mathcal{P}}$, *a potentially infinite set of* states.
- $rand_{\mathcal{P}}$, *for each state* $s \in states_{\mathcal{P}}$, *is a probability distribution over* $states_{\mathcal{P}}$.
- $start_{\mathcal{P}}$, *a state from* $states_{\mathcal{P}}$ *known as the* start state.
- $msg_{\mathcal{P}}$, *a* message generation function *that maps* $states_{\mathcal{P}} \times \mathcal{I}_{\perp}$ *to* $\mathcal{M}_{\perp}$.
- $out_{\mathcal{P}}$, *an* output generation function *that maps* $states_{\mathcal{P}} \times \mathcal{I}_{\perp} \times \mathcal{R}_{\perp}$ *to* $\mathcal{O}_{\perp}$.
- $freq_{\mathcal{P}}$, *a* frequency selection function *that maps* $states_{\mathcal{P}} \times \mathcal{I}_{\perp}$ *to* $[\mathcal{F}]$.
- $trans_{\mathcal{P}}$, *a state transition function mapping* $states_{\mathcal{P}} \times \mathcal{R}_{\perp} \times \mathcal{I}_{\perp}$ *to* $states_{\mathcal{P}}$.

Definition 4 (Algorithm). *An algorithm* $\mathcal{A}$ *is a mapping from* $[n]$ *to processes.*

Definition 5 (System). *A system* $(\mathcal{E}, \mathcal{A}, \mathcal{C})$, *consists of an environment* $\mathcal{E}$, *an algorithm* $\mathcal{A}$, *and a channel* $\mathcal{C}$.

Definition 6 (Execution). *An* execution *of a system* $(\mathcal{E}, \mathcal{A}, \mathcal{C})$ *is an infinite sequence*

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

where for all $r \geq 0$, S_r *and* R_r^S *map each* $i \in [n]$ *to a process state from* $\mathcal{A}(i)$, 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. $C_0 = cstart_{\mathcal{C}}$, $E_0 = estart_{\mathcal{E}}$, *and* $\forall i \in [n] : S_0[i] = start_{\mathcal{A}(i)}$.
2. *For every round* $r > 0$:
   (a) $\forall i \in [n] : R_r^S[i]$ *is selected according to distribution* $rand_{\mathcal{A}(i)}(S_{r-1}[i])$, R_r^C *is selected according to* $crand_{\mathcal{C}}(C_{r-1})$, *and* R_r^E *is selected according to* $erand_{\mathcal{E}}(E_{r-1})$.
   (b) $I_r = ein_{\mathcal{E}}(R_r^E)$.
   (c) $\forall i \in [n] : M_r[i] = msg_{\mathcal{A}(i)}(R_r^S[i], I_r[i])$ *and* $F_r[i] = freq_{\mathcal{A}(i)}(R_r^S[i], I_r[i])$.
   (d) $N_r = crecv_{\mathcal{C}}(R_r^C, M_r, F_r)$.
   (e) $\forall i \in [n] : O_r[i] = out_{\mathcal{A}(i)}(R_r^S[i], I_r[i], N_r[i])$.
   (f) $\forall i \in [n] : S_r[i] = trans_{\mathcal{A}(i)}(R_r^S[i], N_r[i], I_r[i])$, $C_r = ctrans_{\mathcal{C}}(R_r^C, M_r, F_r)$, *and* $E_r = etrans_{\mathcal{E}}(R_r^E, O_r)$.

In each round: first the processes, environment, and channel transform their states (probabilistically); then the environment generates inputs to pass to the processes; then the processes each generate a message to send (or $\perp$ if they plan on receiving) and a frequency to use; then the channel returns the received messages to the processes; then the processes generate output values to pass back to the environment; and finally all automata transition to a new state.

Definition 7 (Execution Prefix). *An execution prefix of a system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$, is a finite prefix of some execution of the system. The prefix is either empty or ends with an environment state assignment E_r, $r \geq 0$. That is, it contains no partial rounds.*

2.2 Trace Probabilities

To capture the probability of various system behaviors we start by defining the function Q:

Definition 8 (Q). *For every system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$, and every execution prefix 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, the product of the probabilities of state transitions in α as described by $rand_{\mathcal{A}}$, $crand_{\mathcal{C}}$, and $erand_{\mathcal{E}}$.*

Next, we define a *trace* to be a finite sequence of vectors from $\mathcal{I}_\perp^n \cup \mathcal{O}_\perp^n$; i.e., a sequences of inputs and outputs passed between an algorithm and an environment. And we use T to describe the set of all traces Using Q, we can define functions that return the probability that a system generates a given trace. First, however, we need a collection of helper definitions to extract traces from prefixes. Specifically, the function io maps an execution prefix to the subsequence consisting only of the $\mathcal{I}_\perp^n$ and $\mathcal{O}_\perp^n$ vectors. The function cio, by contrast, maps an execution prefix α to $io(\alpha)$ with all $\perp^n$ vectors removed. Finally, the predicate $term$ returns $true$ for an execution prefix α if and only if the output vector in the final round of α is not $\perp^n$.

Definition 9 (D & D_{tf}). *For every system $(\mathcal{E}, \mathcal{A}, \mathcal{C})$, and every trace β :*
$D(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) = \sum_{\alpha | io(\alpha)=\beta} Q(\mathcal{E}, \mathcal{A}, \mathcal{C}, \alpha)$ *and*
$D_{tf}(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) = \sum_{\alpha | term(\alpha) \wedge cio(\alpha)=\beta} Q(\mathcal{E}, \mathcal{A}, \mathcal{C}, \alpha)$.

Both D and D_{tf} return the probability of a given system generating a given trace. 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 tf indicates it is *time-free*; e.g., it ignores the time required between the generation of trace elements.)

2.3 Problems

We define a problem and provide two definitions of solving a problem—one that considers empty rounds (those with $\perp^n$) and one that does not. In the following, let E be the set of all possible environments.

Definition 10 (Problem). *A problem P is a function from environments to a set of functions from traces to probabilities.*

Definition 11 (Solves & Time-Free Solves). *We say algorithm $\mathcal{A}$ solves problem P using channel $\mathcal{C}$ if and only if $\forall \mathcal{E} \in E, \exists F \in P(\mathcal{E}), \forall \beta \in T : D(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) = F(\beta)$. We say $\mathcal{A}$* time-free *solves P using $\mathcal{C}$ if and only if: $\forall \mathcal{E} \in E, \exists F \in P(\mathcal{E}), \forall \beta \in T : D_{tf}(\mathcal{E}, \mathcal{A}, \mathcal{C}, \beta) = F(\beta)$.*

For some of the proofs that follow, we need to restrict our attention to environments that are indifferent to delays.

Definition 12. *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. *$ein_{\mathcal{E}}(\hat{s}) = \perp^n$.*
2. *$erand_{\mathcal{E}}(\hat{s})(\hat{s}) = 1$.*
3. *$etrans_{\mathcal{E}}(\hat{s}, \perp^n) = \hat{s}$.*
4. *for every non-empty output assignment O, $etrans_{\mathcal{E}}(\hat{s}, O) = etrans_{\mathcal{E}}(s, O)$.*

When a delay tolerant environment receives output $\perp^n$ in some state s, it transitions to a special *marked* version of the current state, denoted $\hat{s}$, and cycles on this state until it next receives a non-$\perp^n$ output. In other words, it behaves the same regardless of how many consecutive $\perp^n$ outputs it receives. We use this definition of a delay tolerant environment to define a delay tolerant problem.

Definition 13 (Delay Tolerant Problem). *We say a problem P is delay tolerant if and only if for every environment $\mathcal{E}$ that is not delay tolerant, $P(E)$ returns the set containing every trace probability function.*

3 Implementing Channels

Here we construct a precise notion of implementing a channel with another channel as a special case of a problem. In the following, we say an input value is *send enabled* if it is from $(send \times \mathcal{M}_{\perp} \times \mathcal{F})$. We say an *input assignment* (i.e., vector from $\mathcal{I}_{\perp}^n$) is send enabled if all inputs values in the assignment are send enabled. Similarly, we say an output value is *receive enabled* if it is from $(recv \times \mathcal{R}_{\perp})$, and an *output assignment* (i.e., vector from $\mathcal{O}_{\perp}^n$) is receive enabled if all output values in the assignment are receive enabled. Finally, we say an input or output assignment is *empty* if it equals $\perp^n$

Definition 14 (Channel Environment). *An environment $\mathcal{E}$ is a channel environment if and only if it satisfies the following criteria: (1) It is delay tolerant; (2) it generates only send enabled and empty input assignments; and (3) it generates a send enabled input assignment in the first round and in every round $r > 1$ such that it received a receive enabled output vector in $r - 1$. In every other round it generates an empty input assignment.*

These constraints require the environment to pass down messages to send as inputs and then wait for the corresponding received messages, encoded as algorithm outputs, before continuing with the next batch messages to send. This formalism is used below in our definition of a channel problem. The natural pair to a channel environment is a channel algorithm, which behaves symmetrically.

Definition 15 (Channel Algorithm). *We say an algorithm $\mathcal{A}$ is a channel algorithm if and only if: (1) it only generates receive enabled and empty output assignments; (2) it never generates two consecutive received enabled output assignments without a send enabled input in between; and (3) given a send enabled input it eventually generates a receive enabled output.*

Definition 16 ($\mathcal{A}^I$). *Each process $\mathcal{P}$ of the channel identity algorithm $\mathcal{A}^I$ behaves as follows. If $\mathcal{P}$ receives a send enabled input $(send, m, f)$, it sends message m on frequency f during that round and generates output $(revc, m')$, where m' is the message it receives in this same round. Otherwise it sends $\perp$ on frequency 1 and generates output $\perp$.*

Definition 17 (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 $P_{\mathcal{C}}(\mathcal{E})$ returns the set containing every trace probability function.*

The effect of combining $\mathcal{E}$ with $\mathcal{A}^I$ and $\mathcal{C}$ is to *connect* $\mathcal{E}$ directly with $\mathcal{C}$. With the channel problem defined, we can conclude with what it means for an algorithm to implement a channel.

Definition 18 (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 $\mathcal{C}'$.*

4 Composition

We prove two useful composition results. The first simplifies the task of solving a complex problem on a weak channel into implementing a strong channel using a weak channel, then solving the problem on the strong channel. The second result simplifies proofs that require us to show that the channel implemented by a channel algorithm satisfies given automaton constraints.

4.1 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 $\mathcal{C}'$. We conclude with a corollary that generalizes this argument to a sequence of channel implementation arguments that start with $\mathcal{C}'$ and end with $\mathcal{C}$. Such compositions are key for a layered approach to radio network algorithm design.

Composition Algorithm Overview. 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 receiving only $\perp^n$ from the composed algorithm—necessitating its delay tolerance property). When $\mathcal{A}_\mathcal{C}$ finishes computing the received messages, we *unpause* $\mathcal{A}_P$ and finish the emulated round using these messages. The only tricky point in this construction is that when we pause $\mathcal{A}_P$ we need to store a copy of its input, as we will need this later to complete the simulated round once we unpause. The formal definition follows:

Definition 19 (The Composition Algorithm: $\mathcal{A}(\mathcal{A}, \mathcal{A}_\mathcal{C})$). *Let $\mathcal{A}_P$ be an algorithm and $\mathcal{A}_\mathcal{C}$ be a channel algorithm that implements channel $\mathcal{C}$ using channel $\mathcal{C}'$. Fix any $i \in [n]$. To simplify notation, let $A = \mathcal{A}(\mathcal{A}_P, \mathcal{A}_\mathcal{C})(i)$, $B = \mathcal{A}_P(i)$, and $C = \mathcal{A}_\mathcal{C}(i)$. 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:
  - *$siminput(s \in states_A, in \in \mathcal{I}_\perp)$: the function evaluates to $\perp$ if $s.status =$ paused, and otherwise evaluates to input:*
    $(send, msg_B(s.prob, in), freq_B(s.prob, in))$.
  - *$simrec(s \in states_A, in \in \mathcal{I}_\perp, m \in \mathcal{R}_\perp)$: the function evaluates to $\perp$ if $out_C(s.chan, siminput(s, in), m) = \perp$, otherwise if*
    $out_C(s.chan, siminput(s, in), m) = (recv, m')$ for some $m' \in \mathcal{R}_\perp$, it returns m'.
- $\mathbf{start_A} = (start_B, start_C, active, \perp)$.
- $\mathbf{msg_A(s, in)} = msg_C(s.chan, siminput(s, in))$.
- $\mathbf{freq_A(s, in)} = freq_C(s.chan, siminput(s, in))$.
- $\mathbf{out_A(s, in, m)}$ *: let $m' = simrec(s.chan, siminput(s, in), m)$. The out_A function evaluates to $\perp$ if $m' = \perp$, or $out_B(s.prob, s.input, m')$ if $m' \neq \perp$ and $s.state = passive$, or $out_B(s.prob, in, m')$ if $m' \neq \perp$ and $s.state = active$.*
- $\mathbf{rand_A(s)(s')}$ *: the distribution evaluates to $rand_C(s.chan)(s'.chan)$ if $s.status = s'.status = paused$, $s.input = s'.input$, and $s.prob = s'.prob$, or evaluates to $rand_B(s.prob)(s'.prob) \cdot rand_C(s.chan)(s'.chan)$ if $s.status = s'.status = active$ and $s.input = s'.input$, or evaluates to 0 if neither of the above two cases hold.*
- $\mathbf{trans_A(s, m, in)} = s'$ *where we define s' as follows. As in our definition of out_A, we let $m' = simrec(s.chan, siminput(s, in), m)$:*
  - *$s'.prob = trans_B(s.prob, m', s.input)$ if $m' \neq \perp$ and $s.status = paused$, or $trans_B(s.prob, m', in)$ if $m' \neq \perp$ and $s.status = active$, or $s.prob$ if neither of the above two cases hold.*
  - *$s'.chan = trans_C(s.chan, m, siminput(s, in))$.*

- $s'.input = in$ *if* $in \neq \bot$*, otherwise it equals* $s.input$*.*
- $s'.status = active$ *if* $m' \neq \bot$*, otherwise it equals* $paused$*.*

We now prove that this composition works (i.e., solves P on $\mathcal{C}'$). Our strategy uses channel-free prefixes: execution prefixes with the channel states removed. We define two functions for extracting these prefixes. The first, *simpleReduce*, removes the channel states from an execution prefix. The second, *compReduce*, extracts the channel-free prefix that describes the emulated execution prefix of $\mathcal{A}_P$ captured in an execution prefix of a (*complex*) system that includes a composition algorithm consisting of $\mathcal{A}_P$ and a channel implementation algorithm.

Definition 20 (Channel-Free Prefix). *We define a sequence* α *to be a channel-free prefix of an environment* $\mathcal{E}$ *and algorithm* $\mathcal{A}$ *if and only if there exists an execution prefix* α' *of a system including* $\mathcal{E}$ *and* $\mathcal{A}$*, such that* α *describes* α' *with the channel state assignments removed.*

Definition 21 (simpleReduce). *Let* $\mathcal{E}$ *be a delay tolerant environment,* $\mathcal{A}_P$ *be an algorithm, and* $\mathcal{C}$ *a channel. Let* α *be an execution prefix of the system* $(\mathcal{E}, \mathcal{A}_P, \mathcal{C})$*. We define* $simpleReduce(\alpha)$ *to be the channel-free prefix of* $\mathcal{E}$ *and* $\mathcal{A}_P$ *that results when remove the channel state assignments from* α*.*

Definition 22 (*compReduce*). *Let* $\mathcal{E}$ *be a delay tolerant environment,* $\mathcal{A}_P$ *be an algorithm,* $\mathcal{A}_\mathcal{C}$ *be a channel algorithm, and* $\mathcal{C}'$ *a channel. Let* α' *be an execution prefix of the system* $(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_\mathcal{C}), \mathcal{C}')$*. We define* $compReduce(\alpha')$ *to be the channel-free prefix of* $\mathcal{E}$ *and* $\mathcal{A}_P$ *that describes the emulated execution of* $\mathcal{A}_P$ *encoded in the composition algorithm state. If the final round of* α' *is in the middle of an emulated round of* $\mathcal{A}_P$ *(that is, the simulated output of* $\mathcal{A}_\mathcal{C}$*—described by* simoutput *in the formal definition of the composition algorithm—is* $\bot^n$ *for this round), we return the special marker* null*. This matches the intuition that the emulated execution is undefined for such prefixes. Otherwise, we return the channel-free prefix* α *defined as follows:*

1. *Divide* α' *into* emulated rounds*. Each emulated round begins with a round in which* siminput *returns a send enabled input for all processes and ends with the next round in which* simrec *returns a message at every process. (Recall,* simrec *and* siminput *are defined in the definition of the composition algorithm.) It is possible that this is the same round; i.e., if the* $\mathcal{A}_C$ *emulation of* $\mathcal{C}$ *used only a single round.*
2. *For each emulated round* r *of* α'*, we define the corresponding round* r *of* α *as follows:*
   - *(a) Set the randomized algorithm state* (R^S_r) *equal to the algorithm state described in the* prob *component of the algorithm state from the first round of the emulated round.*
   - *(b) Set the send message and frequency assignments equal to the values generated by applying* siminput *to the first round of the emulated round.*
   - *(c) Set the receive message assignments equal to the values generated by applying* simrec *to the last round of the emulated round.*

(d) *Set the input assignment equal to the input assignment from the first round of the emulated round.*
(e) *Set the output assignment equal to the output assignment from the last round of the emulated round.*
(f) *Set the final algorithm state (S_r) equal to the final algorithm state of the last round of the emulated round.*
(g) *Set the final environment state to the the final environment state of the last round of the emulated round.*

In other words, we are extracting the simulation of A_P running on C that is captured in the composition algorithm. Roughly speaking, we are projecting the algorithm state onto the *prob* component and removing the rounds in which $\mathcal{A}_P$ was paused. Though in reality it is slightly more complicated as we also have to "glue" the initial states of the first round and the final states of the last round of the emulated round together (for both the algorithm and environment). In addition, we also have to replace the message and frequency assignments with the values emulated in the composition algorithm.

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 $\mathcal{A}_P$, unfolds the same as $\mathcal{A}_P$ running by itself.

Lemma 1. *Let $\mathcal{E}$ be a delay tolerant environment, $\mathcal{A}_P$ be an algorithm, and $\mathcal{A}_\mathcal{C}$ be a channel algorithm that implements $\mathcal{C}$ with $\mathcal{C}'$. Let α be a channel-free prefix of $\mathcal{E}$ and $\mathcal{A}_P$. It follows:*

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

Proof. To simplify notation, we begin with two helper definitions. Let $S'_s(\gamma)$, for channel-free prefix γ of $\mathcal{E}$ and $\mathcal{A}_P$, contain every prefix α' of $(\mathcal{E}, \mathcal{A}_P, \mathcal{C})$ such that $simpleReduce(\alpha') = \gamma$. Let $S'_c(\gamma)$ contain every prefix α'' of $(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_\mathcal{C}), \mathcal{C}')$ such that $compReduce(\alpha'') = \gamma$.

We proceed by induction on the rounds of α. Assume there are R total rounds. We use the notation $\alpha[r]$, $0 \le r \le R$, to refer to the prefix of α through round r. Our hypothesis assumes that for a given r, $0 \le r < R$:

$$\sum_{\alpha' \in S'_s(\alpha[r])} Q(\mathcal{E}, \mathcal{A}_P, \mathcal{C}, \alpha') = \sum_{\alpha'' \in S'_c(\alpha[r])} Q(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_\mathcal{C}), \mathcal{C}', \alpha'')$$

The base case of $r = 0$ is trivial, as $S'_s(\alpha[0])$ and $S'_c(\alpha[0])$ both contain the single unique start state of their respective system. All systems generate their start state with same probability: 1.

We continue with the inductive step. Fix some $r < R$ for which our hypothesis holds. Consider any $\alpha' \in S'_s(\alpha[r])$ and $\alpha'' \in S'_c(\alpha[r])$. Let s_e by the final environment state in α'. Let s_p^i, $\forall i \in [n]$, be the final state for process i in α'. We first claim that s_e is also the final environment state in α'' and s_p^i, $\forall i \in [n]$, is the state stored in the *prob* component of the final process i state in α''. This follows from our assumption that $compReduce(\alpha'') = simpleReduce(\alpha')$.

Now let us consider how the two systems unfold for round $r+1$. We begin with the input. In the *simple* system (i.e., $(\mathcal{E}, \mathcal{A}_P, \mathcal{C})$), the new environment state is chosen by the distribution $erand_{\mathcal{E}}(s_e)$ which then uniquely determines the input for this round. In the *complex* system (i.e., $(\mathcal{E}, \mathcal{A}(\mathcal{A}_P, \mathcal{A}_{\mathcal{C}}), \mathcal{C}')$), the new environment state is chosen by the same distribution. It follows that the inputs for round $r+1$ are chosen the same in both systems.

We turn next to the algorithms random transitions. For each process i, both systems probabilistically chose the state for this $r+1$ using the same distribution $rand_{\mathcal{A}_P(i)}(s_p^i)$. The messages and frequencies are uniquely determined by this choice of state and the input.

We continue with the received messages. By the definition of *implements*, given a message and frequency assignment, the emulation of $\mathcal{A}_C$ will return a given received message assignment with the same probability as the actual channel $\mathcal{C}$. Therefore, the probability of each possible received message assignment—given the same messages sent—is the same in both systems. The output is uniquely determined by these messages and the input.

We conclude with the final state transitions. If the emulation of $\mathcal{A}_C$ took multiple rounds, the delay tolerant environment $\mathcal{E}$ has been cycling on $\hat{s}_e$. The transition of $\hat{s}_e$, given the non-$\perp^n$ input eventually generated by the composition algorithm, behaves identically to the same transition given s_e (by the definition of delay tolerant), so, given the same outputs, both systems transition to the same environment state. The same holds true for the process states. As both systems unfold according to the same distributions, their probabilities of generating $\alpha[r+1]$ (by their respective *reduce* functions) are the same. □

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

Theorem 1 (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:

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

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

We continue with a series of 4 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, 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 prefix, 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}:

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

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 1 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 1 (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 1 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 1 $j-1$ times to arrive at algorithm $\mathcal{A}_{P,j}$ that time-free solves P using $\mathcal{C}_j$.

4.2 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 shown in Sect. 5, it is often easier to talk about all channels that satisfy a property than to talk about a specific channel.) You can 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.

Composition Channel Overview. 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 prefix 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 prefixes 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.

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

Definition 23 (*toinput*)**.** *Let the function toinput map pairs from* $\mathcal{M}_\perp^n \times [\mathcal{F}]^n$ *to the corresponding send enabled input assignment describing these messages and frequencies.*

Definition 24 (Environment-Free Prefix). *We define a sequence of assignments* α *to be an environment-free prefix of an algorithm* $\mathcal{A}$ *and channel* $\mathcal{C}$ *if and only if there exists an execution prefix* α'*, of a system including* $\mathcal{A}$ *and* $\mathcal{C}$*, such that* α *describes* α' *with the environment state assignments removed.*

Definition 25 (State Extension). *Let* α *be an environment-free prefix of some 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 prefix α by the next states of the channel and algorithm. We continue with a longer extension.

Definition 26 (I-Output Extension). *Let* α *be an environment-free prefix of some algorithm* $\mathcal{A}$ *and channel* $\mathcal{C}$*. Let* α' *be a state extension of* α*. We define a* 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 enabled output.*

In other words, we extend our state extension with a particular input, after which we run it with empty inputs until it outputs are receive enabled (if this never happens, the extension is infinite).

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

Definition 27 (The Composition Channel: $\mathcal{C}(\mathcal{A}, \mathcal{C}')$). *Let* $\mathcal{A}$ *be a channel algorithm and* $\mathcal{C}'$ *be a channel. To simplify notation, let* $C = \mathcal{C}(\mathcal{A}, \mathcal{C}')$*. We define the composition channel* C *as follows:*

1. **cstates$_{\mathbf{C}}$** *contains two types of states:* simple states *that encode only an environment-free prefix of $\mathcal{A}$ and $\mathcal{C}'$; and* complex states *that encode an environment-free prefix α of $\mathcal{A}$ and $\mathcal{C}'$, a state extension α' of α, and a table that maps every pair, ($M \in \mathcal{M}_{\perp}^{n}$,$F \in [\mathcal{F}]^{n}$) to an ($toinput(M,F)$)-output extension of α'.*
   (For any simple state s, we use the notation $s.pre$ to describe the environment-free prefix encoded in s. For any complex state c, we use $c.pre$ to describe the environment-free prefix, $c.ext$ to describe the state extension, and $c.oext(M,F)$, for message and frequency assignments M and F, to describe the output extension encoded in the table for those assignments.)
2. **cstart$_{\mathbf{C}}$** *equals the simple state s_0 where $s_0.pre$ describes the 0-round environment-free prefix of $\mathcal{A}$ and $\mathcal{C}'$.*
3. **crand$_{\mathbf{C}}$** *is defined for a simple state s, as follows: For every complex state q: if $q.pre \neq s.pre$, then $crand_C(s)(q) = 0$. Else, the distribution returns q with the probability described by the probability that $q.pre$ extends to $q.ext$ multiplied by the product of the probabilities of every extension in $q.oext$ conditioned on $q.ext$ and the relevant input.*
4. **ctrans$_{\mathbf{C}}$**, *when passed complex channel state q, message assignment $M \in msg_{\perp}^{n}$, and frequency assignment $F \in [\mathcal{F}]^{n}$, returns the simple state s, where $s.pre = q.oext(M,F)$.*
5. **crecv$_{\mathbf{C}}$**, *when passed complex channel state q, message assignment $M \in msg_{\perp}^{n}$, and frequency assignment $F \in [\mathcal{F}]^{n}$, returns the messages encoded in the receive enabled output of the final round of $q.oext(M,F)$.*

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

Definition 28 (ex)**.** *Let α be an execution prefix of a 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. Let s be the final state of $\mathcal{C}(\mathcal{A},\mathcal{C}')$ in α. We define $ex(\alpha)$ to be the execution prefix of $(\mathcal{E},\mathcal{A},\mathcal{C}')$ that results when we take the environment-free prefix $s.pre$ and then add in states of $\mathcal{E}$ to match the inputs encoded in $s.pre$. To do so, we first add the states of $\mathcal{E}$ from α to the corresponding rounds in $s.pre$ where the inputs generated from those states appear. In all other rounds in $s.pre$ we add the marked version of the most recent preceding state from $\mathcal{E}$. (Recall, because $\mathcal{E}$ is a channel environment it is also delay tolerant, meaning the addition of these marked states leaves a valid prefix.)*

Notice, the new rounds added to α in *s.pre*—that is, the rounds that capture the simulation of $\mathcal{A}$ and $\mathcal{C}'$ captured by the composition channel—all return $\perp^n$ as output. Therefore, by simply adding marked versions of the most recent preceding $\mathcal{E}$ state to these new rounds, the resulting evolution of $\mathcal{E}$ in the prefix remains valid.

Definition 29 ($comp$)**.** *Let α' be an execution prefix of a system $(\mathcal{E},\mathcal{A},\mathcal{C}')$, $\mathcal{E}$ be channel environment, and $\mathcal{A}$ be channel algorithm. Let $comp(\alpha')$ be the set that contains every execution prefix α of $(\mathcal{E},\mathcal{A}^{I},\mathcal{C}(\mathcal{A},\mathcal{C}'))$ such that $ex(\alpha) = \alpha'$.*

Theorem 2 (The Composition Implementation Theorem). *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 $\mathcal{C}'$.*

To prove this theorem, we begin with a useful helper lemma.

Lemma 2. *Let α' be an execution prefix of a system $(\mathcal{E},\mathcal{A},\mathcal{C}')$, where $\mathcal{E}$ is a channel environment, $\mathcal{A}$ is a channel algorithm, $\mathcal{C}'$ is a channel, and the final output in α' is receive enabled. It follows: $Q(\mathcal{E},\mathcal{A},\mathcal{C}',\alpha') = \sum_{\alpha\in comp(\alpha')} Q(\mathcal{E},\mathcal{A}^I,\mathcal{C}(\mathcal{A},\mathcal{C}'),\alpha)$*

Proof. Divide α' into execution fragments, each beginning with a round with a send enabled input and ending with the round containing the corresponding receive enabled output. We call these fragments *emulated rounds*. By our assumption that α' ends with a receive enabled output, no part of α' falls outside of an emulated round.

Fix some emulated round e_r of α'. Let I be the send enabled input passed down by the environment during e_r. Let M and F be the message and frequency assignments encoded in I.

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 prefix obtained by removing the environment state assignments from the prefix 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, however, because the entries can vary freely in the other rows of $oext$.)

We prove the following claim which we will subsequently wield to prove the lemma:

1. Let p_I describe the probability that e_r occurs given the prefix of α' through e_r-1 and the input I. It follows:

$$\sum_{q\in X} crand_{\mathcal{C}}(s)(q) = p_I$$

To prove our claim, we unwind the definition of $crand_{\mathcal{C}}$ and then simplify:

$$\begin{aligned}\sum_{q\in X} crand_{\mathcal{C}}(s)(q) &= \sum_{q\in X} Pr[q.ext|q.pre] \prod_{(M',F')} Pr[q.oext(M',F')|s.ext,(M',F')] \\ &= p_I \sum_{q\in X} \prod_{(M',F')\neq(M,F)} Pr[q.oext(M',F')|s.ext,(M',F')] \\ &= p_I\end{aligned}$$

Notice, the final simplification depends on the fact that this product considers every possible combination of extensions for the non-(M,F) rows.

We now apply this claim, which concerns only a single emulated round, to prove the lemma, which concerns the entire prefix 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 execution prefix of α' through emulated round r. Notice, because we assumed that α' ends with a receive enabled 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, given 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 prefix generated by removing the environment assignment states from $\alpha'[r]$.

We know the probability that $\mathcal{E}$ passes down I for 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 prefixes in $comp(\alpha'[r])$. Finally, by applying claim 1 from above, we determine that given a prefix that ends in s_r, the probability that it transform by $crand_{\mathcal{C}}$ to a state that describes $\alpha'[r+1]$ (given I) equals the probability that $\alpha'[r]$ transforms to $\alpha'[r+1]$ (also given I). This combines to prove the inductive step.

To summarize: the probability of a prefix in $comp(\alpha'[r])$ equals the probability of $\alpha'[r]$. The probability of a prefix in $comp(\alpha'[r])$ extending to a prefix in $comp(\alpha'[r+1])$, given the input I, is the same as $\alpha'[r]$ extending to $\alpha'[r+1]$ given I. Finally, the probability of I is the same in both.

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

We now return to the main theorem.

Proof (of Theorem 2). 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: $S_{comp} = (\mathcal{E}, A^I, \mathcal{C})$, and $S = (\mathcal{E}, \mathcal{A}, \mathcal{C}')$. We now rewrite our equality regarding D^{tf} in terms of Q:

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

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 2, we know:

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

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 execution prefixes 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 prefix included in the second sum is covered by one of these *comp* sets. Let α be a prefix 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 $comp(ex(\alpha))$. $\square$

5 Case Study

We highlight the power and flexibility of our framework with a simple example. We begin by defining two types of channels: p-reliable and t-disrupted. The former is an idealized single-hop single-frequency radio channel with a probabilistic guarantee of successful delivery (e.g., as considered in [15]). The latter is a realistic single-hop radio channel, comprised of multiple independent frequencies, up to t of which might be permentantly disrupted by outside sources of interference (e.g., as considered in [16]). We then describe a simple algorithm $\mathcal{A}_{rel}$ and sketch a proof that it implements the reliable channel using the disrupted channel. Before defining the two channel types, however, we begin with this basic property used by both:

Definition 30 (Basic Broadcast Property). *We say a channel $\mathcal{C}$ satisfies the basic broadcast property if and only if for every state s, message assignment M, and frequency assignments F, $N = crecv_{\mathcal{C}}(s, M, F)$ satisfies the following:*

1. *If $M[i] \neq \bot$ for some $i \in [n]$: $N[i] = M[i]$.*
   (Broadcasters receive their own messages.)
2. *If $N[i] \neq \bot$, for some $i \in [n]$, then there exists a $j \in [n] : M[j] = N[i] \land F[j] = F[i]$.*
   (If i receives a message then some process sent that message on the same frequency as i.)

3. *If there exists some $i, j, k \in [n], i \neq j \neq k$, such that $F[i] = F[j] = F[k]$, $M[i] \neq \perp$, $M[j] \neq \perp$, and $M[k] = \perp$, it follows that $N[k] = \perp$.*
   (Two or more broadcasters on the same frequency cause a collision at receivers on this frequency.)

Definition 31 (p-Reliable Channel). *We say a channel $\mathcal{C}$ satisfies the p-reliable channel property, $p \in [0, 1]$, if and only if $\mathcal{C}$ satisfies the basic broadcast property, and there exists a subset S of the states, such that for every state s, message assignment M, and frequency assignments F, $N = crecv_{\mathcal{C}}(s, M, F)$ satisfies the following:*

1. *If $F[i] > 1 \wedge M[i] = \perp$, for some $i \in [n]$, then $N[i] = \perp$.*
   (Receivers on frequencies other than 1 receive nothing.)
2. *If $s \in S$ and $|\{i \in [n] : F[i] = 1, M[i] \neq \perp\}| = 1$, then for all $j \in [n]$ such that $F[j] = 1$ and $M[j] = \perp$: $N[j] = M[i]$.*
   (If there is a single broadcaster on frequency 1, and the channel is in a state from S, then all receivers on frequency 1 receive its message.)
3. *For any state s', $\sum_{s \in S} crand_{\mathcal{C}}(s')(s) \geq p$.*
   (The probability that we transition into a state in S—i.e., a state that guarantees reliable message delivery—is at least p.)

Definition 32 (t-Disrupted Channel). *We say a channel $\mathcal{C}$ satisfies the t-disrupted property, $0 \leq t < \mathcal{F}$, if and only if $\mathcal{C}$ satisfies the basic broadcast channel property, and there exists a set $B_t \subset [\mathcal{F}]$, $|B_t| \leq t$, such that for every state s, message assignment M, and frequency assignment F, $N = crecv_{\mathcal{C}}(s, M, F)$ satisfies the following:*

1. *If $M[i] = \perp$ and $F[i] \in B_t$, for some $i \in [n]$: $N[i] = \perp$.*
   (Receivers receive nothing if they receive on a disrupted frequency.)
2. *If for some $f \in [\mathcal{F}], f \notin B_t$, $|\{i \in [n] : F[i] = f, M[i] \neq \perp\}| = 1$, then for all $j \in [n]$ such that $F[j] = f$ and $M[j] = \perp$, $N[j] = M[i]$, where i is the single process from the above set of broadcasters on f.*
   (If there is a single broadcaster on a non-disrupted frequency then all receivers on that frequency receive the message.)

Consider the channel algorithm, $\mathcal{A}_{rel}$, that works as follows: The randomized transition $rand_{\mathcal{A}_{rel}(i)}$ encodes a random frequency f_i for each process i in the resulting state. This choice is made independently and at random for each process. If a process $\mathcal{A}_{rel}(i)$ receives an input from $(send, m \in \mathcal{M}, 1)$, it broadcasts m on frequency f_i and outputs $(recv, m)$. If the process receives input $(send, \perp, 1)$ it receives on f_i, and then outputs $(recv, m')$, where m' is the message it receives. Otherwise, it outputs $(recv, \perp)$. We now prove that $\mathcal{A}_{rel}$ implements a reliable channel using a disrupted channel.

Theorem 3. *Fix some t, $0 \leq t < \mathcal{F}$. Given any channel $\mathcal{C}$ that satisfies the t-disrupted channel property, the algorithm $\mathcal{A}_{rel}$ implements a channel that satisfies the $(\frac{\mathcal{F}-t}{\mathcal{F}^n})$-reliable channel property using $\mathcal{C}$.*

Proof (Sketch). By Theorem 2 we know $\mathcal{A}_{rel}$ implements $\mathcal{C}(\mathcal{A}_{rel},\mathcal{C})$ using $\mathcal{C}$. We are left to show that $\mathcal{C}(\mathcal{A}_{rel},\mathcal{C})$ satisfies the $(\frac{\mathcal{F}-t}{\mathcal{F}^n})$-reliable channel property. Condition 1 of this property follows from the definition of $\mathcal{A}_{rel}$. More interesting is the combination of 2 and 3. Let B_t be the set of disrupted frequencies associated with $\mathcal{C}$. A state s returned by $crand_{\mathcal{C}(\mathcal{A}_{rel},\mathcal{C})}$ is in S if the final state of $\mathcal{A}_{rel}$ in $s.ext$ encodes the same f value for all processes, and this value is not in B_t. Because each process chooses this f value independently and at random, this occurs with probability at least $(\frac{\mathcal{F}-t}{\mathcal{F}^n})$. $\square$

Next, imagine that we have some algorithm $\mathcal{A}_P$ that solves a delay tolerant problem P (such as randomized consensus, which is easily defined in a delay tolerant manner) on a $(\frac{\mathcal{F}-t}{\mathcal{F}^n})$-reliable channel. We can apply Theorem 1 to directly derive that $\mathcal{A}(\mathcal{A}_P,\mathcal{A}_{rel})$ solves P on any t-disrupted channel $\mathcal{C}'$. In a similar spirit, imagine we have an algorithm $\mathcal{A}_{rel}^{+}$ that implements a (1/2)-reliable channel using a $(\frac{\mathcal{F}-t}{\mathcal{F}^n})$-reliable channel, and we have an algorithm $\mathcal{A}_{P'}$ that solves delay tolerant problem P' on a (1/2)-reliable channel. We could apply Corollary 1 to $\mathcal{A}_{P'}$, $\mathcal{A}_{rel}^{+}$, and $\mathcal{A}_{rel}$, to identify an algorithm that solves P' on our t-disrupted channel. And so on.

6 Conclusion

In this paper we present a modeling framework for synchronous probabilistic radio networks. The framework allows for the precise definition of radio channels and includes a pair of composition results that simplify a layered approach to network design (e.g., implementing stronger channels with weaker channels). We argue that this framework can help algorithm designers sidestep problems due to informal model definitions and more easily build new results using existing results. Much future work remains regarding this research direction, including the formalization of well-known results, exploration of more advanced channel definitions (e.g., multihop networks or adversarial sources of error), and the construction of implementation algorithms to link existing channel definitions.

References

1. Abramson, N.: The Aloha system - Another approach for computer communications. The Proceedings of the Fall Joint Computer Conference **37** (1970) 281–285
2. Roberts, L.G.: Aloha packet system with and without slots and capture. In: ASS Note 8. Advanced Research Projects Agency, Network Information Center, Stanford Research Institute (1972)
3. Kleinrock, L., Tobagi, F.: Packet switching in radio channels. IEEE Transactions on Communications **COM-23** (1975) 1400–1416
4. Hajek, B., van Loon, T.: Decentralized dynamic control of a multiaccess broadcast channel. IEEE Transactions on Automation and Control **AC-27** (1979) 559–569
5. Bar-Yehuda, R., Goldreich, O., Itai, A.: 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) (1992) 104–126

6. Chlamtac, I., Weinstein, O.: The wave expansion approach to broakodcasting in multihop radio networks. IEEE Transactions on Communications **39** (1991) 426–433
7. Clementi, A., Monti, A., Silvestri, R.: Round robin is optimal for fault-tolerant broadcasting on wireless networks. Journal of Parallel and Distributed Computing **64**(1) (2004) 89–96
8. Kowalski, D., Pelc, A.: Broadcasting in undirected ad hoc radio networks. In: The Proceedings of the International Symposium on Principles of Distributed Computing. (2003) 73–82
9. Kowalski, D., Pelc, A.: Time of deterministic broadcasting in radio networks with local knowledge. SIAM Journal on Computing **33**(4) (2004) 870–891
10. Chockler, G., Demirbas, M., Gilbert, S., Lynch, N., Newport, C., Nolte, T.: Consensus and collision detectors in radio networks. Distributed Computing **21** (2008) 55–84
11. Wu, S.H., Smolka, S.A., Stark, E.W.: Composition and behaviors of probabilistic I/O automata. In: The Proceedings of the International Conference on Concurrency Theory. (1994)
12. Segala, R.: Modeling and verification of randomized distributed real-time systems. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (June 1995)
13. Cheung, L.: Reconciling Nondeterministic and Probabilistic Choices. PhD thesis, Radboud University Nijmege (2006)
14. Newport, C., Lynch, N.: Modeling radio networks. Technical report, MIT CSAIL (2009)
15. Bar-Yehuda, R., Goldreich, O., Itai, A.: Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. Distributed Computing **5** (1991) 67–71
16. Gilbert, S., Guerraoui, R., Kowalski, D., Newport, C.: Interference-resilient information exchange. In: The Proceedings of the Conference on Computer Communication. (2009)