Building a Theory of Distributed Systems: Work by Nancy Lynch and Collaborators

Nancy Lynch

January 7, 2025

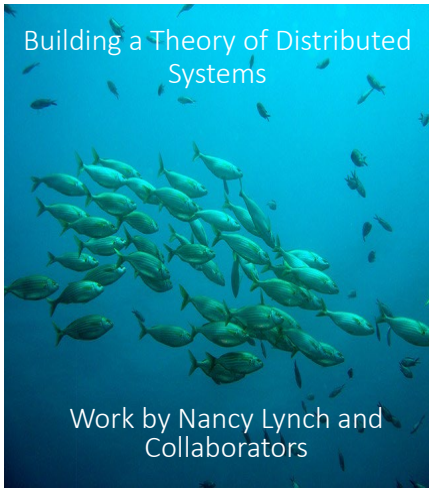

Figure 1: If this were a book, this could be its cover design. The school of fish is a kind of natural distributed system. The school could also represent all the collaborators.

1 Introduction

In this manuscript, I summarize research by myself and my very many students and other collaborators, on developing a theory for the field of distributed computing. I hope that it provides an interesting look at some of the early work that helped to build this field.

The main theme of our research in distributed computing theory is simply this: *Distributed systems are everywhere, and we need a usable formal theory to support their design and development.* Moreover, a formal theory for distributed computing requires new kinds of models and theoretical techniques,

different from those used for sequential computing. The new theory should support formal description of distributed algorithms and systems, proofs of their correctness, and analysis of their performance. Also, it should support proofs of impossibility results. Impossibility results are very hard to prove for sequential algorithms, but they are feasible for distributed algorithms, because the platforms on which distributed algorithms run are much less well-behaved.

Contents of this manuscript: Our research in distributed computing theory spans from 1976 until the present (almost 50 years!), and is contained in many hundreds of papers—far too many for any reader to sort through. To make this manageable, I have selected a small number of "key publications" on which to focus: 25 papers and 3 books. In choosing these, I have favored papers that were influential in the field, papers that were influential in our own later work, papers that I was especially invested in, and some other favorites. I apologize ahead of time for the many other wonderful papers that I have omitted, particularly some excellent papers by my graduate students. Although I have focused the discussion on these 28 publications, along the way I mention many more.

The manuscript begins with a brief Section 2 describing my background in complexity theory, which is what I worked on before I got interested in distributed computing. Sections 3 and 4 describe our early work on the beginnings of the field, through 1990, starting at Georgia Tech and continuing at MIT. My work at Georgia Tech included basic algorithms and lower bounds for shared-memory distributed systems solving problems such as mutual exclusion and resource allocation. It also included preliminary work on the problem of distributed consensus, which became a very popular research direction for the field, and on general formal models for distributed computing. My early MIT work involved extensive research on distributed consensus, including exact and approximate versions of the problem, and work based on different timing models. Formal models for asynchronous distributed computing were another focus, as well as applications to distributed database transaction processing.

Sections 5 and 6 describe work at MIT during the period 1990-2005. This included two distinct threads of research. Section 5 describes work on algorithms and lower bounds, including work on communication protocols, on timing aspects of algorithms, on variations of the consensus problem, and on data consistency. I also wrote my Distributed Algorithms textbook during this time. Section 6 describes work on formal models and methods for distributed systems, including work on timed system models, hybrid (continuous/discrete) system models, and probabilistic system models. Section 6 also includes some applications of these formal models and methods to some data management algorithms, timed and hybrid systems, and security protocols.

Section 7 summarizes our recent work (after 2005) on algorithms for wireless networks, including algorithms for distributed data management and algorithms built over a Virtual Node abstraction layer. Finally, Section 8 touches very briefly on our recent work on biological distributed algorithms, specifically, for insect colonies and brain networks.

Accessing the papers: For some help in accessing the key papers, here is a link to a web page that contains a list of the key publications listed in this manuscript, with some information about places where they can be found: https://groups.csail.mit.edu/tds/lynch-papers-highlights.html. In most cases, I have pointed to the publishers' pages for the final versions of the papers. For a few papers that I cannot locate easily, or for which we have made significant corrections post-publication, we have included a .pdf.

2 Complexity Theory

Abstract complexity theory: I wrote my PhD thesis in the subfield of *abstract complexity theory*, working with Profs. Albert Meyer and Michael Fischer. The paper [105] summarizes my PhD work. Briefly, this paper and thesis extended Manuel Blum's axiomatic treatment of the computational complexity of partial recursive functions to relatively computable functions, as computed, for example, by Turing machines with oracles. The paper went on to study reducibilities that are defined by complexity bounds.

So does this have anything to do with the topic of this manuscript, which is the foundations of distributed computing? Well, a general and loose connection is that reducibilities are about solving computational problems in terms of other computational problems. This is a theme that arises as key in distributed computing theory, in the form of decomposing complex distributed algorithms into smaller pieces.

Polynomial-time reducibilities: The paper [73], joint with Richard Ladner and Alan Selman, moved away from abstract complexity theory to study *polynomial-time reducibilities*, such as the two defined by Cook [23] and Karp [68] respectively. The point of the paper [73] was that many different forms of polynomial-time reducibility had been defined, or could be defined, varying according to the structure of the reducibility. The paper [73] established the relative strengths of the different reducibilities, with inclusion and separation results.

But after this I changed research areas...

3 Early Explorations of Theory for Distributed Computing, at Georgia Tech, 1976-1981

I started a faculty job at Georgia Tech's School of Information and Computer Science in 1976, where I spent my first year or two exploring new areas of research. I was looking for places where theoretical computer science methods might apply to practical computing. Pretty quickly, I gravitated to the new area of distributed computing.

At that time, the Arpanet was fairly new, and researchers and developers had begun talking about a new type of computing that they called "distributed computing". At Georgia Tech, I was influenced by Prof. Phil Enslow's work and advocacy for the new area. Phil wrote an interesting position paper [67] that delineated many features that a distributed system should have. Basically, his vision amounted to a high-level, general-purpose programming platform running over a communication network like the Arpanet. It was supposed to manage distributed data processing, and also coordination of tasks to be performed on many computers.

I became inspired by these possibilities. Phil suggested many papers for me to read, notably, papers on distributed database concurrency control. He understood that I was interested in theoretical aspects of distributed computing, and pointed me to relevant papers by Leslie Lamport and Edsger Dijkstra. I became very interested in working in the area.

As I began thinking about distributed systems issues, I talked with Mike Fischer at one of the theoretical computer science conferences. It turned out that Mike was already working on problems in this area, with his PhD student Gary Peterson. Their emphasis was on shared-memory algorithms for mutual exclusion; see [116]. Mike and I began discussing the area, along with Gary and my (first) PhD student Jim Burns.

Before too long, in around 1978, Mike arranged to take a sabbatical at Georgia Tech in order to work with me and Jim Burns on this topic. Together we hosted a series of short visits, by Leslie Lamport, Eshrat Arjomandi, Alan Borodin, and others. At this time, Lamport was already well known for his early work on distributed computing theory, in particular, for the Bakery Algorithm and a collection of algorithms for implementing read/write registers; he had also begun working on the problem of distributed consensus. Also during this time, Prof. Nancy Griffeth joined the School at Georgia Tech; she worked on database algorithms, and also began working with us on distributed resource allocation algorithms.

Overall, I think my years at Georgia Tech were quite productive, in terms of producing concrete theoretical results in the new research area of distributed computing theory. I discuss some of the highlights of this work below.

3.1 Key publications

3.1. James E. Burns, Paul Jackson, Nancy A. Lynch, Michael J. Fischer, and Gary L. Peterson. Data requirements for implementation of N-process mutual exclusion using a single shared variable. Journal of the ACM, 29(1):183–205, January 1982.

3.2. James E. Burns and Nancy A. Lynch. Bounds on shared memory for mutual exclusion. Information and Computation, 107(2):171–184, December 1993. Originally appeared in Jim Burns's thesis, around 1981.

3.3. Nancy A. Lynch and Michael J. Fischer. On describing the behavior and implementation of distributed systems. Theoretical Computer Science, 13(1):17–43, 1981. Special issue on Semantics of Concurrent Computation.

3.4. Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch. Efficiency of synchronous versus asynchronous distributed systems. Journal of the ACM, 30(3):449–456, July 1983.

3.5. Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. Information Processing Letters, 14(4):183–186, June 1982.

3.2 Mutual exclusion in shared-memory models

We started by considering the problem of *mutual exclusion* in shared-memory systems. We considered the problem both with *test-and-set* shared memory and with *read/write* shared memory.

Bounds on test-and-set shared memory for mutual exclusion and resource allocation: The first paper I was involved in, in the area of distributed computing theory, was Paper 3.1 [14]. This was based on a shared memory model, in which a collection of N processes operated concurrently and asynchronously with respect to each other, communicating using a small shared memory. The goal was to arbitrate access to critical regions of their code. To do this, the processes executed two protocols, a *trying protocol* to try to gain access to the critical region, and an *exit protocol* to leave the region gracefully.

This paper followed the paradigms of sequential algorithms, although for a very different type of setting. This paper gave formal, mathematical definitions of the mutual exclusion problem, and presented algorithms and nearly-matching lower bounds for this problems. The problems in [14] involved mutual exclusion with a variety of *liveness* and *fairness* guarantees, ranging from simple absence of deadlock, to avoidance of lockout, to bounds on the number of times one process could bypass another.

The paper was somewhat influenced by a series of papers by Dijkstra, Knuth, and others, on clever algorithms for mutual exclusion. However, a difference was that those papers assumed read/write shared memory, whereas our paper used a stronger type of shared memory, which we called *test-and-set* memory. In our version of test-and-set, a process could, in one step, read the value of a single shared variable and make some arbitrary changes to both its local state and the shared variable, all atomically.

For me, an earlier, and stronger influence was a previous unpublished University of Southern California technical report by Profs. Armin Cremers and Tom Hibbard. I learned about this work when I was on the USC faculty together with Cremers and Hibbard, sometime during the years 1973-1976. Their paper used the test-and-set model. It showed that two processes cannot achieve

mutual exclusion with fairness using a shared variable that can take on only two values.

I found this exciting, because it demonstrated that in distributed computing theory, one can hope to prove nontrivial impossibility results. This was quite different from the situation in the theory of sequential algorithms, in which lower bound results were (and still are) very hard to prove. The reason is that distributed algorithms run on very difficult platforms, with many agents, each of which uses only local information. These difficulties mean that problems are much harder to solve. The work of Cremers and Hibbard demonstrated how distributed systems might support interesting lower bounds, which later proved to be a key component of work in this area.

Our Paper 3.1 [14] generalized the earlier work of Cremers and Hibbard to more processes and more types of fairness conditions. It showed that one can prove nontrivial lower bound results for distributed algorithms.

3.1. James E. Burns, Paul Jackson, Nancy A. Lynch, Michael J. Fischer, and Gary L. Peterson. Data requirements for implementation of N-process mutual exclusion using a single shared variable. Journal of the ACM, 29(1):183–205, January 1982.

Briefly, we assumed a single shared variable (this restriction doesn't matter for the powerful test-and-set model), and tried to determine the size of a variable that is sufficient to solve the mutual exclusion problem. It turned out that the answer depends strongly on the type of fairness conditions required.

For instance, a simple 2-valued semaphore is adequate if we do not require any fairness but just absence of deadlock. But if we require fairness to all requesting processes, we need some coordination. Here is where we encountered the characteristic difficulties of distributed computing with limited communication—how could separate processes, operating independently and asynchronously, manage to coordinate for coherent access to their critical regions? It seemed that things could get very confusing and chaotic. The way out that we found in this work was to use an abstraction—a conceptual *Virtual Supervisor* process, which could be emulated by the other processes. The role of the Virtual Supervisor was to offload the coordination information into its own state instead of using the small shared variable, and to manage the coordination via communication with the competing processes using the narrow-bandwidth shared variable. In this way, we might say that this paper introduced the important idea of levels of abstraction for distributed algorithms.

The algorithms of [14] were not particularly practical. But they used some interesting algorithmic strategies, mainly the Virtual Supervisor idea.

The paper also contained nontrivial lower bounds for mutual exclusion with fairness guarantees. These used explicit, intricate constructions of executions that lock out some processes, given too few values of the shared variable. These results were inspired by, and generalized considerably, the 2-process construction of Cremers and Hibbard.

We continued our work with many papers delineating the power of the test-

and-set shared memory model to solve problems related to mutual exclusion. For example, in [49], we extended the model and problem to consider multiple resources and process failures.

Limitations for read/write shared memory: The next paper, Paper 3.2 [15], deviated from the test-and-set model, assuming instead much weaker *read/write* shared memory, in which variables are accessed using only separate read and write operations. The main difficulty with this model is that processes can overwrite each other's updates, which can cause information to be lost. We again studied the mutual exclusion problem, this time without any fairness requirement, just absence of deadlock. For the read/write case, unlike the test-and-set case, it turns out the number of variables matters. So now we considered the number of variables needed, rather than their size.

The results of [15] originally appeared in Jim Burns's Georgia Tech PhD thesis, around 1981. However, for some reason, we did not get around to publishing this until a full 12 years later.

3.2. James E. Burns and Nancy A. Lynch. Bounds on shared memory for mutual exclusion. Information and Computation, 107(2):171–184, December 1993.

The paper contains a simple algorithm using only n single-writer, multi-reader, binary valued shared variables for n processes. This is in contrast to Lamport's Bakery Algorithm, which has unbounded-size variables, and to Dijkstra's mutual exclusion algorithm, which uses multi-writer shared variables.

The more interesting result in the paper (to me) is the lower bound. It says that n processes require at least n shared read/write variables to solve this version of mutual exclusion. No matter how many values the variables may take on, and even if the variables allow multiple writers, we still need at least as many variables as processes!

The proof of this lower bound is by a really clever construction, mainly due to Jim. The key is that it is possible to maneuver a process so that it is poised, "about to write" a shared variable. Then we can force activity of the other processes, affecting that variable, then resume the poised process and overwrite whatever was written by the other processes. This allows us to hide the activity of the other processes. If we can maneuver multiple processes so they are poised, "about to write" several different shared variables, then we can hide the activity of other processes that affect all of those variables.

3.3 Models for distributed systems

Even in our earliest days of working on algorithms and lower bounds for shared-memory distributed systems, we felt the need for new formal models to support the algorithmic work. We were already aware that we wanted to help start a new research field of distributed computing theory, and it seemed clear that the field should have its own general models to provide a foundation.

The field of sequential complexity theory was based on well-established automata-theoretic models, like Turing machines and Random Access Machines (RAMs). Researchers working on sequential algorithms shared a common foundation for their algorithmic work. For synchronous parallel shared-memory computing, researchers used models such as Parallel Random Access Machines (PRAMs). But now we needed something quite different—a model for algorithms that were supposed to run on distributed systems consisting of asynchronously operating, interacting processes. It was not obvious at all what such a model should look like.

We did know that we wanted to base our model on a foundation of set theory and automata theory, rather than on any particular logical language. This was consistent with the situation for sequential algorithms and synchronous parallel algorithms. But it contrasted with a large body of work that was going on at the time on models for concurrent systems, by Hoare, Milner, and other *process algebraicists*. Their work described concurrent processes using formal logical expressions, rather than using automata. Systems could be built up from simpler components using formally-defined operators, composition being the most important one. The operators became part of an algebraic language for describing systems of processes, and algebraic equations were used to assert equivalence of systems built using different algebraic expressions. Essentially all reasoning about the systems was carried out within a formal logical system.

But since we were emphasizing algorithms and complexity, we preferred a different style, based on set theory and automata. We did not want to force the reasoning about the algorithms—their correctness and performance—into any particular logical framework, but rather, wanted to be able to use the full power of mathematics.

General model for asynchronous shared-memory computing: Since we started working on theory for distributed computing by studying shared-memory algorithms, we defined our first version model for systems of asynchronous processes interacting using shared memory. Our initial modeling paper was Paper 3.3 [88]:

3.3. Nancy A. Lynch and Michael J. Fischer. On describing the behavior and implementation of distributed systems. Theoretical Computer Science, 13(1):17–43, 1981. Special issue on Semantics of Concurrent Computation.

The paper is a sort of manifesto. It outlines our goals in terms of creating a general model as the foundation for a new algorithmic theory for distributed systems, and explains our particular design choices. It explains why we need new models for distributed algorithms, different from those for sequential algorithms. We also need new kinds of problem definitions—not just functions as before, but definitions that include ongoing behavior and nondeterminism. We also need new definitions of what it means for an instance of the model to "solve" a particular problem—something that is obvious for sequential algorithms.

Section 2 of the paper contains the formal definitions for our automata-theoretic shared-memory model. It also introduces a *composition operator* for systems, which is intended to be useful in describing systems in terms of more primitive systems. Section 3 describes how to define a *problem* to be solved by an automaton, as a set of (finite and/or infinite) sequences of steps involving shared variables. Section 4 describes what it means for a distributed algorithm, expressed as an automaton within our model, to *solve* a problem, also expressed within our model. Specifically, the set of sequences comprising the behavior of the algorithm can be any subset of the set of sequences defining the problem.

The paper continues by defining a useful measure of *time complexity* for asynchronous distributed systems, based on assuming real-time upper bounds on the time between basic events such as processes accessing shared variables. In contrast to the usual handling for sequential algorithms, we do not consider the costs of local computation, which are generally regarded as insignificant for distributed algorithms compared to the costs of interaction. The paper goes on to present examples illustrating how very different distributed algorithms can be used to solve the same problem—here, the problem of fair mutual exclusion. It includes analysis of the time complexity of the three algorithms, and a comparison.

As it turned out, the particular model of this paper did not end up seeing widespread use as a foundation for the field. However, it was a direct precursor for the more impactful *Input/Output Automata* model of Lynch and Tuttle [101, 102]. Note that the I/O Automata work includes treatment of levels of abstraction. Levels of abstraction have turned out to be another important way of decomposing distributed algorithms, along with composition, but they do not appear in [88]. I discuss I/O Automata in Section 4.6.

Synchronous vs. asynchronous shared memory systems: While we were working on our algorithms and models for shared-memory distributed computing, Prof. Eshrat Arjomandi from York University visited us at Georgia Tech. She had been working on parallel algorithms for solving graph problems, using PRAM models. Like the models we were considering, the PRAM model is a shared-memory parallel model. However, unlike in our model, its processes operate in synchronous rounds. See, for example, [6].

This led us to explore the difference in computing power, specifically, in time efficiency, between synchronous shared-memory models in which the processes operate in "rounds", and asynchronous shared-memory models in which processes operate at arbitrary speeds relative to each other. This eventually led to Paper 3.4 [7]. This paper contains the perhaps-surprising result that there are some problems that can be solved much faster with a synchronous parallel shared-memory algorithm than with *any* asynchronous shared-memory algorithm. This is not a comparison of the behavior of a particular synchronous algorithm with a particular asynchronous algorithm—rather, it is a result about *all possible* asynchronous algorithms in our model.

3.4. Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch. Efficiency of synchronous versus asynchronous distributed systems. Journal of the ACM, 30(3):449–456, July 1983.

The problem we focused on was a simple abstract problem that we called the "s-session problem". In this problem, all of the processes should cooperate to perform a number s of "sessions", in each of which each process must output at least one signal. Then all processes must halt, producing no further outputs. It is easy to devise a synchronous algorithm that performs s sessions in s rounds, i.e., in time s. However, we show that any asynchronous algorithm that is guaranteed to produce s sessions and then halt must take at least approximately $s \log n$ time, where n is the number of processes. Here time is measured according to the asynchronous time measure described in Paper 3.3 [88].

To prove the lower bound for asynchronous algorithms, we assumed an asynchronous algorithm $\mathcal{A}$ that worked in less time than what we wanted to show. We started with a synchronous execution of $\mathcal{A}$. Then we reordered the steps of the execution, while maintaining the dependencies of the algorithm and reducing the number of sessions to below s. The reordered execution is still an execution of the algorithm, but it fails to solve the s-session problem. This yielded a contradiction.

The significance of this paper was that it removes the following hope: we might like to design asynchronous distributed algorithms by first designing synchronous versions and then transforming them systematically to run in asynchronous systems, while preserving correctness. This result says that such an approach would increase the worst-case running time significantly.

3.4 Other early results

Resource allocation in networks: With Nancy Griffeth, we studied more elaborate resource-allocation problems [89, 46]. We assumed an arbitrary placement of resources at the nodes of a graph network. The problem was to service requests that arrive at random nodes in the network, matching requests to resources. The model used in this work was a network message-passing model rather than a shared-memory model.

The first paper [89] presented request-resource matching algorithms and analyzed their time requirements. We assumed here that requests arrive on-line, and may overlap: new requests may arrive before previous requests have been matched to their resources. The worst case for this algorithm turned out to be when the requests arrive sequentially, because some optimization is possible when concurrent requests interact. The second paper [46] assumed that the requests arrive at random locations, all at once. Here we measured the sum of the lengths of paths connecting requests to their matched resources.

Global states: We also wrote a paper describing a new *global snapshot* algorithm for distributed systems [47]. This paper was inspired by work on distributed database concurrency control. We assumed a system of "ordinary

transactions", which could execute in an interleaved manner, and added to this system a new "snapshot transaction", which was required to appear atomic with respect to the ordinary transactions. The snapshot transaction was required to return a "consistent" global state of the system—one that "could have happened" as a result of running all the ordinary transactions that preceded the snapshot to completion, along with some subset of the ordinary transactions that were concurrent with the snapshot. In this sense, the algorithm produced a *consistent global snapshot*.

A special case of this snapshot algorithm is one in which the "ordinary transactions" correspond to transfers of money from one banking location to another. The snapshot transaction can then be used to calculate the total money at all of the banking locations.

3.5 Distributed consensus

Much of our best-known early work was on the topic of distributed consensus. In this problem, processes receive inputs, which may differ, and must agree on a common output. This must work even if some number (at most f) of processes are faulty. In some versions of the problem, they might just stop. Or they might exhibit worse, *Byzantine* failures, where they act "maliciously", for example, providing inconsistent information to different other processes. This research direction was initiated by earlier work of Leslie Lamport with Marshall Pease and Robert Shostak [115, 77], on agreement for altimeter readings on aircraft, where one of the altimeters might fail.

Leslie visited Mike Fischer and myself at Georgia Tech, for about a week during Mike's sabbatical. In preparation for Leslie's visit, Mike and I read a number of Leslie's papers, including a preliminary (unpublished) manuscript entitled "The Albanian Generals Problem" (the problem was later renamed to the "Byzantine Generals Problem"). The paper considered a synchronous model in which the processes acted in rounds. Communication was by sending and receiving messages. Processes were totally connected for communication.

In reading these papers, we noted that all of the algorithms in the papers used exactly $f + 1$ rounds to reach agreement tolerating f faulty processes. We wondered whether this many rounds were necessary. We tried to find faster algorithms, but failed. Then just before Lamport's visit, we proved a lower bound of $f + 1$ rounds. This result appears in the short Paper 3.5 [48]

3.5. Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. Information Processing Letters, 14(4):183–186, June 1982.

This proof used a *chain argument*. We constructed a chain of executions spanning between two extreme executions in which the decisions for the nonfaulty processes are required to be 0 and 1, respectively. In this case, we considered an execution with all inputs of 0 and no failures, vs. an execution with all inputs of 1 and no failures. Then we constructed the chain, one "link" at

a time, so that at each link between two executions, there is some nonfaulty process that can't distinguish between the two executions. This implies that this process, and hence all the nonfaulty processes, must decide in the same way in both executions. It follows that there can be no particular point in the chain where the decision changes, which yields the contradiction.

Another version of this proof appears in my Distributed Algorithms textbook [86]; That version proves a stronger result, for stopping failures, not just very strong Byzantine failures.

Chain arguments have been used subsequently to prove other lower bound results, including the "FLP" impossibility result [52], which I discuss in Section 4.2. Also, chain arguments have been extended to multiple dimensions, using topological machinery. Such arguments appear in the Godel-prize winning work on the topological structure of asynchronous computability [64], as well as our lower bound for synchronous solutions to k-consensus [19], which I discuss in Section 5.

Thus, Paper 3.5 [48] marked the beginning of our work on consensus, a topic that we continued to study for quite a few years.

In these early papers, we have already seen several lower bounds and other impossibility results. Impossibility results have turned out to be a major research direction within distributed computing theory, with hundreds of such results being proved. We have seen that the main reason why lower bounds and other impossibility results can be proved here is the strong limitations imposed by locality in distributed systems. Each process can see only its own state, the values it reads in shared memory, the messages it receives, etc. The lack of global knowledge is a very strong restriction that enables proof of impossibility results.

We will see more lower bounds in this paper, and others are discussed in the papers [82] and [39].

4 Early Work at MIT: Distributed Consensus, Models, and Atomic Transactions, 1981-1990

I went to MIT for a sabbatical year in 1981-1982, visiting Prof. Barbara Liskov's research group in distributed systems. This was because of my interest in distributed database concurrency control: Barbara's group was then working on a system called Argus, which implemented a programming model based on nested atomic transactions. Transactions are sequences of operations on data objects that are supposed to look "as if" they occurred "atomically", that is, consecutively; however, in their implementations, the operations are generally interleaved. Nested transactions generalize the traditional transaction model by allowing transactions to have subtransactions, and so on. Many interesting algorithms had been developed in the distributed systems community for im-

plementing transactions in distributed systems, including algorithms based on locking, time-stamping, and hybrids of these; optimistic algorithms, and algorithms that used data replication. All could be extended to allow nesting. I was interested in working on theoretical underpinnings for such systems.

I worked with Barbara, and with her group members Bill Weihl and Maurice Herlihy, to model and verify some popular algorithms for distributed database concurrency control, including algorithms based on locking, timestamping, optimistic concurrency control, and replication. This work involved generalizing many of the standard concurrency control algorithms to the case where transactions could be nested; Barbara had already done that for locking, but there was more to do for other algorithms. In addition to algorithm design, this turned into an exercise in formal modeling and verification of complex distributed algorithms with strong correctness requirements.

In addition, I continued work on distributed consensus and consensus-related problems, mainly with Mike Fischer. This work culminated at the end of the year with the famous "FLP" result on impossibility of consensus.

Another interesting event during this sabbatical year was the start of the Principles of Distributed Computing (PODC) Conference. The first of these conferences was held in Ottawa in the summer of 1982, and the conference has been going strong since then. As I recall, I played some role in starting up this conference, while the main organizers were Mike Fischer, Robert Probert, and Nicola Santoro.

At the end of the sabbatical year, I accepted an offer of a tenured Associate Professorship at MIT, where I have remained for the 40+ years since then. Along with the offer, I was given a five-year chair, the first "Ellen Swallow Richards Chair", funded by the MIT Alumnae with the purpose of bringing senior women faculty to MIT. At that time, there were very few.

For the next few years, I built my research group and worked on a combination of consensus algorithms and lower bounds, and nested transaction algorithms and system modeling. The work on nested transactions contributed, in turn, to the development of the Input/Output automaton model for asynchronous systems communicating via shared actions.

Although this was all theoretical work, I remained for these years in the distributed systems group at MIT, since distributed computing was still not a popular theoretical discipline.

4.1 Key publications

4.1. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382, April 1985.

4.2. Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. Journal of the ACM, 33(3):499–516, July 1986.

4.3. Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization. Information and Control, 62(2-3):190–204, August/September 1984.

4.4. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. Journal of the ACM, 35(2):288–323, April 1988.

4.5. Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. Distributed Computing, 1(1):26–39, January 1986.

4.6. Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, pages 137–151, Vancouver, British Columbia, Canada, August 1987.]

4.7. Nancy Lynch and Mark Tuttle. An introduction to Input/Output Automata. CWI-Quarterly, 2(3):219–246, September 1989. Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

4.8. Nancy A. Lynch, Michael Merritt, William E. Weihl, and Alan D. Fekete. Atomic Transactions. Morgan Kaufmann series in data management systems. Morgan Kaufmann, 1993.

4.2 FLP

Paper 4.1 [52] is, by far, my most cited paper. We obtained the main result in late summer of 1982, at the end of my sabbatical year and just before I started my real job on the faculty at MIT. We first published the result in a Principles of Database Systems conference in 1983 [51]. The final journal version appeared in 1985 [52]. The result is generally known as FLP, after the authors.

The result says that it is impossible to reliably reach agreement in an asynchronous network, with the possibility of even a single stopping failure.

4.1. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382, April 1985.

As I recall, I thought of this, at first, as a purely theoretical problem. Having matching upper and lower bounds for the number of rounds needed for consensus in synchronous distributed systems [48], it seemed natural to consider what happens in the asynchronous case. I did not know how this would turn out ahead of time; I worked both on trying to find an algorithm and trying to prove an impossibility result. Some sense that this might be solvable arose from our early thoughts on the *approximate agreement problem*, discussed in Section 4.3,

for which natural synchronous algorithms had straightforward extensions to the asynchronous setting.

Mike Fischer and I met at MIT during the summer of 1982 and discussed this problem. Mike said that he had also been thinking about this problem, motivated by some distributed database systems issues suggested to him by Butler Lampson during a visit by Mike to Xerox PARC. Mike Paterson joined the collaboration when he visited Mike Fischer at Yale sometime after our MIT meeting. We worked on both trying to find an algorithm and trying to prove an impossibility result, and ended up proving our impossibility result.

The FLP result was stated in terms of simple stopping failures, and required only a single failure! The proof was quite elementary, but certainly not obvious. In fact, as I recall, some leading researchers couldn't quite believe it was true and had to review it several times. The proof was based on assuming that a solution exists, and then characterizing how the decision could be made in such a solution. It turns out that the decision can be localized to what happens at a single location in the network, based simply on the order of arrival of two different messages at one node. But if the node in question fails, the rest of the system cannot distinguish the order in which the messages arrived, so cannot decide differently in the two cases.

As it turned out, the result was widely appreciated in both the distributed computing theory and distributed systems communities. Certainly it was an interesting, surprising theoretical result. But also, it had significance for the practical distributed systems community. Namely, the problem was closely related to the distributed transaction commit problem, where the processes involved in processing a transaction must agree on whether the transaction should commit or abort. Systems researchers were trying to design algorithms to solve the commit problem, and were making informal claims about what their algorithms guaranteed. Our impossibility result seemed to contradict what they wanted to claim.

I admit to being surprised by how much attention this result got, in the distributed systems community. Though I was quite pleased with the result and proof as theory.

Our paper was awarded the second Dijkstra Prize (after Lamport's famous "Time, Clocks,..." paper). Here is a citation for that award, written by Jennifer Welch and Nir Shavit:

> The result of this paper (commonly known as FLP) is that, surprisingly, it is impossible for a set of processors in an asynchronous distributed system to agree on a binary value, even if only a single processor is subject to an unannounced crash. Although the result was motivated by the problem of committing transactions in distributed database systems, the proof is sufficiently general that it directly implies the impossibility of a number of related problems, including consensus.
>
> This result has had a monumental impact in distributed computing, both theory and practice. Systems designers were motivated to clar-

ify their claims concerning under what circumstances the systems work.

On the theory side, people have attempted to get around the impossibility result by changing the system assumptions or the problem statement. Work on changing the system assumptions includes the study of partially synchronous models and of various kinds of failure detectors. Modified problem statements include randomized algorithms, approximate agreement, k-set agreement, and condition-based approaches.

The proof technique used in FLP, valency arguments, has been used and adapted to show many other impossibility and lower bound results in distributed computing. These include impossibility results for consensus, k-set consensus, and renaming in various models, and lower bounds on contention and on the number of rounds for synchronous consensus.

The FLP result forms the basis of work on the wait-free hierarchy, in which data types are classified and compared according to the maximum number of processes for which they can solve wait-free consensus. The calculation of consensus numbers relies on valency arguments.

Finally, work on applying ideas from topology to fault-tolerant distributed computing were inspired by the posing of the k-set consensus problem, which in turn was inspired by the FLP result.

4.3 More results related to consensus

During the sabbatical year 1981-1982 and the next few years at MIT, we continued working on problems related to distributed consensus. There were many questions to consider, and this general topic became a popular direction for theoretical computer science research during that time. My collaborators during this time were my own group members, and others including Mike Fischer, Bill Weihl, and Jim Burns.

Low-communication consensus: One of our first results involved improving the amount of communication needed for consensus in synchronous systems. The papers by Lamport et al. [115, 77] gave algorithms for Byzantine agreement that used an exponential amount of communication—the number of bits exchanged was exponential in the number of faulty processes that were tolerated. Dolev and Strong [32] obtained a polynomial-communication algorithm, and we improved this further, obtaining an algorithm that required only $O(nt + t^3 \log t)$ bits [29].

Approximate agreement: Another consensus-related problem that we considered was that of *approximate agreement*, which we introduced in Paper 4.2

[31].

4.2. Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. Journal of the ACM, 33(3):499–516, July 1986.

This problem is a variant of the Byzantine Generals problem in which processes start with arbitrary real values rather than values from a discrete domain, and in which they must agree approximately rather than exactly. We devised algorithms to reach approximate agreement in both synchronous and asynchronous systems. The asynchronous agreement algorithm is especially interesting because it contrasts with the FLP impossibility result, which says that exact agreement with guaranteed termination is not attainable in an asynchronous system with even one faulty process. It turns out that the situation is quite different if we require only approximate agreement.

The algorithms operate in rounds; for the asynchronous case, this required defining a new fault-tolerant notion of asynchronous rounds. The algorithms work by successive approximation. At each round, they use a *fault-tolerant averaging function*—one that discards a number of extreme values corresponding to the number of process failures to be tolerated. We proved a convergence rate that depends on the ratio between the number f of faulty processes and the total number n of processes. We also proved lower bounds on the convergence rate, which imply that our algorithms are optimal.

Clock synchronization: We also studied the problem of *distributed clock synchronization*. This used a model that is somewhere between synchronous and asynchronous. Namely, processes have individual real-valued "clocks" that they can read and use in determining their behavior. The clocks of different processes may differ slightly from each other, and they may run at rates that differ slightly from that of real time. Therefore, they may drift apart. Other complications include (possibly Byzantine) faulty processes and variations in communication time. It is desirable to keep the clocks synchronized, as closely as possible, by making small adjustments or using them to implement approximately-synchronized *logical clocks*.

As I recall, the approximate agreement problem that we introduced in [31] was originally inspired by the clock synchronization problem, since it may be viewed (roughly speaking) as a simpler special case.

A collection of fault-tolerant clock synchronization algorithms appeared in [76, 107, 62]. My PhD student Jennifer Lundelius (Welch) and I contributed to this direction [125] with a new clock synchronization algorithm that was directly inspired by our asynchronous approximate agreement algorithm of [31]. As in that algorithm, the new clock synchronization algorithm proceeds in rounds, adjusting the clocks using a fault-tolerant averaging function.

In addition, in Paper 4.3 [80], Jennifer and I proved matching upper and lower bound results for a simple special case of the problem, in which clocks

do not drift and there are no failures. However, the clocks at different processes may have different initial values, and the communication delay between the processes is uncertain. In terms of the number n of processes and a bound ϵ on the uncertainty in communication delay, we obtained matching upper and lower bounds of $O(\epsilon(1 - 1/n))$ for the achievable closeness of synchronization. The basic technique involves constructing alternative executions by shifting the times of events at different processes.

4.3. Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization. Information and Control, 62(2-3):190–204, August/September 1984.

Finally, in another interesting paper on fault-tolerant clock synchronization around the same time, Dolev, Halpern, and Strong [30] proved impossibility of Byzantine-fault-tolerant clock synchronization for $3f$ processes and f failures. We noted that the Byzantine agreement algorithms of Lamport et al. also used $3f + 1$ processes, and the authors proved that this was necessary. These results suggest that there might be something inherent about needing $3f + 1$ processes to tolerate f Byzantine failures, in general. I will have more to say about this in Section 4.5.

The Byzantine Firing Squad problem: Jim Burns and I invented a new synchronization problem, called the *Byzantine Firing Squad* problem [117], and developed a new algorithm to solve it. The problem assumes that processes operate in synchronous rounds but do not have a common start time. We assumed that one or more nonfaulty processes receive an external START signal at some point, and all the nonfaulty processes are supposed to fire at some later round. Moreover, if any nonfaulty process fires in some round, then all the nonfaulty processes must fire in that same round. As for the previous Byzantine agreement and Byzantine clock synchronization algorithms, our Byzantine Firing Squad algorithm also uses $3f + 1$ processes to tolerate f Byzantine failures. This was not surprising, because our algorithm uses a standard Byzantine agreement algorithm to agree on a firing preference at every round.

4.4 Consensus with partial synchrony

The strong impossibility result in the FLP paper was worrisome, because the consensus problem is very important to solve in practice, including in situations in which failures might occur. We (and many others) considered how to get around the problem. Our solution appears in Paper 4.4 [38]. It involves using notions of *partial synchrony*, which are between pure synchrony and pure asynchrony.

4.4. Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. Journal of the ACM, 35(2):288–323, April 1988.

Our paper covers several types of failures and notions of partial synchrony. The simplest case involves simple stopping failures and a notion of partial synchrony in which message delays are bounded after some *Global Stabilization Time (GST)*. The algorithm for this case requires a majority of nonfaulty processes.

The first key idea in the paper was to consider the safety requirements (agreement and validity) as separate from, and higher priority than, the termination requirement. The safety requirement should never be permitted to be violated, regardless of the occurrence of failures or timing anomalies. However, the termination requirement might be permitted to depend on stability in the system's behavior, with respect to timing and failures. This seemed reasonable, for use in practice.

The second key idea was to have the algorithm make multiple attempts to achieve consensus, where each attempt involved a protocol, led by a coordinator, to gather enough votes for agreement on a particular value. If the system stabilized for long enough, then an attempt initiated during the stable period would be guaranteed to complete.

The danger of this approach is that multiple attempts could lead to contradictory decisions by different processes. This motivated the third key idea, which was a mechanism designed to keep the results of different partial attempts consistent. For this, we used a protocol in which processes can "lock" proposed values. Each lock is associated with a particular consensus attempt. A process can lock a value for a certain attempt when it learns that the coordinator of that attempt has proposed that value for a decision value. A process can release a lock when it hears about a lock for a different value for a later attempt.

To determine a value to propose, the coordinator of an attempt gathers information from a majority of processes. The requirement is that each of the processes in the majority must deem the value to be "acceptable", meaning that it doesn't have a lock on any different value. Once the coordinator determines the value to propose, it sends messages to all processes, requesting that they lock this value for this attempt. If it hears that a majority have done this, the coordinator can decide on this value.

This paper was a precursor of the well-known Paxos algorithm [75]. Paxos used the same basic idea for consensus, but incorporated this consensus protocol into a larger protocol for implementing an ongoing replicated state machine; basically, Paxos achieves consensus on each successive update. Also, Paxos was designed to tolerate more concurrency in the attempts—our attempts were sequential—but the same sort of consistency mechanism still works. Paxos is also designed to tolerate more practical types of failures, such as crashes that obliterate volatile memory. Variants of Paxos have been engineered and widely used in practice.

The significance of the work in [38] was well recognized in both the theoretical and practical distributed systems communities. We designed this as a theoretical result, but it turned out to have considerable significance in practice. This paper was awarded the 2007 Dijkstra Prize. Here is the citation (which I think was written by Dahlia Malkhi):

This paper introduces a number of practically motivated partial synchrony models that lie between the completely synchronous and the completely asynchronous models, and in which consensus is solvable. It gave practitioners the right tool for building fault tolerant systems, and contributed to the understanding that safety can be maintained at all times, despite the impossibility of consensus and progress is facilitated during periods of stability. These are the pillars on which every fault tolerant system has been built for two decades. This includes academic projects such as Petal, Frangipani, and Boxwood, as well as real life data centers, such as the Google file system.

In distributed systems, balancing the pragmatics of building software that works against the need for rigor is particularly difficult because of impossibility results such as the FLP theorem. The publication by Dwork, Lynch, and Stockmeyer was in many respects the first to suggest a path through this thicket, and has been enormously influential. It presents consensus algorithms for a number of partial synchrony models with different timing requirements and failure assumptions: crash, authenticated Byzantine, and Byzantine failures. It also proves tight lower bounds on the resilience of such algorithms.

The eventual synchrony approach introduced in this paper is used to model algorithms that provide safety at all times, even in completely asynchronous runs, and guarantee liveness once the system stabilizes. This has since been established as the leading approach for circumventing the FLP impossibility result and solving asynchronous consensus, atomic broadcast, and state-machine replication.

In particular, the distributed systems engineering community has been increasingly drawn towards systems architectures that reflect the basic split between safety and liveness cited above. Dwork, Lynch, and Stockmeyer thus planted the seed for a profound rethinking of the ways that we should build, and reason about, this class of systems. Following this direction are many foundational solutions. First, these include state machine replication methods such as Lamport's seminal Paxos algorithm and many group communication methods. Another important branch of research that directly follows this work is given by Chandra and Toueg's unreliable failure detector abstraction, which is realized in the eventual synchrony model of this paper. As Chandra and Toueg write: "we argue that partial synchrony assumptions can be encapsulated in the unreliability of failure detectors. For example, in the models of partial synchrony considered in Dwork et al. it is easy to implement a failure detector that satisfies the properties of Diamond-W." Finally, the insight by Dwork, Lynch, and Stockmeyer also led to various timed-based models of partial synchrony, such as Cristian

and Fetzer's Timed-Asynchronous model and others.

4.5 Easy impossibility proofs:

We (and others) observed the common $3f + 1$ processes bound for Byzantine fault-tolerant algorithms, including Byzantine agreement, Byzantine clock synchronization, and the Byzantine Firing Squad problem. All the known algorithms for these problems used $3f + 1$ processes to tolerate f faults. Earlier papers [77] and [30] included proofs that $3f + 1$ is a lower bound, for Byzantine agreement and clock synchronization, respectively. These were special-case proofs, for particular problems.

Mike Fischer, Michael Merritt, and I thought that there must be some common reason that $3f + 1$ was necessary for accomplishing anything interesting with Byzantine faults. In Paper 4.5 [50], we described a systematic way to prove such lower bounds, and applied it to five problems: Byzantine agreement, weak Byzantine agreement [74], approximate agreement, clock synchronization, and the Byzantine Firing Squad problem. I think that our approach yielded some unifying insight.

4.5. Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. Distributed Computing, 1(1):26–39, January 1986.

The key idea of the approach can be expressed nicely at a high level. Consider the special case of showing that three processes cannot solve Byzantine agreement if one of the processes might be faulty. Assume for contradiction that such a system of three processes, $\mathcal{A}$, exists. Now construct another system $\mathcal{S}$, this one of six processes, which is essentially two copies of the 3-process system $\mathcal{A}$. In $\mathcal{S}$, start half the processes with input 0 and the other half with input 1. Then, under some reasonable assumptions, it turns out that we can deduce behaviors for the 6-process system $\mathcal{S}$ by using the correctness requirements of the 3-process system $\mathcal{A}$. This depends on the fact that processes in $\mathcal{S}$ cannot distinguish themselves from their counterparts in $\mathcal{A}$, provided that a Byzantine faulty process in $\mathcal{A}$ pretends to be a string of four processes in $\mathcal{S}$. Deducing enough behaviors for the 6-process system $\mathcal{S}$ yields a contradiction.

As usual, the proof depends on the limitations of local knowledge in a distributed system. The proof is expressed generally and abstractly, not dependent on precise details of the model.

Very similar proofs hold for weak Byzantine agreement, approximate agreement, Byzantine clock synchronization, and Byzantine Firing Squad. In fact, this paper essentially provides a general approach to proving such lower bounds. In addition to all of these results about the number of required processes, the paper also shows general limitations of $2f + 1$ on network connectivity.

By 1986, distributed computing theory had become an established field of theoretical study, and impossibility results had become a main characteristic of the

new field. At some point there were so many, that I felt compelled to write a paper summarizing all of those that I could find at the time [82].

4.6 Models

In parallel with working on consensus problems, I did a deep dive into the theory of distributed database algorithms, based on *atomic transactions*. Barbara Liskov introduced me to *nested transactions*, a generalization of the traditional transaction model that allows transactions to have subtransactions which were atomic with respect to each other, and so on. Many interesting algorithms were developed for implementing distributed transactions, and all could be extended to the nested case.

I will return to nested transactions in Section 4.7. Here, I mention them as one of my motivations for developing new formal models for distributed systems. Other motivating examples included projects on modeling resource allocation in networks [101], synchronizers [44], distributed minimum spanning tree algorithms [124], communication channels [92], shared atomic objects [11], and dataflow systems [100].

What we needed for all of this work was a new model for asynchronous distributed systems, in which processes communicate with each other, not using shared memory as in [88], but using abstract input and output actions. A shared action model seemed better than a shared memory model for network-style examples like those listed above. The result was the Input/Output Automata mathematical framework. We presented this in two papers, Papers 4.6 [101] and 4.7 [102].

4.6. Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, pages 137–151, Vancouver, British Columbia, Canada, August 1987.

4.7. Nancy Lynch and Mark Tuttle. An introduction to Input/Output Automata. CWI-Quarterly, 2(3):219–246, September 1989. Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

The papers give definitions for individual I/O Automata, including a treatment of *fair executions* that capture the idea that separate automata, or even *tasks* within automata, continue taking steps regardless of what other automata do. This notion is essential for a treatment of concurrently executing processes, but did not appear in previous models for concurrent systems.

I/O Automata include distinguished *input and output actions*. I/O Automata are *input-enabled*, which means that any input action may occur from any automaton state, in other words, an automaton cannot block its inputs. In contrast, in the process-algebraic models that were popular at the time, actions are not classified as input vs. output, which meant that actions that are shared by different automata could cause low-level synchronization delays. Such

delays would not be convenient in a model that is intended to be used for analyzing the time complexity of algorithms. The distinction between input and output actions, together with the input-enabling property, remove the problem of synchronization delays.

The papers [101] and [102] also define *problems* to be solved by I/O Automata; a problem is modeled as a set of sequences of input and output actions. The papers also define what it means for an I/O automaton to *solve* a problem: an automaton solves a problem if the set of its external behaviors is a subset of the set defined by the problem. The paper also define *composition* for I/O Automata and showed that it respects external behavior.

A main emphasis in this work was the use of *levels of abstraction* for presenting and verifying distributed algorithms. A distributed algorithm might be modeled abstractly and proved correct in terms of the abstract model. Then more detailed, lower-level versions of the algorithm could be given, and shown to be correct by mapping them formally to the abstract model, in a way that preserves external behavior. So algorithm correctness carries over from the high-level algorithm to the lower-level version. This idea has turned out to be crucial as a way of understanding and proving correctness of distributed algorithms.

As an example to illustrate levels of abstraction, in [101], we described a simple "arbiter" algorithm that fairly allocates a single unshareable resource among processes located at the nodes of an undirected acyclic graph. We first presented the algorithm at a high level, in terms of request tokens and a resource token moving around the graph. This would be the way that someone might explain the operation of the algorithm on the blackboard, except that we gave a formal version. Then we gave a lower-level algorithm in terms of actual processes and communication channels. We proved that the higher-level algorithm fairly allocates the resource, and that the lower-level algorithm implements the higher-level algorithm, in a formal sense, and so also fairly allocates the resource.

Another paper [100] demonstrates how an important principle studied in some other models of concurrency can be expressed and proved using I/O Automata. We defined a subclass of I/O Automata called *determinate*, which means just that its input/output relation is a function. We showed that determinate I/O Automata compute continuous functions; moreover, the function associated with a composition of determinate automata is also continuous, and can be characterized as the least fixed-point of a certain continuous functional associated with the network. This latter result was known as *Kahn's Principle* in the concurrency theory area. Although the result was already known, our contribution here lies in the fact that the I/O Automata model can express the result easily and yields extremely simple proofs.

The I/O Automata model has been used fairly extensively to describe different types of asynchronous distributed algorithms. For example, my books *Atomic Transactions* [84] and *Distributed Algorithms* [86] use the I/O Automata model as the foundation for presenting all of the asynchronous distributed algorithms contained therein. Herlihy used (a slightly simplified version of) I/O Automata as the foundation for his theory for wait-free synchronization [63], and Herlihy and Shavit used them for their well-known work on the topological

structure of asynchronous computability [64]. Chockler, Keidar and Vitenberg used I/O Automata as a unifying framework to describe and unify many specifications that were developed in the group communications research area [22]. Manadhata and Wing used them as the basis for developing metrics for software security [106]. Abraham et al. used them to explain the Byzantine disk Paxos algorithm [3]. Doherty at al. used them to specify transactional memory [28]. And so on.

4.7 Nested transactions

I have already mentioned our work on nested transactions, several times. This was a major effort to understand, in rigorous terms, a collection of algorithms for implementing the nested transactions programming framework for distributed databases. My collaborators in this work included Michael Merritt, Bill Weihl, Alan Fekete, Maurice Herlihy, James Aspnes, and my PhD student Ken Goldman.

Our work was originally inspired by the Argus system designed by Barbara Liskov and her research group. Argus used a simple distributed locking algorithm, but we also studied algorithms based on timestamps, algorithms that were hybrids of locking and timestamps, optimistic algorithms, and algorithms that used replicated data. Some of these algorithms had been designed for ordinary single-level transactions, but could be extended to allow multiple levels of nesting.

When we began looking at this area, there was little supporting theory; we were interested in developing such a theory. We developed the formal foundations for the area. We modeled the requirements of transaction systems, and the many different distributed algorithms that meet these requirements. We gave complete correctness proofs. Overall, we produced a large body of work represented in many papers; I won't go into these results in any detail here. This work culminated later in Publication 4.8, the book [84]. The work all rests on the I/O Automata framework.

4.8. Nancy A. Lynch, Michael Merritt, William E. Weihl, and Alan D. Fekete. Atomic Transactions. Morgan Kaufmann series in data management systems. Morgan Kaufmann, 1993.

So far, I have described our early work on building a theory for distributed systems, including basic work on theoretical algorithms and impossibility results, formal models for distributed systems, and applications such as nested transactions. This work contributed to establishing the theory of distributed systems as a research field.

Our main contributions up to 1990 included developing formal foundations for describing distributed algorithms, and using them to describe and prove properties of many theoretical distributed algorithms and practical distributed systems. We also proved many new impossibility results. In doing this, we demonstrated that impossibility results are characteristic of the field of dis-

tributed algorithms, because of the very strong limitations of local knowledge. We also designed a few interesting algorithms, most notably the Dwork, Lynch, Stockmeyer algorithm for consensus with partial synchrony.

By 1990, distributed computing theory was an active research field. It included much interesting theory, but also connections with various kinds of distributed systems. The Principles of Distributed Computing (PODC) conference was well established and active.

Around the year 1990, I moved from the Distributed Systems Group at MIT to the Theoretical Computer Science Group; our work had become more theoretical during the previous nine years, diverging from the distributed systems engineering work. Meanwhile, distributed computing theory had become a recognized discipline within theoretical computer science. We continued after 1990 with theoretical work on algorithms and impossibility results, and modeling and verification. Later, we moved to consider different types of distributed systems: wireless networks and mobile systems, and most recently, biological systems.

In the rest of this paper, I will describe some of our work since 1990. This includes additional algorithms and impossibility results in Section 5, and work on formal models of distributed systems in Section 6. This new work considered not just synchronous and asynchronous systems, but other types of systems, such as timed systems, hybrid continuous-discrete systems, which one might use for distributed software that interacts with an environment, and probabilistic systems. I will also mention, in Sections 7 and 8, some recent work on wireless network algorithms and biological distributed algorithms.

5 Later Work: Algorithms and Impossibility Results, 1990-2005

During the period 1990-2005, I worked with my students and other collaborators on many algorithms and lower bounds for problems involving communication, consensus, and data consistency. Timing aspects of algorithms were a major focus. I have selected a few papers and tried to identify some important themes.

Also during this period, I wrote my "Distributed Algorithms" textbook [86]. It summarized what I thought were the most important ideas of distributed computing theory up to that point.

5.1 Key publications

5.1. Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. Journal of the ACM, 41(1):122–152, January 1994.

5.2. Soma Chaudhuri, Maurice Herlihy, Nancy A. Lynch, and Mark R. Tuttle. Tight Bounds for k-Set Agreement. Journal of the ACM, pages 47(5):912-943, September, 2000.

5.3. Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):48-51, June 2002.

5.4. Nancy A. Lynch. Distributed Algorithms. Morgan Kaufmann, 1996.

5.2 Communication

Alan Fekete, Yishay Mansour, and I explored capabilities for communication between two processes on a single channel, as in the *data link* communication layer in the OSI stack of communication layers. We found two results, both giving inherent limitations for this layer of communication. One showed impossibility of reliable communication if the physical channels could reorder messages—unless we added some extra mechanism such as sequence numbers. The other showed impossibility of reliable communication in the presence of processor crashes.

Both results appeared in preliminary form in [92]. Later journal versions involved other collaborators John Spinelli, Yehuda Afek, Hagit Attiya, Michael Fischer, Da-Wei Wang, and Lenore Zuck, and appeared in [43] and [5]. Again, I/O Automata were used as the underlying model.

5.3 Timing aspects of algorithms

An important focus of our algorithmic work during these years involved timing aspects of distributed algorithms.

Time cost of achieving wait-freedom: In [10], we considered the time cost of requiring fault-tolerance in solving the problem of approximate agreement. This paper recasts the approximate agreement problem from [31] in terms of read/write shared-memory computation. The paper contains a collection of results articulating the costs of requiring the strong *wait-free* fault-tolerance property, which says that a process that keeps taking steps is guaranteed to eventually terminate, regardless of the speed or failure of other processes.

For example, Theorem 7.3 in [10] says that any wait-free algorithm for the n-process approximate agreement problem has time complexity at least $\log n$. The proof considers an execution in which all processes start with 0, which results in a decision of 0 by some process p_i. If the schedule is too short, then there is not enough time for all the processes to "influence" p_i's decision, so there is some other process p_j that does not influence p_i. Then we construct an alternative execution in which p_j starts with 1 and runs on its own, before anyone else begins. By the wait-free requirement, p_j must decide on its own, and must decide 1. Now running all the other processes as before leads to contradictory decisions.

Mutual exclusion using timing assumptions: In [99], we considered what happens to the number-of-register and time costs of solving the mutual exclusion problem in a shared read/write memory model, when we strengthen the

asynchronous model to include some assumptions about timing, specifically, upper and lower bounds on step times. For this model, we devised an algorithm that guarantees mutual exclusion and deadlock-freedom, using only two shared read/write registers. This circumvents the lower bound result of [15], using timing assumptions. Its time cost is also low, depending on the parameters describing the timing uncertainty. The algorithm guarantees its safety property (mutual exclusion), even if the algorithm is run asynchronously, while the liveness property (deadlock-freedom) depends on the timing assumptions. We also proved a nearly-matching lower bound tradeoff between the number of registers and the time bound; this proof uses techniques like those in [15], extended to the timing-based setting.

The paper [9] also considers mutual exclusion in a setting with timing assumptions. This paper contains algorithms and lower bounds for mutual exclusion in certain centralized and distributed settings, in terms of the parameters describing the timing uncertainty.

A point of possible interest here is that this work was couched as an attempt to begin developing a general theory, with upper and lower bounds, for systems with approximate knowledge of timing. The underlying shared-memory model used here was expressed in terms of a preliminary type of "Timed I/O Automata" model, basically I/O Automata with added time bounds for the tasks. We developed more general Timed I/O Automata models later, see Section 6.

Consensus: Paper 5.1 [8] provides a study of the time costs of reaching consensus in a setting with timing assumptions. The paper contains new algorithms and lower bounds for consensus in the uncertain-timing setting. The key discovery is that the inherent time complexity of consensus in the uncertain-timing setting corresponds essentially to one "long round", whose time cost depends on the timing uncertainty, plus $O(f)$ "short rounds", whose time cost is independent of the timing uncertainty.

5.1. Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. Journal of the ACM, 41(1):122–152, January 1994.

For the upper bound, we designed an algorithm that uses a simple failure detector, implemented using timeouts. Briefly, the algorithm executes alternating rounds, in which processes try to decide 0 at even-numbered rounds and 1 and odd-numbered rounds. Each round is classified as either *quiet* or $non-quiet$; in a $non-quiet$ round, every process receives an explicit message telling it to advance to the next round, whereas in a *quiet* round, some process fails to receive such a message. We show that *quiet* rounds are short, whereas $non-quiet$ rounds may be long, with their time depending on the timing uncertainty. Occurrence of a $non-quiet$ round leads to rapid termination.

For the lower bound, we used an intricate argument that includes a combination of many ingredients from our previous work on impossibility results: a

chain argument as in [48], plus a bivalence argument as in [52], plus arguments about shifting, stretching, and shrinking time as in [80, 99].

Gradient clock synchronization: In [40], my PhD student Rui Fan and I introduced the distributed *gradient clock synchronization* problem. As in traditional distributed clock synchronization, we considered a network of nodes equipped with hardware clocks with bounded drift. Nodes compute logical clock values based on their hardware clocks and message exchanges, and the goal is to synchronize the nodes' logical clocks as closely as possible. The new requirement is that the skew between any two nodes' logical clocks be bounded by a nondecreasing function of the uncertainty in message delay (distance) between the two nodes. That is, we require nearby nodes to be closely synchronized, and allow faraway nodes to be more loosely synchronized. We proved a lower bound of $\Omega(d + \log D \log \log D)$ on the worst case clock skew between two nodes at distance d from each other, where D is the diameter of the network. This showed that clock synchronization is not a local property, in the sense that the clock skew between two nodes depends not only on the distance between the nodes, but also on the size of the entire network.

Our theoretical effort on algorithms and lower bounds for timed systems led us to another, parallel research effort, on developing formal "Timed I/O Automata" models for distributed systems with timing assumptions and guarantees. We wanted these models to serve as the basis for theoretical work as described here, and also, to be useful for practical system description. See Section 6 for more about this modeling work.

5.4 k-Set Agreement

We continued work related to fault-tolerant distributed consensus in the basic synchronous model. Previous results on fault-tolerant consensus in the synchronous model showed that $f+1$ rounds are both necessary and sufficient, over a range of different types of failures. Paper 5.2 [20] generalizes these bounds to the problem of *k-set consensus*, which allows k possible decisions to be output, instead of just one. It considers stopping failures only. It turns out, perhaps surprisingly, that $\lfloor f/k \rfloor + 1$ rounds are both necessary and sufficient to solve k-set consensus for f failures. Thus, allowing k possible answers reduces the time for ordinary consensus by dividing it by k.

5.2. Soma Chaudhuri, Maurice Herlihy, Nancy A. Lynch, and Mark R. Tuttle. Tight bounds for k-Set agreement. Journal of the ACM, pages 47(5):912-943, September, 2000.

The algorithm is straightforward: every process keeps propagating the minimum value that it has ever seen. It turns out that the number of distinct minimum values remaining at different processes decreases to at most k by the stated number of rounds.

The lower bound extends the lower bound ideas from [48] to k dimensions, now allowing executions to "morph" gradually between $k+1$ extreme executions in each of which the initial value is fixed at one of the values $0, \ldots, k$, and no failures occur. In each execution, we can identify the output values for all the non-failed processes. If the executions are too short, then we can apply Sperner's lemma from algebraic topology to identify a single execution in the structure that exhibits $k+1$ different output values for non-failed processes.

5.5 Data consistency

We continued working on distributed database issues during this time. Here are two highlights.

Eventual serializability: In [41], we presented a new specification for a data service with guarantees that are weaker than the common notion of serializability, together with an algorithm that meets the specification. Our specification and algorithm are based on earlier work by Ladin et al. [72]. We gave a more abstract, general, formal specification of the guarantees of an *eventual serializable* distributed data service. We gave an algorithm that generalizes the one in [72] so that it applies to arbitrary data types and accommodates more kinds of ordering constraints.

The basic idea is that the system maintains a partial ordering of operations, which gets refined over time to a total ordering. Operations may get immediate responses that don't reflect their entire prefix in the final order, but the client may specify constraints on which previous operations must be taken into account. (This sounds vaguely reminiscent of what happens in many blockchain algorithms.)

Consistency, Availability, and Partition-Tolerance (CAP): Paper 5.3 [58] is a short paper by my PhD student Seth Gilbert and myself that formalizes a well-known informal conjecture by Eric Brewer, about capabilities of distributed data management systems. The paper was published in SIGACT News, rather than in a standard journal or conference. Nevertheless, it became my third most-referenced publication, exceeded only by my "Distributed Algorithms" book and the FLP paper. This may be because the paper is rather easy to read, and also because its results seem to say something that is meaningful to distributed systems researchers and developers working on web-based data services.

The paper formalizes what we thought Brewer meant by his claim that it is not possible, in general, to implement distributed data services in such a way as to achieve all three of the properties of *Consistency, Availability, and Partition-Tolerance*. We expressed this claim formally, in terms of read/write objects distributed in a network of asynchronous processes. Besides requiring data consistency (atomicity), we required that all operations must terminate, even in the presence of lost messages.

5.3. Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of Consistent, Available, Partition-tolerant web services. SIGACT News, 33(2):48-51, June 2002.

We proved the impossibility result using a familiar type of network-splitting argument. We then went on to consider the problem using variations on the model: some involved modifying the timing assumptions, while others involved weakening the consistency guarantees. Some of these variations maintained impossibility, while others admitted solutions.

This work led to quite a few later papers and much discussion. I won't try to survey these, but just point to a 10-years-later issue of IEEE Computer magazine that was largely devoted to perspectives on the CAP theorem [1].

5.6 Distributed Algorithms book

I also wrote my "Distributed Algorithms" textbook [86], Publication 5.4, during this time period. It has proved to be a successful textbook and reference for the field. It was based on my course notes from over 10 years of teaching an MIT graduate Distributed Algorithms course.

5.4. Nancy A. Lynch. Distributed Algorithms. Morgan Kaufmann, 1996.

6 Later Work: Formal Models and Methods, 1990-2005

During the years 1990-2005, in addition to continuing our work on algorithms and impossibility results as I described in Section 5, my collaborators and I engaged in another major research effort, on developing formal models and proof methods for describing and reasoning about distributed systems. The need for formal models and methods arose both from our work on theoretical algorithms, and from our attempts to model real distributed systems.

We already had three major papers on formal models for distributed systems [14, 101, 102]. These contained models for asynchronous distributed systems, with the beginnings of treatment of timing. The new work involved models for asynchronous and timing-dependent systems, hybrid continuous/discrete systems, and probabilistic systems.

This work got me involved in the formal methods research community, based mainly in Europe, and represented by such conferences as CONCUR, CAV, LICS, and RTSS. Key collaborators during this time were my postdocs Frits Vaandrager and Dilsun Kirli Kaynar, and my PhD student Roberto Segala.

In this section, I describe our formal methods work during 1990-2005. in the final Subsection 6.4, I give some examples of system modeling and verification projects that we carried out during this time, based on our various formal models.

6.1 Key publications

6.1. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O Automata. Information and Computation, 185(1):105-157, August 2003.

6.2. Dilsun Kirli Kaynar, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. The Theory of Timed I/O Automata, Second Edition. Synthesis Lectures on Distributed Computing Theory. Morgan and Claypool Pub- lishers, 2010.

6.3. Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. Nordic Journal of Computing, 2(2):250–273, August 1995.

6.4. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Observing Branching Structure through Probabilistic Contexts. Siam Journal on Computing, 37(4):977-1013, September 2007.

6.5. Alan Fekete, Frans Kaashoek and Nancy Lynch. Implementing Sequentially-Consistent Shared Objects Using Group and Point-to-Point Communication . Journal of the ACM, 45(1):35-69, January, 1998.

6.6. Alan Fekete, Nancy Lynch, and Alex Shvartsman. Specifying and Using a Partitionable Group Communication Service. ACM Transactions on Computer Systems, 19(2):171-216, May 2001.

6.2 Timed (and untimed) system models

My work with Frits Vaandrager focused on basic automata models for concurrent systems, first for untimed systems, and then for timed systems.

Untimed system models: Our initial work on untimed systems resulted in the paper [103]. The focus of that work was the use of abstraction mappings of various kinds to reason about the correctness of systems modeled as automata; this general idea was already present in [101]. The paper [103] contains many new results but can also be read as a comprehensive survey of the area.

The paper covers several different types of mappings, starting with simple *refinements*; a refinement is a (single-valued) function from the state of the lower-level automaton to the state of the higher-level automaton, which preserves step-by-step behavior.

The paper continues with *forward simulations*, which are multivalued mappings that again preserve the step-by-step behavior; now the preservation condition uses existential quantification over the possible choices of the state after the transition. Forward simulations are more general, more abstract versions of the concrete *history variable* mechanism studied earlier by Owicki and Gries [114] and by Abadi and Lamport [2]. We continued with *backward simulations*, which

again are multivalued mappings, but that use existential quantification over the possible choices of the state before the transition. Backward simulations are general, abstract versions of the *prophecy variable* mechanism of [2]. The paper also contains combinations of forward and backward simulations, which are shown to be complete for expressing implementation relationships between automata.

Timed system models: During these years, we worked quite a lot on developing formal models for timed systems, i.e., systems with some timing assumptions. Our model evolved to its final version [69] in a series of steps.

First, Hagit Attiya and I followed our work in [9] with another paper that focused on modeling issues for timed systems [87]. There, we defined timed automata in terms of untimed I/O Automata plus "boundmaps" which express upper and lower bounds on the time required for the next steps of particular tasks to occur. We intended this as a general, systematic way of incorporating time information into I/O Automata. By this time, we were convinced of the value of two particular proof methods for untimed automata: invariant assertions, and abstraction mappings. In order to make these methods work for I/O Automata with boundmaps, we developed a systematic way of incorporating the timing information given in boundmaps into the automaton state. We did this with special new state components representing predictions of when the tasks will next perform a step. With this addition to the model, it was easy to adapt invariant and abstraction mapping methods to work in timed systems.

In [104], Frits Vaandrager and I generalized the model used in [87] further, to allow a timed automaton to have states with arbitrary structure, and to have both discrete transitions and time-passage steps. We removed the requirement for the particular "task" structure of I/O Automata. This generalization turned out to be able to support the full range of mappings (refinements, forward simulations, backward simulations, etc.) that we presented for untimed systems in [103]. A key aspect of these timed automata is their notion of external behaviors: now they are not just traces, which are sequences of actions, but timed traces, which are sequences of actions with associated times of occurrence.

Frits Vaandrager, Roberto Segala, and I produced a further extension in Paper 6.1 [97]. In order to capture the behavior of real-time systems, such as robot-control systems, we extended the model to include continuous behavior. The combination of discrete and continuous behavior, i.e., *hybrid* behavior, was a growing research direction at the time, represented by the new Hybrid Systems Computation and Control conference. In addition to time-passage steps as in [104], we included *trajectories*, which describe the evolution of a state through continuous time. The resulting model still supports composition and abstraction; the complete theory appears in [97].

6.1. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O Automata. Information and Computation, 185(1):105-157, August 2003.

Actually, this paper [97] goes further, allowing not just internal state, but

also input and output variables, to evolve continuously. This provides support for describing continuous interaction between components, in addition to the usual discrete interaction via shared actions. But this generality led to some technical complications involving composition (now we had to worry about solvability of systems of differential equations).

The final version of our Timed I/O Automata model appears in the monograph [69], Publication 6.2. The Timed I/O Automata in this monograph have all the generality of our Hybrid I/O Automata in [97], except for the external (input and output) variables. Thus, the model includes trajectories describing evolution of the internal state, but the only communication between components is by discrete input and output actions. This simplification did not seem to reduce the practical expressive power of the model much, but it did simplify the theory related to composition and, we thought, would make the model easier to use for applications.

The monograph [69] contains the complete theory of Timed I/O Automata, including definitions for timed automata and timed system behavior, invariants, and simulation relations. It contains the theory for the composition and hiding operators, and many simple examples.

6.2. Dilsun Kirli Kaynar, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. The Theory of Timed I/O Automata, Second Edition. Synthesis Lectures on Distributed Computing Theory. Morgan and Claypool Pub- lishers, 2010.

6.3 Probabilistic system models

Paper 6.3 [121] shows how probabilistic systems can also be understood using abstraction mappings. The conference version of this appeared in CONCUR in 1994 [120], and just recently won a Test-of-Time award from the CONCUR community.

6.3. Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. Nordic Journal of Computing, 2(2):250–273, August 1995.

To summarize these papers, I give the citation from the CONCUR Test-of-Time award:

> The paper "Probabilistic simulations for probabilistic processes", published by Roberto Segala and Nancy Lynch at CONCUR 1994, receives one award for introducing the 'simple' probabilistic automata model. Unlike earlier attempts to embrace probabilities, transition targets here are probability distributions over states, and this makes it possible to lift core process algebraic results in a very elegant manner. Probabilistic automata have quickly been recognised as the pivotal link between classical concurrency theory and the theory of discrete-state Markov processes. They have become

the central subjects of probabilistic model checking, and are echoed in a range of very influential modelling formalisms including probabilistic timed automata, probabilistic hybrid automata, and Markov automata.

There is also a published interview of Roberto and myself about this paper, by Luca Aceto [4].

Roberto's 1995 PhD thesis, entitled "Modeling and Verification of Randomized Distributed Real-Time Systems", defined untimed and timed Probabilistic I/O Automata and set out their theory [119].

In the following years, Roberto and I continued working on probabilistic automata, joined at some point by Frits Vaandrager. A key difficulty arose in our study of composition of probabilistic automata, as a consequence of the fact that our automata can include both probabilistic and nondeterministic choices. Namely, this combination implies that, for a notion of external behavior to be compositional, it must expose "too much" of the internal branching structure of the automaton. This is made precise and proved in Paper 6.4 [98]. Roughly speaking, the problem arises when the nondeterministic choices get resolved by an adversarial "scheduler" that makes unfavorable choices based on knowledge of internal aspects of the prior execution.

6.4. Nancy Lynch, Roberto Segala, and Frits Vaandrager. Observing Branching Structure through Probabilistic Contexts. Siam Journal on Computing, 37(4):977-1013, September 2007.

To get around the problem demonstrated in [98], it seems necessary to restrict the power of the adversarial scheduler that controls the nondeterministic choices within a probabilistic automaton. Such restrictions have appeared in some of our subsequent work, such as [21] and [16]. In [21], we proposed a restricted scheduler based on a predetermined schedule of components and an adversarial scheduler within components. In [16], we described a framework for modeling and verifying security protocols, with a restricted scheduler that is based on an oblivious, global scheduler for tasks (a concept borrowed from I/O Automata), coupled with a local scheduler for resolving nondeterminism among the choices of actions within a task. We called the resulting model *Task-Structured Probabilistic I/O Automata*.

Other types of restricted schedulers also arise in some of our more recent work on wireless network communication and neural networks.

6.4 Verification examples

We used our formal methods work as the basis for many projects on modeling and verifying distributed systems, including distributed data processing systems, real-time control systems, and security protocols. Here is a brief overview, broken down according to the type of model used.

Asynchronous systems: The nested transaction work that I discussed in Section 4 was an extended case study, presenting many algorithms based on ones that were developed in the distributed systems community (generalized in some cases). We put everything in a uniform framework, in terms of I/O Automata. We published a comprehensive book summarizing the entire modeling project [93]. We also modeled and verified other distributed systems algorithms, and in some cases, found errors in the algorithms and helped to fix them. I describe a few examples here.

First, in Paper 6.5 [42], my PhD student Alan Fekete and I worked with Frans Kaashoek to try to understand, in formal terms, an interesting distributed algorithm that Frans had helped to develop for the Orca shared-object system, at the Vrije University in Amsterdam.

6.5. Alan Fekete, Frans Kaashoek and Nancy Lynch. Implementing Sequentially-Consistent Shared Objects Using Group and Point-to-Point Communication . Journal of the ACM, 45(1):35-69, January, 1998.

The algorithm implements a sequentially consistent collection of shared read/update objects in an asynchronous distributed network. It caches objects in the local memory of processors according to application needs. Each read operation accesses a single copy of the object, while each update accesses all copies. A process uses broadcast communication when it sends messages to all the copies of an object, and uses point-to-point communication when it sends a message to a single copy, and when it returns a reply. Copies of an object are kept consistent using a strategy based on sequence numbers for broadcasts.

We presented the algorithm using two layers of abstraction. The lower layer uses the given broadcast and point-to point communication services, plus sequence numbers, to implement a new, powerful communication service called a *context multicast channel*. The higher layer uses the context multicast service to manage the object replication consistently. We described and verified both layers and their combination using I/O Automata and abstraction methods.

Actually, while attempting to verify our model of the existing Orca algorithm, we found a logical error in the implemented algorithm, namely, it omitted sequence numbers from some messages that needed to include them. We produced a corrected version of the algorithm and verified that. The corrected algorithm was then incorporated into the Orca system.

Second, in Paper 6.6 [45], Alan Fekete, Alex Shvartsman, and I provided a new formal specification of a *partitionable Group Communication Service (GCS)*. GCSs were a topic of great interest in the distributed systems research community in the late 1990s, as building blocks for fault-tolerant distributed systems. A GCS is based on a *group membership service*, which maintains changing versions of the set of members of the group. The GCS then manages communication among the current group members.

6.6. Alan Fekete, Nancy Lynch, and Alex Shvartsman. Specifying and Using a Partitionable Group Communication Service. ACM Transactions on Computer

Systems, 19(2):171-216, May 2001.

Initially, it was not completely clear to us what the formal specification for the GCS should be. To pin down the details, we developed our specification for the GCS in the context of a particular application: an *ordered broadcast* service. Specifically, we designed an algorithm to implement ordered broadcast over a GCS, and proved the algorithm to be correct, based on our new formal specification of the GCS. This work served to clarify some notions that had been discussed less formally in the systems research community.

I also collaborated with Ken Birman and Jason Hickey at Cornell, with the aim of understanding the behavior of their Ensemble GCS, in formal terms. Ensemble was then a widely used GCS that supported distributed programming by providing precise guarantees for synchronization, message ordering, and message delivery. We used I/O Automata again, to formalize, specify, and try to verify the correctness of the Ensemble implementation [65]. Once again, the verification effort uncovered an algorithmic error in the implementation, which was subsequently repaired in the implementation.

Timed and hybrid systems: We used our timed automata and hybrid automata models extensively, for verifying algorithms derived from practical real-time systems. Here I mention a few examples.

First, in the early 1990s, I worked with Butler Lampson and my PhD student Jorgen Sogaard-Anderson on modeling and verifying two *At-Most-Once message delivery* protocols. These were patterned, respectively, after the standard protocol for setting up network connections used in TCP and other transport protocols, and a clock-based protocol of Liskov, Shrira, and Wroclawski [78]. We specified the algorithms and verified their key properties, using our Timed I/O Automata model. The proofs used invariants and abstraction mappings.

Second, I worked with Prof. Shankar Sastry and members of his Berkeley hybrid systems group, in particular, his PhD student John Lygeros, on modeling the behavior of automated vehicle systems. In 1998, we carried out a project modeling strings of automated vehicles and proving certain safety properties for their behavior [81]. We identified an emergency vehicle maneuver—one in which one car brakes to decelerate as fast as possible—and identified conditions under which crashes are guaranteed to be avoided. This work used our Hybrid I/O Automata model, and inductive proof methods, taking into account both discrete and continuous transitions. We also identified conditions under which crashes may occur; these proofs are by construction.

Third, in a project involving John Lygeros and my PhD student Carolos Livadas, we developed a model for the behavior of the recently-introduced aircraft *Traffic alert and Collision Avoidance System (TCAS)*. In our paper [79], we provided an abstract model of the essential behavior of the main TCAS collision avoidance algorithm, and proved that it in fact works to avoid aircraft collisions. Our work again used our Hybrid I/O Automata model. We presented this work to TCAS developers at Lincoln Labs. Our approach complemented

the detailed code discussions that were the norm at that time for developers of such safety-critical software. We advocated for the use of abstract modeling and formal verification, in addition to detailed code discussions.

Fourth, in 2002, my PhD student Sayan Mitra and I carried out a project with Prof. Eric Feron in MIT's Aero-Astro department and his student [108]. This involved modeling a toy helicopter system used for instructional purposes and verifying its safety properties. Again, we used Hybrid I/O Automata.

Finally, in a little-known, simple paper [83], I showed two ways in which abstraction mappings could be used for real-time systems modeled using Hybrid I/O Automata: for expressing the relationship between a derivative-based description of a system and an explicit functional description, and to express the relationship between a system description in which corrections are made at discrete sampling points and a version in which corrections are made continuously. Both types of relationships can be proved with simple forward simulation relations.

Probabilistic systems: Our main case study here involved verification of basic security protocols using *Task-Structured Probabilistic I/O Automata* [16]. Specifically, using Task-Structured Probabilistic I/O Automata, we modeled and verified some basic security protocols following Ran Canetti's *Universal Composability* approach. For details, see, for example, [17].

In Sections 5 and 6, I have summarized our contributions during 1990-2005 to the continuing active development of the field of distributed computing theory. Our work included theoretical algorithms and lower bounds, new formal models and methods for reasoning about distributed algorithms and systems, and applications. This work was part of the very active development of the field of distributed computing theory during these years.

Our main contributions during this time included some new of the new theoretical algorithms and lower bounds, including various results involving timing in distributed algorithms. They also included applications of the theory to understand issues arising in the distributed systems community, such as Brewer's CAP conjecture, They included new formal models for timed, hybrid, and probabilistic distributed systems, and applications of these models to verify systems algorithms. They also included a major textbook for the field,

So far, I have described work on algorithms related to traditional distributed systems, including timed and hybrid systems. After 2005, our focus shifted to new types of distributed systems: wireless networks and biological systems. In these systems, noise, uncertainty, and change predominate, and so the systems must be flexible, robust, and adaptive. We continued carrying out the same general kinds of research, namely, defining problems to be solved in these systems, developing algorithms, and defining formal foundations.

7 Still Later Work: Wireless Network Algorithms, 2005 and Later

We continued working on algorithms and lower bounds after 2005, but now focusing on wireless systems, including mobile wireless systems. We considered ad hoc wireless networks, as might be used, for example, by soldiers in hostile territory, first responders in a disaster area, or swarms of robots. The main challenge was to design new algorithms for important problems (communication, searching, maintaining data, coordinating agents). We also wanted to define good abstraction layers to mask some of the difficulties and make it easier to develop applications for ad hoc wireless networks.

7.1 Key publications

7.1. Seth Gilbert, Nancy Lynch, and Alex Shvartsman. RAMBO: A Robust, Reconfigurable Atomic Memory Service for Dynamic Networks. Distributed Computing, 23(4):225-272, December 2010. This is a slightly corrected version of the journal version.

7.2. Matthew Brown, Seth Gilbert, Nancy Lynch, Calvin Newport, Tina Nolte, and Michael Spindel. The Virtual Node Layer: A Programming Abstraction for Wireless Sensor Networks. ACM SIGBED Review, 4(3), July 2007. Also, Proceedings of the the International Workshop on Wireless Sensor Network Architecture (WWSNA), Cambridge, MA, April, 2007.

7.3. Seth Gilbert, Nancy Lynch, Sayan Mitra, and Tina Nolte. Self-Stabilizing Robot Formations Over Unreliable Networks. ACM Transactions on Autonomous and Adaptive Systems, 4(3):17.2-17.27, July 2009.

7.4. Fabian Kuhn, Nancy Lynch and Rotem Oshman. Distributed Computation in Dynamic Networks. Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010), Cambridge, MA, pages 513-522, June 2010.

7.5. Alejandro Cornejo, Fabian Kuhn, Ruy Ley-Wild, and Nancy Lynch. Keeping Mobile Robot Swarms Connected. In Idit Keidar, editor, Distributed Computing, DISC 2009: 23rd International Symposium on Distributed Computing, Elche/Elx, Spain, September 23-25 2009, volume 5805 of Lecture Notes in Computer Science, pages 496-511, 2009. Springer.

7.2 Reconfigurable atomic memory

In 2002, we developed the *RAMBO (Reconfigurable Atomic Memory for Basic Objects)* algorithm for maintaining the appearance of globally-shared read/write memory in changing networks of mobile devices [85]. This might be used, for

example, by a company of soldiers operating in a hostile environment, without access to fixed wireless infrastructure.[1]

The algorithm replicated each object at a set of nodes. Read and write operations were implemented by accessing read quorums and write quorums of copies, respectively. That was sufficient for relatively stable situations, in which only small, transient changes occurred. For larger and more permanent changes, the algorithm also supported explicit *reconfiguration* operations. To move from an old configuration to a new one, the algorithm used the simple trick of employing both configurations for a while, during the period when the configuration was changing.

The final version of the paper appeared in 2010, in Paper 7.1 [61], after several earlier versions.

7.1. Seth Gilbert, Nancy Lynch, and Alex Shvartsman. RAMBO: A Robust, Reconfigurable Atomic Memory Service for Dynamic Networks. Distributed Computing, 23(4):225-272, December 2010. This is a slightly corrected version of the journal version.

From the abstract:

> In this paper, we present Rambo, an algorithm for emulating a read/write distributed shared memory in a dynamic, rapidly changing environment. Rambo provides a highly reliable, highly available service, even as participants join, leave, and fail. In fact, the entire set of participants may change during an execution, as the initial devices depart and are replaced by a new set of devices. Even so, Rambo ensures that data stored in the distributed shared memory remains available and consistent.
>
> There are two basic techniques used by Rambo to tolerate dynamic changes. Over short intervals of time, replication suffices to provide fault-tolerance. While some devices may fail and leave, the data remains available at other replicas. Over longer intervals of time, Rambo copes with changing participants via reconfiguration, which incorporates newly joined devices while excluding devices that have departed or failed. The main novelty of Rambo lies in the combination of an efficient reconfiguration mechanism with a quorum-based replication strategy for read/write shared memory.
>
> The Rambo algorithm can tolerate a wide variety of aberrant behavior, including lost and delayed messages, participants with unsynchronized clocks, and, more generally, arbitrary asynchrony. Despite such behavior, Rambo guarantees that its data is stored consis-

[1] For me, this work was inspired by start of the war in Afghanistan, soon after the events of September 11, 2001. I imagined the difficulties that soldiers roaming in an unfamiliar area would have in communicating, and especially, in maintaining reliable data. Of course, no cell towers would be available, so everything needed to be implemented on the mobile devices themselves.

tently. We analyze the performance of Rambo during periods when the system is relatively well-behaved: messages are delivered in a timely fashion, reconfiguration is not too frequent, etc. We show that in these circumstances, read and write operations are efficient, completing in at most eight message delays.

7.3 Virtual Nodes

We introduced the concept of *Virtual Nodes (VNs)*, as an abstraction for building applications over mobile wireless networks. The idea is to allow the rather chaotic collection of mobile nodes to cooperate to implement more reliable and stable abstract state machines called Virtual Nodes, which could then be programmed as if they were real machines.[2] Our initial efforts focused on Virtual Mobile Nodes [34], but we later emphasized Virtual Stationary Nodes at fixed geographical locations.

This work led to many papers including [34, 35, 33, 36, 13, 112, 113, 57], PhD theses for Tina Nolte [111] and Seth Gilbert [56], and MEng theses for Matt Brown [12] and Mike Spindel [122]. I highlight three papers here: an early paper [35] on using VNs to implement atomic memory in mobile networks, a basic position paper [13] summarizing the general Virtual Node approach, and an application of VNs to robot swarm motion coordination [59].

First, the paper [35] describes a way in which (reliable) mobile nodes can implement read/write atomic memory. The idea is simply to have the mobile nodes implement Virtual Stationary Nodes at known, fixed locations, and to have the Virtual Stationary Nodes implement a quorum-based atomic read/write memory algorithm. In this setting, a VN might fail, when its local area becomes empty of real mobile nodes. The paper discusses limited reconfiguration mechanisms, which can be used to bring a failed VN up-to-date when its local area becomes repopulated.

Paper 7.2 [13] is a short position paper motivating and describing the Virtual Node approach. It contains an interesting example of *Virtual Traffic Lights*, which could be implemented by computers on the cars. Actually, that was not practical at the time this paper was written since not all cars then had on-board computers, but now this would be quite feasible.

7.2. Matthew Brown, Seth Gilbert, Nancy Lynch, Calvin Newport, Tina Nolte, and Michael Spindel. The Virtual Node Layer: A Programming Abstraction for Wireless Sensor Networks. ACM SIGBED Review, 4(3), July 2007. Also, Proceedings of the the International Workshop on Wireless Sensor Network Architecture (WWSNA), Cambridge, MA, April, 2007.

Finally, Paper 7.3 [59] gives an example of an application of Virtual Stationary Nodes to robot swarm motion coordination, for example, guiding robots to

[2] Virtual Nodes are reminiscent of the Virtual Supervisor abstraction that we used in our very first paper on distributed computing theory [14]. That was for a shared memory model.

surround a hazardous waste spilll.

7.3. Seth Gilbert, Nancy Lynch, Sayan Mitra, and Tina Nolte. Self-Stabilizing Robot Formations Over Unreliable Networks. ACM Transactions on Autonomous and Adaptive Systems, 4(3):17.2-17.27, July 2009.

Other notable papers describe uses of Virtual Nodes to assist in message routing and to implement Virtual Air-Traffic Controllers.

7.4 Computing in dynamic networks

In this subsection and the next, I describe two more projects involving computing in mobile networks. Paper 7.4 [71] is a theoretical paper by postdoc Fabian Kuhn, PhD student Rotem Oshman, and myself, containing algorithms and lower bounds for dynamic networks in which the network topology changes from round to round.

7.4. Fabian Kuhn, Nancy Lynch and Rotem Oshman. Distributed Computation in Dynamic Networks. Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010), Cambridge, MA, pages 513-522, June 2010.

The paper assumes a worst-case model in which the communication links for each round are chosen by an adversary, and nodes do not know who their neighbors for the current round are before they broadcast their messages. We require correctness and termination even in networks that change continually. Our results rely on a network connectivity property that we called T-*interval connectivity*, which says that, in every T consecutive rounds, there is a stable connected spanning subgraph.

For this model, the paper contains algorithms by which the nodes can determine the size of the network, and compute any function of their initial inputs, in $O(n^2/T)$ rounds. It also contains lower bounds for the token dissemination problem, which requires the nodes to disseminate information to all the nodes in the network.

7.5 Robot swarm algorithms

PhD student Alejandro Cornejo led a project on algorithms for various motion-planning tasks in robot swarms; see, for example, [25, 26, 27]. In particular, Paper 7.5 [25] contains the design for a *connectivity service* that can adapt any robot swarm motion planner to ensure that the swarm remains globally connected for communication throughout the computation. The connectivity service does not interfere with progress of the robots.

7.5. Alejandro Cornejo, Fabian Kuhn, Ruy Ley-Wild, and Nancy Lynch. Keeping Mobile Robot Swarms Connected. In Idit Keidar, editor, Distributed Computing, DISC 2009: 23rd International Symposium on Distributed Computing,

Elche/Elx, Spain, September 23-25 2009, volume 5805 of Lecture Notes in Computer Science, pages 496-511, 2009. Springer.

We also have recent work on robot swarm computations, but it is a bit too preliminary to discuss here.

7.6 The Dual Graph model

All of the work I have described so far for wireless networks assumes that communication is reliable. We also considered unreliable communication, formalized in terms of a static *Dual Graph* model. A Dual Graph consists of a graph G of edges that support reliable communication and a super-graph G' containing additional edges over which messages might or might not be delivered. The options for delivery are assumed to be controlled by an adversary. In addition, the networks include another type of communication unreliability: message collisions in which colliding messages are lost.

For this difficult setting, we produced many papers containing upper and lower bounds for solving problems such as local and global broadcast, for example, [70, 53, 54, 18, 96, 60].

Generally speaking, the Dual Graph model was so difficult that we ended up proving mostly lower bounds on the time costs for solving problems. These lower bounds served to establish a difference in power between wireless models with reliable and unreliable communication. We also tried to design algorithms for the Dual Graph model that worked as well as possible, but their performance was not great.

One reason why the algorithms for the Dual Graph model did not perform too well is that they included a combination of probabilistic and nondeterministic choices: the algorithms are probabilistic, and message delivery is nondeterministic, potentially controlled by an adversary. As I noted in Section 6.3, that can cause technical problems with regard to composition. In the case of typical wireless network algorithms, like backoff algorithms, it also causes efficiency problems; see [60]. Our way around the problem in [60] is, again, to weaken the power of the adversary. In this case, we assume that the adversary is stochastic rather than worst-case.

Thus, our contributions to the wireless network area mainly involved new algorithms and abstractions to try to make programming of such networks tractable. In the remaining section of this paper, I will briefly discuss our recent work on biological distributed algorithms.

8 Biological Distributed Algorithms, 2012 and later

During the last 12 years or so, my group has been focusing on very different kinds of distributed algorithms, namely, those arising in biological systems. Most

biological systems are distributed—think of colonies of insects, and networks of cells such as brain networks—so it is natural to view them in terms of distributed algorithms. But they are distributed algorithms of a particular kind: flexible, robust, and adaptive.

Biological systems, of course, include many biological details. However, we expect that less detailed, abstract models of these systems can be defined, and can be treated as abstract distributed algorithms. We hope that some of the algorithm design and analysis methods from traditional distributed computing theory will carry over to these new kinds of distributed systems. This could contribute a new type of understanding to the field of biology. We also hope that, once we understand how the biological mechanisms work, we will be able to adapt them for use in engineered systems such as robot swarms and neural networks.

My group and I have mainly worked on two kinds of biological systems. The work is still in progress, and preliminary, so I will not say too much about it now.

8.1 Insect colonies

Our project on insect colony algorithms has involved designing algorithms that capture key aspects of real insect colony behavior, modeling the algorithms formally, simulating them to try to infer properties of their behavior, and in some cases, following the simulation results with analysis.

The main problems that we have considered so far include searching for food, reaching consensus on a new nest ("house-hunting") [55, 126], task allocation [24, 118, 123, 37], and density estimation [109, 110].

Generally, we have been successful at modeling these types of insect colony behavior abstractly, and have observed and proved properties that are consistent with experimental observations by insect biologists. We have made a few observations from our modeling work that might suggest new experiments.

8.2 Brain Networks

The main theme of our work on brain networks is to model mechanisms that might be used in the brain, for solving various brain problems. We have treated these mechanisms as abstract distributed algorithms and have proved properties of their behavior.

Some of our representative papers consider the *Winner-Take-All* problem [95], concept representation and memorization [66], and representation and learning of hierarchically-structured concepts [90, 91]. We also developed a basic *formal modeling framework* for brain network algorithms [94], based on synchronous, stochastic *Spiking Neural Networks (SNNs)*.

9 Conclusions

My collaborators and I have worked for many years on theory for distributed systems, in order to help understand their capabilities and limitations. This work has included:

- Models for distributed system problems and algorithms.
- Some new algorithms.
- Rigorous proofs, and discovery of errors in existing algorithms.
- Impossibility results, expressing inherent limitations.
- General foundations for modeling and analyzing distributed systems, and
- Applications to distributed data-management systems, wired and wireless communication systems, real-time systems, and biological systems.

I hope you have enjoyed this summary of our contributions to the field of Distributed Computing Theory. I will probably revisit this in the future with more discussion of the newer work.

References

[1] *The CAP Theorem's Growing Impact*. IEEE Computer, 2012.

[2] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991. URL: https://www.sciencedirect.com/science/article/pii/030439759190224P, https://doi.org/https://doi.org/10.1016/0304-3975(91)90224-P doi:https://doi.org/10.1016/0304-3975(91)90224-P.

[3] Ittai Abraham, Gregory V. Chockler, Idit Keidar, and Dahlia Malkhi. Byzantine disk paxos: optimal resilience with byzantine shared memory. *Distributed Comput.*, 18(5):387–408, 2006. https://doi.org/10.1007/s00446-005-0151-6 doi:10.1007/s00446-005-0151-6.

[4] Luca Aceto. An interview with Nancy Lynch and Roberto Segala, CONCUR Test-of-Time Award recipients. processalgebra.blogspot.com, April 2020.

[5] Yehuda Afek, Hagit Attiya, Alan D. Fekete, Michael J. Fischer, Nancy A. Lynch, Yishay Mansour, Da-Wei Wang, and Lenore D. Zuck. Reliable communication over unreliable channels. *J. ACM*, 41(6):1267–1297, 1994. https://doi.org/10.1145/195613.195651 doi:10.1145/195613.195651.

[6] Eshrat Arjomandi and Derek G. Corneil. Parallel computations in graph theory. In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 13–18. IEEE Computer Society, 1975. https://doi.org/10.1109/SFCS.1975.24 doi:10.1109/SFCS.1975.24.

[7] Eshrat Arjomandi, Michael J. Fischer, and Nancy A. Lynch. Efficiency of synchronous versus asynchronous distributed systems. *Journal of the ACM*, 30(3):449–456, July 1983.

[8] Hagit Attiya, Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41(1):122–152, 1994. https://doi.org/10.1145/174644.174649 doi:10.1145/174644.174649.

[9] Hagit Attiya and Nancy A. Lynch. Time bounds for real-time process control in the presence of timing uncertainty. *Inf. Comput.*, 110(1):183–232, 1994. https://doi.org/10.1006/inco.1994.1030 doi:10.1006/inco.1994.1030.

[10] Hagit Attiya, Nancy A. Lynch, and Nir Shavit. Are wait-free algorithms fast? *J. ACM*, 41(4):725–763, 1994. https://doi.org/10.1145/179812.179902 doi:10.1145/179812.179902.

[11] Bard Bloom. Constructing two-writer atomic registers. *IEEE Trans. Computers*, 37(12):1506–1514, 1988. https://doi.org/10.1109/12.9729 doi:10.1109/12.9729.

[12] Matthew Brown. Air traffic control using virtual stationary automata. Master of Engineering Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, September 2007.

[13] Matthew Brown, Seth Gilbert, Nancy A. Lynch, Calvin C. Newport, Tina Nolte, and Michael Spindel. The virtual node layer: a programming abstraction for wireless sensor networks. *SIGBED Rev.*, 4(3):7–12, 2007. https://doi.org/10.1145/1317103.1317105 doi:10.1145/1317103.1317105.

[14] James E. Burns, Paul Jackson, Nancy A. Lynch, Michael J. Fischer, and Gary L. Peterson. Data requirements for implementation of n-process mutual exclusion using a single shared variable. *J. ACM*, 29(1):183–205, 1982. https://doi.org/10.1145/322290.322302 doi:10.1145/322290.322302.

[15] James E. Burns and Nancy A. Lynch. Bounds on shared memory for mutual exclusion. *Inf. Comput.*, 107(2):171–184, 1993. https://doi.org/10.1006/inco.1993.1065 doi:10.1006/inco.1993.1065.

[16] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Task-structured probabilistic I/O automata. *Journal of Computer and System Sciences*, 94:63–97, 2018.

[17] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses D. Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Analyzing security protocols using time-bounded task-pioas. *Discret. Event Dyn. Syst.*, 18(1):111–159, 2008. https://doi.org/10.1007/s10626-007-0032-1 doi:10.1007/s10626-007-0032-1.

[18] Keren Censor-Hillel, Seth Gilbert, Fabian Kuhn, Nancy Lynch, and Calvin Newport. Structuring unreliable radio networks. *Distributed Computing*, 27(1):1–19, 2014.

[19] Soma Chaudhuri, Maurice Herlihy, Nancy A. Lynch, and Mark R. Tuttle. A tight lower bound for k-set agreement. In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 206–215. IEEE Computer Society, 1993. https://doi.org/10.1109/SFCS.1993.366866 doi:10.1109/SFCS.1993.366866.

[20] Soma Chaudhuri, Maurice Herlihy, Nancy A. Lynch, and Mark R. Tuttle. Tight bounds for k-set agreement. *J. ACM*, 47(5):912–943, 2000. https://doi.org/10.1145/355483.355489 doi:10.1145/355483.355489.

[21] Ling Cheung, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Switched PIOA: parallel composition via distributed scheduling. *Theor. Comput. Sci.*, 365(1-2):83–108, 2006. https://doi.org/10.1016/j.tcs.2006.07.033 doi:10.1016/j.tcs.2006.07.033.

[22] Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: A comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, dec 2001. https://doi.org/10.1145/503112.503113 doi:10.1145/503112.503113.

[23] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. https://doi.org/10.1145/800157.805047 doi:10.1145/800157.805047.

[24] Alejandro Cornejo, Anna R. Dornhaus, Nancy A. Lynch, and Radhika Nagpal. Task allocation in ant colonies. In Fabian Kuhn, editor, *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, volume 8784 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2014. https://doi.org/10.1007/978-3-662-45174-8_4 doi:10.1007/978-3-662-45174-8_4.

[25] Alejandro Cornejo, Fabian Kuhn, Ruy Ley-Wild, and Nancy Lynch. Keeping mobile robot swarms connected. In Idit Keidar, editor, *23rd International Symposium on Distributed Computing (DISC 2009), Elche/Elx, Spain, September 23-25, 2009*, volume 5805 of *Lecture Notes in Computer*

Science, pages 496–511. Springer, 2009. Also, Extended version Technical Report MIT-CSAIL-TR-2009-027, MIT CSAIL, Cambridge, MA, June 2009.

[26] Alejandro Cornejo and Nancy Lynch. Fault-tolerance through k-connectivity. In *IEEE International Conference on Robotics and Automation (ICRA 2010): Workshop on Network Science and System Issues in Multi-Robot Autonomy*, Anchorage, Alaska, May 2010.

[27] Alejandro Cornejo and Nancy Lynch. Reliably detecting connectivity using local graph traits. In Mohamed Mosbah Chenyang Lu, Toshimitsu Masuzawa, editor, *Principles of Distributed Systems (OPODIS 2010), Tozeur, Tunisia, December 2010*, volume 6490 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 2010.

[28] Simon Doherty, Lindsay Groves, Victor Luchangco, and Mark Moir. Towards formally specifying and verifying transactional memory. *Formal Aspects Comput.*, 25(5):769–799, 2013. https://doi.org/10.1007/s00165-012-0225-8 doi:10.1007/s00165-012-0225-8.

[29] Danny Dolev, Michael J. Fischer, Robert J. Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Inf. Control.*, 52(3):257–274, 1982. https://doi.org/10.1016/S0019-9958(82)90776-8 doi:10.1016/S0019-9958(82)90776-8.

[30] Danny Dolev, Joseph Y. Halpern, and H. Raymond Strong. On the possibility and impossibility of achieving clock synchronization. *J. Comput. Syst. Sci.*, 32(2):230–250, 1986. https://doi.org/10.1016/0022-0000(86)90028-0 doi:10.1016/0022-0000(86)90028-0.

[31] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, 1986. https://doi.org/10.1145/5925.5931 doi:10.1145/5925.5931.

[32] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 401–407, New York, NY, USA, 1982. Association for Computing Machinery. https://doi.org/10.1145/800070.802215 doi:10.1145/800070.802215.

[33] Shlomi Dolev, Seth Gilbert, Limor Lahiani, Nancy Lynch, and Tina Nolte. Timed virtual stationary automata for mobile networks. In *Principles of Distributed systems: 9th International Conference on Principles of Distributed Systems (OPODIS 2005), Pisa, Italy, December 12-14, 2005*, volume 3974 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2006. Also, Technical Report MIT-LCS-TR-979a, MIT CSAIL, Cambridge, MA 02139, August 2005.

[34] Shlomi Dolev, Seth Gilbert, Nancy A. Lynch, Elad Schiller, Alexander A. Shvartsman, and Jennifer L. Welch. Virtual mobile nodes for mobile ad hoc networks. In Rachid Guerraoui, editor, *Distributed Computing, 18th International Conference, DISC 2004, Amsterdam, The Netherlands, October 4-7, 2004, Proceedings*, volume 3274 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 2004. https://doi.org/10.1007/978-3-540-30186-8_17 doi:10.1007/978-3-540-30186-8_17.

[35] Shlomi Dolev, Seth Gilbert, Nancy A. Lynch, Alexander A. Shvartsman, and Jennifer L. Welch. Geoquorums: implementing atomic memory in mobile *ad hoc* networks. *Distributed Comput.*, 18(2):125–155, 2005. https://doi.org/10.1007/s00446-005-0140-9 doi:10.1007/s00446-005-0140-9.

[36] Shlomi Dolev, Limor Lahiani, Nancy Lynch, and Tina Nolte. Self-stabilizing mobile node location management and message routing. In Sebastien Tixeuil Ted Herman, editor, *Self-Stabilizing Systems: Seventh International Symposium on Self-Stabilizing Systems (SSS 2005), Barcelona, Spain, October 26-27*, volume 3764 of *Lecture Notes in Computer Science*, pages 96–112. Springer, 2005. Also, Technical Report MIT-LCS-TR-999, MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, August 2005.

[37] Anna R. Dornhaus, Nancy A. Lynch, Frederik Mallmann-Trenn, Dominik Pajak, and Tsvetomira Radeva. Self-stabilizing task allocation in spite of noise. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 201–211. ACM, 2020. https://doi.org/10.1145/3350755.3400226 doi:10.1145/3350755.3400226.

[38] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. https://doi.org/10.1145/42282.42283 doi:10.1145/42282.42283.

[39] Faith Ellen and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16:121–163, 09 2003. https://doi.org/10.1007/s00446-003-0091-y doi:10.1007/s00446-003-0091-y.

[40] Rui Fan and Nancy Lynch. Gradient clock synchronization. *Distributed Computing*, 18(4):255–266, November 2006.

[41] Alan D. Fekete, David Gupta, Victor Luchangco, Nancy A. Lynch, and Alexander A. Shvartsman. Eventually-serializable data services. *Theor. Comput. Sci.*, 220(1):113–156, 1999. https://doi.org/10.1016/S0304-3975(98)00239-4 doi:10.1016/S0304-3975(98)00239-4.

[42] Alan D. Fekete, M. Frans Kaashoek, and Nancy A. Lynch. Implementing sequentially consistent shared objects using broadcast

and point-to-point communication. *J. ACM*, 45(1):35–69, 1998. https://doi.org/10.1145/273865.273884 doi:10.1145/273865.273884.

[43] Alan D. Fekete, Nancy A. Lynch, Yishay Mansour, and John Spinelli. The impossibility of implementing reliable communication in the face of crashes. *J. ACM*, 40(5):1087–1107, 1993. https://doi.org/10.1145/174147.169676 doi:10.1145/174147.169676.

[44] Alan D. Fekete, Nancy A. Lynch, and Liuba Shrira. A modular proof of correctness for a network synchronizer (research summary). In Jan van Leeuwen, editor, *Distributed Algorithms, 2nd International Workshop, Amsterdam, The Netherlands, July 8-10, 1987, Proceedings*, volume 312 of *Lecture Notes in Computer Science*, pages 219–256. Springer, 1987. https://doi.org/10.1007/BFb0019807 doi:10.1007/BFb0019807.

[45] Alan D. Fekete, Nancy A. Lynch, and Alexander A. Shvartsman. Specifying and using a partitionable group communication service. *ACM Trans. Comput. Syst.*, 19(2):171–216, 2001. https://doi.org/10.1145/377769.377776 doi:10.1145/377769.377776.

[46] Michael J. Fischer, Nancy D. Griffeth, Leonidas J. Guibas, and Nancy A. Lynch. Optimal placement of identical resources in a tree. *Inf. Comput.*, 96(1):1–54, 1992. https://doi.org/10.1016/0890-5401(92)90053-I doi:10.1016/0890-5401(92)90053-I.

[47] Michael J. Fischer, Nancy D. Griffeth, and Nancy A. Lynch. Global states of a distributed system. *IEEE Trans. Software Eng.*, 8(3):198–202, 1982. https://doi.org/10.1109/TSE.1982.235418 doi:10.1109/TSE.1982.235418.

[48] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982. https://doi.org/10.1016/0020-0190(82)90033-3 doi:10.1016/0020-0190(82)90033-3.

[49] Michael J. Fischer, Nancy A. Lynch, James E. Burns, and Allan Borodin. Resource allocation with immunity to limited process failure. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 234–254, 1979. https://doi.org/10.1109/SFCS.1979.37 doi:10.1109/SFCS.1979.37.

[50] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Comput.*, 1(1):26–39, 1986. https://doi.org/10.1007/BF01843568 doi:10.1007/BF01843568.

[51] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In Ronald Fagin and Philip A. Bernstein, editors, *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23,*

1983, Colony Square Hotel, Atlanta, Georgia, USA, pages 1–7. ACM, 1983. https://doi.org/10.1145/588058.588060 doi:10.1145/588058.588060.

[52] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. https://doi.org/10.1145/3149.214121 doi:10.1145/3149.214121.

[53] Mohsen Ghaffari, Bernhard Haeupler, Nancy A. Lynch, and Calvin C. Newport. Bounds on contention management in radio networks. In Marcos K. Aguilera, editor, *Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings*, volume 7611 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2012. https://doi.org/10.1007/978-3-642-33651-5_16 doi:10.1007/978-3-642-33651-5_16.

[54] Mohsen Ghaffari, Nancy A. Lynch, and Calvin C. Newport. The cost of radio network broadcast for different models of unreliable links. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 345–354. ACM, 2013. https://doi.org/10.1145/2484239.2484259 doi:10.1145/2484239.2484259.

[55] Mohsen Ghaffari, Cameron Musco, Tsvetomira Radeva, and Nancy A. Lynch. Distributed house-hunting in ant colonies. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 57–66. ACM, 2015. https://doi.org/10.1145/2767386.2767426 doi:10.1145/2767386.2767426.

[56] Seth Gilbert. *Virtual Infrastructure for Wireless Ad Hoc Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, August 2007.

[57] Seth Gilbert, Nancy Lynch, Sayan Mitra, and Tina Nolte. Self-stabilizing mobile robot formations with virtual nodes. In Sandeep S. Kulkarni and Andre Schiper, editors, *Stabilization, Safety and Security of Distributed Systems, 10th International Symposium (SSS 2008), Detroit, Michigan, November 2008*, volume 5340 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2008.

[58] Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002. https://doi.org/10.1145/564585.564601 doi:10.1145/564585.564601.

[59] Seth Gilbert, Nancy A. Lynch, Sayan Mitra, and Tina Nolte. Self-stabilizing robot formations over unreliable networks. *ACM Trans. Auton. Adapt. Syst.*, 4(3):17:1–17:29, 2009. https://doi.org/10.1145/1552297.1552300 doi:10.1145/1552297.1552300.

[60] Seth Gilbert, Nancy A. Lynch, Calvin Newport, and Dominik Pajak. On simple back-off in unreliable radio networks. *Theor. Comput. Sci.*, 806:489–508, 2020. https://doi.org/10.1016/j.tcs.2019.08.027 doi:10.1016/j.tcs.2019.08.027.

[61] Seth Gilbert, Nancy A. Lynch, and Alexander A. Shvartsman. Rambo: a robust, reconfigurable atomic memory service for dynamic networks. *Distributed Comput.*, 23(4):225–272, 2010. https://doi.org/10.1007/s00446-010-0117-1 doi:10.1007/s00446-010-0117-1.

[62] Joseph Y. Halpern, Barbara Simons, H. Raymond Strong, and Danny Dolev. Fault-tolerant clock synchronization. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 89–102. ACM, 1984. https://doi.org/10.1145/800222.806739 doi:10.1145/800222.806739.

[63] Maurice Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, jan 1991. https://doi.org/10.1145/114005.102808 doi:10.1145/114005.102808.

[64] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999. https://doi.org/10.1145/331524.331529 doi:10.1145/331524.331529.

[65] Jason Hickey, Nancy A. Lynch, and Robbert van Renesse. Specifications and proofs for ensemble layers. In Rance Cleaveland, editor, *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, volume 1579 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 1999. https://doi.org/10.1007/3-540-49059-0_9 doi:10.1007/3-540-49059-0_9.

[66] Yael Hitron, Nancy A. Lynch, Cameron Musco, and Merav Parter. Random sketching, clustering, and short-term memory in spiking neural networks. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 23:1–23:31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. https://doi.org/10.4230/LIPIcs.ITCS.2020.23 doi:10.4230/LIPIcs.ITCS.2020.23.

[67] Philip H. Enslow Jr. What is a "distributed" data processing system? *Computer*, 11(1):13–21, 1978. https://doi.org/10.1109/C-M.1978.217901 doi:10.1109/C-M.1978.217901.

[68] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. https://doi.org/10.1007/978-1-4684-2001-2_9 doi:10.1007/978-1-4684-2001-2_9.

[69] Dilsun Kirli Kaynar, Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. *The Theory of Timed I/O Automata, Second Edition.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2010. https://doi.org/10.2200/S00310ED1V01Y201011DCT005 doi:10.2200/S00310ED1V01Y201011DCT005.

[70] Fabian Kuhn, Nancy A. Lynch, Calvin C. Newport, Rotem Oshman, and Andréa W. Richa. Broadcasting in unreliable radio networks. In Andréa W. Richa and Rachid Guerraoui, editors, *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 336–345. ACM, 2010. https://doi.org/10.1145/1835698.1835779 doi:10.1145/1835698.1835779.

[71] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 513–522. ACM, 2010. https://doi.org/10.1145/1806689.1806760 doi:10.1145/1806689.1806760.

[72] Rivka Ladin, Barbara Liskov, and Liuba Shrira. Lazy replication: exploiting the semantics of distributed services. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, PODC '90, page 43–57, New York, NY, USA, 1990. Association for Computing Machinery. https://doi.org/10.1145/93385.93399 doi:10.1145/93385.93399.

[73] Richard E. Ladner, Nancy A. Lynch, and Alan L. Selman. Comparisons of polynomial-time reducibilities. In Robert L. Constable, Robert W. Ritchie, Jack W. Carlyle, and Michael A. Harrison, editors, *Proceedings of the 6th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1974, Seattle, Washington, USA*, pages 110–121. ACM, 1974. https://doi.org/10.1145/800119.803891 doi:10.1145/800119.803891.

[74] L. Lamport. The weak byzantine generals problem. *J. ACM*, 30(3):668–676, July 1983. https://doi.org/10.1145/2402.322398 doi:10.1145/2402.322398.

[75] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998. https://doi.org/10.1145/279227.279229 doi:10.1145/279227.279229.

[76] Leslie Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. 32(1):52–78, 1985.

[77] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. https://doi.org/10.1145/357172.357176 doi:10.1145/357172.357176.

[78] Barbara Liskov, Liuba Shrira, and John Wroclawski. Efficient at-most-once messages based on synchronized clocks. *ACM transactions on computer systems*, 9(2):125 – 142, 1991. https://doi.org/10.1145/103720.103722 doi:10.1145/103720.103722.

[79] C. Livadas, J. Lygeros, and N.A. Lynch. High-level modeling and analysis of the traffic alert and collision avoidance system (tcas). *Proceedings of the IEEE*, 88(7):926–948, 2000. https://doi.org/10.1109/5.871302 doi:10.1109/5.871302.

[80] Jennifer Lundelius and Nancy A. Lynch. An upper and lower bound for clock synchronization. *Inf. Control.*, 62(2/3):190–204, 1984. https://doi.org/10.1016/S0019-9958(84)80033-9 doi:10.1016/S0019-9958(84)80033-9.

[81] John Lygeros and Nancy Lynch. Strings of vehicles: Modeling and safety conditions. In Thomas A. Henzinger and Shankar Sastry, editors, *Hybrid Systems: Computation and Control*, pages 273–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[82] N. Lynch. A hundred impossibility proofs for distributed computing. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, PODC '89, page 1–28, New York, NY, USA, 1989. Association for Computing Machinery. https://doi.org/10.1145/72981.72982 doi:10.1145/72981.72982.

[83] Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In Aurel Cornell and Dan Ionescu, editors, *Real=Time Systems: Modeling, Design, and Applications, AMAST Series in Computing*, volume 8. World Scientific Publishing Company, 2005.

[84] Nancy Lynch, Michael Merritt, William Weihl, and Alan Fekete. *Atomic Transactions*. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[85] Nancy Lynch and Alex Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In D. Malkhi, editor, *Distributed Computing: 16th International Symposium on DIStributed Computing (DISC 2002), Toulouse, France, October 2002*, volume 2508 of *Lecture Notes in Computer Science*, pages 173–190. Springer-Verlag, 2002. Also, Technical Report MIT-LCS-TR-856, MIT Laboratory for Computer Science, Cambridge, MA.

[86] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[87] Nancy A. Lynch and Hagit Attiya. Using mappings to prove timing properties. *Distributed Comput.*, 6(2):121–139, 1992. https://doi.org/10.1007/BF02252683 doi:10.1007/BF02252683.

[88] Nancy A. Lynch and Michael J. Fischer. On describing the behavior and implementation of distributed systems. *Theor. Comput. Sci.*, 13:17–43, 1981. https://doi.org/10.1016/0304-3975(81)90109-2 doi:10.1016/0304-3975(81)90109-2.

[89] Nancy A. Lynch, Nancy D. Griffeth, Michael J. Fischer, and Leonidas J. Guibas. Probabilistic analysis of a network resource allocation algorithm. *Inf. Control.*, 68(1-3):47–85, 1986. https://doi.org/10.1016/S0019-9958(86)80028-6 doi:10.1016/S0019-9958(86)80028-6.

[90] Nancy A. Lynch and Frederik Mallmann-Trenn. Learning hierarchically-structured concepts. *Neural Networks*, 143:798–817, 2021. https://doi.org/10.1016/j.neunet.2021.07.033 doi:10.1016/j.neunet.2021.07.033.

[91] Nancy A. Lynch and Frederik Mallmann-Trenn. Learning hierarchically-structured concepts II: overlapping concepts, and networks with feedback. In Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, and Dennis Olivetti, editors, *Structural Information and Communication Complexity - 30th International Colloquium, SIROCCO 2023, Alcalá de Henares, Spain, June 6-9, 2023, Proceedings*, volume 13892 of *Lecture Notes in Computer Science*, pages 46–86. Springer, 2023. https://doi.org/10.1007/978-3-031-32733-9_4 doi:10.1007/978-3-031-32733-9_4.

[92] Nancy A. Lynch, Yishay Mansour, and Alan D. Fekete. Data link layer: Two impossibility results. In Danny Dolev, editor, *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 149–170. ACM, 1988. https://doi.org/10.1145/62546.62572 doi:10.1145/62546.62572.

[93] Nancy A. Lynch, Michael Merritt, William E. Weihl, and Alan D. Fekete. *Atomic Transactions*. Morgan Kaufmann series in data management systems. Morgan Kaufmann, 1993.

[94] Nancy A. Lynch and Cameron Musco. A basic compositional model for spiking neural networks. In Nils Jansen, Mariëlle Stoelinga, and Petra van den Bos, editors, *A Journey from Process Algebra via Timed Automata to Model Learning - Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*, volume 13560 of *Lecture Notes in Computer Science*, pages 403–449. Springer, 2022. https://doi.org/10.1007/978-3-031-15629-8_22 doi:10.1007/978-3-031-15629-8_22.

[95] Nancy A. Lynch, Cameron Musco, and Merav Parter. Winner-take-all computation in spiking neural networks. *CoRR*, abs/1904.12591, 2019. URL: http://arxiv.org/abs/1904.12591, http://arxiv.org/abs/1904.12591 arXiv:1904.12591.

[96] Nancy A. Lynch and Calvin Newport. A (truly) local broadcast layer for unreliable radio networks. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 109–118. ACM, 2015. https://doi.org/10.1145/2767386.2767411 doi:10.1145/2767386.2767411.

[97] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, 2003. https://doi.org/10.1016/S0890-5401(03)00067-1 doi:10.1016/S0890-5401(03)00067-1.

[98] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. Observing branching structure through probabilistic contexts. *SIAM J. Comput.*, 37(4):977–1013, 2007. https://doi.org/10.1147/S0097539704446487 doi:10.1147/S0097539704446487.

[99] Nancy A. Lynch and Nir Shavit. Timing-based mutual exclusion. In *Proceedings of the Real-Time Systems Symposium - 1992, Phoenix, Arizona, USA, December 1992*, pages 2–11. IEEE Computer Society, 1992. https://doi.org/10.1109/REAL.1992.242681 doi:10.1109/REAL.1992.242681.

[100] Nancy A. Lynch and Eugene W. Stark. A proof of the kahn principle for input/output automata. *Inf. Comput.*, 82(1):81–92, 1989. https://doi.org/10.1016/0890-5401(89)90066-7 doi:10.1016/0890-5401(89)90066-7.

[101] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In Fred B. Schneider, editor, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1987*, pages 137–151. ACM, 1987. https://doi.org/10.1145/41840.41852 doi:10.1145/41840.41852.

[102] Nancy A. Lynch and Mark R. Tuttle. An introduction to Input/Output Automata. *CWI-Quarterly*, 2(3):219–246, September 1989. Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands. Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, November 1988.

[103] Nancy A. Lynch and Frits W. Vaandrager. Forward and backward simulations: I. untimed systems. *Inf. Comput.*, 121(2):214–233, 1995. https://doi.org/10.1006/inco.1995.1134 doi:10.1006/inco.1995.1134.

[104] Nancy A. Lynch and Frits W. Vaandrager. Forward and backward simulations, II: timing-based systems. *Inf. Comput.*, 128(1):1–25, 1996. https://doi.org/10.1006/inco.1996.0060 doi:10.1006/inco.1996.0060.

[105] Nancy Ann Lynch, Albert R. Meyer, and Michael J. Fischer. Relativization of the theory of computational complexity. *Transactions of the American Mathematical Society*, 220:243–287, 1976.

[106] Pratyusa K. Manadhata and Jeannette M. Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386, 2011. https://doi.org/10.1109/TSE.2010.60 doi:10.1109/TSE.2010.60.

[107] Keith Marzullo and Susan S. Owicki. Maintaining the time in a distributed system. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 295–305. ACM, 1983. https://doi.org/10.1145/800221.806730 doi:10.1145/800221.806730.

[108] Sayan Mitra, Yong Wang, Nancy Lynch, and Eric Feron. Safety verification of model helicopter controller using hybrid input/output automata. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control*, pages 343–358, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[109] Cameron Musco, Hsin-Hao Su, and Nancy A. Lynch. Ant-inspired density estimation via random walks. *Proc. Natl. Acad. Sci. USA*, 114(40):10534–10541, 2017. https://doi.org/10.1073/pnas.1706439114 doi:10.1073/pnas.1706439114.

[110] Cameron Musco, Hsin-Hao Su, and Nancy A. Lynch. Ant-inspired density estimation via random walks. *CoRR*, abs/1603.02981v2, 2019. URL: http://arxiv.org/abs/1603.02981v2, http://arxiv.org/abs/1603.02981v2 arXiv:1603.02981v2.

[111] Tina Nolte. *Virtual Stationary Timed Automata for Mobile Networks*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, February 2009.

[112] Tina Nolte and Nancy Lynch. Self-stabilization and virtual node layer emulations. In Toshimitsu Masuzawa and Sebastien Tixeuil, editors, *Stabilization, Safety, and Security of Distributed Systems: Proceedings of Ninth International Symposium (SSS 2007), Paris, France, November 2007*, volume 4838 of *Lecture Notes in Computer Science*, pages 394–408. Springer, 2007.

[113] Tina Nolte and Nancy Lynch. A virtual node-based tracking algorithm for mobile networks. In *International Conference on Distributed Computing Systems (ICDCS 2007)*, Toronto, Canada, June 2007.

[114] Susan S. Owicki and David Gries. An axiomatic proof technique for parallel programs I. *Acta Informatica*, 6:319–340, 1976. https://doi.org/10.1007/BF00268134 doi:10.1007/BF00268134.

[115] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980. https://doi.org/10.1145/322186.322188 doi:10.1145/322186.322188.

[116] Gary L. Peterson and Michael J. Fischer. Economical solutions for the critical section problem in a distributed system (extended abstract). In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 91–97. ACM, 1977. https://doi.org/10.1145/800105.803398 doi:10.1145/800105.803398.

[117] F.P. Preparata. *Advances in Computing Research*. Number v. 4 in Advances in Computing Research. JAI Press, 1987. URL: https://books.google.com/books?id=OMEnAQAAMAAJ.

[118] Tsvetomira Radeva, Anna R. Dornhaus, Nancy A. Lynch, Radhika Nagpal, and Hsin-Hao Su. Costs of task allocation with local feedback: Effects of colony size and extra workers in social insects and other multi-agent systems. *PLoS Comput. Biol.*, 13(12), 2017. https://doi.org/10.1371/journal.pcbi.1005904 doi:10.1371/journal.pcbi.1005904.

[119] Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1995. Also, MIT/LCS/TR-676.

[120] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR '94, Concurrency Theory, 5th International Conference, Uppsala, Sweden, August 22-25, 1994, Proceedings*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 1994. https://doi.org/10.1007/978-3-540-48654-1_35 doi:10.1007/978-3-540-48654-1_35.

[121] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. *Nord. J. Comput.*, 2(2):250–273, 1995.

[122] Mike Spindel. Simulation and evaluation of the reactive virtual node layer. Master of Engineering Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, May, 2008.

[123] Hsin-Hao Su, Lili Su, Anna R. Dornhaus, and Nancy A. Lynch. Ant-inspired dynamic task allocation via gossiping. In Paul G. Spirakis and Philippas Tsigas, editors, *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings*, volume 10616 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2017. https://doi.org/10.1007/978-3-319-69084-1_11 doi:10.1007/978-3-319-69084-1_11.

[124] Jennifer L. Welch, Leslie Lamport, and Nancy A. Lynch. A lattice-structured proof of a minimum spanning. In Danny Dolev, editor, *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 28–43. ACM, 1988. https://doi.org/10.1145/62546.62552 doi:10.1145/62546.62552.

[125] Jennifer L. Welch and Nancy A. Lynch. A new fault-tolerance algorithm for clock synchronization. *Inf. Comput.*, 77(1):1–36, 1988. https://doi.org/10.1016/0890-5401(88)90043-0 doi:10.1016/0890-5401(88)90043-0.

[126] Jiajia Zhao, Nancy A. Lynch, and Stephen C. Pratt. The power of population effect in *Temnothorax* ant house-hunting: A computational modeling approach. *J. Comput. Biol.*, 29(4):382–408, 2022. https://doi.org/10.1089/cmb.2021.0369 doi:10.1089/cmb.2021.0369.