# Term project assignment

**Proposal due:** Thursday, October 28, 1999
**Final writeup due:** Thursday, December 9, 1999
**Optional presentation:** Friday, December 10, 1999, 9AM - 1PM

## 1 Overview

An important requirement for 6.852 this year is a term project in which you are asked to *specify, model, and analyze a distributed algorithm of your own choosing.* Our bias is toward algorithms with practical interest, and especially toward algorithms used for reliable distributed computing or reliable communication. Also, you should choose an algorithm whose runtime behavior has some interesting logical subtleties: the algorithm should not be completely trivial, nor should it be complicated just because it handles a lot of boring bookkeeping. Rather, try to find an algorithm that does something reasonably interesting, in a reasonably interesting way.

The algorithm you choose can come from any of several possible sources: from your own research, from other courses, from descriptions of distributed systems you have learned about (or want to learn about), from the research literature on distributed systems, etc.

It is also possible for you to choose something from the literature on theoretical distributed algorithms. However, if you do this, make sure that the algorithm is not one that someone has already modeled and analyzed thoroughly. You can find one that has been presented only semi-formally, and do a more careful, complete job. Or, you can develop and study an improved version of a published algorithm.

The project involves specifying carefully some key properties that your algorithm is supposed to guarantee, modeling your algorithm's behavior precisely (using I/O automata or timed automata), and proving that your algorithm satisfies the claimed properties. These properties may involve safety, liveness, or performance.

You will be working on this project in a "small team" of one, two, or three people. We highly recommend a team size of two; three can be unwieldy, and one isn't much fun... Try to form your own teams, and ask us for help if you are having trouble.

## 2 Hints on defining a project

- Try to choose something that you already care about, like an algorithm you're already working on for a research project.
- Try to find an example where the key difficulties involve interesting dynamic behavior, rather than something static (e.g., how to place resources in order to optimize some graph-theoretic measure).
- Where you will spend your time on your project will depend in part on the kind of algorithm you choose. If you extract an algorithm from a real, complicated system, you may find yourself spending a great deal of time just specifying what the algorithm is and what it is supposed to do. If this turns out to be difficult, you may not have time to carry out complete correctness proofs and performance analyses, and will have to just sketch these. On the other hand, if

you choose an algorithm from the theoretical distributed algorithms literature, you may find it easy to model the algorithm and state the interesting properties, which will leave you more time to carry out complete proofs and analyses.

- If you are taking Prof. Balakrishnan's graduate course on networking, 6.892, concurrently with 6.852, it is possible for you to carry out a project that will satisfy both courses. In fact, this is a good opportunity to carry out a really in-depth study of a communication algorithm, using several methods at once.

  If you choose this option, we (and Prof. Balakrishnan) will expect you to choose a larger project than the usual one for either course separately. A good combination project might involve studying a proposed communication algorithm, modeling it and simulating it, proving its basic correctness properties, performing experiments to evaluate its performance, proving some performance properties, and comparing the proved performance with the observed performance.

## 3 Hints on how to specify and analyze a distributed algorithm

Many examples of algorithm specifications and proofs appear in the course text and in the papers cited during the course (especially those near the end). Use these as prototypes to guide you.[1]

- Start by identifying the properties you want to show that your algorithm satisfies. These may be safety, liveness, or performance properties. Some may involve fault-tolerance.
- You can describe properties by lists (as is done in most of the course text), or by a high-level automaton service specification. (See [FLS] for some examples of this.) If you use an automaton specification, make it as nondeterministic as you can.
- If your algorithm's correct behavior depends on any assumptions about the environment in which it runs, you will need to articulate those assumptions carefully. One useful device is to model the algorithm's environment as a very abstract automaton. This automaton only imposes the restrictions needed by the algorithm, and is otherwise completely nondeterministic.
- Model failures using failure input events. This allows you to state fault-tolerance properties as trace properties.
- State performance properties using conditional guarantees, conditioned on assumptions about the environment or the underlying system (e.g., "if no new process failures occur from some point on, then...", or "if all messages are delivered within time $d$, then...")
- Model the system using automata or timed automata (see Chapter 23), depending on whether it is important to consider timing assumptions and guarantees.
- Model all components at as high a level of abstraction as you can. In particular, if you choose an algorithm from a real system, you should not try to model it at a low level, close to the actual program code. (That will make the models ugly and the proofs infeasible.) Rather, express the idea of the algorithm in terms of high-level operations on high-level data types.

[1]A possible source of good hints for writing "good" specifications can be found in "Specifications of Concurrent and Distributed Systems" by William E. Weihl (Chapter 3 of "Distributed Systems", edited by Sape Mullender).

- Proofs can use operational reasoning, or better, invariants, simulation relations, and composition (pasting and projection) theorems.
- You should probably avoid probabilistic analysis where possible. Thus, instead of saying that an algorithm works well with high probability, you might say that it works well under certain conditions, e.g., "when there are at most $f$ process failures and when messages are always delivered within time $d$". Conditional guarantees provide more information than the summarizing probabilistic statements. Guaranteed amortized bounds, which talk about an average over occurrences in a single execution, are fine.
- Of course, if you consider a randomized algorithm, probabilistic claims will be unavoidable. In such a case, you should avoid trying to talk about the composition of several probabilistic systems. (We are not studying composition of probabilistic systems in this class.)

## 4 Some sources for project ideas

**Your own research:** If you are working on any distributed algorithms for your own research, especially any involving fault-tolerant distributed computing or fault-tolerant communication, use those.

**Research conference proceedings:** Look over proceedings of recent research conferences focusing, entirely or in part, on distributed algorithms or systems. Such conferences include:

1. The ACM PODC (Principles of Distributed Computing) conferences. The research presented at this conference generally has a theoretical flavor, but not all the algorithms are modeled and verified formally.

   For example, in PODC99, you might consider the paper "Resource Discovery in Distributed Networks" by Harchol-Balter, Leighton, and Lewin, or the paper "Fast Set Agreement in the Presence of Timing Uncertainty" by Michailidis.

2. ICDCS (the International Conference on Distributed Computing Systems) includes a track on distributed algorithms. It also contains a wide range of papers on not-so-theoretical aspects of distributed systems, many of which contain interesting algorithms.

   For example, in ICDCS99, you might consider "The Inter-Group Router Approach to Scalable Group Composition" by Johnson, Jahanian, and Shah; "NAP: Practical Fault-Tolerance for Itinerant Computations" by Johansen, Marzullo, Schneider, and Jacobsen; "Distributed Deadlock Detection and Resolution Based on Hardware Clocks" by Mayo and Kearns; "Fast and Fair Mutual Exclusion for Shared Memory Systems" by Huang; "Processing Transactions over Optimistic Atomic Broadcast Protocols" by Kemme, Pedone, Alonso, and Schiper; "Self-Stabilizing Neighborhood Synchronizer in Tree Networks" by Johnen, Alima, Datta, and Tixeuil.

3. The annual DISC (Distributed Computing) conference. Again, this has a theoretical flavor.

4. The biannual SOSP (Symposium on Operating Systems Principles) focuses on pragmatic, experimental results about system designs. It does contain many papers about interesting designs, which could provide good source material.

Specialty conferences on topics involving aspects of distributed computing, such as communication, security, or distributed data management, are also sources for interesting distributed algorithms.

**Courses:** MIT courses that cover some distributed algorithms include:

- 6.892, Networking

  This course covers many communication protocols, involving a wide range of current topics in communication. Routing and congestion management protocols should provide particularly good source material. Choose an algorithm, model it formally, state and prove correctness claims, and state and prove conditional performance claims.

- 6.826, Principles of Computer Systems

  This course includes designs for a wide range of types of systems, including multiprocessor and network caching systems, distributed transaction-processing systems, and communication systems. Many of the designs are expressed in the formal language SPEC. Such descriptions can provide a starting point for automaton modeling, proofs, and performance analysis.

- 6.857, Network and Computer Security

  Security protocols can be modeled using automata. However, you will have to use some special mechanisms for modeling limitations on knowledge and access—talk to us about this.

**Some other ideas:** Most of these ideas are related to current research projects in the Theory of Distributed Systems research group.

- Resource allocation. The paper "The Bancomat problem..." by Sussman and Marzullo, listed in course handout 4, describes a resource allocation algorithm and analyzes it using a new metric. This work could be done more formally. You can also consider whether the metric is the "right" one to use.

- Group membership. A new group membership algorithm by Keidar, Sussman, Marzullo, and Dolev seems to have real performance advantages over previous group membership algorithms. You could try to capture these advantages by formal (conditional) performance claims. (Contact: Idit Keidar, idish@theory.lcs.mit.edu)

- Group communication. The paper "Structured virtual synchrony: exploring the bounds of virtually synchronous group communication" by Guo, Vogels, and van Renesse (7th ACM SIGOPS European Workshop, 1996), gives a brief, informal description of an algorithm used in the Horus group communication system. You can work this out more carefully.

- Group communication. The paper "The Totem single-ring ordering and membership protocol" by Amir, Moser, Melliar-Smith, Agarwal, and Ciarfella, listed in course handout 4, describes some algorithms used in the Totem group communication system. These algorithms could be modeled using I/O automata, and some of their key properties identified and proved.

- Highly available servers. Papers by Anker, Dolev, and Keidar and by Mishra and Pang, listed in handout 4, describe strategies for implementing highly available servers. The algorithms do not seem hard; the difficult problem here seems to be specifying carefully what availability guarantees the algorithms make.

- Local consistency. The paper "Efficient state transfer in partitionable environments", by Amir, Chockler, Dolev, and Vitenberg, describes a simple protocol that repairs partitions in networks, bringing previous-separated processes back into "consistent" states. The notion of consistency is "local" in that it involves only the processes that are currently connected. Can you formulate a good definition for "local consistency" and show that this protocol satisfies it? (Contact: Idit Keidar, idish@theory.lcs.mit.edu)

- Data replication. A new data replication algorithm, implementing an atomic read/write register, has been developed by De Prisco. It has already been formalized in terms of I/O automata. A proof of correctness is needed. Performance analysis would also be interesting. (Contact: Roberto De Prisco, robdep@theory.lcs.mit.edu)

- Weakly consistent transaction memories. Implementations of multiprocessor memories are very much like distributed algorithms, and can be studied using the same modeling and analysis techniques. For example, Luchangco has designed a new type of memory that supports atomic transactions, but whose individual (read and write) operations satisfy somewhat weaker consistency guarantees than they do in atomic objects. He has a simple proposed implementation, using an "optimistic" algorithm. You could model this and prove its correctness. Alternatively, you could help develop and prove an algorithm that uses weakly consistent transaction memory plus some extra synchronization to implement strongly consistent (atomic) transactional memory. (Contact: Victor Luchangco, `victor_l@theory.lcs.mit.edu`.)

- Other multiprocessor memory implementations. You can study any of a variety of implementation algorithms from the multiprocessor architecture research literature, and verify that they implement particular types of memories. Recent work in Prof. Arvind's group provides a good source here.

- Security protocols. A key distribution protocol developed by Diffie, van Oorschot, and Weiner protects against "man-in-the-middle" attacks. (We have already modeled this protocol using automata.) Can you formulate and prove the algorithm's main guarantees? (Contact: Prof. Lynch)

- Distributed database commit. Designing commit protocols designed to work in real networks is harder than designing them to work in idealized synchronous network systems. Model a practical protocol for distributed database commit, and try to express its guarantees.

# 5 Your output

## 5.1 Proposal

This should be only one page. It should:

1. List your team members.

2. Identify the algorithm you want to consider, with citations or other information about where you are finding out about it.

3. Say roughly what the algorithm is intended to do (what problem does it solve?), and why you think it's interesting.

4. Describe briefly how the algorithm works.

5. List some interesting properties you will try to prove that the algorithm satisfies. (You will not be bound to prove exactly what you list in the proposal.)

### 5.2 Final writeup

This should be about 15 pages long. It should contain the detailed work proposed in the proposal. Specifically, it should:

1. Give a good informal description of the algorithm.
2. Give a formal description of properties you believe the algorithm satisfies.
3. Give a formal automaton model for the algorithm.
4. Prove that the algorithm satisfies the properties.

### 5.3 Presentation

This is optional. (Really.)

However, we have found that, after spending many hours modeling and analyzing an algorithm, many students would like to tell someone else about what they did. So we are providing a time slot at the end of the term where everyone can say a little about their projects. We'll supply refreshments.