# Distributed Algorithms for Dynamic Networks

March 18, 2014

Welles Robinson

Senior Thesis

Department of Computer Science

Georgetown University

**Abstract**

Abstract

## 1 High-Level Description of Goal

Ignore for now - The goal of this paper is to simplify the creation of distributed algorithms for dynamic networks by demonstrating that any algorithm that works for the broadcast variant of the synchronous model with a star topology can be made to work for the broadcast variant of the synchronous model with any topology. We will do so by describing a simulation algorithm that, if run on every node in the broadcast model, will match perfectly the output of the nodes of the centrally controlled model.

## 2 Models

Synchronous Broadcast Model

The synchronous broadcast model is a variant of the synchronous network model (definition taken from Lynch textbook - how to cite?). The synchronous network model is defined with respect to a directed graph $G = (V, E)$. We define $n$ to denote $|V|$, the number of nodes in the graph. An algorithm is a set of instructions to be followed by the nodes. When we say the network executes an algorithm $a$, this means each node in the network is running a copy of the $a$. (For simplicity, we refer to a copy of the algorithm running on node $u$ as simply node $u$.) In the execution, the nodes proceed in lock-step repeatedly performing the following two steps:

1. Following the algorithm, decide which messages, if any, to send to their neighbors in $G$.

2. Receive and process all incoming messages from their neighbors.

The combination of these two steps is called a round.

The synchronous broadcast model is different from the synchronous network model in two significant ways. First, the synchronous broadcast model is defined with respect to a connected graph $G = (V, E)$ with bi-directional edges. Second, nodes don't pass individual messages directly to their neighbors. Instead, nodes broadcast one message per round that is sent all neighbors.

Additionally, we assume that nodes have comparable unique identifiers and that nodes are in one of two high-level states, active or deactive. When a node is active, it performs the two steps, sending and receiving messages, that constitute a round. When a node is deactive, it performs neither of the two steps that constitute a round. We say a node is activated when its state changes from deactive to active. When a node is activated, it always begins in an initial state such that it has no knowledge of a global round counter. We say a node is deactivated when its state changes from active to deactive. When a node is deactivated, it resets all local variables such that if it activates, it activates in an initial state.

The only restriction that we place on the activation and deactivation of nodes in G is that the active subset of the graph G must always be connected. We emphasize that other than this minimal restriction the activation and deactivation of nodes are uncontrolled by the algorithm.

# 3 Problem Definition

The reliable broadcast problem provides arbitrary nodes in the broadcast model with messages to send to all active nodes in the network. This problem contains an environment that communicates with individual nodes through a send and receive interface implemented by the node. The environment passes a message to be sent to a node and that node sends a "done" signal to the environment through the send interface. We assume the environment will wait until it has received a done message from a node before giving it another message to send. The receive interface is used by a node to notify the interface that it received a specific message.

An algorithm $a$ that solves the reliable broadcast problem will implement the send and receive interface. An algorithm is said to solve the reliable broadcast problem if it satisfies the following properties:

1. Liveness Property: If a node $u$ running algorithm $a$ is passed a message to send through its send interface, it will eventually respond with a "done" signal to the environment.

2. Safety Property #1: Assume node $u$ is passed a message $m$ throught its send interface at round $r$ and it replies with a "done" signal in some later

round $r_1$. Let $A(r, r_1)$ be the set of nodes that are active in every round in the interval from $r$ to $r_1$. It must be the case that every node in $A(r, r_1)$, with the exception of $u$, passes message $m$ to the environment through its receive interface at some point between rounds $r$ and $r_1$.

3. Safety Property #2: Assume some node $u$ passes a message $m$ to its environment through its receive interface in some round $r$ and then passes a different message $m_1$ in a later round $r_1$. It follows that there is no node $v$ that passes up message $m_1$ before message $m$.

4. Safety Property #3: Assume some node $u$ passes a message $m$ to its environment through its receive interface, the following two conditions must hold:

   (a) $u$ has not previously passed $m$ to the environment

   (b) some node previously received $m$ from the environment through its send interface

# 4 Algorithm

1. Run unmodified Leader Election without Network Information

   (a) Nodes have UIDs

   (b) Each process runs Terminating Synchronous Breadth First Search and the node that manages to terminate elects itself leader and tells the other nodes to terminate

2. The Simulation

   (a) Assume that each and every round of the given algorithm has a finite repetitions of a step, which is comprised of two parts

      i. The star node sends a broadcast message to all the leaf nodes (Broadcast part)

      ii. All leaf nodes send a receive message to the star node in response to the broadcast message (Receive part)

   (b) Each step is simulated by the leader node running a modified instance of terminating synchronous BFS

      i. The search message sent by a parent node to its children nodes is modified to be "search" plus the broadcast message, which is dictated by the output of the input algorithm A1

      ii. Upon receiving a search message, a node runs the input algorithm A2 on the broadcast message portion and then creates the receive message portion using the output of A2

iii. The done message sent by a child node to its parent node is modified to be “done” plus both the UID and the receive message of the child node as well as any done messages received by the child node
iv. Upon receiving a search message, a node writes the broadcast message portion to its communication log
v. Upon sending a done message, a node writes the receive message portion to its communication log
vi. The leader simulates sending a search message to itself and then simulates sending a done message back to itself
vii. This algorithm terminates when the leader has received done messages from all of its children and a simulated done message from itself

Static Model - All the nodes turn on at the same time

Simulation Algorithm takes one input, algorithm A, the algorithm to be simulated. Algorithm A can be broken into two distinct algorithms, A1, the algorithm run by the star process, and A2, the algorithm run by the leaf processes.

Member Variables - maxID (UID); parent (UID); totalChildren (int); childCount (int); wait (int); Message has a root (a round, the UID); a id of the sender (UID); a type search, choose, done; a receiver (UID), defaults to NULL;

Dynamic Addition Model - Nodes turn on at various times but they don’t turn off The leader elected will be the node with the highest UID out of all of the nodes that turned on at round 1

Variables - maxRoot - (a round, the UID); parent (UID); totalChildren (int); childCount (int); wait (int); Message has a root (a round, the UID); a id of the sender (UID); a type search, choose, done; a receiver (UID);

# 5 Analysis

**Lemma 5.1.** *For the given network, a node will eventually set leader to true and no more than one node will have leader equal to true at the beginning of any round $r$.*

*Proof.* One node will eventually set leader to true (Lemma 5.4). No more than one node will have leader set to true at any point (Lemma 5.2). □

**Definition 1.** *Let $u_{max}$ be the ID of the process with the maximum UID in the network.*

**Definition 2.** *Let BFS instance $b_i$ refer to an instance of the terminating breadth-first search protocol initiated by process with ID $i$.*

**Lemma 5.2.** *For every round $r$, at most one node has leader = true at the beginning of round $r$.*

*Proof.* A node with ID $i$ will only set leader = true if the BFS instance $b_i$ terminates. A BFS tree $b_i$ will terminate only if $i$ equals $u_{max}$ (Lemma 5.3). Only the process with ID $u_{max}$, will set leader = true.

□

**Lemma 5.3.** *A BFS instance $b_i$ will only terminate if $i$ equals $u_{max}$.*

*Proof.* Termination of a BFS instance $b_j$ requires all other processes in the network to send a done message to $b_j$. Given BFS instance $b_j$ where $j < u_{max}$, there is at least one process, the process with ID $u_{max}$, that will never reply done to $b_j$. Therefore, $b_j$ will never terminate. □

**Lemma 5.4.** *One node will eventually set its variable leader to true.*

*Proof.* A BFS instance $b_i$ will eventually terminate if every node in the network runs $b_i$. Every node in the network will eventually run $b_{u_{max}}$ so $b_{u_{max}}$ will eventually terminate and the process with ID $u_{max}$ will set leader = true. □

**Definition 3.** *For a given node, define $r_t$ as the consecutive rounds of the main execution that directly map to round $t$ of the reference execution.*

**Lemma 5.5.** *For a given node and any round $t$ in the reference execution, $r_t$ exists.*

*Proof.* For the star node in the reference execution, the simulation of one round of its execution is defined as beginning when $u_{max}$ broadcasts its message and ending when $u_{max}$ has received messages from all of its children. For any given child node $a$ in the reference execution, the simulation of one round of its execution is defined as between when $a$ broadcasts its message and when $a$ receives the leader's message from its parent. □

**Definition 4.** *For a given node $a$ in the reference execution, define $s_a$ as the node in the main execution that simulates $a$.*

**Lemma 5.6.** *By induction, the main execution correctly simulates the reference execution for any round $q$.*

*Proof.* By Induction.

Base Case $q = 0$: Before round 0, the main execution has correctly simulated the reference execution because all the communication logs are empty and therefore equivalent.

Inductive Hypothesis: Suppose the theorem holds for all values of $q$ up to $k$.

Inductive Step: Let $q = k + 1$. In round $k$ of the reference execution, the star node sends message $m$ to every child node and records 'sent $m$' in its log. In the main execution, $u_{max}$ broadcasts $m$ and records 'sent $m$' in its log. In the reference execution, every child node receives $m$ and records 'received $m$' in its log. In the main execution, the children of $u_{max}$ receive $m$, record 'received $m$' in its log, and broadcast $m$ to their children. Eventually, every node in the main execution network receives $m$ and records 'received $m$' in its log.

In the reference execution, a given child node $a$ sends message $m_1$ to the star node and records 'sent $m_1$' in its log. The star node receives $m_1$ and records 'received $m_1$' in its log. In the main execution, $a$ eventually broadcasts $m_1$ to its parent and records sent $m_1$ in its log. $a$'s parent receives $m_1$, which is broadcasted up the network until it eventually reaches $u_{max}$, which records 'received $m_1$' in its log.

After round $k + 1$, the list of the logs of the reference execution will be equivalent to that of the main execution. So the theorem holds for $q = k + 1$. By the principle of mathematical induction, the theorem holds for all rounds in the execution.

□

```
initVariables() ;
for round 1 ... r do
    for each message m in Inbox do
        if m.maxID > maxID then
            updateMaxRoot() ;
        end
        if m.maxID == maxID then
            if m.type == choose AND receiver == myUID then
                childCount++ ;
                totalChildren++ ;
            end
            if m.type == done AND receiver == myUID then
                childCount– ;
                if childCount == 0 then
                    sendDoneMsg( ) ;
                end
            end
        end
        if m.maxID < maxID then
            msg = (type=search, sender=myUID, maxID=maxID) ) ;
            Outbox.enqueue(msg) ;
        end
    end
    if wait != 0 AND childCount == 0 then
        wait– ;
        if wait == 0 then
            sendDoneMsg( ) ;
        end
    end
    for each message m in Outbox do
        broadcast(m);
    end
    myRound++ ;
end
```

**Algorithm 1:** Simulation Algorithm for Static Model

```
myRound == 0 ;
maxID = myUID ;
message m = (type=search, sender=myUID, maxID=maxID) ;
```

**Algorithm 2:** initVariables method

```
maxID = m.maxID;
parent = m.sender;
childCount = totalChildren = 0; msg1 = (type=choose, sender=myUID,
maxID=maxID, receiver=m.sender) ;
Outbox.enqueue(msg1) ;
msg2 = (type=search, sender=myUID, maxID=maxID) ;
Outbox.enqueue(msg2) ;
wait = 3 ;
```

**Algorithm 3:** updateMaxRoot method

```
msg = ( type=done, sender=myUID, maxID=maxID ) ;
Outbox.enqueue( msg ) ;
```

**Algorithm 4:** sendDoneMsg method

```
for each action in A1 do
    if leader == true then
        message = (type=r.action, sender=myUID) ;
        broadcast ( message ) ;
        msg = ( response to message ) ;
        commLog.write (response to message) ;
    end
    for each round r do
        for each message m in Inbox do
            if m.sender == parent then
                forwardMsgToChildren( ) ;
            end
            if m.receiver == myUID then
                add m to msg ;
                childCount– ;
                if childCount == 0 then
                    Outbox.enqueue(msg) ;
                end
            end
        end
        for each message m in Outbox do
            broadcast(m) ;
        end
    end
end
```

**Algorithm 5:** Static Simulation Algorithm

msg = (response to m, receiver = parent) ;
commLog.write(response to m) ;
**if** *totalChildren == 0* **then**
    Outbox.enqueue(msg) ;
**else**
    forwardMsg = (type = m.type, sender=myUID) ;
    Outbox.enqueue(forwardMsg) ;
**end**

**Algorithm 6:** forwardMsgToChildren

```
initVariables() ;
for round 1...r do
    for each message m in Inbox do
        if m.root > maxRoot then
            updateMaxRoot() ;
        end
        if m.root == maxRoot then
            if m.type == choose AND receiver == myUID then
                childCount++ ;
                totalChildren++ ;
            end
            if m.type == done AND receiver == myUID then
                childCount– ;
                if childCount == 0 then
                    sendDoneMsg() ;
                end
            end
        end
        if m.root < maxRoot then
            msg = (type=search, sender=myUID, root=(r=maxRoot.r+1,
            id=maxRoot.id) ) ;
            Outbox.enqueue(msg) ;
        end
    end
    if wait != 0 AND childCount == 0 then
        wait– ;
        if wait == 0 then
            sendDoneMsg() ;
        end
    end
    for each message m in Outbox do
        broadcast(m);
    end
    myRound++ ;
    maxRoot = (r=maxRoot.r+1, id=maxRoot.id) ;
end
```

**Algorithm 7:** Simulation Algorithm for the Dynamic Addition Model

```
myRound == 0 ;
maxRoot = (r=myRound, sender=myUID) ;
message m = (type=search, id=myUID, root=maxRoot) ;
```

**Algorithm 8:** initVariables method for Dynamic Addition Model

maxRoot = m.root;
parent = m.sender;
childCount = totalChildren = 0; msg1 = (type=choose, sender=myUID, root=(r=maxRoot.r+1, id=maxRoot.id), receiver=m.id) ;
Outbox.enqueue(msg1) ;
msg2 = (type=search, sender=myUID, root=(r=maxRoot.r+1, id=maxRoot.id) ) ;
Outbox.enqueue(msg2) ;
wait = 3 ;

**Algorithm 9:** updateMaxRoot method for Dynamic Addition Model

msg = ( type=done, sender=myUID, root=(r=maxRoot.r+1, id=maxRoot.id), receiver=parent ) ;
Outbox.enqueue( msg ) ;

**Algorithm 10:** sendDoneMsg method for Dynamic Addition Model