Distributed Algorithms for Dynamic Networks

April 9, 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 messages to arbitrary nodes in the synchronous broadcast model to send to all active nodes in the network. This problem assumes there is an environment at each node u that communicates with u through an interface with three commands, *send*, *receive* and *acknowledge*. We refer to the environment at node u as E_u.

Using the *send* command, E_u can pass a message m to u, which u is expected to send to all other nodes in the network. Once all the other nodes have received m, u is expected to pass a "done" signal to E_u using the *acknowledge* command. We assume E_u will not pass another message to u until it has received a "done" signal from u. When a node u receives a message m from another node, it uses the *receive* command to notify E_u about m.

An algorithm A is said to solve the reliable broadcast problem if it implements the *send*, *receive* and *acknowledge* commands and satisfies the following properties (assume all messages are unique):

1. Liveness Property: If a node u running algorithm $\mathcal{A}$ is passed a message by E_u through its *send* command, u will eventually send a "done" signal to E_u using the *acknowledge* command.

2. Safety Property #1: Assume node u is passed a message m by E_u at round r and u sends a "done" signal in some later round r'. Let $A(r, r')$ be the set of nodes that are active in every round in the interval from r to r'. It must be the case that every node in $A(r, r')$ passes message m to its environment through its *receive* command at some point between rounds r and r'. In addition, no node, including those not part of $A(r, r')$, will pass m to its environment after round r'.

3. Safety Property #2: Assume some node u passes a message m to its environment through its *receive* command in some round r and then passes a different message m' in a later round r'. It follows that no node in the network passed message m' to its environment before message m.

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

    (a) u has not previously passed m to its environment

    (b) some node previously received m from its environment through its *send* command

4 Algorithm

4.1 Reliable Broadcast with Simultaneous Activation

The first step of the algorithm is to run the leader election protocol, which involves building a spanning tree in the network and electing a leader of the tree. When a node activates, it starts running an instance of terminating breadth-first search (BFS) whose source is its ID. If a node hears of a BFS instance whose source is less than the source of its current BFS instance, it will cease running its current instance and start running the new instance. When an instance of BFS terminates successfully, the source node of that instance will elect itself leader of the tree and broadcast a confirm message through the tree. Only one instance of BFS will terminate so only one node will elect itself leader. When a node receives the confirm message, it becomes a confirmed member of the spanning tree. The spanning tree refers specifically to the tree whose source has elected itself leader. While a node can receive a *send* command from its environment at any point of execution, it won't begin the protocol (described below) for distributing the message throughout the tree until it has become a confirmed member of the tree.

A node u receives a message m from E_u using the *send* command. u broadcasts message m with the instructions that only its parent should forward m. Each node that forwards m keeps track of which child sent it m. m continues to be passed up the tree from child to parent until it reaches the leader of the tree, u_{max}. m is guaranteed to reach u_{max} because u_{max} is an ancestor of every

node in the tree. When u_{max} receives a message, it adds that message to its send queue.

If another message is not being broadcast through the tree, u_{max} will dequeue the next message m, if any exists, off its send queue. Next, u_{max} will notify its environment of message m using the *receive* command and then broadcast m. All of the children of u_{max} will receive m and perform the same two actions as u_{max}, notifying their respective environments and broadcasting m, and in this way, m will eventually be passed by every node to its environment. Every node will pass m to its environment only once because a node will only process m when it is sent by its parent and a node can only have one parent in the tree. A node that receives m and doesn't have any children will send a message to their parent confirming that they have received message m. When a node has received confirmation messages from all of its children, it will send a message to its parent confirming that it (and all its children) have received m. When u_{max} has received confirmation messages from all of its children, it will perform the following two actions simultaneously. u_{max} will dequeue the next message m', if any exists, off its send queue and repeat the above protocol.

Additionally, u_{max} will send a message to u telling it that all nodes in the network have received m. This message to u follows the same path as the original message from u to u_{max} as each node broadcasts it with the instruction that it only should be forwarded by the specified node, the child that sent it to the parent. When u receives this message, it notifies its environment using its *acknowledge* command.

4.2 Reliable Broadcast with Staggered Activation

The algorithm for reliable broadcast with simultaneous activation works for reliable broadcast with staggered activation with the following modifications.

We assume that every node knows the current global round. This assumption was unnecessary in the simultaneous activation case because the local round and the global round were identical.

In the leader election protocol, the source of a BFS instance started by node u is the combination of the global round when u activated and the ID of u. A node will choose to run the BFS instance whose source has the lowest global round of activation. If the sources of multiple BFS instances have the same global round of activation, a node will choose the BFS instance whose source has the lowest ID. By the definition of the model, nodes have unique IDs so multiples BFS instances won't have the same ID.

Once the leader has been elected and sent a confirm message through the tree, it is still possible for nodes to activate. When a node is a confirmed member of the spanning tree, it broadcasts a message at the beginning of every round declaring that it is part of the spanning tree and telling its neighbors to join. When a node u activates, it follows the leader election protocol and attempts to start a tree with itself as source. If u has a neighbor v that is part of the spanning tree, u will receive v's message, join the spanning tree and choose v as its parent in the tree. If u does not have a neighbor that is part of the spanning

4

tree, it is connected to a node belonging to the tree by the defintion of the model and eventually one of its neighbor will join the spanning tree and then u will join the spanning tree.

4.3 Reliable Broadcast with Staggered Activation and De-activation

The algorithm for reliable broadcast with staggered activation and deactivation is completely different from the algorithm for reliable broadcast with only activations.

In contrast, the deactivation case does not involve electing a leader or building a stable spanning tree. Instead, a node floods the network with the message and the exact global round to execute the *receive* command on the message.

A node u receives a message m from its environment at round r using the *send* command. u broadcasts message m, with the instructions that every node that receives m should execute the *receive* command on m at a specified round r' where $r' = r+n$ where n is an upper bound on the number of nodes in the network. u will execute the *acknowledge* command on m at round $r' + 1$.

If a node has multiple *receive* commands to execute in the same round, it executes them in ascending order of the UID of the source node of the message, which is attached to the message. Every node maintains an internal list of messages. At the beginning of every round, every node will broadcast all of the messages. When a node executes a *receive* command on a message, it removes that message from its internal list. When a node hears a broadcast about a message for the first time, it adds that message to its internal list.

5 Analysis

5.1 Leader Election with Simultaneous Activation

Lemma 5.1. *A node will eventually elect itself leader and no more than one node will have leader equal to true at the beginning of any round.*

Proof. One node will eventually elect itself leader (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 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. $\square$

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. $\square$

5.2   Reliable Broadcast with Simultaneous Activation

Theorem 5.5. *The algorithm solves the reliable broadcast problem with simultaneous activation.*

Proof. This algorithm satisfies the liveness property(Lemma 5.6), the first safety property(Lemma 5.7), the second safety property(Lemma 5.8) and the third safety property(Lemma 5.9) of the reliable broadcast problem with simultaneous activation. $\square$

Lemma 5.6. *A message m that is broadcast by the leader through the spanning tree formed by the leader election protocol will eventually be seen by every node and the leader will eventually receive a confirmation message that all nodes have seen m.*

Proof. m will be broadcast by the leader and received by all of the leader's children. The children will similarly broadcast m. Eventually, every node in the tree will receive m because every node in the tree is a descendant of the leader.

By the definition of the algorithm, a leaf node will send a confirm message to its parent upon receiving m. By definition of a tree, every non-leaf node has a descendant that is a leaf node. As a result, every non-leaf node, including the leader, will eventually receive a confirm message from its children.

$\square$

Lemma 5.7. *The algorithm satisfies the liveness property of the reliable broadcast problem with simultaneous activation.*

Proof. Assume node u running algorithm $\mathcal{A}$ is passed a message m by E_u through its *send* command. When u is a confirmed member of the spanning tree, it sends m to the leader of the tree. There will be a stable path between u and the leader so m is guaranteed to reach the leader. When m reaches the leader, it is placed in its send queue. Eventually, m reaches the front of the send queue and is dequeued by the leader. The leader broadcasts m to the network and eventually receives a confirm message that all the nodes in the network have seen m (Lemma **??**). When the leader has received confirmation messages from all of its children, it will notify u using the same stable path. When u

6

receives this message from the leader, it sends a "done" signal to E_u using the *acknowledge* command.

□

Lemma 5.8. *The algorithm satisfies the first safety property of the reliable broadcast problem with simultaneous activation.*

Proof. In the simultaneous activation case, $A(r, r')$ is the set of all nodes in the network so the first safety property requires that all nodes in the network receive m between round r and r'. Assume node u is passed a message m by E_u at round r and u sends a "done" signal in some later round r'. u will first send m to the leader of the tree. The leader will broadcast m at some round after r. The leader eventually receives a confirmation that all nodes have seen m and passed m to their environments through their *receive* command (Lemma **??**). Then, the leader sends a confirm message to u, which then sends a "done" signal at round r' to E_u using the *acknowledge* command.

□

Lemma 5.9. *The algorithm satisfies the second safety property of the reliable broadcast problem with simultaneous activation.*

Proof. Safety Property #2: Assume some node u passes a message m to its environment through its *receive* command in some round r and then passes a different message m' in a later round r'. It follows that no node in the network passed message m' to its environment before message m.

When a node is passed a message m by its environment, it sends m to the leader of the tree. When the leader receives m, it can either pass m to its environments through its *receive* command and broadcast m to the rest of the tree or it can place m in its queue if another message is passing through the tree. As a result, the order of messages received is dictated by the leader and the leader will always wait for every node to have received one message before sending another message. □

Lemma 5.10. *The algorithm satisfies the third safety property of the reliable broadcast problem with simultaneous activation.*

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

1. u has not previously passed m to its environment

2. some node previously received m from its environment through its *send* command

Per the algorithm, nodes only pass a message m to their environment when they receive m from their parent in the tree. A given node has only one parent and so only receives m once. □

5.3 Reliable Broadcast with Staggered Activation

Theorem 5.11. *The algorithm solves the reliable broadcast problem with staggered activation.*

Proof. This algorithm satisfies the liveness property(Lemma 5.11), the first safety property(Lemma 5.12), the second safety property(Lemma 5.15) and the third safety property(Lemma 5.16) of the reliable broadcast problem with staggered activation. □

Lemma 5.12. *The algorithm satisfies the liveness property of the reliable broadcast problem with staggered activation.*

Proof. Liveness Property: If a node u running algorithm $\mathcal{A}$ is passed a message by E_u through its *send* command, u will eventually send a "done" signal to E_u using the *acknowledge* command.
(Lemma 5.6) □

Lemma 5.13. *The algorithm satisfies the first safety property of the reliable broadcast problem with staggered activation.*

Proof. Safety Property #1: Assume node u is passed a message m by E_u at round r and u sends a "done" signal in some later round r'. Let $A(r, r')$ be the set of nodes that are active in every round in the interval from r to r'. It must be the case that every node in $A(r, r')$ passes message m to its environment through its *receive* command at some point between rounds r and r'.

Assume node u is passed a message m by E_u at round r and u sends a "done" signal in some later round r'. In the staggered activation case, $A(r, r')$ is identical to $A(r)$ which is the set of all nodes that have activated by round r. If all the nodes in $A(r, r')$ are part of the spanning tree of by round r, this is identical to the first safety property in the case of simultaneous activation given that the activation of any nodes won't affect the path between a node and the leader (Lemma 5.14).

Otherwise, the nodes that aren't part of the tree before round r will become part of the tree in time to receive the message (Lemma 5.13).
□

Lemma 5.14. *By induction, any node u that activates before round r and is q hops from the tree will receive a message m that is sent to any given node from an environment during round r.*

Proof. By Induction.

Base Case $q = 0$: A node u that is zero hops away from the tree is part of the tree and will receive the message m (Lemma 5.7).
Base Case $q = 1$: A node u that is one hop from the tree before round r will be a confirmed member of the tree before round $r + 2$. During round r, u receives a broadcast from v, its neighbor that is part of the tree, telling it to join the tree. During round $r + 1$, u responds to v telling v that it has chosen v as its parent.

As a result, before round $r + 2$, u is a confirmed member of the tree. By the algorithm, once a node receives a message to send to the rest of the tree, that node waits two additional rounds before sending it to its children or replying done to its parent if it has no children.

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

Inductive Step: Let $q = k + 1$.

v, the node that is k hops from the tree at round r, receives m at a later round r'. For v to have received m in round r', it must have been a confirmed member of the tree in round r'. As a confirmed member of the tree, v must have broadcast a message telling its neighbors to join the tree. u, which is a neighbor of v, receives this message in round r' and is a confirmed member of the tree by round $r' + 1$. As the parent of u, v will broadcast m to u in round $r' + 3$ after the two round wait period. $\qquad\square$

Lemma 5.15. *The path from the leader of a tree to any node in the tree will not change with the activation of any node.*

Proof. Every node is connected to the leader through their parent and nodes do not change their parent once they have joined the tree. $\qquad\square$

Lemma 5.16. *The algorithm satisfies the second safety property of the reliable broadcast problem with staggered activation.*

Proof. Safety Property #2: Assume some node u passes a message m to its environment through its *receive* command in some round r and then passes a different message m' in a later round r'. It follows that no node in the network passed message m' to its environment before message m.

(Lemma 5.8)

$\qquad\square$

Lemma 5.17. *The algorithm satisfies the third safety property of the reliable broadcast problem with staggered activation.*

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

1. u has not previously passed m to its environment

2. some node previously received m from its environment through its *send* command

(Lemma 5.9) $\qquad\square$

5.4 Reliable Broadcast with Activations and Deactivations

Theorem 5.18. *The algorithm solves the reliable broadcast problem with deactivations.*

Proof. This algorithm satisfies the liveness property(Lemma 5.18), the first safety property(Lemma 5.20), the second safety property(Lemma 5.21) and the third safety property(Lemma 5.22) of the reliable broadcast problem with activations and deactivations. □

Lemma 5.19. *The algorithm satisfies the liveness property of the reliable broadcast problem with staggered activation.*

Proof. Liveness Property: If a node u running algorithm $\mathcal{A}$ is passed a message by E_u through its *send* command, u will eventually send a "done" signal to E_u using the *acknowledge* command if u doesn't deactivate.

By the algorithm, u will send a "done" signal to E_u using the *acknowledge* command after n+1 rounds. The only way u will not send the "done" signal is if u deactivates, which is acceptable under liveness. □

Lemma 5.20. *Every node in $A(r, r')$ sees m within n rounds*

Proof. By the model definition, the network is guaranteed to have the stability property of at worst 1-interval connectivity. In a network with 1-interval connectivity where nodes broadcast every round, a message is guaranteed to be seen for the first time by at least one node every round if there exists any node in the network has not seen the message (Distributed Computing in Dynamic Networks). After $n - 1$ rounds of broadcasting message m in a network with 1-interval connectivity where n is the total number of nodes in the network, every node that was active for all $n - 1$ rounds will have seen m.
□

Lemma 5.21. *The algorithm satisfies the first safety property of the reliable broadcast problem with staggered activation.*

Proof. Safety Property #1: Assume node u is passed a message m by E_u at round r and u sends a "done" signal in some later round r'. Let $A(r, r')$ be the set of nodes that are active in every round in the interval from r to r'. It must be the case that every node in $A(r, r')$ passes message m to its environment through its *receive* command at some point between rounds r and r'.

Every node in $A(r, r')$ sees m within n rounds (Lemma 5.19). By definition of the algorithm, every node that receives m passes m to its environment during round $r' - 1$. Therefore, every node in $A(r, r')$ passes m to its environment during round $r' - 1$, which is between round r and round r'.
□

Lemma 5.22. *The algorithm satisfies the third safety property of the reliable broadcast problem with deactivation.*

Proof. Safety Property #2: Assume some node u passes a message m to its environment through its *receive* command in some round r and then passes a different message m' in a later round r'. It follows that no node in the network passed message m' to its environment before message m.

By definition of the algorithm, every node that receives a message will pass that message to its environment at a pre-determined round. Given two messages, m associated with round r_1 and m' associated with round r_2, all nodes will pass m and m' to their environments in the same order. There are two fundamental cases, when r_1 and r_2 are equal and when r_1 and r_2 are not equal.

If r_1 is less than r_2, all nodes that receive both m and m' will pass m to their environment before passing m'. If r_2 is less than r_1, all nodes that receive both m and m' will pass m' to their environment before passing m'.

If r_1 is equal to r_2, then every node will use the UID attached to the message to determine the order of execution. If $m_{sourceID}$ is greater than $m'_{sourceID}$, then every node executes m before m'. If $m'_{sourceID}$ is greater than $m_{sourceID}$, then every node executes m' before m. $m'_{sourceID}$ cannot be equal to $m_{sourceID}$ because a single node cannot get a message from its environment while it has an unacknowledged message out there.

□

Lemma 5.23. *The algorithm satisfies the third safety property of the reliable broadcast problem with staggered activation.*

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

1. u has not previously passed m to its environment

2. some node previously received m from its environment through its *send* command

(Lemma 5.9)

□