Distributed Algorithms for Dynamic Networks

April 18, 2014

Welles Robinson

Senior Thesis

Department of Computer Science

Georgetown University

Abstract

Abstract

1 Introduction

In the last decade, the importance of distributed computing has grown almost as quickly as the amount of data being generated. Distributed systems, like the open-source Hadoop Distributed File System, have revolutionized how we handle and process large amounts of data efficiently. Content distribution networks, like Akamai, have used distributed algorithms to drastically increase the performance and robustness of the Internet. Peer-to-peer networks, like BitTorrent, rely on distributed algorithms and have revolutionized entire industries like the music industry.

Reflecting these trends, the majority of the work in distributed computing has focused on what we term static distributed networks, which assume a relatively constant network topology and connectivity. These models are better equipped to deal with a node failing than changes to the topology of the network. This choice is logical when the goal is to build a peer-to-peer network which is overlaid on top of the Internet, which ensures reliable connections between nodes. Similarly, this choice makes sense in the context of big data centers where computers are hard wired to each other.

However, if distributed algorithms are to play a large role in the exciting future of wireless technology, it is imperative to adopt new approaches and models that embrace dynamic topologies. Dynamic topology captures the idea that not only network participants and topology aren't known ahead of time

but also that they may change significantly during execution. There exist many potential applications for distributed systems in this growing field. Take for example, the growing phenomenon known as the "Internet of things" where more and more household devices gain wireless capabilities. As many of these devices move locations frequently and may move out of direct range of a router, it will become necessary to have distributed algorithms for reliable, wireless coordination for dynamic topologies.

The rise of powerful smartphones portend another enormous need for distributed algorithms that can handle mobile nodes connected wirelessly. Currently, nearly all wireless communication happens in a centralized manner where phone communicate via cellular towers. Utilizing the principles of distributed computing would increase the range of these towers and increase the performance and efficiency of cellular networks an important consideration given the increasing strain on these networks. Even data centers, long the domain of static distributed computing, have begun experimenting with wireless connectivity to reduce power use and cost. The potential of distributed algorithms on the growing field of wireless technology mandate significant work on algorithms for distributed networks with dynamic toplogies.

This paper examines the reliable broadcast problem in distributed networks with dynamic topologies. It requires nodes to pass messages to the network and reach consensus on the proper order to store these messages. The reliable broadcast problem is a key primitive for implementing replicated state machines. The replicated state machine approach, which involves creating multiple replicas of each process so that the network can recover from process failures, is the general approach for implementing fault-tolerant services in distributed systems.

This paper proposes provably correct solutions to the reliable broadcast problem in three different distributed network settings. These network settings stand in contrast to the those used in existing papers that examine mobile nodes in distributed systems by Brown et al [**?**] and Chockler et al [**?**] that make assumptions about prior knowledge of network topology that aren't practical in the real world. All the settings, particularly the third, used in this paper allow a much more dynamic network topology and consequently offer a much more practical model for the real world.

The first setting assumes that all nodes in the network activate at the same time and remain active for the duration of the execution. The second second allows nodes to activate at different points during execution but assumes that once a node activates, it will remain active for the duration of the execution. While the first two settings don't allow particularly dynamic topologies, they demonstrate the best-case efficiency and performance in a well-behaved network. The final setting allows nodes to activate and deactivate at will and as a result, permits a very dynamic topology.

The major contributions of this thesis are as follows. First, this thesis provides a correct and useful definition for the problem of reliable broadcast in distributed networks. Second, we provide new, provably correct algorithms that solve the problem of reliable broadcast in the model for the three dynamic topologies. Third, these algorithms use general strategies that will be useful for

the development of other algorithms in this model. Fourth, we provide simulation results of the algorithms in real world networks to explore average case performance beyond the formally proven worst-case bounds.

2 Model

We consider a broadcast variant of the standard synchronous message passing model of distributed computation [**?**, **?**]. 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 for the first time, it always begins in an initial state with 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 maintains all local variables such that if it reactivates, it has knowledge of its previous state. The ability to retain previous state knowledge after deactivations leaves the decision of how to treat the reactivation of a node up to the algorithm.

In this thesis, we place three sets of restrictions on the behavior of the nodes. In all cases, the active subset of the graph G must always be connected. The first case requires simultaneous activation of the nodes meaning that all nodes in the network must activate at the same time and remain active for the duration of the execution. The second case allows for staggered activation of nodes meaning that a node in the network may activate at any point but must remain active for the duration of the execution once it has activated. The third case allows

for staggered and repeated activations and deactivations of nodes meaning that a node in the network may activate or deactivate without limit at any point during execution.

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 unless u deactivates at some point after receiving the *send* 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

In the model section, we introduced three sets of restrictions on node behavior. Each set of restrictions requires its own algorithm. While the algorithms for the first and second set of restrictions share many similarities, they are very different from the algorithm for the third set of restrictions.

4.1 Reliable Broadcast with Simultaneous Activation

The first step of the algorithm is to run the leader election protocol, which builds a spanning tree in the network and elects the 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 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. m is guaranteed to reach the leader because the leader is an ancestor of every node in the tree. When the leader receives a message, it adds that message to its send queue.

If another message is not being broadcast through the tree, the leader will dequeue the next message m, if any exists, off its send queue. Next, the leader will notify its environment of message m using the *receive* command and then broadcast m. All of the children of the leader will receive m and perform the same two actions as the leader, 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 the leader has received confirmation messages from all of its children, the leader will dequeue the next message m', if any exists, off its send queue and repeat the above protocol.

At the same time, the leader 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 the leader 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 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 Deactivation

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

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

Definition 1. *Let u_{min} be the ID of the process with the minimum 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. *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 $b_{u_{min}}$ so $b_{u_{min}}$ will eventually terminate and the process with ID u_{min} will set leader = true. □

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

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 $j > u_{min}$, there is at least one process, the process with ID u_{min}, that will never reply done to b_j. Therefore, b_j will never terminate. □

Lemma 5.4. *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_{min} (Lemma 5.3). Only the process with ID u_{min} will set leader = true. □

5.2 Leader Election with Staggered Activation

Theorem 5.5. *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 5.6). No more than one node will have leader set to true at any point (Lemma 5.8). □

Definition 3. *Let u_{min} be the ID of the process with the minimum global round of activation and the minimum UID of the nodes that at that round in the network.*

Definition 4. *Let BFS instance b_i refer to an instance of the terminating breadth-first search protocol initiated by process with ID i.*

Lemma 5.6. *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 $b_{u_{min}}$ so $b_{u_{min}}$ will eventually terminate and the process with ID u_{min} will set leader = true. □

Lemma 5.7. *A BFS instance b_i will only terminate if i equals u_{min}.*

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 $j > u_{min}$, there is at least one process, the process with ID u_{min}, that will never reply done to b_j. The process with ID u_{min} will have been activate before or at the same round as every process in the network, including the process with ID j, so j will require the process with ID u_{min} to reply done, which it never will. Therefore, b_j will never terminate. □

Lemma 5.8. *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_{min} (Lemma 5.7). Only the process with ID u_{min} will set leader = true. □

5.3 Reliable Broadcast with Simultaneous Activation

Theorem 5.9. *The algorithm solves the reliable broadcast problem with simultaneous activation.*

Proof. This algorithm satisfies the liveness property (Lemma 5.11), the first safety property (Lemma 5.12), the second safety property (Lemma 5.13) and the third safety property (Lemma 5.14) of the reliable broadcast problem with simultaneous activation. □

Lemma 5.10. *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. Every child of a non-leaf node eventually sends a confirm message because the subtree of every node ends with all leaf nodes. As a result, every non-leaf node, including the leader, will eventually receive a confirm message from its children.

□

Lemma 5.11. *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 5.10). When the leader has received confirmation messages from all of its children, it will notify u using the same stable path. When u receives this message from the leader, it sends a "done" signal to E_u using the *acknowledge* command.

□

Lemma 5.12. *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 5.10). 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 5.13. *The algorithm satisfies the second safety property of the reliable broadcast problem with simultaneous activation.*

Proof. 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'.

If u passes m to its environment, m must have been broadcast by the leader. If the leader broadcasts m, every node in the network will receive m and notify

its environment and eventually the leader will receive a confirm message. If u receives m' after m, then the leader must have broadcast m' after m because the leader will not begin to broadcast another message, like m', until it has received a confirm message that all of the nodes in the network, including u, have seen m. As a result, no node in the network will pass m' to its environment before m.

□

Lemma 5.14. *The algorithm satisfies the third safety property of the reliable broadcast problem with simultaneous activation.*

Proof. 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.

□

5.4 Reliable Broadcast with Staggered Activation

Theorem 5.15. *The algorithm solves the reliable broadcast problem with staggered activation.*

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 staggered activation. □

Lemma 5.16. *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 after they are a confirmed member of the spanning tree. □

Lemma 5.17. *A given node u that receives a message m at round r will eventually send a confirmation message to its parent.*

Proof. If u is a leaf node at round $r + 2$, u will immediately send a confirmation message so any nodes joining the tree as children of u won't affect the propagation of the confirm message up the tree.

If u is a non-leaf node at round $r + 2$, u will broadcast m. All active nodes that consider u to be their parent before $r + 2$, will process m. If a node v considers u to be its parent at round $r + 2$, u will consider v to be its child no later than round $r + 3$. According to the algorithm, u will only wait to hear confirm messages from nodes that it considers to be its children at the end of round $r + 3$ so u will only wait to hear from nodes that processed m and are guaranteed to eventually send a confirm message to u.

As a result, u is guaranteed to send a confirm message to its parent.

□

Lemma 5.18. *The algorithm satisfies the liveness property of the reliable broadcast problem with staggered 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 (Lemma 5.16). 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 5.17). When the leader has received confirmation messages from all of its children, it will notify u using the same stable path. When u receives this message from the leader, it sends a "done" signal to E_u using the *acknowledge* command.

□

Lemma 5.19. *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 5.12).
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.

□

Lemma 5.20. *The algorithm satisfies the first safety property of the reliable broadcast problem with staggered activation.*

Proof. 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 (Lemma 5.12) given that the activation of any nodes won't affect the path between a node and the leader (Lemma 5.16) and any node that receives a message will eventually send a confirmation message to its parent (Lemma 5.17).

If all the nodes in $A(r, r')$ are not part of the spanning tree of by round r, 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 5.19) and any node that receives a message will eventually send a confirmation message to its parent (Lemma 5.16).

□

Lemma 5.21. *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.

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'.

The leader of the tree only broadcasts one message at a time as after sending a message, it waits to receive a confirmation message before sending the next message in its send queue. If u passes m to its environment before passing m' then the leader must have broadcast m before it broadcast m'. As a result, any node that received m and m' must have received m before m'.

□

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

Proof. 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.

□

5.5 Reliable Broadcast with Activations and Deactivations

Theorem 5.23. *The algorithm solves the reliable broadcast problem with activations and deactivations.*

Proof. This algorithm satisfies the liveness property(Lemma 5.24), the first safety property(Lemma 5.26), the second safety property(Lemma 5.27) and the third safety property(Lemma 5.28) of the reliable broadcast problem with activations and deactivations. □

Lemma 5.24. *The algorithm satisfies the liveness property of the reliable broadcast problem with activations and deactivations.*

Proof. 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 5.25. *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 any node in the network has not yet 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 5.26. *The algorithm satisfies the first safety property of the reliable broadcast problem with staggered activation.*

Proof. Every node in $A(r, r')$ sees m within n rounds (Lemma 5.25). 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 5.27. *The algorithm satisfies the third safety property of the reliable broadcast problem with deactivation.*

Proof. By definition of the algorithm, every node that receives a message will pass that message to its environment at a pre-determined round. Assume two messages, m and m', are received by multiple nodes. m will be executed by all nodes at round r and m' will be executed by all nodes at round r'.

If r is greater than r', then all nodes that receive m and m' and are active for both r and r' will pass m' to their environment before they pass m. If r' is greater than r, then all nodes will that receive m and m' and are active for both r and r' will pass m to their environment before they pass m'.

If r is equal to r', then nodes will use the UID attached to the message to determine the order of execution. If m_{ID} is greater than m'_{ID}, then all nodes that receive m and m' and are active for both r and r' executes m before m'. If $m'_{sourceID}$ is greater than $m_{sourceID}$, then all nodes that receive m and m' and are active for both r and r' executes m' before m. $m'_{sourceID}$ cannot be equal to $m_{sourceID}$ because a single node cannot receive a message from its environment while it has an unacknowledged message out there.

□

Lemma 5.28. *The algorithm satisfies the third safety property of the reliable broadcast problem with staggered activation.*

Proof. Per the algorithm, nodes keep a list of messages to execute and only add a message m if m is not already in the list. A node will only execute message m at its specified round if m is in its list at that round.

□