Implementing the Reliable Broadcast Service in Dynamic Distributed Networks

May 5, 2014

Welles Robinson

Senior Thesis

Department of Computer Science

Georgetown University

Abstract

The importance of distributed computing has grown in recent years with the creation and accumulation of massive amounts of data, the prevalence of mobile devices and the development of peer-to-peer networks. The vast majority of the existing distributed theory literature has focused on static distributed networks, which assume a relatively constant network topology and participant set. While reasonable for relatively stable settings such as data centers, the assumptions of this approach will not hold for wireless networks, which feature dynamic sets of participants and variable topologies because the nodes are extremely mobile and the links unstable. Unlocking the potential of distributed computing in wireless networks requires the implementation of primitives used by static distributed algorithms in this more dynamic distributed network setting. To this end, this thesis describes and proves correct three novel algorithms that implement a reliable broadcast service in dynamic distributed networks with varying assumptions for node behavior. The first two proposed algorithms assume nodes will not deactivate during execution and consequently demonstrate strong performance and efficiency. The third algorithm allows nodes to activate and deactivate during execution although this comes at a slight cost to performance. Finally, to demonstrate the practicality of these algorithms in realistic networks, this thesis also presents simulation results on both synthetic and real world data sets of varying size and topology.

1 Introduction

In the last decade, the importance of distributed computing has grown rapidly as a result of the enormous increase in the amount of data being processed and shared. However, distributed systems only works as well as their underlying algorithms. For example, systems like Hadoop, Akamai and BitTorrent, which have seriously impacted big data storage, the Internet and the music industry respectively, all rely on provably correct distributed algorithms.

Reflecting the aforementioned examples, the majority of the work in distributed algorithms has focused on what we term *static distributed networks*, which assume a relatively stable network topology and set of participants. These algorithms are designed to handle node failures but not changes to the network topology or participant set. This choice is logical when designing peer-to-peer networks overlaid on the Internet, which guarantees reliable connections between nodes. Similarly, this design decision makes sense in the context of big data centers where computers are hard wired together in a fixed topology.

However, if distributed algorithms are to play a large role in the exciting future of wireless technology, it is imperative to develop algorithms for what we term *dynamic distributed networks*, which allow for variable network topologies and participant sets. The core appeal of wireless devices is their ability to move and maintain connectivity. However, the movement of nodes changes the underlying network topology and participant set, both of which are not handled by *static distributed networks*. Many potential applications of distributed systems exist in this rapidly growing field. For example, consider the increase in household devices with wireless capabilities, which is commonly referred to as the "Internet of things". Many of these devices move location frequently, but still need to engage in reliable, wireless coordination with the other scattered devices.

The rise of powerful smartphones portend another enormous need for algorithms for *dynamic distributed networks*. Currently, nearly all wireless communication passes through cellular towers. Distributed computing in a peer-to-peer networking between cell phones would increase the effective range of these towers and as well as the performance and efficiency of cellular networks. These are important considerations given the increasing strain on cellular networks. Even data centers, long the domain of static distributed algorithms, have begun experimenting with wireless connectivity to reduce power use and cost. The potential of distributed algorithms in the growing field of wireless technology motivates the need for significant work on algorithms for *dynamic distributed networks*.

This thesis implements a reliable broadcast service in *dynamic distributed networks*. Reliable broadcast requires nodes to disseminate messages, which are processed in the same order by all nodes in the network. Reliable broadcast is a key primitive for implementing replicated state machines, a common strategies for implementing fault-tolerant services in distributed systems. Reliable broadcast can also be used to solve the consensus problem, which is at the core of many distributed systems.

This thesis describes and proves correct three new implementations of a reliable broadcast service in *dynamic distributed networks*. As detailed in the related work section, some prior work investigates reliable broadcast style problems in *dynamic distributed networks*. However, these papers by Brown et al [**?**], Chockler et al [**?**] and Gao et al [**?**] make assumptions that are not practical in the real world. For example, these papers require that nodes have knowledge of their absolute location and remain within a physical area that is determined prior to execution. To the best of our knowledge, this thesis is the first examination of a reliable broadcast style problem in a *dynamic distributed networks*.

The first implementation we provide of a reliable broadcast service assumes that all nodes in the network activate before the same global round and remain active for the duration of the execution (Simultaneous Activation). The second implementation allows nodes to activate before different rounds of execution but assumes that after a node activates, it remains active for the remainder of the execution (Staggered Activation). The third implementation assumes nodes can activate or deactivate before each and every round (Variable Activation and Deactivation). While the first two sets of assumptions do not allow for significant changes to the network topology and participant set during execution, they demonstrate the best-case efficiency and performance in a reasonably well-behaved *dynamic distributed networks*. While the third implementation correctly implements reliable broadcast for all three sets of assumptions about node behavior related to activations and deactivations, it does so with a worse average case performance than both the first and second implementation.

The major contributions of this thesis are as follows. First, it provides a correct and useful definition for reliable broadcast in *dynamic distributed networks*. Second, it describes and proves correct three novel algorithms for reliable broadcast in *dynamic distributed networks*given the different sets of assumptions regarding node activation and deactivation during execution. Third, these algorithms use general strategies that will be useful for the development of other algorithms for *dynamic distributed networks*. Finally, this thesis provides simulation results for the algorithms in both synthetic and real world networks to explore average case performance beyond the formally proven worst-case bounds.

2 Related Work

Extensive theoretical work has been done for static distributed networks. We refer you to Schneider for a more detailed discussion of that literature. The replicated state machine approach discussed by Schneider is a general method for implementing fault-tolerant static distributed systems [**?**]. Their approach assumes a fully connected network topology and a stable set of participants. That approach is not sufficient for this work because we assume only a connected network topology and allow a variable participant set.

Kuhn et al. use the model of dynamic distributed networks that most closely matches our own model [**?**]. Kuhn et al. describe and prove correct algorithms for token dissemination and counting in networks with T-interval connectiv-

ity. Token dissemination and counting are lower level problems than reliable broadcast. However, they are primitives for reliable broadcast and in fact, this thesis leverages Kuhn et al.'s proof of all-to-all token dissemination in networks with at least 1-interval connectivity. In contrast to our model, which permits a dynamic participant set, their model assumes a static set of participants.

Other papers that examine dynamic distributed networks create a reliable virtual layer on top of the variable topology of the underlying dynamic distributed network [**?**, **?**, **?**].

Brown et al. describe and implement a virtual node layer on top of a dynamic distributed network. This high level abstract layer, the Virtual Node Layer, allows algorithms to be developed for a network with stable topology and participant set without making any assumptions about the underlying network topology or participant set. However, this approach requires that nodes have knowledge of their absolute location. Moreover, the position of the virtual nodes and therefore the range of the underlying network must be determined prior to execution.

Chockler et al. describe a protocol for emulating virtual infrastructure in collision-prone wireless networks. They describe a convergent history agreement protocol to address the problem of lost messages in collision-prone ad-hoc wireless networks. While unreliable communication due to collisions is a fundamental problem of wireless communication, it is a much lower level problem than implementing a reliable broadcast service. Our model assumes reliable communication between neighboring nodes. Chockler et al. use a similar approach as Brown et al's Virtual Node Layer and share the same requirements that nodes must know their absolute location and that the location of the virtual infrastructure must be determined prior to execution.

Gao et al. describe a new approach to data processing for cellular devices that leverages principles of distributed computing. Like Brown et al. and Chockler et al., Gao et al. use a static and reliable abstract layer, the Virtual Core Layer, which is emulated by unreliable and mobile cellular devices. On top of the Virtual Core Layer, Gao et al. implement a shared memory layer that allows cellular devices to access data stored across a wide geographic region. Their paper assumes that the nodes, the cellular devices, have internet access. It also shares the virtual infrastructure requirement that nodes must know their absolute location and that the location of the virtual infrastructure must be determined prior to execution.

3 Model

We consider a broadcast variant of the standard synchronous message passing model of distributed computation [**?**, **?**]. The synchronous message passing model is defined with respect to a directed graph $G = (V, E)$. We define $n = |V|$ as 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 $\mathcal{A}$, this means each node in the network is running a copy of $\mathcal{A}$. (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. Receiving and processing all incoming messages from their neighbors.

The combination of these two steps is called a *round*.

The synchronous broadcast model that we study in this paper is different from the synchronous message passing model in three significant ways. First, the synchronous broadcast model is defined with respect to a connected graph $G = (V, E)$ with bi-directional edges. Second, nodes do not pass individual messages directly to their neighbors. Instead, nodes broadcast their messages at the beginning of every round and every message is received by every neighbor. Finally, in the synchronous message passing model, nodes know their neighbors in advance, while nodes do not have any prior knowledge of their neighbors in the synchronous broadcast model.

We assume that nodes have comparable unique identifiers and that nodes are in one of two high-level states, active or inactive. When a node is active, it performs the two steps, sending and receiving messages, that constitute a round. When a node is inactive, it performs neither of the two steps that constitute a round. We say a node is activated when its state changes from inactive to active. We assume nodes only change states between rounds. 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 inactive. When a node is deactivated, it maintains all local variables such that if it reactivates, it has knowledge of its previous state and knowledge of the current global round. The ability to retain previous state knowledge after deactivating allows the algorithm to treat a node's reactivation either as a new activation by restoring the default state or a reactivation by leaving the previous state unchanged.

This thesis uses three sets of assumptions about the behavior of nodes with respect to activations and deactivations. The first set assumes simultaneous activation of nodes, meaning that all nodes in the network must activate before the same round and remain active for the duration of the execution (Simultaneous Activation). The second set allows for staggered activation of nodes, meaning that a node in the network may activate before any round of execution but must remain active for the duration of the execution once it has activated (Staggered Activation). The third set allows for nodes to activate and deactivate without limit before the beginning of any round of execution (Variable Activation and Deactivation). All three sets of assumptions assume that the active subset of the graph G must be connected as a single component before the beginning of every round.

4 Reliable Broadcast Service

A reliable broadcast service assumes there is an environment at each node u that communicates with u using one of three commands, *send*, *receive* and *acknowledge*. We refer to the environment at node u as E_u. A reliable broadcast service allows any given node u in the network to disseminate a given message m received from its environment to all the nodes in the network and ensures that all nodes that pass m to their environment using the *receive* command do so in the same order relative to other messages disseminated through the network.

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. When a node v learns about a message m, it uses the *receive* command to notify E_v about m. When 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.

An algorithm A is said to correctly implement a reliable broadcast service if it implements the *send*, *receive* and *acknowledge* commands and satisfies the following properties (in the following, assume without loss of generality that 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 passes 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

5 Algorithms

In the model section, we introduced three sets of assumptions for node behavior with respect to activation and deactivation. In this section, we present a new algorithm for each set of assumptions. Each set of assumptions is strictly more stringent than the previous set so, for example, the algorithm for reliable broadcast under the third set of assumptions would also work under both the first and second set of assumptions. We present multiple algorithms to maximize performance under each set of assumptions. The remainder of this sections details each of the algorithms proposed in this thesis. Each algorithm is proven correct in the following section.

5.1 Simultaneous Activation Reliable Broadcast Algorithm

At the beginning of round 1, every node in the network activates and begins running the simultaneous activation reliable broadcast. The first step of the simultaneous activation reliable broadcast algorithm, which we call the leader election subroutine, is for each node to run the terminating synchronous breadth-first search algorithm (SynchBFS) as described and analyzed by Lynch [**?**]. The nodes will use SynchBFS to elect a leader as follows. When a node u activates, it starts running an instance of SynchBFS with itself as the source. Each node attempts to spread its instance of SynchBFS through the network by disseminating a message m. In this case, the message m contains the ID of the source of the SynchBFS instance being disseminated. Message dissemination begins with a node u broadcasting m to its neighbors. In the next round, u's neighbors broadcast m to their neighbors and in the way, m is sent to every node in the network. The node that sends message m to node v is called v's parent for message m. If node t is the parent of v, we say that v is a child of t. In message dissemination, it is necessary for the source node to know when all the nodes in the network have seen the message. To this end, a node that receives a message will reply "finished" to its parent when all of its children reply "finished" to it. In this way, the "finished" messages propagate back up to the source. A message is said to have terminated once the source of the message has received"finished" messages from all of its children.

This particular message dissemination is modified so that a node will only process and broadcast the message if it has not heard of an instance of SynchBFS with a smaller source. A node is said to participate in an instance of SynchBFS b if the source of b is the smallest source seen by the node. A node stops participating in an instance of SynchBFS if it hears about an instance of SynchBFS with a smaller ID. Therefore only the SynchBFS instance started by the node with the minimum ID in the network will terminate. Once this instance of SynchBFS terminates, its source will elect itself leader and disseminate a confirm message throughout the network using the general approach described above. When a node receives the confirm message, it becomes a confirmed member of the spanning tree rooted at the source of the SynchBFS instance that terminated.

While a node can receive a *send* command from its environment at any point

during execution, it won't do anything with the message until it is a confirmed member of the tree. Assume a node u receives a message m from E_u using the *send* command and is confirmed member of the spanning tree. First, u sends the message m to the leader of the network by broadcasting m with the instructions that only its parent should forward m. Each node that forwards m keeps track of which child node sent m to it. Message m continues to be passed up the tree from child to parent until it reaches the leader of the tree. Message 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 the end of its send queue.

If a message is not being disseminated 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 disseminate m in the manner described in the leader election subroutine. When a node receives m from its parent, it notifies its environment using the *receive* command. When m has terminated, the leader will dequeue the next message m', if any exists, off its send queue and disseminate m' through the network in the same way as m.

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. The leader broadcasts this message with the instruction that it only should be forwarded by the node that sent m to the leader in the first place. Eventually, this message will reach u, which notifies its environment using its *acknowledge* command.

5.2 Staggered Activation Reliable Broadcast Algorithm

Our staggered activation reliable broadcast algorithm is the same as our simultaneous activation reliable broadcast algorithm with the following modifications. At the beginning of round 1, every node in the network that is active for round 1 activates and begins running the staggered activation reliable broadcast. The other nodes begin executing the staggered activation reliable broadcast algorithm when they activate. Under the assumption of staggered activation, once a node activates, it remains active for the duration of the execution.

In the leader election subroutine of simultaneous activation reliable broadcast algorithm, the source of a SynchBFS instance started by node u is the ID of u. In the leader election subroutine of staggered activation reliable broadcast algorithm, the source of a SynchBFS instance started by node u is the combination of the global round when u activated and the ID of u. For example, in the staggered activation reliable broadcast algorithm, a node will stop running a SynchBFS instance if it hears about a SynchBFS instance with a lowest global round of activation or a SynchBFS instance with the same global round of activation and a smaller ID. By the definition of the model, nodes have unique IDs so multiples BFS instances cannot have the same ID. This change ensures that the leader will always be part of the first group of nodes to activate.

The another significant change is that nodes that are confirmed members

of the tree broadcast a message every round telling their neighbors to become a confirmed member of the tree. This is necessary because even after a leader has been elected and sent a confirm message through the tree, it is still possible for nodes to activate. When a node activates, it follows the leader election subroutine. If a node is told to join the confirmed tree by one of its neighbors at any point during the leader election subroutine, it stops running its current instance of SynchBFS, chooses the sender as its parent and becomes a confirmed member of the spanning tree.

The final significant change is for message dissemination by the leader of the tree. We incorporate an additional one round delay between node v receiving a message m from its parent and node v either responding "finished" to its parent if it has no children or broadcasting m to its children if it has any. For example, in the simultaneous activation reliable broadcast algorithm, if leaf node v receives a message from its parent during round r, it broadcasts a "finished" message in response at the beginning of round $r + 1$. However, in an identical situation in the staggered activation reliable broadcast algorithm, node v broadcasts a "finished" message in response at the beginning of round $r + 2$.

5.3 Variable Activation and Deactivation Reliable Broadcast Algorithm

The general approach of the variable activation and deactivation reliable broadcast algorithm is different from that of the algorithms for reliable broadcast under the assumption of only activations. This algorithm does not run SynchBFS to elect a leader or rely on a stable tree. Instead, it floods messages through the network.

At the beginning of round 1, every node in the network that is active for round 1 activates and begins running the variable activation and deactivation reliable broadcast. All the other nodes begin executing the variable activation and deactivation reliable broadcast algorithm when they activate. A node that reactivates resumes running the variable activation and deactivation reliable broadcast according to their previous state.

Every node in the network maintains a set of messages. A node adds a message to its set in one of two ways. First, a node adds a message m to its set when it receives m from its environment using the *send* command. Second, it adds a message m to its set when it receives a broadcast from one of its neighbors containing m, a message that does not exist in its set. Every round, nodes broadcast their set of messages to their neighbors.

A message in the variable activation and deactivation reliable broadcast algorithm has an execution round, which specifies the exact round that every node should execute the *receive* command on a message. Assume a node u receives a message m from its environment at round r using the *send* command. The execution round for m is r', which is equal to round $r + n$ where n is an upper bound on the number of nodes in the network. If a node has multiple *receive* commands to execute in the same round, it executes them in ascending order of the ID of the message, which is the ID of the node that received the

message from its environment. A node removes m from its set of messages after it executes a *receive* command on m. Node u, the node that received m from its environment using the *send* command, executes the *acknowledge* command on m at round $r' + 1$.

6 Analysis

In this section, we prove the correctness of the three algorithms described in the previous section. The analysis sections for the first two algorithms begin with a proof of the leader election subroutine in their respective settings as correct leader election is integral to the reliable broadcast protocols.

6.1 Simultaneous Activation Reliable Broadcast Algorithm

In this section, we prove the correctness of the algorithm assuming simultaneous activation of nodes. First, we prove the correctness of the leader election subroutine and then leverage this knowledge to prove the correctness of the overall simultaneous activation reliable broadcast algorithm.

6.1.1 Leader Election Subroutine

We prove the correctness of leader election by separately proving the liveness and safety property of the subroutine.

Theorem 6.1. *Under the assumption of simultaneous activation, the simultaneous activation reliable broadcast algorithm will eventually elect a single node to be leader as indicated by the leader node setting its leader variable to true.*

Proof. To prove that a single node elects itself leader, it is sufficient to prove both the liveness and safety property of the given leader election subroutine. Liveness guarantees that at least one node will eventually elect itself leader (Lemma 6.2) while safety guarantees that no more than one node will have its leader variable set to true at any point (Lemma 6.4). □

We now define notation that will simplify the statement and proof of the lemmata that follow. Unless redefined, this notation will also be used in the lemmata for the leader election theorem for the staggered activation reliable broadcast algorithm.

Definition 1. *Let u_{min} be the ID of the node with the minimum ID in the network.*

Definition 2. *Let SynchBFS instance b_i refer to an instance of terminating synchronous breadth-first search (SynchBFS) initiated by the node with ID i.*

Lemma 6.2. *One node will eventually set its leader variable to true.*

Proof. A SynchBFS instance b_i will eventually terminate if every node in the network participates in b_i. Every node in the network will eventually participate in $b_{u_{min}}$ because when a node hears about $b_{u_{min}}$, it will stop participating in its current SynchBFS instance and start participating in $b_{u_{min}}$ because it is the SynchBFS instance with the minimum ID in the network. As a result, $b_{u_{min}}$ will eventually terminate and the node with ID u_{min} will set leader = true. □

Lemma 6.3. *A SynchBFS instance b_i will only terminate if i equals u_{min}.*

Proof. For any instance b_j to terminate, every node except the node with id j needs to send a "finished" message at some point. Given BFS instance b_j where $j > u_{min}$, there is at least one node, the node with ID u_{min}, that will never send a "finished" message in b_j. Therefore, b_j will never terminate. □

Lemma 6.4. *For every round r, at most one node has its leader variable equal to true at the beginning of r.*

Proof. A node with ID i will only set its leader equal to true if its BFS instance, b_i, terminates. A BFS tree b_i will terminate only if i equals u_{min} (Lemma 6.3). Only the node with ID u_{min} will set its leader variable equal to true. □

6.1.2 Overall Correctness

Drawing on the proof of correctness of the leader election subroutine, we prove the simultaneous activation reliable broadcast algorithm satisfies the four properties of reliable broadcast and therefore correctly implements the reliable broadcast service.

Theorem 6.5. *Under the assumption of simultaneous activation, the simultaneous activation reliable broadcast algorithm correctly implements the reliable broadcast service.*

Proof. Under the assumption of simultaneous activation, this simultaneous activation reliable broadcast algorithm satisfies the liveness property (Lemma 6.9), the first safety property (Lemma 6.10), the second safety property (Lemma 6.11) and the third safety property (Lemma 6.12) of the reliable broadcast problem. □

Lemma 6.6. *A given message m that is disseminated by the leader will reach every node in the network.*

Proof. For a message m to be disseminated by the leader, it means the leader must broadcast m, which will be received and processed by all of the leader's children. These children will broadcast m in the next round and so m will be received and processed by all of the their children. This pattern continues until eventually every node in the network receives m because every node in the network is a descendant of the leader.

□

Lemma 6.7. *For a given message m that is disseminated by the leader, the leader will eventually receive a "finished" message from all of its children signifying that all nodes in the network have seen message m.*

Proof. According to the subroutine, a leaf node will send a "finished" message to its parent upon receiving m. Every child of a non-leaf node eventually sends a finished message because the subtree of every node ends with all leaf nodes. As a result, the leader will eventually receive a "finished" message from all of its children, signifying that all nodes in the network have seen message m.

□

Lemma 6.8. *For a given node u that is a confirmed member of the tree, there exists a stable path between u and the leader.*

Proof. A node can send a message to its parent by broadcasting a message and specifying that its parent is the intended recipient. If node u is a confirmed member of the tree, node u can send a message m to the leader of the tree by sending a m to its parent with the instructions that the intended recipient should forward m to its parent until m reaches the leader. Message m is guaranteed to reach the leader of the tree because the leader is an ancestor of node u. Once a node is a confirmed message of the tree, its parent will not change so this path is guaranteed to be stable.

□

Lemma 6.9. *The simultaneous activation reliable broadcast algorithm satisfies the liveness property of a reliable broadcast service.*

Proof. Assume node u running algorithm $\mathcal{A}$ is passed a message m by E_u through its *send* command. If u is not a confirmed member of the spanning tree, it will store message m until it becomes a confirmed member of the spanning tree. Node u will eventually become a confirmed member of the tree because the leader election subroutine will eventually result in one node electing itself leader (Lemma 6.1) and any message disseminated by the leader, including the "confirm" message, will be seen by every node in the network (Lemma 6.6).

If node u is a confirmed member of the spanning tree, it sends m to the leader of the tree. If node u is a confirmed member of the tree, there exists a stable path between u and the leader so m is guaranteed to reach the leader (Lemma 6.8). When m reaches the leader, it is placed in its send queue. The message in the front of the leader's send queue will be dequeued if the network is not already in the node of disseminating another message. A message that is disseminated through the network is guaranteed to terminate (Lemma 6.7). Given that all of the messages in front of m in the queue will eventually be disseminated and terminate, m is guaranteed to reach the front of the queue, be disseminated and terminate. When message m terminates, the leader will notify u by sending m along the stable path between the leader and u (Lemma 6.8). When u receives this message from the leader, it sends a "done" signal to E_u using the *acknowledge* command thus satisfying the liveness property of reliable broadcast.

□

Lemma 6.10. *The simultaneous activation reliable broadcast algorithm satisfies the first safety property of a reliable broadcast service.*

Proof. Under the assumption of simultaneous activation, $A(r, r')$ is the set of all nodes in the network so the first safety property of reliable broadcast 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'. Node u first sends m to the leader of the tree along a stable path (Lemma 6.8). The leader will disseminate m to the network at some round after r and all the nodes in the network will receive m and pass m to their environments through their *receive* command (Lemma 6.6). The leader eventually receives a "finished" message from all of its children and then sends m back to u, which then sends a "done" signal at round r' to E_u using the *acknowledge* command (Lemma 6.7).

□

Lemma 6.11. *The simultaneous activation reliable broadcast algorithm satisfies the second safety property of a reliable broadcast service.*

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, the leader must have disseminated m to the network. If the leader disseminates m, every node in the network will receive m and notify its environment and eventually the leader will receive a confirm message (Lemma 6.6) (Lemma 6.7). 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. (Lemmata 6.6 and 6.7)

□

Lemma 6.12. *The simultaneous activation reliable broadcast algorithm satisfies the third safety property of a reliable broadcast service.*

Proof. The third safety property of reliable broadcast service with simultaneous activation requires that if a node u passes a message m to its environment, it has not passed m to its environment before and some node in the network previously received m from its environment. It follows from the definition of the algorithm that any node u that is not the leader of the tree will pass a message m to its environment only if and when it receives m from its parent in the tree. A node will also only pass a message to its environment if it is a confirmed member of the tree. If a node is a confirmed member of the tree, it has only one parent, which does not change during the execution. As a result, u will only node and pass m to its environment once. If node u is the leader of the tree, it will only pass m to its environment when it dequeues m from its queue, which will only

happen once for each message. Therefore, a node u will only pass a message m to its environment at most one time.

The only way in which u will receive a message m to pass to its environment is if m is disseminated to the network by the leader. The leader will only disseminate a message m if m is sent to it by its own environment or by a node that received m from its environment. Therefore, a node u will only pass a message m to its environment if m has been received by a node in the network from its environment.

□

6.2 Staggered Activation Reliable Broadcast Algorithm

In this section, we prove the correctness of the staggered activation reliable broadcast algorithm under the assumption of staggered activation. First, we prove the correctness of the leader election subroutine and then leverage this knowledge to prove the correctness of the overall staggered activation reliable broadcast algorithm.

6.2.1 Leader Election Subroutine

Under the assumption of staggered activation, we prove the correctness of leader election by separately proving the liveness and safety property of the subroutine.

Theorem 6.13. *Under the assumption of staggered activation, the leader election subroutine of the staggered activation reliable broadcast algorithm will eventually elect a single node to be leader as indicated by the leader node setting its leader variable to true.*

Proof. To prove that a single node elects itself leader, it is sufficient to prove both the liveness and safety property of the given leader election subroutine. Liveness guarantees that at least one node will eventually elect itself leader (Lemma 6.14) while safety guarantees that no more than one node will have its leader variable set to true at any point (Lemma 6.16).

□

We now define notation that will simplify the statement and proof of the lemmata that follow. We redefine u_{min} but do not redefine b_i and will use the definition of b_i from the simultaneous activation reliable broadcast algorithm section.

Definition 3. *Let u_{min} be the ID of the node which has the minimum global round of activation in the network. If multiple nodes have the minimum global round of activation, then u_{min} will be the node with the minimum ID out of the nodes with the minimum global round of activation.*

Lemma 6.14. *At least one node will eventually set its leader variable equal to true.*

Proof. A SynchBFS instance b_i will eventually terminate if every node in the network participates in b_i. Every node in the network will eventually participate in $b_{u_{min}}$ because when a node hears about $b_{u_{min}}$, it will stop participating in its current SynchBFS instance and start participating in $b_{u_{min}}$ because it is the SynchBFS instance whose source has the minimum combination of global round of activation and ID in the network. As a result, $b_{u_{min}}$ will eventually terminate and the node with ID u_{min} will set leader equal to true. □

Lemma 6.15. *A SynchBFS instance b_i will only terminate if i equals u_{min}.*

Proof. For any instance b_j to terminate, every active node except the node with id j needs to send a "finished" message at some point. By definition of u_{min}, the node with ID u_{min} activates before or at the same round as every node in the network, including the node with ID j. As a result, the termination of b_j where $j \neq u_{min}$ requires the node with ID u_{min} to send a "finished" message. Given BFS instance b_j where $j > u_{min}$, there is at least one node, the node with ID u_{min}, that will never send a "finished" message in b_j. Therefore, b_j will never terminate. □

Lemma 6.16. *For every round r, at most one node has leader equal to 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 6.15). Only the node with ID u_{min} will set leader equal to true. □

6.2.2 Overall Correctness

Drawing on the proof of correctness of the leader election subroutine, we prove the staggered activation reliable broadcast algorithm satisfies the four properties of reliable broadcast and therefore correctly implements the reliable broadcast service.

Theorem 6.17. *The staggered activation reliable broadcast algorithm correctly implements a reliable broadcast service.*

Proof. This staggered activation reliable broadcast algorithm satisfies the liveness property (Lemma 6.20), the first safety property (Lemma 6.22), the second safety property (Lemma 6.23) and the third safety property (Lemma 6.24) of the reliable broadcast problem under the assumption of staggered activation. □

Lemma 6.18. *The path from the leader to a node u in the network will not change after u becomes a confirmed member of the tree rooted at the leader.*

Proof. Being a confirmed member of the tree means that the node is part of the spanning tree rooted at the leader. There exists a path from every node u that is a confirmed member of the tree to the leader because the leader is the root of the tree and therefore an ancestor of every node in the tree. This path travels up the tree from node to parent until it reaches the leader, which is the root of

the tree. This path will not change because nodes do not change their parent after they become a confirmed member of the tree and nodes do not deactivate under the assumptions of staggered activation. □

Lemma 6.19. *For a given message m that is disseminated by the leader, the leader will eventually receive a "finished" message from all of its children signifying that all nodes in the network have seen message m.*

Proof. Assume node u is a child node of the leader of the tree and receives a message m from the leader at round r. By definition of the algorithm, a node that receives a message m at round r will process the message at round $r+1$. If node u is a leaf node at the end of round $r+1$, u will send a "finished" message to its parent, the leader, at the beginning of round $r+2$.

If u is not a leaf node at round $r+1$, u will broadcast m at $r+2$, which will be processed by all of its children. All active nodes that consider u to be their parent at the beginning of $r+2$, will process m. At the end of round $r+3$, the children of u process the message in the same way as node u. If they are leaf nodes, they send a "finished" message to their parent, node u, at the beginning of round $r+4$. Otherwise, they broadcast m to their children. In this way, the message m travels through the tree until it reaches a leaf node. These leaf nodes will respond "finished" to their parents. When a non-leaf node receives "finished" messages from all of its children, it sends a "finished" message to its parent. In this way, "finished" messages propagate up the tree until u receives "finished" messages from all of its children. At this point, u will send a "finished" message to its parent, the leader. In this way, the leader will eventually receive a "finished" message from all of its children.

□

Lemma 6.20. *The staggered activation reliable broadcast algorithm satisfies the liveness property of a reliable broadcast service.*

Proof. Assume node u running algorithm $\mathcal{A}$ is passed a message m by E_u through its *send* command. If u is not a confirmed member of the spanning tree rooted at the leader, it will become a confirmed member soon after the leader election subroutine terminates (Lemma 6.13). When u is a confirmed member of the spanning tree rooted at the leader, it sends m to the leader of the tree. There is a stable path between the leader and any confirmed node so m is guaranteed to reach the leader (Lemma 6.18). When m reaches the leader, it is placed in its send queue. The message in the front of the leader's send queue will be dequeued if the network is not already in the node of disseminating another message. A message that is disseminated through the network is guaranteed to terminate as all of the leader's children will eventually send the leader a finished message (Lemma 6.19). Given that all of the messages in front of m in the queue will eventually be disseminated and terminate, m is guaranteed to reach the front of the queue, be disseminated and terminate. When message m terminates, the leader will notify u by sending m along the stable path between the leader and u (Lemma 6.18). When u receives this message

from the leader, it sends a "done" signal to E_u using the *acknowledge* command thus satisfying the liveness property of reliable broadcast.

□

Lemma 6.21. *Assume node u activates before round r and is q hops from the closest node that is a confirmed member of the spanning tree rooted at the leader. Node u will receive any message m that is sent to a given node v from an environment during or after round r.*

Proof. This proof implicitly assumes that a leader has been elected by round r. If a leader has not been elected, there can be no confirmed members of the spanning tree rooted at the leader. This assumption does not invalidate the proof. No node will start the reliable broadcast for a message received from the environment until they are a confirmed member of the spanning tree rooted at the leader, which requires the election of a leader. For the purpose of the proof, we can allow r to be the round at which v begins the reliable broadcast service for m because the round at which v begins the reliable broadcast service for m will be greater than or equal to the round at which v receives m from its environment. In short, this assumption reduces the amount of time that u has to become a confirmed member of the spanning tree, making this proof stronger not weaker.

We prove the lemma statement by induction on q, the number of hops from u to the closest confirmed member of the spanning tree rooted at the leader.
Inductive Hypothesis: The theorem holds for all values of q up to k
Base Case ($k = 0$): A node u that is zero hops away from the tree is part of the tree and will receive the message m (Lemma 6.10).
Inductive Step: Assuming the theorem holds for all values of q up to k, we will show it holds for $q = k + 1$

Assuming node u is $k + 1$ hops from the closest confirmed member of the spanning tree rooted at the leader, there must exist a node v, which is a neighbor of u and is k hops from the closest confirmed member. By the inductive hypothesis, v receives m at a later round r'. By the definition of the algorithm, nodes that are not confirmed members of the tree will ignore messages being disseminated to the network by the leader. In order to receive message m, node v must be a confirmed member of the tree by the beginning of round r'. As a confirmed members of the tree, v will broadcast a message at the beginning of every round, starting at round r, to its neighbors telling them to join the spanning tree rooted at the leader. Node u will receive this message from node v in round r' and become a confirmed member of the tree rooted at the leader by the end of round r'. By the definition of the algorithm, as a non-leaf node, v will broadcast message m with a one round delay. When v broadcasts message m at the beginning of round $r' + 2$, node u will be a confirmed member of the tree and process m.

□

Lemma 6.22. *The staggered activation reliable broadcast algorithm satisfies the first safety property of a reliable broadcast service.*

Proof. Assume node u is passed a message m by E_u using the *send* command 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 6.10) given that the later activation of nodes will not affect the path between a confirmed node and the leader (Lemma 6.18) and that any node that receives a message will eventually send a confirmation message to its parent (Lemma 6.19).

If all the nodes in $A(r, r')$ are not part of the spanning tree of by round r, the nodes that are not part of the tree before round r will become part of the tree in time to receive the message (Lemma 6.21) and any node that receives a message will eventually send a confirmation message to its parent (Lemma 6.18).

□

Lemma 6.23. *The staggered activation reliable broadcast algorithm satisfies the second safety property of a reliable broadcast service.*

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 disseminated by the leader. If the leader disseminates m, every node in $A(r, r')$ will receive m and notify its environment and eventually the leader will receive a confirm message (Lemmata 6.19 and 6.22). 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 6.24. *The staggered activation reliable broadcast algorithm satisfies the third safety property of a reliable broadcast service.*

Proof. The third safety property of reliable broadcast service with staggered activation requires that if a node u passes a message m to its environment, it has not passed m to its environment before and some node in the network previously received m from its environment. It follows from the definition of the algorithm that any node u that is not the leader of the tree will pass a message m to its environment only if and when it receives m from its parent in the tree. A node will also only pass a message to its environment if it is a confirmed member of the tree. If a node is a confirmed member of the tree, it has only one parent, which does not change during the execution. As a result, u will only node and pass m to its environment once. If node u is the leader of the tree, it will only pass m to its environment when it dequeues m from its queue, which will only happen once for each message. Therefore, a node u will only pass a message m to its environment at most one time.

The only way in which u will receive a message m to pass to its environment is if m is disseminated to the network by the leader. The leader will only disseminate a message m if m is sent to it by its own environment or by a node that received m from its environment. Therefore, a node u will only pass a message m to its environment if m has been received by a node in the network from its environment.

□

6.3 Variable Activation and Deactivation Reliable Broadcast

In this section, we prove the correctness of the algorithm assuming variable activation and deactivation of nodes. This section differs considerably from the previous sections in that it does not have a proof of correctness for leader election as this algorithm does not have a leader election subroutine. This section proves correct the variable activation and deactivation reliable broadcast algorithm by separately proving the four properties of reliable broadcast.

Theorem 6.25. *Assuming variable activation and deactivation, the variable activation and deactivation reliable broadcast algorithm correctly implements a reliable broadcast service.*

Proof. Assuming variable activation and deactivation, the variable activation and deactivation reliable broadcast algorithm satisfies the liveness property (Lemma 6.26), the first safety property (Lemma 6.28), the second safety property (Lemma 6.29) and the third safety property (Lemma 6.30) of a reliable broadcast service. □

Lemma 6.26. *The variable activation and deactivation reliable broadcast algorithm satisfies the liveness property of a reliable broadcast service.*

Proof. Assume node u running algorithm $\mathcal{A}$ is passed a message m by E_u through its *send* command. By the definition of 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 our definition of liveness. □

Lemma 6.27. *Assume node u running algorithm $\mathcal{A}$ is passed a message m by E_u through its send command at round r. For a later round $r' = r + n$, every node in $A(r, r')$ will receive message m by the end of round $r' - 1$.*

Proof. By the model definition, the network graph of active nodes is connected every round. A dynamic graph that is guaranteed to be connected in every round satisfies the 1-interval connectivity property defined by Kuhn et al. [**?**].

Kuhn et al. prove that token dissemination can be solved using flooding in n rounds where n is an upper bound on the number of nodes in the networks in 1-interval connected graphs with a static participant set. With regards to the dissemination of token t, it is possible to break the graph into two subgraphs,

the nodes that have received t and the nodes that have not received t. As long as there are nodes that have not received t, the subgraph of nodes that have received t grows by at least one node per round. The graph of the network is connected so at least one node, v, from the subgraph of nodes that have received t will have at least one neighbor, v', from the subgraph of nodes that have not received t. At the beginning of the round, node v will broadcast token t to node v'.

The core principle of token dissemination using flooding holds in our network with activations and deactivations. Assume message m is being disseminated through the network instead of token t. With regards to the dissemination of message m, it is again possible to break the graph into two subgraphs, the nodes that have received m and the nodes that have not received m. We define round r'' to be any round between round r and round $r' - 1$. In our model, nodes do not activate or deactivate during a round so the intra-round change for our network will be identical to that of the static network used by Kuhn et al.

As a result, either the subgraph of nodes that have received m increased by at least one node during round r'' or that all nodes active for round r'', $A(r'')$, had already seen m at the beginning of round r''. $A(r'')$ is a superset of $A(r, r')$. If there exists a round r'' where $A(r'')$ has seen m at the beginning of the r'' then the $A(r, r')$ has received message m. Otherwise, there are $n - 1$ rounds during which the subgraph of nodes that have received m grew by at least one node. By the end of these $n - 1$ rounds, which corresponds to the end of $r' - 1$, the subgraph of nodes that have received m must be of minimum size $1 + (n - 1)$ or n. Given that n is the number of nodes, this means every node in the network, which is a superset of $A(r, r')$, receives message m by the end of round $r' - 1$.

□

Lemma 6.28. *The variable activation and deactivation reliable broadcast algorithm satisfies the first safety property of a reliable broadcast service.*

Proof. Assume node u running algorithm $\mathcal{A}$ is passed a message m by E_u through its *send* command at round r. For round r' where $r' = r + n$, every node in $A(r, r')$ will receive message m by the end of round $r' - 1$ (Lemma 6.27). By definition of the algorithm, any node that has seen message m and is active for round $r' - 1$ will pass m to its environment at $r' - 1$. Therefore, every node in $A(r, r')$, which is a subset of the nodes that have seen m and are active for $r' - 1$ will pass message m to their environment during round $r' - 1$, which is between round r and round r'.

□

Lemma 6.29. *The variable activation and deactivation reliable broadcast algorithm satisfies the second safety property of a reliable broadcast service.*

Proof. By the definition of the algorithm, a message m will contain a round r_m, which is the round that a node is expected to pass m to its environment. Assume two messages, m and m', are received by multiple nodes.

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

If m_r is equal to $m_{r'}$, then nodes will use the ID of the node that received the message m'' from its environment, m''_{ID}, 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 m_r and $m_{r'}$ executes m before m'. If m'_{ID} is greater than m_{ID}, then all nodes that receive m and m' and are active for both m_r and $m_{r'}$ executes m' before m. m'_{ID} cannot be equal to m_{ID} because a node cannot receive a message from its environment while it is waiting to acknowledge a previous message.

□

Lemma 6.30. *The variable activation and deactivation reliable broadcast algorithm satisfies the third safety property of a reliable broadcast service.*

Proof. The third safety property of reliable broadcast service requires that if a node u passes a message m to its environment, it has not passed m to its environment before and some node in the network previously received m from its environment. By definition of the algorithm, any node u maintains the set of messages that it has received with a round of execution greater than or equal to round r where r is the current global round of execution. Since node u only passes messages to its environment from its list, which will only have a single copy of each message, u will only pass each message to its environment one time.

By the definition of the algorithm, a node will only add a message m to its set that it did not receive from a neighbor if it receives m from its environment. The only way for a message m to begin being disseminated through the network is if m is passed to a node by its environment.

□

7 Experiments

In this section, we discuss our empirical evaluation. To demonstrate and evaluate our proposed reliable broadcast algorithms, we developed a trace-based simulator that simulates the activations and deactivations of nodes on different networks that vary in size and topology. The motivation behind the simulation is to demonstrate the average time performance, message complexity, goodput and message latency of our algorithms in realistic networks. These additional performance metrics are one way to differentiate the algorithms because all three of our reliable broadcast algorithms share the same worst-case time bound. Finally, we want to show the effect of network topology and size on the performance of our algorithms.

The remainder of this section is organized as follows. We begin by describing the design and architecture of the simulator. This is followed by a description

of the data sets used in our simulations. Next, we analyze the results of our simulations. Finally, we discuss the results and the next steps.

7.1 Simulator Design and Architecture

The simulator is built in Python using SimPy, a discrete-event simulator. SimPy makes it simple to create and simulate processes, which interact with each other through shared resources. The system is divided into three modules, the overlord, the nodes and the nodes' environments.

The overlord handles all the inter-node communication along with the activations and deactivations of nodes. This design decision ensures that nodes have no access to no information about their neighbors or rounds of activation or deactivation. The overlord can be reused to handle the inter-node communication and activation and deactivation of nodes for any other algorithm that runs in the synchronous broadcast model. It contains no code that is specific to any of our reliable broadcast algorithms. The overlord keeps track of global rounds and tells every node when to begin each round. If a node is not active for a round r, the overlord simply never tells the node to begin round r. Every node sends its broadcast messages for that round to the overlord, which sends a copy to each of its active neighbors.

While there is only one overlord per simulation, there are many nodes for each simulation. Each node interacts with the overlord as described above. Each node also has a connection to its own environment. The environment simulates the environment from a reliable broadcast service. A node and its environment may only interact using the *send*, *receive* or *acknowledge* commands. When an environment is not waiting on an *acknowledge* command, it will wait a random number of rounds from a range before sending a new message to the node using the *send* command. We call the upper range on the number of rounds the environment's "delay".

7.2 Data Sets

As one of our main goals of the simulations is to examine the effect of topology on the performance of our algorithms, we predominately used seven small network graphs of varying topology for our experiments. The topologies used include a clique, a lattice, a random graph, a ring, a small world, a star and a tree. With the exception of the lattice topology, which had nine nodes, all of these topologies had ten nodes.

In addition to experimenting with networks of varying topology, we also ran experiments with a random graph topology network of varying network size. We ran the simulation for networks of size 50, 100 and 150. We ran these simulations to understand the effect of the network size on the performance of our algorithms.

7.3 Results

In this section, we examine the effect of certain variables on the performances of our algorithms and we compare the performances of our algorithms to each other. First, we examine the effect of network topology on the goodput of the simultaneous activation algorithm and identify which metrics influence goodput. Next, we compare the performance of each of our algorithms with respect to goodput and message complexity in a small network. Finally, we compare the scalability of our algorithms by examining the performance metrics of our algorithms on networks of 50, 100 and 150 nodes.

7.3.1 Varying Topologies

The first graph we consider shows the performance of the simultaneous activation algorithm in networks of varying topology. The environment waits a number of rounds, which is chosen uniformly at random from the range of 5 to the delay of the environment, before passing a message to the node. We map the delay used for each execution on the x-axis of this graph. The lower the delay, the more messages will need to be disseminated and the more the network will become congested. The y-axis is the goodput, which is the total number of received messages passed to the environment during the simulation execution, which in this case lasted 1000 rounds. Each topology is plotted on the graph and defined in the legend. In the graph of goodput versus delay for Simulation Activation with regard to topology (figure 1), it is clear that the Simultaneous Activation algorithm performs much better on some topologies than others. In particular, it performs best on the clique and star topologies while it performs worst on the ring topology. This is logical because the speed of message dissemination is related to the diameter of the graph and clique and star graphs have much smaller diameters relative to size than do ring graphs.

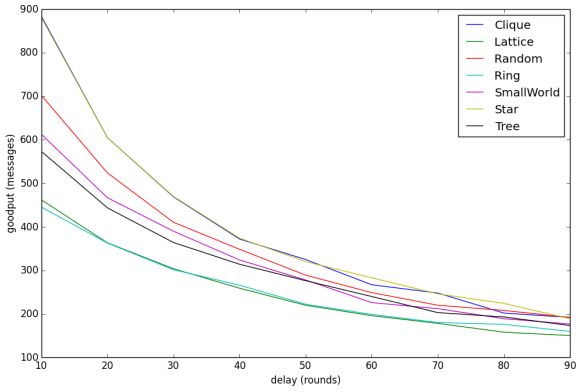

Figure 1: Goodput for various topologies

7.3.2 Algorithm Comparison

Another goal of our simulation is to differentiate between our three algorithms. This is important because each set of assumptions is a strict superset of the previous set of assumptions so the varied activation and deactivation algorithm will work not only for its set of assumptions but also for the set of assumptions for both the staggered activation algorithm and the simultaneous activation algorithm. As a result, if the earlier algorithms do not offer some type of performance benefit relative to the later algorithms, they are not useful.

The following graph (figure 2) plots the goodput of each algorithm when run on an identical network topology under the assumptions of simultaneous activation. Similar to the above graph (figure 1), the delay of the environment is the x-axis and the goodput is the y-axis. The simultaneous activation algorithm performs noticeably better than the staggered activation algorithm, which performs better than the varied activation and deactivation algorithm.

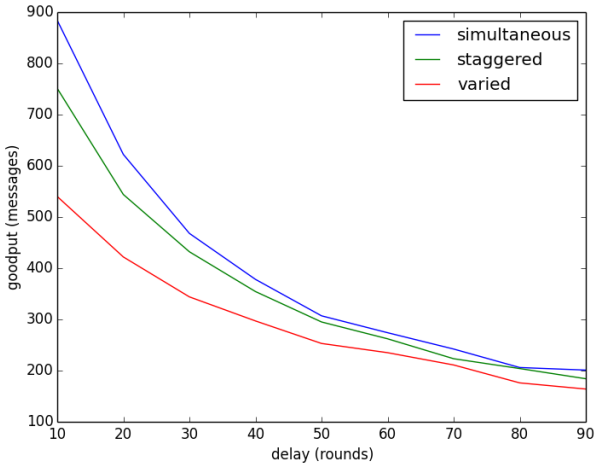

Figure 2: Goodput for all Algorithms

Next, we examine the message complexity of each of the algorithms under the same assumptions as above (figure 3). The message complexity of the varied activation and deactivation algorithm is much higher than that of the staggered activation algorithm, which is much higher than that of the simultaneous activation algorithm. The increased message complexity of the varied activation and deactivtion algorithm is logical because the algorithm is a variant of message flooding, which is known for its high message complexity.

7.3.3 Scalability

In order to demonstrate the practicality of our algorithms for larger networks, we ran simulations of the simultaneous activation, staggered activation and varied

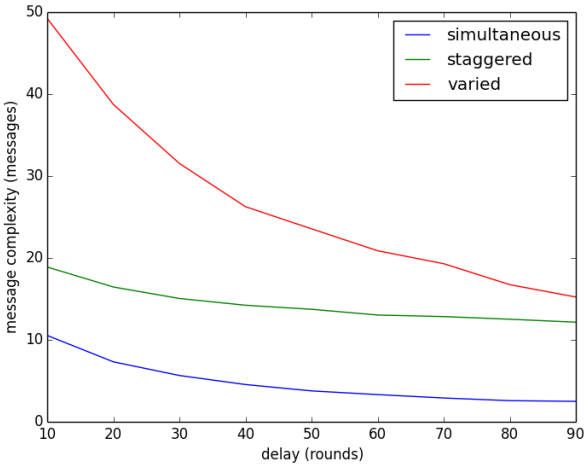

Figure 3: Message Complexity for all Algorithms

activation and deactivation algorithms on networks of size 50, 100 and 150.

7.4 Discussion

8 Conclusions