The Accelerated Ring Protocol: Ordered Multicast for Modern Data Centers

by

Amy Babay

A thesis submitted to The Johns Hopkins University in conformity with the requirements for the degree of Master of Science, Engineering.

Baltimore, Maryland

May, 2014

Abstract

Ordered multicast is an important building block for many distributed applications, and there are many existing protocols providing this service, which is often referred to as total order broadcast or atomic broadcast. However, few of the existing protocols were specifically designed for high-performance in modern data center environments. This thesis introduces a new ordered multicast protocol, called the Accelerated Ring protocol, that provides high throughput and low latency in modern data center environments.

Like other token-based protocols, the Accelerated Ring protocol places the protocol participants in a logical ring, and circulates a token around this ring. Each participant is allowed to send new multicasts to the other participants upon receiving the token. The key idea behind the Accelerated Ring protocol is that a participant can begin to send multicasts when it receives the token, but, unlike in other protocols, it may release the token before it finishes multicasting. Each participant updates the token to reflect all messages it will multicast during the current token round before beginning to multicast. It can then pass the token to the next participant at any point during the time it is multicasting. Since the token includes all the information the next participant needs, the next participant can begin multicasting as soon as it receives the token, even if its predecessor on the ring has not yet completed its multicasts. Sending the token before completing all multicasts allows the token to circulate the ring faster. This acceleration of token improves both throughput and latency.

We implemented the protocol as a prototype and evaluated its performance in 1-Gig and 10-Gig networks. We also implemented the full protocol in the open-source Spread Toolkit and evaluated its performance relative to the previous version of the toolkit in a 1-Gig local area network, virtualized 1-Gig local area settings, and a 10-Gig local area network. The results of these evaluations demonstrate the significant benefit of accelerating the token in all of these environments, both in terms of throughput and in terms of latency. The Accelerated Ring Spread implementation achieves throughput over 900 Mbps on a 1-Gigabit network and close to 2.5 Gbps on a 10-Gigabit network. At the highest throughput that both the Accelerated Ring Spread implementation and Spread 4.3 can comfortably sustain on a 1-Gig network (about 600 Mbps), the Accelerated Ring protocol reduces latency by 77% for agreed delivery and by 63% for safe delivery. On a 10-Gig network, the latency reduction is about 49% for agreed delivery and 26% for safe delivery (with throughput at 1.2 Gbps). The implementation of the protocol in Spread is open-source, and the Accelerated Ring protocol is Spread's standard protocol for data center environments and local area networks as of Spread version 4.4.

Acknowledgments

First, and most importantly, I would like to thank my family. Without their love, support, and guidance over the past (almost) 24 years, I would certainly not be where I am today. I would like to thank my parents, Kim and Jim Babay, for instilling the values that have allowed me to be succesful and for always being ready with advice or just being available to listen when I needed them. My parents taught me the value of working hard and taking pride in my work, while also reminding me to relax once in a while. Whenever the challenges of school or life feel overwhelming, a phone call to my parents can always brighten my day. I would also like to thank my sister, Emily Babay, for being a great friend and role model. Emily has always been there to share big news, as well as minor frustrations or amusing discoveries. She is one of the most driven people I know and has provided me with an outstanding example of how to set goals and make them happen. Emily's accomplishments inspire me and motivate me as I work toward my own goals. I'd also like to thank Emily for her excellent writing advice over the years, and I hope at least some of it was put to good use in this thesis.

I thank my advisor, Yair Amir, for being an incredible mentor, and for believing in my abilities, while constantly pushing me to improve. Yair's passion for teaching and dedication to his students is the reason that I am an engineer today. As I was finishing my undergraduate studies and undecided about my plans for the future, Yair volunteered his time to teach me his Distributed Systems course one-on-one and encouraged me to pursue a Masters in Computer Science. Without his committment to helping students, my career path would be entirely different. Yair's dedication to both his technical work and to making everyone around him better are things I aim to emulate in my own life, and I look forward to continuing to work with and learn from him.

I would like to thank all the other current members of the Distributed Systems and Networks lab for making it an enjoyable place to work and for always being willing to help me out. Daniel Obenshain, Tom Tantillo, Marco Platania, and Jeff DallaTezza all impress me with their intelligence and technical skills, as well as with their generosity. Even when they have their own deadlines and pressures, they are never too busy to answer my questions or help me investigate our cluster setup. Their professionalism and high standards for their work push me to be better as well.

I would also like to thank the former members of the Distributed Systems and Networks lab that I have had the opportunity to meet, including Cristina Nita-Rotaru, John Schultz, John Lane, Jonathan Stanton, Nilo Rivera, Jacob Green, and Jonathan Kirsch. Each of them has been willing to help me in whatever ways they could, and I look forward to working with many of them more in the future. I would particularly like to thank John Schultz for working with me on experimental Spread code. The opportunity to work on code with someone so talented was a great experience, and

ACKNOWLEDGMENTS

I learned a lot. I am especially grateful to John Schultz and Jonathan Stanton for working hard to get the Accelerated Ring code into the Spread open source release, despite their busy schedules.

I thank Chris Paxton for his love, support, and patience, and for helping me learn how to maintain high standards for my work without letting it take away from other important aspects of my life. I thank Amal Abukar and Reeve Hunsaker for remaining true friends despite the time that has passed and the distance between us.

Amy Babay
May 2014

This work was supported in part by DARPA grant N660001-1-2-4014. Its contents are solely the responsibility of the authors and do not represent the official view of DARPA or the Department of Defense.

Contents

Chapter 1

Introduction

Ordered multicast is an important building block for many distributed applications, and there are many existing protocols providing this service, which is often referred to as total order broadcast or atomic broadcast. However, few of the existing protocols were specifically designed for high-performance in modern data center environments. This thesis introduces a new ordered multicast protocol, called the Accelerated Ring protocol, that provides high throughput and low latency in modern data center environments. The Accelerated Ring protocol is based on the Totem Single Ring protocol [1, 2], and achieves significant performance improvements over that protocol in modern environments.

Like the Totem Ring protocol and other token-based protocols, the Accelerated Ring protocol places the protocol participants in a logical ring, and circulates a token around this ring. Each participant is allowed to send new multicasts to the other participants upon receiving the token. The key idea behind the Accelerated Ring protocol is that a participant can begin to send multicasts when it receives the token, but, unlike in other protocols, it may release the token before it finishes multicasting. Each participant updates the token to reflect all messages it will multicast during the current token round before beginning to multicast. It can then pass the token to the next participant at any point during the time it is multicasting. Since the token includes all the information the next participant needs, the next participant can begin multicasting as soon as it receives the token, even if its predecessor on the ring has not yet completed its multicasts.

However, the fact that the token can reflect messages that have not yet been sent requires careful handling of other aspects of the protocol. Retransmissions cannot be requested as soon as the token indicates that a message has been sent, since this may result in requesting many unnecessary retransmissions for messages that have not yet been received because they were not yet sent, not because they were lost.

In order to obtain the maximum benefit from passing the token early, it was also necessary to consider when the token should be processed relative to other messages in the case that a token message and other messages are received and available for processing at the same time. The token should be processed quickly, to maintain the benefit of accelerating its rotation, but if it is processed too early, unnecessary retransmissions may be requested, or excessive overlap in the sending of different participants may cause messages to be lost.

The ability to send the token before all multicasts are completed allows the token to circulate the ring more quickly, reduces or eliminates periods in which no participant is sending (due to the need to wait for the next participant to receive and process

the token before sending can begin again in the original Ring protocol), and allows for controlled parallelism in sending. As a result, the protocol is able to simultaneously provide higher throughput and lower latency than the original protocol.

A detailed analysis of the expected performance of both the Accelerated Ring protocol and the original Ring protocol is developed in this thesis. The expected throughput and latency for two levels of ordering guarantees provided by the protocol (agreed and safe delivery) are analyzed in terms of the propagation delay on the network, the time needed for a participant to send all the multicasts that it will send in a round, the time needed to complete a token rotation, the number of participants in the ring, and the protocols' flow control parameters. The analysis allows the expected throughput and latency to be calculated in the best, worst, and average cases for both protocols in both a fully-loaded system and a lightly-loaded system.

The Accelerated Ring ordering protocol is implemented as a prototype, and its performance is evaluated. The empirical performance evaluation shows that the protocol achieves 1 Gbps throughput on a 1-Gigabit network and up to 4 Gbps throughput on a 10-Gigabit network. In both settings, the Accelerated Ring protocol is able to achieve higher throughput than the original Ring protocol for all parameter settings included in the evaluation, and it is able to simultaneously provide lower latency as well. For example, on a 1-Gigabit network, for the parameter settings that allowed the Accelerated Ring protocol and the original Ring protocol to each achieve their highest observed throughput (which were different settings for each protocol), the throughput of the Accelerated Ring protocol was 1.02 Gbps, which was 21% higher than that of the original Ring protocol, and the latency of the original protocol for both agreed and safe delivery was over 2 times higher than the latency of the Accelerated Ring protocol. For parameter settings allowing the Accelerated Ring protocol to achieve 95% of its maximum throughput, its throughput is still 16.7% higher than the maximum throughput achieved by the original protocol, and latency for both agreed and safe delivery is reduced by nearly 80% relative to the original protocol. On a 10-Gig network, the Accelerated Ring protocol achieved 53% higher throughput at its observed maximum of 4.59 Gbps than the original protocol at its maximum, and provided 23% lower latency for agreed delivery and 11% lower latency for safe delivery. For parameter settings allowing the Accelerated Ring protocol to achieve over 90% of its maximum observed throughput on a 10-Gig network, it achieved 40% higher throughput than the maximum observed throughput of the original protocol, while reducing agreed delivery latency by 65% and safe delivery latency by 60%.

The full Accelerated Ring protocol is also implemented in the open-source Spread Toolkit. This implementation is evaluated and compared to version 4.3 of Spread, which uses a variant of the Totem Single Ring protocol. This evaluation was done on a 1-Gigabit local area network, a virtualized 1-Gigabit local area environment, and a 10-Gigabit local area network. In all environments tested, the Accelerated Ring version of Spread was able to achieve better throughput and latency than Spread 4.3. On a 1-Gigabit LAN, the Accelerated Ring version of Spread achieved over 900 Mbps

throughput. When both the Accelerated Ring version and version 4.3 of Spread were run at the same throughput of 700 Mbps, the Accelerated Ring protocol provided 80% lower agreed delivery latency. While virtualization introduces significant overhead for such I/O intensive tests, resulting in a 42% reduction in throughput for running in a Xen virtual machine compared to native CentOS in one experiment, the benefit of the Accelerated Ring protocol is similar in virtual environments. For example, for the same flow control parameter settings, the Accelerated Ring protocol obtained 55-60% better throughput than the original protocol in both virtualized and native environments in a test where message sizes were varied. On a 10-Gigabit network, the Accelerated Ring version of Spread achieved nearly 2.5 Gbps throughput. When both versions of Spread were run with clients sending at an aggregate rate of 1.2 Gbps, the Accelerated Ring version provided 48% lower latency for agreed delivery.

The main contributions of this work are:

1. The specification of the Accelerated Ring protocol.
2. An analytical model for the performance of this new protocol and an improved analytical model for the performance of the original Ring protocol.
3. A prototype implementation of the Accelerated Ring ordering protocol.
4. An evaluation of the prototype implementation in 1-Gig and 10-Gig local area networks.
5. A full practical implementation of the protocol in the open-source Spread Toolkit.
6. An evaluation of the practical implementation in a 1-Gig local area network, a 1-Gig local area virtualized environment, and a 10-Gig local area environment.
7. The release of the Accelerated Ring implementation in Spread as part of Spread's open-source code, and the adoption of the Accelerated Ring protocol as the standard protocol for data center and local area environments in Spread.

1.1 Thesis Organization

The remainder of this chapter surveys related work. Chapter 2 describes the system model, defining the terminology used throughout the thesis and specifying the failure model assumed and message ordering and delivery guarantees provided. Chapter 3 provides a detailed description of the Accelerated Ring protocol. Chapter 4 analyzes the expected throughput and latency of the Accelerated Ring protocol, as well as the original Ring protocol on which it is based. Chapter 5 describes the implementation and evaluation of a prototype of the Accelerated Ring protocol's ordering algorithm. Chapter 6 describes the implementation and evaluation of the complete Accelerated Ring protocol in the Spread Toolkit. Chapter 7 concludes the thesis.

1.2 Related Work

There are many existing total order broadcast protocols, and these protocols are surveyed and classified by Défago et al. in [3]. Défago et al. classify algorithms based on their mechanisms for ordering messages and define five classes of total order broadcast protocols: fixed sequencer, moving sequencer, privilege-based, communication history, and destinations agreement. Algorithms in the moving sequencer and privilege-based classes are often based on the idea of using a token circulating a logical ring, and therefore share many of the basic principles of the Accelerated Ring protocol. In token-based moving sequencer protocols, the current token-holder is the sequencer and determines the sequence of broadcast messages, but any participant may initiate a broadcast at any time. In contrast, in token-based privilege-based protocols, the initiator of a broadcast message determines its sequence in the total order, but a participant is only able to broadcast while it holds the token (and is granted the privilege to broadcast).

1.2.1 Token-based Total Order Broadcast Protocols

Work on token-based algorithms for total order broadcast has a long history. Chang and Maxemchuk introduce an early token-based protocol for total order broadcast [4]. Similarly to the Accelerated Ring protocol, this protocol circulates a token around a logical ring and uses the token to provide message ordering and learn of message stability. However, this protocol differs from the Accelerated Ring protocol in the way the token is used. Following the classification scheme of [3], the protocol of [4] is a moving sequencer protocol. The token-holder assigns a sequence number to each broadcast message and transmits an acknowledgment for the messages that both signifies its receipt of the message and conveys the messages sequence number to the other participants. In contrast, the Accelerated Ring protocol would be classified as a privilege-based protocol, in which receiving the token grants the privilege to broadcast messages.

The Pinwheel protocols order broadcast messages using a mechanism similar to that of [4], where responsibility for assigning sequence numbers rotates among participants in a logical ring, but introduce several optimizations designed to improve performance, including new mechanisms for determining message stability [5].

The protocol most similar to the Accelerated Ring protocol is the Totem single-ring ordering algorithm [1, 2]. This is a token-based privilege-based protocol in which the token is passed around a logical ring of participants, and each participant is able to initiate broadcasts while it holds the token. To create a total order on broadcast messages, the token carries the last sequence number assigned to a broadcast message, allowing the next recipient of the token to begin ordering its messages with the correct sequence numbers. Upon receiving the token, the new token-holder sends all the broadcast messages it will initiate in the current round. For each broadcast it initi-

ates, the token-holder increments the sequence number on the token, stamps the new sequence number on the message, and broadcasts the message. When it has finished broadcasting, it passes the token to the next participant, with the sequence number equal to the last broadcast initiated. The Accelerated Ring protocol also uses the token to indicate the sequence number at which a participant should begin ordering new broadcasts but introduces a new broadcasting procedure to improve the protocol's performance. The Accelerated Ring protocol employs the same mechanisms for flow control and determining message stability as the Totem single-ring ordering algorithm. The membership algorithm of Totem is used in conjunction with the Accelerated Ring ordering protocol to handle token loss, participant crash and recovery, and network partitions and merges. While it shares many similarities with the Totem ordering algorithm, the Accelerated Ring protocol is tailored for modern 1-Gigabit and 10-Gigabit networks and is able to achieve significantly better throughput and latency in these environments.

1.2.2 Recent High-throughput Total Order Broadcast Protocols

The Accelerated Ring protocol is designed to provide high throughput on modern networks. This goal is similar to that of other recent work on total order broadcast aimed at achieving high throughput. LCR [6] and Ring Paxos [7] are two protocols specifically designed to achieve high throughput, and the evaluation in [7] shows that Ring Paxos is able to achieve 900 Mbps of throughput using a 1-Gigabit switch and that LCR is able to achieve 950 Mbps. The evaluation in [7] shows that both of these protocols provide a significant performance improvement over other Paxos implementations, as well as the Spread Toolkit [8], which uses a protocol based on the Totem Ring protocol. Because the Accelerated Ring protocol shares many similarities with Totem's Ring protocol and was implemented in Spread, this thesis also provides throughput results for Spread (before the Accelerated Ring protocol was added to it) for comparison. However, the results reported in this thesis show Spread achieving much higher throughput than in the evaluation in [7]. This is likely due to the setup used in [7]. When Spread clients run on different machines than the Spread daemons, the TCP connection between the client and daemon can limit the overall throughput; much higher throughput is achieved when clients run on the same machines as the daemons and connect via IPC. The throughput results for Ring Paxos and LCR in [7] still represent an improvement over Spread's throughput based on the results for Spread's original protocol in this thesis, but the improvement is approximately 50%, not an order of magnitude. Both LCR and Ring Paxos achieve throughput similar to that of the Accelerated Ring protocol for 1-Gigabit local area networks, which is close to the maximum possible, but they use different methods to do so.

Ring Paxos is based on the Paxos algorithm [9] but employs efficient commu-

nication patterns that enable it to provide much better throughput than previous Paxos-based implementations of total order broadcast. In the basic Paxos algorithm, one participant, the coordinator, assigns sequence numbers to the messages being broadcast and sends them to the other participants (often called acceptors). Acceptors acknowledge the assignment of a sequence number to a message if it does not conflict with any assignment they previously acknowledged. Once a majority of quorum of acceptors acknowledge an assignment, the sequence number is considered to be bound to that message, and it can be delivered once all messages with lower sequence numbers have been delivered.

In Ring Paxos, acknowledgments are forwarded around a ring composed of a majority quorum of acceptors. Forwarding messages around a ring allows the protocol to achieve high throughput by avoiding the need for all acknowledgments to be sent to the coordinator (or to all acceptors, depending on the variant of Paxos) in other Paxos implementations. In addition, Ring Paxos uses IP-multicast to efficiently disseminate broadcast message content (values) and notifications that a majority quorum has acknowledged (accepted) the assignment of particular sequence numbers to particular values (decision messages). While Ring Paxos offers better performance than Paxos, it should be noted that it pays for this performance with less favorable liveness properties; if even one member of the ring fails, the ring will need to be re-formed. In some sense, this is similar to the liveness properties of Totem, in that the failure of a ring member can impede progress by forcing a reconfiguration of the ring, although Ring Paxos only needs to place a majority quorum of acceptors on the ring.

While Ring Paxos provides excellent throughput on 1-Gigabit networks, it offers different trade-offs than the Accelerated Ring protocol, which may make the Accelerated Ring protocol better-suited for certain environments. First, Ring Paxos, like other Paxos variants, requires the coordinator to propose a sequence number for each message and send each message to all other participants. This causes the load on the coordinator to be significantly higher than the load on other participants. In [7], CPU usage at the coordinator is reported to be 88%, compared with 24% for an acceptor. This may reduce the protocol's ability to take advantage of faster networks, such as 10-Gigabit networks that are in use in data centers today and likely to become more common, since processing at the coordinator could become a bottleneck, on today's processors. In contrast, the Accelerated Ring protocol allows any participant to initiate messages, and each participant does approximately the same amount of work under normal operating conditions.

In addition, the Accelerated Ring protocol offers more flexible semantics than Paxos-based approaches to total order broadcast. While Paxos requires a majority quorum to be up and connected to make progress, the Accelerated Ring protocol can continue to operate with well-defined semantics in the presence of partitions. An efficient state-machine replication service can be built on top of the Accelerated Ring protocol or the original Ring protocol of Totem, as described in [10], but the protocol can support a wider range of applications.

In LCR, both broadcast messages and acknowledgments are forwarded around a logical ring of participants. The total order is defined by logical vector clocks stamped on each broadcast message, and a participant can deliver a message m when it receives an acknowledgment for message m. Because all messages are forwarded around the ring in the order in which they are received, receiving an acknowledgment for message m ensures that a participant has already received all messages preceding m in the agreed order. LCR focuses on proving throughput optimality under a round-based communication model.

The performance evaluation in [6] shows that the protocol obtains near-optimal throughput on 100-Megabit networks, and the evaluation in [7] shows that it also achieves close to optimal throughput on 1-Gigabit networks, reporting its throughput at up to 950 Mbps. However, LCR does not use IP-multicast, leading to inefficient use of network and processing resources. In order to forward a message around the ring, each participant must both receive that message from its predecessor on the ring and send that message to its successor on the ring (with the exception that the initiator of that message does not need to receive it, and the predecessor of the initiator does not need to send it). In contrast, in the Accelerated Ring protocol, each participant must receive each message (which is required in any broadcast protocol), but only the participant that initiates the message needs to send it, since it will multicast the message to all participants. While this does not appear to harm its performance on 1-Gigabit networks, the overhead of having participants send every message may limit the protocol's ability to extend to faster networks with today's processors.

Unlike the Accelerated Ring protocol, LCR does not provide well-defined semantics when network partitions occur and requires perfect failure detection to meet its safety guarantees. LCR guarantees uniform agreement, which it defines as the property that "If any process p_i utoDelivers any message m in the current view, then every correct process p_j in the current view eventually utoDelivers m". However, if some process p_j is with process p_i in view V but is wrongly excluded from the next view (possibly because it became unreachable due to a network partition), and p_i delivers m in V, p_j is not guaranteed to deliver m. The assumption of a perfect failure detector is needed to prevent wrongly excluding a participant from a view. The Accelerated Ring protocol provides Extended Virtual Synchrony semantics [11, 10], which do not rely on perfect failure detection and allow progress with well-defined guarantees in the presence of network partitions.

Chapter 2

System Model

This chapter defines the terminology used throughout this thesis, presents the failure modes tolerated by the Accelerated Ring Protocol, and specifies the message delivery and ordering guarantees provided by the protocol.

2.1 Terminology

In this thesis, each process running the Accelerated Ring protocol is referred to as a *participant* in the protocol. Typically, each participant will run on a separate physical machine. Each participant is assumed to be able to communicate with all other participants via unicast and multicast. However, if IP-multicast is not available, a logical multicast in which the participant sending a multicast message unicasts to each other participant may be used instead, at the price of reduced performance.

A message is said to be *generated* by the application when it is created at the application level. In the model implementation described in Chapter 5, messages are generated by the protocol participants. However, in the practical implementation described in Chapter 6, messages are generated by separate client processes running a real application. Each client connects to one of the protocol participants and submits the messages it generates to that participant to be ordered.

A message is said to be *broadcast* by a participant when the participant sends that message on the network to the set of participants. Following [3], [6], and [7], among others, this thesis uses the term *broadcast* to denote the act of sending a message that is received by all members of the set of participants. However, it is important to note that the broadcast is only logical; the protocol is implemented using a combination of multicast and unicast.

A message is said to be *initiated* by a participant when it is broadcast for the first time. In the case that a message needs to be retransmitted, it may be sent by multiple participants, however each message is only initiated once.

A participant is said to *receive* a broadcast message when the message arrives on the network after being broadcast by some participant.

A participant is said to *deliver* a message when it passes the message to the application level in accordance with the ordering guarantees required for that message. The possible levels of ordering guarantees for message delivery are specified below in Section 2.3.

A *configuration* is the current set of protocol participants. All participants in a configuration agree on the members of that configuration, and the logical ring

includes all members of the configuration. The configuration is established as part of the membership protocol, which is beyond the scope of this thesis. In the model implementation described in Chapter 5, a static configuration is assumed. However, the practical implementation described in Chapter 6 handles configuration changes.

2.2 Failure Model

The Accelerated Ring protocol provides reliable, ordered message delivery and tolerates message loss. Together with the membership algorithm of Spread, which is based on the Totem membership algorithm [1, 2], the protocol also tolerates token loss, crash failure, recovery, network partition, and network re-merge, providing Extended Virtual Synchrony semantics [11, 10]. The protocol does not tolerate Byzantine faults. Because the membership algorithm is exactly the algorithm used by Spread, it is not discussed in this thesis. However, the implementation of the Accelerated Ring ordering protocol in Spread is discussed in Chapter 6. The semantics of Extended Virtual Synchrony and the algorithms used by Spread to meet these guarantees are described in [11, 10], but the relevant properties are reviewed below in Section 2.3.

2.3 Service Semantics

Five distinct levels of ordering guarantees can be provided: Reliable, FIFO, Causal, Agreed, and Safe. The guarantees of Agreed delivery subsume those of FIFO and Causal ordering and can be provided at exactly the same cost, so only Reliable, Agreed and Safe delivery are discussed. The service level required for a particular message is specified when that message is generated at the application level.

2.3.1 Reliable Delivery

Reliable delivery of a message m initiated by participant p guarantees that p will deliver m unless it crashes. If m is initiated in configuration C and no configuration change ever occurs, m will be delivered by every member of configuration C. Reliable delivery does not provide any additional guarantees regarding the order in which messages are delivered. Therefore, a participant can deliver a reliable message as soon as it is received.

2.3.2 Agreed Delivery

Agreed delivery of a message m in configuration C (where m was initiated either in C or in the previous configuration) guarantees that in addition to meeting the guarantees of Reliable delivery, for every participant p that delivers m, p has already delivered

every m' that it will deliver such that m' precedes m in the agreed order. Unlike reliable messages, agreed messages may not be able to be delivered immediately after they are received, if some preceding messages have not yet been delivered. The agreed order is defined by the following properties:

1. Each message m has a unique position in the agreed order (no message is delivered at multiple positions, and no two messages are delivered at the same position).

2. A message m is delivered at the same logical time by all participants that deliver m. If participants p and q both deliver m and m', and p delivers m' before m, q delivers m' before m, even if p and q deliver m and/or m' in different configurations.

3. If m' precedes m in the causal order for the configuration in which m was initiated, m' precedes m in the agreed order. The causal order is defined by the following properties:

   (a) Message m' precedes m in the causal order if participant p initiates m' in C before p initiates m in C.

   (b) Message m' precedes m in the causal order if participant p delivers m' in C before p initiates m in C.

The set of messages a given participant p is obligated to deliver is defined more precisely by the Extended Virtual Synchrony model in [11, 10], but the following conditions must hold:

1. (Causality) If message m causally depends on m' (i.e. m' precedes m in the causal order for the configuration in which m was initiated), p must deliver m' before delivering m.

2. (Agreed Order) If some participant q that delivered the same configuration C as p delivered m' in C before delivering m in C, p must deliver m' before delivering m.

3. (Virtual Synchrony) If participants p and q both deliver the same two consecutive configurations, they deliver exactly the same set of messages in each of those configurations.

In the context of a single configuration, where partitions are assumed not to occur (or one partition is treated as the primary component, and any participant that does not belong to the primary component is considered faulty), Agreed delivery satisfies

the requirements of non-uniform total order broadcast, as specified by Défago et al. [3].

However, because the Accelerated Ring protocol tolerates partitions and allows members of all partitions to initiate new broadcasts, it provides a somewhat different semantics when partitions are allowed. No protocol that allows all partitions to make progress (rather than allowing only a single partition designated as the primary component to make progress while all other partitions are blocked) can satisfy the (nonuniform) Agreement property of [3] when partitions are allowed. The nonuniform Agreement property states "If a correct process TO-delivers a message m, then all correct processes eventually TO-deliver m", where *TO-deliver* denotes the act of delivering a message in accordance with the total ordering guarantees. In the Accelerated Ring protocol and other protocols allowing progress in multiple partitions, correct participants may not deliver the same set of messages when messages are sent while the participants are in separate partitions or just before a partition occurs. The semantics of Extended Virtual Synchrony, in contrast, specify precisely which messages participants are guaranteed to deliver based on the sequence of configurations of which they are members. Services that require blocking non-primary components, such as state-machine replication, can be built on top of the Accelerated Ring protocol as in [10], but by providing Extended Virtual Synchrony semantics, the Accelerated Ring protocol is able to offer a richer model with well-defined guarantees when network partitions occur.

2.3.3 Safe Delivery

Safe delivery provides all the guarantees of agreed delivery and additionally guarantees that if any participant p delivers a safe message m in configuration C, p knows that all members of C have received and will deliver m, unless they crash.

Similarly to the relationship between Agreed delivery and non-uniform total order broadcast, Safe delivery satisfies the requirements of uniform total order broadcast as specified in [3] when partitions are assumed not to occur. However, the uniform agreement property holds only for members of the configuration in which the message was delivered. A participant that delivers a safe message in configuration C knows that all members of C will deliver the message, unless they crash, but participants that are not members of C may never deliver the message, even if they never crash.

Chapter 3

Accelerated Ring Protocol

The Accelerated Ring protocol is designed to provide a reliable, totally ordered broadcast service with high throughput, low latency, and flexible semantics in modern data center environments. The Accelerated Ring ordering protocol is integrated with Spread [8] to provide a complete group communication system that guarantees Extended Virtual Synchrony semantics.

This chapter describes the operation of the protocol in the normal case, without token loss, partitions, or crashes. Failures are not addressed in this thesis, since these are handled by the membership algorithm, which is unchanged from the variation of the original Totem Ring protocol [1, 2] implemented in Spread.

3.1 Protocol Description

The main idea behind the ordering algorithm is that a token circulates a logical ring composed of the participants in the protocol, and each participant can initiate broadcasts upon receiving the token. The information carried by the token allows each participant to correctly assign sequence numbers to its messages in the total order. This same basic principle has been used by existing total order broadcast algorithms. Défago et al. classify algorithms of this type as *privilege-based* and survey several other existing algorithms based on the idea of circulating a token that gives the token-holder the privilege to broadcast [3].

However, the Accelerated Ring protocol differs from typical privilege-based protocols in that a participant does not need to maintain possession of the token for the duration of its broadcasting period. Once a participant has updated the token with the information that the next participant in the ring will need to correctly sequence its broadcasts, it is able to release the token, while continuing to broadcast messages. This improves throughput by allowing for controlled parallelism in broadcasting and reducing periods in which no process is broadcasting. If the token can only be passed once a participant has finished broadcasting, there is necessarily a period in which no broadcasting can occur between the time that one participant finishes sending and the time that the next participant receives and processes the token. The Accelerated Ring Protocol, in contrast, allows for continuous broadcasting. The number of messages sent after the token can be tuned to allow for a controlled amount of overlap in the broadcasting of neighbors on the ring.

The following description assumes a static set of participants, with no participant failures, network partitions, or token loss, although the addition of the membership

algorithm removes these restrictions. The description assumes that the current membership of the ring has been established and some participant has sent the first regular token. Under these assumptions, there are two cases in which a participant performs an action: it receives the token, or it receives a broadcast message.

3.1.1 Message Types

The Accelerated Ring protocol uses two types of messages: broadcast messages and token messages. The two message types carry different information and trigger different actions in the protocol.

3.1.1.1 Broadcast Messages

Broadcast messages carry the application content that the protocol is being used to send. Each broadcast message has the following fields:

1. *seq*: The sequence number of this message. This corresponds to its position in the total ordering.
2. *pid*: The ID of the participant that initiated this message. Each participant has a unique ID between 1 and R, where R is the total number of participants (the size of the ring). This ID corresponds to the participant's position in the ring.
3. *round*: The token round in which this message was initiated. The use of this field is discussed in Section 3.1.4.
4. *payload*: This is the content of the message. This is generated by the application and is not used or inspected by the protocol.

3.1.1.2 Token Messages

Token messages pass control information among the protocol participants that is used to ensure a correct total ordering and provide flow control. The token carries the following fields relevant to the ordering protocol:

1. *seq*: The last sequence number claimed by a participant. The token recipient may initiate broadcasts with sequence numbers starting at $seq + 1$.
2. *aru* (all-received-up-to): This field is used to determine when a broadcast message has been received by all participants, and therefore may be delivered according to the semantics of Safe delivery and/or garbage-collected.
3. fcc (flow control count): The total number of messages broadcast during the last rotation of the token.

4. *rtr* (retransmission requests): The list of sequence numbers of messages being requested for retransmission (because they were lost by some participant).

3.1.2 Token Handling

Upon receiving the token, a participant broadcasts messages, updates the token fields, passes the token to the next participant, and delivers and discards newly deliverable or unneeded messages. A key feature of the Accelerated Ring protocol is that the participant does not need to broadcast all of the messages that it will send during the current token rotation prior to passing the token. To allow this, broadcasting is split into pre-token and post-token phases.

3.1.2.1 Pre-token Broadcast Phase

In the pre-token phase, the participant answers any retransmission requests that it can and determines the full set of new broadcasts it will initiate during this round. Therefore, by the end of the pre-token broadcast phase, the participant knows exactly what messages it will send during the current token round. All retransmissions are sent during the pre-token broadcast phase, but new broadcasts may be sent in either the pre-token or post-token broadcast phase.

The number of new broadcasts that the participant is allowed to initiate depends on the configurable flow control variables $Global\_window$ and $Personal\_window$. The $Global\_window$ is the maximum total number of broadcasts that can be initiated during one rotation of the token. The $Personal\_window$ is the maximum number of broadcasts that can be initiated by a single participant during one rotation of the token. The number of new broadcasts the participant can initiate during the current round is computed as the minimum of the number of generated messages that this participant is responsible for broadcasting but has not yet broadcast, $Global\_window$, $Personal\_window$, and $Global\_window + Personal\_window - fcc$, where fcc is the value of the fcc field on the token the participant received.

The number of broadcasts that will actually be sent during the pre-token phase depends on the configurable parameter $Accelerated\_window$. The $Accelerated\_window$ denotes the maximum number of messages a participant is allowed to broadcast *after* passing the token to the next participant. Thus, the number of messages sent during the pre-token phase will be the total number of broadcasts the participant can send this round, minus the $Accelerated\_window$. If the participant will send $Accelerated\_window$ or fewer messages this round, no new messages are broadcast during the pre-token phase.

To determine the set of messages that it will send during the current round, which is necessary to properly update the token, while broadcasting only those messages that must be sent during the pre-token phase, the participant creates a queue of messages. This queue will hold at most $Accelerated\_window$ messages at a time. The participant

proceeds to prepare messages as if it were going to send all of its messages prior to passing the token. However, for each message that it prepares, instead of immediately sending that message, it places it in its queue. If this causes the queue to contain more than *Accelerated_window* messages, it removes the first message in the queue and broadcasts it. This procedure continues until the participant has enqueued all of the messages it is able to send in the current round, which ends the pre-token broadcast phase. Thus, at the end of the pre-token broadcast phase, the queue holds either *Accelerated_window* messages or all the messages that the participant is able to send that round (if it can send fewer than *Accelerated_window* messages, either because of flow control constraints or because it has fewer than *Accelerated_window* messages waiting to be sent).

Because the computation required to prepare and enqueue a message takes much less time than actually sending a message, this queueing adds only a trivial amount of time and processing to the total token-handling time, compared with the process of sending all messages before the token without queueing. When the *Accelerated_window* parameter is a large fraction of the total messages a participant broadcasts in a round, the pre-token broadcast phase is very short, and the token is passed to the next participant quickly.

3.1.2.2 Updating the Token

The *seq* field of the token is set to the highest sequence number that has been assigned so far. For each new broadcast message enqueued during the pre-token broadcast phase, *seq* is incremented, and the resulting sequence number is assigned to the broadcast message being prepared. Because *seq* is updated as messages are enqueued, and all messages that will be sent by this participant in this round are enqueued before this participant releases the token, the *seq* field on the token released by a participant reflects the highest sequence number that the participant will broadcast during the current round, including both the pre-token and post-token broadcast phases.

The *aru* field is updated according to the rules in [1]. Each participant tracks its local *aru*, which is the highest sequence number such that this participant has received all messages with sequence numbers less than or equal to that sequence number. If the participant's local *aru* is less than the token *aru*, it sets the token *aru* equal to its local *aru*. If the participant had lowered the token *aru* on a previous round and the received token's *aru* has not changed since it set it, it sets the token *aru* equal to its local *aru*. If the received token's *aru* and *seq* fields are equal, and this process does not need to lower the *aru*, the token *aru* is incremented with *seq* as new broadcasts are initiated.

The fcc field is also updated as in [1]. The total number of retransmissions and new broadcasts this participant sent in the previous round are subtracted from the received token's fcc value, and the total number of retransmissions and new broadcasts the participant is sending in the current round are added to the received

token's fcc value.

The rtr field is updated to remove retransmission requests that the participant answered in this round and to add the sequence numbers of any messages that this participant has missed. The fact that participants can pass the token before completing their broadcasts for the round slightly complicates retransmission requests, since the seq field of the received token may include messages that have not actually been sent yet. In order to avoid unnecessarily retransmitting messages, the participant only requests retransmissions for missing messages up through the value of the seq field on the token it received in the previous round, rather than the value on the token it just received. The mechanism for guaranteeing that all received messages with sequence numbers up through the seq value of the previous token have been processed by this participant is discussed in Section 3.1.4.

The updated token is then sent to the next participant in the ring before the post-token broadcast phase begins.

3.1.2.3 Post-token Broadcast Phase

During the post-token broadcast phase, the participant simply flushes the queue created in the pre-token broadcast phase by broadcasting each message remaining in the queue. This completes the participant's broadcasting for the current round.

3.1.2.4 Delivering and Discarding

As the final step of token handling, the participant delivers any messages that are newly able to be delivered in accordance with their required semantics and discards any delivered messages that are now known to have been received at all participants and therefore will never be requested for retransmission. The procedure the Accelerated Ring protocol uses to determine which messages can be delivered and discarded is the same as the procedure described in [1, 2] and is described below.

A participant can deliver an agreed message once it has received and delivered all messages with lower sequence numbers. Thus, if a participant has received and delivered all messages up through the seq value of the received token, it will be able to immediately deliver any agreed messages that it broadcasts in the current round.

A participant can deliver a safe message once it has received and delivered all messages with lower sequence numbers and knows that all other participants have received and will deliver the message (unless they crash). Therefore, the participant can deliver all safe messages with sequence numbers less than or equal to the minimum of the aru on the token it sent this round and the aru on the token it sent last round. Since every participant had the opportunity to lower the aru during the round, every participant must have at least the minimum of these two values.

Every message that meets the requirement for safe delivery can be discarded after being delivered. Since safe delivery requires that every participant has the message,

it will no longer be requested for retransmission.

3.1.3 Broadcast Handling

Upon receiving a broadcast message, a participant inserts it into its buffer of messages, ordered by its sequence number. If this message allows the participant to advance its local *aru* (i.e. the sequence number of the packet is equal to the local $aru + 1$), it delivers all undelivered messages in the order of their sequence numbers until it reaches a sequence number higher than the local *aru* (i.e. a message it has not received) or a message requiring safe delivery. Messages requiring safe delivery cannot be delivered until the token *aru* indicates that they have been received by all participants. To maintain the total order, no agreed messages with sequence numbers higher than that of the first undelivered safe message can be delivered until after that safe message is delivered.

3.1.4 Selecting a Message to Handle

In general, broadcast and token messages are handled as they arrive. However, when messages arrive simultaneously or nearly simultaneously, it is possible to have both token and broadcast messages available for processing at the same time. In this case, the protocol must decide which message type to handle first. Logically, this is accomplished by assigning different priorities to the message types at different times. A message type is said to have *high priority* if no messages of another type will be processed as long as some message of that type is available to be processed. Therefore, when broadcast messages have high priority, no token message will be processed as long as a broadcast message is available to be processed, and vice versa. The implementation of the message type priorities is discussed in Chapters 5 and 6.

The key issue is that a participant should not process a token until it has received and processed all the broadcast messages that receiving that token will cause it to request for retransmission (or as many as it will receive, in the case that there is loss and legitimate retransmission requests are needed). If these messages have not yet been processed, but were not actually lost, they will be unnecessarily requested. However, in order to keep the token moving as quickly as possible and maximize performance, the token should be processed as soon as it is safe to do so without causing unnecessary retransmissions. Note that decisions about when to process messages of different types can impact performance but do not affect the correctness of the protocol.

There are two methods that can be used for deciding when to raise the priority of token messages. The first method allows the token to circulate more quickly but provides less strict flow control, while the second method is more conservative. The first method allows a participant to read the token as soon as it receives and processes the first broadcast message from the participant that will send it the token, while the

second method does not allow a participant to read the token until it has received and processed the first message that its predecessor sent after sending the token. The second method ensures that a participant processes all messages sent before the token before processing the token (assuming messages generally arrive in the order in which they were sent). The first method only ensures that a participant will process all messages that it may request for retransmission before processing the token.

A participant always gives broadcast messages high priority upon receiving and processing a token. The only difference between the two priority-switching methods is when token messages are given high priority.

In the first method, each participant maintains a variable *Received_token_rounds* that it increments each time it receives the token. Each broadcast message is stamped with its sender's current value of *Received_rounds* when it is prepared to be sent.

A participant gives the token high priority after receiving a broadcast message stamped with a round number higher than its current value of *Received_token_rounds* from its immediate predecessor in the ring.

Switching priorities according to these rules guarantees that at the time that a participant gives the token high priority, it has processed all (received) messages sent by its predecessor in the previous round (since all of those messages must have been sent before the first broadcast message it sent for this round). Moreover, this guarantees that the participant has processed all (received) messages sent by any participant in the previous round, since all other participants in the ring would not have passed the token in the current round before sending all of their messages from the previous round.

Notice that it is not sufficient for a participant to wait until it receives a broadcast message with a round number higher than its *Received_token_rounds* from any other participant to switch the token to high priority. It must wait for such a message from its immediate predecessor in the ring.

To see why this is the case, consider a ring of participants A, B, C, where the token is passed from A to B to C and back to A. Consider the n^{th} rotation of the token. Receiving the token in this round will cause a participant to request any missing messages that were sent during the $(n-1)^{th}$ rotation of the token.

Assume C has not yet received the token in rotation n. C should not give the token high priority upon receiving a broadcast from A marked with round n. This is because B may still be broadcasting in its post-token phase of rotation $n-1$ up until it sends its first broadcast in token rotation n. This means that A may be sending round n broadcasts while B is still sending round $n-1$ broadcasts.

Although B will not actually release the token in round n until it is completely finished with its round $n-1$ broadcasts, the fact that B's round $n-1$ broadcasts may overlap with A's round n broadcasts can cause unnecessary retransmissions if C is not processing messages at line speed. It is possible that C simultaneously has A's round n broadcasts, B's round $n-1$ broadcasts, and B's round n broadcasts (in that order) waiting to be processed, as well as the n^{th} token. In this case, if C changes the token

to high priority upon processing the first round n message from A, it will process the token before processing any more of the waiting broadcast messages, causing it to request the waiting round $n-1$ messages from B. Therefore, a participant can only give token messages high priority upon receiving a broadcast message in for the next round from its immediate predecessor. Notice, however, that this means that if the predecessor of some participant never initiates broadcasts, that participant will never give the token high priority.

In the second method, each participant maintains a variable *Received_token_rounds* that it increments each time it receives the token, as in the first method. Each participant also maintains a variable *Sent_token_rounds* that it increments each time it sends the token. Each broadcast message is stamped with its sender's current value of *Sent_token_rounds* just before it is sent. Because messages are not stamped until just before being sent, all messages that a participant sends before sending the token will have a lower round number than those sent after the token.

A participant gives the token high priority after receiving a broadcast message stamped with a round number higher than its current value of *Received_token_rounds* from its immediate predecessor in the ring.

Switching priorities according to these rules not only guarantees that a participant has processed all (received) messages sent in the previous round, but also guarantees that it has processed all (received) messages sent before the token by its predecessor in the current round, since its predecessor does not increment the *Sent_token_rounds* variable that it stamps on its messages until it has sent the token.

Notice that for an accelerated window of 0, where no messages are sent after the token, broadcast messages will never be given high priority using the second method, and the flow control behavior is the same as that of the original Ring protocol as implemented in Spread. However, broadcast messages will be given high priority even when the accelerated window is 0 if the first method is used. If all messages are sent after the token (i.e. the accelerated window is equal to the personal window), the two priority switching methods are equivalent.

If all participants are able to receive and process messages at the rate at which they are sent, the priority switching method does not have an impact, since the token will be received, and therefore processed, after all messages sent before it have been received and processed. However, when participants do not process messages at exactly the rate they arrive and can receive the token before they have finished processing all previously received messages, the two priority-switching methods offer different performance trade-offs. The first maximizes the speed with which the token can circulate the ring. The second slows the token slightly to reduce the number of unprocessed broadcast messages that can build up at a participant but still maintains the acceleration of the token by processing it at its correct place in the stream of messages (after messages sent before it, but before messages known to have been sent after it).

Chapter 4

Expected Performance Analysis

Allowing each participant to pass the token before completing its broadcasts for the round allows the token to circulate around the ring faster and allows a controlled amount of parallel broadcasting, which enables the Accelerated Ring protocol to achieve high throughput. However, the fact that broadcast messages are not guaranteed to be sent at the time that the token's *seq* field is updated to include that message can increase latency in terms of token rotations for certain operations. However, it should be noted that since the token circulates faster, each round takes less time. Because of this, an increase in latency in terms of token rotations does not necessarily correspond to real increase in latency, and can even correspond to a decrease in real latency. The following section presents an analytical model for the expected performance of both the Accelerated Ring protocol and the original Ring protocol of [1, 2]. Empirical evaluations of the protocol's effect on throughput and broadcast delivery latency are presented in Chapters 5 and 6.

4.1 Definitions

Expected performance is analyzed below in terms of the parameters R, B, P, and N. R denotes the time needed to complete one full rotation of the token and depends on the number of participants in the ring, as well as the number of messages each participant is allowed to send in each round. B denotes a *broadcasting period*, which is the time required for a single participant to consecutively send the maximum number of messages allowed in a round (including the token). B can be tuned by adjusting the $Personal\_window$ parameter, since the $Personal\_window$ is the maximum number of messages a participant can send in round. P denotes the propagation delay, or the time required for a message to be received by all participants after it is sent. The analysis assumes that the propagation delay between each pair of participants is the same. N denotes the number of participants in the ring.

4.2 Throughput

In the original Ring protocol, each participant can only broadcast while it holds the token, and a participant only passes the token to the next participant in the ring after it has finished broadcasting. Because of this, no broadcasts can be sent during the time between one participant finishing its broadcasts for the current round and

the next participant receiving and processing the token and beginning to broadcast. Moreover, at most one participant can broadcast at any given time.

When the time between one participant sending the token and the next participant beginning to broadcast is short, the fact that no broadcasting occurs during this time has little impact on performance. When a single process can saturate the network bandwidth, allowing only one participant to send at time does not reduce performance and can provide useful flow control. However, in modern data center environments, these conditions are not met. Modern 1-Gigabit and 10-Gigabit switches introduce latency due to the buffering used to achieve high throughput. Virtualized environments introduce additional latency due to the need to access hardware through the hypervisor, rather than directly. In addition, on 10-Gigabit networks, a single process sending from a single thread is typically unable to saturate the network using a common processor (for 2014) and sending messages of normal sizes (i.e. without using jumbo frames).

The Accelerated Ring protocol reduces the impact of latency and allows for controlled parallelism in sending. Since participants do not need to wait until they have finished broadcasting for the current round before passing the token to the next participant, the broadcasting of one participant can overlap with the time needed for the next participant to receive and process the token. Furthermore, the number of messages allowed to be sent after releasing the token can be increased to enable two or more participants to broadcast simultaneously, although this must be tuned to avoid loss due to too many participants broadcasting at once.

To analyze this improvement in throughput, consider the total bytes that can be broadcast during one rotation of the token and the time required to complete a token rotation. Assuming a fully-loaded system, where each participant sends the maximum number of messages it is permitted each time it receives the token, each participant will send $Personal\_window$ messages in a round, since the $Personal\_window$ is the protocol parameter that determines the maximum number of messages a participant may send in a round. This analysis assumes that the $Window$ protocol parameter, which limits the total number of messages broadcast in a round is set high enough to allow all participants to initiate a full $Personal\_window$ of broadcasts. Assuming that all messages contain the same number of bytes (Msg_size), total number of bytes broadcast in a round is $N \times Personal\_window \times Msg\_size$. For the same number of participants, personal window, and message size, exactly the same number of bytes will be broadcast in a single token round in both the original and accelerated protocols.

However, the time to complete a token rotation (R) differs for the original Ring protocol and the Accelerated Ring protocol. Therefore, in comparisons of the two protocols, R_O is used to denote the time to complete a token rotation in the original protocol, while R_A is used to denote the time to complete a token rotation in the Accelerated Ring protocol. In both protocols, the time to complete a token rotation can be calculated by adding the time a single participant holds the token after receiving

it to the time it takes the next participant to receive the token after it is released, and multiplying the result by the number of participants. In the original Ring protocol, a participant holds the token for the duration of its broadcasting period, while in the Accelerated Ring protocol participants may pass the token before completing their broadcasts.

Calculating R_O in terms of B, P, and N is straightforward. In a single round in a fully-loaded system, each participant sends the maximum number of messages and then passes the token. The next participant will begin broadcasting once it receives the token. The amount of time required to send the maximum number of allowed messages is defined to be B, and the time required for the next participant to receive the token after its predecessor on the ring finishes broadcasting is the propagation delay P. Therefore, the total time to complete a round in the original protocol is $(B + P) \times N$.

In the Accelerated Ring protocol, a participant does not need to send all of its broadcasts before passing the token. The *Accelerated_window* protocol parameter determines the number of messages (out of the *Personal_window*) that can be sent after passing the token. Therefore, a participant sending a full personal window of messages will send $Personal\_window - Accelerated\_window$ messages, pass the token, and then send *Accelerated_window* more messages. The time that a single participant holds the token is therefore $\left(1 - \frac{Accelerated\_window}{Personal\_window}\right) B$. $\frac{Accelerated\_window}{Personal\_window}$ gives the fraction of the personal window sent after the token, so $\left(1 - \frac{Accelerated\_window}{Personal\_window}\right)$ gives the fraction of the personal window sent before the token. Since B gives the time required to send the entire personal window, $\left(1 - \frac{Accelerated\_window}{Personal\_window}\right) B$ gives the time to send the fraction of the personal window sent before the token, which is the time that a single participant holds the token. As in the original protocol, one propagation delay is required for the next participant to receive the token after it is released, so the total time need to complete a round is $\left(\left(1 - \frac{Accelerated\_window}{Personal\_window}\right) B + P\right) \times N$. However, this assumes that a participant is able to complete its broadcasting period before the next time it receives the token. When a large fraction of the personal window is sent after the token and the propagation delay is short, this may not be the case. The protocol enforces that a participant must complete its broadcasting period before processing the next token, so the time to complete a round is actually $\max\left(\left(\left(1 - \frac{Accelerated\_window}{Personal\_window}\right) B + P\right) \times N, B\right)$.

Thus, throughput for the original Ring protocol can be expressed as:

$$\frac{N \times Personal\_window \times Msg\_size}{R_O} = \frac{N \times Personal\_window \times Msg\_size}{(B + P) \times N}$$

Throughput for the Accelerated Ring protocol can be expressed as:

$$\frac{N \times Personal\_window \times Msg\_size}{R_A} = \frac{N \times Personal\_window \times Msg\_size}{\max\left(\left(\left(1 - \frac{Accelerated\_window}{Personal\_window}\right) B + P\right) \times N, B\right)}$$

When the accelerated window is 0, the accelerated ring protocol behaves like the original protocol. As the accelerated window increases, up to the point that it is equal to the personal window, the time needed to complete a round decreases, causing throughput to increase. In practice, setting the accelerated window equal to the personal window is not always the best parameter setting, since too much overlap in the broadcasting periods of different participants can cause message loss, but in general, the experimental results in Chapters 5 and 6 confirm that throughput increases as the accelerated window increases.

4.3 Retransmission Latency

If participants send all broadcasts before passing the token, token recipients can assume that they should have received all broadcast messages with sequence numbers less than or equal to the value of the *seq* field on the token. However, in the Accelerated Ring protocol, some messages with sequence numbers less than or equal to the *seq* field may not have been sent yet. A token recipient is only guaranteed that all broadcast messages with sequence numbers less than or equal to the token it received in the *previous* round have been sent.

In order to avoid requesting retransmissions for messages that have not been received, not because they were lost, but because they were not sent before the token was received, participants only request retransmission of messages with sequence numbers up through the *seq* value of the previous token. However, in the case that a broadcast message is actually lost, it will take one additional token rotation for the message to be requested and retransmitted.

In the data center environments for which the protocol is designed, appropriate flow control parameters result in a negligible number of retransmissions. Therefore, the additional latency for receiving messages that need to retransmitted does not affect performance during normal operation. A wide range of settings for the $Personal\_window$ and $Accelerated\_window$ flow control parameters can be used to provide excellent throughput with no loss.

4.4 Agreed Delivery Latency

Agreed delivery latency is defined as the time required for a message initiated by a randomly selected participant q to be delivered in agreed order by a randomly selected participant p, measured from the time that the message is generated. Recall that a message is said to be *generated* when it is created at the application level and *initiated* when it is broadcast to the set of participants.

For a participant to deliver a message in agreed order, the participant must have received that message, and have received and delivered all messages that precede it

in the agreed order. In the normal course of operation, receiving and delivering all preceding messages means receiving and delivering all messages with lower sequence numbers. From the time that a message with sequence number n is generated, to the time that it is delivered by some participant p, several events must occur:

1. The participant q that will initiate message n must receive the token.
2. All messages with sequence numbers lower than n must be broadcast.
3. Message n must be broadcast.
4. Participant p must receive and deliver all messages with sequence numbers lower than n.
5. Participant p must receive message n.

The agreed delivery latency is analyzed below in terms of the parameters R, B, P, and N defined in Section 4.1. This analysis assumes that no messages are lost. It also assumes that messages are not generated faster than they can be broadcast. That is, participants do not build up a queue of messages waiting to be initiated; a message can always be broadcast the next time the process that will initiate it receives the token.

4.4.1 Receiving the Token

Because a participant cannot initiate new broadcasts until it receives the token, a newly generated message n cannot be broadcast until the next time that the participant q that will initiate it receives the token. As a factor of R, the expected time to receive the token does not depend on whether the original or accelerated protocol is used or the load on the system. Note however, that the value of R depends on both the protocol used and the load on the system, since a single token round can be completed in less time when fewer messages are sent in that round, and each round in the Accelerated Ring protocol generally takes less time than a round in the original Ring protocol.

In the worst case, the initiator q releases the token immediately before message n is generated and must wait one full token round to receive the token and broadcast this message. In the best case, the participant initiating the message receives the token just after message n is generated and can broadcast it immediately. On average, the participant initiating the message is expected to receive the token half a round after the message is generated.

4.4.2 Broadcasting n and Preceding Messages

To deliver a message in agreed order, a participant must first deliver all preceding messages. Since a message clearly cannot be delivered before it is sent, both message n and all messages preceding n in the agreed order must be broadcast before n can be delivered. This section analyzes the time needed for n and all preceding messages to be broadcast.

First consider a single message introduced in a lightly loaded system. In this case, we can assume that all previous messages have already been delivered at the time that the message is initiated, and the message's initiator does not have any other messages to initiate. In this case, no additional time is needed to broadcast messages preceding n in the agreed order, and the participant initiating message n can broadcast it as soon as it receives the token. This applies to both the original and accelerated protocols.

Now, consider a more heavily loaded system, in which each participant sends a full burst of messages on each token receipt. In the original protocol, all participants other than the initiator of message n have already sent all the messages they will initiate for this round, since they do not pass the token until they have done so. In the best case, n is the first message the initiator will broadcast, and it is sent immediately after the initiator receives the token. In the worst case, this is the last message the initiator will broadcast, meaning it is sent at the end of the initiator's broadcasting period, so it requires one additional broadcasting period after the initiator receives the token for this message and all previous messages to be sent. On average, the message is expected to be broadcast halfway through the initiator's broadcasting period or $\frac{1}{2}B$ after the initiator receives the token.

The analysis for the Accelerated Ring protocol is more complex, due to the fact that the token can be (and typically is) passed before all messages reflected in its *seq* value have been sent. As in the original protocol, message n is the first one sent in the initiator's broadcasting period in the best case, last in the worst case, and is sent halfway through its initiator's broadcasting period on average. However, when the message is sent, it is not guaranteed that all messages with lower sequence numbers have been sent. All messages sent during the broadcasting periods of the other participants on the ring in the current round must be sent (and received) before this message can be delivered. However, since those broadcasting periods can overlap with one another, as well as with the broadcasting period of this message's initiator, some messages with lower sequence numbers may not be sent until after message n has been sent, and therefore will not be received until after message n has been received (assuming equal propagation delays for all messages).

In the worst case, n is the last message sent in its initiator's broadcasting period, so the time for it to be sent after q receives the token is exactly B. Under the reasonable assumption that each participant takes a similar amount of time to broadcast (i.e. that B is the same for all participants), all previous messages will be sent by the

end of the initiator's broadcasting period, since all other participants started their broadcasting periods before the initiator and therefore will finish their broadcasting periods by the time the initiator finishes broadcasting. This means that all preceding messages must be sent by the time n is sent, so the time for n and all preceding messages to be sent after q receives the token is exactly the time for n to be sent, which is B.

While this analysis assumes that all participants take the same amount of time to broadcast, which is assumed to be the normal case, notice that if some participant is slow, the time for all preceding messages to be broadcast is bounded by $2R$. Each participant must finish its broadcasting for the previous round before it processes the next token, so every message initiated in a given round is guaranteed to be sent before its initiator releases the token in the next round.

For the average and best cases, it is not necessary to wait for the initiator's entire broadcasting period to end. However, it is not enough to just wait for this message to be sent, since the initiator's broadcasting period may overlap with the broadcasting periods of other participants, and all preceding messages must be sent before this message can be delivered in agreed order.

Therefore, to analyze the best and average cases, it is necessary to consider the amount of overlap between the broadcasting periods of message n's initiator and the other participants in the ring. Since all participants are assumed to take the same amount of time to broadcast, amount of time that q's broadcasting period overlaps with any previous broadcasting period is just the amount of time that it overlaps with the broadcasting period of q's immediate predecessor on the ring. Any previous broadcasting period started before that of q's predecessor and therefore will end before q's predecessor finishes broadcasting.

To account for this overlap, the variable X is introduced. X corresponds to the time required for a participant to finish broadcasting after its successor on the ring has received the token. To calculate the value of X, recall that N denotes the number of participants in the ring. Therefore, $\frac{R}{N}$ is the time that a single participant holds the token during a round. That is, from the time that a participant receives the token, the amount of time it takes for its successor on the ring to receive the token is $\frac{R}{N}$. B denotes the amount of time it takes for a single participant to send the maximum number of broadcasts it is allowed for a single round. Therefore, $B-\frac{R}{N}$ corresponds to the amount of time needed for a participant to finish broadcasting after its successor on the ring has already received the token, which is exactly X. However, $B-\frac{R}{N}$ may be negative; it is possible that each participant finishes sending before its successor receives the token, so there is no overlap in sending. This occurs when the time a participant takes to send its remaining messages after it releases the token is less than the propagation delay needed for the token to reach the next participant. To account for this case, X is defined as $\max\left(B-\frac{R}{N}, 0\right)$.

The above definition calculates X by considering the entire duration of a participants broadcasting period, from the time it receives the token to the time it releases

the token, and subtracting the time that it takes for its successor on the ring to receive the token, from the time it received the token. Intuitively, X could also be calculated by considering only the time it takes a participant to finish sending after it releases the token and subtracting the time required for the token to reach the next participant after it is released. Using this alternative formulation, X is $\max\left(\frac{W_{accelerated}}{W_{personal}}B - P, 0\right)$, where $W_{accelerated}$ is the accelerated window, and $W_{personal}$ is the personal window. $\frac{W_{accelerated}}{W_{personal}}$ is the fraction of the personal window that is sent after a participant releases the token, so $\frac{W_{accelerated}}{W_{personal}}B$ gives the time during which a participant continues to send after releasing the token. Since it takes one propagation delay for a participant's successor to receive the token after it is released, the time that the participant continues to broadcast after its successor receives the token is $\frac{W_{accelerated}}{W_{personal}}B - P$. As before, it is possible that each participant finishes sending before its successor receives the token (i.e. when the time it sends after releasing the token is less than the propagation delay). In this case, there is no overlap, giving the formulation $\max\left(\frac{W_{accelerated}}{W_{personal}}B - P, 0\right)$.

In the best case, message n is the first message broadcast by its initiator in the current round, so no additional time is needed for the initiator of message n to complete its previous broadcasts. Thus, in the best case, the total time needed for all previous broadcasts to be sent after q receives the token is just the time needed for q's predecessor to finish broadcasting after q receives the token, which is exactly X (which is 0 if the broadcasting periods do not overlap, and is the overlap given by $B - \frac{R}{N}$ otherwise).

To analyze the average case, two cases must be considered. Either message n is sent before q's predecessor's messages have all been sent, or n is sent after all preceding messages have been sent. In the case that n is sent before q's predecessor has finished broadcasting, the additional time required to ensure that n and all preceding messages have been after q receives the token is exactly the time that it takes for all of q's predecessor's messages to be sent after q receives the token. As in the best case analysis, this time is the amount of overlap, X.

In the case that n is sent after all preceding messages have been sent, the time required for n and all preceding messages to be sent after q receives the token is the amount of time it takes for n to be sent, from the time q receives the token. At most, this time is B, in the case that n is the last message q sends in this round. Since all preceding messages have already been sent, this time is at least X, since that is the time required for q's predecessor to finish broadcasting. The average time for a message sent after all preceding messages have been sent is therefore $\frac{X+B}{2}$.

To complete the analysis, we calculate the proportion of messages that fall into each of the two cases. The fraction of messages that will be sent before all preceding messages have been sent is the fraction of the burst that can be sent in the time it takes for a participant's predecessor to finish sending, which is $\frac{X}{B}$. The proportion of

messages sent after all preceding messages have already been sent is just the remainder of the burst, or $1 - \frac{X}{B}$. The average total time needed to broadcast a message and all preceding messages is then the proportion of messages sent while previous participants are still sending, multiplied by the time to deliver such a message, plus the proportion of messages sent after all previous participants have finished sending, multiplied by the average time to deliver a message in this case. This gives $\left(\frac{X}{B} \times X\right) + \left(\left(1 - \frac{X}{B}\right) \times \frac{X+B}{2}\right)$ as the average time needed to send message n and all preceding messages after q receives the token.

Notice that the value of this expression depends on the value of X and is always between $\frac{1}{2}B$ and B. When X is zero, broadcasting periods do not overlap at all. In this case, each message is sent after all messages preceding it in the agreed order have been sent, so the average time for a message and all preceding messages to be sent after its initiator receives the token is the just the average time for that message to be sent after its initiator receives the token, which is half a broadcasting period. If it were possible for the token to circulate the ring instantaneously, so that $\frac{R}{N}$ would be 0 and X would be equal to B, the broadcasting periods of a message's initiator and all other participants would completely overlap and all messages would be sent in parallel. In that case, one broadcasting period would be required for all preceding messages to be sent, since that would be the time needed for the broadcasting periods of all other participants must be completely finished.

4.4.3 Receiving n and Preceding Messages

For message n to be delivered, all messages preceding it in the agreed order must be received and delivered, and message n must be received. After n and all preceding messages have been broadcast, the additional time needed to receive these messages is the propagation delay P. From the time that the last message with a sequence number less than or equal to n is sent, the time for that message to be received is exactly P. Since the propagation delay is assumed to be the same for all messages, all messages sent before this last message will be received before it, so one propagation delay after the last message with a sequence number less than or equal to n is sent, n and all preceding messages have been received and can be delivered in order.

4.4.4 Total Latency

The total agreed delivery latency in the best, worst, and average cases can be determined by adding the time required for the initiator to receive the token, the time required for message n and all preceding messages to be sent, and the time for n and all preceding message to be received in each case. These results are summarized in Table 4.1. Note that while the B used in the table corresponds to the same amount of time for both the original and accelerated protocols, the R in the table is different for the two protocols. R corresponds to one rotation of the token; because the

		Original	Accelerated $X = \max\left(B - \frac{R_A}{N}, 0\right) = \max\left(\frac{W_{accelerated}}{W_{personal}}B - P, 0\right)$
Best	Lightly-loaded	P	P
	Fully-loaded	P	$X + P$
Average	Lightly-loaded	$\frac{1}{2}R_O + P$	$\frac{1}{2}R_A + P$
	Fully-loaded	$\frac{1}{2}R_O + \frac{1}{2}B + P$	$\frac{1}{2}R_A + \left(\frac{X}{B} \times X\right) + \left(\left(1 - \frac{X}{B}\right) \times \frac{X+B}{2}\right) + P$
Worst	Lightly-loaded	$R_O + P$	$R_A + P$
	Fully-loaded	$R_O + B + P$	$R_A + B + P$

Table 4.1: Agreed Delivery Latency

Accelerated Ring protocol rotates the token faster, the R in the Accelerated Ring protocol corresponds to a shorter amount of time than the R in the original protocol. For this reason, R_O is used to denote the duration of a round in the original protocol, while R_A is used to denote the duration of a round in the Accelerated Ring protocol. As discussed in the throughput analysis in Section 4.2, for a fully-loaded system, $R_O = (B + P) \times N$, and $R_A = \max\left(\left(\left(1 - \frac{Accelerated\_window}{Personal\_window}\right) B + P\right) \times N, B\right)$. In practice, for a fully-loaded system, a single token rotation takes about 50% longer in the original protocol than in the Accelerated Ring protocol for the best choices of personal and accelerated windows. In a lightly loaded system, where messages are sent infrequently, B can be considered to be 0 (since participants will generally take little or no time to send), and both R_O and R_A approach $P \times N$, which is just the N propagation delays need to pass the token N times.

First, consider an idle or lightly-loaded system. In this case, we assume that all previous messages were already sent, received, and delivered at the time that the message is generated. Therefore, no additional time is required to send the preceding messages, and the latency depends only on the time required for the initiator to receive the token and the time for message n to be received for both the original and accelerated protocols. In the best case, the participant gets the token immediately after the message is generated and sends the message immediately upon receiving the token, so the message is received and can be delivered after one propagation delay, P. In the worst case, it takes one full token round for the initiator to receive the token that allows the message to be broadcast, plus one propagation delay for the message

to be received and delivered. On average it takes half a round for the initiator to receive the token, plus one propagation delay for the message to be received and delivered.

In a fully-loaded system, where each participant sends as many broadcast messages as it is allowed on each token rotation, we must additionally account for the time required for all preceding messages to be broadcast. As discussed above, for the original protocol this takes no time in the best case, half a broadcasting period on average, and a full broadcasting period in the worst case. Therefore, agreed delivery for the original protocol takes one propagation delay in the best case, requires half a token rotation plus half a broadcasting period plus one propagation delay on average, and requires a full token rotation plus a full broadcasting period plus one propagation delay in the worst case.

In the accelerated protocol, broadcasting message n and all preceding messages takes one broadcasting period in the worst case, since the message is not sent until the end of the initiator's broadcasting (and all preceding messages have already been sent at that time). In the best case, the additional time required for all preceding messages to be sent is just the time needed for the initiator's predecessor to finish broadcasting after the initiator receives the token, which is the overlap period X, as discussed above. On average, the additional time needed for this message and all preceding messages to be sent will be between B and $\frac{1}{2}B$, depending on the degree of overlap between the initiator's broadcasting period and its predecessor's broadcasting period. If they overlap completely (which can only happen if the token is able to complete a rotation instantaneously), one full broadcasting period (B) is required to ensure that all of the predecessor's messages are sent. If they do not overlap at all, only the messages sent by message n's initiator need to be considered, and since n will be sent halfway through its initiator's broadcasting period on average, the average time required for it to be sent is $\frac{1}{2}B$. In general, the average time required for n and all preceding messages to be sent is $\left(\frac{X}{B} \times X\right) + \left(\left(1 - \frac{X}{B}\right) \times \frac{X+B}{2}\right)$. In all cases, an additional period of one propagation delay is needed for message n and all preceding messages to be received. Therefore, agreed delivery for the accelerated protocol requires one broadcasting period plus one propagation delay in the best case, half a token rotation plus one half to one broadcasting period plus one propagation delay on average, and a full token rotation plus one more full token rotation plus one propagation delay in the worst case.

4.5 Safe Delivery Latency

Safe delivery latency is defined as the time required for a message initiated by a randomly selected participant q to be delivered in accordance with the requirements of safe delivery by a randomly selected participant p, measured from the time that the message is generated at the application level.

For a participant to deliver a safe message, the participant must have received that message, have received and delivered all messages that precede it in the agreed order, and know that all other participants in its current configuration have received that message and all preceding messages. As discussed in the description of the delivery and discarding procedures, a participant knows that all participants have received a message with sequence number n once it releases the token with a *seq* value of at least n on two consecutive rotations. Once that requirement is met, the message fulfills the requirements for safe delivery. Therefore, from the time that a message with sequence number n is generated, to the time that it is delivered by some participant p, the following events must occur:

1. The participant q that will initiate this message must receive the token.
2. All messages with sequence numbers lower than n must be broadcast.
3. Message n must be broadcast.
4. All participants must receive (and deliver) all messages with sequence numbers lower than n.
5. All participants must receive message n.
6. The token's *aru* field must be raised to a value of at least n.
7. Participant p must release the token twice with an *aru* of at least n.

As in the analysis of agreed delivery, the safe delivery latency is analyzed below in terms of the parameters R (duration of a token round), B (broadcasting period), and N (number of participants in the ring). The overlap factor X, defined as $\max\left(B - \frac{R}{N}, 0\right)$ is also used as in the agreed delivery analysis to analyze the expected performance of the Accelerated Ring protocol. As before, the analysis assumes that no messages are lost and that messages are not generated at a faster rate than they can be broadcast.

4.5.1 Receiving the Token

The time required for the message initiator to receive the token is described in the analysis of agreed delivery latency. For both the original and accelerated protocols, it is immediate in the best case, half a round on average, and one round in the worst case.

4.5.2 Broadcasting n and Preceding Messages

The time needed to ensure that all preceding messages are broadcast is also described in the analysis of agreed delivery latency. For a lightly-loaded system, in which all preceding messages are assumed to already be delivered at the time this message is generated, this step does not add any latency for either the original or accelerated protocol.

For a fully-loaded system, in the original protocol, only preceding messages initiated in the current round by the same participant q that will initiate this message n need to be considered, since all preceding messages initiated by other participants are required to have been sent before the initiator of this message received the token. In the best case, message n is the first message q will initiate in this round, so broadcasting preceding messages will not add any latency. In the worst case, it is the last message q will initiate in this round, so it will not be sent until the end of q's broadcasting period. On average, this message will be sent halfway through q's broadcasting period.

In the Accelerated Ring protocol, in the worst case, n is the last message sent during its initiator's broadcasting period, and all preceding messages have already been sent at the time n is sent, since all other participants started their broadcasting periods for the current round before q, and therefore will finish broadcasting before the end of q's broadcasting period, under the reasonable assumption that all participants are able to broadcasts at the same speed. In the best case, n is the first message that its initiator q will send in the current round, so n is sent as soon as q receives the token, and the additional time for all preceding messages to be sent is the time it takes q's predecessor to finish broadcasting for the current round. As explained in the discussion of agreed delivery latency, this time is X (the overlap between the broadcasting periods of q and q's predecessor, or the amount of time need for q's predecessor to finish broadcasting after q receives the token). The average time for n and all previous messages to be sent after q receives the token is calculated as $\left(\frac{X}{B} \times X\right) + \left(\left(1 - \frac{X}{B}\right) \times \frac{X+B}{2}\right)$. For all messages sent by q before q's predecessor has finished its broadcasting period, the time needed for all preceding messages to be sent is the time for q's predecessor to finish its broadcasting period, which is X. The fraction of q's messages that will be sent before its predecessor finishes broadcasting is $\frac{X}{B}$. For all messages sent by q after q's predecessor has finished its broadcasting period, the time needed for n and all preceding messages to be sent is the time it takes q to send n after it receives the token, which is $\frac{X+B}{2}$ on average. The fraction of q's messages that fall into this category is $\left(1 - \frac{X}{B}\right)$.

4.5.3 Receiving n and Preceding Messages

As explained in the discussion of agreed delivery latency, after n and all preceding messages have been broadcast, one propagation delay is needed for all participants

to receive n and all preceding messages.

4.5.4 Raising the *aru*

In the original Ring protocol, raising the token's *aru* does not add latency for safe delivery. Assuming no messages have been lost, the participant initiating message n should receive the token with the *aru* value equal to the *seq* value. When the token's *seq* and *aru* values are equal, the participant can increment the *aru* along with *seq* as it broadcasts new messages. Therefore, when the participant releases the token, the token's *aru* will be at least the sequence number of any message it broadcast in that round. Since message n is sent before the token, each other participant will receive the broadcast before receiving the token, assuming no message loss, and thus will not lower the token *aru*. Therefore, when q releases the token, its *seq* value will be at least n, and it will never be lowered.

However, in the Accelerated Ring protocol, the sequence number on the token can reflect messages that have not yet been sent, so the token *aru* will typically be lower than its *seq* value, even when no messages are lost.

First, consider a single broadcast message sent in a lightly loaded system, where there is time for the token's *aru* to catch up to its *seq* between each broadcast. In this case, the token's *seq* and *aru* values will be equal at the time that the message is sent. Assume that this is the n^{th} broadcast and is initiated by participant A in a ring composed of participants A, B, C, and D. In this case, we assume that all participants have already received and processed messages 1 through $n-1$ and participant A receives the token with both the *aru* and *seq* equal to $n-1$. Therefore, it increments both the *aru* and *seq* values to n before releasing the token.

If the *Accelerated_window* parameter is 0, no messages are sent before the token, and the analysis is the same as for the original protocol. However, assuming that the *Accelerated_window* parameter is set to at least 1, which is the normal case, the token is sent by A before broadcast message n. Participant B therefore will most likely receive and process the token before receiving broadcast n. Since B has not received message n, B's local *aru* must be less than n. Since we assume all participants have received messages and processed messages 1 through $n-1$, B's local *aru* is $n-1$. Therefore, in accordance with the rules described in Section 3.1.2.2, B will lower the token *aru* to $n-1$ before releasing it. Because of this, the token *aru* cannot be raised until the next time B releases the token. When B receives the next token, message n is guaranteed to be sent (otherwise A would not pass the token), so B's local *aru* will have risen to n and it will raise the token *aru* to n when it receives it. Assuming no message loss, all participants now have message n, so the token *aru* will not be lowered again. In this case, the time needed to raise the token's *aru* is $\frac{R}{N} + R$, since one N^{th} of a token rotation is needed for the token to get B, who then lowers the *aru*, and one additional rotation of the token is needed from that point for B to receive the token again and raise its *aru*.

However, in a more heavily loaded system, in which participants are continuously broadcasting, the token's *seq* and *aru* are unlikely to be equal every time a message is broadcast. In this case, the latency for a safe message to be delivered at every participant depends on the position of the token at the time that the message is received by all participants relative to the position of the last participant to lower the token *aru* in the ring. Only the participant that last lowered the token's *aru* can raise the token's *aru* to include this message. Therefore, the token *aru* cannot be raised to at least this message's sequence number until the last participant to lower the token *aru* receives this message and subsequently receives the token. In the worst case, the participant that can raise the token *aru* releases the token immediately before receiving the last message it needs to raise the token *aru* to at least n (i.e. the last message with a sequence number less than or equal to n that it had not yet received). In this case, one additional token rotation is needed for the token *aru* to be raised after the last message with a sequence number less than or equal to n is received. In the best case, the token arrives at the participant that can raise the token *aru* immediately after that participant receives the message that allows it to raise the *aru*, and raising the *aru* adds no additional latency. On average, the token is half a round from the participant that can raise the *aru*, so an additional half round of latency is incurred.

4.5.5 Seeing the Raised *aru*

The time needed for any given participant p to release the token twice with an *aru* of at least n, which allows it to deliver message n, depends on the relative positions in the ring of p and the participant that raised the *aru* to at least n. In the best case, p is the participant that raised the *aru*, and only one token round is needed after p raises the token's *aru* for p to see the raised *aru* a second time. In the worst case, p is the immediate predecessor of the participant that raised the *aru*. In this case, it takes one token round, minus the final token pass back to the participant that raised the *aru*, for p to see the raised token *aru* the first time, and then one more round for p to receive the token again, adding a total latency of $2R - \frac{R}{N}$. On average, p is expected to be half a token round from the participant that raised the *aru*, so it takes half a token round for p to see the raised *aru* the first time and then one more round for p to receive the token again, adding $1\frac{1}{2}R$ in latency.

Notice that the latency added in the best case is the time required for the message to be delivered by the first participant that delivers it, while the latency added in the worst case is the time required for the message to be delivered by every participant in the ring. The added latency in the average case is the time that delivering the message is expected to take for any randomly selected participant.

		Original	Accelerated $X = \max\left(B - \frac{R_A}{N}, 0\right) = \max\left(\frac{W_{accelerated}}{W_{personal}}B - P, 0\right)$
Best	Lightly-loaded	R_O	$2R_A + \frac{R_A}{N}$
	Fully-loaded	R_O	$R_A + X + P$
Average	Lightly-loaded	$2R_O$	$3R_A + \frac{R_A}{N}$
	Fully-loaded	$2R_O$	$2\frac{1}{2}R_A + \left(\frac{X}{B} \times X\right) + \left(\left(1 - \frac{X}{B}\right) \times \frac{X+B}{2}\right) + P$
Worst	Lightly-loaded	$3R_O - \frac{R_O}{N}$	$4R_A$
	Fully-loaded	$3R_O - \frac{R_O}{N}$	$4R_A + B + P - \frac{R_A}{N}$

Table 4.2: Safe Delivery Latency

4.5.6 Total Latency

The total latency to deliver a safe message with sequence number n from the time that the message is generated at the application level to the time it is delivered at a randomly selected participant p is determined by the time needed to complete each of the steps described above. The best, average, and worst case latencies for both the original Ring protocol and the Accelerated Ring protocol are summarized in Table 4.2.

4.5.6.1 Original Ring Protocol

For the original protocol, only two of the steps described above need to be considered: the time for the initiator to receive the token, and the time for a participant delivering the message to see the raised *aru* twice. To determine the total latency, we can just add the time needed for the initiator to receive the token to the additional time needed for a participant delivering the message to see the raised *aru* twice after the initiator has received the token. Because the initiator sends all its messages before sending the token, and we assume that no messages are lost, the initiator of message n will always raise the token *aru* to at least n, and it will not be lowered after that. Therefore, it is simpler to analyze the time needed to see the raised *aru* twice from the time that the initiator receives the token, rather than from the time that the *aru* is raised; then the broadcasting and receiving of n (and preceding messages) and the raising of the *aru* occurs in parallel with the rotation of the token needed for participants to see the raised *aru*.

This also means that the load on the system does not change the analysis; the load can increase the time needed for n to be sent after its initiator receives the token, but this is included in the token round needed for participants to see the raised aru the first time. Notice that while the analysis is the same for the lightly-loaded and fully-loaded cases, the measured latency will be higher in the fully loaded case, since each token round takes longer when each participant sends more messages during the round.

In the best case, the initiator of message n, participant q receives the token just after q is generated, so receiving the token does not add any latency. Let the randomly selected participant p be the initiator of message n (i.e. $p = q$). Then p releases the token with an aru of at least n for the first time just after it broadcasts n, and only one additional token rotation is needed for p to see the token with a sufficiently high aru for the second time. Therefore, the total latency to deliver the message in the best case is one token rotation.

On average, it will take half a token round for q to receive the token and be able to broadcast message n. A randomly selected participant is expected to be half a round away from the message's initiator, on average, so the average time for p to see a token with an aru of at least n on two consecutive rotations is $1\frac{1}{2}R$: half a round for the token to get to p the first time after being released by q with an aru of at least n, plus one more rotation to see it the second time. This gives a total latency of 2 token rounds for the average case.

In the worst case, it takes one full round for q to receive the token, and p is q's immediate predecessor on the ring, so it takes one round minus the final pass of the token back to q for p to see the raised token aru the first time and one more round for p to receive the token again and deliver the message, giving a total latency of $3R - \frac{R}{N}$.

4.5.6.2 Accelerated Ring Protocol

For the Accelerated Ring analysis, the steps of receiving the token, broadcasting message n and all preceding messages, receiving message n and all preceding messages, raising the token aru, and seeing the raised aru twice are all considered separately. This is necessary because the initiator of message n is generally not able to raise the aru to a value of at least n, so it is necessary to consider the time needed for the aru to be raised and then consider the additional time for p to see the aru after it has been raised to at least n. In addition, the case of a fully-loaded system where each participant sends the maximum number of messages it is allowed each time it receives the token must be considered separately from the case of a lightly loaded system, where broadcasts only occur occasionally and the token's aru typically has time to catch up to its seq value between broadcasts, since this affects when the token aru can be raised.

First, consider the best case analysis for a lightly-loaded system. In the best case, the message's initiator receives the token immediately after the message is generated.

The initiator broadcasts message n immediately upon receiving the token, and since the system is lightly-loaded, all preceding messages are assumed to have already been sent. Since q is assumed to receive the token with equal *seq* and *aru* values, due to the assumption of a lightly loaded system, q will raise the token *aru* to at least n. However, because q will send the token before sending message n, q's successor on the ring will lower the *aru*, so the *aru* will not be permanently raised to at least n until one round later, when q's successor receives the token again and can raise the *aru*. Therefore, from the time q receives the token that allows it to broadcast n, the time needed for the token *aru* to be raised is $R + \frac{R}{N}$, since it takes one N^{th} of a rotation for q successor to get the token the first time and lower the *aru* and then one round for it to receive the token the second time and raise the *aru*. After one more rotation of the token, q's successor on the ring will have seen the token twice with a sufficiently high *aru* to deliver the message. Therefore, in the best case, when the randomly selected participant p is q's successor on the ring, two token rotations are needed to deliver the message, from the time it is generated.

Next, consider the best case for a fully-loaded system. As before, q receives the token immediately after message n is generated and broadcasts it immediately. However, since all participants are sending the maximum number of messages, all messages preceding n are not sent until q's predecessor finishes broadcasting, which occurs a period of X after q receives the token. In addition, since q will not receive the token with equal *seq* and *aru* values, it will not raise the token *aru*. The token *aru* will be raised the first time that the participant that is able to raise the *aru* (i.e. the one that last lowered it) receives the token after receiving this message. From the time that it is sent, the last message of n and all preceding messages will be received after one propagation delay. In the best case, this participant receives the token immediately after receiving the last message that it needs to raise the *aru* to at least n, so no additional latency is added to raise the token *aru* after the last message needed is received. After one additional round, the participant that raised the *aru* will receive the token again with a sufficiently high *aru* and be able to deliver the message. Therefore, in the best case, for a fully-loaded system, the message can be delivered a period of $X + P + R$ after it is generated.

In the average case, in a lightly-loaded system, it takes half a round for q to receive the token and no additional time for message n and preceding messages to be sent (because of the assumption that the system is lightly-loaded so all preceding messages have been sent before message n is generated). As in the lightly-loaded best case analysis, q will raise the *aru*, but the next participant in the ring will lower it, so one additional round is needed for that participant to raise the *aru* to at least n, giving a total time of $R + \frac{R}{N}$ for the *aru* to be permanently raised after q receives the token that allows it to broadcast message n. After one more round, the participant that raised the *aru* will be able to deliver the message, but on average half a round more than that is needed for any randomly selected participant to deliver the message. Thus, in the average case, in a lightly loaded system, message n can be delivered by

the selected participant p three rounds after it is generated.

In a fully-loaded system, it still takes half a round for q to receive the token, on average. However, in this case, we cannot assume that all previous messages have been sent. As discussed in the analysis of agreed delivery latency, on average the time for n and all preceding messages to be sent is between $\frac{1}{2}B$ and B, depending on the amount of overlap in the broadcasting periods of different participants, and can be calculated as $\left(\frac{X}{B} \times X\right) + \left(\left(1 - \frac{X}{B}\right) \times \frac{X+B}{2}\right)$. For all these messages to be received, one propagation delay is needed after message n and all preceding messages are sent. On average, the participant that is able to raise the token's aru is expected to be half a round away from wherever the token is after n and all preceding messages have been received, so an additional half a round is needed for the token aru to be raised. As in the lightly-loaded system, the participant that raised the aru will be able to deliver n after one more token round, but one more half a round is needed, on average, for any given participant to see the raised aru a second time and deliver the message. Therefore, the average time to deliver a message in a fully-loaded system is between $2\frac{1}{2}R + \frac{1}{2}B + P$ and $2\frac{1}{2}R + B + P$ and can be calculated more precisely as $2\frac{1}{2}R_A + \left(\frac{X}{B} \times X\right) + \left(\left(1 - \frac{X}{B}\right) \times \frac{X+B}{2}\right) + P$.

In the worst case, it takes a full token round for q to receive the token. In a lightly loaded system, no preceding messages need to be sent, and q sends message n immediately upon receiving the token, so broadcasting n and preceding messages does not add any additional latency. As in the best and average case analysis for the lightly loaded system, q raises the token aru, q's successor lowers it, and then q successor raises the token aru the next round, so a total additional latency of $R + \frac{R}{N}$ is needed for q's successor to raise the token aru to at least n after q receives the token that allows it to broadcast message n. One token round is needed for the participant that raised the aru to see the raised aru a second time and deliver the message, but in the worst case it takes one more round, minus the final token pass back to the participant that raised the aru for the selected participant p to deliver the message (in the case that $p = q$). Adding the latency for each step gives a total of four rounds to deliver a message in the worst case in a lightly-loaded system.

In a fully-loaded system, it still takes a full token round for q to receive the token in the worst case. However, in the fully-loaded system we cannot assume that all preceding messages have been sent at the time that q receives the token, and it takes up until the end of q's broadcasting period for n and all preceding messages to be sent in the worst case. One additional propagation delay is then required for these messages to be received. From the time that n and all preceding messages are received, it takes one token round to raise the aru in the worst case (if the participant that can raise the aru releases the token just before receiving the last message it needs to raise the aru to at least n). As before, one more round is needed for the participant that raised the aru to see it again, and one round, minus the final token pass more than that is needed for the last participant to see the raised aru a second time. This gives a total latency of four rounds plus one broadcasting period, plus one propagation

delay, minus one N^{th} of a token rotation in the worst case for a fully-loaded system.

Chapter 5

Accelerated Ring Model Evaluation

This chapter describes the implementation and evaluation of a model of the Accelerated Ring protocol. The model is implemented in about 1000 lines of C code and was used to develop the protocol and evaluate its impact on performance relative to the original protocol. The results of the evaluations show that for the same throughput, the Accelerated Ring protocol provides significantly lower latency than the original protocol, and for the same latency, the Accelerated Ring protocol is able to provide significantly higher throughput than the original Ring protocol. For example, for the personal window setting at which the Accelerated Ring protocol first achieves above 980 Mbps throughput on a 1-Gigabit network, the Accelerated Ring protocol improves throughput by 56%, while at the same time improving agreed delivery latency by 23% and improving safe delivery latency by 17%. The throughput at this point is 980.4 Mbps, while agreed delivery latency is 1252 microseconds, and safe delivery latency is 4891 microseconds. The maximum throughput reached by the Accelerated Ring Protocol in these tests is 1.022 Gbps on a 1-Gigabit network (throughput can be higher than 1 Gbps when all participants send, since each participant does not need to receive its own messages), and 4.595 Gbps on a 10-Gigabit network.

5.1 Implementation Considerations

Because the Accelerated Ring Model is intended to be a model of the protocol's behavior, rather than a complete system, it offers more limited failure handling and message generation capabilities than the complete implementation discussed in Chapter 6. However, this implementation accurately models the protocol's performance and its techniques for selecting messages to process and avoiding the need for a participant to receive and process its own multicasts are similar to those used in the complete Spread implementation.

5.1.1 Failure Handling

The Accelerated Ring Model is designed to provide a simple way to evaluate the normal-case performance of the Accelerated Ring protocol. Because of this, the model does not implement the protocol's membership algorithm, and therefore does not tolerate participant failures or network partitions. However, the ordering algorithm handles message loss through the retransmission requests on the token, and the implementation handles token loss by retransmitting the token if no message is received

within a timeout period.

5.1.2 Message Generation

The model implementation does not support independent clients that connect to the protocol participants and introduce messages into the system. Instead, messages are generated by the protocol participants. In order to obtain accurate average latency measurements, the model provides two options for message generation. As one option, messages can be generated when the participant receives the token, simulating the best-case scenario in which messages are received just before the token arrives and therefore can be sent without the delay incurred by waiting for the token. Alternatively, messages can be generated immediately after the participant releases the token. This simulates the worst-case scenario in which the participant releases the token just before a message is generated and received from the application level and must wait one full token round before being able to broadcast the message. Average latency measurements are obtained by running the model in both modes and averaging the results.

5.1.3 Message Transmission

Broadcast messages are sent to all participants via IP-multicast. Token messages are sent from a participant to its successor on the ring via unicast.

5.1.4 Selecting a Message to Handle

As discussed in the protocol description in Section 3.1.4, it is possible for a participant to have both token and broadcast messages available for processing at the same time. When this occurs, the protocol dictates that the participant should choose to process messages of whichever type currently has high priority. The conditions for deciding which message type is given high priority are discussed in Section 3.1.4. In order to allow the participant to select the message type that it should handle, token messages and broadcast messages are sent on separate ports, and each participant opens one socket for receiving broadcast messages and one socket for receiving token messages. In this way, the participant can select which message type to process by choosing the socket from which it reads messages.

Each participant maintains a variable indicating which message type currently has high priority, according to the protocol. Messages are available, the participant first checks the socket corresponding to the message type that currently has high priority, and it will only read from the other socket if there is no message of the high priority type available.

5.1.5 Processing Self-Multicast Messages

When a participant initiates a broadcast message by multicasting it to the set of participants, that message is never sent back to that participant on the network (under normal IP-multicast behavior). This means that the theoretical maximum throughput when all participants initiate broadcast messages at the same rate is $\frac{N}{N-1}$ times the network capacity, where N is the number of participants in the ring. This is because the receiving capacity of any participant only needs to be divided among the other $N-1$ participants, but it can also initiate and deliver its own messages at the same rate as those of any other participant.

While a participant's multicasts are never returned to it on the network, they may be internally looped back to it and received and processed just like any other message sent to the multicast group. However, in the Accelerated Ring protocol, it is not necessary for a participant to receive its own broadcasts. It processes each broadcast that it initiates and inserts it into its buffer of packets awaiting deliver at the time that it initiates the message. Therefore, to avoid the computational expense of reading and processing its own multicast messages, each participant disables this loopback behavior by setting the IP_MULTICAST_LOOP socket option to 0 on the socket that it uses for sending messages.

5.2 Performance Evaluation

All tests of the model implementation were run on 8 Dell PowerEdge R210 I servers, with Intel Xeon E3-1270v2 3.50 GHz processors, 16 GBytes of memory, and 1-Gigabit and 10-Gigabit Ethernet connections. Tests were run using a 1-Gigabit Cisco switch as well as a 10-Gigabit Arista switch. All tests of the Accelerated Ring model implementation were run with the protocol variant that uses the first method described in Section 3.1.4 for selecting a message to handle, which maximizes the token rotation speed. Note that for figures comparing different accelerated window settings, when two points appear for an accelerated window of 0, the first is for the original protocol, while the second is for the Accelerated Ring protocol with the accelerated window set to 0. Because of the priority-switching method used, the Accelerated Ring protocol with an accelerated window of zero is not equivalent to the original Ring protocol, as discussed in Section 3.1.4.

5.2.1 1-Gig Evaluation

Figures 5.1, 5.2, and 5.3 show how throughput and latency change with the accelerated window parameter for personal windows between 5 and 100. These tests were all run on 8 machines, with each machine sending the same number of messages. Each machine sent as fast as possible; on each token rotation, each machine sent a full personal window of messages (unless it was not allowed to send a full personal window

due to excessive retransmissions). For each personal window setting tested, a range of accelerated window settings was tested. Note that for smaller personal windows, fewer accelerated window settings could be tested, since the accelerated window may not be greater than the personal window.

As discussed in the analysis of the expected throughput in Section 4.2, as the accelerated window increases relative to the personal window, the total throughput achieved by the Accelerated Ring protocol is expected to increase. The accelerated window is the number of messages that a participant can send after releasing the token, so as the fraction of messages that may be sent after passing the token increases, the degree of acceleration, or the speed with which the token circulates the ring increases. As the token rotates faster, the time to complete a round decreases, but the total number of messages sent in a round stays the same, as long as the personal window does not change.

In general, Figure 5.1 confirms that for a given personal window, throughput increases as the accelerated window increases. However, for personal windows of 40 or more messages, Figure 5.1 shows a sharp drop in throughput after the maximum throughput for that personal window is reached. This drop in throughput occurs when many messages start to be lost, and the need to retransmit these messages degrades throughput. Moreover, when a token message is lost, a 5 millisecond timeout must trigger before the token is re-sent and progress can be made.

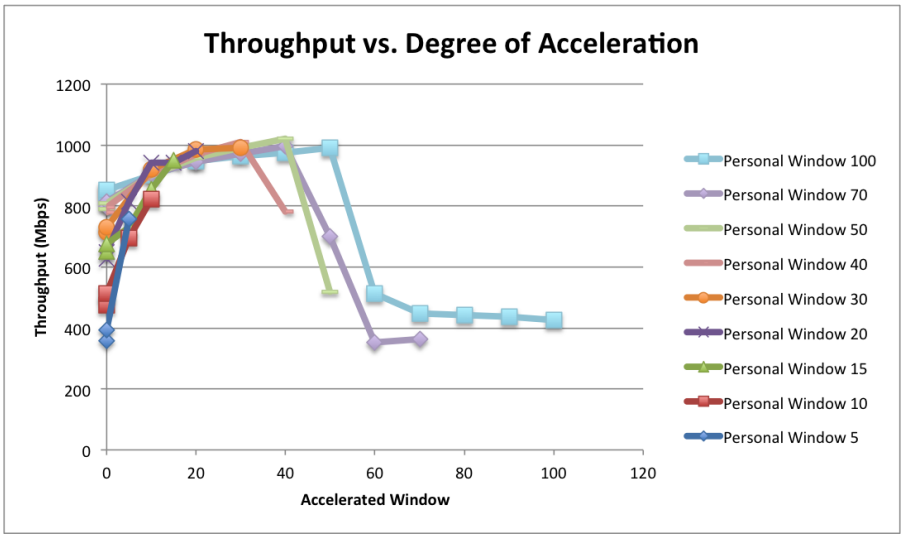

Figure 5.1: Throughput as accelerated window parameter increases, for personal windows between 5 and 100 on a 1-Gigabit network.

Figures 5.2 and 5.3 show the latency for agreed and safe message delivery, respectively, for the same personal and accelerated window settings shown in Figure 5.1. These figures show that both agreed and safe delivery latency increase as the personal

window increases and slightly decrease as the accelerated window increases, up until message loss causes a large spike in latency (these spikes are omitted from the figures for personal windows greater than 40 so that the latency differences for settings that do not result in severe loss would be visible). While Figure 5.1 shows that increasing the personal window allows higher throughput to be achieved with a lower degree of acceleration, Figures 5.2 and 5.3 show that large personal windows incur high latency. Using a smaller personal window setting (e.g. 20 to 40 messages) with a large degree of acceleration, in contrast, achieves high throughput while keeping latency low.

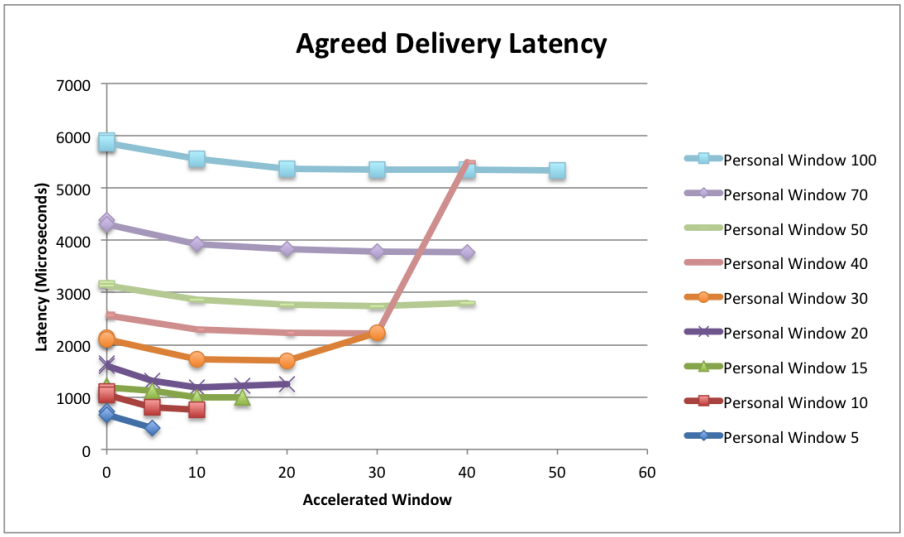

Figure 5.2: Agreed delivery latency as accelerated window parameter increases, for personal windows between 5 and 100 on a 1-Gigabit network.

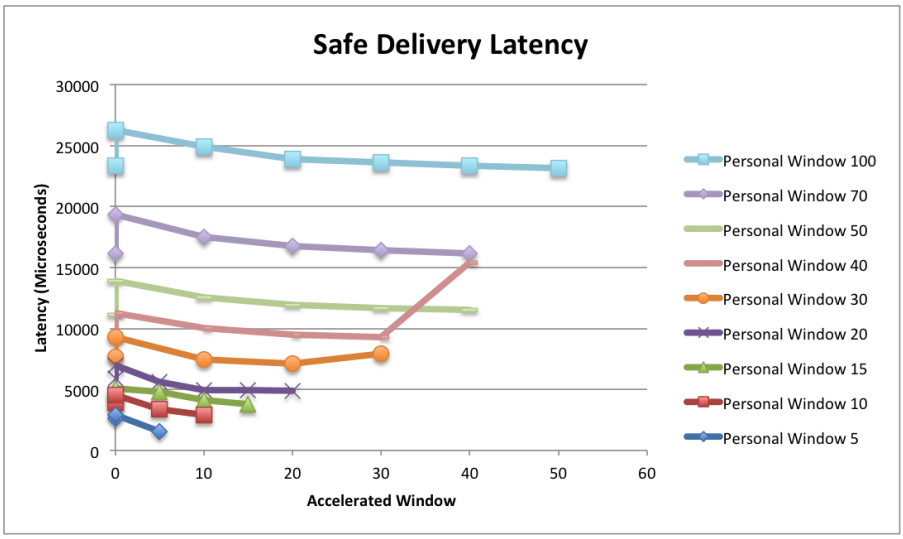

Figure 5.3: Safe delivery latency as accelerated window parameter increases, for personal windows between 5 and 100 on a 1-Gigabit network.

Figures 5.4, 5.5, and 5.6 are based on the same test results as Figures 5.1, 5.2, and 5.3. For each personal window setting used in those figures, the best accelerated window setting was selected and plotted against the same personal window setting for the original protocol. The best accelerated window setting was defined as the setting that resulted in the lowest safe delivery latency, which typically corresponded to high throughput and low agreed delivery latency as well.

These figures show that when appropriate accelerated window settings are selected for each personal window setting, the Accelerated Ring protocol is able to achieve significantly higher throughput that the original Ring protocol, while simultaneously providing lower or similar latency for both agreed and safe delivery. For example, for a personal window of 30, the best accelerated window setting was 20, throughput was 986 Gbps, the average agreed delivery latency was 1698 microseconds, and the average safe delivery latency was 7099 microseconds. In contrast, for the original protocol, a personal window of 30 resulted in throughput of 713 Mbps, agreed delivery latency of 2140 microseconds, and safe delivery latency of 7675 microseconds. The best throughput obtained by the original protocol in these tests was 840 Mbps with a personal window of 100, but this resulted in agreed delivery latency of 5904 microseconds and safe delivery latency of 23378 microseconds.

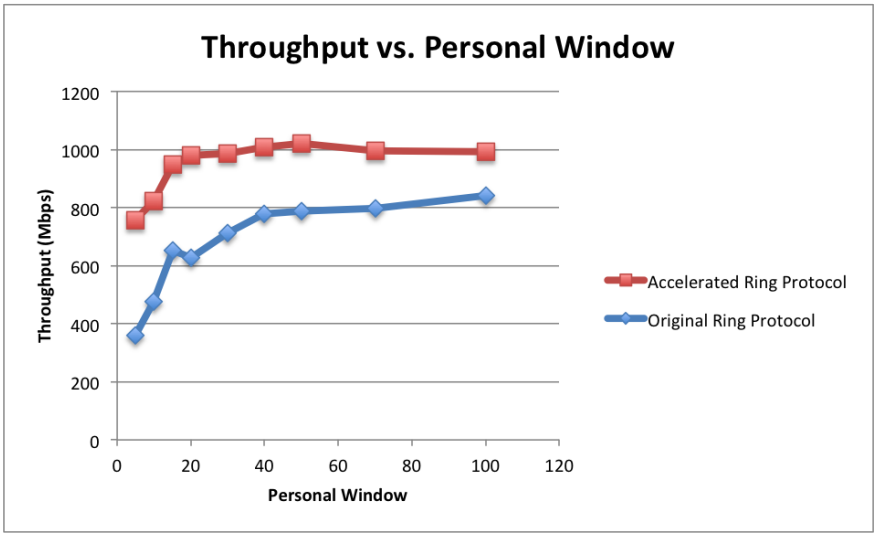

Figure 5.4: Throughput as personal window parameter increases on a 1-Gigabit network. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

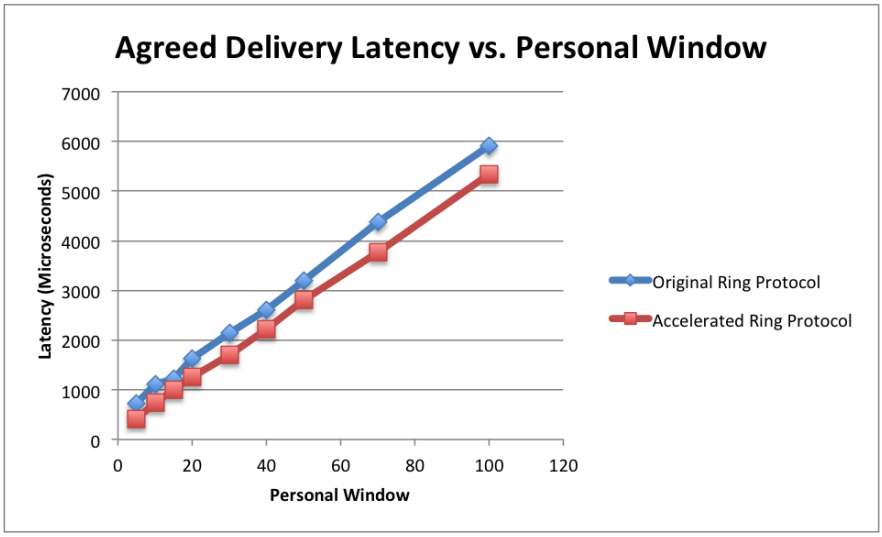

Figure 5.5: Agreed delivery latency as personal window parameter increases. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

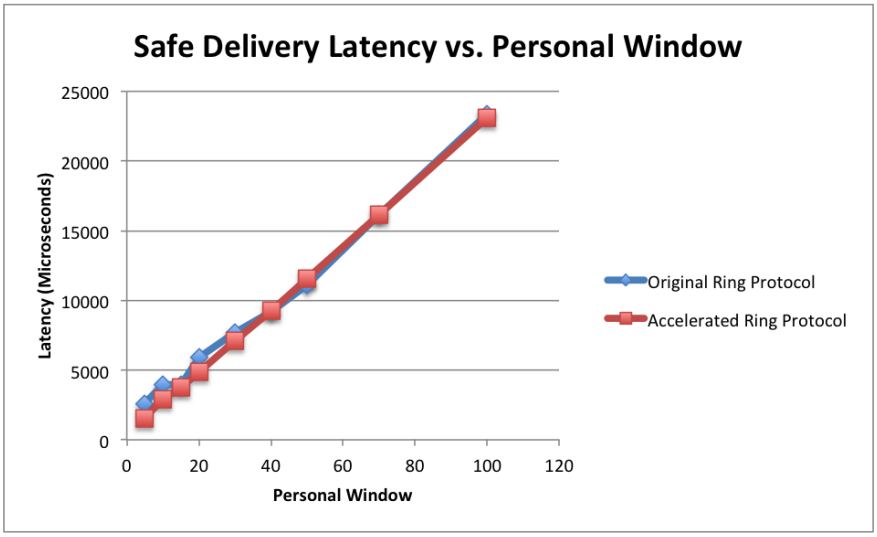

Figure 5.6: Safe delivery latency as personal window parameter increases. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

5.2.2 10-Gig Evaluation

The same tests described in the 1-Gig evaluation were also performed using a 10-Gigabit Arista switch, while keeping the setup otherwise the same. Like Figures 5.1, 5.2, and 5.3 in the 1-Gig evaluation, Figures 5.7, 5.8, and 5.9 show how throughput and latency change as the accelerated window increases for personal window settings between 5 and 100.

Figure 5.7 shows that the same pattern observed in the 1-Gig evaluation, where throughput increases as the accelerated window increases, also holds on faster networks. For smaller personal windows (up to about 40 messages), the effect is very similar, while for larger personal windows (e.g. 70 or 100 messages), throughput does not increase as rapidly as the accelerated window increases. In addition, there is no message loss to cause a drop in throughput like that seen in the 1-Gig evaluation. For the large personal window of 100, the protocol is able to achieve throughput of over 4.3 Gbps with no acceleration and accelerating the token raise the throughput to over 4.5 Gbps. Note that while the degree of acceleration has a less dramatic effect on throughput for large personal windows, the Accelerated Ring protocol with an accelerated window achieves significantly higher throughput than the original protocol, due to the priority switching method used. For a personal window of 100, the original protocol achieves just under 3 Gbps, while the Accelerated Ring protocol with an accelerated window of 0 achieves over 4.3 Gbps. This can be seen more clearly in Figure 5.10.

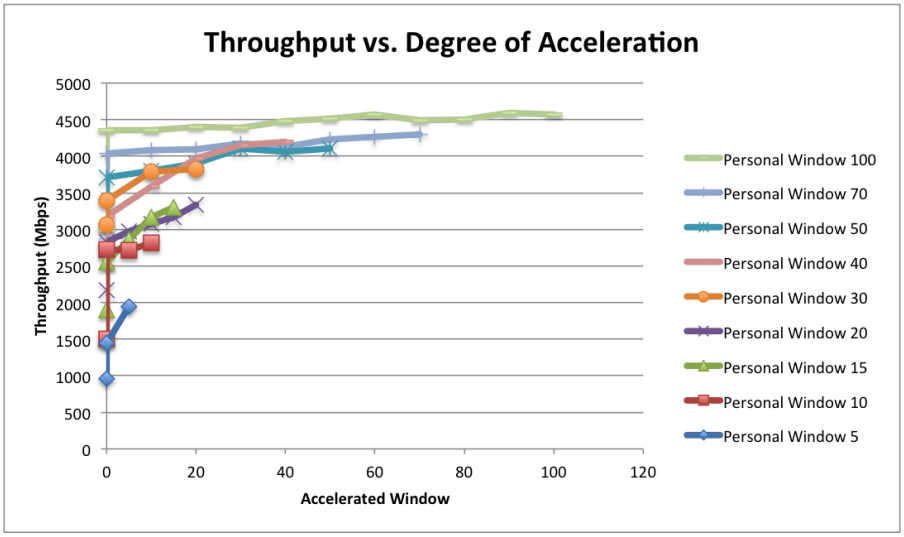

Figure 5.7: Throughput as accelerated window parameter increases, for personal windows between 5 and 100 on a 10-Gigabit network.

Increasing the degree of acceleration in the Accelerated Ring protocol significantly improves the trade-off between throughput and latency. Figures 5.8 and 5.9 show that both agreed and safe delivery latency increase as the personal window increases but generally decrease or stay close to constant as the accelerated window increases. While an accelerated window of 0 can provide over 4 Gbps of throughput for personal windows of 70 and 100, as shown in Figure 5.7, the Accelerated Ring protocol is also able provide throughput over 4 Gbps with a personal window of 40 (by using an accelerated window of 30 or 40), which incurs much lower latency for both agreed and safe delivery, as shown in Figures 5.8 and 5.9.

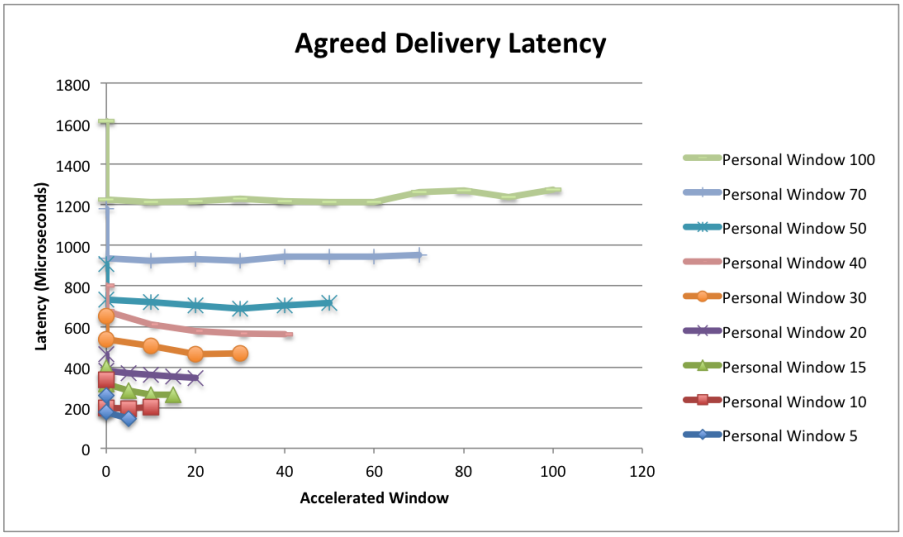

Figure 5.8: Agreed delivery latency as accelerated window parameter increases, for personal windows between 5 and 100 on a 10-Gigabit network.

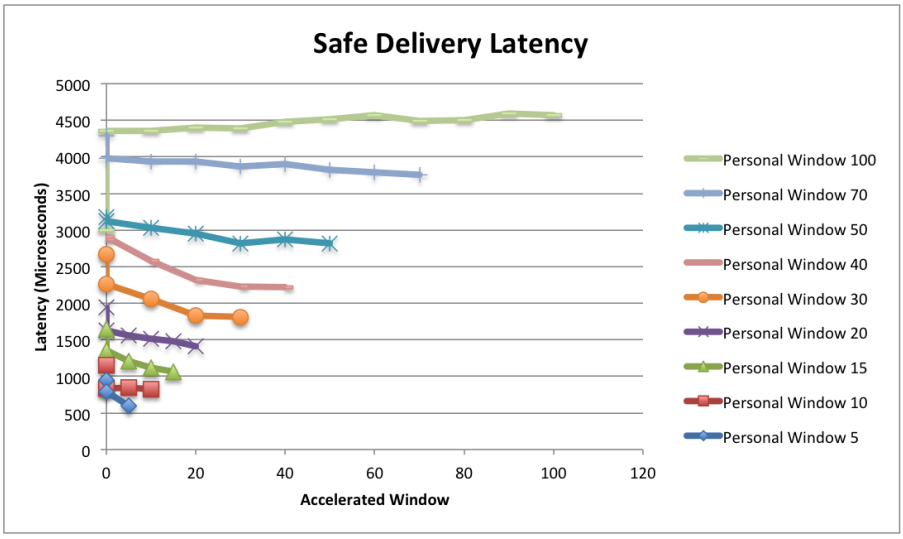

Figure 5.9: Safe delivery latency as accelerated window parameter increases, for personal windows between 5 and 100 on a 10-Gigabit network.

Figures 5.10, 5.11 and 5.11, show the difference between the accelerated and original Ring protocols and the improvement in the throughput-latency trade-off more clearly. For each personal window setting, the best accelerated window setting was selected, and the throughput, agreed delivery latency, and safe delivery latency for that setting was plotted against the throughput and latencies provided by the original protocol for the same personal window setting. As in the 1-Gig evaluation, these

figures show that the Accelerated Ring protocol is able to achieve higher throughput than the original Ring protocol for each personal window setting that was tested, while providing similar or lower latency at the same time.

For example, the Accelerated Ring protocol reaches throughput of over 4.1 Gbps with a personal window of 40, using an accelerated window of 40. This is 68% higher than the throughput achieved by the original protocol with a personal window of 40. At the same time, the Accelerated Ring protocol's agreed delivery latency of 562 microseconds is 30% lower than the agreed delivery latency of the original protocol, and the Accelerated Ring protocol's safe delivery latency of 2220 microseconds is 23% lower than the safe delivery latency of the original ring protocol at the same point. Even when the original protocol uses a personal window of 100, the Accelerated Ring protocol's throughput with a personal window of 40 is 40% higher, and its improvement in latency is even greater. In this case, the Accelerated Ring protocol's agreed delivery latency is 65% lower than that of the original protocol, and its safe delivery latency is 60% lower.

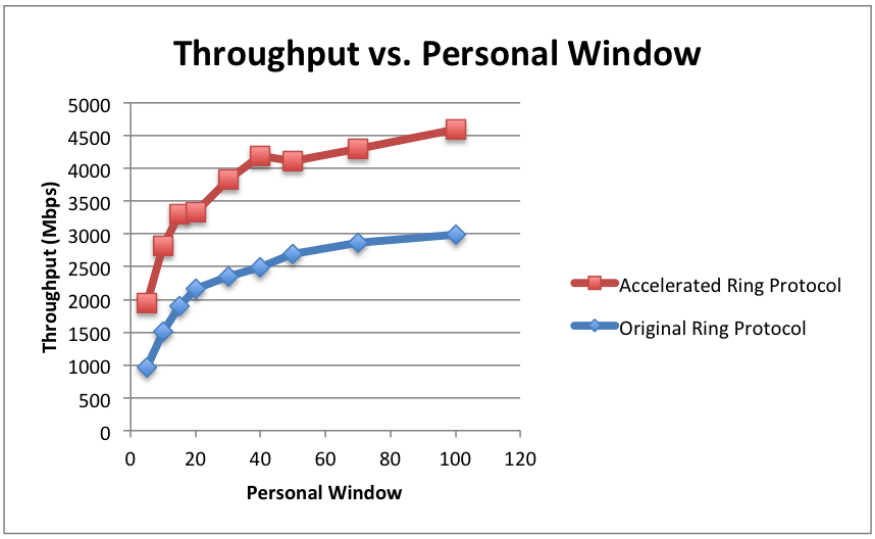

Figure 5.10: Throughput as personal window parameter increases. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

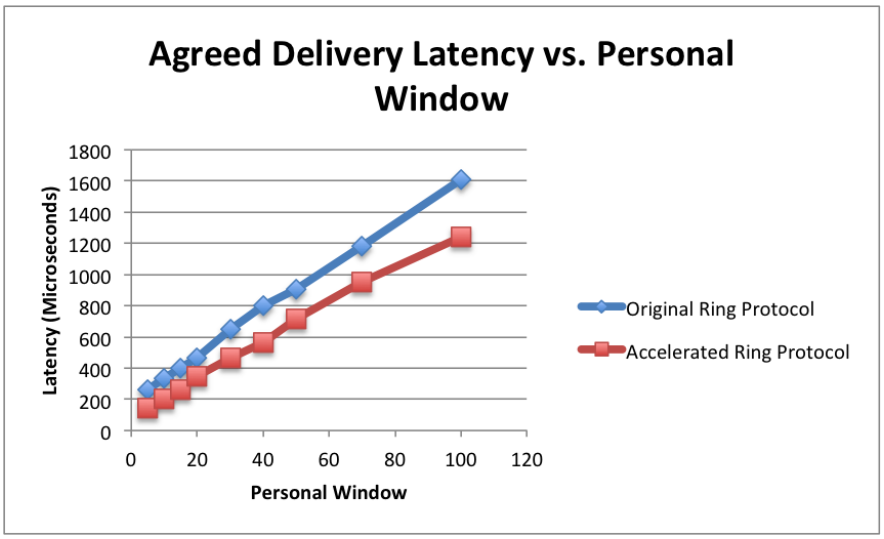

Figure 5.11: Agreed delivery latency as personal window parameter increases. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

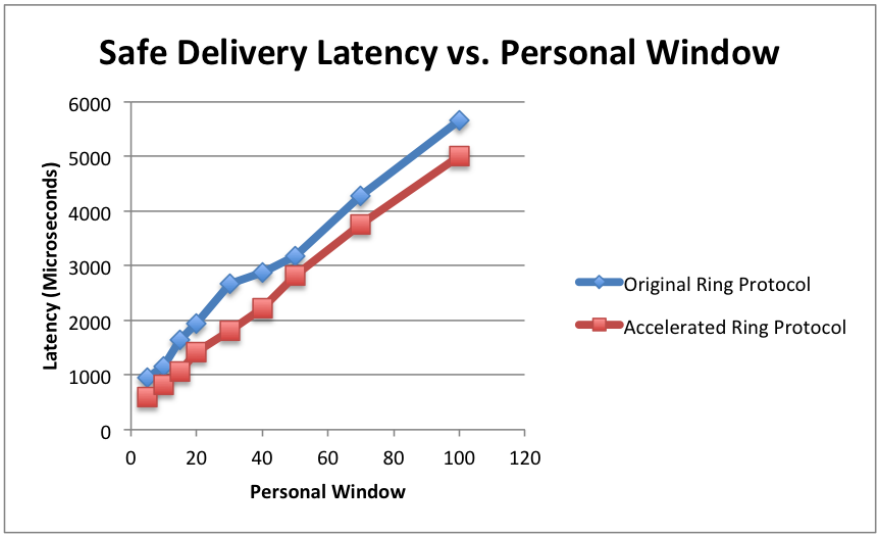

Figure 5.12: Safe delivery latency as personal window parameter increases. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

Chapter 6

Spread Toolkit Evaluation

The Accelerated Ring message-ordering protocol was implemented in the Spread toolkit[8] and is used as the standard protocol for datacenter environments and local area network settings. The original protocol, which is a variant of the Totem Single Ring protocol [1, 2], is still available in Spread and is the standard protocol for wide area environments. The Accelerated Ring protocol uses the same membership algorithm as Spread, so this implementation simply uses Spread's existing membership implementation to provide a complete group communication system that tolerates crashes, recoveries, network partitions, and network merges. This implementation is publicly available at www.spread.org. A version of Spread using the Accelerated Ring protocol was first released as an experimental version based on Spread 4.3, and the standard release of Spread uses the Accelerated Ring protocol starting with version 4.4.

6.1 Implementation Considerations

The Spread implementation of the Accelerated Ring protocol uses the same techniques as the model implementation to select messages to handle and to avoid the need for participants to receive and process their own multicast messages, and similarly transmits broadcast messages using IP-multicast and token messages by unicast. However, as a real system, the messages sent in the Spread implementation originate from separate clients. The need to efficiently support application messages of varying sizes introduces an additional implementation consideration.

6.1.1 Message Generation and Client Communication

Unlike the model implementation, this practical implementation uses a client-daemon architecture in which a set of daemons act as the protocol participants, and clients running some application can connect to these daemons to send messages generated by their particular application. Each client connects to one daemon and submits its messages to that daemon. That daemon will be responsible for initiating the broadcasts of that client's messages to the other participants and for delivering messages addressed to groups the client has joined to that client. Spread provides a multigroup multicast service, where each client can join one or more groups, and each message is addressed to one or more groups. Each client receives exactly the messages addressed to the groups they have joined, and all ordering guarantees apply across, as well as

within groups (i.e. there is one agreed order for all messages, regardless of the groups to which they are sent). A client may either run on the same machine as the daemon to which it connects and communicate with the daemon via IPC or run on a remote machine and communicate with the daemon via TCP.

6.1.2 Selecting a Message to Handle

The Spread implementation of the Accelerated Ring protocol selects a message to handle in the same way as the Accelerated Ring model implementation. Broadcast and token messages are sent on different ports and received on separate sockets, and a participant reads from the socket corresponding to the message type that has high priority and only reads from the other socket when no high priority messages are available for processing.

The concept of changing message priorities did not exist in Spread before the addition of the Accelerated Ring protocol. Previous versions of Spread would read broadcast messages as long as they were available and would only process the token when no broadcast messages were available. Because the Accelerated Ring protocol is designed to circulate the token around the ring faster, the concept of switching the priorities of broadcast and token messages at appropriate points in the protocol had to be added to allow participants to process the token as soon as it is safe for them to do so without causing unnecessary retransmissions.

6.1.3 Message Packing

In Chapter 3, the protocol is described as operating on broadcast messages that are assumed to be equivalent to the messages generated at the application level as well as to the messages that are sent on the network. However, in the Spread implementation, there is a distinction between application-level messages, protocol packets, and frames sent on the network. An application message can be between 1 and 100,000 bytes and is given a message header by Spread. A protocol packet is the unit on which the ordering protocol operates and includes a header with the relevant protocol information, such as the sequence number of the packet, the ID of the participant that initiated the packet, and the round in which the packet was initiated. A packet is limited to 1472 bytes (including headers), so a single application message may be split into multiple packets. In addition, to improve efficiency, multiple small messages may be packet together and sent on the network in a single frame. As of version 4.3 of Spread, this message-packing was done immediately before the frame was sent on the network, after application messages had been given packet headers and split into multiple packets, if necessary. Therefore, each packet contained one full message or a fragment of one large message, and each frame sent on the network contained one or more packets.

However, this message-packing scheme needed to be changed to accommodate the Accelerated Ring protocol. Because small messages were not packed together until after they were assigned headers at the protocol level and passed down to be prepared to be sent on the network, the protocol level did not have an accurate count of the number of frames that were actually being sent. The token-passing acceleration provided by the Accelerated Ring protocol depends on allowing a configurable number of messages to be sent before the token is sent, where messages are assumed to be equivalent to network frames. If a participant sends a certain number of packets before the token, but those packets are small and fit in a smaller number of frames, the degree of acceleration will be less than intended.

Therefore, in versions of Spread since the implementation of the Accelerated Ring protocol, message-packing is done at the protocol level. Each packet may contain a fragment of a large application message, or one or more smaller application messages. Each frame sent on the network consists of exactly one protocol packet. This makes counting the number of frames sent before the token simple, since they correspond directly to protocol packets. This packing scheme also reduces the overhead imposed by packet headers, since there is only one packet header per frame, rather than one packet header per message within that frame. In addition, this method improves retransmission handling. Previously, if a participant lost a frame containing multiple packets, it would need to request the retransmission of each packet separately, even though they were originally transmitted as a single unit. In the new scheme, each frame corresponds to exactly one packet, so only one retransmission request is needed to recover any lost frame.

6.2 Performance Evaluation

6.2.1 1-Gig Evaluation

As in the model evaluation, the impact of the accelerated window and personal window parameters in a fully-loaded system was evaluated. For Spread, one Spread daemon was run on each of the 8 machines, and each Spread daemon had two local clients connected to it. One client sent 1350 byte messages as fast as it could, while the other client received all the messages sent by all the sending clients. Figure 6.1 shows that a similar pattern is observed for Spread as for the model. Throughput increases as the accelerated window increases, up until the point that message loss degrades throughput (for personal windows of 40 or more, some of the highest accelerated window settings were not tested due to the severity of loss). The maximum throughput achieved is over 900 Mbps.

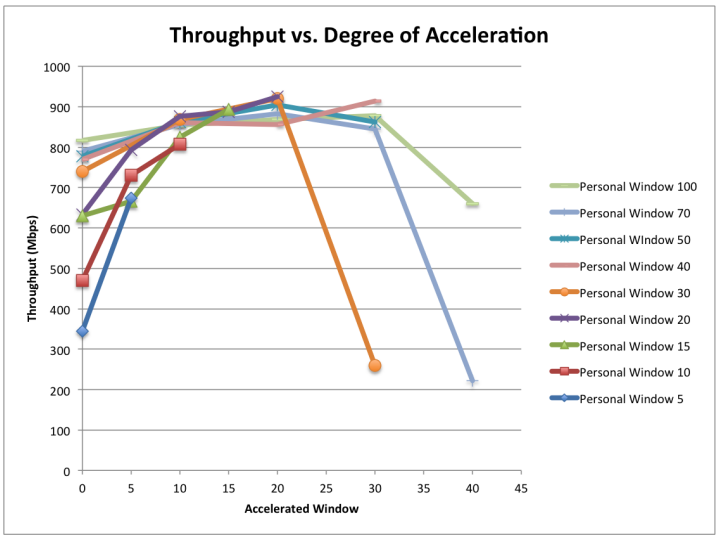

Figure 6.1: Throughput as accelerated window parameter increases, for personal windows between 5 and 100 on a 1-Gigabit network.

To compare the throughput obtained by the Accelerated Ring protocol to that achieved by the original Ring protocol, the same test was run using Spread version 4.3, which does not include the Accelerated Ring protocol. The throughput of Spread 4.3 for each personal window setting was then compared to the throughput achieved by the Spread implementation of the Accelerated Ring protocol for the same personal window setting for the best accelerated window setting for that personal window. Here, the best accelerated window setting was defined as the setting that provided the highest throughput. This is shown in Figure 6.2.

Figure 6.2 shows that for each personal window setting from 5 to 100, the Accelerated Ring protocol achieves higher throughput that than the original protocol. It also shows that the Accelerated Ring protocol is able to achieve its maximum throughput with a much lower personal window than the original protocol.

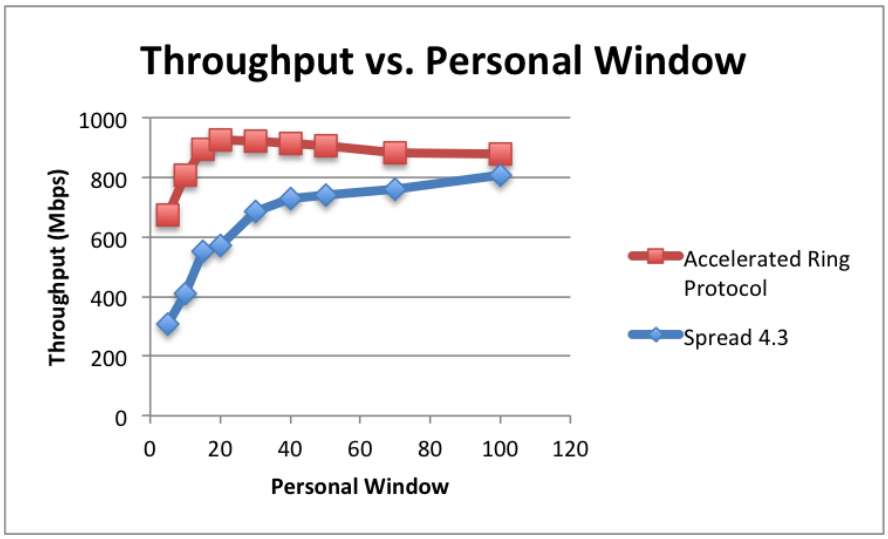

Figure 6.2: Throughput as personal window parameter increases on a 1-Gigabit network. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

To evaluate the change in the trade-off between throughput and latency for the Accelerated and original Ring protocols, we ran 8 Spread daemons, each with one local sending client and one local receiving client and varied the rate at which the sending clients injected messages into the system. For the Accelerated Ring protocol, a personal window of 20 and an accelerated window of 20 were used, since this is the smallest personal window for which the protocol was able to achieve its maximum throughput. For Spread 4.3, a personal window of 200 was used to allow the protocol to achieve sufficiently high throughput for the comparison. The results of these experiments are shown in Figures 6.3 and 6.4 for agreed and safe delivery latency, respectively. Note that using a larger personal window for Spread 4.3 does not adversely impact its latency for low sending rates. While higher personal windows increase message delivery latency in a fully-loaded system, when clients send at less than the maximum rate, the latency is the same for any personal window that can support that level of throughput.

Figure 6.3 shows that agreed delivery latency increases as throughput increases for both the Accelerated Ring Spread implementation and for Spread 4.3. However, the rate at which latency increases relative to throughput is lower for the Accelerated Ring protocol than for the original protocol, allowing the Accelerated Ring protocol to support the same or higher throughput than Spread 4.3 with lower latency. For an aggregate throughput of 700 Mbps, the highest throughput that both protocols comfortably support, the agreed delivery latency of the accelerated protocol is 80% lower than that of the original protocol.

At the point that the protocol is not able to support the throughput at which

the clients attempt to send, latency spikes for both the Accelerated and original Ring protocols (these spikes are omitted from the graph to better show the difference between the protocols). For the Accelerated Ring protocol, latency spikes when clients attempt to send at an aggregate rate of 1 Gbps, while for the original protocol, the spike occurs when clients attempt to send at an aggregate rate of 800 Mbps.

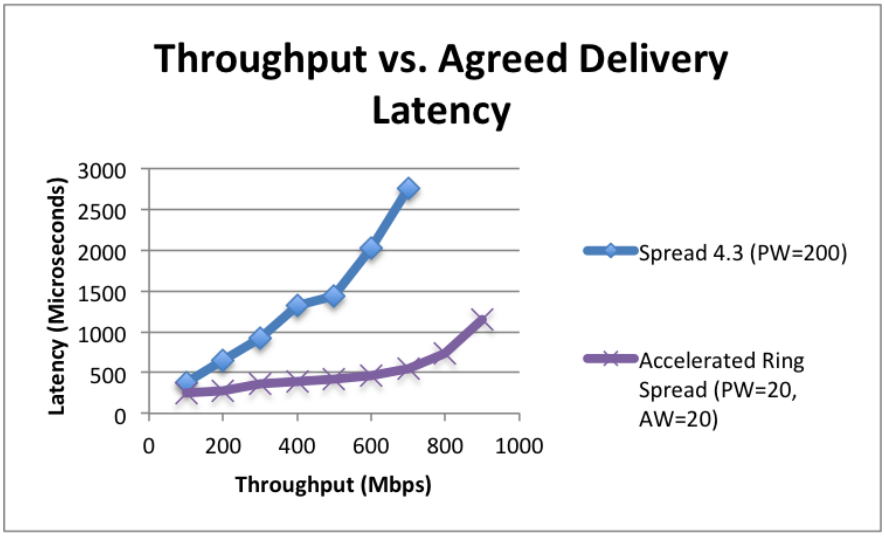

Figure 6.3: Agreed delivery latency as throughput increases on a 1-Gigabit network. For the Accelerated Ring protocol, the personal and accelerated windows are both set at 20. For Spread 4.3, the personal window is set at 200.

Figure 6.4 shows that safe delivery latency follows a similar pattern to agreed delivery latency as throughput increases. Latency increases as throughput increases and spikes at the point that the protocol cannot support the desired throughput. The latency of the Accelerated Ring protocol is lower than that of the original protocol overall and increases more slowly. At 600 Mbps, the highest latency both protocols comfortably support, the latency of the Accelerated Ring protocol is 63% lower than that of the original protocol.

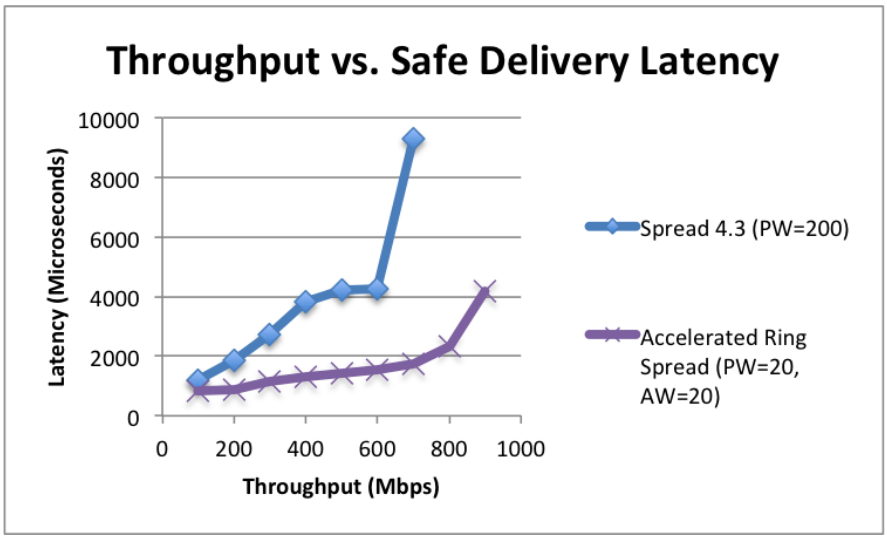

Figure 6.4: Safe delivery latency as throughput increases on a 1-Gigabit network. For the Accelerated Ring protocol, the personal and accelerated windows are both set at 20. For Spread 4.3, the personal window is set at 200.

All previous experiments in this chapter ran on 8 Spread daemons and used messages of 1350 bytes, which is the largest message that fits in a single packet in Spread. Figures 6.5, 6.6, 6.7, and 6.8 show how varying the size of messages and number of daemons affects performance. The experiments shown in these figures all used a personal window of 20 and used an accelerated window of 20 for the Accelerated Ring protocol.

Figures 6.5 and 6.5 show results from experiments in which each daemon has one local client. Every other daemon in the ring has a local sending client, while each of the remaining daemons has a local client that receives all the messages sent by the sending clients. Sending clients attempt to send as fast as possible.

Figure 6.5 shows that the number of daemons in the ring has little impact on the overall throughput. Higher throughput is obtained with larger messages due to the fact that message headers take up a higher proportion of the total bytes sent for small messages (and we only count clean application data toward the total throughput), and the processing overhead is higher for small messages. At the maximum throughput achieved in this test, with 100 KB messages, the Accelerated Ring protocol provides over 50% higher throughput than Spread 4.3 for all numbers of daemons tested.

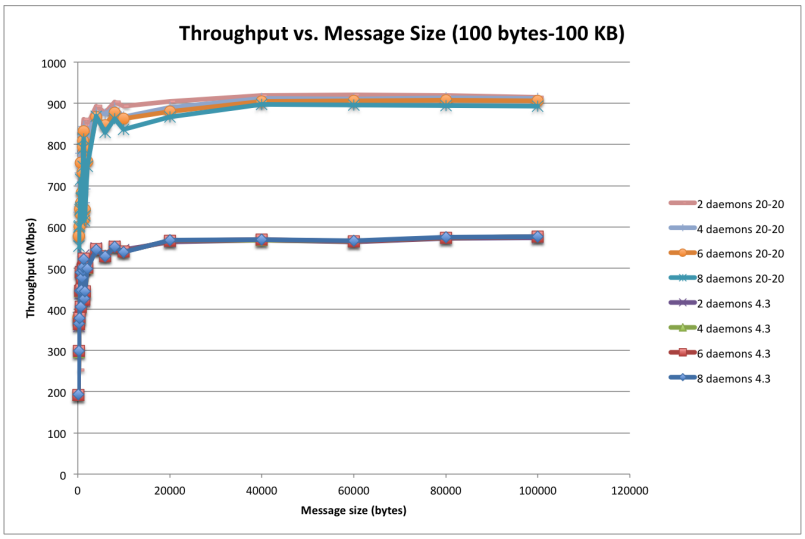

Figure 6.5: Throughput as message size increases, for a personal window of 20 on a 1-Gigabit network, with a sending client on every other daemon and a receiving client each daemon without a sending client. For the Accelerated Ring protocol, the accelerated window is also set to 20.

Figure 6.6 shows the throughput obtained when smaller messages (100-1500 bytes) are sent. This figure shows drops in throughput at 700 bytes and 1400 bytes for both protocols and all numbers of daemons. This is due to the message-packing scheme used in Spread. At 700 bytes, it is no longer possible to pack more than one message into a single packet. At 1400 bytes, each message can no longer fit in a single packet; it must be split across more than one packet.

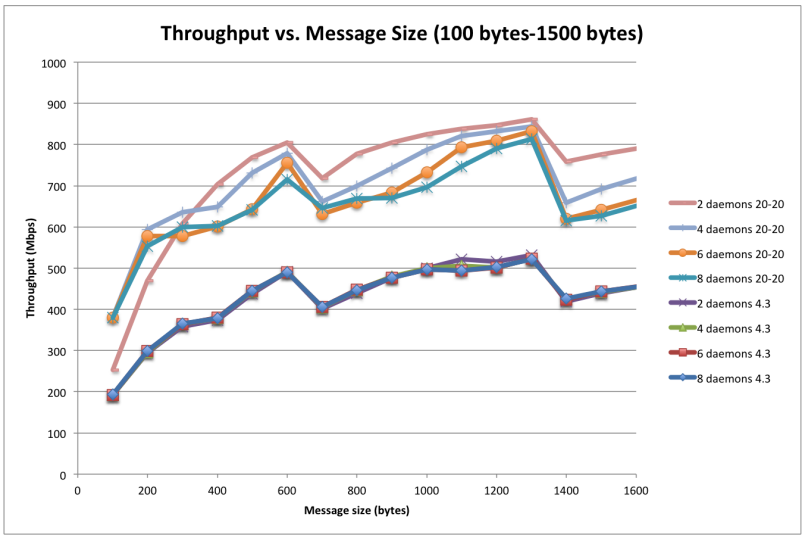

Figure 6.6: Throughput vs. message size for small messages, for a personal window of 20 on a 1-Gigabit network, with a sending client on every other daemon and a receiving client each daemon without a sending client. For the Accelerated Ring protocol, the accelerated window is also set to 20.

Figures 6.7 and 6.8 show experiments similar to those in Figures 6.5 and 6.6. The only difference in the experiments is that for those in Figures 6.7 and 6.8, each Spread daemon had two local clients connected; one sending and one receiving the messages of all senders.

In this case, the number of daemons affects performance for the Accelerated Ring protocol. While the experiments with 6 and 8 daemons result in slightly higher throughput than in the case where only every other daemon has a sending client, for the experiments with 2 and 4 daemons, throughput is significantly higher than in the tests with every other daemon having a sending client. In fact, throughput is above 1 Gbps for both 2 and 4 daemons. As discussed in Section 5.1.5, it is not necessary for a daemon's own multicasts to be returned to it on the network, allowing a daemon's incoming bandwidth to be divided among the other daemons. When the number of daemons is small, a larger fraction of the total messages are multicast by each daemon, so not receiving one's own messages has a larger impact. Notice that decreasing the number of daemons does not result in increased throughput in Spread 4.3. Although it is still the case that a daemon's own messages are not returned to it on the network, Spread 4.3 enforces that only one daemon can multicast at a time so the same benefit does not occur.

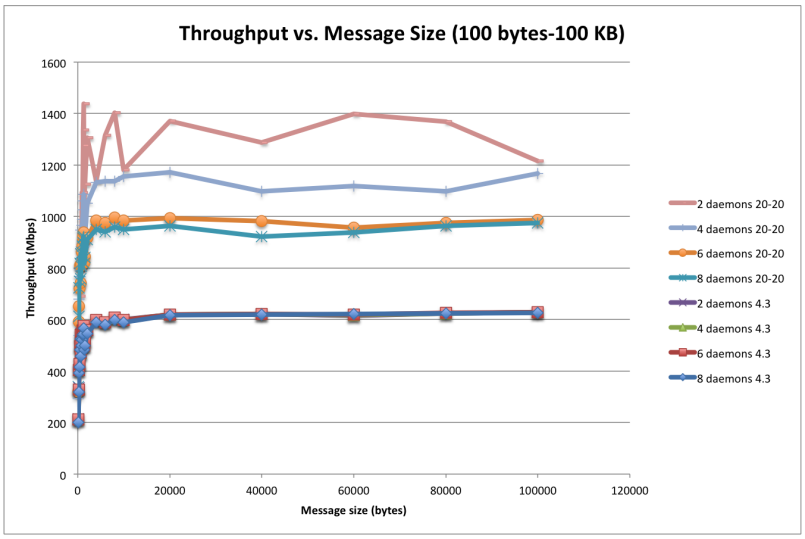

Figure 6.7: Throughput as message size increases, for a personal window of 20 on a 1-Gigabit network, with a sending client and a receiving client on each daemon. For the Accelerated Ring protocol, the accelerated window is also set to 20.

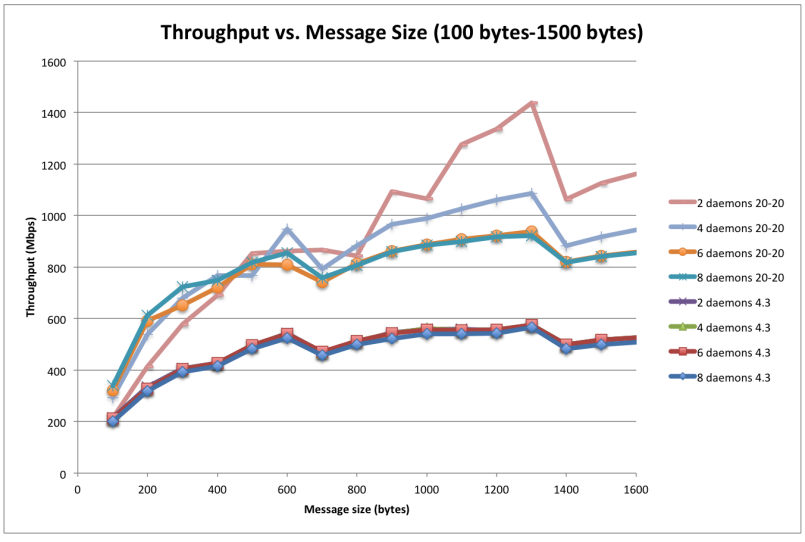

Figure 6.8: Throughput vs. message size for small messages, for a personal window of 20 on a 1-Gigabit network, with a sending client and a receiving client on each daemon. For the Accelerated Ring protocol, the accelerated window is also set to 20.

6.2.2 1-Gig Evaluation in Virtualized Settings

To assess the performance of the Accelerated Ring Spread implementation relative to Spread 4.3 in modern data centers, we additionally evaluated the protocol in virtual-

ized settings like those often used in public clouds. Figure 6.9 shows the performance of the Accelerated Ring Spread implementation and Spread 4.3 in several different environments: the non-virtualized CentOS setting used in all of the other experiments, in a Xen virtual machine, and in Xen's Dom0. In the tests shown in Figure 6.9, 8 daemons were used, and every other daemon in the ring had a local sending client, while the remaining daemons had a local receiving client.

Figure 6.9 shows that virtualization incurs a significant reduction in overall throughput, but the Accelerated Ring protocol shows a similar benefit over the original protocol in all environments. In all environments, the maximum throughput achieved by the Accelerated Ring protocol in these tests was 55-60% higher than the maximum throughput achieved by the original protocol. However, the maximum throughput achieved running in Xen's Dom0 is 31% lower than the maximum throughput achieved in native CentOS, and the maximum throughput achieved in a Xen virtual machine is 42% lower than that achieved in a native CentOS environment.

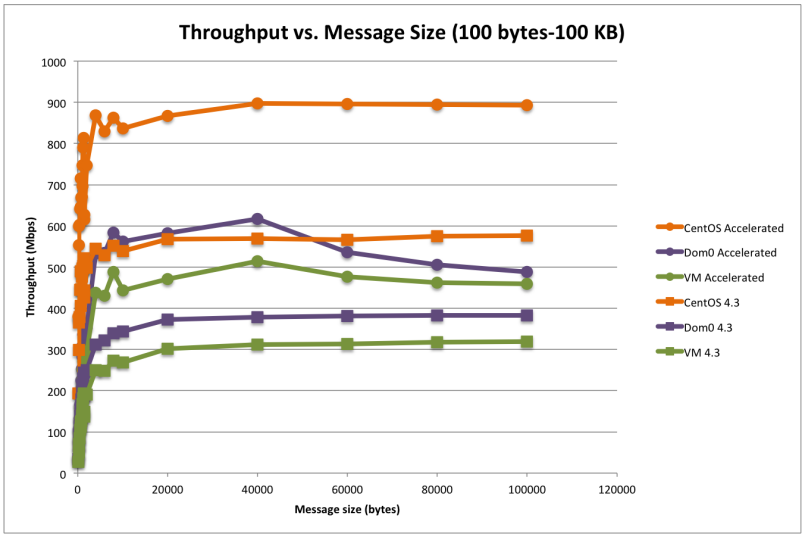

Figure 6.9: Throughput in virtualized and native environments on a 1-Gigabit network

6.2.3 10-Gig Evaluation

To evaluate the performance of the Accelerated Ring Spread implementation on faster networks like those often used in modern data centers, the same tests described in the 1-Gig evaluation were also performed using a 10-Gig Arista switch.

Figure 6.10 shows how the accelerated window parameter affects throughput for the Accelerated Ring protocol as implemented in Spread. For a range of personal and accelerated window settings, 8 Spread daemons were run, each with one client sending 1350 byte messages as fast as possible and one client receiving all messages.

Similarly to the 10-G evaluation of the Accelerated Ring model implementation, this evaluation shows that throughput increases with both the accelerated window and the personal window for smaller personal windows (less than 30).

Differently from the model evaluation, however, the evaluation in Figure 6.10 shows that throughput is almost completely constant with the accelerated window for large personal windows, and lower throughput is obtained with larger personal windows. Even for an accelerated window of 0, the throughput obtained when the personal window is 20 is higher than the throughput obtained when the personal window is 40 or more. This appears to be due to the looser flow control permitted by the priority-switching method used in the protocol variant evaluated in this section. A participant is allowed to process the token as soon as it receives any message from its predecessor in a round greater than the last round in which it received the token. When participants are not able to process messages at the rate at which they are received, which can occur on fast networks, large personal windows can result in message loss from too much overlapping sending, even when the accelerated window is 0.

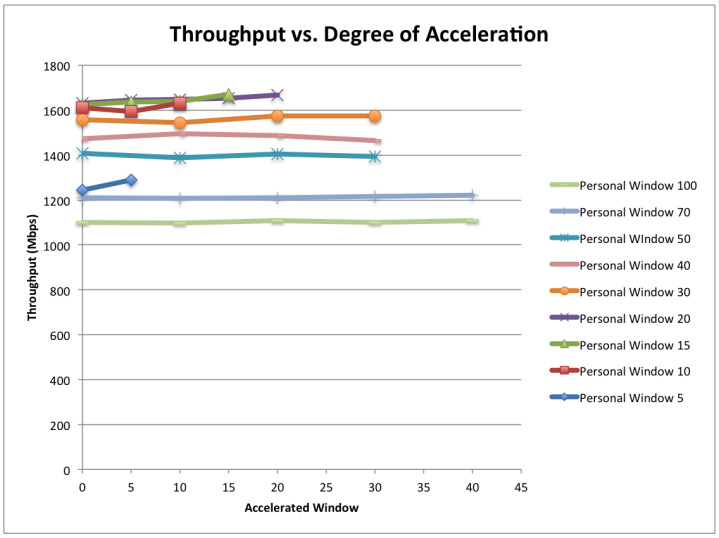

Figure 6.10: Throughput as accelerated window parameter increases, for personal windows between 5 and 100 on a 10-Gigabit network.

Figure 6.11 shows the results of the same test when the stricter flow control method is used (i.e. when a participant does not process the token until it receives the first broadcast message sent after its predecessor sent the token in the current round). These results follow a more intuitive pattern. For an accelerated window of 0, increasing the personal window generally increases throughput, and for personal windows of 20 or less, increasing the accelerated window increases throughput. However, for larger personal windows, increasing the accelerated window allows too much

overlap in sending and causes loss that degrades throughput.

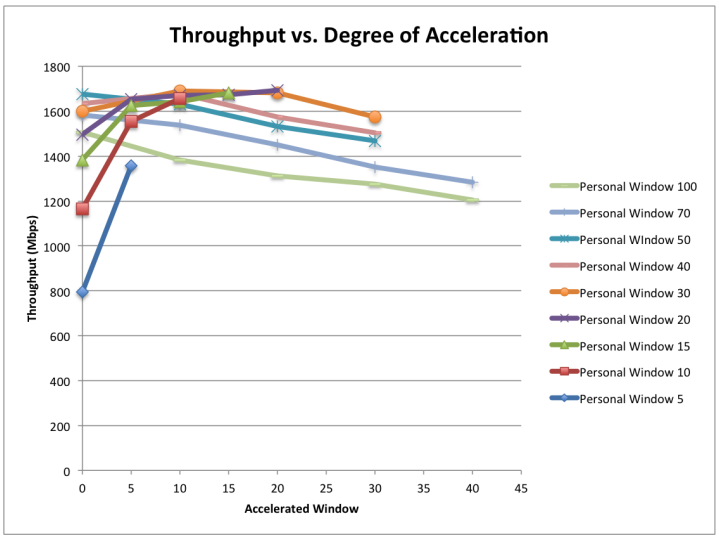

Figure 6.11: Throughput as accelerated window parameter increases, for personal windows between 5 and 100 on a 10-Gigabit network, using the less aggressive priority-switching variant for the Accelerated Ring protocol.

Figure 6.12 compares the throughput achieved for the best accelerated window setting for each personal window for the Accelerated Ring protocol to the throughput achieved for each personal window setting in the original protocol. This figure shows that both protocols achieve approximately the same maximum throughput. However, the Accelerated Ring protocol achieves its maximum throughput at a smaller personal window setting, and its throughput degrades for large personal window settings. This is because the priority-switching method used in this experiment allows token to be processed earlier and may cause loss for large personal windows. Note, however, that there is no reason to use a large personal window in the Accelerated Ring protocol; the maximum throughput can be achieved with a personal window of 20. Therefore, for appropriate flow-control parameter settings, the Accelerated Ring protocol does not reduce throughput compared to the original protocol.

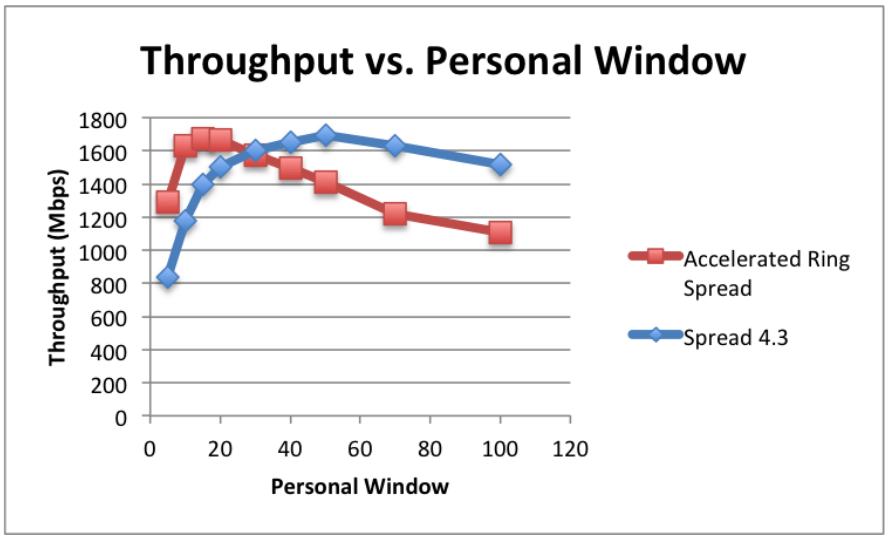

Figure 6.12: Throughput as personal window parameter increases on a 10-Gigabit network. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

Figure 6.13 shows that when the protocol variant that does not allow participants to process the token before processing all messages sent before the token is used, the throughput of the Accelerated Ring protocol never drops below that of the original protocol, since the two protocols behave in exactly the same way with respect to flow control when the accelerated window is set to zero. For personal windows of 50 or more, increasing the accelerated window beyond 0 leads to loss and reduces throughput, but at an accelerated window of 0, the throughput is the same as for the original protocol.

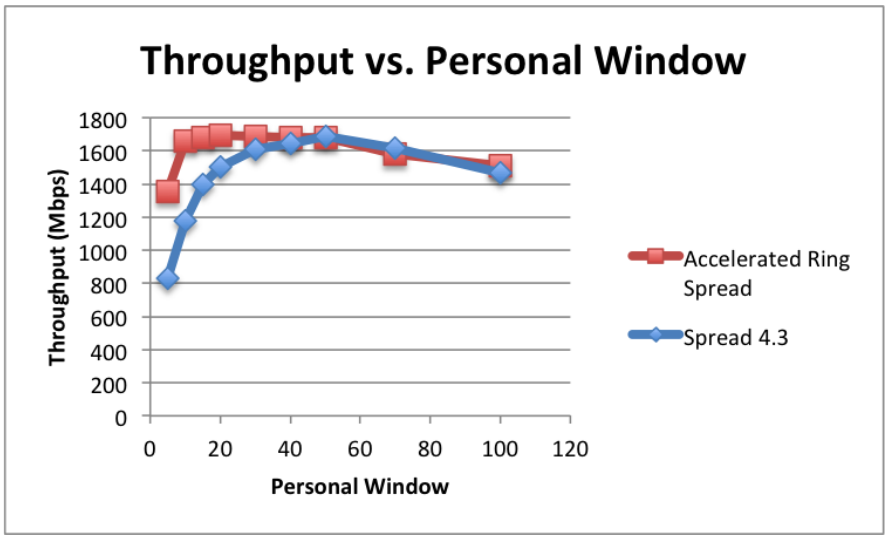

Figure 6.13: Throughput as personal window parameter increases on a 10-Gigabit network, using the less aggressive priority-switching variant for the Accelerated Ring protocol. For the Accelerated Ring protocol, the accelerated window is fixed at the best setting.

Figures 6.14 and 6.15 show the trade-off between throughput and latency for the Accelerated Ring Spread implementation and Spread 4.3 on a 10-Gigabit network. As in the 1-Gig evaluation, 8 Spread daemons are used, each with one sending and one receiving client. Sending clients inject messages into the system at a specified rate (the aggregate rate for all clients is shown on the x-axis of the figure), and receiving clients receive all messages.

Figure 6.14 shows that the Accelerated Ring protocol provides better latency for agreed message delivery than Spread 4.3 for each sending rate tested. When clients send at an aggregate rate of 1.2 Gbps, which is the highest rate both protocols supported, the Accelerated Ring protocol's agreed delivery latency is 49% lower than that of the original protocol.

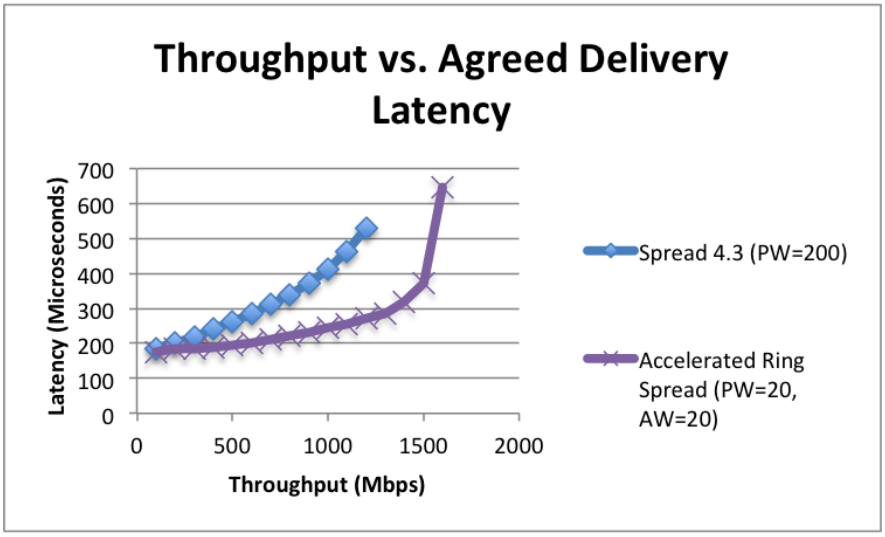

Figure 6.14: Agreed delivery latency as throughput increases on a 10-Gigabit network. For the Accelerated Ring protocol, the personal and accelerated windows are both set at 20. For Spread 4.3, the personal window is set at 200.

Figure 6.15 shows that the Accelerated Ring protocol provides better latency for safe message delivery than Spread 4.3 for aggregate sending rates of at least 400 Mbps. When clients send at an aggregate rate of 1.3 Gbps, which is the highest rate both protocols supported, the Accelerated Ring protocol's safe delivery latency is 27% lower than that of the original protocol.

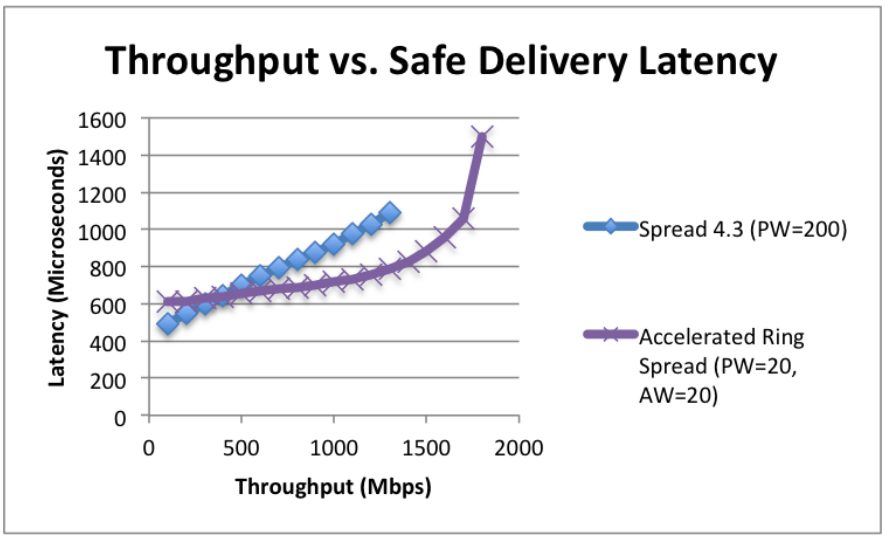

Figure 6.15: Safe delivery latency as throughput increases on a 10-Gigabit network. For the Accelerated Ring protocol, the personal and accelerated windows are both set at 20. For Spread 4.3, the personal window is set at 200.

Figures 6.16, 6.17, 6.18, and 6.19 show how throughput varies with message size and the number of Spread daemons in the ring. In the experiments shown in figures 6.16 and 6.17, every other daemon in the ring has a local client sending as fast as possible, while each remaining daemon has a local client receiving all messages sent by the sending clients. Figure 6.16 shows that throughput increases as message size increases, since smaller messages impose greater processing and header overhead. As shown in Figure 6.17, the difference between the protocols is reduced for small messages, as the overhead of processing these messages is high, and the CPU becomes a bottleneck. For large messages, the Accelerated Ring protocol improves throughput by 20-45% over Spread 4.3 in tests where every other daemon in the ring has a sending client and by 10-30% when every daemon in the ring has a sending client. The throughput achieved by the Accelerated Ring protocol with 100 KB messages and 8 daemons in the ring is 2.48 Gbps with 8 sending clients and 2.08 Gbps with 4 sending clients (one on every other daemon in the ring). For Spread 4.3, the throughput for 100 KB messages and 8 daemons is 1.93 Gbps with 8 sending clients and 1.48 Gbps with 4 sending clients.

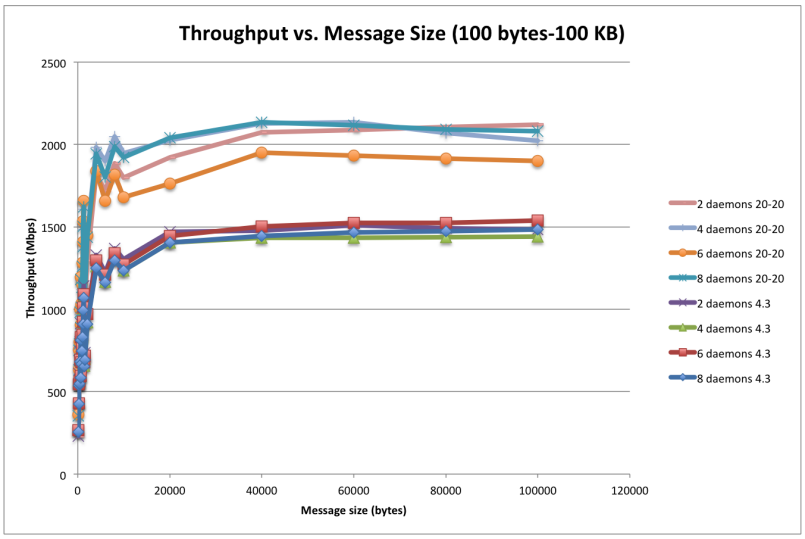

Figure 6.16: Throughput as message size increases, for a personal window of 20 on a 10-Gigabit network, with a sending client on every other daemon and a receiving client each daemon without a sending client. For the Accelerated Ring protocol, the accelerated window is also set to 20.

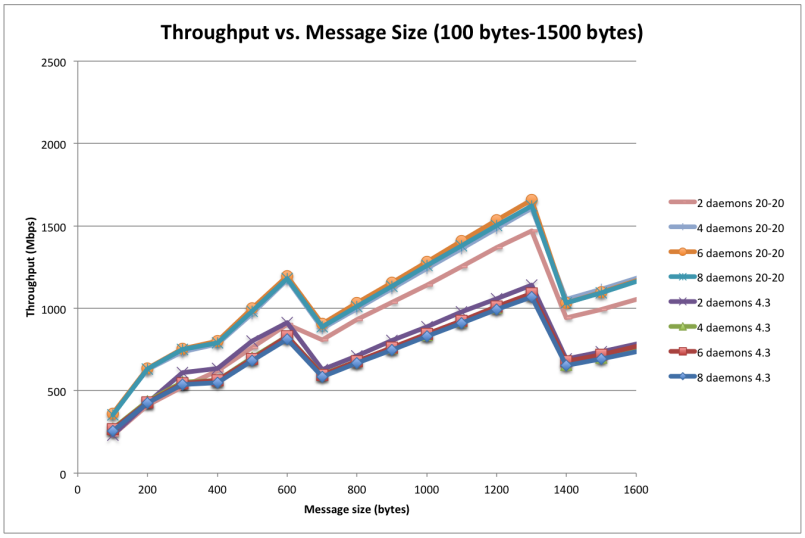

Figure 6.17: Throughput vs. message size for small messages, for a personal window of 20 on a 10-Gigabit network, with a sending client on every other daemon and a receiving client each daemon without a sending client. For the Accelerated Ring protocol, the accelerated window is also set to 20.

Figures 6.18 and 6.19 show results from experiments like those in Figures 6.16 and 6.19, assessing the impact of the number of daemons in the ring and the size of the

messages that are sent on throughput. However, for the experiments in Figures 6.18 and 6.19, each daemon has both a sending and receiving client, rather than just one or the other.

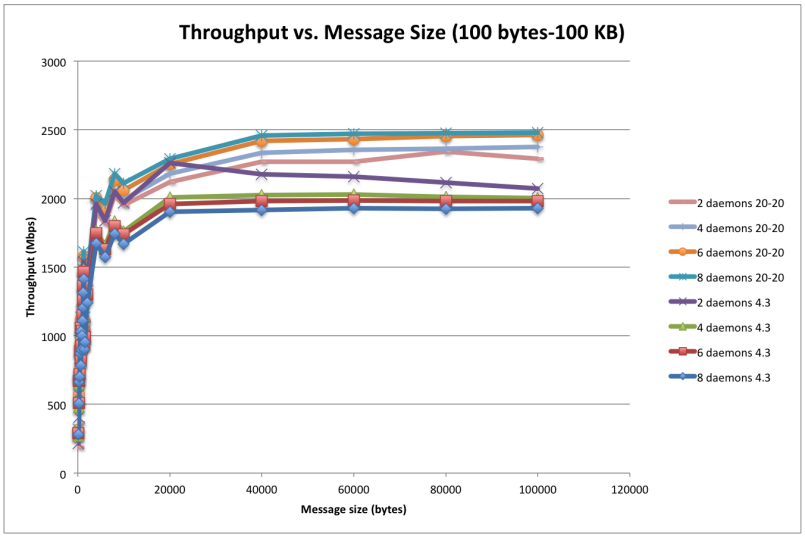

Figure 6.18: Throughput as message size increases, for a personal window of 20 on a 10-Gigabit network, with a sending client and a receiving client on each daemon. For the Accelerated Ring protocol, the accelerated window is also set to 20.

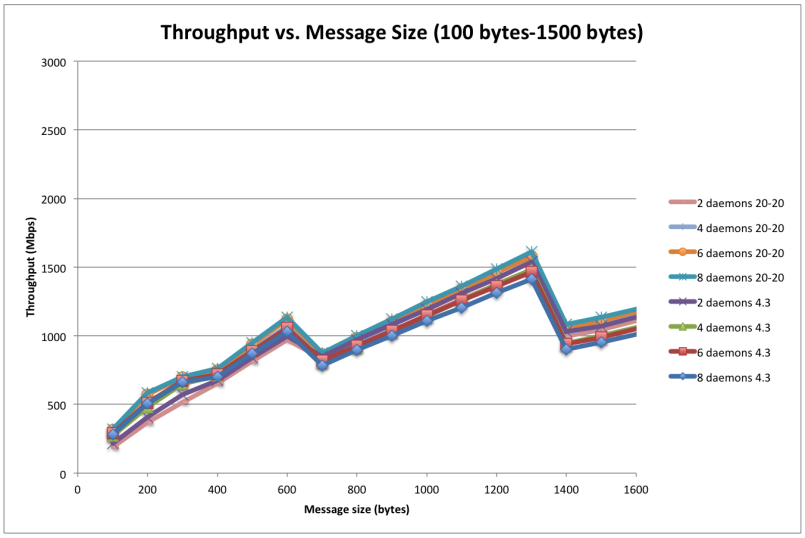

Figure 6.19: Throughput vs. message size for small messages, for a personal window of 20 on a 10-Gigabit network, with a sending client and a receiving client on each daemon. For the Accelerated Ring protocol, the accelerated window is also set to 20.

Chapter 7

Conclusions

This thesis introduced the Accelerated Ring protocol for ordered multicast in modern data center environments. This protocol is based on the idea that a token circulates a logical ring composed of the protocol participants, and each participant is able to send messages when it receives the token. The Accelerated Ring protocol improves performance over similar protocols, like the Totem Single Ring protocol on which it is based, by circulating the token around the ring more quickly. Rather than holding the token for the entire time it is sending, each participant is able to pass the token to the next participant before completing its sending for the round.

We analyzed the expected performance of the Accelerated Ring protocol as well as the original Ring protocol in detail. We implemented a prototype of the Accelerated Ring protocol's ordering algorithm and evaluated its performance in 1-Gigabit and 10-Gigabit LANs. The 1-Gig performance evaluation showed when the two protocols use similar flow control parameters, the Accelerated Ring protocol improves throughput by 25-70%, reduces agreed delivery latency by 10-30% relative to the original protocol and provides similar safe delivery latency to the original protocol or improves safe delivery latency by up to 25%. The 10-Gig evaluation showed an increase of 50-90% in throughput and a decrease of 10-40% in both agreed and safe delivery latency relative to the original protocol.

We also implemented the Accelerated Ring protocol in the Spread Toolkit, creating a complete practical implementation of the protocol. Evaluations of this practical implementation's performance in 1-Gigabit LANs in both native and virtualized environments, and in 10-Gigabit LANs showed performance improvements over previous versions of Spread similar to the improvements seen in the model evaluation. The Spread implementation of the Accelerated Ring protocol was made open-source and is publicly available at www.spread.org.

Bibliography

[1] Y. Amir, L. E. Moser, P. Melliar-Smith, D. A. Agarwal, and P. Ciarfella, "Fast message ordering and membership using a logical token-passing ring," in *Proceedings of the 13th International Conference on Distributed Computing Systems*. IEEE, 1993, pp. 551–560.

[2] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella, "The totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems (TOCS)*, vol. 13, no. 4, pp. 311–342, 1995.

[3] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 372–421, 2004.

[4] J.-M. Chang and N. F. Maxemchuk, "Reliable broadcast protocols," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 3, pp. 251–273, 1984.

[5] F. Cristian and S. Mishra, "The pinwheel asynchronous atomic broadcast protocols," in *Proceedings of the Second International Symposium on Autonomous Decentralized Systems*. IEEE, 1995, pp. 215–221.

[6] R. Guerraoui, R. R. Levy, B. Pochon, and V. Quéma, "Throughput optimal total order broadcast for cluster environments," *ACM Transactions on Computer Systems (TOCS)*, vol. 28, no. 2, 2010.

[7] P. J. Marandi, M. Primi, N. Schiper, and F. Pedone, "Ring paxos: A high-throughput atomic broadcast protocol," in *International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2010, pp. 527–536.

[8] Spread Concepts LLC, "The Spread Toolkit." [Online]. Available: http://www.spread.org

[9] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998.

[10] Y. Amir, "Replication using group communication over a partitioned network," Ph.D. dissertation, Hebrew University of Jerusalem, 1995.

[11] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal, "Extended virtual synchrony," in *Distributed Computing Systems, 1994., Proceedings of the 14th International Conference on*. IEEE, 1994, pp. 56–65.

Vita

Amy Babay was born in Erie, Pennsylvania in 1990. She received her B.A. in Cognitive Science from Johns Hopkins University in May 2012 and then switched fields to engineering. She joined the Distributed Systems and Networks Lab at Johns Hopkins University as an M.S.E. student in Computer Science and completed her Masters in May 2014.