QUALITY AND RELIABILITY ENGINEERING INTERNATIONAL
Qual. Reliab. Engng. Int. 2002; **18**: 165–184 (DOI: 10.1002/qre.473)

FIGHTING FIRE WITH FIRE: USING RANDOMIZED GOSSIP TO COMBAT STOCHASTIC SCALABILITY LIMITS

INDRANIL GUPTA, KENNETH P. BIRMAN* AND ROBBERT VAN RENESSE
Cornell University, Ithaca, NY 14853, USA

SUMMARY

The mechanisms used to improve the reliability of distributed systems often limit performance and scalability. Focusing on one widely-used definition of reliability, we explore the origins of this phenomenon and conclude that it reflects a tradeoff arising deep within the typical protocol stack. Specifically, we suggest that protocol designs often disregard the high cost of infrequent events. When a distributed system is scaled, both the frequency and the overall cost of such events often grow with the size of the system. This triggers an $O(n^2)$ phenomenon, which becomes visible above some threshold sizes. Our findings suggest that it would be more effective to construct large-scale reliable systems where, unlike traditional protocol stacks, lower layers use randomized mechanisms, with probabilistic guarantees, to overcome low-probability events. Reliability and other end-to-end properties are introduced closer to the application. We employ a back-of-the-envelope analysis to quantify this phenomenon for a class of strongly reliable multicast problems. We construct a non-traditional stack, as described above, that implements virtually synchronous multicast. Experimental results reveal that virtual synchrony over a non-traditional, probabilistic stack helps break through the scalability barrier faced by traditional implementations of the protocol. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: group communication; reliable multicast; scalability; randomized algorithms; non-traditional stack; virtual synchrony

1. INTRODUCTION

The scalability of distributed protocols and systems is a major determinant of success in demanding settings. This paper focuses on the scalability of distributed protocols, in group communication systems, providing some form of guaranteed reliability. Examples include the virtual synchrony protocols for reliable group communication [1], scalable reliable multicast (SRM) [2], and reliable multicast transport protocol (RMTP) [3]. We argue that the usual architecture for supporting reliability exposes mechanisms of this sort to serious scalability problems.

Reliability can be understood as a point within a spectrum of possible guarantees. At one extreme of this spectrum one finds very costly, very strong guarantees (for example, the Byzantine Agreement used to support Replicated State Machines). Replicated database systems represent a step in the direction of weaker, cheaper goals: one-copy serializability, typically implemented by some form of multiphase commit using a quorum read and write mechanism. Lamport's Paxos architecture and the consensus-based approach of Chandra and Toueg achieve similar properties. Virtual synchrony, the model upon which we focus here, is perhaps the weakest and cheapest approach that can still be rigorously specified, modeled and reasoned about. Beyond this point in the spectrum, one finds best-effort reliability models, lacking rigorous semantics, but more typical of the modern Internet—SRM and RMTP are of this nature. The usual assumption is that, being the weakest reliability properties, they should also be the cheapest to achieve and most scalable. One surprising finding of our work is that this usual belief is incorrect.

Traditionally, one discusses reliability by posing a problem in a setting exposed to some class of faults. Fault-tolerant protocols solving the problem

*Correspondence to: K. P. Birman, Department of Computer Science, Cornell University, 4130 Upson Hall, Ithaca, NY 14853, USA. Email: ken@cs.cornell.edu

Contract/grant sponsor: DARPA/AFRL-IFGA; contract/grant number: F30602-99-1-0532
Contract/grant sponsor: NSF-CISE; contract/grant number: 9703470
Contract/grant sponsor: AFRL-IFGA Information Assurance Institute
Contract/grant sponsor: Microsoft Research
Contract/grant sponsor: Intel Corporation

Copyright © 2002 John Wiley & Sons, Ltd.

can then be compared. The protocols cited above provide multicast tolerance of message loss and endpoint failures, and discussions of their behavior would typically look at throughput and latency under normal conditions, message complexity, background overheads, and at the degree to which failures disrupt these properties.

Oddly, even thorough performance analyses generally focus on the extremes—performance of the protocol under ideal conditions (when nothing goes wrong), and performance impact when injected failures disrupt execution. This paper adopts a different perspective; it looks at reliable protocols under the influence of what might be called mundane transient problems, such as network or processor scheduling delays and brief periods of packet loss. One would expect reliable protocols to ride out such events, but we find that this is rarely the case, particularly if we look at the impact of a disruptive event as a function of scale. In fact, reliable protocols degrade dramatically under this type of mundane stress, a phenomenon attributable to low-probability events that become both more frequent and more costly as the scale of the system grows.

After presenting these arguments, we shift attention to an old class of protocols that resemble NNTP, the gossip-based algorithm used to propagate 'news' on the Internet. These turn out to be scalable under the same style of analysis that predicts poor scalability for their non-gossip counterparts.

This finding leads us to suggest that protocol designers should fight fire with fire, employing probabilistic techniques that impose a constant and low system-wide cost to overcome these infrequent system-wide problems. In particular, we employ probabilistic reliability mechanisms at lower levels of the protocol stack, employing higher-level end-to-end layers that send very few additional messages. This is different from traditional protocol stacks that attempt to achieve reliability at the lower layers. In the case of reliable multicast, this yields a style of protocol very different from the traditional receiver-driven reliability or virtual synchrony solutions. Our approach reflects tradeoffs: the resulting protocols are somewhat slower in small-scale configurations, but faster and more stable in large ones.

We test this hypothesis by applying the approach to virtually synchronous multicast protocols. Traditional implementations of virtual synchrony, as in Amoeba, Ensemble, Eternal, i-bus, Isis, Horus, Phoenix, Rampart, Relacs, Transis, Totem, etc. [1], provide very good performance at small group sizes (i.e. perhaps 16 to 32 members), but partition away and break down at larger group sizes. However, the specification is popular with builders of distributed systems, and the systems mentioned above have been widely used in settings such as the Swiss Stock Exchange and the French Air Traffic Control System [1,4]. As noted earlier, our interest in the model stems both from its popularity and because it appears to be the weakest of the models that still offers rigorous guarantees.

We first use a back-of-the-envelope analysis to try and quantify the growth rate of disruptive overheads for this model. The analysis reveals that using traditional stacks inherently limits the scalability of these protocols (Gray *et al*. reach a similar conclusion with respect to database scalability in [5]). We next propose, analyze, and experiment with a non-traditional stack-based implementation of a virtually synchronous reliable multicast.

Our analysis and experiments reveal that such a probabilistic stack scales to much larger group sizes than a traditional one, even though it does not give as good a throughput at small group sizes. Although we still encounter some limits to scalability, these are far less severe than for standard implementations of the model.

In Section 2, we detail several problems with traditional reliable protocol stacks. In Sections 3 and 4, we outline a proposal to build lower stack layers with gossiping mechanisms. In Sections 5 and 6, we analyze and evaluate the performance of traditional as well as the new non-traditional stack for virtual synchrony. Section 7 summarizes our findings.

2. SCALABILITY AND RELIABILITY

Reliable multicast comes in many flavors. This section considers two points within the spectrum, studying their scalability properties. First, we look at the virtual synchrony model [6]. The model offers strong fault-tolerance and consistency guarantees to the user, including automated tracking of group membership, reporting membership changes to the members, fault-tolerant multicast, and various ordering properties. Scalability has been studied in a previous work on the Horus system [7] and we briefly summarize the findings, which point to a number of issues. We believe that these problems are fundamental to the way in which virtual synchrony is usually implemented, and that similar experiments would have yielded similar results for other systems in this class.

Next, we discuss multicast protocols in which reliability is 'receiver-driven'—group membership is not tracked explicitly, and the sender is consequently unaware of the set of processes which will receive each message. Receivers are responsible for joining

Copyright © 2002 John Wiley & Sons, Ltd.

themselves to the group and must actively solicit retransmissions of missing data. The systems community has generally assumed that because these protocols have weaker reliability goals, they should scale better; well-known examples include the reliable group multicast of the V system, RMTP, SRM, TIB, etc. SRM has been described in the greatest detail and a simulation was available to us, so we focus on that protocol. Perhaps surprisingly, the protocol scales as poorly as the virtually synchronous one, although for slightly different reasons. Moreover, we believe the same phenomena would limit scalability of the other protocols in this class. This finding leads us to draw some general conclusions that motivate the remainder of the paper.

2.1. *Throughput instability*

Consider Figure 1, which illustrates a problem familiar to users of virtually synchronous multicast. The graph shows the throughput that can be sustained by various sizes of process groups (32, 64 and 96 members), measuring the achievable rate from a sender attempting to send up to 200 messages/s to the group. This experiment was produced on an SP-2 cluster with no hardware multicast. We graph the impact of a 'perturbation' on the throughput of the group, using an optimized implementation that set throughput records among protocols in this class. A single group member was selected, and forced to sleep for randomly selected 100 ms intervals, with the probability shown on the x-axis. Each throughput value was calculated at an unperturbed process, using 80 successive throughput samples, gathered during a 500 ms period. The message size was 7 kbyte. The data was gathered on a cluster-style parallel processor, but similar results can be obtained for LANs, as reported in [4,8].

Focusing on the 32-member case, we see that the group can sustain a throughput of 200 messages/s in the case where no members are perturbed. As the perturbed member experiences growing disruption, throughput *to the whole group* is eventually impacted. The problem arises because the virtual synchrony reliability model forces the sender to buffer messages until all members in the group acknowledge receipt; as the perturbed member becomes less responsive, the message buffer fills up and throughput is affected. Moreover, flow control in the sender could also begin to limit the transmission bandwidth. Indeed, our experiment employed a fixed amount of buffering space at the sender. With this in mind, an application designer might consider adjusting the buffering

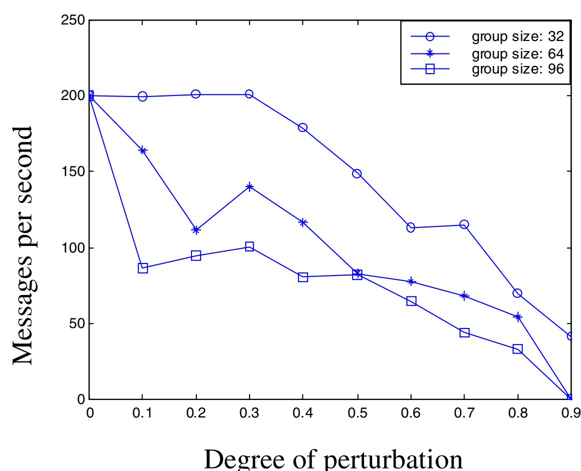

Figure 1. Throughput instability. Throughput instability grows as a reliable multicast protocol is scaled to large group sizes

parameters of the system to increase sender-side buffer capacities, and designing the application itself to communicate asynchronously, in the hope that the underlying communication system might soak up delays, much as TCP's sliding window conceals temporary rate mismatches between the sender and receiver.

Such a strategy is unlikely to succeed. Notice that for any fixed degree of perturbation, performance drops as a function of group size, primarily because of the linear time required by the two-phase commit protocol per multicast[†]. As a result, the knee of the curve shifts left as we scale up: with 64 members, performance degrades even with one member sleeping as little as 10% of the time, and with 96 members the impact is dramatic. The perturbation introduced by our experiment[‡] is not such an unlikely event in a real system: random scheduling or paging delays could easily cause a process to sleep for 100 ms at a time, and such behavior could also arise if the network became loaded and messages were delayed or lost. Indeed, some small amount of perturbation would be common on any platform shared with other applications, especially if some machines lack adequate main memory, have poor cache hit rates, or employ slow and flaky network links. Our graph suggests that a strategy focused on sender-side buffering would apparently need buffering space to grow at least linearly with the group size, even with just one perturbed member.

[†]This means that even in the presence of (unreliable) hardware multicast, one would observe trends similar to those in Figure 1.

[‡]We should note that similar results are obtained when multiple processes are subjected to perturbation, and when the network is disrupted by injecting packet delays or losses. The throughput graphs differ, but the trend is unchanged.

Copyright © 2002 John Wiley & Sons, Ltd.

In fact, the buffering required might be worse than linear, as the group-wide perturbation rate tends to rise linearly with group size.

2.2. *Micropartitions*

Faced with this sort of problem, a system designer who needs to reliably sustain a steady data rate might consider setting failure detection thresholds of the system more and more aggressively as a function of scale. The idea would be to knock out the slow receiver, thereby allowing the group as a whole to sustain higher performance. To a reader unfamiliar with virtually synchronous multicast, the need to make failure detection more aggressive as a function of system size may not seem particularly serious. However, such a step is precisely the last thing one wants to do in a scalable reliable group multicast system. The problem is that in these systems, membership changes carry significant costs. Each time a process is dropped from a group, the group needs to run a protocol (synchronized with respect to the multicast stream) adjusting membership, and reporting the change to the members.

The problem gets worse if the failure detector is parameterized so aggressively that some of the dropped processes will need to rejoin. Erroneous failure decisions involve a particularly costly 'leave/rejoin' event. We will term this a *micropartitioning* of the group, because a non-crashed member effectively becomes partitioned away from the group and later the partition (of size one) must remerge. In effect, by setting failure detection parameters more and more aggressively while scaling the system up, we approach a state in which the group may continuously experience micropartitions, a phenomenon akin to thrashing. Moreover, aggressive failure detection mechanisms ensuring that a process will be dropped for even very minor perturbations, brings us into a domain in which a typical *healthy* process has a significant probability of being dropped because of the resulting noise.

Note that this problem would be expected to grow with the square of the size of the group. This is because the frequency of mistakes is at least linear[§] with the size of the group, and the cost of a membership change in systems of this sort is also linear with the group size. Section 5 fleshes out this observation.

This phenomenon is familiar to the designers of large reliable distributed systems. For example, the developers of the Swiss Exchange trading system (an all-electronic stock exchange, based on the Isis Toolkit) comment that they were forced to set failure detection very aggressively, but that this in turn limited the number of machines handled by each 'hub' in their architecture [4].

2.3. *Convoys*

One possible response to the scalability problem presented above is to structure large virtually synchronous systems hierarchically, as a tree of process groups. Unfortunately, when the hierarchy is several levels deep, this option is also limited by random events disrupting performance.

Consider a small group of processes within which some process is sending data at a steady rate, and focus on the delivery rate to a healthy receiver. For any of a number of reasons, the rate is likely to be somewhat bursty unless data is artificially delayed. For example, messages can be lost or discarded, or may arrive out of order; normally, a reliable multicast system will be forced to delay the subsequent messages until the missing ones are retransmitted. The sender may be forced to use flow control. Messages may pass through a router, which will typically impose its own dynamics on the data stream. This is illustrated in Figure 2. Data enters a hierarchical group at a steady rate (illustrated by the evenly spaced black boxes), but layer by layer becomes bursty. This phenomenon is widely known as the 'Convoy' phenomenon.

Kalantar [9] found that each new level of group amplifies the burstiness of its data input source. He notes that the problem is particularly severe when the upper levels of the hierarchy maintain a multicast ordering property, such as totally-ordered message delivery. When messages arrive out of order, such a property forces delays, but also results in the delivery of a burst of ordered messages when the gap has been filled; thus, the next layer sees a bursty input which can trigger flow control mechanisms (even if the original flow would not have required flow control). If the top-level group multicasts at a high data rate, the gaps between bursts represent dead time and effectively reduce the available bandwidth, while the bursts themselves are likely to exceed the available bandwidth.

Kalantar suggests a few remedies. First, hierarchical systems might be designed to enforce weak ordering properties near the sender, reintroducing stronger guarantees close to the receiver. However, this design point has never been explored in practice, in part because ordering and reliability are hard to separate. A second option involves delaying messages on

[§] For all-to-all heartbeating schemes for failure detection [15], the rate of erroneous failure detections grows as $O(n^2)$, since any member might misdiagnose a failure of any other member.

Copyright © 2002 John Wiley & Sons, Ltd.

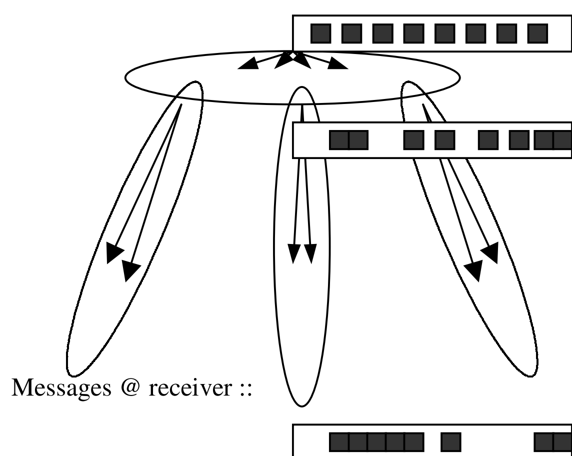

Figure 2. Message convoys. Message convoys in hierarchical groups

receipt (layer by layer, or end-to-end) to absorb the expected degree of rate variations. However, this would demand a huge amount of buffering and the delays could be large. Kalantar concludes that lacking one of these mechanisms, large hierarchically-structured process groups will perform poorly.

2.4. *Request and retransmission storms*

One might speculate that the problems seen above are specific to the virtual synchrony reliability model. However, several researchers studying the SRM protocol have observed a related scalability phenomenon.

SRM is a reliable multicast protocol that uses a receiver-driven recovery mechanism, whereby the onus falls on the receiving process to join itself to the transmission group (an IP multicast group within the Internet), to begin collecting data, and to request retransmissions of missing data. SRM is based on a model called application-level framing, which basically extends the end-to-end model into the multicast domain. The idea is that the IP multicast layer is oblivious to the protocol using it, hence the SRM request and retransmission mechanisms reside in the application (in a library). One consequence is that although the protocol uses IP multicast to send messages, retransmission requests and retransmissions, the IP multicast layer is oblivious to the manner in which it is being used. The only control available to the protocol itself is in the value used for the IPMC time-to-live (TTL) field, which limits the number of hops taken before a packet is dropped.

Since IP multicast is an unreliable protocol, a multicast packet might be dropped by a router (or the sender's operating system), and hence fail to reach large numbers of receivers, although this is not the usual situation. To avoid subjecting the full set of participants to a storm of requests and retransmissions, SRM uses a timer-based delay scheme reminiscent of exponential backoff. Processes delay requests (for missing data) for a randomly selected period of time, calculated to make it likely that if a subtree of the IP multicast group drops a message, only one request will be issued and the data will be retransmitted only once. The protocol also uses the TTL values in a manner intended to restrict retransmissions to the region within which the data loss occurred.

Unfortunately, recent studies have shown that the SRM tactics are not as effective as might be hoped, particularly in very large networks (hundreds or thousands of members) subject to low levels of random packet loss or link failures. At least three simulation studies have demonstrated that under these conditions, a large percentage of the packets sent trigger multiple requests, each one of which, in turn, triggers multiple multicast retransmissions.

The data shown in Figure 3 [8] reflects this problem; similar findings have been reported in [10,11]. ns-2 (which includes a simulation of SRM, including its adaptive mechanisms) was used to graph the rate of requests for retransmissions and repairs (retransmissions) for groups of various sizes. We constructed a simple four-level tree topology, injecting 100 210-byte messages/s, and setting parameters as recommended by SRM's developers. We set a system-wide message loss probability at 0.1% on each link, and measured overhead at typical processes (with other topologies and noise rates we get similar graphs).

The intuition is that the basic SRM mechanisms are ultimately probabilistic. As a network becomes large, the frequency of low-probability events grows at least linearly with the size of the network. For example, as a network scales, there will be processes further and further apart which may each (independently) experience a packet loss. By symmetry, these have some probability of independently and simultaneously requesting a retransmission, and even with SRM's 'scalable session messages', variability in network latency may be such that neither request inhibits the other. Each process that receives such a request and has a copy of the multicast in its buffers, has some probability of resending it. Again, although there is an inhibitory mechanism, it is probable that more than one process may do so. Thus, as the network is scaled

Copyright © 2002 John Wiley & Sons, Ltd.

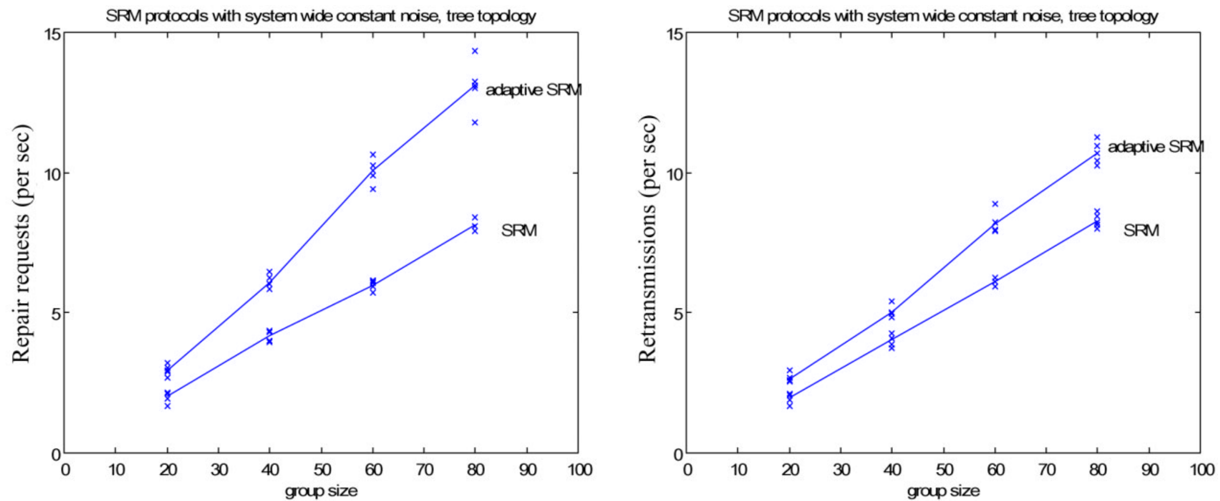

Figure 3. SRM scalability. As the size of the group increases, a low level of background noise (0.1% in this case) can trigger high rates of requests (left) and retransmissions (right) for the SRM protocols, at each group member. Most of these are duplicates. Notice that the data rate is being held constant; only the size of the group is increased in these experiments

and the global frequency of these low-probability events rises, one begins to observe growing numbers of requests for each multicast packet. Depending on the network topology, each request may result in multiple retransmissions of the actual data. Although the latter problem is not evident in the simple tree used to construct Figure 3, with a 'star' topology and the same experimental setup, SRM sends roughly three to five repairs for each request. All but the first are 'duplicates'. Thus, the aggregate overhead rises with group size, and the effect can be considerably worse than linear¶.

Note that although SRM has reliability goals which are weaker than those of virtual synchrony, once again we encounter a mechanism with costs linear in system size and frequency growing, perhaps linearly, with system size. Indeed, while intuition would suggest that because SRM has weaker goals than virtual synchrony it should be cheaper and more scalable, we see here that SRM scalability is no better than that of virtual synchrony, at least for the implementations studied.

We thus observe that the throughput degradation experienced for virtual synchrony protocols is not unique to that reliability model. In the case of SRM, the problems described above contribute to a growth in background load (seen in Figure 3), unstable throughput much like that shown in Figure 1, and can even overload routers to such a degree that the system-wide packet loss rate will rise sharply. Should this occur, a scale-driven performance collapse is likely.

Moreover, the issues described above are also seen in other reliability mechanisms. While brevity limits our discussion in this paper, it is not difficult to see that similar problems would arise in RMTP [3], publish–subscribe message bus protocols such as the one in TIB, and indeed in most large-scale multicast architectures (based, of course, on the published descriptions of such systems).

Forward error correction (FEC) is a pro-active mechanism whereby the sender introduces redundancy into the data stream so that the receiver can reconstruct lost data [12]. However, FEC entrusts the message-sending node to send all the copies of a multicast, leading to a throughput limitation similar to that in the traditional multicast protocols (Figure 1).

3. BIMODAL MULTICAST

Not all protocols suffer the behavior seen in these reliable mechanisms. In particular, Bimodal Multicast, a protocol reported in [8], scales quite well and easily rides out the same phenomena that cause problems with these other approaches.

Bimodal Multicast is a 'gossip-based' protocol that closely resembles the old NNTP protocol (employed by network news servers), but runs at much higher speeds. The protocol has two sub-protocols. One of them is an unreliable data distribution protocol similar to IP multicast, and in fact IP multicast can be used for

¶It might appear to the reader that adjusting the backoff timing parameters less aggressively in the SRM protocol would reduce such overhead. Unfortunately, doing so results in a linear increase in multicast dissemination latency.

Copyright © 2002 John Wiley & Sons, Ltd.

this purpose if it is available[‖]. Upon arrival, a message enters the receiver's *message buffer*. Messages are delivered to the application layer in FIFO order, and are garbage collected out of the message buffer after some period of time.

The second sub-protocol is used to repair gaps in the message delivery record, and operates as follows. Each process in the system maintains a list containing some random subset of the full system membership. In practice, we weight this list to contain primarily processes from close by—processes accessible over low-latency links. These details go beyond the scope of the current paper, and we refer the reader to [13].

At some rate (but not synchronized across the system) each participant selects one of the processes in its membership list at random and sends it a *digest* of its current message buffer contents. This digest would normally just list messages available in the buffer; for example, 'messages 5–11 and 13 from sender $s, \ldots$'. Upon receipt of a gossip message, a process compares the list of messages in the digest with its own message buffer contents. Depending upon the configuration of the protocol, a process may *pull* missing messages from the sender of the gossip by sending a retransmission *solicitation*, or may *push* messages to the sender by sending unsolicited retransmissions of messages apparently missing from that process, or do both (*push–pull*).

This simplified description omits a number of important optimizations to our implementation of the protocol. We sometimes use unreliable multicast with a regional TTL value instead of unicast, in situations where it is likely that multiple processes are missing copies of the message. A weighting scheme is employed to balance loads on links: gossip is performed primarily to nearby processes over low-latency links and rarely to remote processes, over costly links that may share individual routers. The protocol uses both gossip pull and gossip push, the former for 'old' messages and the latter for 'young' ones. Finally, every message need not be buffered at every process; a hashing scheme is used to spread the buffering load around the system, with the effect that the average message is buffered at enough processes to guarantee reliability, but the average buffering load on a participant decreases with increasing system size. Further details are available in [8].

Bimodal Multicast imposes loads (per multicast) on participants that are logarithmic with the system size, and the protocol is simple to implement and inexpensive to run. More important from the perspective of this paper, however, the protocol overcomes the problems cited earlier for other scalable protocols. Bimodal Multicast has tunable reliability that can be matched to the needs of the application (basically, reliability is increased by increasing the length of time before a message is garbage collected—this time typically varies with the logarithm of the group size). The protocol gives very steady data delivery rates with predictable, low variability in throughput. For soft-real-time applications this can be extremely useful. Also, the protocol imposes constant loads on links and routers (if configured correctly), which avoids network overload as a system scales up. All of these properties are preserved as the size of the system increases.

The reliability guarantees of the protocol are midway between the very strong guarantees of virtual synchrony and the much weaker best-effort guarantees of a protocol like SRM or a system like TIB. We will not digress into a detailed discussion of the nature of these guarantees, which are probabilistic, but it is interesting to note that the behavior of Bimodal Multicast is predictable from certain simple properties of the network on which it runs. Moreover, the network information required is robust in networks like the Internet, where many statistics have heavy-tailed distributions with infinite variance. This is because gossip protocols tend to be driven by successful message exchanges and hence by the 'good' quartile or perhaps half of network statistics. In contrast, protocols such as SRM or RMTP often include round-trip estimates of the mean latency or mean throughput between nodes; estimates that are problematic in the Internet where many statistical distributions are heavy-tailed and hence have ill-defined means and very large variances.

Bimodal Multicast is just one of the many presentations of gossip-based data replication mechanisms. Similar protocols can also be used in lazy database consistency [14], failure detection [15], data aggregation, or in *ad hoc* networking settings. For the purposes of this paper, the scalability of the one-to-many configuration of the protocol is the most interesting issue.

Figure 4 [8] compares the performance of the Bimodal Multicast primitive with SRM and adaptive SRM under similar noise conditions in the underlying network. The plot shows that SRM imposes per-message overheads on each group member that increases linearly with group size, while the load due to Bimodal Multicast is logarithmic with group size. Other studies [16] have shown similar advantages

[‖]Because IP multicast is often disabled in modern networks, however, Bimodal Multicast more often runs over a very lightweight unicast-based tree management and multicasting mechanism of our own design.

Copyright © 2002 John Wiley & Sons, Ltd.

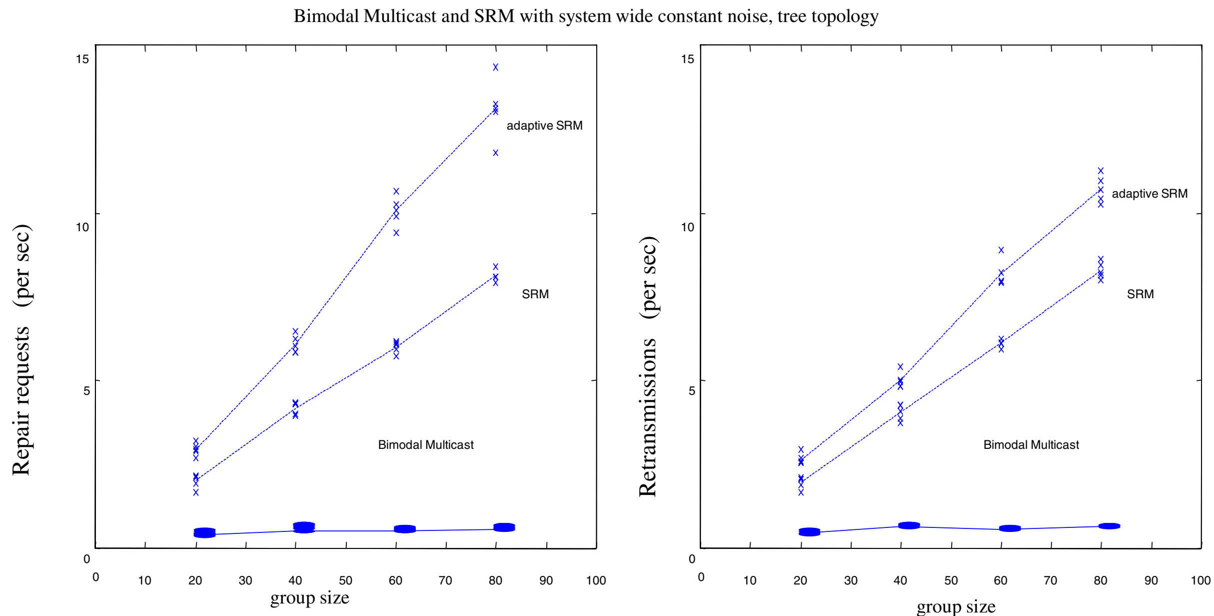

Figure 4. Bimodal Multicast. Bimodal Multicast imposes per-member overheads that are logarithmic in group size. Versions of SRM impose a linear load, as shown in Figure 3

for Bimodal Multicast over RMTP. The reader is encouraged to refer to these papers for extensive comparative studies.

4. FINDINGS SO FAR AND PROVIDING STRONGER GUARANTEES

We can generalize from the phenomena described above. Distilling these down to their simplest form, and elaborating slightly, we obtain the following.

- With the exception of the Bimodal Multicast protocol, each of these reliability models involves a costly, but infrequent, fault-recovery mechanism.
  - Traditional virtual synchrony protocols employ sender-side buffering, flow control, failure detection and membership-change protocols, with a cost that is at least proportional to the size of the group.
  - SRM has a solicitation and retransmission mechanism that involves multicasts; when a duplicate solicitation or retransmission occurs, all participants process an extra message.
  - FEC tries to reduce retransmission requests to the sender by encoding redundancy in the data stream. The multicast sender's responsibility to transmit all of its redundant copies results in scalability limitations similar to those of the traditional protocols.
- All of the protocols we have reviewed are potentially at risk from convoy-like behaviors. Even if data is injected into a network at a constant rate, as it spreads through the network router scheduling delays and link congestion can make the communication load bursty. While this phenomenon has actually been observed in hierarchical implementations of virtual synchrony, we have not seen anything comparable in our work with Bimodal Multicast, or with virtual synchrony running over Bimodal Multicast.
- Many protocols (SRM, RMTP) depend on configuration mechanisms that are sensitive to network routing and topology. Over time, network routing can change in ways that take the protocol increasingly far from optimal, in which case the probabilistic mechanisms used to recover from failures can be increasingly expensive or inaccurate. Periodic reconfigurations, the obvious remedy, introduce a disruptive system-wide cost.

4.1. *General insight*

Is there a general insight to be drawn from these observations?

- Many reliable multicast protocols depend upon assumptions that usually hold, but may have a small probability of being violated. As a system

Copyright © 2002 John Wiley & Sons, Ltd.

scales, however, the frequency of violations grows.

- These protocols typically have some form of recovery mechanism with a potentially global cost. As the system scales up, this cost grows.
- Each of these protocols is built in layers (i.e. as a protocol stack). The problems cited arise within the lowest layers, although they then propagate to higher layers triggering large-scale disruptions of performance.
- Sender-side responsibility for reliable multicast dissemination is a direct outcome of the traditional stack architecture.

More broadly, we argue that these are all consequences of a protocol stack architecture in which *reliability is provided by the lowest layers of the stack*, and in which *randomized phenomena* are threats to performance or some other property guaranteed by the system. The insight is then that as we scale the system to larger and larger settings, the absolute frequency of these probabilistic events rises, and hence the performance of the system degrades.

4.2. *Our solution—a non-traditional stack architecture*

We envision a protocol stack where the lower layers use randomized strategies (such as Bimodal Multicast) to fight random network events, while layers closer to the application guarantee the reliability required by the application. We believe that such a stack would help fight fire (random network unreliability events) with fire (randomized algorithms) at lower layers, while guaranteeing good performance at large distributed group sizes by placing the reliability requirement at a layer closer to the application. For problems that are difficult to solve, or are considered to be inherently unscalable (such as virtual synchrony), the new probabilistic stack would suffer from this limitation too, but would give better scalability than traditional stacks.

5. SCALING VIRTUAL SYNCHRONY

In the next few sections, we attempt to validate our hypothesis by attacking the problem of achieving virtual synchrony in distributed process groups. Informally, virtual synchrony is maintained in a process group if each group member (an application process over a virtual synchrony stack) sees both (a) multicast events, and (b) group membership changes (also called a ‘view change’), in the same order [6]. For simplicity, we require all multicast events in our system to be seen by all group members (effectively, we allow only broadcasts within the group).

Virtual synchrony is a very strong reliable multicast model; implementing the model is known to be as hard as solving distributed consensus [17]. From a practical point of view, the power of the model accounts for its popularity: one can easily implement high-performance versions of state-machine algorithms. Yet, if high performance is the primary reason for using the model, one would obviously hope that performance is sustainable as the size of a group is scaled up. Here, we use a back-of-the-envelope scalability analysis to try and quantify the question (Section 5.1).

We analyze two possible implementations—the traditional implementation using reliable multicast at the lower layers, and our non-traditional stack, with Bimodal Multicast at the lower layer. Our analysis in Section 5.1 reveals that *both* approaches inherently suffer from an $O(n^2)$ degradation in throughput (multicasts per second) with increasing group size, and reasonable assumptions on the characteristics of the underlying network and nodes. However, the analysis also reveals that the new probabilistic stack approach potentially scales much better than the traditional stack.

This leads us to further evaluate our hypothesis by designing, building and testing one such non-traditional stack for virtual synchrony. We describe the architecture of this system in Section 5.2, and experimental findings in Section 6. Our findings do support our hypothesis, and we report stable throughput performance at group sizes larger than traditional virtual synchrony protocols have so far been able to achieve.

5.1. *Scaling virtual synchrony—a back-of-the-envelope analysis*

This section presents a back-of-the-envelope analysis for two different implementations of virtual synchronous multicast. This analysis seeks to identify scaling trends of the different protocols, and thus ignores constants in asymptotic estimates. Section 5.1.1 discusses the behavior of the traditional unicast-based implementation of virtual synchrony, as in the Isis system [18]. Section 5.1.2 examines properties of virtual synchrony within a non-traditional protocol stack.

For simplicity, our analysis in this section assumes the presence of only the second sub-protocol of Bimodal Multicast (see Section 3), i.e. the gossip-

Copyright © 2002 John Wiley & Sons, Ltd.

Table 1. Parameters used in the back-of-envelope analysis

n	Total number of group members
s	Number of senders in the group
T	Total (non-negative) throughput of the group (multicasts/s)
μ	Individual member failure rate
p_{ml}	Probability of multicast delivery failure (for simplicity, assumed independent across messages and recipients)
B	Maximum network bandwidth allotted to each group member

based push dissemination. Henceforth, we will use the shorthand 'Pbcast' (for 'Probabilistic Broadcast') for this phase of Bimodal Multicast. We use the notation shown in Table 1 in the rest of our analysis.

p_{ml} is the network message loss probability for the traditional stack analysis in Section 5.1.1. When we use Pbcast in Section 5.1.2, p_{ml} is used to derive the message reliability as a function of the number of gossip rounds used for the multicast (we specify this more concretely where needed). Although p_{ml} is assumed to be uniform and independent across all messages in order to simplify the analysis, the results of the analysis hold if this parameter is the time-bounded loss rate of messages, across the system. Our analysis only considers a constant number of senders (s). When the number of senders varies with group size, the message load on receiver-side members is difficult to characterize (e.g. message losses due to buffer overflows, etc.), and is not accounted for in our analysis.

5.1.1. Virtual synchrony, unicast implementation. Consider virtual synchrony implemented entirely with unicast messages, as in traditional systems such as Isis [1]. This approach is based on the multicast layer (lower layer) attempting to reliably multicast messages by using unicast (or point-to-point) messages for multicasting. The higher layer uses multiphase commit protocols to change membership.

The optimal (non-negative) throughput T^* achievable under this implementation can be expressed as follows:

$$\frac{T}{s} = \left[1 - \mu n \left(\frac{n}{B}\right)\right] \frac{1}{n/B}$$

The first term on the right-hand side comes from the effective time left for transmitting 'useful' multicasts, obtained by discounting the time the group is 'blocked' while changing views. μn is the number of view changes per unit time, and n/B is the time taken to verify the view with all group members. The second term $(1/(n/B))$ is simply the rate at which a particular member can transmit its messages or check acknowledgements—this arises from constant buffering at each member, as in the traditional virtual synchrony implementation. The above equation gives us:

$$T^* = \max\left\{(B - \mu n^2)\frac{s}{n}, 0\right\} \tag{1}$$

Note that when the numerator becomes negative, the throughput goes to zero.

In the actual implementation, however, each non-faulty group member verifies the stability of its own multicasts directly with other group members during the view changes. Thus, by a similar explanation as above,

$$\frac{T}{s} = \left[1 - \mu n \left(\frac{n + p_{\text{ml}} T(1/\mu n)n}{B}\right)\right] \frac{1}{n/B}$$

where each view change involves updating the membership (the n term above) and repairing 'holes' in the received message list for the last view for each of the group members (the term $p_{\text{ml}} T(1/\mu n)n$ above). This gives us:

$$T = \max\left\{\frac{B - \mu n^2}{(n/s) + p_{\text{ml}}}, 0\right\} \tag{2}$$

which is fairly close to the optimal throughput T^*.

This analysis reveals that the virtual synchrony model faces an inherent scalability problem due to the cost of view changes (the $(-\mu n^2)$ term above). However, even when this does not have an effect on the throughput, the sender-side multicast responsibility results in an unscalable throughput (the n/s term in the denominator above). As a result, even for a problem such as state machine replication [19] that is weaker than virtual synchrony (since there is no extra cost for view changes), the sender-side responsibility of reliable multicast makes the overall group throughput unscalable.

5.1.2. Virtual synchrony, implemented over Pbcast. Now, consider a virtual synchrony implementation that uses a distributed multicast dissemination mechanism, repairing holes at the

Copyright © 2002 John Wiley & Sons, Ltd.

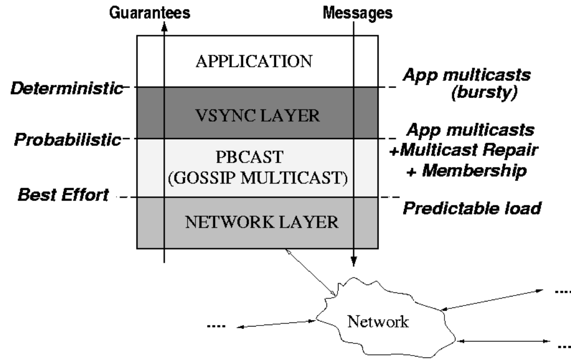

Figure 5. New non-traditional stack for virtual synchrony

higher Vsync (virtual synchrony) layer. Figure 5 shows an example of such a stack where the dissemination mechanism is the Pbcast primitive. The higher Vsync layer repairs holes in the multicast message stream delivered by the Pbcast layer. The ordered stream of multicasts is delivered by the Vsync layer to the application layer.

Below, we first ignore the cost of view changes, and analyze the optimality of different multicast dissemination schemes (lower layer) for reliable multicast applications (i.e. where every member has to receive at least one copy of every multicast, and in the same global order). The optimality is calculated with respect to the tradeoff between the number of multiple copies of a multicast received by a typical group member, and the number of repairs that are transmitted by the upper layer in the protocol stack, in order to guarantee reliability. This analysis applies to problems such as state machine replication. We then analyze the achievable throughput under the stronger virtual synchrony model by accounting for the cost of view changes.

Abstractly, let M be the number of copies of a single multicast that a member receives from the lower dissemination layer. For the Pbcast implementation, since each member gossips a given multicast for $O(\log(n))$ gossip rounds and to randomly chosen targets, we have $M = O(\log(n))$.

To maximize the throughput, one would want to minimize the number of copies M of a multicast that a member receives from the multicast dissemination layer, as well as the number of repairs sent by the upper protocol layer, in order to repair 'holes' in the message delivery stream from the multicast dissemination layer. The second term is $p_{\mathrm{ml}}^M n$, since p_{ml}^M is the likelihood of a given member receiving none of the copies of a multicast. The expression to be minimized, is thus:

$$L = p_{\mathrm{ml}}^M n + M$$

Setting $\mathrm{d}L/\mathrm{d}M = 0$ gives us:

$$M = \frac{\log(n) + \log(1/p_{\mathrm{ml}})}{\log(1/p_{\mathrm{ml}})} \tag{3}$$

or $M = O(\log(n))$, which is exactly what the Pbcast primitive achieves with each member gossiping a given multicast for $O(\log(n))$ gossip rounds. So, in a sense, the *Pbcast primitive is a provably optimal multicast dissemination mechanism* with respect to the minimum load on participants required to achieve reliable delivery.

Not surprisingly, implementations of reliable multicast in the Isis/Horus-style stack, as well as over SRM behavior (Figure 3), can also be characterized by a similar analysis. In Horus, the dissemination layer consists of each member verifying the stability of the message with every other member, with the higher Vsync layer repairing the holes. In SRM, each member receives multiple copies of each multicast (repairs) that, from Figure 3, is linear with the group size. The value of the expression L for this is thus obtained by setting $M = O(n)$. From the above analysis, using any of these two mechanisms at the lower multicast dissemination protocol layer gives a (provably) sub-optimal implementation of reliable multicast.

We now analyze the achievable throughput of the implementation of virtual synchrony when implemented over Pbcast, as depicted in Figure 5. This analysis adds to the cost of view changes, and its effect on throughput.

The optimal throughput T^* under this implementation is:

$$\frac{T}{s} = \left[1 - \mu n \left(\frac{n}{B}\right)\right] \left(\frac{s \log n}{B}\right)^{-1}$$

where $((s \log n)/B)^{-1}$ is the optimal rate at which a multicast is disseminated to all the group members if there were no message losses. This gives us:

$$T^* = \max\left\{(B - \mu n^2)\frac{1}{\log n}, 0\right\} \tag{4}$$

which is *asymptotically higher than the optimal throughput under the traditional implementation of virtual synchrony* (obtained in Equation (1)). Also notice that the throughput is independent of the number of senders.

A real implementation of the stack in Figure 5 would need to repair message holes at the view changes. We propose using a leader committee with

Copyright © 2002 John Wiley & Sons, Ltd.

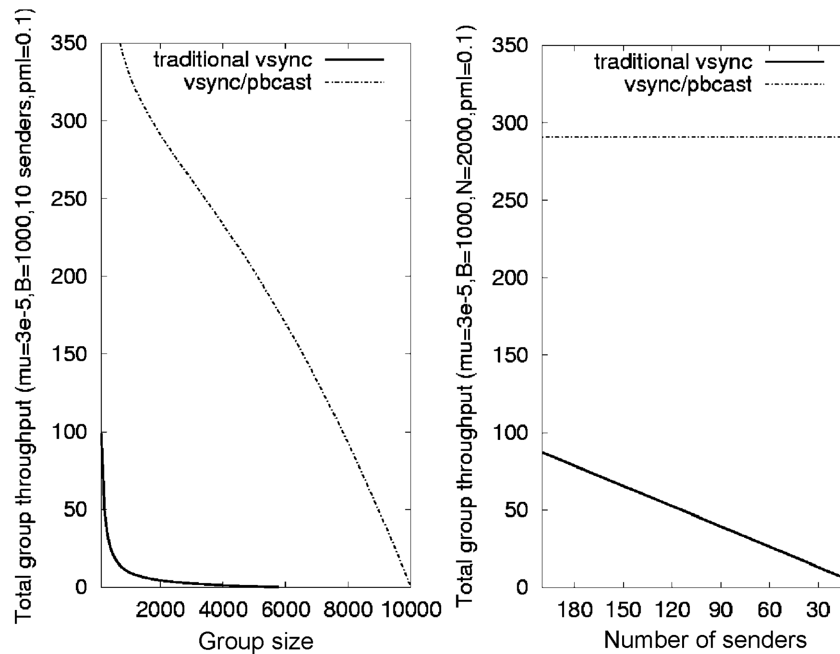

Figure 6. Virtual synchrony scales better over Pbcast (‘vsync/pbcast’) than within the traditional stack (‘traditional vsync’). The curves plotted represent Equations (2) and (5)

k members (call it the ‘k-committee’ or ‘leader-committee’) for this purpose. The k-committee maintains virtual synchrony even though all other group members may be well behind in the virtual synchrony. All multicasts are first communicated to the committee, where they attain stability, are assigned a global sequence number and then are gossiped about in the group by the sender. The committee membership can migrate among members, using less-loaded members at any time. Such a k-committee can be elected using any of the several committee election strategies found in the literature. The failure of a committee member will result in the inclusion of some other non-faulty member into the committee. The failure of a non-committee member will be seen as yet another event by the entire k-committee. Both these kinds of failures result in view changes, first in the k-committee, then at the members, which receive the view notification just like any other multicast, through the Pbcast primitive. Note that this scheme is only $(k-1)$-fault-tolerant to committee member failures, but can tolerate any number of faults in the general group membership.

The number of gossip rounds a multicast survives (and is gossiped about) in the group before garbage collection is $s(C\log(n))$, where $C\log(n)$ is the number of rounds each multicast is gossiped. We define $p_{\mathrm{ml}}(C)$ as the Pbcast delivery loss probability at any group member. We then have $p_{\mathrm{ml}}(C) \sim p_{\mathrm{ml}}^{-C\log(n)}$ [8].

The throughput of this scheme can thus be expressed as:

$$\frac{T}{s} = \left[1 - \mu n\left(\frac{n + p_{\mathrm{ml}}(C)T(1/\mu n)n}{kB}\right)\right] \times \left(\frac{s(C\log n)}{B}\right)^{-1}$$

$$\Rightarrow T = \max\left\{\frac{B-(\mu n^2/k)}{C\log n + (p_{\mathrm{ml}}(C)n/k)}, 0\right\} \quad (5)$$

This approach thus scales almost as well as the optimal T^*, since C can be adjusted as $C > (\log(1/p_{\mathrm{ml}}))^{-1}$ so that $p_{\mathrm{ml}}(C)n$ is a constant or smaller. Thus, as n rises, T would fall very slowly below T^*.

Notice from the optimal T^* calculations in Equations (1) and (4) above that *neither the traditional nor the probabilistic non-traditional stack approaches to virtual synchrony is inherently scalable*, since as n grows, μn^2 would approach (fixed) B, and T would go to zero. This phenomenon remains even if B is scaled linearly with n. However, notice that the optimal throughput of virtual synchrony is asymptotically better with the non-traditional stack. Figure 6 plots the above throughput equations (Equations (2) and (5)), for purposes of comparison of the trends of the throughput degradation—the values themselves do not mean much (since the equations lack constants). The figures show that at a small

Copyright © 2002 John Wiley & Sons, Ltd.

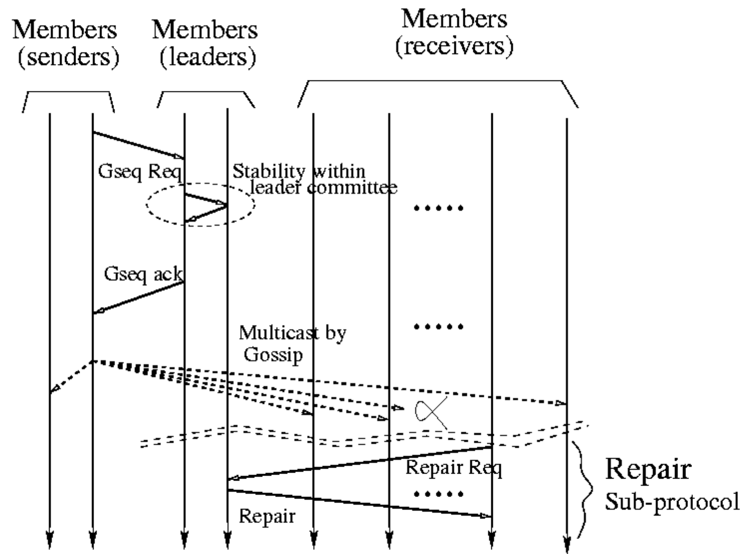

Figure 7. Stages in sending a multicast to the group

number of senders, with rising group size, the non-traditional stack approach ('vsync/pbcast') degrades more gracefully than the traditional implementation. The traditional implementation appears to improve as the number of senders increases, but the approach is limited by the group size scalability itself.

5.2. *Algorithm description*

In this section, we give details of the implementation of the non-traditional stack of Figure 5, with a k-committee. This approach bears resemblance to Kaashoek's token based approach [20], and the server-based approach of [21], although there are algorithmic differences.

Each multicast from a sender member goes through the stages shown in Figure 7. The Vsync layer at the sender first sends the message to a leader-committee member, as a request to assign it a global sequence number (`Gseq Req` message). The message is assigned a global sequence number by total ordering within the leader-committee. On receiving the acknowledgement with this information (`Gseq ack` message), the Vsync layer at the sender submits the message to the Pbcast layer underneath it for dissemination by gossiping.

Insofar as the Pbcast layer does not guarantee that multicast messages reach all group members, we need a mechanism to overcome possible packet loss. Accordingly, the Vsync layer at a member times-out if it does not receive an in-sequence multicast after some time. The Vsync layer then requests a repair from one of the leader committee members—we henceforth call this the 'repair sub-protocol'. The frequency of such repairs is minimized by the good reliability characteristics of the Pbcast layer. During periods when no multicasts are being sent, the repair protocol timeout at members is increased in order to avoid a repair implosion at the leader. However, the timeout must still be small enough so that an isolated, sporadic multicast would be discovered were it to be lost in the Pbcast layer. We currently vary the timer from a minimum of 10 ms, when a high rate of multicasts is being received, to a maximum of 1 s, when multicasts are infrequent.

View changes are initiated by the leader-committee. Unlike traditional virtual synchrony mechanisms [1,6], view changes are initiated periodically (here, once every 20 s), and not on every member failure. This delays the reporting of new views, but reduces the impact of view change events on throughput.

Every view change consists of the following phases, as shown in Figure 8. The leaders first establish/reconfirm the membership within the leader-committee itself. The leader-committee then multicasts a `Prenewview` message throughout the group (by using the Pbcast primitive). Non-faulty group members reply back with an acknowledgement—an `AckPrenewview` message. The leader-committee waits some time for these acknowledgements (1 s in our implementation), agrees on the group membership for the next view, and multicasts it out as a `Newview` message multicast via the Pbcast layer. This `Newview` message specifies any changes to the group membership, and is assigned

Copyright © 2002 John Wiley & Sons, Ltd.

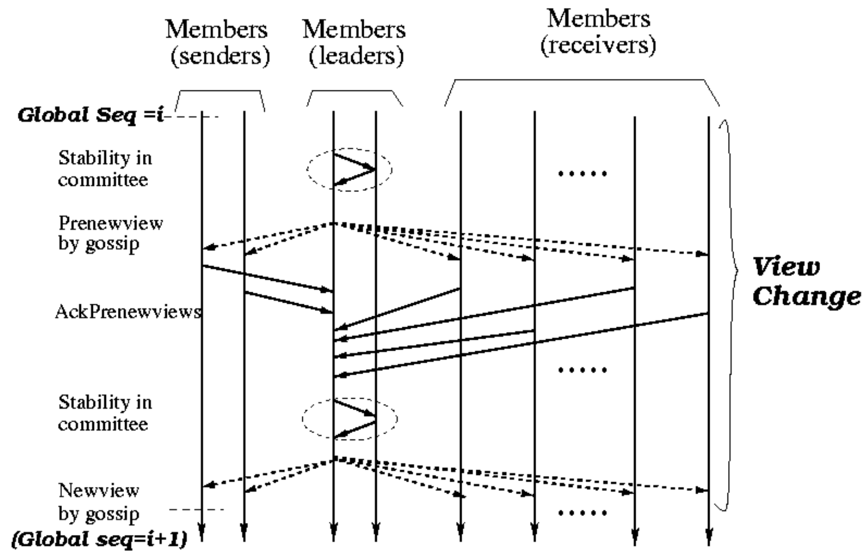

Figure 8. Anatomy of a view change

the next available global sequence number. This makes each `Newview` message a multicast within the group, with guaranteed delivery by the repair sub-protocol. This also preserves the total ordering of each view change with respect to other multicasts and view changes.

This view change mechanism has some shortcomings, most notable the possibility of an ack-implosion at the leader-committee members. Our solution to this technique is to have members defer their acknowledgements (`AckPrenewview` messages) randomly in time.

The periodic view changes reduce the overhead of a view change per failure, but might result in increasing the time for the view change—although this increase is linear in the group size, our experiments show that it is typically small in the absence of any perturbed or failed member (see Section 6). The periodic view change offers a checkpoint beyond which messages can be garbage-collected at members. The approach would thus let us use a single state-transfer to initialize multiple new members, an opportunity not evaluated here. Use of a gossip-style failure detection service [15] or epidemic-style membership [22] will also help eliminate the costlier `Prenewview-Ackprenewview` phase of the view change mechanism. This would, however, substantially affect the group's multicast throughput, as the failure detection gossips would compete with the multicast gossips in the Pbcast layer. Exploration of these areas is beyond the goal of this paper and is left for future study.

Note that a multicast-sending member learns the view in which the message will be delivered in the group only when it receives the `Gseq ack` from the leader-committee for that message. This is in keeping with the specification of the original Isis protocol [18]. The multicast dissemination and the repair sub-protocol are blocked during view changes, in order to reduce the number of involuntary member droppings. The leader-committee delays pending `Gseq Req` messages (Figure 7) until the end of the view change process.

Loss of `Gseq Req` messages and view changes at the committee members might lead to a phenomenon similar to the convoy problem described in Section 2.3. However, the extent of the problem is small here, since there are effectively only two hierarchy levels. Our experiments show that if leaders and senders stay unperturbed, view changes are the only times that senders stay blocked.

Figure 9 summarizes our discussion. It shows the typical state of a group at a hypothetical instant in time. The 'most complete' event history is maintained at the leader-committee, which has delivered the greatest number of virtually synchronous messages and views (bold lines in Figure 9). At a general member, however, the Pbcast layer may have failed to deliver some messages to the Vsync layer. This results from the probabilistic guarantees of the protocol, which is not required to achieve atomicity and may create 'holes' in the message delivery stream to the Vsync layer. The resulting holes must be repaired, leading virtual synchrony at that member to lag behind the current advancing frontier of messages delivered by Pbcast (gray lines in Figure 9). If the gap is not repaired quickly by the Pbcast protocol

Copyright © 2002 John Wiley & Sons, Ltd.

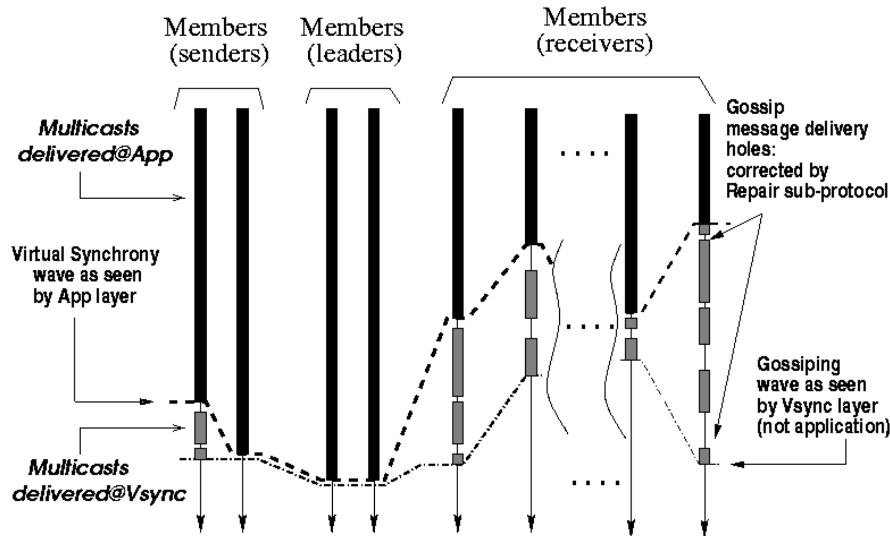

Figure 9. Lazy virtual synchrony. States of a set of group members depicted as partial histories. Virtually synchronous message deliveries are most advanced at the leaders; members lag both in terms of virtual synchrony delivery (bold lines) and also Pbcast delivery events (gray lines)

itself, it will eventually be subject to our repair sub-protocol. We quantify the potential size of this gap in Section 6 and show that the gap is insignificant for healthy members, and that the lazy nature of this protocol allows slightly-perturbed members to catch up with the rest of the group. A heavily-perturbed member would diverge and drop out from the virtually synchronous membership view.

6. EXPERIMENTAL EVALUATION

In this section, we describe performance results of an implementation of virtual synchrony using a non-traditional probabilistic stack, as presented in Section 5.2.

The experimental environment was as follows. The processing nodes used were a mix of machines (nodes), each with a 300 or 450 MHz Pentium II processor, 130 Mbyte RAM, and running Windows 2000, with hardware multicast disabled. The network was a 100 Mbps Ethernet, with little or no external load. All experiments refer to an implementation of the non-traditional stack for virtual synchrony, as described in Section 5.2, implemented over the Winsock API in the Win32 environment.

In order to maximize the size of the test configurations considered, we resorted to placing multiple group members on each node. This number was limited by the interaction of multiple group members on a single node, due to the effects of scheduling delays, memory utilization and network contention. Experimentally, we observed that the CPU utilization with up to four members per node was limited to 20%, and the throughput performance for single member/node and four members/node were comparable, for up to a group size of 16. The CPU utilization at the leader member's node, however, was 100%. Accordingly, we used exactly four members per node, with the exception of the leader members, which were put on individual nodes.

To mimic the experimental conditions that triggered scalability problems in Figure 1, our initial experiments in Sections 6.1–6.4 use one leader member in the committee, elected statically, and one multicast sender (a different member). Multiple leaders increase the fault-tolerance and load-balancing properties of the scheme. Our experiments in Section 6.5 reveal that increasing committee size results in a rise in the sender-side latency between the `GseqReq` generation and multicast delivery. However, stable throughput can still be maintained by increasing the size of the sender-side buffer for waiting `GseqReq` messages.

The view change period at the leader was fixed at 20 s, with a timeout of 1 s for the `AckPrenewview` messages. Unless otherwise stated, no involuntary member droppings at view changes were encountered during our experiments.

The following sections investigate the effect of increasing group size and member perturbations on the group throughput (Sections 6.1 and 6.2), as well as measurements of message loads on different group members (Section 6.4). Section 6.3 demonstrates the lazy nature of the protocol as depicted in Figure 9, and Section 6.5 investigates the effect of larger committee sizes. Section 6.6 summarizes the experimental findings.

Copyright © 2002 John Wiley & Sons, Ltd.

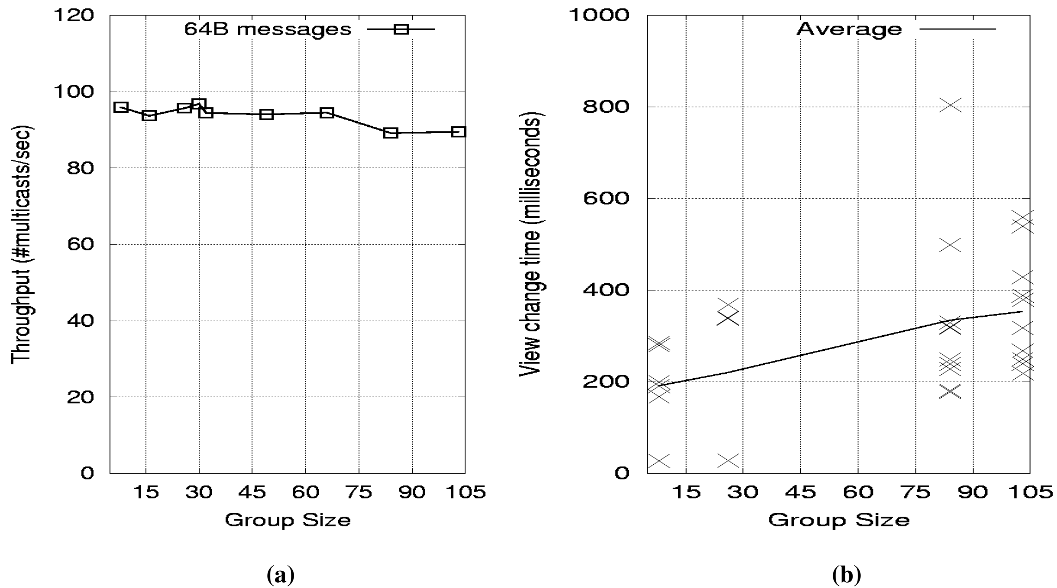

Figure 10. Throughput scalability of non-traditional stack. For our protocol: (a) throughput scales gracefully with group size; (b) average time for view changes (no member failures) increases with group size

6.1. *Scalability*

Figure 10(a) shows the maximum achievable stable throughput in groups of up to 105, for a single sender sending 64-byte messages. Each message is sent as a separate multicast and there is no packing of multiple messages into each protocol packet. Each point on this curve is the average of a 60 s stable run (i.e. no involuntary member droppings at view changes) of the protocol. We observe that the throughput remains stable until about 70 members, after which it begins to slowly drop off. The reason for the drop is not unscalability of the Pbcast dissemination mechanism, but the longer times taken for view changes, as shown in Figure 10(b). (Recall that in our protocol, normal messages are blocked during view changes.) This phenomenon points to an inherent scalability limitation for virtual synchrony, but as can be seen in Figure 10(b), is a very modest phenomenon. The stability of the throughput attained by the non-traditional stack, at these high group sizes, is better than that which traditional virtual synchrony implementations can achieve.

The curve in Figure 11 gives throughput for larger message sizes. Horus, using its ‘protocol accelerator’, is able to pack about 1000 4-byte messages into a 4-kbyte packet, and has quoted a peak rate of 85 000 multicasts/s for a four-process group [23]. Our protocol is slower, but not drastically—with similar packing, we would achieve 20 000 multicasts per second with up to 70 members.

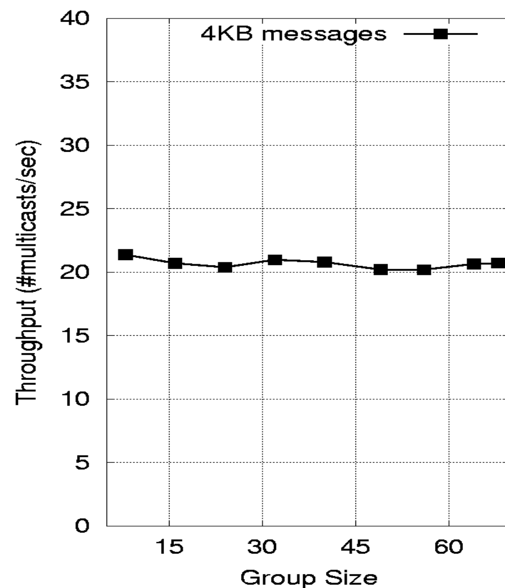

Figure 11. Maximum throughput achievable. With 4-kbyte multicasts, it is stable with group size. With message packing, this is about four times slower than the peak throughput in a group of four members using the traditional Horus implementation

6.2. *Insensitivity to perturbations*

Figure 12 shows the effect of perturbations at a single member on the group throughput for a group with 66 members. One of the group members was placed on an otherwise unloaded machine (i.e. no other group members on this machine), and made to sleep for randomly selected 100 ms intervals, with

Copyright © 2002 John Wiley & Sons, Ltd.

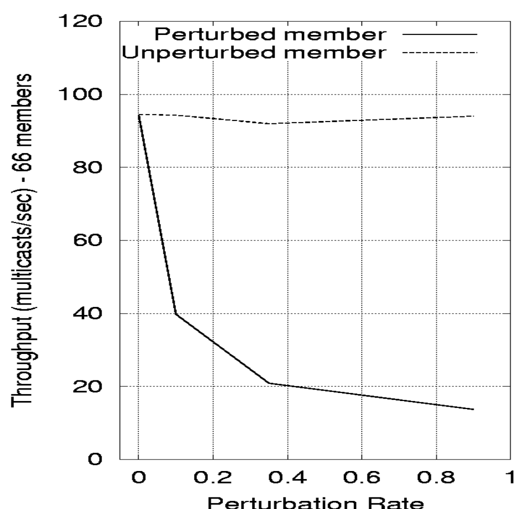

Figure 12. Perturbation at a member has very little effect on the group's throughput

the perturbation probability shown on the x-axis. The throughput (multicasts/s) was measured over a 60 s interval, at both the perturbed and other unperturbed members. Figure 12 shows the throughput for one such unperturbed member and the perturbed member. In all instances of perturbation probability above 0.01, the perturbed member quickly fell behind the rest of the group and was observed to drop out at the next view change after the start of the perturbation. The effect of this perturbation on the group's throughput is minimal, being limited to the effect of the extra time taken during the view change (in our case, 1 s) when the perturbed member drops out and the leader stays blocked. We did notice that an occasional healthy member would fall behind and drop out at a subsequent view change—however, this was a rare phenomenon.

Figure 12 thus shows that our non-traditional stack is relatively insensitive to the perturbations that caused such dramatic degradation in Figure 1. We obtained similar results at a number of group sizes smaller than 100, and we believe that the same behavior will hold even for larger groups.

6.3. *Laziness of the new non-traditional stack*

Recall from Figure 9 that our protocol allows message delivery at members to lag behind delivery at the leader. Figures 13(a) and 13(b) demonstrate the presence of the two 'waves' of message delivery in the group, as predicted. These plots are a snapshot of a 0.6 s message delivery sequence at three members in a group with 32 members: (1) the advancing wave of virtual synchrony closely following the wave of message delivery at the leader (labeled 'Msgs@Leader'); (2) delivery of messages at the Vsync and the App layers from underneath (i.e. latest message deliveries in the Pbcast and the virtual synchrony streams, respectively), as seen at a healthy member ('Msgs@Vsync' and 'Msgs@App' in Figure 13(a), respectively) and a perturbed member (same labels in Figure 13(b)). Also shown is the delivery of a view change (`Newview`) message at the different layers at the two members. Notice that our approach effectively has the leader 'drive' the waves of delivery at the Vsync and App layers of the non-traditional stack at a group member. Figure 13(a) shows that virtual synchrony at a healthy group member lags behind the leader because of holes in the Pbcast delivery stream, but recovers lazily due to the repair sub-protocol. For example, the separation created at time $t = 0.15$ s between the advancing virtual synchrony frontier at the leader and at the member (created by holes in the Pbcast delivery stream), is recovered at time $t = 0.5$ s. Perturbed members, on the other hand, could begin to lag behind the leader by several milliseconds, as in Figure 13(b). As Figure 13(a) shows, there is good chance of perturbed members catching up with the leader (through the repair sub-protocol) if they recover soon. Members that stay perturbed for too long, as in Figure 13(b), diverge and ultimately drop out of the group (an event not shown on plot). Note that the perturbed member (Figure 13(b)) has no effect on the message delivery stream at the unperturbed member (Figure 13(a)), a property we observed in Section 6.2.

6.4. *Message load on members*

Some multicast protocols impose a high background load and deliver large numbers of messages to the application. Our protocol is generally well behaved in these respects. Figures 14 and 15 show the number of messages received at the Vsync layer at the sender and at a group member. The plot shows a scenario over a 400 s interval during which the sender transmitted 100 multicasts/s during two brief periods: $t = 240$ to $t = 260$ s and $t = 540$ to $t = 580$ s. In between, we increased the size of the group from 32 members to 64 members. As is evident from the graphs, during periods of quiescence, even with membership change, there is only a very light load, due to checking for possible lost multicasts. During periods of sender activity, the Pbcast layer delivers most of the multicasts, and the number of repairs is negligible.

The measured load at the Vsync layer in the leader member is an average of 199.4 messages/s in the

Copyright © 2002 John Wiley & Sons, Ltd.

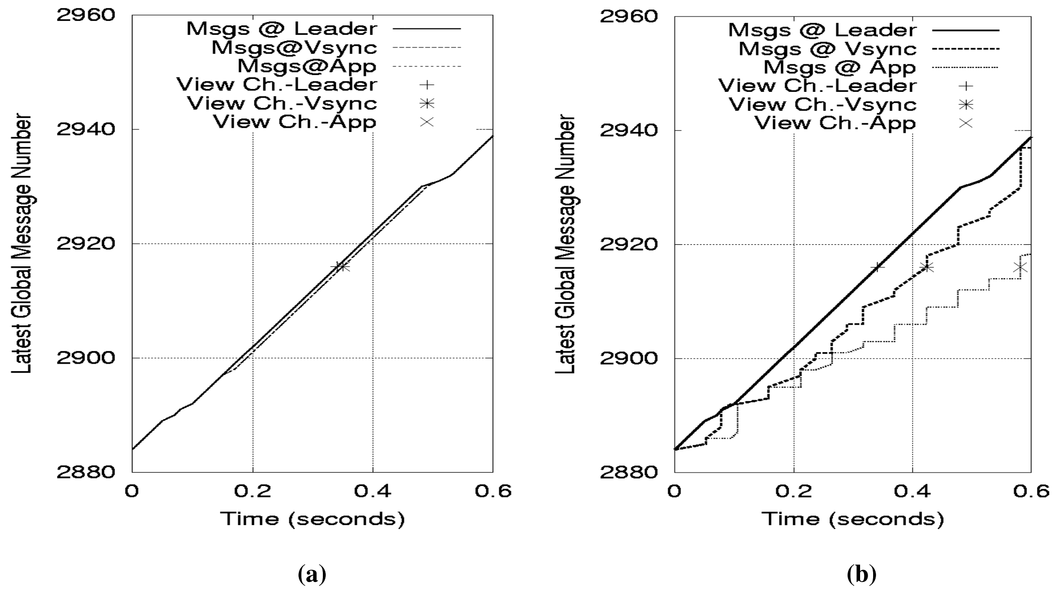

Figure 13. Delivery waves at Vsync, App layers. At the leader and (a) a healthy member and (b) a perturbed member that eventually drops out

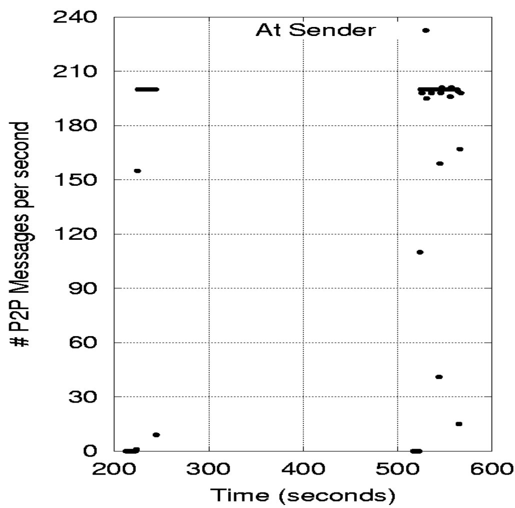

Figure 14. Message load at a sender with sporadic transmission

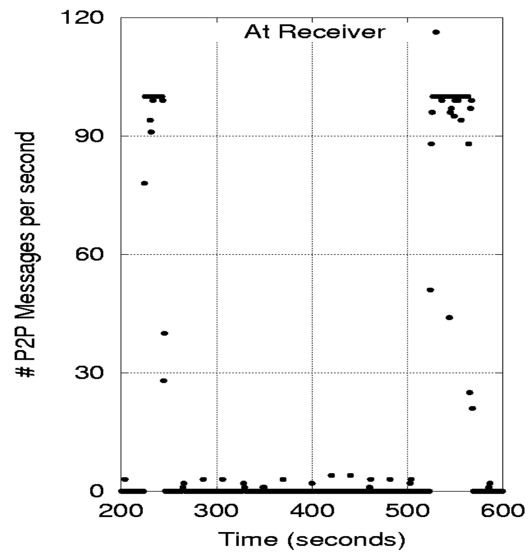

Figure 15. Message load at a group member with sporadic transmission

presence of an active sender in the group. These messages mostly comprise the `Gseq Req` messages and the multicast delivery (each at a rate of about 100 messages/s). In the absence of a sender in the group, each group member sends repair solicitations at the rate of 1/s. Subsequently, the load on the leader is 1 message/s for each group member**.

**This scheme could be generalized to adapt the rate of repair solicitations to the recent rate of message delivery.

6.5. *Effect of larger leader committees*

The experiments in Sections 6.1–6.4 were performed for a leader-committee with one member. Increasing the leader-committee membership size results in a rise in latency between `Gseq Req` generation for a multicast and its delivery at the sender (see the first two columns of Table 2). This latency increase results from the overhead associated with stabilizing the multicast within the committee. Our implementation hides the effect of sporadic rises in latency of `Gseq req`

Copyright © 2002 John Wiley & Sons, Ltd.

Table 2. Leader-committee latency. For a single sender attempting to transmit 100 64-byte multicasts/s, latency at the sender member (between `Gseq Req` generation and final receipt of the multicast) rises with committee size. Stable group throughput is maintained using a sender-side, constant-size buffer containing messages waiting for `Gseq acks`. The group in this experiment contains only the committee members and the sender

Committee size (k)	Average latency at the sender (ms)	Maximum measured latency at the sender (ms)	Group throughput (multicasts/s)
1	1.39	192.09	97.64
2	1.68	27.56	98.32
3	2.16	15.02	97.93
4	3.31	157.56	98.33

(see the third column of Table 2) by using a sender-side message buffer, with a size that depends on the committee size. As a result, stable overall group throughput can be sustained for a committee of up to four members (see the last column of Table 2).

6.6. *Summary of experimental results*

The data presented in Sections 6.1–6.5 has demonstrated that using a non-traditional, probabilistic stack helps virtual synchrony scale better than in traditional stacks. Groups of up to and above 100 members can sustain stable throughput (multicasts/s), with perturbations on individual members having practically no effect on message delivery flow in the rest of the group. This is in contrast to the behavior of traditional stacks discussed in Section 2. Members that recover from perturbation catch up lazily with the advancing frontier of virtual synchrony in the group, while members that stay perturbed for too long eventually drop out of the group. The load on the sender and receiver members, as well as the leader-committee members is low, even with only one member in the committee. Multiple leader-committee members (k) result in an increase in fault-tolerance ($k - 1$), and message delivery latency, but do not affect throughput.

7. SUMMARY AND CONCLUSIONS

The fundamental question posed by our paper concerns the scalability of strong reliability properties. We focused on the weaker parts of the spectrum, known from the literature, and used a variety of methods (analysis, experimentation) to understand their scalability.

Our analysis led us to recognize a number of problems that are common when one undertakes to 'scale up' a reliable multicast-based system, and we argued that these can be understood as the outcome of a battle between random phenomena and deterministic properties. Traditional protocols exhibit symptoms of unscalability that include throughput instability, flow control problems, convoys seen in ordered multicast delivery protocols, and high rates of duplicated retransmission requests or un-needed retransmitted data packets in protocols using receiver-driven reliability. We trace these problems to a form of complexity argument, in which the growth in size of a system triggers a disproportionate growth in overhead. Our work suggests that many of the best known and most popular reliable protocol architectures degrade with system size. In a mechanical sense, this phenomenon stems from properties of traditional stacks, such as sender-based reliable multicast dissemination, and enforcement of reliability flow control in low protocol layers.

The alternative we propose involves building a non-traditional stack: lower layers are gossip-based and have probabilistic properties, while stronger properties are introduced with an end-to-end mechanism (such as virtual synchrony, as we have used in this paper) closer to the application. Experiments confirm that this approach yields substantial immunity to the scalability limits mentioned. In the case of virtual synchrony, our end-to-end mechanism imposes scalability limits of its own, but the solution degrades slowly and is far more stable at large group sizes than traditional implementations.

Randomized low-level phenomena that compromise system-wide performance are an unrecognized but serious threat to scalability. While we often brush concerns about 'infrequent events' to the side when designing services, in the context of a scalability analysis it becomes critical that we confront these issues, and their costs. Probabilistic gossip mechanisms fight fire with fire: they overcome infrequent disruptive problems with mechanisms having small, localized costs. In a world where scalability of network mechanisms is rapidly becoming the most important distributed computing challenge, appreciating the nature of these effects and architecting systems to minimize their disruptive impact is an issue which is here to stay.

Copyright © 2002 John Wiley & Sons, Ltd.

ACKNOWLEDGEMENTS

The authors wish to thank Oznur Ozkasap and Zhen Xiao for Figures 1, 3 and 4. The authors also thank Ben Atkin for his helpful comments.

The authors were supported in part by DARPA/AFRL-IFGA grant F30602-99-1-0532 and in part by NSF-CISE grant 9703470, with additional support from the AFRL-IFGA Information Assurance Institute, from Microsoft Research and from the Intel Corporation.

REFERENCES

1. Birman KP. *Building Secure and Reliable Network Applications*. Manning Publications and Prentice-Hall: Greenwich, CT, 1997.
2. Floyd S, Jacobson V, McCanne S, Liu C, Zhang L. A reliable multicast framework for light-weight sessions and application level framing. *Proceedings of the ACM SIGCOMM '95 Conference*, Cambridge, MA, August 1995. Association for Computing Machinery: New York, NY, 1995.
3. Paul S, Sabnani K, Lin K, Bhattacharya S. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communication* 1997; **15**(3):407–421.
4. Piantoni R, Stancescu C. Implementing the Swiss Exchange Trading System. *Proceedings 27th International Symposium on Fault-tolerant Computing*, Seattle, WA, June 1997. IEEE Computer Society Press: Los Alamitos, CA, 1997; 309–313.
5. Gray J, Helland P, O'Neil P, Shasha D. The dangers of replication and a solution. *Proceedings of the ACM SIGMOD '96 Conference*, Montreal, QC, June 1996. Association for Computing Machinery: New York, NY, 1996.
6. Birman KP, Joseph TA. Exploiting virtual synchrony in distributed systems. *Proceedings of the 11th Symposium on Operating Systems Principles*, Austin, TX, November 1997. Association for Computing Machinery: New York, NY, 1997; 123–138.
7. van Renesse R, Birman KP, Maffeis S. Horus, a flexible group communication system. *Communications of the ACM* 1996; **39**(4):76–83.
8. Birman KP, Hayden M, Ozkasap O, Xiao Z, Minsky Y. Bimodal Multicast. *ACM Transactions on Computer Systems* 1999; **17**(2):41–88.
9. Kalantar M, Birman K. Causally ordered multicast: The conservative approach. *Proceedings International Conference on Distributed Computing Systems '99*, Austin, TX, June 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999.
10. Liu C. Error recovery in Scalable Reliable Multicast. *PhD Dissertation*, University of Southern California, December 1997.
11. Lucas M. Efficient data distribution in large-scale multicast networks. *PhD Dissertation*, University of Virginia, May 1998.
12. Rubenstein D, Kurose J, Towsley D. Real-time reliable multicast using proactive forward error correction. *Proceedings 8th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98)*, Cambridge, UK, July 1998. IEEE Computer Society Press: Los Alamitos, CA, 1998.
13. van Renesse R. Scalable and secure resource location. *Proceedings 33rd Hawaii International Conference on System Sciences*, Maui, HI, January 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
14. Demers A *et al*. Epidemic algorithms for replicated data management. *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing (PODC '87)*, Vancouver, British Columbia, Canada, August 1997. Association for Computing Machinery: New York, NY, 1997; 1–12.
15. van Renesse R, Minsky Y, Hayden M. A gossip-style failure detection service. *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, Lake District, UK. Springer: New York, NY, 1998.
16. Xiao Z, Birman KP. Providing efficient robust error recover through randomization. *Proceedings International Workshop on Applied Reliable Group Communication*, Phoenix, AZ, April 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
17. Chandra TD, Hadzilacos V, Toueg S, Charron-Bost B. On the impossibility of group membership. *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing (PODC '96)*, Philadelphia, PA. Association for Computing Machinery: New York, NY, 1996; 322–330.
18. Birman KP, van Renesse R. Software for reliable networks. *Scientific American* 1996; **274**(5):64–69.
19. Schneider F. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys* 1990; **22**(4):299–319.
20. Kaashoek MF, Tanenbaum AS, Verstoep K. Group communication in Amoeba and its applications. *Distributed Systems Engineering* 1993; **1**(1):48–58.
21. Keidar I, Khazan R. A client-server approach to virtually synchronous group multicast: Specifications and algorithms. *Proceedings 20th International Conference on Distributed Computing Systems*, Taipei, Taiwan, April 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000; 344–355.
22. Golding R, Taylor K. Group membership in the epidemic style. *Technical Report UCSC-CRL-92-13*, University of California at Santa Cruz, May 1992.
23. van Renesse R. Masking the overhead of protocol layering. *Proceedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, September 1996. Association for Computing Machinery: New York, NY, 1996.

Authors' biographies:

Indranil Gupta is a doctoral student in the Department of Computer Science at Cornell University. His current research interests lie in designing and building scalable and reliable protocols for distributed and peer-to-peer systems. He has also co-authored conference and journal publications in the areas of real-time systems and *ad hoc* networking. (His homepage is http://www.cs.cornell.edu/gupta).

Kenneth Birman is a Professor in the Department of Computer Science at Cornell University, which he joined after receiving his PhD from the University of California, Berkeley in 1981. He heads the Spinglass project at Cornell. He has also founded two companies, Isis Distributed Systems (acquired by Stratus Computer in 1993) and Reliable Network Solutions (http://www.rnets.com).

Robbert van Renesse is a Senior Research Associate in the Department of Computer Science at Cornell University, as well as co-founder and Chief Scientist of Reliable Network Solutions, Inc. He holds a PhD from the Free University of Amsterdam. His work focuses on development of fault-tolerant distributed systems and network communication protocols. He has published over 75 papers and is a member of the IEEE and the ACM.

Copyright © 2002 John Wiley & Sons, Ltd.