Spinglass: Secure and Scalable Communication Tools for Mission-Critical Computing

Kenneth P. Birman, Robbert van Renesse, Werner Vogels
Dept. of Computer Science, Cornell University, Ithaca, New York
{ken,rvr,vogels}@cs.cornell.edu;
http://www.cs.cornell.edu/info/projects/spinglass/spinglass-main.htm

Abstract

Most existing communications technologies are either not scalable at all, or scale only under carefully controlled conditions. This threatens an emerging generation of mission-critical but very large computing systems, which will need communication support for such purposes as system management and control, policy administration, data dissemination, and to initiate adaptation in demanding environments. Cornell University's Spinglass project has discovered that "gossip-based" protocols can overcome scalability problems, offering security and reliability even in the most demanding settings. Gossip protocols emulate the spread of an infection in a crowded population, and are both reliable and stable under forms of stress that can disable more traditional protocols. Our effort is developing a new generation of gossip-based technology for secure, reliable large-scale collaboration and soft real-time communications – even over global networks.

1. Introduction

Distributed computing will be central to advances in a broad range of critical applications, including intelligence information systems, military command and control, air traffic control, electric power grid management, telecommunications, and a vast array of web-based commercial and government applications. Indeed, a massive rollout of such systems is already underway. Yet while impressive capabilities have been easy to develop and demonstrate in small-scale settings, once deployed these systems often stumble badly.

Software that runs securely and reliably in small-scale mockups may lose those properties as numbers of users, the size of the network, and transaction processing rates all increase. Whereas small networks are well-behaved, any sufficiently large network behaves like the public Internet, exhibiting disruptive overloads and routing changes, periods of poor connectivity and throughput instability. Failures rise in frequency simply because the numbers of participating components are larger. A scalable technology must ride out such forms of infrastructure instability, imposing loads that are either constant, or growing very slowly, as a function of system size and network span.

Our studies reveal that very few existing technologies have the necessary properties. Most, including the most prevalent commercial software, exhibit scalability problems when subjected to even modest stress. This finding reveals an imminent (and growing) threat to the full spectrum of emergent mission-critical computing systems. If we can't solve the scalability problem, and develop a methodology yielding applications that remain secure and robust even when failures occur – indeed, even under attack, or during denial-of-service episodes – then the very technologies that hold the greatest promise for major advances will prove to be the Achilles Heel of a future generation of mission-critical military and public-sector enterprises.

The Spinglass project is working to overcome scalability barriers, starting with an idea that was first proposed in the context of replicated information management systems. Several early systems in this domain employed what were called "epidemic-style" or "gossip" update algorithms, whereby sites periodically compare their states and reconcile inconsistencies, using a randomized mechanism for deciding when and with whom each participant will gossip. Traditionally, such systems used gossip protocols at low speeds. Our work employs gossip at very high speeds, yielding a new generation of protocols that have an unusual style of probabilistic reliability guarantees – guarantees of scalability, performance, stability of throughput even under stress, and scalability of throughput even when a significant rate of packet loss is occurring. These properties hold even on Internet-like platforms.

[1] This work was supported in part by DARPA/AFRL-IFGA grant F30602-99-1-0532 in the DARPA-ITO FTN program, managed by Doug Maughan. Additional support was provided by Nasa JPL under the REE program. Work on the Galaxy project is supported in part by a grant from Jim Gray at Microsoft Research BARC.

Gossip protocols lend themselves to theoretical analysis, making it possible to predict their behavior with high confidence. However, the focus of our work at Cornell is mostly practical: we are using gossip, together with other more traditional mechanisms, to develop new generations of scalable communications software and network management services for a wide variety of settings.

Spinglass treats security and authentication as important considerations, and our software is designed to coexist with modern firewalls and intrusion-detection solutions. Although no distributed system can continue to operate correctly if communication is completely disabled, we will show that Spinglass protocols (notably the Astrolabe subsystem) can often ride out attacks that would cripple conventional acknowledgement based reliability tools. This suggests that Astrolabe could be a valuable adjunct to intrusion detection mechanisms, many of which can be disabled by flooding the network or using other denial-of-service mechanisms. Such attacks do cause Astrolabe to degrade, as discussed in Section 7, but it continues to report information and hence remains useful. Moreover, the technology has no servers or other single point-of-failure.

Our objective in this paper is to review the scalability problem and to summarize our approach to solving it. The need for brevity limits the technical detail here, but other publications are available for the interested reader who wishes to understand exactly how the technology can be implemented, or to see additional experimental results going beyond the ones reproduced here. Availability of our software is discussed in Section 11.

2. Scalability

The scalability of distributed protocols and systems is a major determinant of success in demanding systems. For example, consider the recent field-test of the Navy's Cooperative Engagement Capability (CEC). During the period Sept. 13-27, 2000, this system (which offers an over-the-horizon cooperative targeting capability for naval warships) was subjected to a very modest stress test. The value of this system depends upon timely identification of threats and rapid determinations concerning the ship that should respond to each threat. Threats may be incoming missiles moving at several times the speed of sound, and correct behavior implies deadlines of about a second for the communication subsystem. The subsystem had been demonstrated capable of meeting the requirements under laboratory conditions with small numbers of participating computing systems. Yet under load, when even small numbers of ships were added to the system, the underlying Data Distribution System (DDS) became unstable, either failing outright or delivering data after much more than the one-second threshold. (Defense News, October 16, 2000). In effect, the CEC failed, and did so under relatively benign conditions in which the only variable that changed was the number of participants and the rate of events being handled.

This paper focuses on scalability of distributed protocols providing some form of guaranteed reliability. Examples include the virtual synchrony protocols for reliable group communication [1, 4, 5, 16], scalable reliable multicast [7], and reliable multicast transport protocol [20]. In this section we'll try to show that the usual architecture for supporting reliability exposes mechanisms of this sort to serious scalability problems.

Traditionally, one discusses reliability by posing a problem in a setting exposed to some class of faults. Fault-tolerant protocols solving the problem can then be compared. The protocols cited above provide a multicast capability tolerant 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 the degree to which failures disrupt these properties.

Oddly, even careful performance analyses generally focus on two extreme cases: performance of the protocol under ideal conditions, when nothing goes wrong, and the disruptive impact of a failure. This paper adopts a different perspective; investigating 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 that reliable protocols would 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 (system and network size). On the contrary, reliable protocols degrade dramatically under this type of mundane stress, a phenomenon attributable to low-probability events that become both more likely and more costly as the scale of the system grows.

Here, we limit ourselves to a summary of our findings with respect to the growth rate of disruptive overheads for a number of widely used multicast protocols. Elsewhere [2], we present a more detailed analysis of the same scenarios, modeled after the work of Gray *et al.* [8], where a similar conclusion is reached with respect to database scalability. It is clear that scalability represents a widespread problem affecting a broad range of technologies and systems.

But the picture is not entirely bleak. After presenting these arguments, we shift attention to a new class of protocols based on an idea from NNTP (network-news transport protocol), the gossip-based algorithm used to propagate "news" in the Internet, and Clearinghouse, the directory replication technology developed at Xerox Parc

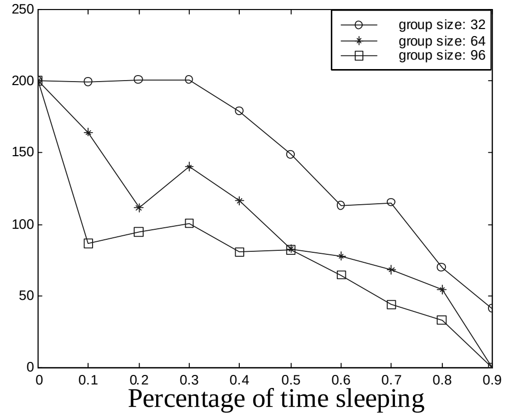

Figure 1: Sustainable throughput (multicasts/sec) drops as group size increases [3]

in the 1980's. These turn out to be scalable under the same style of analysis that predicts poor scalability for their non-gossip counterparts.

3. Scalability and Reliability

Reliable multicast comes in many flavors. This Section of the paper focuses on two well-known points within the spectrum, asking about their scalability properties. First, we look at the virtual synchrony model [1], which offers rather strong fault-tolerance and consistency guarantees to the user. These include automated tracking of group membership, reporting of membership changes to the members, fault-tolerant multicast, and various ordering properties. Communication systems using this approach include the Amoeba reliable multicast, Ensemble, i-Bus, Isis, Horus, Phoenix, Relacs, Rampart, Transis, Totem, etc. This paper focuses on the behavior of the Horus system, but reaches conclusions that would also apply to other members of 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 that will receive each message. Receivers are responsible for joining themselves to the group and must actively solicit retransmissions of data that they miss. Well known examples of multicast protocols using this approach include the reliable group multicast of the V system, RMTP (reliable multicast transport protocol), SRM (scalable reliable multicast), TIB (Teknekron Information Bus), etc. SRM has been described in the greatest detail and a simulation was available to us, so we focus on it. However, we believe our conclusions would apply to any of the protocols in this class.

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 eager to send as many messages as possible to a randomly selected, healthy receiver. We graph the effect 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 100ms 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 500ms period. The message size was 7KBytes. The data was collected on a cluster-style parallel processor, but similar results can be obtained for LANs, as reported in [17, 3].

Focusing on the 32-member case, we see that the group can sustain a throughput of 200 messages per second 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, flow control in the sender begins to limit its transmission bandwidth (our experiment employed a fixed amount of buffering space at the sender).

With this in mind, an application designer might consider adjusting the buffering parameters of the system to increase sender-side buffer capacities, and designing the application 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 sender and receiver.

Unfortunately, such a strategy is unlikely to succeed. Notice that for any fixed degree of perturbation, performance drops as a function of group size, and that 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[2] is not such an unlikely

[2] As an aside, 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.

event in a real system: random scheduling or paging delays could easily cause a process to sleep for 100ms at a time, and such behavior could also arise if the network became loaded. 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 a slow network link. Our graph suggests that a strategy focused on sender-side buffering would apparently need buffering space at least linear in the group size.

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 reliable multicast, the need to make failure detection more aggressive as a function of system size may not seem like a particularly serious concern. 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.

One could argue that with larger perturbation values and less aggressive settings of failure detection parameters, the system is just behaving according to its design. But if we make the detection mechanism extremely aggressive so that a process will be dropped for even very minor perturbations, one enters a domain in which a typical *healthy* process has a significant probability of being dropped anyhow, because of random phenomena not controllable by the designer.

Costs associated with micropartitions rise in frequency with the square of the size of the group. This is because the frequency of mistakes is at least linear[3] in the size of the group, and the cost of a membership change is also linear in the group size: a quadratic effect.

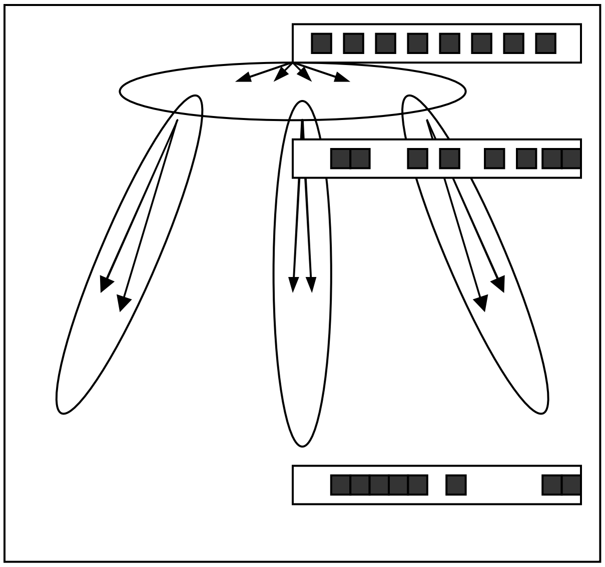

Figure 2: Message “convoys” often arise when groups are cascaded in a hierarchical manner

The phenomenon just described 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 [17].

Convoys

The obvious response to the scalability problem just presented is to structure large virtually synchronous systems hierarchically, as a tree of process groups. Unfortunately, this option is also limited by disruptive random events, albeit in a different way.

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

[3] Abstractly, the likelihood of erroneous failure detections grows as $O(n^2)$, since any member might mis-diagnose a failure of any other member, but in practice mistaken detections are not quite so frequent.

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, where data enters a hierarchical group at a steady rate (illustrated by the evenly spaced black boxes), but layer by layer, becomes increasingly bursty.

This phenomenon is familiar to the database and packet routing communities, which refer to it as a "convoy." Kalantar, studying the implications of such burstiness in hierarchical process groups, finds that each new level of group amplifies the burstiness of its data input source [12]. 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. The problem is that when messages arrive out of order, the ordering requirement forces delays but also results in the delivery of a burst of ordered messages when the gap has been filled; the next layer thus sees a bursty input that can trigger flow control mechanisms (even if the original flow would not have required flow control). The problem is particularly severe if the top-level group sends multicasts at a high data rate, since 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. 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 receipt (layer by layer, or end-to-end) to absorb the expected degree of rate variations. But this would demand a huge amount of buffering. Kalantar concludes that as traditionally implemented, hierarchically-structured process groups will be complex to manage and may perform poorly.

Request and Retransmission Storms

One might speculate that the problems seen above are specific to the virtual synchrony reliability model. However, a related scalability phenomenon has been observed by several researchers studying the SRM protocol, a "scalable reliable multicast" protocol that uses a receiver-driven recovery mechanism.

Virtual synchrony is very stringent model, providing guarantees strong enough to support replicated data wherein any copy is as good as any other. In contrast, SRM employs a best-effort model, in which 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.

IP multicast is an unreliable protocol; hence, a multicast packet might be dropped by a router (or the sender's operating system), failing to reach large numbers of receivers. 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. Group members 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.

The general belief is that because protocols like SRM have (relatively) weak reliability goals, they should scale much better than virtual synchrony, which has a very stringent reliability goal. Unfortunately, several 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. Basically, it isn't hard to drive SRM overheads through the roof.

The data shown in Figure 3 reflects this problem and is reprinted from [3]; similar findings have been reported by [13, 14]. 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 4-level tree topology, injecting 100 210-byte messages/second, 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).

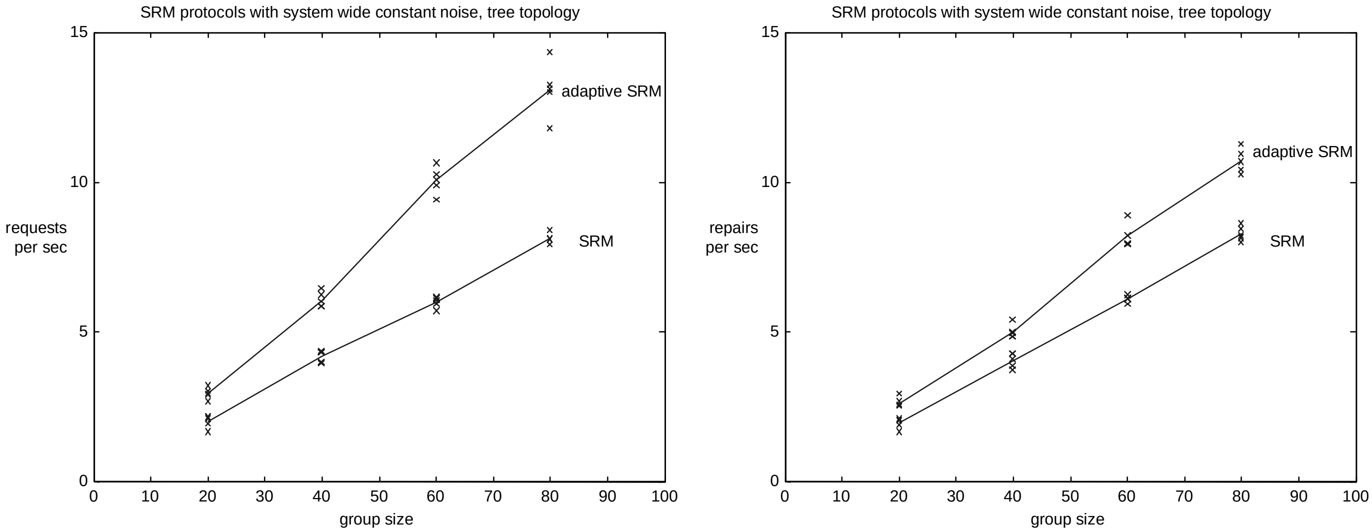

Figure 3: As the size of the group increases, a low level of background noise (0.1% in this case) can trigger high rates of requests (left) and retransmissions (right) for the SRM protocols. Most of these are duplicates. Notice that the data rate is being held constant; only the size of the group is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

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 that 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, with some probability more than one process may do so. Thus, as the network is scaled 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.

Notice that although SRM has reliability goals very remote from those of virtual synchrony, once again we encounter a mechanism with costs linear in system size and frequency growing, perhaps linearly, in system size. The costs are those associated with sending and receiving superfluous multicasts, and the frequency is basically a function of the aggregated path-lengths over which multicast messages will travel. Thus, not only is the linear growth in overhead seen in Figure 3 unsurprising, one might have speculated that it should be worse than linear. (In fact, it is possible to design experimental scenarios that provoke quadratic overhead growth, although we have not done so here).

At the start of this section, we observed that the throughput degradation experienced for the virtual synchrony protocols is not unique to that reliability model. In the case of SRM, the problems just described contribute to growth in background load seen in Figure 3 and to unstable throughput much like that graphed in Figure 1, and can overload routers to such a degree that the system-wide packet loss rate will rise sharply. Should this occur, a performance collapse is likely.

Other approaches

The issues just described are also seen in other reliability mechanisms. Elsewhere, we have extended this analysis to consider other "scalable" protocols, such as RMTP (a well known protocol that was recently standardized by the IETF). The same issue arises in publish-subscribe message bus protocols such as the one in TIB (probably the most successful multicast-like product line), and in other large-scale multicast architectures. Even forward error correction (FEC), a proactive mechanism whereby the sender introduces redundancy into the data stream so that the receiver can reconstruct lost data, degrades as a function of scale, because of the increasing likelihood that an overloaded network will drop multiple packets, overwhelming the error correction mechanism. To bound the sender's load in applications where lost data must be resent, it becomes necessary to increase the degree of redundancy in a manner proportional to the number of receivers.

So striking is this problem, and so widespread, that one might wonder how distributed computing could possibly be such a success, given the pattern of poor scalability just cited. In fact, these problems arise mostly in technologies that provide *reliability guarantees*, and few of these have ever been deployed on a really large scale, except in tightly controlled settings.

A reasonable speculation is that we are unaware of the limited scalability of these kinds of technologies, and the systems built over them, because for all the discussion, relatively few really large-scale systems have been built so far. This will soon change, but at the time of this writing, the really large-scale systems tend to be things like the Internet, and the various web-services hosted upon it. Clearly, the Internet itself scales reasonably well, provided that one doesn't care about brief episodic disruptions. Web servers are a great success, as long as occasional inability to access a server isn't a concern. Indeed, the striking pattern is not that nothing scales, but rather that *large-scale systems are prone to disruption*. To the extent that we introduce mechanisms that try to overcome this disruption, and especially when we do so at a low level, we arrive at solutions that scale poorly. In this sense, the technologies just reviewed scale poorly precisely because they attempt to operate reliably: a diabolic tradeoff indeed! Fortunately, some of the very oldest work in distributed computing points to a way out.

4. Epidemic Protocols

Not all protocols suffer the behavior seen in these reliable mechanisms. Up to now, we've focused on multicast protocols. Within this class of mechanisms, Bimodal Multicast, a protocol reported in [3], scales quite well and easily rides out the same phenomena that cause problems with these other approaches to reliability and scalability.

Bimodal multicast is a gossip-based protocol that closely resembles the old NNTP protocol (employed by network news servers), but running 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 this purpose if it is available. (Because IP multicast is often disabled in wide-area networks, Bimodal Multicast more often runs over a very lightweight unicast-based tree management and multicasting mechanism of our own design, using IP multicast if the feature is available but without depending upon it). 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[4]. At some agreed 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: "messages 5-11 and 13 from sender *s*, …" for example. 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.

This simplified description omits a number of important optimizations to the protocol. In practice, we use gossip not just for multicast reliability, but also to track system membership [19, 9]. We sometimes use unreliable multicast with a regional TTL value instead of unicast, notably 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 done primarily to nearby processes over low-latency links and rarely to remote processes, over costly links that may share individual routers [22]. The protocol switches between gossip pull and gossip push, using the former for "young" messages and the latter for "old" ones. Finally, we don't actually buffer every message 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.

Bimodal Multicast has a number of beneficial properties. The protocol imposes constant loads on participants: during each gossip round, a process sends a single message, receives (with high probability) a single message, and may be asked to retransmit at most a bounded amount of data. Given a small amount of

[4] In practice, we bias this list to favor nearby processes – those accessible over low-latency links. We also need to tunnel through firewalls, and hence communication to genuinely remote processes is handled in a slightly different manner. However, these details go beyond the scope of the current paper.

information about network topologies, loads on communication links can also be kept constant, irrespective of the system size. The mechanisms supporting the protocol can be implemented as an event-driven state machine that never blocks and requires just a small amount of buffering, making it inexpensive to run.

Most 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 (reliability is increased by increasing the length of time before a message is garbage collected, but this also causes buffering and I/O costs to rise). The protocol gives very steady data delivery rates with predictable, low, variability in throughput. For real-time applications, this can be extremely useful. And the protocol imposes constant loads on links and routers (if configured correctly), which avoids network overload as a system scales up. All of these characteristics 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 won't 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 needed 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" network statistics. In contrast, protocols such as SRM or RMTP often include round-trip estimates of the mean latency or mean throughput between nodes. Such estimates are problematic in the Internet where many statistical distributions are heavy-tailed and hence have ill-defined means and very large variances.

Although Bimodal Multicast has a very different reliability model than does virtual synchrony (the model is more similar to the best-effort guarantees of SRM or RMTP), it is possible to superimpose stronger reliability models over the infrastructure offered by the Bimodal protocol. In particular, we have begun to work with an implementation of virtual synchrony that operates over Bimodal Multicast. The resulting protocol represents something of a tradeoff. For small groups, its performance is not as good as in a more traditional virtual synchrony implementation. In larger groups, however, we've seen that virtual synchrony becomes unstable and eventually breaks down. Virtual synchrony over Bimodal Multicast scales far better, remaining robust and steady with apparently constant costs independent of the size of the group. This observation motivates the discussion that follows.

5. Randomized Limits to Scale

We can generalize from the phenomena enumerated above. Distilling these down to their simplest form, and elaborating slightly:

- ❑ With the exception of the gossip protocols, each of these reliability models involves a costly fault-recovery mechanism intended for infrequent use:
  - ▪ Virtual synchrony employs flow control, failure detection and membership-change protocols; when incorrectly triggered, the cost is 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. As the group size grows, however, the average multicast path length increases; hence the risk of a multi-packet loss rises. The sender will see increasingly many retransmission requests (consuming a scarce resource), or the redundancy of the stream itself must be increased (resulting in a bandwidth degradation and a system-wide impact).
- ❑ Again with the exception of the gossip protocols, the mechanisms we've 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. Such bursts have a global effect, in the sense that processes in the application may become overwhelmed by a high volume of data, or may run "dry" for so long that the user is disrupted (as might happen in a multicast media delivery setting). At best, bursty traffic may trigger flow control at inappropriate times, reducing overall throughput. Perhaps more serious, bursty traffic increases the likelihood that an overloaded router may drop large numbers of packets, requiring a costly recovery. However, whereas this phenomenon has actually been observed in hierarchical implementations of virtual synchrony, we have not seen anything comparable in our work with other protocols, or with virtual synchrony running over Bimodal Multicast.
- ❑ Many protocols (SRM, RMTP) depend upon 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 seem increasingly expensive. Periodic reconfigurations, the obvious remedy, introduce a disruptive system-wide cost.

In contrast, the gossip mechanisms used in NNTP, the Xerox Clearinghouse system, and the Bimodal Multicast protocol appear to scale without these kinds of problems. Throughput is stable (at least, if measured over sufficiently long periods of time – gossip protocols can be rather *unstable* if metered on a short time scale). Overheads are flat and predictable, and can be balanced with information about network topology, so that links and routers won't become overloaded. And, the levels of reliability achieved are very high – indeed, potentially as high as those of the protocols purporting to offer stronger guarantees, if one considers the possibility that such protocols sometimes reconfigure themselves, incorrectly excluding a process as "faulty" when it may actually merely be the victim of bad luck.

Is there a general insight that relates these observations?

- ❑ Many reliable multicast protocols depend upon assumptions about the usual model of operation. These have a small probability of being violated. As a system scales 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. The problems cited arise within the lowest layers, although they then propagate to higher layers triggering large-scale disruptions of performance.

More broadly, we argue that these are all consequences of a protocol stack architecture in which *the lowest layers of the stack provide reliability*, 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.

In contrast, gossip protocols can be understood as using an inverted protocol stack. In this approach, recovery occurs at the lowest level and operates with probabilistic behavior and guarantees. Higher-level mechanisms do their best to impose end-to-end properties on the lower level properties, but typically do so without introducing extra communication for the purpose of repairing gaps in the message sequence. In effect, we adopt the view that the lower level mechanisms have already done what can be done to achieve data consistency among the participants, and we must content ourselves with the outcome. These protocols, then, offer probabilistic guarantees.

Probabilistic guarantees may sound like a contradiction in terms, because common sense suggests that anything but an absolute reliability guarantee would be the equivalent of no reliability at all. Our work suggests that this is not at all the case. First, it is possible to design mechanisms that have stronger guarantees, such as virtual synchrony, and yet reside in an end-to-end manner over the basic network architecture afforded by our gossip infrastructure. As just noted, when virtual synchrony is implemented this way, it scales relatively well; multicast throughput remains steady even under the kinds of stress seen in Figure 1, although there may be long delays when process group views change if the system membership has become very large. But we are also finding ways of embedding probabilistic guarantees directly into useful tools that applications might find valuable in their own terms, without trying to superimpose some stronger (arguably, less natural) reliability abstraction over the basic properties of the protocol.

Thus, while layering virtual synchrony over Bimodal Multicast may be useful, many data dissemination systems can operate directly over Bimodal Multicast. The properties of the protocol are well matched to the needs of systems that do require a degree of reliability, but can overcome *bounded* rates of error. The term bounded is the key here; unlike a traditional unreliable network mechanism that can behave arbitrarily badly if luck turns against the user, Bimodal Multicast overcomes all but the most severe outages and behaves in a predictable manner at all times.

Similarly, the Section that follows describes Astrolabe, a technology that offers probabilistic guarantees for data managed in a scalable table. Tables are a common and widely supported programming abstraction (database relations, spreadsheets, etc); hence the idea of a table made up of data drawn from various parts of the network is both natural and easy to work with. Astrolabe provides this model with probabilistic reliability guarantees, and has the same scalability and robustness properties seen for Bimodal Multicast.

6. Four Probabilistic Tools

Earlier we noted that the focus of the Spinglass project is on practical software tools that can really be used. In this section and the next, we first review the tools we are currently developing, and then describe some applications that we view as especially promising early targets for the technology.

Bimodal Multicast

As described above, Bimodal Multicast is a one-to-many (or several-to-many) communications mechanism that achieves tunable, probabilistically reliable data delivery. The probability of successful outcomes, where all operational processes receive every multicast, can be made arbitrarily high by adjusting the parameters governing the frequency with which participants gossip and the length of time that they buffer copies of received multicasts. In ongoing work at Cornell University, we are exploring issues such as operation in wide-area networks with firewalls surrounding secured enclaves, using formal tools to characterize the conditions that a network must have in order to run this protocol, and understanding the kinds of network problems that Bimodal Multicast is capable of overcoming.

Astrolabe

Astrolabe [18] is a system built using the same gossip mechanisms employed in Bimodal Multicast, but in support of a completely different data model. The basic idea of Astrolabe is to support a distributed shared memory in the form of a hierarchical table. The leaves of the hierarchy are regional tables in which there is one row per participating computer or application program, and where the columns contain application-defined data (this could be something small, like an indication of the security level of the machine or the version number of a program running on it or a load, or something large, like an XML object describing a database or web page). Within a region, all participants can see the entire regional table but each can only update its own row.

Higher levels of the hierarchy are formed using what we call a summary function. A summary function is a computation on a column of a regional table that reduces the contents of that column to a single value. For example, *minimum* could be used as a summary function for a column-reporting load. The contents of a higher-level table will be one row for each of its child regions, with columns defined according to the summary function. While a region only has direct access to its own regional table, all regions can access all of the ancestor tables in the hierarchy. Normally, an ancestor table would list properties of a whole region, like its average communication load, together with attributes of the region, like the IP address of a machine to contact for a given service (incoming mail, database updates, etc). The intention is that the application designer would customize the summary functions.

Astrolabe is a useful technology for scalable system management and resource discovery. In modern computing systems, simply knowing where data can be found or knowing versions of software and configuration information for other machines is a critical and yet poorly supported functionality. Astrolabe automates this job and does so in a manner that is scalable and has predictable delays (they grow slowly with system size), constant overheads, and remarkable stability. Basically, one arranges for summary functions that let the user identify the regions where instances of the resource can be found (for example, "regions where some computer controls a satellite capable of imaging such-and-such a location), and then where a match is found, drills in by expanding the regional table and scanning its individual elements (i.e. to find the machine actually controlling the desired satellite).

Astrolabe's table model is a good match with the new generation of component architectures for PCs and portable devices. Microsoft's automation architecture, used on many devices of this sort, has comprehensive support for table-structured objects through the OLE-DB interfaces (an evolution of the older ODBC interfaces). Astrolabe supports these interfaces, which means that on a PC, we can actually support drag-and-drop access to Astrolabe tables. Such a mechanism would offer stable, fault-tolerant distributed computing without the need for distributed programming, much as a spreadsheet user can work with other spreadsheet users and yet may never need to learn to write any code.

Gravitational Gossip

This variation on the Bimodal Multicast protocol is designed to support large numbers of subgroups within a single network [11]. Subgroups are a traditionally important problem in group communication systems, particularly when supporting publish-subscribe styles of communication, where large numbers of communication groups can arise. With this new protocol, we are able to superimpose large numbers of subgroups on a large Bimodal Multicast group, and can arrange that each group member will receive just the data it desires plus a constant overhead. Moreover (this is the "gravitational" aspect) members of a subgroup can specify a quality rating. A member that wants to receive 100% of the data in a group can do so, paying the full cost for all multicasts in the group. However, if a member only needs part of the data – say, 50% of the sensor readings or 20% of them – it can adjust its rating to correspond to its need. The load associated with the protocol is reduced accordingly. Notice that we are deliberately reducing the reliability of the protocol to cut the load seen by processes with small rating values.

We like to visualize this protocol as emulating a gravitational well. In practice ratings can have any values desired, but these include values that give behavior like that of a gravity fielding this case, messages multicast within the "floor" of the well and flow at full speed to other processes in the floor. With some probability, these messages also ride up the "walls" of the well, but the steeper the wall, the less likely this is to occur, and a message that does ride up the wall is very likely to fall back towards the base. Processes residing on the wall thus see less load and receive just a percentage of the data items.

Obviously, gravitational gossip is only useful in settings where it is meaningful to take actions based on a randomly selected subset of sensor values. However, as discussed below, we know of a number of such applications.

Anonymous Gossip

This direction within our project applies gossip communication to mobile wireless devices [4]. We have a number of applications in mind, but started by looking at wireless multicast in so-called ad-hoc networks, which are common in military applications. Anonymous gossip is a technique for using gossip communication to improve the quality of existing ad-hoc multicast protocols. The idea is to take a multicast protocol (we've considered several, but worked most closely with AODV and the multicast layered over it, MAODV) and then to superimpose a gossip repair mechanism similar to the one in Bimodal Multicast.

Where Anonymous Gossip departs from Bimodal Multicast is that in a mobile wireless setting, little information is available about the identity of peers, since these can change rapidly. For example, a platoon of soldiers may fan out on a hillside, so that the network is always "fully connected" and yet the connectivity of any particular soldier's computer varies widely. The challenge is that in such a network, one has no idea with whom to gossip. Anonymous Gossip solves this problem by sending gossip messages that travel some distance over a randomly chosen path in the ad-hoc network. The eventual receiver replies to the sender.

In [4] we report on an experimental analysis of this technique. We find that it improves the reliability of MAODV, reducing loss rates by as much as two orders of magnitude and also yielding better throughput with lower overheads and lower jitter. We are now extending the basic protocol by looking at other metrics and at the use of gossip to maintain the basic routing infrastructure itself. At the same time, we are starting to look at application-level issues that arise in developing mobile software to run over a gossip infrastructure.

7. Security Issues

Gossip mechanisms raise interesting issues of authentication and security. On the one hand, we face challenges in securing our new tools against intrusion and disruption; on the other, one can view our system as a potentially valuable tool for use in intrusion detection systems and other aspects of security architectures.

With respect to the former question, our challenge is that gossip employs a relaying mechanism. We need to avoid the possibility that a process responsible for relaying a message might undetectably change its contents. Accordingly, our implementations include session authentication mechanisms (access control lists) as well as digital signatures used to validate the correctness of gossiped information. If process *p* sends a multicast or updates a row in Astrolabe, the signature of process *p* for that data travels with the data, so that while a faulty process might corrupt the information, a healthy process will always be able to detect and ignore damaged messages. Astrolabe's summary functions are also secured using signatures; this ensures that new summary functions can only be introduced by users with appropriate administrative permissions.

The power of gossip is that information reaches destinations by following a diversity of paths. In effect, every process is a potential source of data for every other process, and over time the number of possible routes by which information from process *p* might travel to process *q* rises exponentially. For example, suppose that process *p* has detected an event of interest. After *t* time units, $O(2^t)$ processes will have know of that event.

When a system comes under attack, an adversary may manage to corrupt a few messages or disrupt a region of the network. Yet, if any connectivity remains at all[5], the gossip exchange of data will eventually prevail, and data stored within Astrolabe will reach all sites in the system. Thus, our protocols are not just secure in the sense of using digital signatures to authenticate users and to authorize their actions, but they are also relatively

[5] Flooding attacks that dramatically reduce the capacity of the network will cause Astrolabe to exhibit degraded behavior (reporting updates after unusually long delays, and potentially dropping some updates in favor of newer ones). However, such an attack is unlikely to shut Astrolabe down completely. Notice further that Astrolabe does not increase its rate of communication when error rates rise. In contrast, conventional protocols often "melt down" when a flooding attack occurs, sending retransmission requests and extra copies of messages in a futile attempt to overcome errors, which instead merely exacerbates the overload.

tolerant of denial of service attacks. On the down side, when a system comes under attack, one can easily imagine that the rate of change of information monitored using Astrolabe could exceed the speed at which that information can be propagated; in this case, new sensor values begin to overwrite old ones and applications may “miss” some values. This will also occur if the network becomes heavily loaded. Moreover, Astrolabe limits itself to communicating information; nothing about our work provides a guarantee that the information is of high quality, nor do we know how effective Astrolabe’s summary functions will be as a tool for reducing the volume of data that an application monitoring this data might need to process. We believe that Astrolabe is best seen as a potentially valuable tool for the developer of distributed security mechanisms, but one that also brings new issues and complexities.

For example, consider the problem of intrusion detection. Traditionally, this is treated as a monolithic issue. Our work suggests that it might be useful to separate such systems into three parts: event collection, data dissemination, and intrusion detection. We’re not specialists in event collection or in the actual detection problem itself. But Astrolabe, and Bimodal Multicast, might represent useful technologies for data dissemination, and they have properties that would seem to represent advances over those of more conventional solutions.

Similarly, consider the issue of system “control” that arises when a problem is detected. Here, the key to recovery involves orchestrating a cooperative and consistent response while the attack is still underway. Bimodal Multicast and Astrolabe are both options for this purpose, depending on the urgency of the reaction. Both protocols are secure and both will overcome even an active attack, rapidly and consistently notifying participants, which can then switch to a backup network, rekey, or take other actions to repel the aggression.

Yet gossip technologies also introduce new vulnerabilities. The mere act of placing a software system on large numbers of nodes potentially introduces a new common point for attacking those nodes. We’ve noted that Astrolabe uses executable summary functions to compute domain/value data for higher levels of the information hierarchy, and one could imagine situations in which corrupted functions might damage or disable participating nodes. To minimize such risks, Astrolabe employs a very limited functional programming language for these functions, and requires that all introduced information be appropriately signed, but additional hands-on experience with large applications will be needed before we fully understand the policy and administrative implications of treating extremely large systems as manageable, instrumented entities.

8. Applications

We have examined a number of application areas to understand the right roles for our tools in support of emerging scalable computing systems. Here, we briefly describe three representative areas to illustrate some matches between our technology and real-world problems. At Cornell, each of these areas is being examined in much greater detail.

Joint Battlespace Infosphere

Our team includes members of a new AFRL/IT Information Assurance Institute, which was put in place to foster collaboration and cooperation between the Air Force and Cornell on topics arising our of demanding Air Force computing systems. A focus for this work is the Air Force Science Advisory Board report recommending the development of the JBI.

The JBI, like its sibling programs in the Army and Navy, is intended to offer a shared computing platform that links information produced by a diversity of "publishers", in a secure and scalable manner, with a diversity of "subscribers." The intent of the architecture is to create a vast web-like infrastructure that permits anytime, anywhere access to mission-critical data, but under the assumption that unlike the web, the data of interest will be evolving rapidly in real-time.

There is no question that a JBI capability could be an asset of incalculable value if we can successfully develop it and make use of it during military engagements. However, success implies that the platform must be sufficiently scalable and stable to work well even when large numbers of consumers tap into the JBI simultaneously, even when components fail, become overloaded, or come under attack. Existing complex systems scale poorly, becoming fragile in all of these respects.

Cornell is exploring possible applications of Spinglass to the JBI infrastructure. Our goal is to show that an infrastructure such as the JBI can be designed to be intrinsically robust, so that as we scale it up, it continues to work well and has predictable properties. In contrast, were one to build a JBI with off-the-shelf tools, one must fear that it would become unstable as we scale it up, subject to unexpected overloads, degraded service, and perhaps outright breakdown. Use of the Spinglass technology in applications like these will no doubt reveal new kinds of challenges, and we believe that a substantial research effort will be required before we fully understand the capabilities, and limits, of our new technologies. However, the apparent match between Spinglass capabilities and the JBI encourages us to believe that real progress can be made before such limitations are encountered.

Cluster Management with Ising

Modern computing is increasingly organized around powerful servers, such as databases, web servers, and various kinds of enterprise network servers [21]. The inexpensive way to accommodate large numbers of clients is to somehow run these servers on clusters with large numbers of cheap but powerful component machines. Yet managing such clusters, especially in a large network with multiple clusters, can be a tough challenge. As a client of a wide-area cluster farm, even knowing which server to connect to is hard, particularly if that server might crash or migrate.

We have developed a project, Galaxy, which is exploring architectures for supporting very large clusters. Within this effort, the Ising technology is a cluster-embedding of Spinglass protocols and solutions tuned for use in PC clusters. The basic ideas are the same, but because we know how the Ising versions of these protocols will be used, we can undertake a type of context-specific tuning that would not be appropriate in a tool intended for very general settings.

Galaxy virtualizes the cluster: each application sees what appears to be a dedicated platform, on which it superimposes its own resource abstractions and policies, such as job placement, load balancing, etc. The system is flexible about how such problems can be solved, offering standard mechanisms that the application can override by supplying new modules. For example, many applications will want to provide specialized load balancing mechanisms.

Galaxy treats several levels of scale:

- A *farm* is a geographically distributed set of clusters on which a single application is hosted, managed from a centralized location. In the early stages of our effort, we are focused on farms with no more than a small number of clusters, but large farms will eventually be considered as well. The JBI, for example, will probably be a farm in this sense.
- An individual cluster is a reliable array of computers. We consider two cases. A RACS is a reliable array of cloned servers: machines that have identical content and can behave in identical ways. Queries against a rarely changing database system can be directed to a RACS cluster; since any node can handle any request, this offers inexpensive scalability.
- A RAPS architecture is one in which a system, such as a database, is partitioned. For example, perhaps the database is divided into ten equal sized portions, and each query can be routed according to the portion of the data it will access.
- Finally, we support *packed* architectures. Database systems can exploit low levels of parallelism although large parallel databases run into bottlenecks. A server *pack* is a small set of servers on which a database operates in this sort of packed mode.

Galaxy provides mechanisms in support of cluster management and cluster-aware application development, with the goal of understanding the right roles for Spinglass protocols in such a setting, and the right styles of application design where scalability is a requirement. For example, the designers of the JBI will need to know that they can implement a number of scalable communication services, deploy them in a manageable manner onto a scalable hardware platform, and simply add more hardware if the application is successful and load rises. Our hope is that Spinglass, residing in the Ising subsystem of Galaxy, will respond to this need both by offering a way to manage the cluster and also by demonstrating the best ways of exploiting a cluster.

Electric Power Grid

Finally, we are collaborating with a consortium to work on problems arising from the restructuring of the national electric power grid. As has become painfully evident to inhabitants of California, legislatures are increasingly mandating a restructuring of electric power systems to reduce the traditional monopoly structures that prevail in this industry. With competition come both the economic issues that are so much in the news, as well as new technical challenges. Our effort involves matching the Spinglass technology to the needs of these emerging control and management systems.

As an example, the Gravitational Gossip protocol, although useful in many settings, arose as a response to a pattern of communication actually seen in restructured electric power control and management systems. Prior to restructuring, these systems adapted to changing loads in a regional manner. Basically, within each region, it sufficed to monitor line frequency; power laws dictate that frequency falls when load exceeds production and rises if power production exceeds load. Thus, without any form of networked communication, all generators can see when the regional need for power is rising or falling and can adapt in parallel (for example, closing a gate in a dam or reducing the supply of coal to a furnace).

As we restructure and move to a competitive architecture, it may be that Ohio Power and Light will provide the power consumed by Corning Glass in New York under a contract. Now, when the New York region senses an over or underload, it is no longer appropriate to simply adjust production. Suddenly, we need an elaborate communications architecture whereby Ohio can monitor Corning. Moreover, for reasons of grid protection many other devices may need to track the status of the load-following contract between Ohio and Corning, and this data is also useful as an input to the pricing models used in the free market for electric power.

The need for high quality data, however, varies within this collection of players. Very high-resolution information about a contract is needed by the producer and consumer and by the protective relays along the major path between them. Less detailed information is needed by other relays in the region, and even less detail is needed by the various market pricing programs. This is good news, because in a world where every program wants equally good data, our communication costs would otherwise rise as $O(nm)$ where n is the number of processes in the application and m is the number of contracts.

Gravitational Gossip lets us respond to the need in a way that lets each process pay for just the data it will be using. In this way, we can architect the system to load the network in a predictable way, so that communication overload cannot arise even in a worst case scenario (presumably, during a major storm), ensuring stability and speedy response no matter what might transpire.

9. Summary and Conclusions

We briefly examined a number of multicast scalability problems and argued that they can be understood as the outcome of a battle between random phenomena and deterministic properties. These include throughput instability, flow control problems, convoys seen in ordered multicast delivery protocols, and high rates of duplicated retransmission requests or unneeded retransmitted data packets in protocols using receiver-driven reliability. We traced these problems to a form of complexity argument, and suggested that many protocol architectures degrade as $O(n^2)$ or worse in the size of the system. It is especially interesting to realize that this phenomenon may stem in part from the traditional OSI stack, which enforces reliability and performs flow control low in the network.

The alternative we've proposed can be understood as an inversion of the ISO stack, insofar as lower layers are gossip-based and have probabilistic properties, while upper levels introduce stronger properties (such as virtual synchrony, if desired). The most natural presentation of our architecture is in terms of a scalable reliable communication protocol, the Bimodal Multicast, which has reliability and stability properties that can often be used directly (avoiding the need for stronger, more costly reliability guarantees such as virtual synchrony). In effect, we see our architecture as an end-to-end response to a situation in which the traditional manner of building multicast protocols violated the end-to-end methodology. Experiments confirm that this approach yields substantial immunity to the scalability limits just cited.

The same idea can be used in other settings. Cornell's work on Astrolabe, Gravitational Gossip and Anonymous Gossip illustrate a number of other presentations for these kinds of communication protocols, and our work on security issues and applications confirms that these mechanisms can be effective if used appropriately.

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 repair 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 here to stay.

10. Availability

Our research effort at Cornell makes prototype versions of the software developed under DARPA support available to the public, at no fee. These research prototypes can be downloaded by visiting our web pages at http://www.cs.cornell.edu/Info/Projects/Spinglass. However, Cornell is not able to provide support. Hardened product-quality versions of the technologies described herein are available from Reliable Network Solutions, Inc.

11. Acknowledgements

The authors gratefully acknowledge the many graduate students who have contributed to the work reported here, including Mark Hayden, Yaron Minsky, Zhen Xiao, Xiaoming Liu, Indranil Gupta, Kate Jenkins, Ken Hopkinson, Ranveer Chandra, Vanugopalen Ramasubramanian, Adrian Bozdog, Rimon Barr, Ben Atkin and Tibor Janosi. We also wish to thank our colleagues Al Demers, Fred Schneider, Johannes Gehrke and Jon Kleinberg for stimulating dialog about gossip protocols and for their many useful comments and suggestions. Jim Gray and the members of the NT Cluster group at Microsoft have been extremely helpful

in posing questions and making suggestions about the Galaxy project.

12. References

[1] Birman, K.P. *Building Secure and Reliable Network Applications.* Manning Publications and Prentice Hall, 1997.

[2] Birman, K., Gupta, I, Van Renesse, R. Fighting Fire with Fire: Using Randomized Gossip to Overcome Probabilistic Limits to Scalability. In preparation, expected completion March 2001.

[3] Birman, K., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., Minsky, Y. Bimodal Multicast. Cornell University Dept. of Computer Science Technical Report, (Feb. 1998). Submitted to ACM TOCS.

[4] Chandra, R., Ramasubramanian, V. and Birman, K.P. Anonymous Gossip: A Technique for Improving the Reliability of Ad-Hoc Multicast Protocols. International Conference on Distributed Computing Systems, Phoenix Arizona (April 2001).

[5] Cheriton, D and Skeen, D. Understanding the Limitations of Causal and Totally Ordered Communications. *Proc 13th SOSP*, Ashville, N.C. Dec. 1993 (44-57).

[6] Demers, A. *et. al.* Epidemic Algorithms for Replicated Data Management. *Proceedings of the 6th Symposium on Principles of Distributed Computing (PODC),* Vancouver, Aug. 1987, 1-12.

[7] Floyd, S., Jacobson, V., McCanne, S. Liu, C., and Zhang, L.. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proc SIGCOMM `95,* Aug. 1995, Cambridge MA.

[8] J. Gray, P. Helland, P. O'Neil and D. Shasha. The dangers of replication and a solution. *Proceedings 1996 SIGMOD Conference*, June 1996.

[9] Richard Golding and Kim Taylor. Group Membership in the Epidemic Style. Technical report UCSC-CRL-92-13, University of California at Santa Cruz, May 1992.

[10] Guo, K. *Scalable Message Stability Detection Protocols.* Ph.D. dissertation, Cornell University, 1998.

[11] Jenkins, K., Hopkinson, K., and Birman, K. A Gossip Protocol for Subgroup Multicast. *Submitted to ICDCS Workshop on Reliable Group Communication.* Nov. 2000

[12] Kalantar, M and Birman, K. Causally Ordered Multicast: the Conservative Approach. *Proc. ICDCS 1999.* Austin, June 1999.

[13] Liu, C. *Error Recovery in Scalable Reliable Multicast* (Ph.D. dissertation), University of Southern California, Dec 1997

[14] Lucas, M.. *Efficient Data Distribution in Large-Scale Multicast Networks* (Ph.D. dissertation), Dept. of Computer Science, University of Virginia, May 1998

[15] Marzullo, K., Cooper, R., Wood, M., Birman, K. Tools for Distributed Application Management. *IEEE Computer,* 24:8 Aug. 1991. 42-51.

[16] Rebuttals to [5] appearing in Operating Systems Review, January 1994.

[17] Piantoni, R. and Stancescu, C.. Implementing the Swiss Exchange Trading System. FTCS 27 (Seattle, WA), June 1997, 309-313.

[18] Van Renesse, R. Astrolabe: A Scalable Resource Location Service. *Submitted to DISCEX-01.* Dec. 2000.

[19] Van Renesse, R., Minsky Y, and Hayden, M. Gossip-Based Failure Detection Service. In Proc. of Middleware '98. England.

[20] Sanjoy Paul, Sabnani, K., Lin, K. and Bhattacharyya, S. "Reliable Multicast Transport Protocol (RMTP)", IEEE Journal on Selected Areas in Communications, special issue on Network Support for Multipoint Communication, April 97, Vol 15, No. 3

[21] Vogels, W. and Dumitriu, D.M., "An Overview of the Galaxy Management Framework for Scalable Enterprise Cluster Computing", in the Proceedings of the IEEE International Conference on Cluster Computing: Cluster-2000, Chemnitz, Germany, December 2000.

[22] Xiao, Zhen and Birman, Ken. A Randomized Error Recovery Algorithm for Reliable Multicast. *Accepted for publication, IEEE INFOCOM `01. April 2001.*