SWIM: *S*calable *W*eakly-consistent *I*nfection-style Process Group *M*embership Protocol

Abhinandan Das, Indranil Gupta, Ashish Motivala[*]
Dept. of Computer Science, Cornell University
Ithaca NY 14853 USA
{asdas,gupta,ashish}@cs.cornell.edu

Abstract

Several distributed peer-to-peer applications require weakly-consistent knowledge of process group membership information at all participating processes. SWIM is a generic software module that offers this service for large-scale process groups. The SWIM effort is motivated by the unscalability of traditional heart-beating protocols, which either impose network loads that grow quadratically with group size, or compromise response times or false positive frequency w.r.t. detecting process crashes. This paper reports on the design, implementation and performance of the SWIM sub-system on a large cluster of commodity PCs.

Unlike traditional heartbeating protocols, SWIM separates the failure detection and membership update dissemination functionalities of the membership protocol. Processes are monitored through an efficient peer-to-peer periodic randomized probing protocol. Both the expected time to first detection of each process failure, and the expected message load per member, do not vary with group size. Information about membership changes, such as process joins, drop-outs and failures, is propagated via piggybacking on ping messages and acknowledgments. This results in a robust and fast infection style (also epidemic or gossip-style) of dissemination.

The rate of false failure detections in the SWIM system is reduced by modifying the protocol to allow group members to suspect a process before declaring it as failed - this allows the system to discover and rectify false failure detections. Finally, the protocol guarantees a deterministic time bound to detect failures.

Experimental results from the SWIM prototype are presented. We discuss the extensibility of the design to a WAN-wide scale.

1. Introduction

As you swim lazily through the milieu,
The secrets of the world will infect you.

Several large-scale peer-to-peer distributed process groups running over the Internet rely on a distributed membership maintenance sub-system. Examples of existing middleware systems that utilize a membership protocol include reliable multicast [3, 11], and epidemic-style information dissemination [4, 8, 13]. These protocols in turn find use in applications such as distributed databases that need to reconcile recent disconnected updates [14], publish-subscribe systems, and large-scale peer-to-peer systems[15]. The performance of other emerging applications such as large-scale cooperative gaming, and other collaborative distributed applications, depends critically on the reliability and scalability of the membership maintenance protocol used within.

Briefly, a membership protocol provides each process ("member") of the group with a locally-maintained list of other non-faulty processes in the group. The protocol ensures that the membership list is updated with changes resulting from new members joining the group, or dropping out (either voluntarily or through a failure). The membership list is made available to the application either directly in its address space, or through a callback interface or an API. The application is free to use the contents of the list as required, e.g. gossip-based dissemination protocols would use the list to periodically pick target members for gossip.

The reliability and scalability of a membership sub-system can be measured via several performance metrics. Membership changes have to be propagated within the group quickly after their occurrence. The asynchrony and unreliability of the underlying network can cause messages to be lost, leading to false detection of process failures, since a process that is losing messages is indistinguishable from one that has failed [10]. This rate of false positives has to be low. Finally, the protocol needs to be peer-to-peer (not rely on a central server), and impose low message and computation loads on the network and processes.

[*] Author last names are in alphabetical order. The authors were supported in part by NSF CISE grant 9703470, in part by DARPA/AFRL-IFGA grant F30602-99-1-0532, and in part by a grant under NASA's REE program, administered by JPL.

Membership protocols have been difficult to scale in groups with beyond a few dozen processes [11, 16], thus affecting the performance of applications using them. As reported in [16], the main symptoms of bad performance at these group sizes is an increase in either the rate of false failure detections of processes, or the time to detect a failure. [12] identifies the quadratic increase in the message load imposed by such membership protocols as another symptom of the unscalability of traditional protocols for membership maintenance. An example of an application that relies heavily on the membership sub-system is the class of virtually synchronous multicast protocols [3]. Traditional implementations of this specification suffer a drastic reduction in performance, and partitioning, at beyond a few dozen members [11].

This paper presents our effort in the SWIM project to implement a membership sub-system that provides stable failure detection time, stable rate of false positives and low message load per group member, thus allowing distributed applications that use it to scale well. We focus on a weaker variant of group membership, where membership lists at different members need not be consistent across the group at the same (causal) point in time. Stronger guarantees could be provided by augmenting the membership sub-system, e.g. a virtually-synchronous style membership can be provided through a sequencer process that checkpoints the membership list periodically. However, unlike the weakly consistent problem, strongly consistent specifications might have fundamental scalability limitations[1].

The design of a distributed membership algorithm[2] has traditionally been approached through the technique of *heartbeating*. Each process periodically sends out an incremented heartbeat counter to the outside world. Another process is detected as failed when a heartbeat is not received from it for some time. However, actual implementations of heartbeating suffer from scalability limitations. Sending all heartbeats to a central server leads to hot-spot creation. Sending heartbeats to all members (through either network multicast, or gossiping [16]) leads to a message load on the network and group that grows quadratically with the group size. Heartbeating along a logical ring [9] suffers from unpredictability of failure detection time when there are multiple failures. Unfortunately, as the group size rises, so does the likelihood of simultaneous multiple failures.

An extended discussion of reasons behind the inherent unscalability of heartbeat-based membership maintenance mechanisms can be found in [12]. This paper also proposed a randomized distributed failure detector protocol based on members randomly probing each other instead of heartbeating [3]. Mathematical analysis showed that as the group size is scaled up, the protocol's properties of (expected) failure detection time, rate of false positives, and message load per member, are all independent of the group size. This is an improvement over all-to-all heartbeating based protocols that have a linear variation (with group size) of either the detection time for failures or the network bandwidth usage at each member (or an increase in the false positive rate).

Our work in this article is motivated by a realization from the work of [12] that the unscalability of the popular class of all-to-all heartbeating protocols arises from the implicit decision therein to fuse the two principal functions of the membership problem specification: 1) Membership update *Dissemination*: propagating membership updates arising from processes joining, leaving or failing, and 2) *Failure detection:* detecting failures of existing members. The overhead of multicasting heartbeats is eliminated by designing an efficient non-multicast based failure detector, and using the dissemination component only when a membership change occurs. The Membership Dissemination component can be implemented through either hardware multicast or in infection-style.

While [12] presented a failure detection protocol and analyzed it theoretically, our work in the current paper looks at incorporating the Membership Dissemination component in to build a working membership sub-system. In addition, the resulting protocol is augmented by mechanisms that reduce the rate of false positives and give stronger deterministic guarantees on failure detection times at individual processes.

Our system, called SWIM, provides a membership substrate that:

(1) imposes a constant message load per group member;

(2) detects a process failure in an (expected) constant time at *some* non-faulty process in the group;

(3) provides a deterministic bound (as a function of group size) on the local time that a non-faulty process takes to detect failure of another process;

(4) propagates membership updates, including information about failures, in *infection-style* (also gossip-style or epidemic-style [2, 8]); the dissemination latency in the group grows slowly (logarithmically) with the number of members;

(5) provides a mechanism to reduce the rate of false positives by "suspecting" a process before "declaring" it as failed within the group.

While (1) and (2) are properties of the failure detection protocol of [12], (3)-(5) represent our subsequent work in the current paper. Experimental results of a prototype implementation of SWIM running on a PC cluster are discussed. The SWIM protocol can also be extended to work

[1] Discussion of this issue is outside the scope of this paper. The reader is referred to [11].

[2] A "weakly-consistent" adjective is implicitly assumed, and dropped henceforth.

[3] In a sense, the protocol monitors the status of members, randomly, instead of using heartbeating.

over a wide area network (WAN) or virtual private network (VPN), and we touch on this briefly in Section 6.

The rest of the paper is organized as follows. Section 2 summarizes previous work in this area, and the basics of scalable failure detection protocols from [12]. Section 3 describes the basic SWIM protocol, and Section 4 the improvements to the protocol. Experimental results from a prototype implementation are presented in Section 5. We conclude in Section 6.

2. Previous Work

In traditional distributed all-to-all heartbeating failure detection algorithms, every group member periodically transmits a “heartbeat” message (with an incremented counter) to all other group members. A member M_i is declared as failed by a non-faulty member M_j when M_j does not receive heartbeats from M_i for some consecutive heartbeat periods.

Distributed heartbeating schemes guarantee that a faulty member is always detected as such at any non-faulty member (within a time interval after its failure)[4], since a member that has crashed also stops sending heartbeat messages. However, the accuracy and scalability guarantees of these protocols differ, depending on the actual mechanism used to disseminate the heartbeats.

In the simplest implementation, each heartbeat is multicasted to all other group members. This results in a a network load of $\theta(\frac{n^2}{T})$ messages per second (even if IP multicast is used), where T is the failure detection time required by the distributed application. van Renesse et al [16] proposed that heartbeats be disseminated via a robust gossip-style protocol. In this protocol, every t_{gossip} time units, each member gossips, to a few random targets, a $\theta(n)$-sized list of the latest known heartbeat counters received from other members. While gossiping reduces the false positive frequency, a new heartbeat count typically takes, on expectation, $\theta[log(n) \cdot t_{gossip}]$ time units to reach an arbitrary other group member. In order to satisfy the application-specified detection time, the protocol generates a network load of $\theta(\frac{n^2.log(n)}{t_{gossip}})$ bytes a second. The use of message batching to solve this is limited by the UDP packet size limit, e.g. 5B heartbeats (IP address and count) of 50 members would already occupy 250 B, while SWIM generates packets that have a size of at most 135 B, regardless of the group size.

The quadratic increase in the network load results from the communication of heartbeat notification to *all* group members. This can be avoided by separating the failure *detection* operation from that of membership update *dissemination*.

Several hierarchical membership systems have been proposed, e.g. Congress [1]. This belongs to a broader class of solutions where each process heartbeats only a subgroup of processes. This class of protocols requires careful configuration and maintenance of the overlay along which membership information flows, and the accuracy of the protocol depends on the robustness of this graph. In comparison, the design of SWIM avoids the overhead of a virtual graph.

SWIM’s solution to the above unscalability problems described above is based on (a) designing the failure detection and membership update dissemination components separately, and (b) using a non-heartbeat based strategy for failure detection.

Before moving on to describe the SWIM protocol internals, we first lay the foundation for understanding the key characteristics of the efficiency and scalability of distributed failure detector protocols. Several research studies [6, 7, 12, 16], have led to the identification of these basic properties of distributed failure detector protocols (from both theoretical and practical angles), as well as impossibility results related to satisfying them concurrently. The resulting tradeoff is usually determined by the safety and liveness properties required by distributed applications.
These properties are [12]:
(1) *Strong Completeness*: crash-failure of any group member is detected by all non-faulty members [6]);
(2) *Speed of failure detection*: the time interval between a member failure and its detection by *some* non-faulty group member;
(3) *Accuracy*: the rate of false positives of failure detection;
(4) *Network Message Load*, in bytes per second generated by the protocol.

[6] proved the impossibility of building a failure detector over an asynchronous network that is both accurate (no false detections) and strongly complete. However, since a typical distributed application relies on Strong Completeness always holding (in order to maintain up to date information in dynamic groups), most failure detectors, including heartbeating-based solutions, guarantee this property while attempting to maintain a low rate of false positives. SWIM takes the same approach.

In [12], a simple computation identifies the minimal total network load (bytes per second) required to satisfy specified parameters of false detection rate at each member (denoted $\mathcal{PM}(\mathcal{T})$), and detection time ($\mathcal{T}$) in a group of size n. [12] calculates this load as $n \cdot \frac{log(\mathcal{PM}(\mathcal{T}))}{log(p_{ml})}$, where p_{ml} is the probability of a packet drop within the underlying network.

Although this calculation is done under idealized conditions of independent message loss probabilities on each message (p_{ml}), it serves as a good baseline for comparing the scalability of different failure detection protocols. For example, the all-to-all heartbeat protocols discussed in Section 2 have a sub-optimality factor that varies linearly with

[4] This property is called *Strong Completeness.*

group size.

3. The Basic SWIM Approach

As mentioned earlier, the SWIM approach has two components:
(1) a *Failure Detector Component*, that detects failures of members, and
(2) a *Dissemination Component*, that disseminates information about members that have recently either joined or left the group, or failed.

We now lay the ground by describing the basic SWIM protocol. The basic protocol uses the random-probing based failure detector protocol of [12] (Section 3.1) and disseminates membership updates via network multicast (Section 3.2). The SWIM protocol is developed in the succeeding section (Section 4) by refining this initial design.

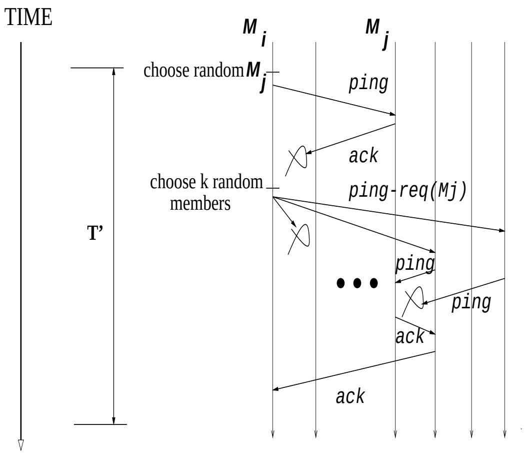


Figure 1. SWIM failure detection: Example protocol period at M_i. This shows all the possible messages that a protocol period may initiate. Some message contents excluded for simplicity.

3.1. SWIM Failure Detector

The SWIM failure detector algorithm [12] uses two parameters: protocol period T' (in time units) and integer k, the size of failure detection subgroups. The protocol does not require clocks to be synchronized across members, and properties of the protocol hold if T' is the average protocol period at group members.

Figure 1 illustrates the working of the protocol at an arbitrary member M_i. During each protocol period of length T' time units (on M_i's local clock), a random member is selected from M_i's membership list (say M_j), and a ping message sent to it. M_i then waits for a replying ack from M_j. If this is not received within a prespecified time-out (determined by the message round-trip time, which is chosen smaller than the protocol period), M_i indirectly probes M_j. M_i selects k members at random and sends each a ping-req(M_j) message. Each of these members in turn (those that are non-faulty), on receiving this message, pings M_j and forwards the ack from M_j (if received) back to M_i. At the end of this protocol period, M_i checks if it has received any acks, directly from M_j or indirectly through one of the k members; if not, it declares M_j as failed in its local membership list, and hands this update off to the Dissemination Component.

In the example of Figure 1, one of the k members manages to complete this cycle of events as M_j is up, and M_i does not suspect M_j as faulty at the end of this protocol period.

The prespecified time-out used to initiate indirect probing is based on an estimate of the distribution of round-trip time within the network, e.g. an average or 99^{th} percentile could be used. Note that the protocol period T' has to be at least three times the round-trip estimate. In our experiments, we use the average measured round-trip time to set the time-out, and our protocol period is significantly larger than this value.

The data contained in each message of this protocol is tagged with the unique sequence number of the protocol period at the initiator (M_i). Notice that the size of ping,ping-req,ack messages is bounded by a constant, and is independent of group size.

The second part of the above protocol uses an indirect probing subgroup of members to relay both pings and acks. The rationale for using this approach, rather than sending k ping messages directly to M_j, or relaying back acks in reply to ping-reqs directly back to M_i, is to avoid the effect of any congestion on the network path between M_i and M_j; this might have led to the dropping of the original ping message or its ack.
This failure detector protocol is analyzed in [12]. Here, we summarize the results of the analysis[5]:
• If each member has a membership list of size n, and a fraction q_f of these are non-faulty, the likelihood of an arbitrary member being chosen as a ping target in a protocol period is $1 - (1 - \frac{1}{n} \cdot q_f)^{n-1}$, which decreases quickly (and asymptotically as $n \rightarrow \infty$) to $1 - e^{-q_f}$.
• As a result, the expected time between failure of an arbitrary member and its detection by *some* process in the group is at most $T' \cdot \frac{1}{1-e^{-q_f}}$. This gives an estimate of the protocol period length in terms of the application-specified expected detection time.
• If q_{ml} is the probability of timely delivery of a packet by the network, independent across all packets, an arbitrary non-faulty member will be falsely detected as failed within

[5] The reader is encouraged to work out these results, or refer to [12].

a protocol period with probability $q_f \cdot (1 - q_{ml}^2) \cdot (1 - q_f \cdot q_{ml}^4)^k \cdot \frac{e^{q_f}}{e^{q_f}-1}$.

- This gives a configurable value for k in terms of the false positive probability required by the application.
- This failure detector satisfies Strong Completeness: a faulty member will eventually be chosen a ping target at each non-faulty member, and deleted from its membership list.
- The expected message load per member imposed by the protocol is a constant that does not vary with group size, and is symmetrical across all members. This load can be calculated from the estimate of k.
- None of these properties depend (except asymptotically) on the group size n.

3.2. Dissemination Component and Dynamic Membership

Upon detecting the failure of another group member, the process simply multicasts this information to the rest of the group as *failed(M_j)* message. A member receiving this message deletes M_j from its local membership list.

Information about newly joined members or voluntarily leaving members are multicast in a similar manner. However, for a process to join the group, it would need to know at least one contact member in the group. This can be realized through one of several means: if the group is associated with a well known server or IP multicast address, all joins could be directed to the associated address. In the absence of such infrastructure [6], join messages could be broadcast, and group members hearing it can probabilistically decide (by tossing a coin) whether to reply to it. Alternatively, to avoid multiple member replies, a static coordinator could be maintained within the group for the purpose of handling group join requests. In fact, existence of multiple coordinators does not affect the correctness of the protocol, and only leads to multiple replies to the join request. Discovery and resolution of multiple coordinators can be done over time through the Dissemination Component. In the current version of SWIM, we have chosen to maintain a coordinator, although there is no reason to preclude any of the other strategies.

4. A More Robust and Efficient SWIM

Section 3 described the basic SWIM protocol that disseminates membership updates (resulting from member joins, leaves or failures) using network multicast. However, network multicast primitives such as IP multicast etc., are only best-effort - message losses within the network can cause arbitrary and correlated non-receipt of membership changes at any group member. In Section 4.1, we describe the design of a Dissemination Component that piggybacks membership updates on the ping and ack messages sent by the failure detector protocol. This completely eliminates the generation of extra packets by the Dissemination Component (viz., multicasts). The only packets generated by SWIM then are pings, ping-reqs and acks, thus giving a constant expected message overhead per group member. This approach results in an infection-style of dissemination, with the associated benefits of robustness to packet losses, and of low latency.

The basic SWIM failure detector protocol, in spite of its calculable accuracy, is subject to slow processes (e.g. ones losing a lot of packets from buffer overflow) declaring several other non-faulty processes as faulty. It is also possible that a process is perturbed for small durations of time, e.g. on an overloaded host. This might cause the process to miss the opportunity to send timely replies to pings received meanwhile, and be mistakenly declared as failed. Section 4.2 presents the *Suspicion* mechanism, where a process that is unresponsive to ping messages, as generated by the SWIM failure detector protocol described in Section 3, is not immediately declared as “faulty”. Instead, the process is declared as “suspected”, and this information spread through the group using the Dissemination Component. After a prespecified time-out (we discuss values for this parameter in Section 5), the suspected process is declared as “faulty” and this information disseminated to the group. However, if the suspected process responds to a ping request before this time-out expires, information about this is disseminated to the group as an “alive” message. The process is then rejuvenated in membership lists at different members without ever having to leave or rejoin the group. This prespecified time-out thus effectively trades off an increase in failure detection time for a reduction in frequency of false failure detections.

The basic SWIM failure detection protocol guarantees eventual detection of the failure of an arbitrary process M_i, at each non-faulty group member M_j. However, it gives no deterministic guarantees on the time between failure of an arbitrary member M_i and its detection at another arbitrary member M_j (in terms of the number of local protocol rounds at M_j). Section 4.3 describes a modification to the original SWIM failure detector protocol that guarantees such a *Time Bounded Completeness* property; the time interval between the occurrence of a failure and its detection at member M_j is no more than two times the group size (in number of protocol periods).

4.1. Infection-Style Dissemination Component

The basic SWIM protocol of Section 3 propagates membership updates through the group using a multicast primitive. Hardware multicast and IP multicast are available on

[6] Absence of centralization is a common design philosophy in peer-to-peer systems today.

most networks and operating systems, but are rarely enabled, e.g., for administrative reasons. The basic SWIM protocol would then have to use a costly broadcast, or an inefficient point-to-point messaging scheme, in order to disseminate the membership updates to all group members. Furthermore, as this multicast is unreliable, membership changes can be disseminated only on a best-effort basis to the group.

Instead, the augmented SWIM protocol eliminates the use of an external multicast primitive altogether. It does so by piggybacking the information to be disseminated on the ping, ping-req and ack messages generated by the failure detector protocol. We call this an infection-style dissemination mechanism as information spreads in a manner analogous to the spread of gossip in society, or epidemic in the general population [8]. Notice that this implementation of the Dissemination Component does not generate any extra packets (such as multicasts) - all "messages" handed to this component are propagated by piggybacking on the packets of the Failure Detection Component.

Bailey [2] presents a deterministic analysis of the spread of an epidemic within a homogeneously mixing group of n members with one initial infected member. The relation between the (expected) number of infected members x (initially 1) and time t, under a contact rate of β per time unit, is obtained as:

$$\frac{dx}{dt} = \beta \cdot x \cdot (n - x) \Rightarrow x = \frac{n}{1 + (n-1)e^{-\beta n t}}$$

In our infection-style dissemination component, the spread of a membership update through the ping and ack messages can be analyzed in a similar manner. With the protocol period treated as a time unit, contact rate β is the probability of contact between any pair of infected and non-infected members, and equals $[1 - (1 - \frac{1}{n})^2] = (\frac{2}{n} - \frac{1}{n^2})$. This gives us $x = \frac{n}{1+(n-1)e^{-(2-\frac{1}{n})t}}$.

Such an epidemic process spreads exponentially fast in the group; after $t = \lambda.log\ n$ rounds of the protocol, where λ is a parameter, the expected number of infected members is $x = \frac{n}{1+(n-1)n^{-(2-\frac{1}{n})\lambda}} \geq n \cdot (1 - \frac{1}{n^{(2-\frac{1}{n})\lambda - 1}})$. A membership update propagated in infection-style by piggybacking will thus reach $(n - n^{-((2-\frac{1}{n})\lambda - 2)})$ group members after $\lambda.log\ n$ protocol periods. To simplify, as n increases (and $\rightarrow \infty$), the estimate for x goes to $(n - n^{-(2\lambda - 2)})$. Setting λ to a small constant suffices to disseminate the epidemic reliably - this is true even at small group sizes, as borne out by our experiments in Section 5.

The literature contains the analysis of several other styles of epidemics [4, 8, 13], with essentially similar conclusions about their probabilistic reliability. These analyses also show that the infection style of dissemination is resilient to process failures and loss of messages within the network, much like the contagiousness of epidemics. Experimental results of our implementation exhibit these characteristics.

A word on the implementation is in order. The SWIM protocol layer at each group member M_i maintains a buffer of recent membership updates, along with a local count for each buffer element. The local count specifies the number of times the element has been piggybacked so far by M_i, and is used to choose which elements to piggyback next. Each element is piggybacked at most $\lambda.log\ n$ times. If the size of this buffer is larger than the maximum number of elements that can be piggybacked on a single ping message (or ack), elements that have been gossiped fewer times are preferred. This is needed as the protocol period is fixed, and the rate of membership changes might temporarily overwhelm the speed of dissemination. Preferring "younger" buffer elements under such circumstances ensures that all membership changes infect at least a few members - when the membership change injection rate quiesces, these changes will propagate through the rest of the group.

Our implementation of this protocol maintains two lists of group members - a list of members that are not yet declared as failed in the group, and a second list of members that have failed recently. Currently, an equal number of buffer of elements is chosen from these two lists for piggybacking, but the scheme could be generalized to adapt to relative variations in process join, leave and failure rates.

4.2. Suspicion Mechanism: Reducing the Frequency of False Positives

In the SWIM failure detector protocol described so far, if a non-faulty group member M_j is (mistakenly) detected as failed by another group member M_i, either due to network packet losses or because M_j was asleep for sometime, or because M_i is a slow process, then M_j will be declared as failed in the group. In other words, a perfectly healthy process M_j suffers a very heavy penalty, by being forced to drop out of the group at the very first instance that it is mistakenly detected as failed in the group. This leads to a high rate of false positives in detecting failures.

We reduce the effect of this problem by modifying SWIM to run a subprotocol, called the *Suspicion* subprotocol, whenever a failure is detected by the basic SWIM failure detector protocol.

The Suspicion subprotocol works as follows. Consider a member M_i that chooses a member M_j as a ping target in the current protocol period, and runs the basic SWIM failure detector protocol period. If M_i receives no acknowledgments, either directly or through the indirect probing subgroup, it does not declare M_j as failed. Instead, M_i marks M_j as a *Suspected* member in the local membership list at M_i. In addition, a $\{$*Suspect* M_j*:* M_i *suspects* $M_j\}$ message is disseminated through the group through the Dissemination Component (in infection-style in our system). Any group member M_l receiving such a message also marks M_j as suspected. Suspected members stay on in the membership list and are treated similar to non-faulty members with regards to ping target selection operation of the SWIM failure detector protocol.

If a member M_l successfully pings a suspected member member M_j during the due course of the basic SWIM protocol, it un-marks the previous suspicion of M_j in its membership list, and spreads an {*Alive* M_j*: M_l knows M_j is alive*} message in the group through the Dissemination Component (in infection-style in our system). Such an *Alive* message un-marks the suspected member M_j in membership lists of recipient members. Notice that if member M_j receives such a message suspecting it, it can start propagating an *Alive* message clarifying its non-failure.

Suspected entries in membership lists expire after a prespecified time-out. If M_j is suspected at some member M_h, and this entry times-out before receipt of an *Alive* message, M_h declares M_j as faulty, drops it from the local membership list, and begins spreading the message {*Confirm* M_j*: M_h declares M_j as faulty*} through the Dissemination Component. This message overrides any previous *Suspect* or *Alive* messages, and cascades in deletion of M_j from the membership lists of all recipients.

This mechanism reduces (but does not eliminate) the rate of failure detection false positives. Notice also that the Strong Completeness property of the original protocol continues to hold. Failures of processes suspecting a failed process M_j may prolong detection time, but eventual detection is guaranteed.

From the above discussion, *Alive* messages override *Suspect* messages, and *Confirm* messages override both *Suspect* and *Alive* messages, in their effect on the local membership list element corresponding to the suspected member M_j. However, a member might be suspected and unsuspected multiple times during its lifetime. These multiple versions of *Suspect* and *Alive* messages (all pertaining to the same member M_j) need to be distinguished through unique identifiers. These identifiers are provided by using a virtual *incarnation number* field with each element in the membership lists. Incarnation numbers are global. A member M_i's incarnation number is initialized to 0 when it joins the group, and it can be incremented only by M_i, when it receives information (through the Dissemination Component) about itself being suspected in the current incarnation - M_i then generates an *Alive* message with its identifier and an incremented incarnation number, and spreads this through the Dissemination Component to the group.

Thus, $Suspect$, $Alive$, and $Confirm$ messages contain the incarnation number of the member, besides its identifier. The order of preference among these messages and their effect on the membership list is specified below.

- {*Alive* $M_l, inc = i$} overrides
  - {*Suspect* $M_l, inc = j$}, $i > j$
  - {*Alive* $M_l, inc = j$}, $i > j$
- {*Suspect* $M_l, inc = i$} overrides
  - {*Suspect* $M_l, inc = j$}, $i > j$
  - {*Alive* $M_l, inc = j$}, $i \geq j$
- {*Confirm* $M_l, inc = i$} overrides
  - {*Alive* $M_l, inc = j$}, any j
  - {*Suspect* $M_l, inc = j$}, any j

It is easy to see that these orders of preference and overriding maintain the desired correctness properties of the Failure Detector Component. The reader familiar with adhoc routing protocols such as AODV [5] will notice the similarity between their use of destination sequence numbers and our incarnation number scheme.

The preference rules and infection-style Dissemination Component also accommodate suspicions of a process by multiple other processes. Preference rules do not depend on the source of suspicion, and the infection-style dissemination spreads a message (*Suspect*, *Alive* or *Confirm*) quicker if there are multiple sources, with exactly the same overhead per process as with one source of infection [8].

4.3. Round-Robin Probe Target Selection: Providing Time-Bounded Strong Completeness

The basic SWIM failure detector protocol described in Section 3 detects failures in an average constant number of protocol periods. Although each process failure is guaranteed to be detected eventually at every other non-faulty process (eventual Strong Completeness), a pathological selection of ping targets across the group might lead to a large delay in the first detection of the process failure anywhere in the group. In the extreme case, this delay could be unbounded as the failed process might never be chosen as a ping target by any other non-faulty process.

This can be solved by the following modification to the protocol. The failure detection protocol at member M_i works by maintaining a list (intuitively, an array) of the known elements of the current membership list, and selecting ping targets not randomly from this list, but in a round-robin fashion. Instead, a newly joining member is inserted in the membership list at a position that is chosen uniformly at random. On completing a traversal of the entire list, M_i rearranges the membership list to a random reordering.

Consider the execution of the SWIM protocol, modified as described above, at member M_i. Once another member M_j is included in M_i's membership list, it will be chosen as a ping target exactly once during each traversal of M_i's membership list. If the size of the membership list is no more than n_i, successive selections of the same target are at most $(2 \cdot n_i - 1)$ protocol periods apart. This bounds the worst case detection time of a process failure of any member by M_i, thus satisfying a *Time Bounded* Completeness property.

The average failure detection time of the original protocol is preserved by this optimization, since the randomization of the membership lists at different members across the group leads to a similar distribution of ping target choices by each member.

5. Performance Evaluation of a Prototype

A prototype of the SWIM protocol was implemented over the Winsock 2 API, and was tested in a large cluster of commodity PCs running Windows 2000. The PC cluster was comprised of 16 450-MHz Dell PII's, 16 1-GHz IBM x220's, and a collection of dual and quad nodes (200-MHz to 500-MHz PII and PIII processors), communicating over a 100 Mbps Ethernet with no external load. Each node contained at most one process group member.

The experimental parameters were set as follows. The number of members chosen for ping-reqs was $K = 1$, and the protocol period used was 2 sec. Each infection (membership update) was piggybacked on at most $(3\lceil log(N + 1)\rceil)$ messages sent by each member. With this value setting, our experimental runs observed no evidence of perpetual partial membership lists at members or involuntary partitions of the group. The suspicion time-out for a suspected member to be declared as failed was also set to this value.

We compared three different versions of the protocol: (1) (*SWIM:Basic*) the basic SWIM protocol of Section 3, modified with the Round-Robin scheme described in Section 4.3, (2) (*SWIM+Inf.*) SWIM with an infection-style Dissemination Component (as described in Section 4.1), and (3) (*SWIM+Inf.+Susp.*) SWIM+Inf. with the Suspicion subprotocol extension described in Section 4.2.

All point-to-point messages sent by the SWIM protocol were UDP packets. The maximum message payload sizes were 15 B in SWIM:Basic, and 135 B in the SWIM+Inf. and SWIM+Inf.+Susp. protocols (since at most 6 membership updates were piggybacked per message[7]).

5.1. Message Loads

Figure 2 shows the measured message send and receive loads imposed by the SWIM failure detector at an arbitrary group member. The readings are taken over a time period spanning over 40 protocol periods. Up to a group size of 55 members, the average overhead stays around 2.0. This matches the analytic estimate as, during each protocol period, a group member sends a single ping and receives the corresponding ack, as well as receives one ping message (on expectation) and sends the corresponding ack. The standard deviation bars indicate that typical message overheads stay low, e.g. at N=28 members, the overhead of sent messages is smaller than 5 messages per protocol period with a probability of 0.99.

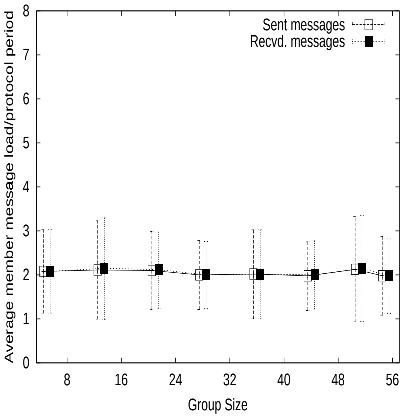


Figure 2. Send and Receive message overheads at a group member. Points perturbed horizontally for clarity.

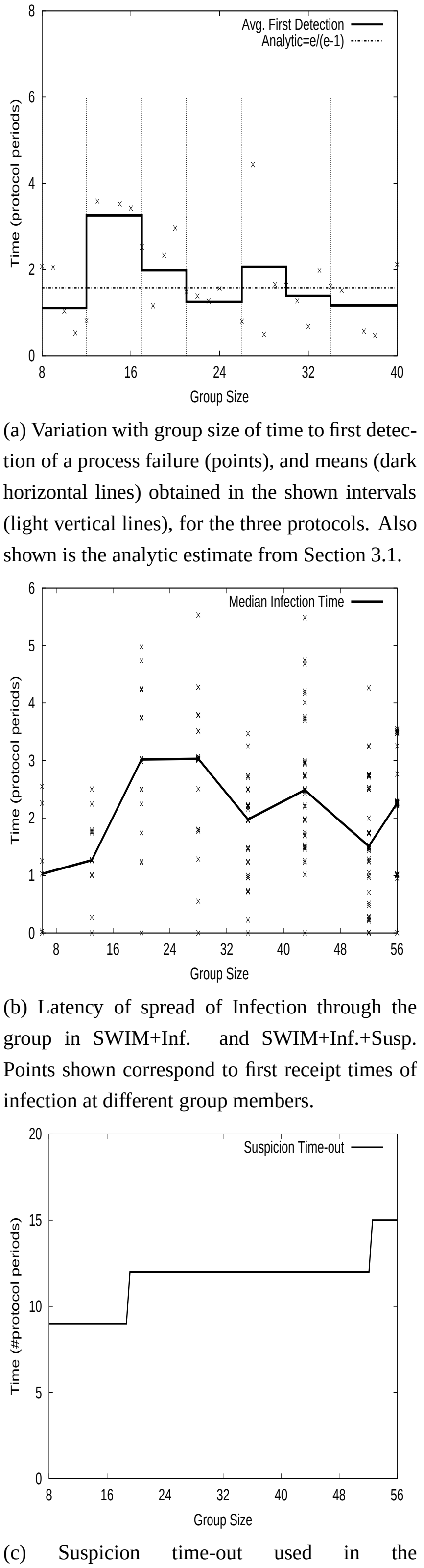
Recollect from our description in Section 4 that this plot shows the *total* message overhead per member in SWIM+Inf. and SWIM+Inf.+Susp, while only the base load in SWIM:Basic (where additional multicasts disseminate membership updates).

5.2. Detection and Dissemination Latency of Membership Updates

Figure 3 shows the break up of failure detection and dissemination times for the three different protocols. Figure 3(a) shows the mean time between a process failure and its first detection at some non-faulty group member. The average detection time does not appear to be correlated to group size and thus matches the analytic estimate. In the SWIM:Basic protocol, the vertical axis values correspond to the interval between a process failure and the instant of multicast of its failure notification. In the SWIM+Inf. and SWIM+Inf.+Susp. protocols, these values correspond to the interval between a failure and the start of spread of the failure notification and suspicion notification infections, respectively.

Figure 3(b) plots the variation with group size of the delay in infection-style dissemination of membership updates. The median latency of dissemination is always only a few protocol periods, and appears to be rise slowly with group size (recall the analysis of Section 4.1 predicted a logarithmic variation of the average infection time with group size).

[7] This parameter could be adapted to high membership update rates, thus trading off overhead for dissemination time.



(a) Variation with group size of time to first detection of a process failure (points), and means (dark horizontal lines) obtained in the shown intervals (light vertical lines), for the three protocols. Also shown is the analytic estimate from Section 3.1.

(b) Latency of spread of Infection through the group in SWIM+Inf. and SWIM+Inf.+Susp. Points shown correspond to first receipt times of infection at different group members.

(c) Suspicion time-out used in the SWIM+Inf.+Susp. protocol.

Figure 3. Detection and Dissemination Time of membership updates.

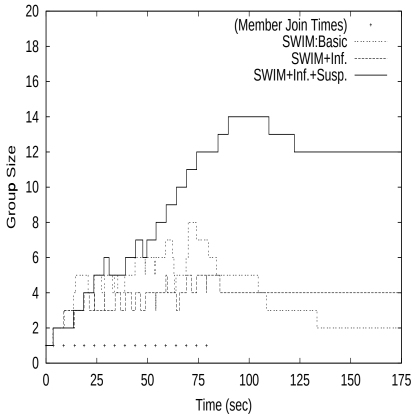


Figure 4. Effect of 10% packet loss rate on a member join sequence (blips at bottom), with conservative protocol parameters.

Figure 3(c) shows the suspicion time-out $(3\lceil log(N+1)\rceil)$, and is applicable only to the SWIM+Inf.+Susp. protocol.

The average time between a process failure and its removal from the membership lists of other members is obtained from the three plots in Figure 3. For the SWIM:Basic protocol, this is the value plotted in Figure 3(a) plus the time taken by a network multicast. For the SWIM+Inf. protocol, this is the sum of the distributions on the vertical axes of Figure 3(a) and Figure 3(b), while for the SWIM+Inf.+Susp. protocol, this delay is the sum of the distributions on the three plots in Figure 3.

5.3. Failure Detection False Positives

The runs of the SWIM+Inf.+Susp. protocol in Figure 3 generated an insignificant number of false failure detections. In order to evaluate the relative benefit of the infection and suspicion mechanisms, we synthesized a high artificial packet drop of 10%. Each packet (including each network multicast in SWIM:Basic) was dropped at the receiver with this loss probability. In our experiment, 17 processes attempted to join a SWIM group sequentially, as shown in Figure 4. The plot shows the variation of group size during and after this joining phase, in spite of the sustained packet loss rate. The plot is cut off at 175 s, after the last observed membership change. This plot shows the benefit of the Suspicion mechanism: while SWIM+Inf.+Susp. attains a stable group size of 12 members, SWIM:Basic and SWIM+Inf. stabilize at 2 and 4 members respectively.

The data in Figure 4 holds for the described experimental settings only. A more aggressive setting of parameters K and the suspicion time-out would increase the robustness of the three protocols. Varying the suspicion time-out also provides a knob to trade off failure detection time with the frequency of false positives. We leave the investigation of such issues for a later article.

6. Conclusions and Future Work

We have presented the design, implementation and performance evaluation of SWIM, a scalable weakly-consistent process group membership protocol. The SWIM project is motivated by the unscalability of heartbeat-based protocols, which are popular with distributed system designers today.

SWIM's solution is based on a separation of the failure detector and membership update dissemination components of the problem. The SWIM failure detector achieves scalability by avoiding heartbeating, and by using a random peer-to-peer probing of processes instead. This provides constant overhead on group members, as well as constant expected detection time of failures. Membership updates are propagated efficiently and reliably in infection-style (epidemic-style), by piggybacking on packets generated by the failure detector protocol. The addition of a suspicion mechanism (with virtual incarnation numbers) reduces the false positive frequency, while trading off failure detection time. A final extension to the protocol guarantees time bounded detection of failures at each non-faulty process.

SWIM can be extended to a Wide-area network (WAN) or a virtual private network (VPN), by weighing ping target choices with topological information, thus reducing bandwidth usage on core network elements inside the network [16]. We are currently evaluating this feature.

This paper has described the design and implementation of SWIM in a specific context, but our results are more generally applicable. Although SWIM's design is targeted at large groups of processes, our analysis and experimental results show that this alternative to all-to-all distributed heartbeating can give an order of magnitude reduction in overhead if used within more medium-sized subgroups, e.g. the replica groups in distributed hash table (DHT) systems such as Chord, Pastry, Opus [15]. The Suspicion Mechanism is generally applicable to any membership system with distinct Failure Detection and Membership Update Component designs.

The extent of continued proliferation of large-scale distributed applications in today's Internet will depend on scalable and efficient design of the protocols that run within these systems. SWIM offers such a solution to the group membership component for these applications.

Acknowledgments: We thank Werner Vogels for help with accessing and using the Galaxy cluster. We also thank Ben Atkin, Ken Birman, Alan Demers, Robbert van Renesse, and the anonymous referees, for reviewing drafts.

References

[1] T. Anker, D. Breitgand, D. Dolev, and Z. Levy. CONGRESS: Connection-oriented group-address resolution service. In *Proc. SPIE'97 on Broadband Networking Technologies*, 1997.

[2] N. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications*. Hafner Press, second edition, 1975.

[3] K. P. Birman. The process group approach to reliable distributed computing. *Comm. of the ACM*, 36(12):37–53, 1993.

[4] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. on Computer Systems*, 17(2):41–88, 1999.

[5] C.E.Perkins and E. Royer. Ad hoc on-demand distance vector routing. In *Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[6] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journ. of the ACM*, 43(2):225–267, 1996.

[7] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. In *Proc. 30th Intnl. Conf. on Dependable Systems and Networks*, pages 191–200, 2000.

[8] A. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. In *Proc. 6th Annual ACM Symp. Principles of Distributed Computing*, pages 1–12. ACM Press, 1987.

[9] S. A. Fakhouri, G. Goldszmidt, I. Gupta, M. Kalantar, and J. A. Pershing. Gulfstream - a system for dynamic topology management in multi-domain server farms. In *Proc. 3rd IEEE Intnl. Conf. on Cluster Computing*, pages 55–62, 2001.

[10] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journ. of the ACM*, 32(2):374–382, 1985.

[11] I. Gupta, K. Birman, and R. van Renesse. Fighting fire with fire: using randomized gossip to combat stochastic scalability limits. To appear in *The Journ. of Quality and Reliability Engineering International*, 2002.

[12] I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On scalable and efficient distributed failure detectors. In *Proc. 20th Annual ACM Symp. on Principles of Distributed Computing*, pages 170–179. ACM Press, 2001.

[13] A.-M. Kermarrec, L. Massoulie, and A. Ganesh. Reliable probabilistic communication in large-scale information dissemination systems. Technical Report MMSR-TR-2000-105, Microsoft Research, Cambridge, UK, 2000.

[14] K. Petersen, M. Spreitzer, D. Terry, M. Theimer, and A.J.Demers. Flexible update propagation for weakly consistent replication. In *Proc. 16th ACM Symp. on Operating Systems Principles*, pages 3–6. ACM Press, 1997.

[15] A. Rowstron and F. Kaashoek, editors. *Proc. 1st Intnl. Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002.

[16] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proc. Middleware*, pages 55–70, 1998.

`http://www.cs.cornell.edu/gupta/swim`