On the Performance of Quorum-Based Systems over the Internet

ABSTRACT
Replicated systems often use quorums in order to increase their performance and availability. We study the running time of quorum-based distributed systems deployed in a widely distributed setting over the Internet. We study a simple primitive that propagates information from a quorum of hosts to either one host or to a quorum of hosts; this primitive occurs in numerous distributed systems. We experiment with centralized and decentralized implementations of this primitive, and with two different quorum systems. We run our experiments on twenty-eight hosts at geographically disperse locations over the Internet.

1. INTRODUCTION
Replication is a fundamental tool in achieving reliability and high availability. There is a cost to keeping replicas consistent: operations to be disseminated to multiple replicas. However, operations need not be disseminated to all the replicas in order to ensure consistency; it suffices to have operations access a majority of the replicas [20], or a collection of replicas that have a majority of votes in a system using voting [8]. More generally, replica management can be based on the notion of a quorum-system. Given a collection of hosts (replicas, servers), a quorum system is a collection of sets of hosts, called quorums, such that every two quorums in the collection intersect. Given a known quorum-system, it often suffices to have each operation access a quorum of the replicas.

The simplest and most common quorum system is majority, where the quorums are the sets that include a majority of the hosts. The majority quorum system has been shown to be the most available one [16][1]. That is, it has the highest probability for surviving. The cost of achieving this availability is that quorums are fairly large. The (or grid) quorum system employs much smaller quorums, of order $\sqrt{n}$. A table quorum system is constructed by arranging the hosts in a $\sqrt{n} \times \sqrt{n}$ table, as shown in Figure 1. A quorum is a union of a full row and one element from each row above the full row. If the number of hosts is not a perfect square, some rows might include empty cells.

Figure 1: The Table quorum system of 6 x 6, with one quorum shaded.

In this paper we evaluate the running time and availability of a primitive that gathers a small amount of information from a quorum and propagates it to a quorum. We call this primitive gather-quorum. An instance of gather-quorum can be initiated by one host, which is called the initiator of this instance. The instance terminates once information from a quorum of hosts has propagated to a quorum of hosts. We measure the overall running time of the primitive, which is the time until information has propagated to a quorum of hosts, as well as the local running time, which is the time until information has propagated to the initiator of this instance. The latter reflects the running time of a primitive that gathers information from a quorum at one host (the initiator) and does not propagate it. The primitive we measure thus reflects two communication patterns that occurs in numerous algorithms and systems, e.g., [12, 19, 17, 6, 9, 13, 14, 21], and hence our study has broad applicability.

In 4 we compare the gather-quorum primitive to a primitive that gathers information from all live hosts, which we call gather-all. The gather-all primitive mimics the communication pattern used, e.g., in group membership systems [5], and replication schemes based on group membership [11, 7, 1] where all the members of a view need to be communicated with. Thus, our comparison of gather-quorum and gather-all can be seen as a comparison of the quorum-based approach with the group membership approach to wide area replica-
tion. Our results show that especially in the presence of lossy communication links, the performance of gather-quorum is orders of magnitude better than that of gather-all.

In Section 5, we look at the impact the set of hosts that are being probed by a gather-quorum algorithm has on system performance. There are typically many quorums in a quorum system, but an instance of the gather-quorum primitive has to gather information from just one of them. This raises the question of how many hosts to probe, or to send requests to. One option is to probe every host in the system, and wait for any of the hosts to respond. At the other extreme, it is possible to probe exactly one quorum and wait for that particular quorum to respond. And in general, it is possible to probe any number of quorums, and wait for one of them to respond. We call the set of hosts probed by the initiator the probe set. There is a tradeoff in choosing the size of the probe set: On one hand, using a smaller probe set reduces the overall load on the system. On the other hand, using larger probe set increases availability, and can potentially improve performance. We study the impact of the choice of probe set on the algorithm running time and availability. Our study shows that increasing the probe set beyond the minimal one, (which probes a single quorum) yields significant performance gains. However, these performance gains taper off beyond a certain probe set size (depending on various factors), where it is no longer cost-effective to increase the probe set.

We compare a centralized versus a decentralized implementation of the gather-quorum primitive. Specifically, We evaluate the following two algorithms implementing the gather-quorum primitive:

- **quorum-to-quorum**, where the initiator sends a message to every host in the probe set, and each host that hears from the initiator sends messages to all hosts in the probe set. The algorithm terminates as soon as quorum hears from a quorum and fails otherwise.

- **quorum-leader**, where the initiator acts as the leader. After the initiator sends a message to every host in the probe set, the hosts respond by sending messages to the leader. Once the leader hears from a quorum, it aggregates the information from hosts in that quorum, and sends a message summarizing all the inputs to all the hosts in the probe set. The algorithm is structured similar to Lamport’s Paxos algorithm [12] and the algorithm used in the Frangipani distributed file system [19].

We study the running time of the gather-quorum primitive over the Internet. Our experiments span twenty-eight hosts, widely distributed over the Internet – in Korea, Taiwan, Israel, Australia, New Zealand, and several hosts across Europe and North America. Some of the hosts reside at academic institutions and others on commercial ISP networks. We present data that was gathered over several months. The hosts communicate using TCP/IP. TCP is a commonly used protocol on the Internet, and therefore evaluating systems that use it is of interest. Moreover, it was feasible for us to deploy a TCP-based system because TCP is a “friendly” protocol that does not generate excessive traffic at times of congestion, and because firewalls at some of our hosts block UDP traffic. We run a single process at each geographical location. We do not address issues related to scaling the number of processes, as we believe that such issues are orthogonal to our study.

We analyze our experimental results, and explain the observed algorithm running times in terms of the underlying network characteristics. Due to the great variability of running times over TCP, especially with high loss rates, the average running time is not indicative of an algorithm’s typical behavior. We therefore focus on the distribution of running times. We devise a simple estimator to predict system performance based on the underlying network characteristics. The estimator assumes independent message loss. We compare our results with those predicted by the estimator, and thus illustrate the effect correlated loss has.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 describes the experiment setup and methodology. Section 4 compares the gather-quorum primitive with gather-all and discusses tradeoffs between the two. Section 5 studies the effect of the probe set on the running time of gather-quorum algorithms and looks at the viability and accuracy of a simple estimator. Section 6 concludes the paper.

2. RELATED WORK

Although quorums systems are widely used, and numerous quorum-based systems have been developed (e.g., [12, 19, 17, 6, 9, 13, 14, 21]), we are not aware of any previous systematic study of the performance of such systems over the Internet. The primary foci for previous research on evaluation of quorum systems were availability and load. Load is typically evaluated assuming a probe set consisting of a single quorum. A common metric used to evaluate the availability of quorum systems is the probability of failure. A quorum system fails when no quorum exists. Different approaches have been used to study this metric. One approach is using probabilistic models to obtain theoretical results on the probability of failure [16, 15]. While this approach is most rigorous, it makes oversimplifying assumptions about the underlying network, e.g., that failure rates are independent and identically distributed (IID). Such assumptions do not hold in today’s Internet. Another approach used by Amir and Wool [2] involves running experiments consisting real hosts connected to the Internet and using a group membership protocol to track failures and network partitions. The availability of different quorum systems is then evaluated based on the traces collected in the experiments. However, none of these studies provide an adequate framework for evaluating the running times of distributed quorum-based systems.

Two recent works study the performance of gather-all or group membership-based solutions over the Internet: Bakr and Keidar [4] evaluate a service similar to our gather-all primitive over the Internet. They show that message loss has a large impact on algorithm running times. We show that, while message loss still does have an impact, its affect is mitigated by settling for the gather-quorum primitive instead of gather-all. Jacobsen et al. [10] analyze algorithms that use group membership for large-scale master-worker computations over the Internet and conclude that using group membership for wide-area computations is not a good idea, since group membership is bound to be highly unstable over the Internet.

3. METHODOLOGY AND SETUP
3.1 The hosts

We use the following 28 hosts (the majority of which are part of the RON testbed [3]) in our experiments: MIT, at the Massachusetts Institute of Technology, Cambridge, MA, USA; MA1 and MA2 at two commercial ISPs in Cambridge, MA, USA; MA3, at a commercial ISP in Martha’s Vineyard, MA, USA; NYU, at New York University, New York, NY, USA; CU, at Cornell University, Ithaca, NY, USA; NY, at a commercial ISP in New York, NY, USA; CMU, at Carnegie Mellon University, Pittsburgh, PA, USA; NC, at a commercial ISP in Dhuram, NC, USA; Emulab, at the University of Utah, Salt Lake City, UT, USA; UT1 and UT2 at two commercial ISPs in Salt Lake City, UT, USA; UCSD, at the University of California San Diego, San Diego, CA, USA; CA1, at a commercial ISP in Foster City, CA, USA; CA2, at Intel Labs in Berkeley, CA, USA; CA3, at a commercial ISP in Palo Alto, CA, USA; CA4, at a commercial ISP in Sunnyvale, CA, USA; Canada (CND), at a commercial ISP in Nepean, ON, Canada; Sweden (SWD), at Lulea University of Technology, Lulea, Sweden; Netherlands (NL), at Vrije University, Amsterdam, the Netherlands; Greece (GR), at the National Technical University of Athens, Athens, Greece; Switzerland (Swiss), at the Swiss Federal Institute of Technology, Lausanne, Switzerland; Israel (ISR1), at the Israel Institute of Technology (Technion), Haifa, Israel; Israel (ISR2), at the Hebrew University of Jerusalem, Jerusalem, Israel; Korea (KR), at Korea Advanced Institute of Science and Technology, Daejon, South Korea; Taiwan (TW), at National Taiwan University, Taipei, Taiwan; Australia (AUS), at the University of Sydney, Sydney, Australia; and New Zealand (NZ), at Victoria University of Wellington, Wellington, New Zealand. All the hosts run either FreeBSD or GNU/Linux or Solaris operating systems.

3.2 Server Implementation

We ran our experiments over TCP. We ran a server at every host, implemented in Java, optimized with the GCJ compiler. Each server has knowledge of the IP addresses and ports of all the potential servers in the system. Asynchronous I/O is implemented using threads.

Every server keeps an active TCP connection to every other server that it can communicate with. We disable TCP’s default waiting before sending small packets (cf. Nagle algorithm, [18, Ch. 19]). Each server periodically attempts to set up connections with other servers to which it is not currently connected. A crontab monitors the status of the server, and restarts it if it is down. Thus, when either a server or communication failure is repaired, connection is promptly reestablished. In case the communication is not transitive, different hosts can have different views of the current set of participants. In case of host or communication failures, an instance of the algorithm may fail to terminate. This situation can be detected by the failure of a TCP connection or by a timeout.

Each server has code implementing both decentralized and centralized gather-all and gather-quorum algorithms. The server periodically invokes the algorithms: it sleeps for a random period, and then invokes one of the algorithms, in round-robin order. Each invocation of an algorithm is called a session. We use randomness in order to reduce the probability of different sessions running at the same time and delaying each other; this is easier than synchronizing the invocations, as the hosts do not have synchronized clocks.

We constantly run ping and traceroute from each host to each of the other hosts, periodically sending ICMP packets, in order to track the routing dynamics, the latency and loss rate between every pair of hosts in the underlying network. The ping and traceroute processes are also monitored by a crontab.

3.3 Running Times and Clock Skews

We use two measures of running time:

- The local running time of a session at that session’s initiator is the clock time elapsing from when this host begins this session and until the same host terminates the session (according to that host’s local clock). In a gather-all algorithm, this is the time until the initiator receives a message from every other live host in its view. In a gather-quorum algorithm, it is the time until it hears from a quorum of hosts.

- The overall running time of a session is the time elapsing from when the initiator begins this session until all the hosts terminate this session. In gather-all, an algorithm terminates when every live host hears from every other host in its view. In gather-quorum, an algorithm terminates when there exists a quorum such that each host in that quorum heard from a quorum.

For each session, each host writes to log its starting time, termination time, and the time it receives messages from other hosts, according to its local clock. Since we do not own the hosts used in our experiments, we were not able to synchronize their clocks. Therefore, in order to deduce the overall running time from the log files, we need to know the skews between different hosts’ clocks. This is not necessary in the case of leader-based algorithms where the local running time gives a good indication of the overall performance since the all phases are relatively symmetric (the same number of messages travel on the same links). However, this is not the case with quorum-to-quorum (or its counterpart all-to-all) where the local running time fails to capture a lot of the dynamics of the system.

We now explain how we estimate the clock differences. Whenever a host A sends a message to host B, it includes in the message its local clock time. When host B receives the message, it computes the difference between its local clock time and the time in the message, and writes this value to log. Denote this value by $\Delta_{AB}$. Assume that B’s clock is $d_{AB}$ time ahead of A’s, and assume that the average message latency from A to B and from B to A is $l_{AB}$. Then on average, $\Delta_{AB} = l_{AB} + d_{AB}$ and symmetrically, $\Delta_{BA} = l_{BA} - d_{AB}$. Therefore, $\Delta_{AB} - \Delta_{BA}$ is, on average, $2d_{AB}$. We approximate the clock difference between A and B as:

$$\text{average}(\Delta_{AB} - \text{average}(\Delta_{BA}))^2$$

This approximation method has some limitations for servers that communicate over TCP: since messages are exchanged are only interested in the times when a quorum hears from a quorum and we disregard data that is irrelevant.
over TCP\(^3\), the latency can vary substantially in case of message loss. Therefore, if a pair of hosts communicate over a lossy link, this method can give a bad approximation for the clock difference. Furthermore, high variations in the (loss-free) latency between a pair of hosts make it harder to distinguish between high TCP latencies that are due to message loss and those that due to variations (routing delays). However, we can detect (and avoid) some of the cases in which lost messages occur in bursts by setting a threshold. If the round-trip time of a sample exceeds the given threshold, then we can discard that sample when computing the clock difference. From our observation, for pairs of hosts with loss rates less than 10%, most losses are clustered around short periods of time (bursts). This increases the accuracy of the above method in detecting losses. The higher we set the threshold, the less likely we are to discard samples that do not reflect a loss, but message losses are also more likely to go undetected. For each pair of hosts, we set the threshold to be twice the average round-trip time plus four times the standard deviation. Moreover, we discovered that when the average clock skew is computed over a long interval, results can be inconsistent, because some hosts experience clock drifts. So instead of taking the average over all samples, we compute the average over samples obtained in shorter intervals (we used intervals that are one hour long).

We fix a base host \(h\), and compute the clock differences between \(h\) and every other host per every 15 minute time interval. Then, all logged running times in this interval are adjusted to \(h\)'s clock, and the overall running time is inferred from the adjusted initiation and termination times. In order to minimize the effect of TCP retransmission delays, it is preferable to choose a host that has reliable links to every other host. In order to check the consistency of our results, we computed the overall running times using three different base hosts: MIT, Emulab, and Cornell. We chose these hosts since the links to them from all hosts were fairly reliable and exhibited a low variation of latency.

Having computed the running times three different ways, we found the results to be fairly consistent. The distributions of overall running times as computed with each of the three hosts were almost identical. Moreover, for over 90% of the sessions with overall running times up to 2 seconds, the three computed running times were within 20 ms. of each other.

### 3.4 Estimating the Running Times

An important part of this study is being able to effectively estimate the running times of algorithms based on the distribution of TCP latencies on the individual underlying links. The ability to estimate the running time helps hosts make better decisions regarding which algorithms to choose and which hosts to probe.

Assume that we know the distribution of running times over all the underlying TCP links. From these, we can extrapolate the distribution of running times of an algorithm that runs over these links. For simplicity, we assume that different messages are routed independently. Then a host \(a\) can predict the probability that the local running time of the leader-based gather-all algorithm is less than \(l\) units of time using the following formula:

\[
Pr[\text{leader}_a < l] = \prod_{h \in U} Pr[\text{rtt}_{ah} < l]
\]

Where \(\text{leader}_a\) is a random variable that denotes the time when \(a\) hears from every host, \(U\) denotes the universe of hosts, and \(\text{rtt}_{ah}\) is a random variable representing the TCP round trip time between hosts \(a\) and \(h\).

Also for a given probe set \(P\), let \(S_i\) be the set of all subsets of \(P\) that are of size \(k\). Then the probability that an initiator \(a\) hears from at least \(m\) hosts (where \(m\) is the minimum size of a majority) within less than \(l\) units of time can be expressed as follows: \(Pr[\text{majority}_a(P) < l] = \sum_{i=m}^{n} \sum_{s \in S_i} \prod_{h \in s} Pr[\text{rtt}_{ah} < l] \prod_{h \in P \setminus s} (1 - Pr[\text{rtt}_{ah} < l])\)

Where \(\text{majority}_a(P)\) is a random variable that denotes the time when \(a\) hears from a majority when it has probed the elements of \(P\). If we have a good estimate of the TCP distributions, we can get a reasonably good approximation of the cumulative distribution of the running time of an algorithm as will be shown in Section 5. The formula is inefficient in practice since the number of terms grows exponentially as we increase the number of hosts. However, it can be simplified (with a little accuracy trade off) by disregarding terms of probabilities of .9 or greater especially since such reliable links usually do not fail independently. The procedure gets even more complicated for the all-to-all and quorum-to-quorum algorithms and we do not show any estimates for these cases.

### 4. COMPARISON WITH GATHER-ALL

This section and the following section present results gathered from two experiments. Experiment I lasted approximately nine and a half days and included a total of 27 hosts. Of the hosts that participated only MIT, UT2, CA1, NL, ISR1, AUS, KR and TW were initiators. Each initiator ran leader once every two minutes on average, and in total, roughly 6700 times. Experiment II lasted approximately five and a half days and included a total of 18 hosts, located in North America, out of which only MIT, CA1, Emulab, CU were initiators. Each initiator ran all-to-all once every two minutes on average, and in total, we accumulated roughly 3700 samples per initiator. In both experiments, hosts sent ping probes to each other once every two minutes.

Even though we did not explicitly run algorithms that implement the gather-quorum primitive, we extrapolated the running times of these algorithms from the data we accumulated by only looking at the response times of quorums for different probe sets and disregarding irrelevant data. In our analysis of experiment I, since we only ran the leader algorithm, it was enough to consider the local running times to get a fair comparison. In both experiments, host crashes and network partitions occurred. Table 1 shows the hosts that participated in Experiment I. The hosts that participated in Experiment II appear in Table 2.

Since the performance of gather-quorum algorithms depend on the probe set, any comparison with gather-all algorithms depends on this parameter. Section 5 provides a more comprehensive look on how probe sets impact performance. However, in this section, we consider only the two extreme cases: minimal probe sets, including exactly one quorum, and complete probe sets, including all hosts. However, independent of the size of the probe set, we can
Table 1: Table quorum system in experiment I.

<table>
<thead>
<tr>
<th>MIT</th>
<th>CU</th>
<th>NYU</th>
<th>Emulab</th>
<th>UCSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>SWD</td>
<td>GR</td>
<td>ISR1</td>
<td>ISR2</td>
</tr>
<tr>
<td>MA1</td>
<td>MA3</td>
<td>NY</td>
<td>UT1</td>
<td>CND</td>
</tr>
<tr>
<td>MA2</td>
<td>AUS</td>
<td>CU</td>
<td>CA1</td>
<td>UT2</td>
</tr>
<tr>
<td>NZ</td>
<td>TW</td>
<td>RR</td>
<td>Swiss</td>
<td>C2A2, C3A3</td>
</tr>
</tbody>
</table>

Table 2: Table quorum system in experiment II.

<table>
<thead>
<tr>
<th>MIT</th>
<th>CU</th>
<th>NYU</th>
<th>CMU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulab</td>
<td>UCSD</td>
<td>UT1</td>
<td>UT2</td>
</tr>
<tr>
<td>MA1</td>
<td>MA2</td>
<td>MA3</td>
<td>NY</td>
</tr>
<tr>
<td>CND</td>
<td>CA1</td>
<td>CA2</td>
<td>CA3</td>
</tr>
</tbody>
</table>

Table 1: Table quorum system in experiment I.

Table 2: Table quorum system in experiment II.

make the following general observations. Gather-quorum algorithms have the advantage that hosts only need to hear from a quorum, (which is usually much smaller than the entire universe of hosts). Therefore, in cases where availability is not an issue, gather-quorum algorithms strictly dominate gather-all algorithms in running time. However, Gather-all algorithms do not fail by definition, since hosts only need to hear from hosts that are currently alive regardless of how many there are. Therefore, in cases of high failure rates (where no quorum exists), gather-all algorithms succeed while gather-quorum algorithms fail.

4.1 Complete probe sets

Figure 3: Experiment II: decentralized algorithms and complete probe sets.

Even though we had several host failures and network partitions during both experiments, they were not frequent enough to bring down the entire quorum system being used; Tables 3 and 4 show the percentage of runs that failed for different probe sets. Therefore, for the duration of both experiments, the probability of a host not finding a live quorum was negligible regardless of the quorum system being used (table or majority).

In a gather-quorum algorithm, each host waits to hear from a set of hosts that is a strict subset of the set of hosts it has to wait to hear from in a gather-all algorithm. The cost of waiting for additional responses is very high due to the great variability in link latencies over the Internet, both across different links, and at different times on the same link. When running TCP over links with non-negligible loss rates, the latency exhibits even greater variability due to TCP’s large timeouts and exponential backoff mechanism. This explains why we observe such overwhelming differences between the running times of gather-quorum and gather-all algorithms. These results are shown in Figures 2 and 3. Figure 2 shows results of the leader algorithm at four different initiators in Experiment I. Figure 3 shows the results of decentralized algorithms (quorum-to-quorum and all-to-all) initiated at MIT during Experiment II. The figures show the cumulative distributions of the running time of each primitive.

However, if we try to determine which quorum system is better, the answer is not as clear. Although we have chosen a table that improves performance at most hosts, with 27 hosts, there is a significant chance that no table exists which is optimal for every host in the system. Our results from this experiment indicate that for most initiators, table-based quorums outperform majority. For each of these initiators, there exists a subset of the optimal 14-host majority for that initiator that forms a quorum based on Table 1. However, for AUS, no such subset exists, which explains why majority provides superior performance in this case. Moreover, majority is more available than table, in the sense that it is more likely to have at least one live quorum [16]. (Although finding this quorum may require using a complete probe set or probing more hosts after a timeout).

4.2 Minimal probe sets

Figure 5: Experiment II: decentralized algorithms and minimal probe sets.

With minimal probe sets we have a different story. In this case, every host deals with a particular quorum instead of any quorum. In the majority quorum system, the probe set is composed of the closest majority to the initiator; in the table quorum system, the probe set is composed of the first row of the table (choosing probe sets and quorum systems is discussed in more detail in Section 5). Even though this particular quorum is usually chosen because it usually has the best availability, its failure probability is higher than that of the entire quorum system. A quorum fails if any of its hosts fail. Since every host has a nonzero probability of failure, the probability of a quorum failing grows exponentially with the size of that quorum. This means that the probability that a particular quorum fails in the majority quorum system is higher than in the table quorum system. If we
look at the graphs in Figures 4 and 5, it is clear that the running time of algorithms that use the table-based quorum system is by far superior to algorithms using majority and gather-all algorithms. This is the case for two reasons. First, since the minimal probe set in the table quorum system is the first row of the table, the number of hosts involved in the algorithm is very small relative to majority and gather-all algorithms. Second, in a well chosen table, hosts in the first row usually have the highest availability.

If we look at the curves for majority and gather-all algorithms, we find that majority dominates in the low latency region and performs worse in the high latency region. This happens for two reasons. First, since the number of hosts in the minimal probe set in the majority quorum system is approximately half of the total number of hosts, the running time will be lower than with the gather-all algorithm when every host in the probe set is available. Second, since the quorum fails whenever any of its elements fail, a majority becomes unavailable for a significant amount of time during which gather-all algorithms continue to succeed (at higher latencies of course).

5. THE SIZE OF THE PROBE SET

We now analyze the relationship between the size of the probe set and the running time of the quorum-leader and quorum-to-quorum algorithms (using both majority and table quorum systems). In particular, we look at how this relationship is influenced by network dynamics (lost messages, latency variation and failures) and the type of algorithm. The results presented in Section 5.1 are from Experiment I. Section 5.2 presents results from Experiment II.

5.1 Quorum-Leader

5.1.1 Majority

Since we have a total of 27 hosts, a majority consists of at least 14 hosts (including the initiator). In this section we look at improvements in the running time as the size of the probe set increases from 14 to 27. For a given initiator, for each instant of leader it initiated, we sort the hosts in ascending order based on the response time for that instant, and assign each host a rank that corresponds to its position in the sorted list. Based on those average ranks we sort the hosts in ascending order. Based on that order we rank hosts from 2 to 27 (of course the initiator being 1). This rank represents the order in which we add hosts to the probe set of each initiator. In general there are several arguments to be made for probe sets that are larger than the minimum. First, because of dynamic nature of the Internet (changing routes, lost messages), the 14 hosts closest to the initiator do not stay the same for the whole duration of the experiment. Message loss, in particular, plays a significant role: any lost message from any of the 14 hosts in the minimal probe set, with high probability, increases the running time of the algorithm beyond the RTT of the 15th host. And no matter how reliable links between the initiator and its closest 14 hosts, they still
Table 3: Experiment I: percentage of failed runs of quorum-leader algorithm using the majority quorum system and different probe sets.

<table>
<thead>
<tr>
<th>probe size</th>
<th>MIT</th>
<th>AUS</th>
<th>UT2</th>
<th>ISR1</th>
<th>TW</th>
<th>KR</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>8%</td>
<td>32%</td>
<td>24%</td>
<td>15%</td>
<td>13%</td>
<td>18%</td>
</tr>
<tr>
<td>15</td>
<td>2%</td>
<td>3%</td>
<td>0%</td>
<td>4%</td>
<td>7%</td>
<td>4%</td>
</tr>
<tr>
<td>16</td>
<td>0%</td>
<td>1%</td>
<td>0%</td>
<td>3%</td>
<td>7%</td>
<td>2%</td>
</tr>
<tr>
<td>17</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>2%</td>
</tr>
<tr>
<td>18</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>2%</td>
</tr>
<tr>
<td>19</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>2%</td>
</tr>
<tr>
<td>20</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>2%</td>
</tr>
<tr>
<td>21</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>2%</td>
</tr>
<tr>
<td>22</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>2%</td>
</tr>
<tr>
<td>23</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>2%</td>
</tr>
<tr>
<td>24</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>3%</td>
</tr>
<tr>
<td>25</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>4%</td>
</tr>
<tr>
<td>26</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>4%</td>
</tr>
<tr>
<td>27</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>4%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 4: Experiment I: percentage of failed runs of quorum-leader algorithm using the table quorum system and different probe sets.

<table>
<thead>
<tr>
<th>rows probed</th>
<th>MIT</th>
<th>AUS</th>
<th>UT2</th>
<th>ISR1</th>
<th>TW</th>
<th>KR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>4%</td>
<td>1%</td>
<td>4%</td>
</tr>
<tr>
<td>2</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>0%</td>
<td>2%</td>
</tr>
<tr>
<td>3</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>0%</td>
<td>2%</td>
</tr>
<tr>
<td>4</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>0%</td>
<td>2%</td>
</tr>
<tr>
<td>5</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>0%</td>
<td>2%</td>
</tr>
</tbody>
</table>

have nonzero probabilities of dropping messages. Second, some hosts also fail during the experiment. However, since failures during the experiment were not very frequent and network partitions were very short, the first factor plays a bigger role in our analysis especially since most failures and network partitions effect all initiators equally while variations in TCP latencies are different for different hosts.

In general every initiator except TW had low varying and highly reliable links (loss rates of 10% or less) to most hosts. TW on the other hand, had many links with highly variable latencies and loss rates of 25% or more (most of the hosts with bad connections to TW were ISPs in North America). For initiators other than TW, our results indicate that optimal performance is achieved with a probe set that contains 19 hosts. The improvements in performance gained by increasing the size of the probe set beyond 19 hosts are negligible. The highest improvements occur when the size of the probe set is increased to 15 and then 16. The marginal rate of return continues to decrease with the number of hosts and diminishes when this number is increased beyond 19. Figure 6 illustrates this observation by showing the cumulative distribution of the running time of runs initiated at AUS and ISR1 for probe sets with different sizes. However, this is not the case with TW. The performance continues to improve significantly as we increase the number of hosts probed by TW to 27. The TW graphs shown in the same figure, show the cumulative distribution of runs initiated at TW for different numbers of hosts. In order to get a better understanding to what is going on with TW, we refer to Table 5 which shows the link characteristics as measured by “ping” from TW to other hosts in the system (the column labeled “TCP connectivity” refers to the percentage of time the TCP connection was up). We can see that hosts that have loss rates of 25% or more to TW also have the highest average latencies. At first glance, it would appear that the problems of high message loss are compounded by the high latency. However, we see that these hosts have the smallest minimum RTTs (highlighted in the table), which means that the best case involves these hosts. We also notice that the standard deviation is highest for those links, which means that the low latency runs are more probable. The probability of getting good running times increases as we send to more of these hosts.

Now the question remains how well can we estimate the optimal size of the probe set given our knowledge of the network characteristics. For a given to probe set, how accurately can we predict the percentage of runs below a certain threshold based on what we know about the TCP latency distributions? As an example, we will use TW and see how well we can approximate the percentage of runs below 1 second for probe sets the contain 14, 15, 16, and 17 hosts. The last column in Table 5 shows the percentage of TCP round trips under 1 second for each link. For simplicity, we will assume that different messages travel through the network independently (not entirely true). We will also restrict our attention to links in which the percentage of TCP RTTs under 1 second is less than 90%.

The probe set of size 14, which contains the first 14 hosts listed in Table 5, includes CA1, NY, UT2, MA2. the percentages of TCP RTTs under 1 second from TW to these four are 58%, 63%, 58%, and 59% respectively. Based on the

5AUS is inside a firewall that filters ICMP traffic.
assumptions we have made, we can estimate the probability that TW hears from a majority \( \Pr_{14} \) as follows:

\[
\Pr_{14} = .63 \times .58 \times .58 \times .59
\]

\[
\approx .13
\]

Indeed the value we measured was .16, which is close to the estimated value. Now how much improvement in the running time can we expect from adding the 15th host (NC) to the probe set? This is the same as the probability of exactly one out of the four lossy hosts failing to make the 1 second threshold and NC succeeding.

\[
Pr_{15} - Pr_{14} = .63(\cdot .37 \times .58 \times .58 \times .59
+ .2 \times .63 \times .42 \times .58 \times .59
+ .63 \times .58 \times .58 \times .41)
\approx .21
\]

The value that we measured was .20. Similarly the improvement we can expect from increasing the size of the probe set from 15 to 16 is the probability of exactly two out of the five lossy hosts failing to make the 1 second threshold and ISR2 succeeding.

\[
Pr_{16} - Pr_{15} = .97(2 \times .41 \times .37 \times .63 \times .58 \times .58
+ .59 \times .37 \times .37 \times .58 \times .58
+ .59 \times .63 \times .63 \times .42 \times .42
+ 4 \times .59 \times .37 \times .63 \times .42 \times .58)
\approx .33
\]

The measured value was .26. The improvement we expect from increasing the size of the probe set from 16 to 17 is the probability of exactly three out of the five lossy hosts failing (or exactly two succeeding) to make the 1 second threshold and UT1 succeeding.

\[
Pr_{17} - Pr_{16} = .55(2 \times .59 \times .37 \times .63 \times .42 \times .42
+ .41 \times .37 \times .37 \times .58 \times .58
+ .41 \times .63 \times .63 \times .42 \times .42
+ .41 \times .37 \times .63 \times .42 \times .58)
\approx .12
\]

The value we measures was .9. More generally if we repeat this computation over a latency interval we get a reasonable approximation of the cumulative distribution over that interval. Figure 7 shows the estimated version of the cumulative distribution of running time at TW which appears in Figure 6. Figure 8 gives a closer look at the margin of error in our estimation. We notice that for small probe sets, our estimator tends to underestimate, and overestimates for larger probe sets. This is a direct result of the independence assumption that we have made. In reality, packet drops at the same node are correlated with a positive correlation factor. The probability of an intersection of events decreases when these events are independent as opposed to positively correlated. Whereas the probability of a union of events increases when these events are independent as opposed to positively correlated.
Figure 6: Experiment I: local running time of quorum-leader with majority quorums and different probe set sizes.
Figure 8: Experiment I: predicted vs. measured local running times of quorum-leader with majority quorums and different probe set sizes at TW.
Table 5: Link characteristics from TW to other hosts during experiment I.

<table>
<thead>
<tr>
<th>Host</th>
<th>Loss Rate</th>
<th>Avg. Ping RTT</th>
<th>STD</th>
<th>Min. Ping RTT</th>
<th>TCP Connectivity</th>
<th>% TCP RTTs under 1 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>TW</td>
<td>0%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>UCSD</td>
<td>3%</td>
<td>232</td>
<td>25</td>
<td>198</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td>Emulab</td>
<td>3%</td>
<td>238</td>
<td>26</td>
<td>216</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td>NYU</td>
<td>3%</td>
<td>273</td>
<td>22</td>
<td>251</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td>MIT</td>
<td>4%</td>
<td>303</td>
<td>398</td>
<td>256</td>
<td>100%</td>
<td>90%</td>
</tr>
<tr>
<td>CMU</td>
<td>4%</td>
<td>289</td>
<td>41</td>
<td>254</td>
<td>100%</td>
<td>90%</td>
</tr>
<tr>
<td>CU</td>
<td>3%</td>
<td>339</td>
<td>127</td>
<td>247</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td>AUS</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>99%</td>
<td>96%</td>
</tr>
<tr>
<td>NL</td>
<td>3%</td>
<td>361</td>
<td>23</td>
<td>339</td>
<td>100%</td>
<td>96%</td>
</tr>
<tr>
<td>CA1</td>
<td>31%</td>
<td>482</td>
<td>626</td>
<td>174</td>
<td>95%</td>
<td>58%</td>
</tr>
<tr>
<td>NY</td>
<td>32%</td>
<td>445</td>
<td>853</td>
<td>234</td>
<td>96%</td>
<td>63%</td>
</tr>
<tr>
<td>SWD</td>
<td>3%</td>
<td>399</td>
<td>59</td>
<td>371</td>
<td>100%</td>
<td>96%</td>
</tr>
<tr>
<td>UT2</td>
<td>30%</td>
<td>743</td>
<td>1523</td>
<td>177</td>
<td>96%</td>
<td>58%</td>
</tr>
<tr>
<td>MA2</td>
<td>28%</td>
<td>742</td>
<td>1517</td>
<td>230</td>
<td>94%</td>
<td>59%</td>
</tr>
<tr>
<td>NC</td>
<td>32%</td>
<td>465</td>
<td>616</td>
<td>255</td>
<td>90%</td>
<td>63%</td>
</tr>
<tr>
<td>ISR2</td>
<td>3%</td>
<td>424</td>
<td>70</td>
<td>400</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td>UT1</td>
<td>27%</td>
<td>979</td>
<td>1847</td>
<td>189</td>
<td>96%</td>
<td>55%</td>
</tr>
<tr>
<td>MA1</td>
<td>29%</td>
<td>645</td>
<td>712</td>
<td>238</td>
<td>96%</td>
<td>57%</td>
</tr>
<tr>
<td>ISR1</td>
<td>4%</td>
<td>551</td>
<td>2682</td>
<td>400</td>
<td>100%</td>
<td>94%</td>
</tr>
<tr>
<td>CA2</td>
<td>30%</td>
<td>606</td>
<td>1094</td>
<td>779</td>
<td>69%</td>
<td>45%</td>
</tr>
<tr>
<td>CR</td>
<td>3%</td>
<td>447</td>
<td>29</td>
<td>419</td>
<td>96%</td>
<td>93%</td>
</tr>
<tr>
<td>CND</td>
<td>35%</td>
<td>686</td>
<td>834</td>
<td>212</td>
<td>93%</td>
<td>51%</td>
</tr>
<tr>
<td>MA3</td>
<td>32%</td>
<td>774</td>
<td>1473</td>
<td>241</td>
<td>96%</td>
<td>54%</td>
</tr>
<tr>
<td>NZ</td>
<td>35%</td>
<td>636</td>
<td>830</td>
<td>274</td>
<td>91%</td>
<td>43%</td>
</tr>
<tr>
<td>KR</td>
<td>11%</td>
<td>357</td>
<td>163</td>
<td>200</td>
<td>42%</td>
<td>40%</td>
</tr>
<tr>
<td>CA3</td>
<td>32%</td>
<td>1047</td>
<td>2091</td>
<td>178</td>
<td>15%</td>
<td>10%</td>
</tr>
<tr>
<td>Swiss</td>
<td>3%</td>
<td>384</td>
<td>22</td>
<td>362</td>
<td>15%</td>
<td>15%</td>
</tr>
</tbody>
</table>

Figure 7: Experiment I: predicted local running times of quorum-leader with majority quorums and different probe set sizes at TW.

In this section, we analyze the performance based on Table 1 which contains 5 rows, with each row containing 5-6 hosts. In this section we look at improvements in the running time of table as the number of table-rows in the probe set increases from 1 to 5. In order to improve performance, we picked the table rows as follows:

- In the first row, we put hosts located at North American universities which were up for the duration of the experiment.
- Our ping traces indicate that hosts located in Europe and Israel are connected to each other by low latency and low loss rate links. Therefore, in order to improve the performance for these hosts, we placed them in the second row (except Swiss which was under firewall restriction for a portion of the experiment).
- In the third row, we put five other hosts in North America that did not crash during the experiment.
- We filled out the last two rows with the remaining hosts.

Every quorum in this setting must include at least one host in the first row. Therefore, while sending to more rows may improve availability in the case of some first row hosts failing, the performance is eventually constrained by the first row. The graphs in Figure 9 shows the performance for different initiators. Depending on where the initiators are located, they see different gains at different row numbers. Note especially that difference in performance between probing one table row and probing all rows is smaller than
Figure 9: Experiment I: local running time of quorum-leader with table quorums and different probe set sizes (measured in table rows).
the difference in performance between probing 14 hosts and probing all hosts (in the majority system). This is usually the case since the first row is filled with hosts that were up for most of the experiment and had reliable connections to other hosts.

5.2 Quorum-to-Quorum

In this section, in order to get meaningful results, we need to look at the overall running times because of the asymmetry of the two phases of the quorum-to-quorum algorithm. For a given probe set, messages travel on the same links regardless of the initiator. Therefore, we only present results from samples initiated by MIT without loss of generality.

5.2.1 Majority

![Figure 10: Experiment II: overall running time of quorum-to-quorum with majority quorums and different probe set sizes at MIT.](image)

Since we have a total of 18 hosts, a majority consists of at least 10 hosts (including the initiator). In this section we perform the same analysis as Section 5.2.1. We only look at the samples that were initiated by MIT, and rank the hosts (as in Section 5.1.1) with respect to MIT. All the hosts in this experiment are close geographically, and communicate with each other over low-latency and low loss-rate links. In addition, throughout the experiment, only two hosts failed (CA1, UT1) and neither were ranked in the top 10. So we would expect minimal improvements in the overall running time gained by increasing the size of the probe set. However, there is another factor to consider in the quorum-to-quorum algorithm. Since the algorithm only terminates when a majority of hosts hear from a majority of hosts, the 10-host majority that is optimal for MIT might not be optimal for other hosts, and probing more than 10 hosts increases the probability of other hosts finding their optimal majority. So how much of a role does this play? In order to find out we compare effects of increasing the number of hosts on the overall and local running times. Figure 10 shows the cumulative distributions of the overall running time for different majorities. Figure 11 shows the cumulative distributions of the local running time for different majorities. From the figures, we observe the following: first, the improvement in performance gained by increasing the size of the probe set from 10 to 11 is “somewhat” greater in the overall running time. Second, sending to 11 hosts is near optimal in the local running time (this is not the case in the overall running time). Since the local running time in this experiment is same as the local running time of the quorum-leader algorithm, the results suggest that optimal running time can be reached with a smaller probe set in the case of quorum-leader than quorum-to-quorum.

5.2.2 Table

![Figure 11: Experiment II: local running time of quorum-to-quorum with majority quorums and different probe set sizes at MIT.](image)

In this section, we analyzed the performance based on Table 2 which contains 4 rows, with each row containing 4-5 hosts. We placed hosts located at universities in the east cost in the first row of the table. In the second row, we put two west cost university hosts and two west coast hosts located at ISPs. We filled the third with the rest of the east cost hosts and put the remaining hosts in the last row. In this particular case, since the hosts in the first row are geographically close to each other and were up for the entire duration of the experiment, the first row was the optimal quorum for every host in the first row. As a result we did not see a significant performance improvement gained by sending to more table rows. Figure 12 illustrates these results.

![Figure 12: Experiment II: overall running time of quorum-to-quorum with table quorums and different probe set sizes (measured in table rows) at MIT.](image)
6. CONCLUSIONS

We have studied the performance of quorum-based systems over the Internet. An important conclusion from our study is that quorum-based systems can actually perform quite well over the Internet. We have compared the running time of quorum-based algorithms to that of algorithms that communicate with all live hosts (gather-all algorithms). We found that quorum-based algorithms, when using large enough probe sets, dramatically out-perform gather-all algorithms. This is especially true for situations in which links are very lossy, where gather-all algorithms completely fail apart and gather-quorum algorithms thrive. We observed this, e.g., in the host in TW in our experiment, where the gather-all algorithm completed in under 4 seconds in only 40% of the runs, and gather-quorum completed in less than 500 ms. in over 95% of the runs. This observation is quite general: it holds for both leader-based and decentralized algorithms, and for the two very different quorum systems we experimented with.

We have studied the impact of the size of the probe set on the performance of quorum-based systems. We found that a small increase in probe set size beyond the minimal one can go a long way in terms of increasing performance as well as availability.

Acknowledgments

Will be added in the final version.

7. REFERENCES