Evaluating the Running Time of a Communication Round over the Internet*

Omar Bakr
MIT
ombakr@mit.edu

Idit Keidar
The Technion and MIT
idish@ee.technion.ac.il

ABSTRACT

We study the running time of distributed algorithms deployed in a widely distributed setting over the Internet using TCP. We consider a simple primitive that corresponds to a *communication round* in which every host sends information to every other host; this primitive occurs in numerous distributed algorithms. We experiment with four algorithms that typically implement this primitive. We run our experiments on ten hosts at geographically disperse locations over the Internet. We observe that message loss has a large impact on algorithm running times, which causes leader-based algorithms to usually outperform decentralized ones.

1. INTRODUCTION

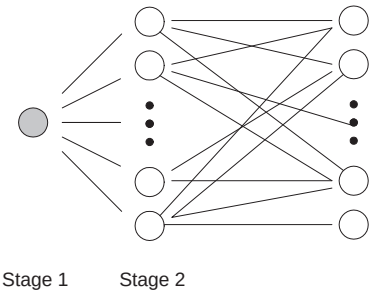
It is challenging to predict the end-to-end running time of a distributed algorithm running over TCP/IP in a wide-area setting. It is also often not obvious which algorithm would work best in a given setting. E.g., would a decentralized algorithm outperform a leader-based one? Answering such questions is difficult for a number of reasons. Firstly, because end-to-end Internet performance itself is extremely hard to analyze, predict, and simulate [7]. Secondly, end-to-end performance observed on the Internet exhibits great diversity [17, 26], and thus different algorithms can prove more effective for different topologies, and also for different time periods on the same topology. Finally, different algorithms can prove better under different performance metrics.

In this paper, we study the running time of distributed algorithms over the Internet. Our experiments span ten hosts, widely distributed over the Internet – in Korea, Taiwan, the Netherlands, and several hosts across the US, some at academic institutions and others on commercial ISP networks. We present data that was gathered over several weeks. 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 does not generate excessive traffic at times of congestion, and because firewalls at some of the hosts we use block UDP traffic.

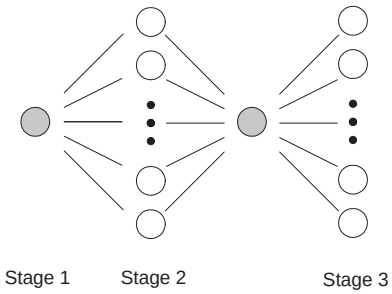
We consider a fixed set of hosts engaged in a distributed algorithm. We evaluate a simple primitive that propagates a small amount of information from every host to all other hosts that are connected to it. This primitive corresponds to a *communication round* executed by a distributed algorithm. The primitive can be initiated by any one of the hosts, called the *initiator*, and it terminates once information from every host has propagated to all of the hosts. Communication rounds of this sort are employed by many different algorithms and systems, e.g., Byzantine agreement [16], atomic commit [8, 23, 11], state-machine replication [15], group membership [13], and updates of routing tables. Thus, our study has broad applicability. We evaluate the following commonly used algorithms implementing the primitive.

- *all-to-all*, where the initiator sends a message to all other hosts, and each host that learns that the algorithm has been initiated sends messages to all the other hosts. This algorithm is structured like decentralized two-phase commit, some group membership algorithms (e.g., [13]), and the first phases in decentralized three-phase commit algorithms, (e.g., [23, 9]). The algorithm flow is depicted in Figure 1(a).
- *leader*, where the initiator acts as the leader. After the initiator sends a message to all other hosts, the hosts respond by sending messages to the leader. The leader *aggregates* the information from all the hosts, and sends a message summarizing all the inputs to all the hosts. This algorithm is structured like two-phase commit [8], and like the first two of three communication phases in three-phase commit algorithms, e.g., [23, 11]. The algorithm flow is depicted in Figure 1(b).
- *secondary leader*, where a designated host (different from the initiator) acts as the leader. The initiator sends a message to the leader, which then initiates the leader-based algorithm. The algorithm flow is depicted in Figure 1(c). This algorithm structure is essentially a spanning tree of depth one, with the secondary leader being the root and all other hosts being leaves. Since our system consists of ten hosts, we did not see the need for a deeper spanning tree.
- *logical ring*, where messages propagate along the edges of a logical ring. This algorithm structure occurs in

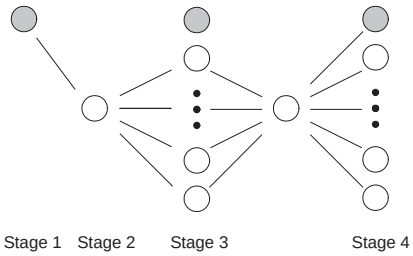
*This work was supported by Air Force Aerospace Research (OSR) contract F49620-00-1-0097 and MURI award F49620-00-1-0327, and Nippon Telegraph and Telephone (NTT) grant MIT9904-12.



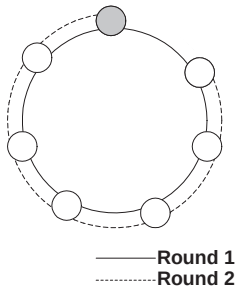
(a) The all-to-all algorithm.



(b) The leader algorithm.



(c) The secondary leader algorithm.



(d) The ring algorithm.

Figure 1: The message flow of the four algorithms. Initiator shown in gray.

several group communication systems, e.g., [1]. The algorithm flow is depicted in Figure 1(d).

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. Using a 2-level hierarchy, algorithms of the sort we consider can be made to work effectively with several hundreds of processes. Such a hierarchy is used, e.g., in [13, 10], where the top level of the hierarchy consists of 5–20 representatives (servers) at disperse geographical locations. Each representative gathers information from and propagates information to processes that are proximate to it. Algorithms like those considered here are typically run among the representatives. Thus, our study is applicable to systems that implement scalability in this manner. Our study is however not applicable to systems that implement massive scalability, e.g., using gossip-based algorithms.

We measure the *overall* running time of an algorithm from the time it starts at some host until it terminates at all hosts, as well as the *local* running time at a given host.

The typical theoretical metric used to analyze the running time of distributed algorithms is the number of message exchange rounds the algorithm performs, or the number of *communication steps* in case of a non-synchronous system (e.g., [21, 12, 13]). According to this metric, we get the following overall running times: 2 communication steps for the all-to-all algorithm; 3 communication steps for the leader algorithm; 4 communication steps for secondary leader; and $2n-1$ steps for the ring algorithm in a system with n hosts. In contrast to what this metric suggests, in Section 5 we observe that in certain settings the secondary leader algorithm achieves the best overall running time, whereas all-to-all often performs the worst. The running time of ring was usually less than double the running times of the other algorithms.

Why does the communication step metric fail to capture actual algorithm behavior over the Internet? First, not all communication steps have the same cost, e.g., a message from MIT to Cornell can arrive within 20 ms., while a message from MIT to Taiwan may take 125 ms. Second, the latency on TCP links depends not only on the underlying message latency, but also on the loss rate. If a message sent over a TCP link is lost, the message is retransmitted after a timeout which is larger than the average round-trip time (RTT) on the link. Therefore, if one algorithm message is lost, the algorithm's overall running time can be more than doubled. Since algorithms that exchange less messages are less susceptible to message loss, they are more likely to perform well when loss rates are high. This explains why the overall running time of all-to-all is miserable in the presence of lossy links. Additionally, message latencies and loss rates on different communication paths on the Internet often do not preserve the triangle inequality [20, 13, 3] because the routing policies of Internet routers often do not choose an optimal path between two hosts. This explains why secondary leader can achieve better performance by refraining from sending messages on very lossy or slow paths.

We analyze our experimental results, and explain the observed algorithm running times in terms of the underlying network characteristics – latency and loss rates. Due to the great variability of running times, the average running time is not indicative of an algorithm's typical behavior. We therefore focus on the *distribution* of running times.

The communication step metric is widely used due to its ease-of-use. Several other performance models, e.g., [6, 25, 18], have been used to analyze distributed or parallel algorithms (cf. Section 2). However, these do not realistically model algorithm behavior over the Internet. At the end of this paper, we suggest a refinement to the standard metric, which gives a more realistic account of an algorithm's efficiency, and at the same time is easy to work with.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 describes the experiment setup and methodology. Section 4 presents the mathematical model we use to analyze our results. The following two sections present and analyze experimental results: Section 5 discusses the impact of message loss on the running times of the all-to-all, leader, and secondary leader algorithms. Section 6 discusses the impact of latency; it studies the all-to-all, leader, and ring algorithms. Section 7 concludes the paper and suggests an alternative performance metric.

2. RELATED WORK

Obtaining data on different aspects of Internet communication is an emerging research direction. Some active research in this area focuses on measuring and analyzing the constancy of Internet path characteristics such as routing, loss, and throughput [26, 17]. Such research focuses primarily on point-to-point communication, and not on the performance of distributed algorithms. Another related project, pursued by Chandra et al. [4], studies the nature of communication failures – duration and location – and how they effect the end-to-end availability of wide-area services. Another study, by Amir and Wool [2], evaluates the availability of different quorum systems over the Internet. These research efforts are orthogonal and complementary to ours.

The fact that Internet routing often does not select optimal paths was previously observed by a number of projects – Detour [20, 19], Moshe [13], and RON [3]. These projects construct overlay networks and improve performance by routing messages over these overlays on better paths than would be chosen by Internet routing. In contrast, we neither assume an overlay infrastructure, nor route messages through hosts that are not participating in the current instance of the algorithm. Moreover, the aforementioned projects use overlays in order to find better paths for *point-to-point* communication only. When an overlay is used at the routing level, as in these projects, messages from the same source that are routed through the same host to different destinations are not merged into a single message. E.g., consider the all-to-all algorithm running over an overlay that routes messages from Taiwan to the Netherlands via Cornell. Taiwan would send identical messages to Cornell and the Netherlands, which would be sent as two separate messages on the link from Taiwan to Cornell. Likewise, the overlay would not combine the information sent from Taiwan and Cornell to the Netherlands into a single message. Such sending of multiple messages increases the probability of some message being lost, which increases the average running time.

Another line of research focuses on providing a theoretical framework for predicting and evaluating the performance of parallel and distributed algorithms. A number of papers, e.g. [6, 25, 18, 22], focus on settings where message processing overhead is significant, and show that this favors algorithms that send fewer messages. While our results also illustrate the advantage of sending fewer messages, the reasons for this are different: in our setting, it is due to high variability of message latency (due to loss) rather than processing overhead, which is negligible in our setting. The conclusions from such studies do not, in general, apply to our setting. E.g., leader has a high processing overhead (at the leader), but this does not hamper it's performance in our setting. Moreover, these analyses assume that the evaluated algorithm is the only source of overhead in the system. In contrast, over the Internet, the evaluated algorithms have little impact on the total overhead of the system.

3. METHODOLOGY

We use the following hosts in our experiments. Universities in the US: *MIT*, at the Massachusetts Institute of Technology, Cambridge, MA; *UCSD*, at the University of California San Diego; *CU*, at Cornell University, NY; *NYU*, at New York University, NY; and *Emulab*, at the University of Utah. Hosts at US commercial ISP networks: *CA* in California and *UT1* and *UT2* in Utah. International hosts: *KR* in Korea, *TW*, at National Taiwan University in Taiwan; and *NL*, at Vrije University in the Netherlands. All the hosts run either FreeBSD or Linux operating systems.

3.1 Server Implementation

At every host we run a server, 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. 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, [24, Ch. 19]). The system implements asynchronous I/O using threads. Every 5 minutes, each server 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. Here, we present performance results only for periods during which all the hosts had identical perceptions of the set of connected hosts. 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 the four algorithms. The server periodically invokes each algorithm: 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 from each host to each of the other hosts, sending a ping probe once a minute, in order to track the latency and loss rate of the underlying network. The ping process is also monitored by a `crontab`.

3.2 Running Times and Clock Skews

We use two measures of running time:

- The *local running time* of a session at a particular host is the clock time elapsing from when this host begins this session and until the same host terminates the session. Where we present performance measurements, we give local running times at the initiator only.
- 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.

Each host writes to log its starting time and termination time for each session, 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.

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 Δ_{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, $\Delta_{BA} = l_{AB} - d_{AB}$. Therefore, $\Delta_{AB} - \Delta_{BA}$ is, on average, $2d_{AB}$. We approximate the clock difference between A and B as:

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

This approximation method has some limitations: since messages are exchanged over TCP, 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. 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 (15 minutes long).

The next step is to fix a 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 chose 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 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 similar. 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.

4. THE MATH: RUNNING TIME DISTRIBUTION OVER TCP/IP

We now explain the mathematical model that underlies the analysis of the experimental results in this paper.

After TCP sends a message, it waits for an acknowledgement. If an acknowledgement does not arrive for a designated retransmission time-out, TCP retransmits the message. TCP's initial retransmission time-out is the estimated average RTT on the link plus four times the mean deviation of the RTT, where both the average and the mean deviation are computed over recent values. If the second copy is also lost, TCP waits twice the amount of time it waited before retransmitting the first lost copy, and this continues to grow exponentially with number of lost copies. [24, Ch. 21]

We estimate the distribution of the TCP latency based on the underlying link latency d and loss probability p. Assume first that d is half the RTT, that losses are independent, and that the latency does not vary, so the RTT's mean deviation is 0. Then the TCP latency is d with probability $1-p$, $3d$ with probability $p(1-p)$, $7d$ with probability $p^2(1-p)$, and so on. This is a rough estimate, as it does not address variations in latency and loss. Correlated loss causes the first peak (at latency d) to occur with higher probability, and causes the tail of the distribution to be sparser; this will be most significant on links with high loss rates. A high variation of latency will shift all the peaks except the first.

We use this estimate to analyze the distribution of the running time of a stage of an algorithm. Let p_i be the probability that the latency of a message sent on link i is at most D (as computed above). Then the probability that an algorithm stage takes at most D time is the product of the probabilities p_i for all the links traversed in this stage. More generally, the running time of a stage is a random variable representing the maximum value of the random variables representing the TCP link latencies, with distributions defined by the RTT and loss rate as explained above. As the number of random variables over which the maximum is computed grows, the expected maximum value increases. This explains why all-to-all, which sends $O(n^2)$ messages in each stage performs much worse than leader, which sends $O(n)$. A similar observation was made in [18].

5. THE EFFECT OF MESSAGE LOSS

This section presents two experiments, each of which lasted three and a half days. Ring was not tested in these experiments. Each of the other three algorithms was initiated by each of the hosts every 7.5 minutes on average, and in total, roughly 650 times. Section 5.1, presents Experiment I, in which the TW host had two links with very high loss rates. We then excluded the TW host, and ran Experiment II, which we present in Section 5.2.

5.1 Experiment I

The NL and UT1 hosts were excluded from this experiment. Table 1 presents the average RTT and loss rate from every host to every other host during the experiment, as observed by ping. The loss rates from TW to UT2 and CA are very high (37% and 42%, resp.), and all the other loss rates are up to 8%. Losses sometimes occur in bursts, where for a period of several minutes all the messages sent on a particular link are lost. The latencies generally vary less, but occasionally we observe periods during which the latency is significantly higher than average.

In this experiment MIT serves as the secondary leader for TW, KR, CU, UT2, NYU, and Emulab. Emulab is the secondary leader for the rest. We chose secondary leaders that had relatively reliable links to all hosts. We used secondary leaders for all hosts in order to have a meaningful comparison. In practice, secondary leaders would only be used for hosts that have poor links.

Due to occasional loss bursts and TCP's exponential backoff, some running times are very high (several minutes long). Thus, the average running time is not representative. In Table 2, we present statistical data about the running times, both overall and local, of the three algorithms. We present the average running time (in milliseconds) taken over runs that complete within 2 seconds. Most runs that experience no more than 2 consecutive losses are included in this average. In Figure 2, we present histograms of the distribution

From	To	KR	TW	MIT	UCSD	CU	NYU	CA	UT2	Emulab
KR	Avg. RTT		387	291	272	265	267	168	479	258
	Loss Rate	—	6%	7%	2%	0%	0%	1%	1%	2%
TW	Avg. RTT	388		243	177	211	220	221	267	186
	Loss Rate	5%	—	8%	3%	3%	4%	41%	37%	4%
MIT	Avg. RTT	300	253		115	40	34	112	99	80
	Loss Rate	6%	8%	—	5%	6%	6%	6%	5%	5%
UCSD	Avg. RTT	289	195	125		91	102	42	105	61
	Loss Rate	2%	4%	5%	—	0%	0%	0%	0%	0%
CU	Avg. RTT	266	211	47	73		9	88	101	47
	Loss Rate	0%	4%	5%	0%	—	0%	1%	0%	0%
NYU	Avg. RTT	267	220	39	83	9		70	78	56
	Loss Rate	0%	4%	5%	0%	0%	—	0%	0%	0%
CA	Avg. RTT	168	223	121	32	88	75		54	78
	Loss Rate	1%	42%	5%	0%	1%	0%	—	0%	0%
UT2	Avg. RTT	479	266	97	88	100	78	50		13
	Loss Rate	1%	37%	5%	0%	0%	0%	0%	—	3%
Emulab	Avg. RTT	258	186	76	48	47	57	74	14	
	Loss Rate	2%	4%	5%	0%	0%	0%	0%	3%	—

Table 1: Network characteristics during experiment I.

of overall running times under 1.3 seconds observed at three of the hosts – MIT which has no lossy links, UT2 which has one lossy link, and TW which has two. The first peak in each histogram represents the overall running time of loss-free runs. The size of the peak illustrates the percentage of the runs of that particular algorithm that were loss-free. The running times over 1 second were sparsely distributed. To illustrate this, we give the percentage of runs that exceed 2, 4, and 6 seconds in Table 2.

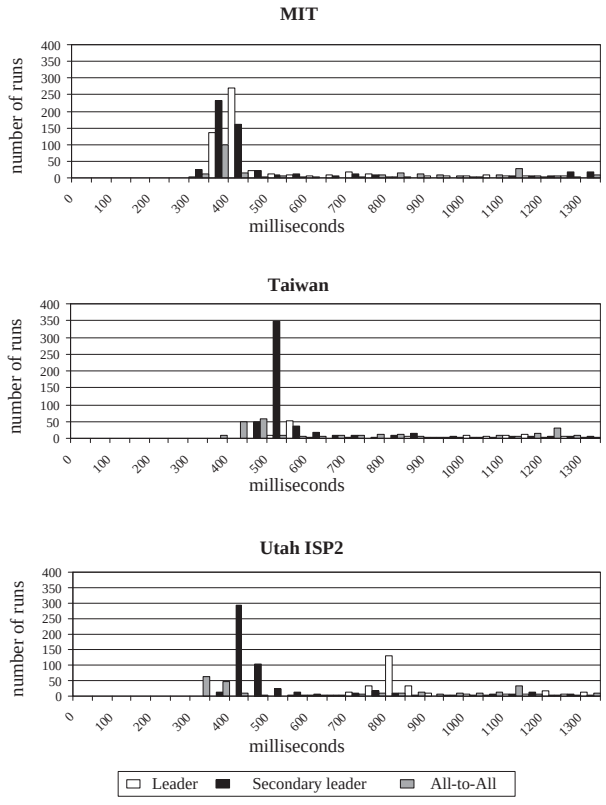


Figure 2: Histograms of overall running times, Experiment I, runs up to 1.3 seconds.

The overall running time of all-to-all is poor: less than half the runs are under 2 seconds. This is because every instance of all-to-all sends two messages over each lossy link, regardless of the initiator. Thus, most instances experience multiple consecutive losses. Leader has a better overall running time except in TW. This is because each instance of leader initiated at TW traverses each lossy link three times. Instances of leader running from other hosts traverse either one or no lossy links. At the three hosts that have lossy links (TW, UT2, and CA), secondary leader achieves the best overall performance by bypassing the lossy links.

All-to-all has the best local running time at hosts that do not have lossy links. It has a better local running time than leader due to cases in which the triangle inequality does not hold. E.g., when UT2 initiates all-to-all, CA receives the first message, on average, after 25 ms., and sends a response to all hosts. KR receives this response, on average, after 84 ms., that is, 109 ms. after UT2 sent the first message. This is earlier than the average time it takes UT2's message to get to KR (240 ms.). Therefore, KR engages in all-to-all from UT2 earlier than in leader from UT2. Similarly, when the first stage message to some host is lost, all-to-all in essence sends it also by a number of alternate paths, one which can prove more effective. This is why the local running time of all-to-all at TW is dramatically better than that of leader.

In the absence of packet loss, the overall running time of leader should be roughly three times the one-way latency on the longest link from the leader, or 1.5 times the RTT. From MIT, the longest link, to KR, has an average RTT of 300 ms. Indeed, the first peak is centered around 400–450 ms. Since all links to MIT other than from TW and KR have significantly shorter latencies (up to 115 ms.), this running time should be experienced whenever there are no losses on the TW and KR links, and at most one or two on each of the other links. Since three messages are sent on each link, and the loss rates of the longest links are 6% and 8%, the probability of no loss occurring on either of the long links is: $.94^3 * .92^3 \approx .65$. Indeed, running times up to 450 ms. occur in 429 out of 659 runs, i.e., 65%.

The longest link from TW is to KR, and its average RTT is 388 ms. Therefore, as expected, the first peak of leader

Initiator	Algorithm	All-to-all Overall	All-to-all Local	Leader Overall	Leader Local	Secondary Overall	Secondary Local
KR	Avg. (runs under 2 sec)	922	550	873	592	695	613
	% runs over 2 sec	55%	6%	15%	8%	12%	6%
	% runs over 4 sec	42%	3%	9%	4%	7%	3%
	% runs over 6 sec	37%	3%	7%	3%	5%	3%
TW	Avg. (runs under 2 sec)	866	645	1120	844	679	607
	% runs over 2 sec	54%	24%	64%	43%	13%	7%
	% runs over 4 sec	40%	19%	43%	30%	7%	4%
	% runs over 6 sec	36%	18%	37%	25%	6%	3%
MIT	Avg. (runs under 2 sec)	811	295	541	335	585	408
	% runs over 2 sec	55%	3%	13%	6%	9%	3%
	% runs over 4 sec	42%	3%	8%	4%	5%	2%
	% runs over 6 sec	37%	3%	6%	3%	4%	2%
UCSD	Avg. (runs under 2 sec)	860	328	473	332	602	420
	% runs over 2 sec	51%	2%	6%	2%	8%	3%
	% runs over 4 sec	41%	2%	5%	2%	5%	1%
	% runs over 6 sec	35%	2%	4%	2%	4%	1%
CU	Avg. (runs under 2 sec)	831	320	577	357	578	392
	% runs over 2 sec	53%	1%	6%	1%	12%	5%
	% runs over 4 sec	40%	2%	4%	1%	8%	4%
	% runs over 6 sec	35%	2%	4%	1%	6%	3%
NYU	Avg. (runs under 2 sec)	860	319	562	348	598	408
	% runs over 2 sec	54%	2%	8%	3%	12%	6%
	% runs over 4 sec	41%	3%	6%	2%	8%	3%
	% runs over 6 sec	35%	2%	5%	2%	6%	3%
CA	Avg. (runs under 2 sec)	850	450	777	553	618	450
	% runs over 2 sec	51%	17%	30%	24%	9%	3%
	% runs over 4 sec	40%	13%	21%	16%	6%	2%
	% runs over 6 sec	35%	11%	19%	15%	5%	2%
UT2	Avg. (runs under 2 sec)	872	513	1031	689	636	452
	% runs over 2 sec	52%	25%	45%	36%	13%	6%
	% runs over 4 sec	42%	21%	34%	28%	8%	4%
	% runs over 6 sec	36%	17%	29%	23%	6%	3%
Emulab	Avg. (runs under 2 sec)	844	320	544	356	633	448
	% runs over 2 sec	52%	2%	8%	3%	10%	5%
	% runs over 4 sec	41%	2%	5%	2%	6%	3%
	% runs over 6 sec	37%	2%	4%	2%	5%	2%

Table 2: Measured running times, milliseconds, experiment I.

from TW is centered around roughly 1.5 times this RTT, at the 550–600 ms. range. This peak includes only 65 of 643 runs (10%). We now explain why. First, observe that if any of the three messages sent on the link to KR or to UT2 is lost, the running time exceeds the peak. The probability of no loss on the KR link is $.95^3 \approx .86$ and the probability of no loss on the UT2 link is $.63^3 \approx .25$. Next, consider the link to CA. In the absence of losses, the response from CA to TW in the second stage arrives after about 221 ms. (the RTT), and the response from KR to TW arrives after about 388 ms. Once TW sends the final stage message to all hosts, the algorithm terminates at all hosts within half the RTT on the longest link, or roughly 194 ms. If either the first message from TW to CA or CA's response is lost once, then the response arrives roughly after 450 ms., assuming low mean deviation of RTTs. This is sufficiently close to the 388 ms. TW has to wait for KR's message, so it falls in the first peak. However, if the final stage message from TW to CA is lost, then CA terminates 332 ms. after TW sends the last message, which adds 138 ms. to the overall running time, and pushes it out of the first peak. Two losses on the link to CA always push this session away from the peak. The last message to CA is not lost with probability 58%. The probability that at most one of the previous messages is lost, and if it is lost, the retransmission is not lost, is: $.58^2 + 2 * .42 * .58^2 \approx .62$. So the probability of the first peak should be $.86 * .25 * .58 * .62 \approx .08$. This is slightly lower than the observed 10%; we hypothesize that this is due to correlated loss, which is significant here due to the high loss rates involved.

The longest link from UT2 is to KR, with an average RTT of 479 ms. Therefore, the peak is around 700–850. We now try to explain why 36% of the runs (230 of 640) are in this range. The probability of having no losses on the KR link is 97%. The link from UT2 to TW is quite erratic. Although the average RTT is 266 ms., the RTT occasionally jumps as high as 800 ms., and standard deviation of RTTs for the entire experiment period is 139 ms. In periods with low RTT variations, when the mean deviation computed by TCP is low, a run with a single loss to TW in one of the first two stages of the algorithm will fall in the first peak. A loss during a period with a high mean deviation or a loss in the last stage of the algorithm pushes the running time out of the peak. The probability that the last message on this link is not lost is 63%. We hypothesize that the mean deviation is low enough to keep us in the peak approximately half the time. With this assumption, we get that the probability of a loss in one of the first two stages not pushing us out of

the peak is 54%, and the probability of the peak should be: $.97 * .63 * .54 \approx .33$, which is close to the observed 36%.

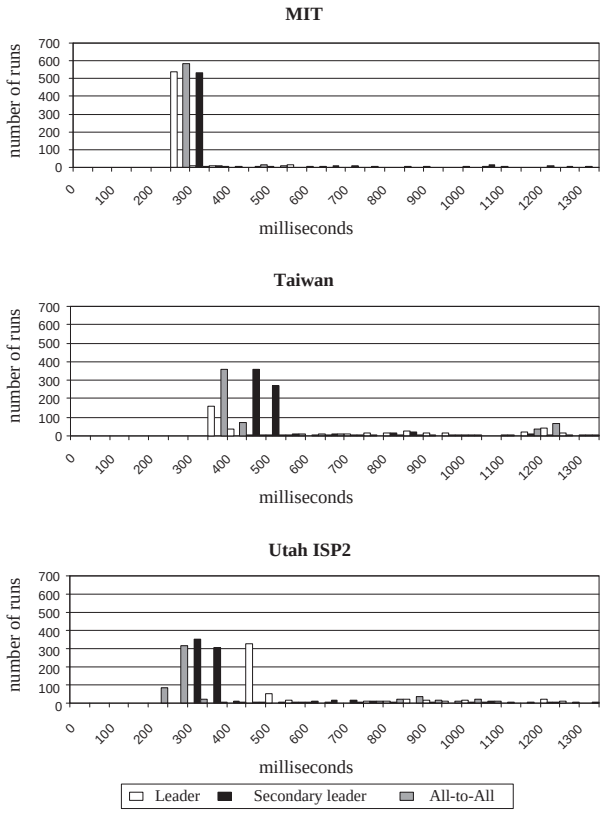


Figure 3: Histograms of local running times, Experiment I, runs up to 1.3 seconds.

Since TW uses MIT as a secondary leader, we expect secondary leader from TW to behave the same as leader initiated at MIT, with an additional delay of 120 ms. (half the RTT between TW and MIT). Indeed, the first peak is centered around 500–550, and includes roughly the same percentage of the runs as leader at MIT ($440/643 = 68\%$). All-to-all's peak exhibits the lowest overall running time, but the percentage of runs in the first peak is very low, and is the same for all initiators.

Figure 3 shows the local running times at the same hosts. The local running time for all-to-all initiated by MIT has a higher peak, as it does not involve any lossy links.

5.2 Experiment II: Excluding the Lossiest Host

We repeated the experiment above without the TW host, which was an end-point on both lossy links. We also excluded UCSD because it was overloaded at the time of the experiment, and we added UT1. The network characteristics are presented in Table 3.

The running times observed in this experiment are summarized in Table 4. In this experiment at least 88% of the runs are under 2 seconds, for all algorithms and all initiators. Even in this setting, all-to-all does not have the best overall running time for any initiator, because even the relatively low loss rates get amplified by the fact that so many messages are sent. Secondary leader works best for most hosts, except for those that are themselves optimal leaders.

When one considers the metric of local running time, we observe that the local running time of all-to-all is always superior to that of leader, regardless of the quality of links. Although they both traverse the same links the same number of times, all-to-all has the advantage that its communication stages may overlap. E.g., when the message from the initiator to one of the hosts is delayed due to loss, that host can hear from another host that the algorithm has initiated before receiving the initiator's late message. In the presence of very lossy links, secondary leader outperforms the other two algorithms both locally and globally since it is the only one that avoids the lossy links altogether.

6. THE IMPACT OF LATENCY

We now present results from Experiment III. In this experiment, we evaluated the all-to-all, leader, and ring algorithms. All the hosts except UT1 participated in this experiment. Each host ran about 510 sessions of each algorithm. Table 5 shows the network characteristics during the experiment. Table 6 summarizes the overall and local running times of the three algorithms. Table 6 gives the average running time for runs under 3 seconds, and the percentage of runs under 3 seconds. We use a threshold of 3 seconds because link latencies in this experiment are higher than in the previous two. In analyzing the results, we highlight the impact of latency on algorithm performance. In Section 6.1, we discuss the running time of the ring algorithm. In Section 6.2, we show how the highest latency link in the system affects the running time of all-to-all. In Section 6.3, we discuss the impact of a link's latency on the significance of loss on that link. Section 6.4 discusses the fact that the triangle inequality does not hold and the impact this has.

6.1 The Running Time of Ring

The message flow in the ring-based algorithm follows the following sequence where each host precedes its neighbor and the first host is the neighbor of the last: NL, Emulab, UT2, CU, NYU, KR, MIT, TW, UCSD, CA. This above ring was chosen based on latency and loss rate measurements from a previous experiment. The chosen ring is nearly optimal and the loss rates on all the ring links are low.

Ring has the highest average running time in the absence of message loss. However, ring has some nice properties: First, the ring algorithm is least affected by message loss. From the network characteristics depicted in Table 5, we observe that in the absence of message loss, the total time it takes a message to circulate around the ring twice is about 1900 ms. Unlike leader and all-to-all, the average overall running time for ring is close to this expectation. The reason for this is that ring sends the fewest messages and uses the most reliable links. Second, the choice of initiator does not have a big impact on the performance of ring, since messages travel over the same links. The only difference between initiating ring from different hosts is that the initiator only receives a message once. This explains why ring sessions initiated at KR have a slightly better overall running time since KR has the longest link. Finally, notice that ring's overall running time is not exactly twice the local running time since the second round is shorter than the first.

6.2 Latency Changes over Time

The longest links in the system were between KR and TW and KR and the NL. The latency of these two links

From	To	KR	MIT	Cornell	NYU	CA	UT2	Emulab	UT1
KR	Avg. RTT		294	261	257	165	452	275	500
	Loss Rate	—	3%	1%	3%	0%	1%	3%	1%
MIT	Avg. RTT	298		43	38	117	117	82	86
	Loss Rate	2%	—	1%	1%	1%	2%	3%	2%
Cornell	Avg. RTT	269	46		16	89	101	47	87
	Loss Rate	1%	1%	—	0%	1%	1%	3%	1%
NYU	Avg. RTT	257	38	16		69	76	60	60
	Loss Rate	3%	1%	0%	—	0%	0%	2%	1%
CA	Avg. RTT	165	115	92	75		47	79	85
	Loss Rate	0%	2%	1%	0%	—	1%	2%	1%
UT2	Avg. RTT	454	109	101	77	47		14	31
	Loss Rate	1%	2%	1%	0%	0%	—	6%	1%
Emulab	Avg. RTT	275	83	47	60	74	15		50
	Loss Rate	4%	4%	2%	2%	2%	6%	—	4%
UT1	Avg. RTT	503	82	82	60	86	30	52	
	Loss Rate	1%	1%	1%	1%	1%	1%	5%	—

Table 3: Network characteristics during experiment II.

	Algorithm:	All-to-all		Leader		Secondary	
Initiator		Overall	Local	Overall	Local	Overall	Local
KR	Avg. (runs under 2 sec)	588	509	758	551	407	388
	% runs over 2 sec	12%	7%	11%	6%	9%	4%
MIT	Avg. (runs under 2 sec)	524	278	465	296	442	311
	% runs over 2 sec	11%	4%	10%	5%	10%	6%
CU	Avg. (runs under 2 sec)	532	277	440	277	471	315
	% over 2 sec	11%	4%	9%	5%	10%	5%
NYU	Avg. (runs under 2 sec)	519	291	449	291	446	296
	% over 2 sec	12%	5%	10%	5%	10%	5%
CA	Avg. (runs under 2 sec)	535	222	378	219	486	367
	% over 2 sec	11%	5%	10%	5%	9%	6%
UT2	Avg. (runs under 2 sec)	500	265	866	498	494	383
	% over 2 sec	10%	5%	11%	6%	9%	5%
Emulab	Avg. (runs under 2 sec)	526	287	506	316	480	338
	% over 2 sec	12%	5%	9%	6%	8%	4%
UT1	Avg. (runs under 2 sec)	495	295	982	571	481	367
	% runs over 2 sec	11%	4%	11%	5%	10%	6%

Table 4: Measured overall and local running times, experiment II.

varied dramatically in the course of the experiment. We now divide the data gathered in this experiment into two periods. In the first period, the link from KR to the NL had an average RTT of 754 ms., and the link from KR to TW had an average RTT of 683 ms. In the second period, the average RTTs from KR to the NL and to TW dropped to 355 ms. and 385 ms., resp. So the average one-way message latency on the longest link dropped by 185 ms. This was the only notable difference between the two periods.

In Figure 4, we show histograms of the measured overall running times of all-to-all from all initiators during each of the two periods. The histograms show runs up to 2 seconds; this includes 23% of the runs during the longer latency period, and 60% of the runs during the shorter latency period. We observe that in the period with high latencies, the best running times are around 500 ms. In the period of low latencies, the first peak occurs at 300 ms., or roughly 200 ms. earlier, which is close to the decrease in the one-way latency on the longest link. As we see, the all-to-all algorithm from all initiators is affected by the increase in latency. In contrast, the only instances of the leader algorithm that were affected by this latency change were those initiated at TW, KR, or the NL. Other instances of the leader algorithm were unaffected. E.g., the first peak of the leader algorithm initiated at Emulab occurs at 300–350 ms. for both periods.

6.3 Latency and Loss

The loss rates from TW to CA and UT2 are 43% and 49% resp. This causes the running times of leader from these hosts to be very high (at least 44% of the runs exceed 3 seconds). The loss rates from CU to CA and UT2 are also fairly high (49% and 31% resp.). In spite of this, only 8% of the runs of leader from CU last over 3 seconds. We see that the lossy links from CU do not impact the overall running time as do the lossy links from TW. This is because the latencies of the lossy links from CU are only about one sixth the longest link latency. Therefore, even two consecutive losses on these links do not impact the overall running time.

6.4 The Triangle Inequality

The average RTT from UCSD to KR is 526 ms. and the average RTT from UCSD to CA is 49 ms., while the average RTT from CA to KR is 152 ms. Although UCSD and CA are geographically close, the average RTT from UCSD to KR is more than 3 times the average RTT from CA to KR. The latency from UCSD to KR can be reduced to less than

From	To	KR	TW	MIT	UCSD	CU	NYU	CA	UT2	Emulab	NL
KR	Avg. RTT		643	547	526	587	588	152	446	521	701
	Loss Rate	—	9%	6%	6%	4%	4%	1%	3%	7%	8%
TW	Avg. RTT	639		235	178	212	222	219	258	187	322
	Loss Rate	10%	—	4%	3%	4%	3%	43%	49%	4%	4%
MIT	Avg. RTT	549	236		97	32	28	98	78	71	150
	Loss Rate	8%	3%	—	0%	0%	0%	1%	2%	1%	0%
UCSD	Avg. RTT	526	179	96		73	84	49	91	48	172
	Loss Rate	6%	3%	0%	—	0%	0%	0%	2%	1%	0%
CU	Avg. RTT	588	211	32	73		9	85	88	47	138
	Loss Rate	4%	4%	0%	0%	—	0%	49%	31%	1%	0%
NYU	Avg. RTT	587	222	28	83	9		70	70	57	138
	Loss Rate	4%	4%	0%	0%	0%	—	0%	2%	1%	0%
CA	Avg. RTT	152	219	102	31	94	78		54	81	161
	Loss Rate	0%	42%	1%	0%	31%	0%	—	2%	4%	1%
UT2	Avg. RTT	446	262	77	91	88	71	50		13	154
	Loss Rate	3%	48%	2%	2%	31%	2%	2%	—	6%	2%
Emulab	Avg. RTT	522	187	70	48	47	57	75	14		145
	Loss Rate	8%	5%	1%	1%	1%	1%	4%	6%	—	1%
NL	Avg. RTT	697	324	155	175	141	143	165	157	49	
	Loss Rate	7%	3%	0%	0%	1%	0%	1%	2%	1%	—

Table 5: Network characteristics during experiment III.

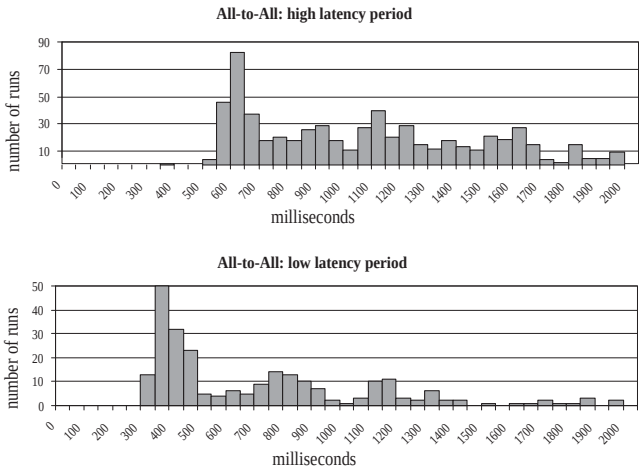


Figure 4: Histograms of overall running times, runs up to 2 seconds, experiment III.

a half by routing messages indirectly through CA.

7. CONCLUSIONS

We measured and analyzed the performance of four common information propagation algorithms over the Internet. We explained the distribution of the algorithms' running times in terms of underlying link latencies and loss rates.

One important lesson one can learn from our observations is that loss rates over the Internet are not negligible. Consequently, algorithms that send many messages often have a high running time, even if the messages are sent in parallel in one communication step. More generally, we learn that some communication steps are more costly than others. E.g., it is evident that propagating information from only *one* host to all other hosts is faster than propagating information from *every* host to each of the other hosts.

We suggest to refine the communication step metric as to encompass different kinds of steps. One cost parameter, Δ_1, can be associated with the overall running time of a step that propagates information from all hosts to all hosts[1]. This step can be implemented using any of the algorithms analyzed in this paper. A different (assumed smaller) cost parameter, Δ_2, can be associated with a step that propagates information from one host to all other hosts. Another cost parameter, Δ_3 can be associated with propagating information from a quorum of the hosts to all the hosts[2], etc.

This more refined metric can then be used to revisit known lower and upper bound results. E.g., [12] presents a tight lower bound of two communication steps for failure-free executions of consensus in practical models. Under the more refined metric, the lower bound is $2\Delta_1$, whereas known algorithms (e.g., [14, 5]) achieve running times of $\Delta_2 + \Delta_3$.

Acknowledgements

We thank all those who are hosting our experiments on their machines. Many of the machines we use belong to the RON project [3] at MIT, which is funded by DARPA; we are especially thankful to Dave Andersen for technical assistance with these machines. The Emulab machine is part of emulab.net, the Utah Network Emulation Testbed, which is primarily supported by NSF grant ANI-00-82493 and Cisco Systems. We thank Geoff Voelker for letting us use his machine in UCSD, and Yuh-Jzer Joung for letting us use his machine in TW. We also thank Sergio Rajsbaum, Roger Khazan, and the referees for helpful comments.

8. REFERENCES

[1] Agarwal, D. A., Moser, L. E., Melliar-Smith, P. M., and Budhia, R. K. The Totem multiple-ring ordering and topology maintenance protocol. *ACM Trans. Comput. Syst. 16*, 2 (May 1998), 93–132.

[2] Amir, Y., and Wool, A. Evaluating quorum systems over the internet. In *IEEE Fault-Tolerant Computing Symposium (FTCS)* (June 1996), pp. 26–35.

[1] Local running times cannot be composed in this manner.

[2] In future experiments we intend to evaluate a primitive that waits for responses from a quorum of hosts.

Initiator	Algorithm	All-to-all Overall	All-to-all Local	Leader Overall	Leader Local	Ring Overall	Ring Local
KR	Avg. (runs under 3 sec)	1197	692	1340	954	1853	1158
	% runs over 3 sec	66%	9%	25%	13%	18%	5%
TW	Avg. (runs under 3 sec)	1139	809	1644	1227	2014	1137
	% runs over 3 sec	64%	28%	84%	69%	22%	4%
MIT	Avg. (runs under 3 sec)	1168	515	896	589	1912	1117
	% runs over 3 sec	67%	3%	13%	6%	18%	6%
UCSD	Avg. (runs under 3 sec)	1172	497	833	558	2040	1115
	% runs over 3 sec	61%	2%	14%	7%	24%	6%
CU	Avg. (runs under 3 sec)	1133	494	1179	703	2076	1120
	% over 3 sec	58%	3%	9%	2%	21%	4%
NYU	Avg. (runs under 3 sec)	1156	516	1183	715	2092	1134
	% over 3 sec	62%	3%	8%	3%	27%	5%
CA	Avg. (runs under 3 sec)	1127	563	992	670	2073	1141
	% over 3 sec	66%	33%	44%	37%	27%	5%
UT2	Avg. (runs under 3 sec)	1120	558	1190	637	2121	1165
	% over 3 sec	64%	51%	60%	53%	30%	8%
Emulab	Avg. (runs under 3 sec)	1108	474	884	594	2066	1133
	% over 3 sec	67%	5%	15%	8%	24%	5%
NL	Avg. (runs under 3 sec)	1161	585	1146	772	2035	1143
	% over 3 sec	65%	3%	16%	7%	25%	5%

Table 6: Measured running times, milliseconds, experiment III.

[3] Andersen, D. G., Balakrishnan, H., Kaashoek, F., and Morris, R. Resilient overlay networks. In *SOSP* (Oct. 2001).

[4] Chandra, B., Dahlin, M., Gao, L., and Nayate, A. End-to-end WAN service availability. In *Third Usenix Symposium on Internet Technologies and Systems (USITS01)* (Mar. 2001).

[5] Chandra, T. D., and Toueg, S. Unreliable failure detectors for reliable distributed systems. *J. ACM 43*, 2 (Mar. 1996), 225–267.

[6] Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K., Santos, E., Subramonian, R., and von Eicken, T. LogP: Towards a realistic model of parallel computation. In *4th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming* (May 1993).

[7] Floyd, S., and Paxson, V. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking 9*, 4 (August 2001), 392–403.

[8] Gray, J. N. Notes on database operating systems. In *Operating Systems: An Advanced Course, Lecture Notes in Computer Science*, vol. 60. Springer Verlag, Berlin, 1978, pp. 393–481.

[9] Guerraoui, R., and Schiper, A. The decentralized non-blocking atomic commitment protocol. In *IEEE International Symposium on Parallel and Distributed Processing (SPDP)* (October 1995).

[10] Guo, K., Vogels, W., and van Renesse, R. Structured virtual synchrony: Exploring the bounds of virtual synchronous group communication. In *7th ACM SIGOPS European Workshop* (September 1996).

[11] Keidar, I., and Dolev, D. Increasing the resilience of distributed and replicated database systems. *J. Comput. Syst. Sci. 57*, 3 (Dec. 1998), 309–324.

[12] Keidar, I., and Rajsbaum, S. On the cost of fault-tolerant consensus when there are no faults – a tutorial. Tech. Rep. MIT-LCS-TR-821, MIT Laboratory for Computer Science, May 2001. Preliminary version in SIGACT News 32(2), pages 45–63, June 2001 (published May 15th 2001).

[13] Keidar, I., Sussman, J., Marzullo, K., and Dolev, D. Moshe: A group membership service for WANs. *ACM Trans. Comput. Syst.* (2002). To appear.

[14] Lamport, L. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (May 1998), 133–169.

[15] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7 (July 78), 558–565.

[16] Lamport, L., Shostak, R., and Pease, M. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems 4*, 3 (July 1982), 382–401.

[17] Paxson, V. End-to-end Internet packet dynamics. In *ACM SIGCOMM* (September 1997).

[18] Rajsbaum, S., and Sidi, M. On the performance of synchronized programs in distributed networks with random processing times and transmission delays. *IEEE Transactions on Parallel and Distributed Systems 5*, 9 (1994), 939–950.

[19] Savage, S., Anderson, T., Aggarwal, A., Becker, D., Cardwell, N., Collins, A., Hoffman, E., Snell, J., Vahdat, A., Voelker, G., and Zahorjan, J. Detour: a case for informed internet routing and transport. *IEEE Micro 19*, 1 (January 1999), 50–59.

[20] Savage, S., Collins, A., Hoffman, E., Snell, J., and Anderson, T. The end-to-end effects of Internet path selection. In *ACM SIGCOMM* (September 1999), pp. 289–299.

[21] Schiper, A. Early consensus in an asynchronous system with a weak failure detector. *Distributed Computing 10*, 3 (1997), 149–157.

[22] Sergent, N. Evaluating latency of distributed algorithms using Petri nets. In *5th Euromicro Workshop on Parallel and Distributed Processing* (London, UK, Jan. 1997), pp. 437–442.

[23] Skeen, D. Nonblocking commit protocols. In *ACM SIGMOD International Symposium on Management of Data* (1981), pp. 133–142.

[24] Stevens, R. *TCP/IP Illustrated*, vol. 1. Addison-Wesley, 1994.

[25] Urbán, P., Défago, X., and Schiper, A. Contention-aware metrics for distributed algorithms: Comparison of atomic broadcast algorithms. In *9th IEEE International Conference on Computer Communications and Networks (IC3N 2000)* (Oct. 2000).

[26] Zhang, Y., Duffield, N., Paxson, V., and Shenker, S. On the constancy of internet path properties. In *ACM SIGCOMM Internet Measurement Workshop* (November 2001).