A Facility For Experimenting Distributed Software in the Internet *†

Yongguang Zhang
Bharat Bhargava
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

Abstract

To test distributed software in the Internet we need many experimental sites in different physical locations. To ease the experimental setup, we have developed a facility for a study of distributed transaction processing in the Internet. The facility emulates remote sites by local machines, and emulates a true Internet communication environment in a local area network, by routing the inter-site communication through actual Internet hosts. This provides a mechanism to conduct experiments without any designated experimental sites except for the local ones, yet the results are as genuine as in the real experiments. It also supports a trace-driven emulation method that measures the inter-site communication of the Internet and uses the measurement data to generate empirical values for delay and message loss between two local sites. We have studied these approaches and observed that they are feasible and reliable. We have realized these two approaches in a communication package called the WANCE tool, which can be used to test distributed software, without changing the application programs. We have evaluated the emulation approach by comparing it with real experiments. The results are statistically acceptable.

*This research is supported by Army Research Lab (System Technology Branch, contract number DAKF11-93-C-0010) and AT&T.

†This paper is an extension of a workshop presentation [1].

1 Introduction

The rapid growth of the Internet has generated a lot of interest in distributed systems connecting information and computing resources of individual computers at different geographical locations. Information that was accessible to only a small group of people till now, will be available to a much broader community of computer users. Software systems running in a local and isolated network environment will move to the Internet environment. Thus, the importance of developing and studying the techniques for providing access to remote computer resources in a controlled, predictable and manageable way, is evident.

To assist in the transition of distributed applications from their current working environment, such as a local area network (LAN), with a small number of sites, to a global environment such as the Internet, with possibly a larger number of sites, we need to study the scalability and performance of these applications in the new networking environment. Although the transition to these wide area network (WAN) environment for some existing software is relatively straight forward, most practical systems will have difficulties in scaling up. This is because most of the research efforts for these application domains have been focused for the LAN environment. We need to study the behavior of Internet with respect to these distributed applications. We can achieve this objective by extensive experimentation and performance analysis of these applications in the Internet environment.

Motivation for experiments One application of the large scale distributed system software that interests us, is the transaction processing in WAN environment. The scalability of transaction processing systems in a WAN environment is an important problem. We believe that it will have a significant impact in the design of the next generation database applications also [2]. A lot of effort has been expended by the database research community in the development of protocols and algorithms for transaction processing in the past two decades [3]. These algorithms have been implemented, analyzed and found to be adequate for the LAN environment. However, not enough studies have been conducted for evaluating the applicability of these protocols in the WAN environment. We want to study the performance implications of various concurrency control, replication control, and commitment algorithms in the Internet environment. Based on these studies we want to tailor these protocols for the WAN environment. The main theme of our research is to understand the above implications through experimental studies and use them to develop new protocols or modify existed ones for the design of a high performance and high availability transaction processing system. This will also provide us with guidelines on how to extend various distributed applications to the Internet.

Problems in experimental setup The Internet is one convenient wide area network testbed available to the research community. We have observed that the availability of large number of sites for experimentation is a problem faced by a researcher. This is because obtaining accounts on various remote sites requires administrative intervention. Furthermore, incompatibility of the computing environments (such as the operating system) makes porting the distributed application software difficult. These factors imply lower flexibility in choosing the sites for the experimentation with distributed software. But repeating the same experiments with different sites is necessary for conclusive evidence and carries a statistical performance significance. We also believe that having experimental sites wide spread makes it more difficult to monitor and control the experiments, as compared to their execution in the LAN environment. The goal of this research is to overcome these problems and develop convenient facilities for experimenting with the distributed software in the Internet environment.

In the rest of this paper we first discuss a number of experimentation procedures that are in use for studying distributed system software in the Internet. We introduce a novel experimental approach, called *Internet emulation experimentation*, which will solve the resource scarcity problem and provide validated results. We also introduce another similar approach called *trace-driven emulation experimentation*, which is an extension of our Internet emulation experimentation approach. We then present the design and implementation of a software tool that supports both of these experimentation methods. We have used our tool to conduct the experiments for distributed transaction processing in the Internet. The validation result is also discussed.

2 Experimental Methods

There have been many attempts for conducting distributed processing system experiments in the Internet Most of the efforts have either been in conducting *real experimentation* using the actual sites on the Internet, or been in *simulation modeling* to imitate the behavior of a distributed system in a WAN.

2.1 Real Experiments

A real experiment runs the distributed software on a set of actual Internet sites. In the literature, many experimental studies have followed this approach. For example, research groups in Vrije University have conducted wide-area experiment using Amoeba system with Cornell in the United States and Amsterdam in the Netherlands as two Internet sites, to study the transparent computing [4, 5, 6]. A team at Columbia University has investigated the performance of distributed processing in the Internet through experimental studies [7]. Two experiments were performed, one to run a distributed processing facility called Camelot between Carnegie-Mellon University (CMU) and Columbia University, and the other to do Webster dictionary lookup between Columbia and University of Washington. Researchers in University of California at Santa Cruz have also studied the performance of distributed processing in the Internet environment [8]. Parameterized results such as mean time to failure and availability of the Internet sites were obtained and used to develop better distributed processing software.

As is evident from the previous studies, real experiments and testing with the actual sites in the Internet works well for systems involving a small number of sites. The geographical distance between experimental sites, the possible administrative barriers, and the need for remote control account for the high overhead in setting up a real experiment. For example, the two Columbia experiments mentioned above involved a professor in Washington and CMU to coordinate for setting up the experiments. Moreover, limitations on the resources available to the individual researchers or research groups implies that the experiment has to be conducted on some designated sites. It cannot be scaled arbitrarily nor can it be easily repeated on some other sites. Therefore, when the number of sites becomes large, more autonomous units are involved, and when the experimentation requires a wider selection of sites, real experiments simply become unrealistic.

2.2 Simulation

When the resource needed for real experiments are not available or too expensive, researchers use simulation to model real experiments. Simulation study can provide good scalability with minimum resources requirement. There have been much simulation study on the behavior of the interconnection networks [9, 10, 11, 12].

Simulation is a good tool to determine the behavior model and the worst-case performance for a distributed system. To study large scale distributed systems under the Internet environment, the simulation heavily depends on the communication model. It is important to adopt a realistic model, to justify the input parameters on Internet performance, and to validate the results. As a successful example, Yang et al developed a fairly accurate interconnection network model using measured delay values for different network elements and used it to study the performance of file transfer in a satellite wide-area network [12]. Danzig et al of University of Southern California developed an empirical workload model to drive simulation experiments. It was based on the analysis of TCP traffic collected at University of Southern California, University of California at Berkeley, and Bellcore [9]. There are also many studies that analyze the Internet preformance and model it. [13, 14, 15, 16]. Better simulation techniques are being developed (e.g. the Flowsim project in University of Southern California).

As stated above, a simulation model must be able to simulate the real world as closely as possible. With over two million hosts in the Internet generating traffic independently, designing a simulation model to approximate the behavior of the real Internet is not an easy problem [17]. Further, we often want to test the distributed software under the realistic and nonstationary conditions, not just under simulated ones. Thus if possible, there will be more interest in real experiments as opposed to the simulation studies.

2.3 Emulation

Emulation is a new way of testing software in the "real-life" Internet without the need for designated remote sites. It overcomes the problem of lacking experimental sites in many real experiments. The basic idea is to emulate a remote site using one local computer, and to emulate a true Internet communication environment in a local area network by routing the inter-site communication through the actual Internet hosts. The software only runs on local computers but the communication goes remote. Since the user-specified routing is supported by most part of the Internet, this kind of experiments can be set up in many sites that have connection to the Internet.

We now illustrate how it works using an example. Assume that we have a two-site distributed system. To carry a task, one site will send messages to the other site ; the second site will send reply messages to the first site. Suppose we want to set up an experiment with the following two sites: `raid10.cs.purdue.edu` (denoted as *computer A*) in Purdue University (West Lafayette, Indiana) and `cs.helsinki.fi` (denoted as *computer X*) in Helsinki University (Helsinki, Finland). Usually, we need to login to the both computers using `telnet` or `rlogin`,[1] run the software, and set up the experiment. In the emulation model, we run the system (*site1*) on the Purdue site as usual. However, we don't run the other peer (*site2*) in the Helsinki site. Instead, we can simply pick another machine at Purdue, say `raid11.cs.purdue.edu` (denoted as *computer B*), which is in the same LAN as *computer A*. A specially designed communication software for the distributed system is used so that all the communication packets from *site1* to *site2* are routed through *computer X* (see Figure 1). Therefore, even though both sites are close to each other and are directly connected by a LAN, the packets travel across the Atlantic Ocean to northern Europe and back. This setup makes the processing of the A–X system nearly identical to that of the A–B system.

Although we use a two-site system as an example, it can be applied to a multi-site system. Figure 1 gives such an example that local *computer B* emulates remote *computer X*, local *computer C* emulates remote *computer Y*, and so on.

If we use this emulation approach, we can conduct distributed system experiments without having control over remote hosts. The rationale behind this approach is based on the observations about the isomorphic behaviors of the two configurations, a distributed system that spans over the Internet and the same system that runs in a LAN. The difference between them is not due to the individual computers of the distributed system (as long as they are comparable models), nor on where these testbed computers are located. It is the communication performance such as the delays between the testbed computers that matters. Since they both have the same communication path over the Internet, we can project conclusions from one system that runs in a LAN (the emulation result), to the other that runs over the Internet (the target configuration).

2.3.1 Two-phase emulation technique

Using the above configuration alone is not always sufficient. In many cases request/reply messages might not have the same size, or sometimes a request message may not result in a reply message. Therefore, we need to repeat the above emulation experiment in a "*mirrored*" configuration, that is, with the pathes of

[1] Both are network utility programs that establish interactive terminal sessions with a remote computer in the Internet.

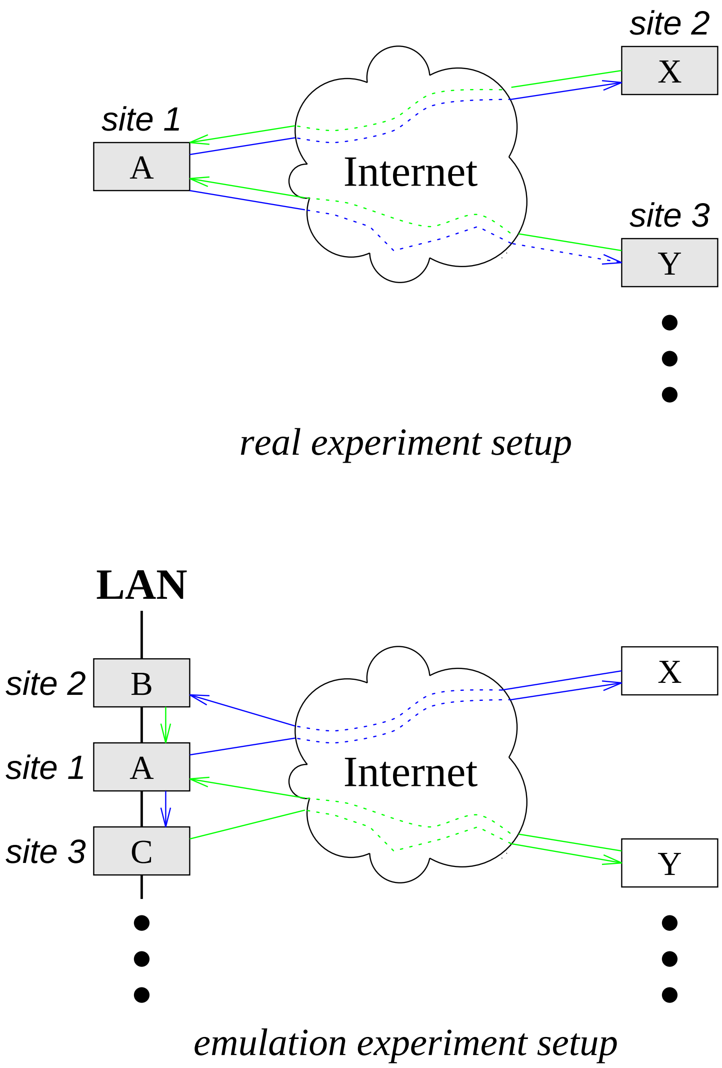

Figure 1: Emulating the real experiment

request and reply messages swapped. We called this a **two-phase** emulation technique. (See Figure 4 for an example.)

In a two-phase emulation experimentation, we repeat the same experiments under two configurations. In the first configuration, all messages from *site1* to *site2* will be routed through *computer X*; all messages from *site2* to *site1* will go directly (as described above). This is the first phase. In the second configuration, all messages from *site1* to *site3* will travel directly in the LAN, while all messages from *site3* to *site1* will be routed through *computer X*. This is the second phase. We should make measurements in both phases of experiments and take the averages of the results.

2.3.2 Applicability and limitations

The advantage of the emulation method over real experiments is that emulation experiments are easy to control and easy to implement. The processing of the distributed systems can be controlled inside a laboratory environment. It requires no designated site; the only requirement for the emulated site is that it can bounce back the messages. No software is required to set up or run on that site. Therefore we can do experiment with virtually *any* hosts in the Internet. This enables us to obtain a large amount of data points under different configurations for statistical purposes, or to carry thorough testing using many different sites. Both would be otherwise impractical in real experiments.

This approach has several limitations. First, it is not suitable for distributed system that are based on asymmetric processing or decentralized control. Take the experiments in Figure 1 for example, if site2 communicates with site1 in a real experiment, it cannot be easily implemented in an emulation experiment. However, many distributed systems are based on a symmetric *client/server* model and has a centralized control site (server). All processing involves only communication between the centralized site (server) and the other participating site (client). Usually there is no direct communications among clients. For such systems, emulation experiments is suitable as long as you place the server in *computer A*.

Secondly, the communication time for a message from one site to another in emulation configuration does not "equal" to that in the real experiment. From Figure 1 we can see that there is overhead of a local message from *site2* to *site1*. However, compared with the message round trip time through the Internet (about hundreds to thousands milliseconds), the delivery time between two local sites (in terms of hundred microseconds to a few milliseconds) is negligible. In fact, we will see in Section 4 that the overhead is really insignificant.

Furthermore, this scheme works only when the distributed system is not based on the clock synchronization among different sites, because of the asymmetric communication time (the time to deliver the reply message was actually taken to return the request message). Since there has not been a practical solution to the clocks drift problem in the Internet, many distributed systems are indeed based on asynchronous coordination among sites.

Compared with simulation, one limitation about the emulation approach is the limited scale. Since the computational part of the distributed system is not emulated, the number of sites that can participate in the experiment is bounded by the number of computers available in the LAN. Although it is possible to have a computer emulating multiple sites, the parameters will be far more complicated.

2.4 Trace-Driven Emulation Method

The emulation approach provides us with an inexpensive alternative to the real experimentation to test the distributed software, but with some limitation as mentioned above. This limitation can be eliminated if we combine the real-time traces of the Internet communication with emulation. We call this a *trace-driven* emulation method.

Experiments using this method are similar to the emulation experiments. We use a local computer to emulate a remote computer in the Internet and run the distributed software on the local computers. Instead of having the inter-site communication packets routed through the remote sites, we make the local sites communicate directly via the LAN, but impose a delay for each such communication packet (Figure 2). This delay is computed from the concurrent traces of packet delivery between the actual remote sites, and this emulates the actual delay for distributed processing. Since this delay is based on the traces on the Internet, it changes continuously and can be used to approximate the "real-life" situation in the Internet. If the delays are found to be infinite, the local sites simply drop the packet to emulate network congestion or network partition.

The values for the delays are based on the real-time sampling of the Internet. For example in Figure 2, local machines A, B, C emulate remote machines X, Y, Z. A sampler program, which constantly measure the communication behaviors among X, Y, and Z (including the communication delay and packet loss), collects the measurement data and exports them to the empirical delay generator. It delays the communication among A, B, C according to the current delay measured at X, Y, Z.

The communication delay between X and Y, in the normal conditions, can be measured by routing a probe packet from A to X to Y then back to A, echoing another probe packet from A to X then back to A, and a third packet from A to Y then back to A. By putting a timestamp in each packet we can record

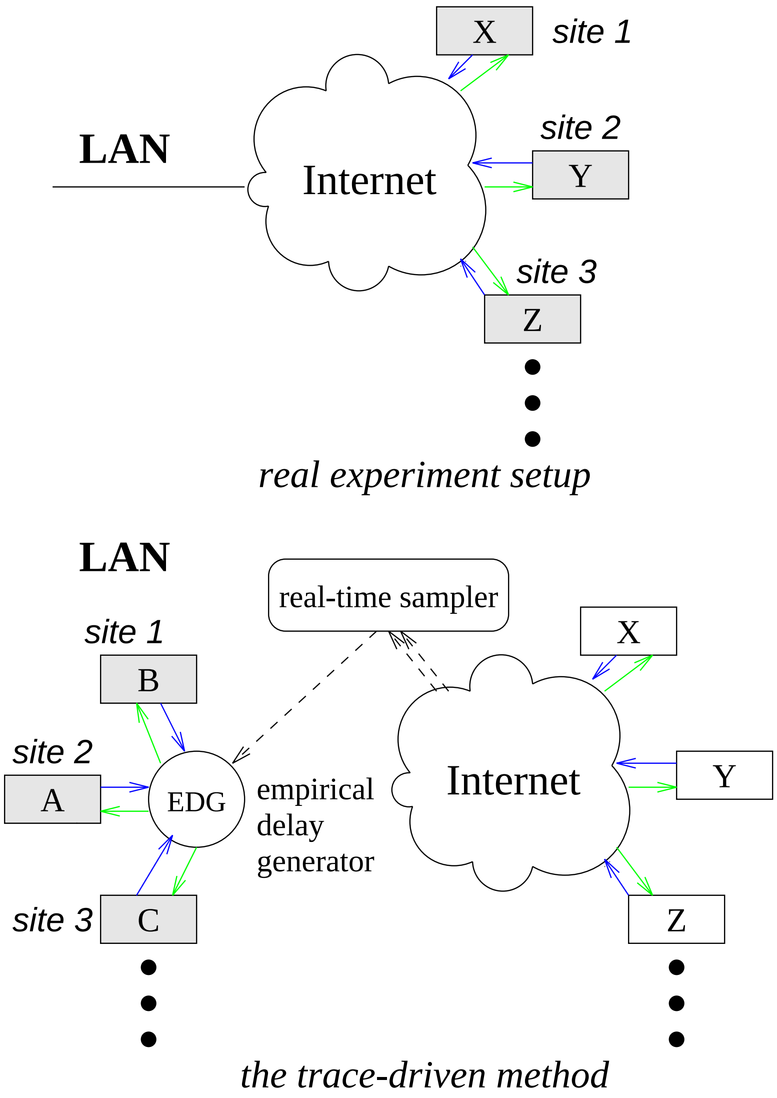

Figure 2: Emulation with real-time communication trace

the delay of the trips and calculate the delay from X to Y:

$$delay(X \to Y) = delay(A \to X \to Y \to A) - delay(A \to X \to A)/2 - delay(A \to Y \to A)/2$$

Thus, we can obtain delays between any pair of sites.

The probability of packet loss between X and Y can be calculated as follows. We repeat the above measurement for delays for a large number of times and count how many packets of the type $A \to X \to Y \to A$, $A \to X \to A$, and $A \to Y \to A$ come back to A. Divide the numbers by the amount of probe packets sent and we have the probabilities of loss in the trip A to X to Y to A, the trip A to X to A, and the trip A to Y to A. Then the probability of loss in the path of X to Y can be calculated by solving the following equation set:

$$P_l(A \to X \to Y \to A) = 1 - (1 - P_l(A \to X))(1 - P_l(X \to Y))(1 - P_l(Y \to A))$$

$$P_l(A \to X \to A) = 1 - (1 - P_l(A \to X))(1 - P_l(X \to A))$$

$$P_l(A \to Y \to A) = 1 - (1 - P_l(A \to Y))(1 - P_l(Y \to A))$$

where P_l denotes the probability of loss in the path. Assuming $P_l(A \to X) = P_l(X \to A)$ and $P_l(A \to Y) = P_l(Y \to A)$, we have

$$P_l(X \to Y) = 1 - \frac{1 - P_l(A \to X \to Y \to A)}{\sqrt{1 - P_l(A \to X \to A)}\sqrt{1 - P_l(A \to Y \to A)}}$$

Since most of the Internet supports the routing and echoing, we can emulate virtually any Internet site without having the control over it. The measurement for delays and packet losses and the experimentation with the distributed software are simultaneous. This approach uses the real processing of the experimental software system. Further combining this with the "communication trace" collected in the real-time, we can reasonably conclude that the experiments conduced under such a setup will reflect the behavior of real experiments.

Compared with the artificial workload approach developed at University of Southern California [9], our trace-driven approach is not based on the analysis of previous collected traces. It generates delays and losses directly from the traces collected at the time of the experiment.

The real-time trace-driven approach eliminates many of the limitations of the emulation approach discussed in the previous subsection. The requirement of symmetric processing and centralized control is no longer necessary, because now each site communicates with any other site in the same way. Also, systems that rely on synchronous coordination can now be supported.

3 Implementation of WANCE Tools

We have implemented a special communication software system that supports the emulation experiments in the Internet. It is called the WANCE (Wide Area Network Communication Emulation) tool. It is a general-purpose communication package for testing distributed software. We wrote the first version of WANCE tool (WANCE-1) in 1992 to support emulation experiments. We recently developed the second version, WANCE-2, to support both emulation method and trace-driven method. In this section we will describe the architecture and some implementation issues.

For emulation experiments, the WANCE tool will capture the communication messages and route them through a pre-defined remote site, according to the message's source and destination addresses. and the user-supplied configuration. It will also ensure that each message arrives at the appropriate destination after reaching the intermediate echo site. For trace-driven experiments, the WANCE tool will spawn an Internet sampler after start up, and then capture the communication messages, and delay it for a period of time suggested by the sampler.

There are many ways to implement a WANCE tool for the Internet, such as the echo service at the protocol level or the packet source routing, depending on the strategy used. A criterion for a good implementation is that it should make the emulation experiment as close to the real experiment as possible.

3.1 Routing Communication Packets

In the emulation model, for each message originated from *computer A* (*site1*) and destinated to *computer B* (*site2*), the WANCE tool has to send it to a remote *computer X* over the Internet and it must be bounced back to *computer B*. Although there is no generally available high-level message forwarding service in the Internet, we can use two kinds of standard Internet services to solve the problem. They are the *loose source routing* option of IP (Internet Protocol) datagram delivery, and the *echo services* in each protocol layer [18].

IP source routing Internet is a packet delivery network. The path of an IP datagram traveling from the sender host to the receiver host in the Internet is usually determined by the gateways, according to network topology and the current connectivity condition. The *loose source routing* option of the IP packet also provides a way for the sender to dictate a path of the packet through the Internet. It is a sequence of Internet addresses specifying that the packet must follow the sequence of Internet addresses before reaching the destination. This feature of Internet delivery can be used to redirect messages in the WANCE tool.

For example, if we want to route a message from source *computer A* through *computer X* to destination *computer B*, we can specify the loose source route option "*A*, *X*, *B*" in every packet of the message before sending it out from *computer A*.

One limitation of the IP source routing technique is its availability. Although source routing is a standard option of IP delivery that every gateway in the Internet is supposed to conform, some implementations of Internet gateways choose to ignore it for the sake of performance. Some even drop those packet that carries the source routing option. Many others, although doing source routing, put packets that carry the source routing option into queues with lower priorities. Such packets may travel slower than others.

Echo services We can also use the echo services in each layer of the Internet protocol, IP (*Internet Protocol*, the fundamental layer), UDP (*User Datagram Protocol*), and TCP (*Transmission Control Protocol*). IP includes ICMP (*Internet Control Message Protocol*) as an integral part that handles error and control messages. The echo service of IP level is implemented by the ICMP echo request/reply messages. Any gateway or host that receives an echo request addressed to it will formulate an echo reply and return it to whichever machine sent the request. UDP adds a port number to the IP layer to distinguish multiple applications running in a same machine. Port number 7 is reserved for the UDP echo services. Whenever a packet arrives at that port, the receiving host sends the identical packet back to the sender. The same service is also implemented in the TCP layer. A program can get back what it writes to a remote host using TCP as long as it connects to the port 7 of the remote host. These echo services returns a message that is identical to what one sends[2], only with a delay of the round-trip communication time across the Internet, from the sender to the echo host.

All these echo services in the three layers can be used in the emulation system to redirect messages. For example, to route a message from source A to destination B through remote computer X, the WANCE tool in A can filter the outgoing communication from A to B. If the protocol is UDP, the WANCE tool sends the messages to X:7/UDP instead. If the protocol is TCP, the WANCE tool rather connects to X:7/TCP.

Usually the echoed message will get back to the sender (A). To make it reach the destination (B), the WANCE tool in computer A simply picks up this returned message and directly passes it to computer B

[2]Some implementations of UDP echo service restrict the size of the packet. For example, the maximum size of an UDP echo packet allowed in SunOS is 1024 bytes.

(which is in the same LAN). Although extra overhead is involved here, the time for the message to travel over the Internet dominates in the total time from A, to X, back to A, and then to B. The overhead for the last leg is a small constant and negligible. An alternative implementation is to set up a special forwarding program running in the gateway of the LAN. All messages are sent to the gateway first. The gateway does the redirection by sending them to *Site2* and upon receiving the echo messages, passes them to the receiver.

3.2 Emulating the Communication with Real-Time Trace

In the trace-driven model, each message from A to B is delayed by a certain amount of time $T_{A,B}(t)$ (where t is the real-time clock value when A sends the message). At the same time, an Internet sampler collects the delay between host E_A and E_B continuously (where E_x is the remote site emulated by the local computer x). It generates an elapse time $delay_{E_A,E_B}(t_i)$, where t_i is the real-time clock when the data is collected.

Unlike emulation where messages are packaged and traveled through the real Internet, the trace-driven method depends on the measurement data. Very often at the time a message needs to be sent, the current measurement data is not ready. We have to rely on the previous measured data and generate the delay value randomly and according to some distribution. Bying using such simulation technique, we can also save the bandwidth by probing the Internet less frequently. The frequency depends on the distance and normal latency between E_A and E_B. We found that a measurement of every five seconds is a moderate sample rate between two sites in the United States.

We now show how we generate the delay from the measurement data. The delay $T_{A,B}(t)$ is estimated from the latest $delay_{E_A,E_B}(t_i)$. When $t = t_i$, we simply take the elapse time $delay_{E_A,E_B}(t_i)$ to be $T_{A,B}(t)$. When $t \neq t_i$, we have a few choices. An easy way is to take the latest $delay_{E_A,E_B}(t_i)$ to be $T_{A,B}(t)$. This assumes that Internet communication behavior won't change significantly in five seconds.

A more precise way is to use the *recursive prediction error* or *stochastic gradient* method, an important technique in estimation and control theory. It has been successfully used in computer communication, e.g. to determine timeout in TCP implementation [19], and to predict the latency [20]. According to this method, the delay expectation $ED(t_i)$ at time t_i can be calculated by

$$ED(t_i) = (1 - w)ED(t_{i-1}) + w \cdot delay_{E_A,E_B}(t_i)$$

i.e. a moving average of the recent samples. Here parameter w is a factor ($0 < w < 1$) that one can choose to weigh more on the recent samples or on the old ones. A typical value for w would be 0.1–0.2, but we pick 0.125 as suggested by [19].

We then compute the expected variation of communication delay as

$$var(t_i) = var(t_{i-1}) + \mid ED(t_i) - delay_{E_A,E_B}(t_i) \mid^2$$

To generate random delay for $T_{A,B}(t)$, we assume normal distribution for the communication delay between two sampling time, and use $ED(t_i)$ as the mean and $mdev(t_i)$ as the mean deviation.

With the frequent sampling of the Internet communication between E_A and E_B (using the method suggested in section 2.4) and with the random delay generation, we can accurately emulate the communication delay and enforce the same delay on communication between A and B.

3.3 WANCE Tools

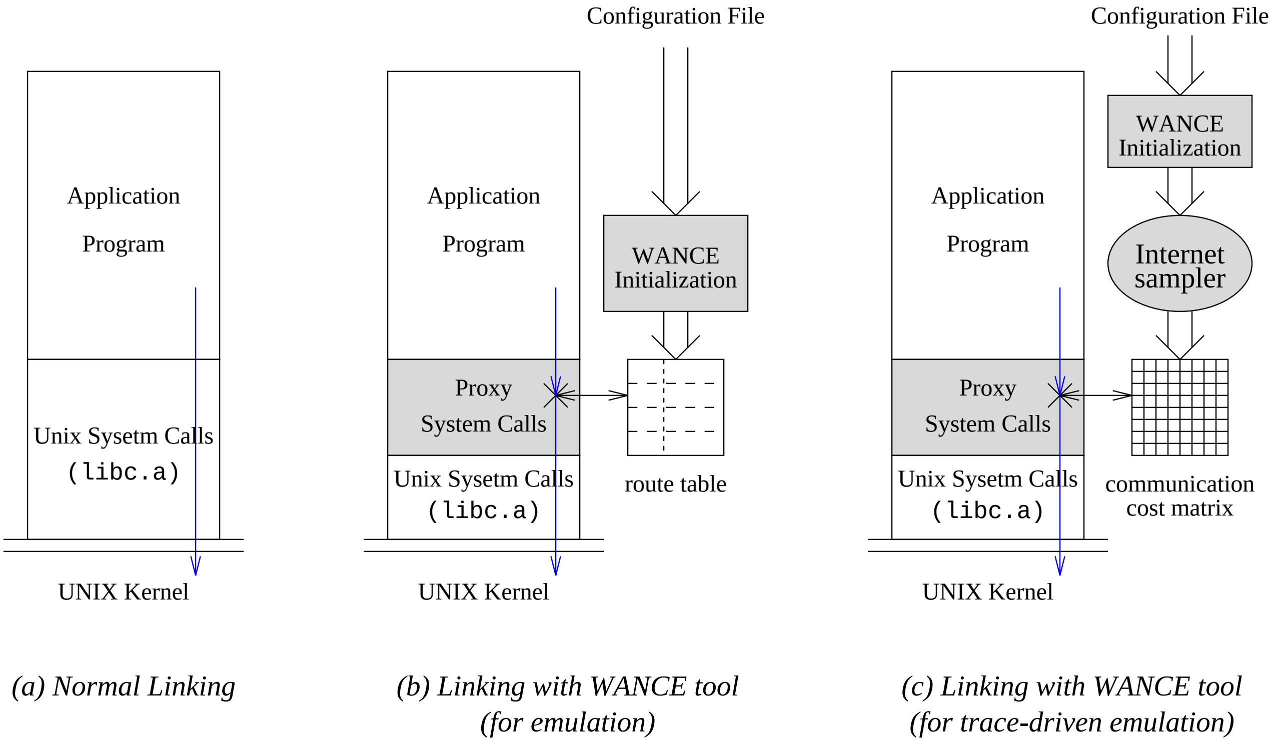

Figure 3: Software structure of the WANCE tools

We implemented WANCE-1 using echo services instead of source routing since the latter was not always available in the gateway of our Purdue network to the Internet. The new WANCE-2 provides both echo and source routing as alternative ways to redirect messages. We developed the WANCE tools under SunOS 4.1. It is based on BSD Unix interprocess communication primitives (the socket interface) and consists of a library of C routines and a set of data collection and analysis programs. For emulation experiments, each module or server of the distributed software must be linked against the WANCE library first. The library functions as a communication subsystem that sits between the distributed software modules and the communication primitives provided by Unix. It provides the same interface for the upper layers but replaces the original communication primitives with special message redirection semantics. A variety of distributed system may be linked to the library with little or no change to the code.

The technique used is "proxy", that is, to replace some of the Unix communication system calls with equivalent versions that change the message delivery semantics. The proxy versions later invoke the original system calls to complete the delivery. Figure 3 shows the structure of the WANCE tools with an application program.

The Unix system calls that are replaced include `connect()`, `accept()`, `sendto()`, `sendmsg()`, `send()`, and also `write()` if used with a socket. For most programs whose communication is mainly handled by these routines, the proxy works satisfactorily.

For more detail on how WANCE tool changes the semantics of these routines, please see the Appendix. It discusses how the WANCE tool is initialized, and how it proxies system calls under different emulation models.

With the WANCE tool, we can easily set up experiments for our experimental distributed software. We have linked the Raid system[3] with WANCE and have been able to conduct experiments on the Internet. The integration was smooth.

[3]Raid is a distributed database system developed in Purdue [21]. We have been using it as an experimental infrastructure for our researches in communication, adaptability, reliability, transaction processing over several years. The interactions between Raid servers are in form of request/reply messages following the "client/server" paradigm, handled by the Raid communication subsystem [22].

4 Validation of the Emulation Model

Emulation is an experimental method that provides a communication environment comparable to that in a real experiment. Before any observations and conclusions are drawn from emulation experiments, we would like to identify the relative error involved in the emulation model. We will answer the question: "how close is the emulation experiment to the real experiment?"

In this section we only discuss the validation of emulation method and omit the study on the trace-driven method. The trace-driven method is based on a distribution model and an estimation method that are well-established in statistics. Comparatively we need a thorough validation study on the emulation method. The overhead of the WANCE tool in the emulation method also applies to the trace-driven method.

4.1 Procedure

Error estimation The relative errors of the emulation method can be estimated by doing a real experiment and an emulation experiment side-by-side and comparing the measured data. These two experiments are actually the same except using different experimental methods. Since there are large intrinsic variations in Internet experiments, they have to be repeated a large number of times. To ensure that both experiments are under the comparable Internet communication conditions, we interleaved the processing in two experiments.

Application We have been conducting studies in distributed transaction processing, so we the Raid distributed database system as the application in our validation experiments. We use the DebitCredit benchmark (also known as TP1 or ET1) [23]. It is a realistic transaction processing benchmark and is well-accepted. This benchmark uses a small banking database, which consists of three relations: the teller relation, the branch relation, and the account relation. The tuples are 100 byte long and contain an integer key and a fixed-point dollar value. In addition, there is a sequential history relation, which records one tuple per transaction. Its tuples are 50 bytes long and contain a teller identifier, a branch identifier, an account number, and the relative dollar value specified in the transaction. A transaction updates one tuple from each of the three relations and appends a logging tuple to the sequential history relation. Hot-spot access is 80%, i.e, 80% of the access focuses on 20% of the data items. 20% of the transactions are read-only transaction. The average number of read/write operations per transaction is six with variance 4.0. A run of the benchmark is a stream of 20 such transactions (total 120 read/write actions). It is sufficient to make observation in an unreliable Internet environment. More details about the benchmark can be found in [22, 23].

The database is fully replicated at both sites. We use ROWA (Read One Write All) protocol in the replication control, two-phase locking protocol in the concurrency control, and two-phase commit protocol in the atomicity control. We set the concurrency level to be one, i.e. one transaction at a time, since we like to distinguish the cause of transaction aborts due to the conflict access to the database items or due to the unreliability of the Internet.

Configuration We used three Sun workstations in the experiments. Two of them are located in our laboratory. We refer them as `raid4` and `raid11` here. `Raid4` is a SPARC IPC and `raid11` is a SPARCstation-1. The third workstation we used is physically located in Hong Kong.[4] We refer this machine as `ust`. It is also a SPARC IPC.

All of the three machines are properly equipped for distributed transaction processing. Both `raid4` and `ust` have the same configuration and run the same version of SunOS 4.1.1, for the purpose of fair comparison between emulation experiments and real experiments.

As a side note, the geographical distance between the two sites in our laboratory and the one in Hong Kong is about 8 thousand miles. The Internet connection between them are through Chicago, San Francisco, and then across Pacific Ocean to Hong Kong.

Figure 4 shows the configurations of the real experiment and the emulation experiment. In the real experiments, site 1 of the two-site database system runs in `raid11` and site 2 runs in `ust`. In the emulation experiments, site 2 runs in `raid4` instead. In phase I of the emulation experiment, messages from site 1 to site 2 are redirected to `ust`, while in phase II, messages from site 2 to site 1 are redirected. This is to study how much bias will be introduced if request and reply messages are not of the same length.

[4] It is the courtesy of Department of Computer Science, the Hong Kong University of Science & Technology for allowing us to use their facility for collaborative research.

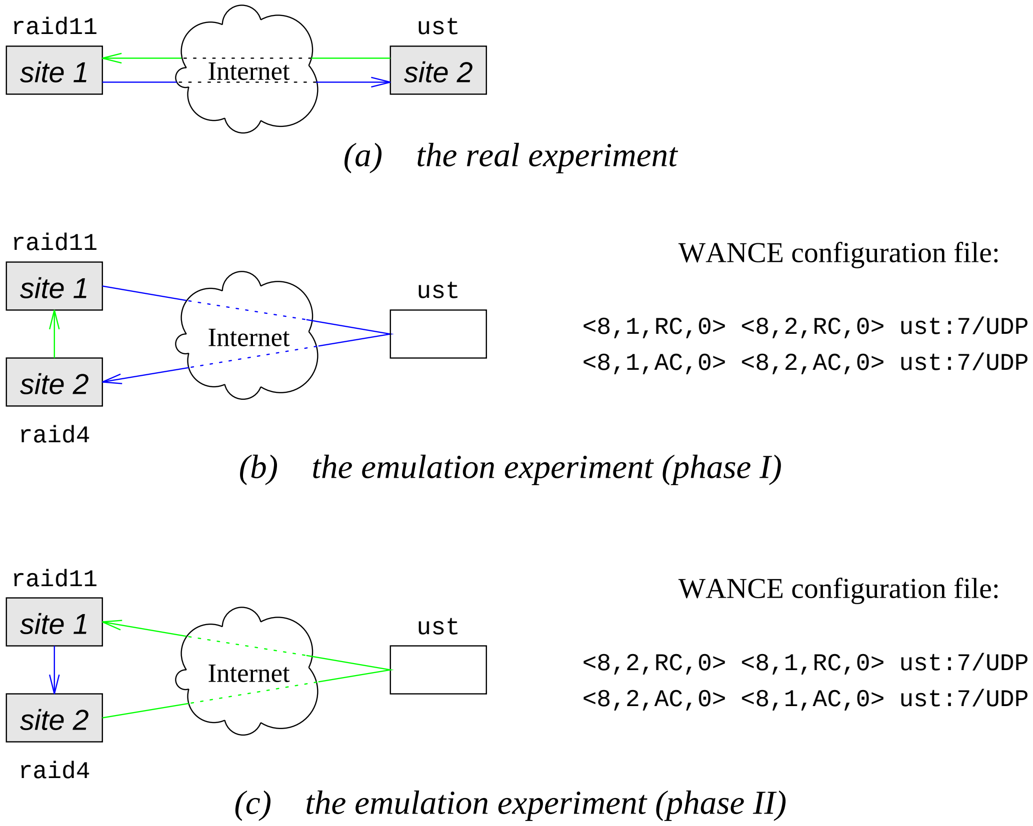

Figure 4: Three configurations of the experiments for error estimation

4.2 Data

We ran the experiments in a weekend when the network load was low. Although the time of day has strong correlation to the performance of message delivery in the Internet, a relatively stable environment made our estimation easier. In each run of the experiment we fed the stream of 20 transactions generated from the benchmark into site 1. We recorded the response time for each transaction, and calculated the average response time of the 20 transactions. We set the timeout to be 2 second for each operation, and the maximum number of restarts of a failed transaction to be three. The whole experiment consisted of 171 successful runs of the three configurations: a real experiment, then phase I of the emulation one, and the phase II. That is, we ran a total of 4000 transactions for each configuration.

We measured three important performance metrics about distributed transaction processing, the estimated mean response time, throughput, and transaction abort rate. The response time is the elapse time between a transaction is submitted to the system and the result (or failure) returned by the system. The throughput is the average number of transactions completed by the system in one second. The transaction abort rate means the percentage of transaction that cannot be finished due to the failure to delivery message in a certain time. Figure 5 shows the samples of the response time (RT), throughput (TP), and the abort rate (AR) for each run of the benchmark transactions.

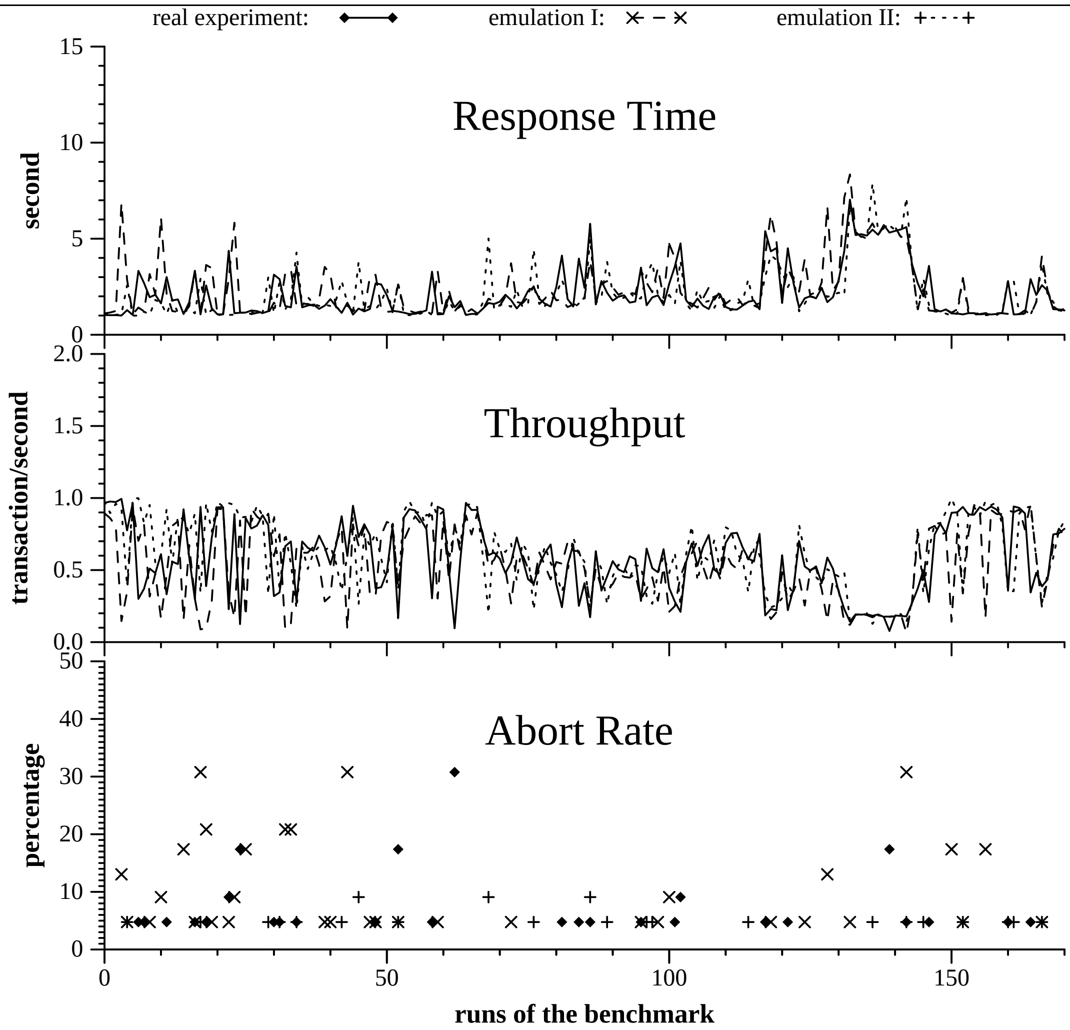

Figure 5: Three configurations of the experiments for error estimation

From Figure 5 we can see that in the samples acquired in both phases of the emulation experiments are very close to that acquired in the "real" experiments. We calculated the estimated mean times of RT, TP, and AR and the standard deviations. (see Table 1). By taking the average of the numbers acquired in both phases of the emulation experiments we got "combined samples" for the emulation experiment. These data were compared to the data acquired from the real experiments and the relative errors[5] were also computed. Table 1 shows such estimated errors.

measures		experimental methods: real experiment	emulation experiment: phase I	phase II	combined
RT	mean (second)	2.145	2.331	2.076	2.204
	relative error	-	8.7%	-3.2%	2.7%
	standard deviation	1.294	1.486	1.322	1.255
TP	mean (per second)	0.581	0.534	0.617	0.576
	relative error	-	-8.2%	6.1%	-1.0%
	standard deviation	0.250	0.255	0.250	0.218
AR	mean (%)	1.176	2.153	0.688	1.420
	relative error	-	83.1%	-41.5%	20.8%
	standard deviation	0.036	0.057	0.019	0.031

(RT = Response Time, TP = Throughput, AR = Abort Rate)

Table 1: The estimated error for emulation experiments

4.3 Results and Discussion

From Figure 5 and Table 1 we can see that the emulation method introduces small relative errors to the experimental data. The data for both response time and throughput, the relative errors are within $\pm 3\%$. Even the standard deviations are very close to each other.

We noticed that neither phase I nor phase II of the emulation experiment alone is sufficient, due to the unbalanced message traffic in both direction. We examined the message log between the two Raid sites.

[5] $(d-r)/r$, where d is the datum from the emulation experiment and r is the datum from the real experiment.

For one benchmark transactions stream, At least 36 remote messages, a total of 7504 bytes (including message headers, 36 bytes per message, ditto below), were from site 1 to site 2. at least another 36 remote messages, a total of 2880 bytes, were from site 2 to site 1, which were only one-fourth the size in the other direction. This is because all the transactions were injected into site 1, read data were fetched locally, but all writes had to go to site 2 as well. In a real experiment, a total of 10384 bytes traveled across the Internet. But in emulation experiment phase I, it were 15008 bytes and in phase II only 5760 bytes. Since small messages travel slightly faster than large messages [22], the transaction response time measured in phase I of the emulation experiment is longer than the actual time measured in the real experiment, but that measured in phase II is shorter. Similar situation applies to the throughput measurements. The average of phase I and phase II, however, is closer to the real data since the average number of messages and the average size of each message are the same as in the real experiments.

The transaction abort rate measurements in emulation experiments are much less accurate. Although the number of remote messages are the same in all configurations, a message that is too large to be shipped in one datagram in the Internet (usually 512 bytes) are fragmented into two or more pieces and hence has two or more times higher probability to be lost or delayed. Lost and delayed messages cause transaction aborts. Therefore the abort rate measured in both phases of the emulation experiments are quite different from the real experiments. Also, transaction abort is one kind of independent failures, which will be much harder to measure than the response time or the throughput. Message loss in Internet is often transient. Even though we interleaved the processing of two experiments, there is no guarantee that a message loss can be repeated in all three experiments. Considering these factors, a 20% error on abort rates by emulation experiment is acceptable.

The experiments clearly show the high fidelity of the emulation method on the measurements of distributed transaction processing performance, including metrics like response time and throughput. We have demonstrated that the emulation experimental method is feasible and reliable.

5 Conclusion

Experimentation is important in understanding the behavior of geographically distributed system software for a dynamic networking environment such as the Internet. We have discussed many ways of doing experimentation over the Internet, namely, real experimentation, simulation, emulation, and trace-driven emulation. Table 2 summarizes the pros and cons of these approaches and suggests the intended applications for each of these approaches.

	pros	cons	suitable experiments
real experiments	most reliable	hard to set up, limited scale	not required to repeat on different set of sites
simulation	inexpensive, easy to scale up	require an accurate model	systems involving large number of sites
emulation	real communication, real processing, easy setup	not good for asymmetric processing or decentralize control	client/server architecture, natural of the experiment require repetition on different sites
trace-driven	easy setup, good for any distributed computing paradigms	communication is simulated, not real	natural of the experiment require repetition on different sites

Table 2: A synopsis of Internet experimentation methods

We have shown that the emulation approach can reduce the complexities of performing experiments. The use of WANCE tool free us from many tedious chores involved in conducting experiments. We can thus concentrate more on the observations and analysis of experimental results. Another major advantage of our approaches is that we are not bound in our experiments to some specific set of sites. We can do experiments with nearly any host in the Internet by using the builtin routing, echoing facilities, or the measuring facilities.

The emulation method has proved to be a feasible and reliable approach. The WANCE tool emulates Internet communication in a laboratory environment. Based on our preliminary experimental studies, the relative errors between real and emulated experiments, in response time and throughput for distributed transaction processing application are within 3%.

Using the WANCE tools we have conducted a serials of experiments on distributed transaction processing in the Internet [24]. The experiments include comparison of the performance of the two-phrase commit protocol versus the three-phrase commit protocol, the performance of locking based and timestamp based concurrency control protocols, and the performance of different data replication strategies. These experiments with WANCE tools will have great impact in transaction processing software. It suggest changes in the ACID properties, Atomicity, Consistency, Isolation, and Durability, for the large scale distributed systems. Early results of this study can be found in another report [24].

Two interesting questions in the study of distributed system software scalability can also be answered using the WANCE tool. The first question is: "How does the distributed software work on the Internet?" and the second and more interesting question is "How well can this software be geographically distributed?" With WANCE tools we can easily vary the span of the experimental sites involving in the distributed system. We can make it very small (e.g. to choose sites from the same campus network) to very large (e.g. to choose sites from different countries). Thus we can obtain the scalability characteristics quickly.

Although the real processing is on the local hosts, the WANCE tool "steals" a few CPU cycles of the emulated remote host for routing or echoing messages. But this is the minimum resource requirement for experimenting with real communications. Compared this with the heavy load on a remote host in a real experiment, such routing and echoing actions, if used with moderation, will not affect the normal processing performance at the remote host. Also, considering the fact that many workstations in many universities are simply idle in late night or weekend, we can be justified the usage of idle resources at remote sites for scientific research. To make sure that the remote computers are indeed idle, we have used the Unix command `rup` to check the system loads before starting our emulation experiments. Moreover, we have been careful and refrained from adding high loads on the Internet. In the trace-driven experiment, we can tune the frequency of communication measurement in order to save Internet bandwidth.

Future work We have conducted a preliminary validation of our WANCE tool in a transaction processing application. We plan to continue our study on the experimental models. We will conduct further experiments to validate the WANCE tool and to find out whether the number of sites in the distributed software will affect the relative errors, and whether the geographical location will affect the accuracy of the emulation experiments. We have used the Purdue–Hong Kong case in our validation experiment, we plan to repeat it in a larger scale involving more sites in United States and other places in the world. We will start it as soon as we collected enough access to computers in these locations.

We will also test the WANCE tool for different applications. For example, we are interested in the performance of displaying multimedia interface over the Internet, which is important in digital library applications. We will first study the accuracy of the emulation experiments for these applications.

Another direction of research on the emulation experimental approach is to find out how we can emulate more than one remote sites in a single local computer. We could run multiple processes of the same distributed software in one host and have the communication between these processes route through the Internet or be delayed by real-time traces. More study is needed to investigate the model. If successful, this may allow us to emulate a large scale systems with the number of distributed system sites greater than the number of physical experimental hosts. This could be another step forward in the infrastructure for Internet experimentation.

The idea of using source routing to support the emulation experiments has other applications. For example, it could be used to emulate a mobile computing environment. In such an environment some sites of a distributed system will be moving frequently in the Internet from a physical host to another [25]. The communication packages have to hop around to try to catch the mobile site. An emulation facility can achieve this by routing the communication packages via many sites in the Internet, dynamically. This method may ease the experimentation in mobile computing and needs investigation.

The idea of measuring the real world and feed the data into the emulation model at real-time is also useful to other area of communication study. For example, it is expensive to set up a large scale wireless communication network for experimental study. But it can be emulated in a local area network by getting wireless communication measurement somewhere else and plug them into the WANCE tool. By combining the two basic technique, we can develop a variety of facilities for experimental study.

References

[1] Y. Zhang and B. Bhargava, "Wance: A wide area network communication emulation system," in *Proc. of IEEE workshop on Advances in Parallel and Distributed Systems*, (Princeton, NJ), pp. 40–45, Oct. 1993.

[2] A. Silberschatz, M. Stonebraker, and J. Ullman, "Database systems: Achievements and opportunities," *Communications ACM*, vol. 34, pp. 110–120, Oct. 1991.

[3] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[4] R. van Renesse, A. S. Tanenbaum, H. van Staveren, and J. Hall, "Connecting RPC-based distributed systems using wide-area networks," in *Proc of 7th Int'l Conf on Dist. Comp. Systems*, (IEEE, Piscataway, NJ), pp. 28–34, 1987.

[5] R. van Renesse, H. van Staveren, and A. S. Tanenbaum, "Performance of the world's fastest distributed operating system," *Operating System Reviews*, vol. 22, pp. 25–34, Oct. 1988.

[6] M. F. Kaashoek, *Group Communication for Distributed Computing Systems*. PhD thesis, Wrije Universititeit, Amsterdam, The Netherlands, Aug. 1992.

[7] C. Pu, F. Korz, and R. C. Lehman, "An experiment on measuring application performance over the Internet," in *Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, (San Diego, CA), May 1991.

[8] D. D. E. Long, J. L. Carroll, and C. J. Park, "A study of the reliability of Internet sites," in *Proceedings of the 10th Symposium on Reliable Distributed Systems*, (Pisa, Italy), pp. 177–186, IEEE, Sept. 1991.

[9] P. B. Danzig, S. Jamin, R. Cácceres, D. J. Mitzel, and D. Estrin, "An empirical workload model for driving wide-area TCP/IP network simulations," *Internetworking: Reseach and Experience*, vol. 3, Mar. 1992.

[10] L. Zhang and D. D. Clark, "Oscillating behavior of network traffic: A case study simulation," *Internetworking: Research and Experience*, vol. 1, Dec. 1990.

[11] D. Mitra and J. Seery, "Dynamic adaptive windows for high speed data networks with multiple paths and propagation delays," *Computer Networks and ISDN Systems*, vol. 25, pp. 663–679, Jan. 1993.

[12] O. W. W. Yang, X.-X. Yan, and K. M. S. Murthy, "Modeling and performance analysis of file transfer in a satellite wide area network," *IEEE Journal on Selected Areas in Communications*, vol. 10, pp. 428–436, Feb. 1992.

[13] S. A. Heimlich, "Traffic characterization of the NSFNET national backbone," in *Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, (Boulder, CO), pp. 257–258, May 1990.

[14] V. Paxson, "Measurements and models of wide area TCP conversations," Tech. Rep. LBL-30840, Lawrence Berkeley Laboratory, May 1991.

[15] J. C. Mogul, "Observing TCP dynamics in real networks," in *Proc. SIGCOMM'92*, (Baltimore, MD), pp. 305–317, Aug. 1992.

[16] A. K. Agrawala and D. Sanghi, "Network dynamics: An experimental study of the Internet," in *Proceedings of GLOBECOM'92*, (Orlando, FL), Dec. 1992.

[17] R. Cácceres, P. B. Danzig, S. Jamin, and D. J. Mitzel, "Characteristics of wide-area TCP/IP conversations," in *Proc. of ACM SIGCOMM'91 Conf*, (Zurich, Switzerland), pp. 101–112, Sept. 1991. in Computer Communication Review 21(4).

[18] D. E. Comer, *Internetworking with TCP/IP.* Englewood Cliffs, NJ: Prentice-Hall, 1988.

[19] V. Jacobson, "Congestion avoidance and control," in *Proc. SIGCOMM'88*, pp. 314–329, 1988.

[20] R. A. Golding, "A weak-consistency architecture for distributed information services," *Computing Systems*, vol. 5, pp. 379–405, fall 1992.

[21] B. Bhargava and J. Riedl, "The Raid distributed database system," *IEEE Transactions on Software Engineering*, vol. 15, June 1989.

[22] B. Bhargava, Y. Zhang, and E. Mafla, "Evolution of communication system for distributed transaction processing in Raid," *Computing Systems*, vol. 4, pp. 277–313, Summer 1991.

[23] J. Gray, ed., *The Benchmark Handbook for Database and Transaction Processing Systems.* San Mateo, CA: Morgan Kaufmann, 1991.

[24] B. Bhargava and Y. Zhang, "A study of distributed transaction processing in wide area networks," Tech. Rep. CS-94-016, Purdue University, Mar. 1994.

[25] E. Pitoura and B. Bhargava, "Dealing with mobility: Issues and research challenges," Tech. Rep. CS-93-070, Purdue University, Nov. 1993.

APPENDICES

In this appendix we explain the detail implementation issues of the WANCE tool. The first part is the configuration file for initializing the WANCE tool. The second part is the system call proxy for routing message in the emulation model. The last part is the system call proxy in the trace-based method.

The configuration file

When the program linking with the WANCE library starts up, it initializes the WANCE module. It first read in a configuration file and parses it. The configuration file is an external file that contains the emulation configuration – a list of local sites and their corresponding remote hosts being emulated. The first line in the configuration file indicates the emulation strategy used:

`Emulation:` *strategy*

strategy can be either "`source routing`", "`echo`", or "`trace-driven`", with means emulation by source routing, by echo services, or by the trace-driven method.

If it specifies "source routing", the rest of configuration file will contain lines like

my-site *local-site* *remote-site*

where *my-site* is to match the computer where the emulation program runs, *local-site* is the address of a local computer (that is to emulate computer *remote-site*), and *remote-site* is the address of the remote computer in the Internet (that is to be emulated by computer *local-site*). This line means a message from this computer to *local-site* should be routed through *remote-site* first. With regarding to the current computer, the second address in the line is the destination address, and the third is the echo address. For the example in Section 2, in configuration file would have the following line:

```
raid10      raid11      cs.helsinki.fi
```

If the configuration file specifies "echo services", the rest of configuration file will contain lines like

my-site *local-site* *remote-site:port/protocol*

here *port/protocol* is the echo services name, for example `7/UDP` for UDP echo service and `7/TCP` for TCP echo service (see Subsection 3.1). This means that from the *my-site* a message sent to *local-site* should be echoed to *remote-site* first, using *port* of *protocol*. Both *local-site* and *remote-site* can also be logical addresses interpreted by the distributed software.

If the configuration file specifies "trace-driven", the rest of configuration file will contain lines like

local-site *remote-site*

This line means that *local-site* is to emulated *remote-site*. All communication between local sites should be delayed the same way as if it were between the corresponding remote sites. For example, if `raid10` is to emulate `mit.edu` and `raid11` is to emulate `bnr.ca`, the configuration file could contain:

```
raid10      mit.edu
raid11      bnr.ca
```

Proxy system calls for emulation with source routing

When the application program starts up, the emulation module first reads the network address of the computer that it runs on. It then parses the configuration file. For each line it builds a "socket option" that sets the source routing path from the echo address to the destination: `char sockopt[12]` = {`131` (IP loose source routing), `11` (size of the option string), `4` (offset of the first address), the echo IP address, the destination IP address, `0` (padding to a quad-octet boundary)}. It then hashes the destination address and the socket option into the WANCE routing table:

destination address `struct in_addr`	socket option `char [12]`
......	
B	..., X, B, ...
......	

The emulation module provides proxy functions for system calls `accept()`, `connect()`, `sendto()`, `sendmsg()`, etc. When these functions are called, if the destination address involved are stored in the WANCE route table (i.e. specified in the WANCE configuration file), the source routing will be turned on by the `setsockopt()` system call. The pseudo code for a simplified version of these proxy system calls is giving below. The actual code is much more complicated, which also includes bookkeeping and routines to clean up the socket options, etc.

```
proxy_accept(s, addr, addrlen)
{
    newsocket = accept(s, addr, addrlen);

    /* search the route table for addr and return its socket option */
    sockopt = searchRouteTable(addr);
    if (sockopt)    /* if found */
        setsockopt(newsocket, IPPROTO_IP, IP_OPTIONS, sockopt, 12);
}

proxy_connect(s, addr, addrlen)
{
    /* search the route table for addr and return its socket option */
    sockopt = searchRouteTable(addr);
    if (sockopt)    /* if found */
        setsockopt(s, IPPROTO_IP, IP_OPTIONS, sockopt, 12);

    newsocket = connect(s, addr, addrlen);

    if (sockopt)
        setsockopt(newsocket, IPPROTO_IP, IP_OPTIONS, sockopt, 12);
}

proxy_sendto(s, msg, len, flags, to, tolen)
{
    /* search the route table for addr and return its socket option */
    sockopt = searchRouteTable(addr);
    if (sockopt)    /* if found */
        setsockopt(s, IPPROTO_IP, IP_OPTIONS, sockopt, 12);

    sendto(s, msg, len, flags, to, tolen);
}

proxy_sendmsg(s, msg, flags)
{
    for each message m in msg do {
```

```
            /* search the route table for m.addr and return its socket option */
            sockopt = searchRouteTable(m.addr);
            if (sockopt)     /* if found */
                setsockopt(s, IPPROTO_IP, IP_OPTIONS, sockopt, 12);

            sendmsg(s, message m only, flags);
        }
    }
```

The statement "`sockopt = searchRouteTable(addr);`" searches the socket option in the route table set up in the initialization for the given destination address. If the search is successful, the socket option is set up so that the subsequent message will travel on the right path. For the connectionless communication, such setting is done each time the application program calls `sendto()` and `sendmsg()`. For connection-oriented communication, the setting is done only when the application program establishes the connection by `connect()` or `accept()` calls. The subsequent transfers, by `send()` or `write()`, will inherit the routing option set to the socket. The extra overhead in searching the table is negligible when the number of sites is moderate and when proper hashing technique is used.

Empirical delay generator in the trace-driven method

Similar to implementation of emulation in the WANCE tool, we again use the "proxy" technique in implementing the trace-driven method. Instead of setting socket options, the proxy routines forward the message to a separate server, the empirical delay generator (EDG). EDG keeps a lists of messages that need to be delayed and attaches each of them with a timer. When the timer expires, EDG sends out the message to its destination. The extra trip, from sender to EDG and then to receiver, is local and can be ignored, since the delay imposed by EDG emulates that in the real Internet and is much longer.

Another sever, the Internet sampler, is always running in the LAN to sample the outside Internet. When the sampler starts, it reads the configuration file and determines the particular sites that need to be measured and the frequency of the measurement. The measurement data is collected and exported to EDG immediately. EDG depends on the measurement number to generate the random delays and packet losses for the local messages.