Mendosus: A SAN-Based Fault-Injection Test-Bed for the Construction of Highly Available Network Services

Bin Zhang, Kiran Nagaraja, Xiaoyan Li, Richard Martin, Thu D. Nguyen
{binzhang, knagaraj, xili, rmartin, tdnguyen}@cs.rutgers.edu

Department of Computer Science, Rutgers University
110 Frelinghuysen Rd, Piscataway, NJ 08854

Abstract

We describe Mendosus, a SAN-based fault-injection test-bed that supports the emulation of LAN-based systems. Our goal is to provide a test-bed for service designers to systematically assess the impact of end-to-end design decisions on the availability and reliability of network services in the presence of realistic fault scenarios. As such, our current implementation is just a first step toward this vision; currently, we are working to extend Mendosus to be able to emulate a rich set of networks, including wireless systems and WAN-connected systems. An emulator must work in real-time, providing a platform that looks and performs as similar to the deployed platform as possible; Mendosus relies critically on the high performance of the underlying SAN system to achieve this goal. Preliminary measurements show that we should easily be able to emulate 100 Mb/s and 1 Gb/s Ethernet LANs using our VIA-based implementation.

1 Introduction

Progress in hardware and networking technologies are fundamentally changing the nature of computing. Computing is no longer constrained to the desktop or static points of entry to monolithic systems. Rather, users carry multiple devices expecting to be surrounded by a rich and networked computing environment which provides a wealth of services. While these services promise unprecedented computing power, connectivity, convenience, and low cost, their construction is a challenging exercise—examples of costly failures abound (e.g., [5, 13, 16]). As such services become woven into our everyday life, users will demand ultra reliability. An excellent example is the wired telephone system. Today, we expect the telephone to always be available and working correctly; we have little patience with service failures.

Against this backdrop of increasing reliability demands by end users, today's services rely on a plethora of systems: the user's PC or PDA, the ISP's routers, and finally, the service provider's cluster [1]. Connecting these system together are an equally complex set of interconnection networks: the user's dial-up or wireless link, the ISP's WANs, and the service providers' LANs and SANs. In this universe of many interconnected systems and networks, there is a high probability of multiple component failures. Service designers know this, and so today's services are implemented to tolerate faults at many levels (e.g., [6, 14]).

Unfortunately, it is currently difficult to validate end-to-end design choices for tolerating network outages and fault conditions. Not only must the tester put together a sufficiently representative test-bed, he must also find a way of reproducing realistic fault scenarios. Often, designers must resort to simplistic and gross fault-injection techniques such as "unplugging" components by hand [10]. Such ad-hoc techniques are undesirable because they can lead to either a false sense of confidence in the system's robustness or wasted resources in over-replicating components to tolerate faults.

We are thus motivated to build a comprehensive emulation-based, fault-injection infrastructure that will allow service designers to more systematically assess the impact of faults on service reliability. As opposed to simulation or analytic modeling, an emulation-based approach allows for a balance between complexity, capturing subtle system interactions, and the ability to change the design space under investigation.

Figure 1 shows our vision: a SAN-connected cluster emulates systems interconnected by networks with lesser performance than the actual SAN. The use of a SAN is critical because the emulation support network must have higher performance in all dimensions than the emulated networks. We believe such a platform can emulate a wide range of networks, ranging from slow dial-up links, error prone wireless links, all the way to fast and robust networks such as the service provider's LAN and WAN. An important goal is to capture a sufficiently powerful fault model to emulate realistic faults, including a variety of operator induced errors.

In this paper, we describe our initial test-bed: a prototype toolkit called Mendosus which allows the emula-

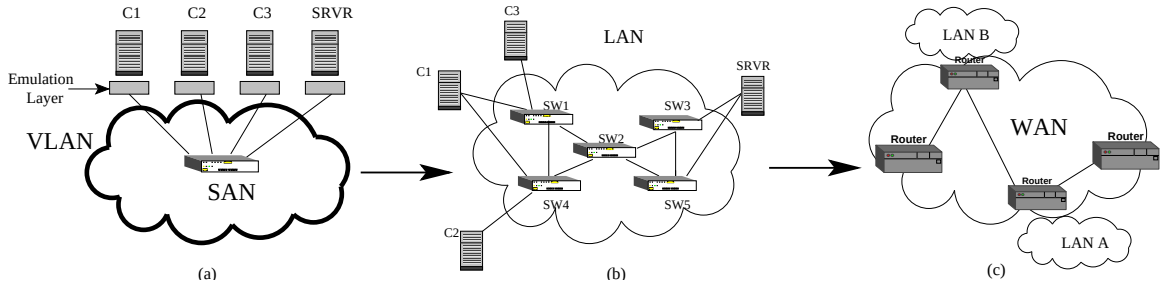

Figure 1: *Our vision is to use a SAN-connected cluster to emulate a variety of interconnected systems. The ultimate goal is to allow for emulation of complex systems, possibly comprised of multiple different LANs, including wireless, interconnected by a WAN.*

tion of a cluster of PCs comprised of arbitrary configurations of Ethernet switches, hubs, NICs, and end nodes (we call this the *virtual* network); each component can optionally have an associated fault profile. The application to be tested runs directly on the PCs. Mendosus then injects faults in real-time, while the application is running, allowing the application to be tested under realistic faulty conditions. Currently, Mendosus runs on PC clusters connected by Giganet's VIA networks. Mendosus relies on the SAN's low latency, high bandwidth, and in-order delivery to allow for the emulation of the virtual network in real-time.

The remainder of the paper is organized as follows. Section 2 describes Mendosus's overall architecture. Sections 3 and 4 describe the major components of Mendosus in more detail. Section 5 presents the results from several microbenchmarks to assess Mendosus's efficiency, and thereby its ability to emulate different networks. Section 6 discusses related work. Section 7 discusses on-going work and concludes the paper.

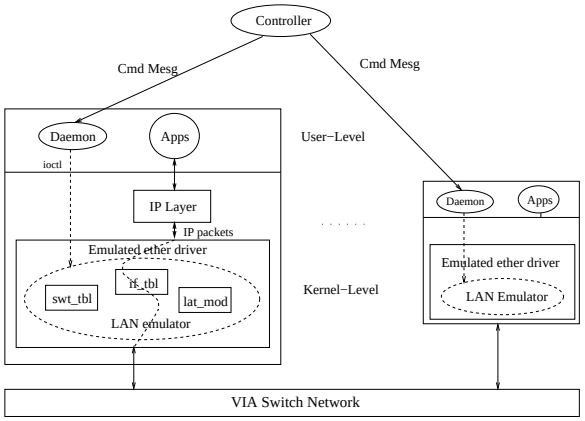

Figure 2: *Mendosus is comprised of three major components: a global controller, a daemon running on each node, and a LAN emulator embedded inside an Ethernet driver running on each node.*

2 Architecture

Our emulator is comprised of three major components, a global controller, a daemon running on each node, and a LAN emulator embedded in an emulated Ethernet driver running on each node; this architecture is shown in Figure 2. When emulation starts, the controller parses a given network configuration file and forwards this information to the daemon of each node. At each node, the daemon sets up the `/etc/hosts` file so that applications can resolve node names to IP addresses on the emulated LAN. The daemon also passes the network topology to the driver. On receiving the information, the driver performs a spanning tree algorithm over the LAN to setup a routing table.

During the emulation, the controller is responsible for injecting faults into the network. Examples include faults such as a link is down, a switch is down, and 15% of packets over a particular link should be randomly dropped. To affect a fault event, the controller contacts the appropriate nodes on the network to tell them of the fault; some events may require all nodes to be contacted, e.g., a switch failure; some events only require one node to be contacted, e.g., a misbehaving NIC. Each contacted node records in a status table maintained by the driver that the particular component is down or is operating in a faulty mode.

Each emulated NIC is presented as an Ethernet device at the corresponding node (a node may have multiple emulated NICs). When packets are handed to the Ethernet driver from the IP layer, the driver determines the emulated route that would be taken by the packet. If any of the component in the route is down, it drops the packet. If any component is in the route is in an intermittent er-

ror state, then the device driver determines whether the packet should be dropped. Otherwise, the packet is forwarded to the destination over the underlying SAN.

At the receiving end, the packet may be buffered for some time, depending on what the "actual" delay should have been in the emulated LAN, before being passed to the IP layer.

3 Controller

The controller is responsible for starting the LAN emulation based on the initial topology specification, injecting faults, and managing topology changes during an emulation run. It maintains a high level consistent view of the entire emulated LAN and can contact the per-node daemons to perform fault injections or real-time topology changes.

Configuration Specification Language. Currently, Mendosus can emulate LANs comprised of the following components: switches, hubs, ports, links, and NICs. The configuration of an emulated LAN is specified using a simple language similar to that used in ns-2 [19]. The configuration must also list the available real resources that are to be used to perform the emulation. We assume that the underlying platform is a fully connected cluster, with the interconnect being a low-latency high-bandwidth SAN.

The following example configuration specifies the network topology shown in Figure 3; the resulting emulation would be run on the real machines a.rutgers.edu and b.rutgers.edu.

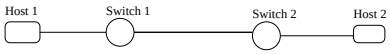

Figure 3: *Example Topology.*

```
set_host Host1
set_host Host2
set_switch Switch1
set_switch Switch2
set_nic Host1 166.111.4.1 martin.my.net
set_nic Host2 166.111.4.2 kevin.my.net
set_link Switch1 Switch2 Link1
set_connection martin.my.net Switch1 Conn1
set_connection kevin.my.net Switch2 Conn2
set_addr Host1 a.rutgers.edu
set_addr Host2 b.rutgers.edu
```

Fault Model. Each component in the emulated LAN can have an associated fault profile. Currently, the fault profile can be one of several known distributions, such as Poisson, Weibull, and constant. The fault profile can also be a trace file, specifying when a component should fail, how it should fail, and when it should be up again. In the future, we plan to build a Java interface to allow for the use of arbitrary code to determine the fault profile of a component. Table 1 shows the current supported fault types.

During the emulation, the controller is responsible for scheduling faults and recoveries according to the given fault profiles. When an event occurs, e.g, a NIC is inadvertently disconnected, the controller contacts the daemon running on the appropriate node and informs it about the event. In turn, the daemon performs an `ioctl` call to the driver with the event. The driver will set the status of the affected emulated component appropriately.

Real-Time Topology Changes. Change in the underlying platform is an unavoidable fact of life. As highly available services continue to operate across spans of years, components will need to be changed and upgraded. Further, the platform may need to be extended or reconfigured to better meet the changing usage patterns of clients. Thus, service providers must be able to assess the impact of changing the computing platform while the service remains on-line.

Mendosus provides support for service designers to change the topology of the emulated system in an ongoing emulation study. Table 2 lists the currently supported operations. Currently, changes in topology are specified in the configuration file along with a time at which they should be enacted. We are working on an interactive interface that would allow the user to modify the topology in real-time.

4 Per-Node Daemon and Device Driver

Much of the per-node implementation of Mendosus resides inside our LAN emulator, which is embedded in an emulated Ethernet driver. The LAN emulator, as shown in Figure 2, maintains three tables as follows:

Switch table This table contains all virtual switches and hubs in the emulated LAN. A virtual switch or hub is comprised of a group of ports, and a field recording its parent switch in the spanning tree.

Network interface table This table contains all network interfaces, their IP addresses, and the

Fault Type	Corresponding Component	Corresponding Fault	Fault parameters
ft_nic_down	Network Interface	NIC temporarily down (eg. driver hangs)	down time
ft_conn_down	NIC to Switch connection	Connection temporarily down (eg. unplug)	down time
ft_sw_down	Switch	Switch temporarily down (eg. power failure)	down time
ft_sw_port_down	Switch	One port temporarily down (eg. duplex mismatch)	down time
ft_link_down	Switch to Switch Link	Link temporarily down (eg. faulty cable)	down time
ft_sw_pktdrop	Switch	Switch drops packet (eg. queue overflow)	drop rate
ft_sw_pktloss	Switch	Switch losses packet (eg. routing error)	loss rate
ft_link_pktloss	Switch to Switch Link	Link drops packet (eg. unprotected cables)	loss rate
ft_link_delay	Switch to Switch Link	Link introduces packet delay (eg. congestion)	loss rate

Table 1: *Types of faults that Mendosus can currently inject.*

Event Type	Corresponding Component	Corresponding Event	Event parameters
topo_nic_rm	Network Interface	NIC removed from the host	N/A
topo_nic_add	LAN Interface	New NIC added	host and virtual-IP
topo_conn_rm	NIC to Switch Connection	Connection removed from the network	N/A
topo_conn_add	NIC to Switch Connection	Connection added to the network	NIC,switch-id and connection-id
topo_sw_rm	Switch	Switch removed from the network	N/A
topo_sw_add	LAN	New switch added to network	switch-id
topo_link_rm	Switch to Switch Link	Link removed from the network	N/A
topo_link_add	LAN	New Link added	both switches and link-id
topo_reconfig	LAN	Switches should reconfig the spanning tree	N/A

Table 2: *Types of topology changes that Mendosus currently support.*

switch/hub that they are connected to.

Connection line table A link between switches and between a switch and an interface is represented by a connection line, which is one entry of the connection line table. For a connection between two switches, the IDs and ports of these two switches are stored in an entry; for a connection between a switch and an interface, the ID and port of the switch and IP address of the interface are stored.

When first started, the LAN emulator uses these three tables to generate a breadth-first spanning tree for routing. The spanning tree is kept as part of the switch table: that is, as already mentioned, each entry for a switch or hub has a pointer to the parent switch/hub in the spanning tree. When a packet is to be sent over the emulated LAN, we walk up the spanning tree from the given source and destination IP addresses to find the least common ancestor. This determines the route that the packet will take from the source to the destination in the emulated LAN. If any component in this path is in a faulty state, the packet is dropped (if the component is in an intermittent error state, then the packet may be dropped according to some random distribution).

The LAN emulator supports multiple virtual network interfaces on each host, which is implemented with IP aliases. Thus, our emulator can emulate network interface failover. That is, when one virtual network interface goes down, the virtual IP address associated with this interface is automatically remapped to another active virtual interface on the same host.

The LAN emulator provides an ioctl-based interface for user-level manipulation of these tables. The main purpose of the per-node daemon is to serve as a conduit for information to be passed from the controller to the driver via this ioctl interface.

Currently, our LAN emulator is embedded inside the emulated Ethernet driver provided by Giganet as part of the cLAN driver. Our implementation currently runs on Linux 2.2.14 with IP aliasing and Giganet cLAN 1.3.0.

5 Performance

We measured the performance of our emulator using two sets of microbenchmarks. These measurements are meant to give the upper bound on the efficiency of networking systems that we can support. That is, we cannot emulate networks that have lower latency or higher throughput than our VIA SAN is capable of delivering. On the other hand, the system should be useful in emulating any networking system that has higher latency and lower throughput capabilities.

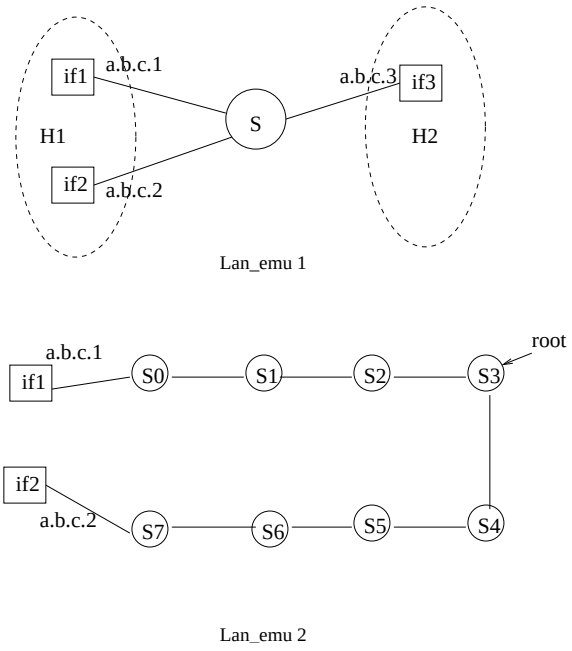

Figure 4: *Emulated systems used for measuring the capability of our emulation network. In Lan_emu 1, messages are being sent from if1 to if3 through 1 intervening switch. In Lan_emu 2, messages are being sent from if1 to if2 through 8 intervening switches.*

Figure 4 shows two LAN configurations that we used to measure the round trip latency of UDP messages and throughput of TCP connections. For latency, our benchmark sent UDP messages with 1 byte of data. For throughput, our benchmark sent 10K messages through the TCP connection. Table 3 shows the results. Note that the bulk of the overhead arises from the TCP/IP and UDP/IP stacks—raw VIA latency is on the order of 8 to 10 usecs—with no noticeable degradation of performance with increasing number of switches that must be traversed in the emulated LAN. These results show that we can easily emulate 100 Mb/s Ethernet networks; we should be able to emulate 1 Gb/s Ethernet network as well (although we cannot emulate level 3 & 4 functionalities that may be supported by switches in these networks). Further, we should be able to emulate a wide range of wireless networks.

6 Related Work

Our work primarily relates to contributions in the fields of network emulation and fault injection, though we borrow ideas from simulation work such as ns-2 [19]. The work at Utah, Emulab [18], a configurable Internet emulator of 330 nodes, offers a testbed for distributed applications. A fraction of the nodes, can be configured as routers, traffic generators or shaper nodes which allow control over packet delays, re-orderings and losses. However, they do not consider failure of individual network components and its affect on routing, connectivity and application behavior.

Unlike our distributed emulation model, the Lancaster emulator[3] uses a centralized process which moderates all traffic flow in the network. NISTNET [12] emulates a single configurable router which routes all traffic between the end-points in WAN-based systems and implements delay distributions, packet re-orderings, packet drops and duplication. The objective of these works is to test the effect of network performance dynamics on sensitive applications and not failure of network components in general. Our emulation toolkit, while not limited to, allows application performance evaluations similar to the above works and the logP [11] work which studies the communication performance of parallel applications to variations in the latency, overhead and bandwidth of the underlying network. While our implementation is at the kernel level and involves no modification or recompilation of the application, DelayLine[8], a user level, WAN emulator library provides a socket interface and involves minimal additions and recompilation of the applications.

In fault-injection research there are numerous projects that look at both hardware and software based fault-injection techniques [4, 7, 9, 15]. Our approach is similar to Orchestra [4], which uses a software based, script-driven probing and fault-injection method to test distributed protocols by introducing a layer in the protocol stack. In [17] embeds the fault injection layer in the NIC and studies how the application is affected by packet drops, corrupted messages, delays, interface resets and hangs. All of the above introduce faults locally and are useful for testing application behavior locally. Though our injection technique is fundamentally similar, we introduce network component failures consistently across entire network and hence observe the effects on a global level. Loki [2] a fault injector from UIUC injects correlated faults by maintaining partial view of global state of the system. Our focus currently is on hardware faults and extending it to inject software correlated faults will be possible by using our central fault injection engine for consistent global state.

7 Conclusions and Future Work

As availability becomes an increasingly important "quality" metric for network services, the need for a re-

	Number of Switches	Round-Trip Latency (usec)	Throughput (MB/sec)
Lan_emu 1	1	54.7	74.1
Lan_emu 2	8	54.6	75.7
Ethernet	N/A	87.6	11.1

Table 3: *Mendosus base latency and throughput.*

alistic, emulation-based test-bed is rapidly emerging. In this paper, we presented Mendosus, a SAN-based fault-injection test-bed that has been designed to allow for emulation of LAN-based systems. Our goal is to provide a test-bed for service designers to systematically assess the impact of end-to-end design decisions on the availability and reliability of network services in the presence of realistic fault scenarios. As such, our current implementation a first step toward this vision.

Future work will extend Mendosus to emulate more classes of networks. In particular, our next objectives will be directed toward wireless LAN's and WANs. The challenges in the wireless space include incorporating realistic fault models, managing the extreme performance mismatch between the wireless network and the SAN, and properly emulating the broadcast nature of the wireless network over point-to-point SAN links.

Future work emulating WAN scenarios includes managing the large volume of traffic through the routers and long-delay links. In addition, correctly emulating WAN router behavior, with their myriad of congestion control algorithms, presents an especially significant challenge.

References

[1] E. Brewer. Lession Learned from Internet Services: ACID vs BASE. Talk presented at the National Science Foundation sponsored workshop on Industrial/Academic Cooperation in Database Systems, Oct. 1998.

[2] M. Cukier, R. Chandra, D. Henke, J. Pistole, and W. H. Sanders. Fault injection based on a partial view of the global state of a distributed system. In *Symposium on Reliable Distributed Systems*, pages 168–177, 1999.

[3] N. Davies, G. S. Blair, K. Cheverst, and A. Friday. A network emulator to support the development of adaptive applications. Technical report, 1995.

[4] S. Dawson, F. Jahanian, and T. Mitton. ORCHESTRA: A fault injection environment for distributed systems. Technical Report CSE-TR-318-96, 26 1996.

[5] Electronic News. Ebay Outages Cast Clouds on Sun. http://www.electronicnews.com/enews/Issue/1999/06211999/25ebayah.asp, June 1999.

[6] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, Distributed Data Structures for Internet Service Construction. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000)*, 2000.

[7] S. Han, K. Shin, and H. Rosenberg. Doctor: An integrated software fault injection environment for distributed real-time systems, 1995.

[8] D. B. Ingham and G. D. Parrington. Delayline: a wide-area network emulation tool. *USENIX, Computing Systems*, 7(3):313–332, 1994.

[9] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham. FERRARI: A tool for the validation of system dependability properties. In *FTCS-22: 22nd International Symposium on Fault Tolerant Computing*, pages 336–344, Boston, Massachusetts, 1992. IEEE Computer Society Press.

[10] Manoj Joshi and Nicholas Brenton, Cisco Systems, Jim Helfrich and Patrick Jones, Network Appliance. High availability for network-attached storage. Technical Report TR 3115, Network Appliance and Cisco Systems, 2000.

[11] R. Martin, A. Vahdat, D. Culler, and T. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. In *International Symposium on Computer Architecture*, Jun 1997.

[12] National Institute of Standards and Technology(NIST). NISTNET emulator. Available at http://snad.ncsl.nist.gov/itg/nistnet/.

[13] Reuters. Britannica Competitors Study Net Strategy. http://news.cnet.com/news/0-1005-200-1435449.html, Nov. 1999.

[14] Y. Saito, B. N. Bershad, and H. M. Levy. Manageability, Availability and Performance in Porcupine: A Highly Scalable Internet Mail Service. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, Charlston, SC, Dec. 1999.

[15] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson, and T. Lin. FIAT w fault injection based automated testing environment. In *Proc. 18th Int. Symp. on Fault-Tolerant Computing (FTCS-18)*, pages 102–107, Tokyo, Japan, 1988. IEEE Computer Society Press.

[16] SiliconValley.internet.com. Ebay Outage Twice This Week. http://siliconvalley.internet.com/news/article/0,„3531_435741,00.html, Aug. 2000.

[17] D. T. Stott, G. Ries, M.-C. Hsueh, and R. K. Iyer. Dependability analysis of a high-speed network using software-implemented fault injection and simulated fault

injection. *IEEE Transactions on Computers*, 47(1):108–119, 1998.

[18] University of Utah Flux Research Group. Utah Network Testbed. Available at http://www.emulab.net.

[19] USC/ISI, UCSB, CMU. Network Simulator ns-2. Available at http://www.isi.edu/nsnam/ns/.