A System Demonstration of ST-TCP

Manish Marwah Shivakant Mishra
Department of Computer Science,
University of Colorado,
Campus Box 0430, Boulder, CO 80309-0430

Christof Fetzer
Department of Computer Science,
Dresden University of Technology
Dresden, Germany D-01062

Abstract

ST-TCP (Server fault-Tolerant TCP) is an extension of TCP to tolerate TCP server failures. Server fault tolerance is provided by using an active-backup server that keeps track of the state of a TCP connection. The backup server takes over the TCP connection if the primary server fails. This take-over is fast, seamless, and completely transparent to the client. This paper provides a system demonstration of a new ST-TCP prototype. The new prototype incorporates a performance-enhanced architecture and addresses application failure scenarios. Five experiments using this prototype are proposed to demonstrate the following useful features: (1) Client-transparent, seamless failover; (2) Insignificant performance overhead of ST-TCP during failure-free periods; and (3) An ability to tolerate all single crash failures at the hardware and operating system levels, and, most crash failures at the application level.

1 Introduction

Web-based services have grown rapidly in the past few years. Products and services sold over the Internet are a substantial part of the revenue of many companies. In fact, there are a significant number of companies that solely depend on the Internet for all their revenues. For these companies in particular, and all companies in general, service outages can be very expensive. For example, an hour of service outage results in an estimated revenue loss of $200K for Amazon Inc. and $6 million for a brokerage firm [4]. Thus, any improvements made towards limiting such service outages can have substantial economic benefits. ST-TCP [2] aims to minimize these service outages by extending TCP to provide TCP connections that can tolerate server failures.

TCP is the protocol of choice for a wide range of popular distributed applications such as http, ssh, FTP, sendmail and Samba. It provides a reliable, ordered, connection oriented, duplex communication stream. In addition, it also provides flow and congestion control. However, it does not provide server fault tolerance. Fault tolerance support at the TCP layer can be very advantageous, and can prove to be more effective than at the application layer. To enable fault tolerance in a TCP based client-server application, cooperation of the client in implementing server fault tolerance is usually required. If any changes are made to this fault tolerance mechanism that affect the client, those changes must be propagated to all the clients. This may be inconvenient at best, and not possible at worst, since the clients are typically widely spread out, numerous, and usually not under the control of the organization that runs the servers. However, if fault tolerance is supported at the TCP layer (transport layer), clients can be made oblivious to server fault tolerance. Thus, implementing fault tolerance in server applications (especially ones that have already been deployed in the field), with fault tolerance support at the TCP layer, is more effective: no fault-tolerance related changes are ever required on the clients.

In light of these potential advantages, several research projects have recently addressed the issue of providing server fault-tolerance in TCP. These include [7], [6], [5], [3], [1], and our system ST-TCP[2]. ST-TCP provides server fault tolerance with the following important properties: (1) No changes are required at the client—neither in client-side TCP code, nor in the client-side application; (2) No functional deviation and insignificant performance deviation from standard TCP behavior during normal (failure-free) operation; (3) Fast and seamless failover during a server failure; and (4) Tolerate all single crash failures at the hardware/operating system layer, and most application crash failures.

In [2], we described an initial design and a prototype implementation of ST-TCP, and presented performance measurements of this initial prototype. In this paper, we describe some design enhancements made to ST-TCP; we describe in detail how ST-TCP addresses different failure scenarios, including application failures, and, finally, we describe the details of the five different experiments that we plan to perform at the conference.

2 Overview of ST-TCP

A schematic of the ST-TCP architecture is shown in Figure 1. It is a primary-backup system with an active backup. The backup taps the traffic between the client and the primary and delivers it to a replica of the primary application running on the backup. The backup uses the same virtual IP address and port number as the primary. Further, during TCP connection initialization, the backup changes its initial sequence number to match that of the primary. This allows it to take over the client-primary TCP connection in the event of the primary's failure. ST-TCP assumes that the primary application is deterministic, that is, the primary application and its replica on the backup go through the same states and produce exactly the same responses to the client if they are supplied with the same input TCP stream.

The application on the backup also generates all client responses, but the network stack on the backup does not send them to the client. However, the client does receive these

Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)
0-7695-2282-3/05 $20.00 © 2005 IEEE

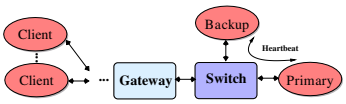

Figure 1: ST-TCP architecture.

segments from the primary. The acknowledgments sent by the client in response to these segments serve as acknowledgments for both the primary and the backup In addition, the primary makes sure that the backup has received a particular byte from the client before discarding it from its buffer. It uses extra TCP receive buffer space for this purpose, keeping client bytes in its buffer until it is notified of their successful receipt by the backup.

A heartbeat (HB) mechanism exists between the primary and the backup for detecting failures. When the backup detects that the primary has failed, it stops suppressing the output segments that it generates for the client and takes over the client-primary TCP connection. The failover appears seamless to the client since it takes a very short time and the same IP address, port number and sequence numbers are used by the backup. Before taking over, the backup also powers the primary down to prevent any danger of dual active servers.

3 Enhancements to ST-TCP

TCP state exchange between primary and backup. In the earlier architecture of ST-TCP [2], the backup received not only all traffic from the client to the primary, but also all traffic from the primary to the client. We observed that this leads to an overloaded NIC (network interface card) or/and CPU on the backup server. In particular, in some scenarios, the backup starts lagging behind the primary. In the initial ST-TCP prototype, the primary interpreted this situation as the backup being failed. This performance issue was addressed, to some extent, in the initial ST-TCP prototype by adding an additional NIC and CPU. One NIC could be used for the client-primary traffic while the other could be used for the primary-client traffic.

However, during our experiments, we noted that the backup does not need to receive the primary-client traffic at all. These segments were used by the backup for the following purposes:

1. The backup examines the sequence numbers acknowledged by the primary in these segments to determine if it has missed any client segments that were received by the primary (and not by the backup).

2. The backup uses these segments as an indication in some situations that the primary has crashed, e.g., if the primary app. crashes but the HB stays up.

Both of these requirements can be addressed without the backup receiving primary-client segments. This can be achieved by having the following information included in the heartbeat messages exchanged between the servers - A) the sequence number of the latest byte received from the client (`LastByteReceived`), and B) the sequence number of the latest byte written to the TCP send buffer by the application (`LastAppByteWritten`). The backup can use the information in A and B for recognizing conditions mentioned in 1 and 2 above, respectively.

The current design of ST-TCP implements this new mechanism and so does not need any additional hardware. Also, this ensures that the backup does not receive and process any more traffic than the primary. However, it is required that the backup machine be preferably faster or at least as fast as the primary. Further, the workload on the backup should not be any more than that on the primary. This would ensure that during normal operation the backup does not excessively lag behind the primary, which could make the primary suspect that the backup has failed. Note that the primary lagging behind the backup is not an issue since the client sends and acks data depending on the primary's response.

Heartbeat Mechanism. In the earlier architecture of ST-TCP [2], the heartbeat (HB) mechanism was implemented by a UDP channel over the IP link. This mechanism created some scenarios where a single failure could not be correctly detected. For example, if the backup NIC failed, the backup would stop receiving regular heartbeat messages and conclude that the primary has failed. In this situation, it will shut down the primary and attempt to take over the TCP connection.

To address this problem, in the new ST-TCP architecture, the HB is exchanged between the primary and the backup over two diverse links. One is over the IP link as before; the second is over a serial link. This secondary link is established by directly connecting the serial ports of the two machines using a null-modem cable. These dual HB links allow the primary and secondary to continue to exchange HB information despite single failures. It also provides a better failure detection in some scenarios, e.g., the one outlined above.

Heartbeat carries the following information: (1) Last byte received from the client (`LastByteReceived`); (2) Last ack received from the client (`LastAckReceived`); (3) Last byte written by the application to the TCP send buffer (`LastAppByteWritten`); and (4) Last byte read by the application from the TCP receive buffer (`LastAppByteRead`). In addition, the information about the generation of a TCP FIN or TCP RST segment is also communicated through the HB. In situations where the primary and the backup send ping requests to the gateway (see Section 4.3), the results of these requests are exchanged via the HB.

The serial link uses RS-232 protocol and typically has a maximum speed of 115.2 kbps. The HB is less than 20 bytes per TCP connection, and assuming a HB every 200ms, this translates to a bandwidth of 0.8 kpbs per TCP connection. Thus, the serial link provides enough bandwidth for around 100 simultaneous TCP connections. If it is expected that the server will be supporting more connections, then it is best to use an additional 10/100 mbps Ethernet NIC on the primary and backup instead of a serial connection. The NICs can be directly connected by a crossover Ethernet cable.

Application Crash Failure. One limitation of providing server fault tolerance at the TCP layer is that it may not be able to adequately handle some server application failures. In particular, consider a situation where the application running on one of the servers (primary or backup) crashes, while its replica running on the other server continues to work correctly. Server fault tolerance provided at the TCP layer is limited in its ability to handle all possible scenarios that may arise under this situation.

Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)
0-7695-2282-3/05 $20.00 © 2005 IEEE

In the new ST-TCP design, we have enhanced failure detection mechanism to detect most application crash failures. All application crash failures cannot be detected since ST-TCP is limited to the information available at the transport layer. Further, since it is a primary-backup system, if the primary application and its replica differ in their response, e.g., one produces a FIN and the other does not, additional information is needed to determine whether the primary or the backup application has failed. The various application failure scenarios including the specific failure instances that ST-TCP may not be able to detect are described in the next section.

4 Failure Detection and Recovery

An important goal of ST-TCP is to tolerate all single crash failures at the server. These failures could be at the hardware, operating system or application level. In the initial ST-TCP prototype, fault tolerance mechanisms mainly addressed HW/OS crashes. However, a common failure scenario is one in which there are no HW/OS crashes and one of the application replica (primary or backup) crashes or hangs, while the other continues to function correctly. This can happen because of differences in resources available on the primary and backup server, e.g., the application on the primary may run out of memory, while the application on the backup has sufficient memory available. We have enhanced ST-TCP to address these scenarios.

A crash failure model is assumed (Byzantine failures are not supported). ST-TCP handles all HW/OS failures, and all those application failures whose underlying cause can be traced back to the HW/OS (e.g., failures arising from memory allocation errors). Specifically, ST-TCP cannot handle failures resulting from software bugs in the application since these are likely to manifest both on the primary and the backup (and thus would count as a double failure). When a failure occurs at the primary server, the client sees one of the following two events: (1) No response from the server, e.g., a crash failure of the server due to HW or OS crash. (2) A TCP FIN or RST segment from the server indicating socket closure, e.g., OS closes socket after application has crashed.

In ST-TCP, single crash failures are masked from the client by suppressing the corresponding failure event and migrating the TCP connection to the backup. Table 1 summarizes all single crash failure conditions, symptoms observed and recovery actions taken.

4.1 HW/OS Crash Failures

A HW/OS crash failure causes the primary to stop sending/receiving any data on the TCP connection (i.e. it does not send or ack any bytes). The backup concludes that the primary server has crashed if it detects HB failure on both links (IP and serial links). The underlying assumption here is that a single failure cannot take both links down, unless the primary has failed. In this case, the backup takes over the TCP connection and shuts the primary down.

Similarly, the primary concludes that the backup has crashed if it detects a HB failure on both links. The primary shuts the backup down and runs in non fault-tolerant mode.

4.2 Application Crash Failures

Managing application failures is more complicated. Since HW/OS is functioning correctly in these scenarios, the TCP layer stays up and HB between the servers also stays up on both the links. It is convenient to consider two separate cases here depending on whether the OS cleans up or does not clean up the failed application state.

4.2.1 Application Crash Failure without Cleanup

In this case, the application has failed, but the TCP socket is not closed. This can happen if neither the OS detects the application failure nor does the application itself close the socket due to the failure. The key point here is that a TCP FIN or RST segment is not generated. No data is sent or received by the application layer, but client bytes received by TCP are acknowledged by the TCP layer as long as the receive buffer does not fill up. Further, any bytes already in the send buffer may be sent out to the client. If the application running on the primary fails in this manner, the backup can observe one or both of the following about the primary application: (1) The application does not read any bytes from its TCP receive buffer (2) The application does not write any bytes to its TCP send buffer.

The last byte read and written by the application on the primary is available to the backup through the HB mechanism. Failure detection is based on the following two parameters: (1) The number of bytes that the primary application lags the backup by (`AppMaxLagBytes`) (2) The time duration for which some number of bytes, already read or written by the backup application, have not been read or written by the primary application (`AppMaxLagTime`).

The two threshold values, `AppMaxLagBytes` and `AppMaxLagTime`, are configurable parameters and influence the failover time. A simple failure detection criteria would be: the primary application is considered failed if it lags behind the backup application by `AppMaxLagBytes` for a short duration of time (e.g., a few sec.) or, a particular byte read/written by the primary application lags the corresponding one at the backup by `AppMaxLagTime`. It is possible that the primary application has not really failed but just degraded in performance. However, if the failure criteria are met, the performance degradation is considered to be severe enough to warrant a failover. There is no danger of a dual active server here, because the backup powers down the primary [2] before taking over the TCP connection. The primary uses a similar criteria for detecting failure of the backup application.

In some instances - when there is no activity on the connection - failure detection may be delayed. However, these failures will be detected when the connection is used again. It should be noted that ST-TCP detects all application failures of the type discussed in this subsection, that is, where a FIN or RST segment is not generated.

4.2.2 Application Crash Failure with Cleanup

In this case, the application failure is detected by the OS. As part of the application cleanup, the OS closes the TCP connection. An example of such a failure is an application crashing as a result of receiving a SEGV signal. Such a failure could also occur if the application detects a failure (e.g., a memory

Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)
0-7695-2282-3/05 $20.00 © 2005 IEEE

	Failure	Location	Symptom	Recovery Action Taken
1	HW/OS crash failure	Primary	Backup detects HB failure on both links	Backup takes over the TCP connection and shuts primary down
		Backup	Primary detects HB failure on both links	Primary runs in non fault-tolerant mode and shuts backup down
2	Application failure (TCP FIN/RST not generated)	Primary	Primary app. lags backup app. by `AppMaxLagBytes` or by `AppMaxLagTime` (see Section 4.2.1)	Backup takes over TCP connection and shuts primary down
		Backup	Backup app. lags primary app. by `AppMaxLagBytes` or by `AppMaxLagTime` (see Section 4.2.1)	Primary runs in non fault-tolerant mode and shuts backup down
3	Application failure (TCP FIN/RST generated)	Primary	TCP FIN/RST generated at primary but not at backup; same symptoms of app. failure as in 2 above (see Section 4.2.2)	FIN/RST suppressed for `MaxDelayFIN`; if failure detected, backup takes over the TCP connection and shuts primary down
		Backup	TCP FIN/RST generated at backup but not at primary; same symptoms of app. failure as in 2 above (see Section 4.2.2)	FIN/RST discarded; if failure detected, primary runs in non fault-tolerant mode and shuts backup down
4	NIC or cable failure	Primary	Both primary and backup detect HB failure on IP link (but not on serial link); backup receives client data but primary does not, or, backup can ping gateway, but primary cannot (see Section 4.3)	Backup takes over the TCP connection and shuts primary down
		Backup	Both primary and backup detect HB failure on IP link (but not on serial link); primary receives client data or acks but backup does not (see Section 4.3)	Primary runs in non fault-tolerant mode and shuts backup down
5	Temporary Network failure	Backup	HB on both links up; backup does not receive some client bytes received by primary	Backup server requests and receives missed bytes from the primary
		Primary	Primary misses bytes; client retransmits	None required; normal TCP behavior

Table 1: Single Failure Scenarios

allocation error) and closes the TCP connection.

The main challenge in detecting this kind of failure is to be able to distinguish between a TCP FIN generated due to a normal closure of the socket and that generated due to an abnormal one. To understand the complexity of this scenario, consider that the application on the primary fails, and the TCP on primary generates a FIN as a result. If this FIN is sent to the client, the TCP connection will be terminated. This is despite the fact that the application on the backup is running correctly.

To address this scenario, ST-TCP requires that a server generating a FIN should immediately communicate the FIN to the other server through the HB. If the primary generates a FIN, it sends it to the client as soon as it learns (via HB) that the backup has also generated a FIN. This scenario is a normal closure of the TCP socket. While a failure can also result in both the primary and the backup producing a FIN, it is a case of double failure and is not currently handled by ST-TCP.

The interesting cases are where the primary and the backup disagree, i.e., only one of them produces a FIN. In this case the server generating the FIN delays sending it to the client for a short period of time, `MaxDelayFIN`, e.g., 1 minute. This is to account for cases where a failure may be detected via other indications during this time. In fact, by delaying the FIN temporarily, this failure scenario during the delayed time period becomes identical to the one described in Section 4.2.1, where no FIN is produced. However, if at the end of `MaxDelayFIN` a failure is not detected, it is assumed that the behavior of the primary is correct.

We decided not to do a failover whenever the primary produced a FIN and the backup did not, since it is quite possible that it is the backup that has failed and hence not produced a FIN for normal socket closure. It should be noted that the primary always immediately sends out a FIN if it has already received a FIN from the client. Furthermore, note that during normal operation – when neither the primary nor the backup has failed – the FIN is not delayed by `MaxDelayFIN`. This only happens if there is a failure.

All failure scenarios where only one of primary or backup generates a FIN are summarized below.

1. Primary application has failed; Backup is working correctly.

   Primary generates a FIN, but backup does not generate a FIN. Here the primary application fails and a FIN is generated. The primary delays sending the FIN for `MaxDelayFIN` time units. During this time it is likely that the backup will detect the primary failure (if the application performs read/write operations on the socket). The backup will shut the primary down and take over the TCP connection. However, if the failure is not detected within `MaxDelayFIN` time units, the primary will send out the FIN to the client.

   Primary does not generate a FIN, but backup generates a FIN. Here the primary application fails and a FIN is not produced. The backup generates a FIN due to normal socket closure. This FIN on the backup is dropped like any other segment sent to the client. The FIN is treated specially here, and although it has a sequence number, it is not considered in the failure detection criteria. The backup will detect the primary application failure if there are other bytes that the backup application reads/writes but the primary does not. In that case, it will shut off the primary, take over the TCP connection, and retransmit the FIN (in fact, the backup has already been retransmitting and dropping the FIN).

2. Primary is working correctly; Backup application has failed.

   Primary generates a FIN, but backup does not generate a FIN. In this case, the primary produces a FIN due to normal socket closure, but since the backup application has failed, the backup does not generate a FIN. Here the primary will wait for at most `MaxDelayFIN` time units before sending out the FIN. Meanwhile, if it

Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)
0-7695-2282-3/05 $20.00 © 2005 IEEE

detects backup's application failure, it will send out the FIN immediately.

Primary does not generate a FIN, but backup generates a FIN. Here the backup TCP generates a FIN due to a failure. This FIN is suppressed. The primary transits to non fault-tolerant mode. This happens at `MaxDelayFIN` if the primary is unable to detect backup's application failure; otherwise, it happens at the time the primary detects backup's application failure.

In the application failure cases described above, the failures are very likely to be detected if the application is reading/writing bytes when the failure occurs. However, if there is no application activity, the application failure may not be immediately detected. This is not a problem for application failures described in the previous section where a FIN is not generated (since failure will be detected when there is some application activity). However, in application failure situations where a FIN is generated, it may not be possible to conclusively distinguish between a normal closure and a failure based solely on information at the TCP layer. This is true for any primary/backup system.

To be able to detect application failures under all circumstances, either additional backup servers have to be deployed, or some additional information is needed from the application layer. Deploying additional backup servers will allow a majority decision to be taken in case of a conflict between the primary and a backup. For additional information from the application layer, an application can support a watchdog mechanism where the application continually sends a heartbeat to a watchdog. The watchdog monitors the application health and informs ST-TCP in case of any failure suspicion.

4.3 Local Network Failures

In this section we discuss network failures that are local to the primary or the backup server, e.g., a NIC failure in a server. We have assumed that both the primary and the backup have a single NIC. Recall from Section 3 that the HB between the primary and the backup is exchanged on two separate links – an IP link and a serial link.

If a local network failure occurs, only the HB on the IP link fails. The servers continue to exchange the HB on the serial link. This enables the two servers to determine that a local network failure has occurred. To determine if the failure has occurred at the primary or at the backup, the servers examine the "last client byte received" information (`LastByteReceived`) in the HB. If the client is sending data, then the server with the NIC failure will not receive them, while the server without NIC failure will receive them. Based on the `LastByteReceived` in the HB, the primary or the backup can determine if the other is lagging behind in terms of the client bytes received. For example, if the backup determines that the primary has lagged behind by greater than a threshold number of bytes, or, that a particular byte has not been received by the primary for more than a threshold period of time, then it shuts the primary down and takes over the connection. These threshold values are configurable. Similarly, the primary shuts the backup down if it determines that the backup is lagging behind.

One limitation of this failure detection method is that it depends on the client sending data. There are several applications, e.g. FTP, that do not require client to send a lot of data. In such cases, this method of failure detection does not work. This problem can be partially solved by having the primary and backup look at the acks received from the client. If the backup NIC is down, the latest client ack information (`LastAckReceived`) received by the primary from the backup via the HB on the serial link will indicate that the backup is behind. However, this does not work if the primary NIC has failed. If the primary NIC is down, the client will not receive any bytes from the server and thus not send any acks.

We have added another mechanism in the new version of ST-TCP to address this case. When the servers detect a failure of the HB on the IP link but not on the serial link, both the primary and the backup send ping requests to their gateway. The results of these requests - that is, if they succeeded or not - are exchanged in the HB via the serial link. If ping requests continue to fail for the primary but succeed for the backup, the backup takes over the TCP connection and shuts the primary down.

Temporary local network failures. Temporary failures in the NIC or the IP stack (e.g. buffer overflow) can lead to packets being dropped. HB stays up on both the links in this case. If the packets are dropped at the backup, the backup requests the primary for the missing bytes. There may be cases where the backup takes a long time to catch up or is unable to catch up. If the additional receive buffer space at the primary fills up, the primary considers the backup failed and runs in non fault-tolerant mode. Note that temporary network failures at the primary are not an issue since these will be taken care of in the normal course of TCP operation – the primary does not ack these bytes and therefore the client will retransmit.

If the primary crashes while the backup is retrieving missed bytes from it, the backup has no way of obtaining these bytes, since primary has already acked them. For critical applications, a logger can be added to the system to address this output commit problem as described in [2]; for other applications, ST-TCP treats this failure as unrecoverable.

5 Planned Demonstrations

We present five experiments that we plan to demonstrate live at the symposium. These experiments are designed to demonstrate different aspects of ST-TCP: (1) Client-transparent, seamless failover to the backup server when the primary fails; (2) Dependence of failover time on HB frequency; (3) Insignificant overhead of ST-TCP during failure-free operation; (4) Failure detection and recovery in case of application crash failures; and (5) Failure detection and recovery in case of NIC failures.

Experimental Setup. Figure 2 shows the experimental setup used for the experiments. An Ethernet switch is used to connect the primary and the backup. We also have the client directly connected to the same switch. The primary and backup are installed with a modified Linux kernel, incorporating changes required to support ST-TCP. Virtual NICs are created using the IP aliasing feature of the Linux kernel on both the primary and backup machines. These VNICs are assigned serviceIP IP address which is the address that the clients connect to for receiving service. The primary and backup are

Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)
0-7695-2282-3/05 $20.00 © 2005 IEEE

also associated with a multicast Ethernet address – multiEA. There is a static ARP protocol entry created on the gateway (the client in this case) mapping serviceIP to multiEA. Any IP packet destined for serviceIP is sent to the multiEA Ethernet address which allows both the primary and the backup to receive all packets sent to the serviceIP address. The HB is exchanged between the servers over the IP link. A duplicate copy of the HB is exchanged over a secondary link as described in Section 3.

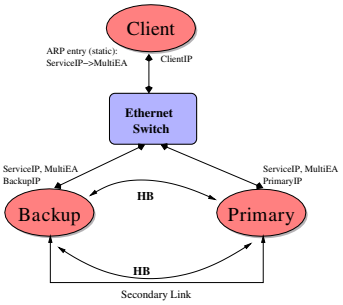

Figure 2: Experimental setup.

Demo 1: Client-Transparent Seamless Failover. The goal of this demonstration is to show that in the event of the primary server failure, ST-TCP provides a client-transparent, fast and seamless failover to the backup. A GUI client-server application is used in this demonstration. The client continually requests and receives data from the server. As the client receives the data, it dynamically updates a pie chart reflecting the percentage of the data already received. While the transfer is in progress, the primary server is crashed so that the TCP connection fails over to the backup. This failover process is seamless to the client as is apparent by observing the progression of the pie chart.

This demonstration also shows that in the absence of ST-TCP, even if a hot backup is available, the failure of the server would lead to a disruption in the service and the client would have to re-connect, unlike in ST-TCP where the failure at worst appears as a glitch to the user.

Demo 2: Dependence of Failover Time on HB Frequency. In this experiment, we examine the dependence of the failover time on the HB frequency. One constituent of the failover time is the failure detection time. The other depends on how much the backup and the client TCP have backed off during the time it took to detect the failure. Recall that TCP backs off exponentially as retransmissions fail; if the primary fails, both the backup and the client (assuming both the server and the client are sending data) would start retransmitting and backing off (in case of backup, since it has not taken over the connection yet, the retransmissions get discarded). Once the failure is detected and the backup takes over, there is still a delay until the next client or backup retransmission before the TCP stream gets re-started. In this demonstration, we try three different values of HB period (200ms, 500ms and 1s) and measure the failover times in each case.

Demo 3: Insignificant Overhead during Normal Operation. In this demonstration, a large file (about 100 MB) is transferred to the client both with ST-TCP enabled and with ST-TCP disabled. We compare the time taken for the file transfer in both of these cases. The aim of this experiment is to show that under normal operation (no failures), the overhead of using ST-TCP is negligible.

Demo 4: Application Crash Failure. The goal of this demonstration is to show that ST-TCP tolerates application failures. The GUI application, used in Demo 1 above, is used here as well. Two different scenarios of application failures are simulated. In the first scenario, the application on the primary crashes but the socket is not closed, and hence a FIN segment is not generated. In the second scenario, the OS cleans up the application and closes the socket, thus, generating a FIN. In both of these scenarios, the application failure is detected and the TCP connection is migrated to the backup.

Demo 5: NIC Failure. This demonstration has two parts. In the first part, we simulate a failure of the NIC at the primary; in the second, that of the backup. In both these cases the HB on the IP link between the servers fails, but the one on the secondary link stays up. The servers use the information in the HB to determine whether the failure has occurred at the primary or at the backup.

6 Conclusions

TCP is the most popular transport-level protocol for constructing distributed applications over the Internet. Current fault tolerance techniques typically require software updates both at the client and the server. This limits the applicability of these techniques. Recently, several research projects have addressed this problem by providing server fault tolerance support at the TCP layer. ST-TCP is one such effort to address this problem.

This paper describes lessons learned from using ST-TCP under different computing environments. In particular, the paper reports on three issues. First, it reports on peculiar behavior of servers under specific computing conditions and discusses design enhancements that ST-TCP has undergone to address them. Second, the paper discusses in detail how ST-TCP addresses different failure scenarios, particularly application failures and local network failures. Finally, the paper describes five experiments that will be demonstrated at the conference.

References

[1] R. R. Koch, S. Hortikar, L. E. Moser, and P. M. Melliar-Smith. Transparent TCP connection failover. In *Proceedings of the IEEE Int. Conf. on Dependable Systems and Networks*, San Francisco, June 2003.

[2] M. Marwah, S. Mishra, and C. Fetzer. TCP server fault tolerance using connection migration to a backup server. In *Proceedings of IEEE Int. Conf. on Dependable Systems and Networks*, San Francisco, June 2003.

[3] M. Orgiyan and C. Fetzer. Tapping TCP streams. In *Proceedings of the IEEE International Symposium on Network Computing and Applications*, February 2002.

[4] D. A. Patterson. A simple way to estimate the cost of downtime. In *Proceedings of LISA '02: Sixteenth Systems Administration Conference*, November 2002.

[5] A. C. Snoeren, D. G. Andersen, and H. Balakrishnan. Fine-grained failover using connection migration. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[6] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory TCP: Connection migration for service continuity over the internet. In *Proceedings of the 22th IEEE International Conference on Distributed Computing Systems*, Vienna, Austria, July 2002.

[7] D. Zagorodnov, K. Marzullo, L. Alvisi, and T. Bressoud. Engineering fault tolerant TCP/IP services using FT-TCP. In *Proceedings of IEEE Int. Conf. on Dependable Systems and Networks*, San Francisco, June 2003.