Appeared in Proc. of IEEE Int. Conf. on Dependable Systems and Networks (DSN 2003), San Francisco, CA, June 22-25, 2003

TCP Server Fault Tolerance Using Connection Migration to a Backup Server

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

Christof Fetzer
AT&T Labs-Research
180 Park Avenue
Florham Park, NJ 07932

Abstract

This paper describes the design, implementation, and performance evaluation of ST-TCP (Server fault-Tolerant TCP), which is an extension of TCP to tolerate TCP server failures. This is done by using an active backup server that keeps track of the state of the TCP connection and takes over the TCP connection whenever the primary fails. This migration of the TCP connection to the backup is completely transparent to the client. Because no changes are required on the client machine, any TCP client can access a ST-TCP server. The performance overhead of ST-TCP over standard TCP is minimal, and during normal operation its behavior is the same as that of a regular TCP. In addition, ST-TCP provides a fast and seamless failover whenever the primary server fails. This is veri£ed by a prototype implementation of ST-TCP in the Linux operating system, and experiments with a number of simulated applications which have different communication characteristics.

1 Introduction

TCP is the most popular transport-level protocol for constructing distributed applications over the Internet. It has been used to construct several commonly used applications and protocols such as FTP, http, telnet, ssh, and sendmail. The main reason for TCP's popularity is its rich set of desirable features, including a reliable, ordered, duplex byte stream, ¤ow control, and congestion control. However, an important feature that TCP does not provide is server fault tolerance. If the machine on which a server is running fails, all TCP connections to the server break and all TCP clients get disconnected from the server. Even if a backup server exists, a new TCP connection has to be established between each client and the backup server and lost packets have to be determined and retransmitted.

With the growing popularity of TCP to construct distributed applications, it is vital that transparent techniques be developed that allow a client to receive uninterrupted and undegraded services despite server failures. Examples of present-day applications that use TCP and require server fault tolerance include live broadcast of events, on-line brokerage £rms, video-on-demand services, and e-commerce. While several large scale systems have been developed to address the issue of high availability in the presence of failures, they are typically too heavy weight and require extensive modi£cation of client programs. A brief overview of these systems and other related work is given in Section 2.

In this paper, we describe the design, implementation, and performance evaluation of ST-TCP (Server fault-Tolerant TCP) that provides support for tolerating the failure of a TCP server. ST-TCP relies on the existence of an active backup TCP server that takes over the TCP connection in case of a primary TCP server failure. This server fault tolerant extension of TCP is completely transparent to the TCP clients, i.e., there is no change required in the client-side TCP and no special wrapper or libraries are needed on the client machines. Standard TCP clients can connect to a ST-TCP server in the same way they connect to a standard TCP server. Also, the clients do not notice any server failure, or service disruptions during a server failover period. Thus, ST-TCP allows standard TCP clients to stay connected and continue to receive the required service despite TCP server failures.

This paper makes several important contributions in the £eld of dependable distributed computing. First, ST-TCP is one of only a handful of approaches designed to deal with server failures at the transport level, and unlike these other approaches ST-TCP is very light weight. There is virtually no performance overhead during failure-free periods, and failover is very fast. Second, unlike most other fault-tolerance approaches, ST-TCP does not require the installation of any extensive infrastructures. ST-TCP only requires minor changes to standard TCP server stack. Finally, ST-TCP is completely transparent to the clients. No changes are needed on the client-side TCP, and no new software

*Also with Avaya Labs, 1300 West 120th Ave., Westminster, CO 80234.

(libraries or wrappers) needs to be installed on the client side. Existing TCP client applications can connect to ST-TCP servers without any modi£cations in their code.

2 Related Work

A number of approaches to provide high availability in the presence of failures have been investigated over the last 20 years. These include group communication systems, primary backup systems, distributed object middle-ware, and transport-level, fault-tolerant systems.

Primary backup systems provide fault-tolerance capabilities by replicating service state on one or more backup servers. Clients interact with the primary server. Backup servers monitor the health of the primary, and in case of a primary server failure, one of the backup server is promoted to act as the new primary server (*failover*). Primary backup techniques have been used to build numerous dependable systems. For example, see [12, 7, 18]. Again, most of these systems require clients to be aware of the protocols being used, and maintain the identity of the server. On a failover, the new primary and the client re-establish the connection. This requires an application level protocol which must be built into the client as well as the server.

There are approaches for achieving fault tolerance aimed at the transport-level protocols such as TCP. These include FT-TCP [1], HydraNet [13], [11], M-TCP [17], and SCTP [16]. ST-TCP presented in this paper belongs to this category.

In FT-TCP[1], server fault tolerance in TCP is provided by using the standard primary backup approach. Two wrappers (SSW and NSW) are put around the TCP server code to forward TCP byte stream to a logger, and ensure that the server state stored at the logger is consistent with the primary server state. In case of a server failure, a new server is started using the state stored in the logger. During this failover period, the client TCP connection is kept alive by sending zero-size window advertisements at regular intervals. ST-TCP improves on FT-TCP by providing a fast failover. The failover time in FT-TCP can be fairly large. This is because a failover in FT-TCP requires failure detection, time for the backup server to start, and time to update the backup server state from all the data saved in the logger (which could be quite large for long running applications). Thus, although the TCP connection does not break here, a client will certainly see a disruption and degradation in service. ST-TCP, on the other hand provides a very fast failover. Note that FT-TCP could in principle be used with active replication and in this way could reduce the fail-over time.

ST-TCP makes different trade-offs compared to FT-TCP. ST-TCP does not only optimize the fail-over time but also the failure-free case. By tapping the packets, the backup does not increase the latency nor does it decrease the bandwidth. However, ST-TCP requires additional hardware (additional logger computers, NICs, and CPUs) in comparison to FT-TCP. ST-TCP is optimized for applications that need high troughput and availability and that are important enough to pay for the extra hardware.

HydraNet-FT [13] is an infrastructure to dynamically replicate services across an internetwork and have replicas provide a single, fault-tolerant service access point to clients. The service access point is facilitated by an IP-redirector that provides one-to-many message delivery to replicas, and many-to-one message delivery from the replicas to the client. ST-TCP is based on a similar idea of redirecting TCP byte stream to backup servers, in addition to the primary server. The main difference between HydraNet-FT and ST-TCP is that the process of redirection is extremely simple (tapping TCP byte stream) in ST-TCP. In fact, there is no separate redirector component in ST-TCP. The main reason for this simplicity is that while ST-TCP is focused on providing server fault-tolerance in TCP, HydraNet-FT aims to provide additional services such as replica management and load balancing. As a result, the design of redirector is complicated.

Another approach for providing fault tolerance at the TCP layer is discussed in [11]. A backup server is used to tap the Ethernet to read all the packets destined for the primary. A layer is added between the TCP and the application layer on the servers as well as the client. When the TCP connection breaks or the primary fails, this layer in the backup server and the client establish a new TCP connection. Although, this makes a connection failure transparent to the client application, it does require modi£cations on the client machine (installing a client wrapper). This defeats our main goal of not requiring any changes on the client end.

A recent work addressing continuity of services implemented using TCP is M-TCP (Migratory TCP) [17]. M-TCP operates in an environment where a service is implemented by a pool of servers. A client establishes a TCP connection with one of the servers. Each server maintains a £ne-grained checkpoint of each TCP connection state. TCP connection migration from its current server S_c to the another server S_n in the server pool depends on the migration policy. For example, it can be initiated by the client when it notices that its service has deteriorated. At this point, S_c transfers the checkpointed state C_p of this connection to S_n, S_n starts a new TCP connection and initializes its state using C_p, and then sends a message to the client to continue its operation.

SCTP [16] provides support for tolerating network congestion and failure by establishing multiple redundant paths between the client and the server. It cannot tolerate server failures. We note that ST-TCP can operate in con-

junction with SCTP. This will result in a more powerful transport-level service that can tolerate both network failures/congestion and server failures.

Several TCP extensions have been proposed to support TCP connections in face of mobility of clients. These include [4, 5, 10, 15]. They typically rely on either a proxy (which is a single point of failure), or maintaining a full connection state at the server/client. In either case, they are not designed to tolerate server failures. Finally, several approaches have been taken to provide continued service of HTTP servers [2, 14].

3 Informal Overview

ST-TCP is based on the idea of primary/backup approach to provide high availability and fault tolerance. To ensure fast failover, ST-TCP maintains active backup servers that can take over the functions of the primary server as soon as any failure of the primary server is detected. To avoid any performance overhead during failure-free periods, ST-TCP requires only a few changes on the server-side TCP that do not affect the normal ¤ow of TCP byte stream.

The main idea in the design of ST-TCP is that one or more backup servers receive the TCP segments exchanged between a client and a primary server by simply tapping the TCP byte stream at an intermediate point in the connection between the client and the server. Figure 1 illustrates this idea. Since every TCP segment exchanged between the client and the primary server is tapped in by the backup server, the backup server learns the complete communication state of the primary server. In particular, we can guarantee that a non-crashed backup server receives all data that the primary server receives. In addition, it also receives all data that the client receives.

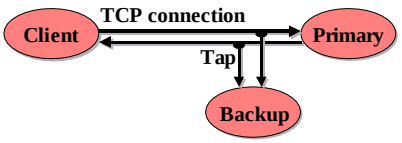


Figure 1. Tapping TCP byte stream.

In case a server is completely deterministic, a backup server can keep its state consistent with the state of the primary server by executing the same sequence of requests as the primary server does. Many servers are however non-deterministic, e.g., timestamps or ids used by the backup server may not be exactly the same as those used by the primary server. Replication of non-deterministic servers has been researched extensively. To keep the primary and backup servers in sync, one can combine this protocol with a *leader/follower* consistency protocol (e.g., [6]). Consistency can be achieved without manually changing the server application code. In what follows, we assume that applications are deterministic or a leader/follower protocol is used to keep the state of the primary and the backup consistent.

A backup server can detect the failure of a primary server when its state becomes inconsistent with, or its (suppressed) replies differ from, that of the primary server. This allows a backup server to *detect* failures of the primary server that are different from crash or performance failures. Crash and performance failures of the primary server are detected by the backup using a simple timeout mechanism. For this reason, the primary server sends periodic heartbeat messages to the backup server. We enforce consistent behavior by making sure that the primary server is never incorrectly suspected to have failed (see Section 3.2).

3.1 Ethernet Tapping

Most local area networks are Ethernet based. Therefore, we assume that servers communicate with clients via a local area Ethernet. The clients are connected to the Ethernet by one or more gateways. This means that all TCP traf£c between the server and the clients can be tapped from the local Ethernet.

Ethernet has originally been a broadcast medium. Broadcast media simplify the tapping of communication streams. [11] discusses different tapping architectures for broadcast Ethernet. However, in recent years most Ethernet installations have been converted to switched Ethernet (see Figure 2). Logically, an Ethernet switch replaces the broadcast medium by a crossbar. This prevents a backup node from tapping the traf£c of the primary node since a node only receives packets addressed to itself.

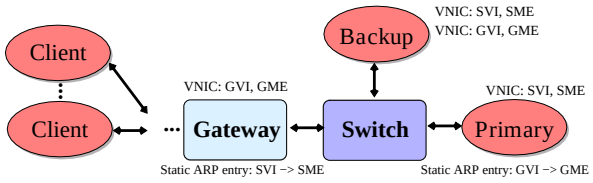


Figure 2. Ethernet switch tapping architecture.

Some managed Ethernet switches provide an option to forward traf£c ¤owing from/to a port to some other port. This feature permits us to sniff all traf£c between the server and the clients. To facilitate tapping in switched Ethernet without this capability, we also investigated a solution that maps a unicast IP address to a multicast Ethernet address to ensure that all packets received or sent by the primary are forwarded by the Ethernet switch to the backup.

The service provided by the primary can be accessed by the clients via a static virtual IP address, SVI. We create

at the backup and the primary node a virtual network interface (VNIC), i.e., a software NIC mapped to a hardware NIC. We assign SVI to these VNICs. Further, we assign a £xed multicast Ethernet address SME to these VNICs on the primary and backup machines. The service IP address SVI is statically mapped onto the multicast address SME in the gateway ARP table. This static mapping is necessary since the IPv4 router requirements RFC [3] disallows an IP router to accept a multicast link layer address in an ARP reply. The mapping of SVI to SME permits the backup node to tap all traf£c from the clients to the server.

To tap the traf£c from the primary server to the clients, we similarly create VNICs in the gateway and backup nodes and assign each with a static virtual IP address (GVI) and a £xed multicast Ethernet address (GME). A static GVI to GME mapping is created in the primary ARP table. The backup node can in this way tap all traf£c from the primary to the client. Note that most gateway machines run modern operating systems like Linux, because they often provide additional services like £rewall and/or VPN functionality. Hence, we can install VNICs on these gateways.

3.2 System Architecture

Dependable systems often have the requirement of having no single point of failure. Avoiding a single point of failure can increase the availability of a system and simpli£es the replacement of components. ST-TCP supports system architecture without a single point of failure like the one depicted in Figure 3. All components are replicated. Primary and backup are connected to two switches that are connected via two loggers to two gateways. One can use the two links to increase the bandwidth. In particular, for full-duplex Ethernet links to the server one would con£gure ST-TCP such that the backup receives the packets to and from the server on two separate Ethernet links.

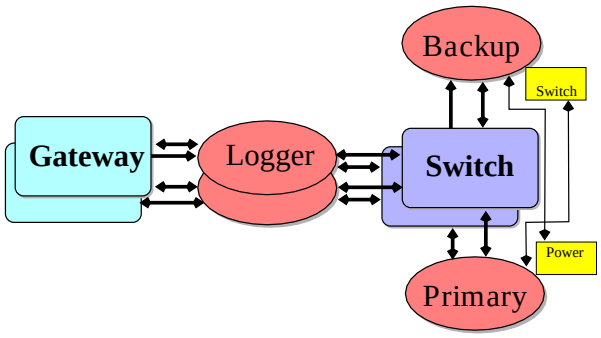


Figure 3. System Architecture without a single point of failure.

For ST-TCP to work correctly, we need a perfect failure detector. In particular, the backup is never permitted to suspect the primary if the primary is still up. To do that, we can use the perfect failure detector protocol of [9] that was designed for this situation. Alternatively, we can use controllable power switches. If the backup suspects the primary, it switches off the power of the primary. This makes sure that the primary is crashed before the backup takes over the IP address of the service.

As we will describe in Section 4, the backup asks the primary for packets that it failed to receive from the Ethernet (but were received by the primary). In case such an omission failure happens together with a crash of the primary, the backup can in certain cases not take over as primary because the complete communication state is not known to the backup. To mask such double failures, one can insert a logger into the network [11]. This logger machine logs all packets on the Ethernet in its main memory for a bounded amount of time. Since the backup will suspect the primary within a bounded amount of time after the primary crashed, the backup can recover all missing packets from the logger.

The logger introduces a very small delay but does not reduce the bandwidth. The memory needed by the logger depends on the maximum bandwidth of the Ethernet and the maximum failover time. Since all traf£c to and from the server has to ¤ow through the logger(s), the logger(s) has (have) the complete communication state. By having two loggers, or a bypass network for a failed logger, one can prevent the logger from becoming a single point of failure.

In addition to avoiding single points of failure, we want to avoid the introduction of performance bottlenecks, i.e., new CPU or bandwidth limitations due to replication. For half-duplex Ethernet one Ethernet NIC has suf£cient bandwidth to tap the traf£c between the primary and its clients. In full-duplex mode the bandwidth of a single Ethernet NIC might not be suf£cient to capture the complete traf£c. In the latter case, one can use one Ethernet NIC to tap the traf£c for each data direction. Note that due to other bandwidth limitations (e.g., WAN bandwidth limitations) one might not always need additional NICs.

Due to the tapping of the Ethernet packets from the primary to the clients, the backup system needs additional CPU resources. Modern operating systems support the parallel processing of network packets. Hence, the addition of CPUs to the backup system can address CPU limitations introduced by the Ethernet tapping. Note that the addition of CPUs can reduce new CPU bottlenecks even if the application itself is only single threaded.

4 Protocol Details

We will describe the protocol details of ST-TCP by focusing on three important design issues: (1) starting the primary and backup servers, (2) operation of the primary and

the backup servers during failure-free periods, and (3) failure detection. In this section, we will focus on the protocol design issues. Implementation details are given in Section 5.

4.1 Initialization

The primary and the backup are con£gured as described in the previous section so that the backup network interface receives all packets destined for the primary. Further, the backup's network interface is setup such that all packets from or to the server are accepted by it and passed to the higher layers in the TCP/IP stack. To ensure that the backup can take over a TCP connection from the primary in case of a primary crash, it needs to shadow the state of that connection, i.e., maintain a consistent state of that TCP connection so that on failover it is indistinguishable from the primary to the client.

The server application is started on both the primary and the backup. Since it is the same application, the same listen port is used on both the machines. To avoid any con¤icts, we assume that the backup is running on a dedicated machine, and not used for any other purpose.

In addition to IP addresses and port numbers, another stateful entity in a TCP connection that the backup must be aware of is the sequence numbers used in that connection. The backup must either use the same sequence numbers as the primary or be able to map its sequence numbers for that connection to those of the primary so that the connection migration works. In ST-TCP, backup uses the same sequence numbers as those used by the primary for a shadowed TCP connection. The steps involved in the initialization of such a TCP connection are described below.

1. To initiate the TCP connection, the client sends a SYN segment to the primary. The backup also receives this SYN segment.

2. In response to the SYN, the TCP layer in both the primary and the backup sends a SYN/ACK to the client. This SYN/ACK is dropped (suppressed) on the backup. The primary's SYN/ACK is received by the client.

3. The client's ACK segment, completing the three way handshake, is used by the backup to modify its own initial sequence number and other variables related to the initial sequence number. After this point, the backup's sequence numbers match those of the primary for this connection.

4.2 Failure-free Period

During failure-free period, all TCP segments exchanged between a client and a server are received by the backup server. The backup server executes as a normal TCP server, except that all replies from the backup server to the client are dropped. ST-TCP has incorporated two additional functionalities – (1) ensure that the state of the backup server is consistent with the state of the primary server, and (2) detect failures of (primary or backup) servers.

For simplicity, let us assume that the server application is deterministic. This implies that the (suppressed) replies of the backup server will be identical to the replies of the primary server, as long as the backup server receives the same sequence of bytes from the client as the sequence of bytes received by the primary. In general, this will be the case since the same sequence of bytes that are received by the primary server are tapped by the backup server. However, it is possible that some TCP segments, received correctly by the primary, get lost before reaching the backup server. For example, this can happen if the IP stack on the backup server drops IP packets because of an IP-buffer over¤ow. Since the primary server receives these segments, it will acknowledge these bytes. This will result in the client-side TCP purging those segments from its send buffer. As a result, there is no way for the backup server to retrieve those lost TCP segments.

We address this problem and the issue of server failure detection by establishing an alternate connection between the primary and the backup server, and modifying the TCP buffer management in the primary server. A separate UDP channel is established between the primary and the backup servers when these servers are started. This channel is used to make sure that the backup server receives all TCP segments from the client that are received by the primary server. The backup server uses this channel to send a request to the primary server asking for missing TCP segments, when it discovers that it has missed some TCP segments. This channel is also used by both the primary and the backup servers to monitor each other by sending periodic heartbeat (HB) messages. Implementation details of how this alternate channel is established are given in Section 5.

During failure-free periods, the primary server needs to make sure that before it discards a received TCP byte from its receive buffer, the backup server has already received it. So, while in standard TCP, a byte received from the client is discarded once it has been read by the application, ST-TCP requires that this byte, in addition, be acknowledged by the backup server before being discarded. The backup server uses the alternate communication channel to acknowledge (to the primary) the sequence numbers of the bytes it has received from the client. The actual strategy for when and how this acknowledgment is sent is described later.

Figure 4 shows the receive buffer of the primary ST-TCP server. For comparison, we have also included the corresponding receive buffer of a standard TCP server. As

shown, the receiver buffer of the primary ST-TCP server maintains an additional pointer (`LastByteAcked`). This pointer is the sequence number of the last byte acknowledged by the backup server to the primary server. `LastByteRead` is the sequence number of the last byte read by the application, `NextByteExpected` is the sequence number of the next byte the TCP server expects to receive, and `LastByteRecd` is the sequence number of the last byte received by the TCP server.

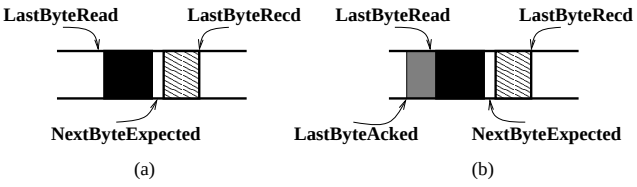


Figure 4. Receive buffer of TCP server: (a) Standard TCP, (b) ST-TCP.

In general, the `LastByteAcked` can be smaller than, equal to, or larger than the `LastByteRead`. If it is smaller than `LastByteRead`, all bytes whose sequence numbers fall between it and `LastByteRead` are the bytes that a standard TCP server would have discarded, but a primary ST-TCP server does not discard. A primary ST-TCP server discards all those bytes whose sequence numbers are smaller than or equal to `LastByteRead` or `LastByteAcked`, whichever is smaller. This strategy ensures that any TCP byte that backup fails to receive from its tapped byte stream can be retrieved from the primary server. In general, we expect that the backup server will be able to receive all the bytes from its tapped TCP stream, and there will not be a need for it to request the bytes from the primary server.

Depending on the acknowledgment strategy that the backup server adopts, it is possible that some bytes will be kept longer in the receive buffer of the primary ST-TCP buffer than a standard TCP buffer. This means that the buffer may £ll up sooner in ST-TCP than in standard TCP. Also, this will reduce the advertised window size that the TCP server sends to client. To compensate for this and to ensure that the ST-TCP behavior remains as close to standard TCP behavior as possible, we double the space allocated for the receive buffer. The Berkeley socket interface allows this on a per connection basis.

With more buffer space available for the receive buffer, the extra buffer space can be managed in a number of ways. We use the simplest approach here. The additional space is used only for storing bytes that have not been acknowledged by the backup server but that have been read by the server application. In other words, ST-TCP logically maintains two receive buffers, each with their own limits. The management of the £rst buffer is identical to the management of the receive buffer in standard TCP, except that some bytes may move to the second buffer before being discarded.

This simple approach ensures that the behavior of the ST-TCP server is identical to the behavior of a standard TCP server from a client's perspective. As long as the backup server keeps sending acknowledgments to the primary server at regular intervals, there will be no difference between the standard TCP server and the ST-TCP server as far as the advertised window size, bytes acknowledged, or any TCP timer calculations are concerned. In other words, during failure-free periods, a client will see no difference between a standard TCP and an ST-TCP.

The behavior of ST-TCP will differ from that of standard TCP if the second buffer £lls up. This can happen when the backup server is too slow in acknowledging the bytes it has received, or if it misses some bytes from its tapped TCP byte stream and requests a transmission of those bytes from the primary server. We address this by instituting an ef£cient acknowledgment strategy in the backup server that is described later in this section.

Clearly, there are other approaches for managing the two buffers that might improve the overall performance of TCP. For example, it is possible to store extra bytes in the second buffer and advertise a larger window size, if the £rst buffer £lls up and there is enough space available in the second buffer. Although these approaches may give better performance results in some situations, they are more complex to implement, and at present, they are not considered.

4.3 Synchronizing The Backup Server

The primary server waits for acknowledgments from the backup server before deleting corresponding bytes from its second receive buffer. The alternate UDP channel between the primary and the backup servers is used for this purpose. The backup server sends acks over this channel containing a sequence number that is one less than its `NextByteExpected` value. The frequency with which these acks are sent has a signi£cant effect on the ST-TCP performance. If these acks are sent very frequently, e.g., after receiving every byte, the primary server may end up spending too much time in processing these acks, thus affecting the net TCP throughput. On the other hand, if these acks are sent only once in a while, the second receive buffer in the primary server will £ll up, again affecting the net TCP throughput.

The backup server in ST-TCP uses a very simple strategy to send these acks. Instead of sending an ack after receiving every byte, the backup server maintains an integer variable, `LastByteAcked`. The value of this variable is the sequence number of the last byte that was acknowledged to the primary server. An ack is sent whenever any of the following events occur (`X` is a con£guration parameter):

- NextByteExpected − LastByteAcked ≥ X, or
- a £xed time interval SyncTime has elapsed since sending the last ack.

The £rst event corresponds to the condition when the backup server has received at least X bytes (in the correct order) from the client since sending the last acknowledgment. The second event corresponds to the condition when less than X bytes (in the correct order) are received from the client in the last SyncTime time units, since sending the last acknowledgment.

The value of X naturally depends on the size of the second receive buffer in the primary server. As a start, we have chosen to £x X as three-fourths the size of the second buffer. For example, if the size of the second buffer if 4 KB, X = 3 KB. The value of SyncTime depends on how frequently the backup server and the primary server need to monitor each other. We use the acks sent by the backup server and its response sent back by the primary (which also serve as heartbeat messages) as a mechanism to monitor the liveness of the primary and the backup servers. We have experimented with different values of SyncTime ranging from 50 milliseconds to 5 seconds.

The extra traf£c introduced by the alternate UDP channel is quite insigni£cant. For example, assume that the total length (including all header overheads down to Ethernet) of an ack packet is 128 bytes, and there is only client traf£c on the LAN (worst case). In this case, one ack packet for every 3 KB of client data increases the LAN traf£c by only 4.17%.

4.4 Failure Detection

We assume that the computers (primary or the backup servers) have crash/performance failure semantics[8]. The failure detection is based on a timeout mechanism. The backup monitors heartbeat (HB) messages from the primary to detect primary's failure and take over its TCP connection.

The primary monitors the HB messages from the backup and the size of its second receive buffer to determine if the backup has failed. On detecting failure of the backup, the primary transitions to non-fault-tolerant mode.

The failure detection mechanism will eventually suspect a crashed computer. However, it might wrongly suspect non-crashed computers. We convert wrong suspicions into correct suspicions by switching off the power of a suspected computer (see also Section 3.2) before propagating the suspicion .

5 Implementation

We have implemented a prototype of ST-TCP. This prototype runs on Linux operating system (kernel 2.2.18), and involved modifying the TCP/IP stack in the kernel. On the backup ST-TCP server, changes were made to synchronize the initial sequence number with that of the primary on TCP connection initialization and to discard TCP segments to the client during failure free periods.

Recall that the primary and the backup servers maintain a UDP channel to help the backup server recover from temporary communication failures of the tapped TCP byte stream, and monitor each other by exchanging Heartbeat messages containing sequence number information. There are at least two ways to establish this UDP channel. One way is to establish this channel in the kernel itself, between the backup and the primary. The advantage of this approach is that sequence number data does not have to be copied from the kernel space to user space. However, it is best to avoid changes in the kernel whenever possible, and although this approach is slightly more ef£cient, it was not followed.

The other approach, which we used, is to implement this channel in a user process outside the kernel. This requires that the user process has access to sequence number data. This is done by using the /proc £lesystem in Linux which allows the user process to open a £le and read this data. In our prototype implementation, when the backup takes over, it sets a ¤ag in the /proc £lesystem to indicate to the kernel that the backup has taken over. As soon as the ¤ag is set, the kernel starts sending the packets to the client instead of dropping them.

6 Performance

To provide a proof of concept, we measured the performance of ST-TCP with simulations of applications representing different communication characteristics. The main focus of these experiments is to show that no changes are required in the client, there is no deviation from standard TCP during failure-free periods, and that the failover from primary to backup is fast.

Three different applications, with differing communication behaviors, are considered. The £rst application (**Echo**) consists of a client sending a small message (about 150 bytes) to the server, and the server responding back with the same message to the client. The client waits to receive the echo response before issuing another request. One run of this application involves 100 such message exchanges. The second application (**Interactive**) consists of a client sending a small request (about 150 bytes) and the server responding with an appropriate reply consisting of moderate size data (10 KB). One run of this application also involves 100 requests and the corresponding responses. A new request is sent only after the response to the previous one is received. The third application (**Bulk transfer**) consists of a client sending a small request (about 150 bytes) to the server, and the server responding with a large data £le. Files sizes of

1 MB, 5 MB, 20 MB and 100 MB are used for this experiment. As an analogy, the communication pattern of **Echo** is similar to the one displayed by `telnet`, the communication pattern of **Interactive** is similar to the one displayed by `http`, and the communication pattern of **Bulk transfer** is similar to the one displayed by `ftp`.

Experimental Setup. We use three machines to run the experiments. The primary and the backup are 800 MHz AMD Athlon PCs, with 512 KB of cache and 256 MB of memory. Both the machines have Linux kernel 2.2.18 installed on them. The client is a 900 MHz Pentium III Laptop also running Linux (although it could be running any OS that provides a TCP/IP stack). All the machines have a 10/100 Mbit network interface card. These three machines are placed on the same LAN using a 10/100 Mbit Ethernet hub. Since the hub broadcasts all traf£c on all ports, the backup can tap into all of the primary's network traf£c. Using an Ethernet switch will lead to a higher throughput.

For each of the three applications, we performed two sets of measurements. The £rst one measures the performance overhead of ST-TCP over standard TCP when there are no failures in the system. As mentioned earlier, an important goal in the design of ST-TCP is to keep this overhead insigni£cant. This measurement is done by separately running the applications, with standard TCP and with ST-TCP. For ST-TCP the applications are run for varying values of the HB interval. The second set of performance measurement measures the time it takes for the backup server to take over the primary server when the primary server crashes. This is also called the *failover* time. As mentioned earlier, an important goal in the design of ST-TCP is to keep this time fairly short, so that a client will barely notice a disruption in service continuity. For all the applications, the failover time is measured for various HB intervals. All measurements taken were repeated at least three times and their average values were used. Further, the TCP timestamp option was disabled on the primary and the backup during these experiments.

6.1 Performance Overhead of ST-TCP

A performance comparison between standard TCP and ST-TCP is shown in Table 1 when there are no failures. For all three applications, there is no signi£cant difference between the average time taken with standard TCP and ST-TCP. Furthermore, for ST-TCP, the average time taken does not differ signi£cantly for different values of the HB. This demonstrates that ST-TCP does not incur any performance overhead over the standard TCP.

6.2 Failover Time in ST-TCP

To measure the failover time, a primary crash failure was introduced while the application was running. The failover time depends on two parameters. First, the time it takes for the backup to detect a failure, which directly depends on HB frequency. In our experiments, the backup concluded that the primary has crashed after missing three consecutive HB from the primary. For example, with an HB every 5 sec, the backup will detect primary crash in 15 to 20 seconds depending on when exactly the failure occurs.

The second parameter determining the failover time is the increase in the value the TCP retransmission timeout (RTO) during the time the backup took to detect the failure. This depends on the RTO when the failure occurred and the RTO backoff algorithm. In Linux, the RTO is computed using the round trip time (RTT) and is increased by a factor of two with every retransmission. The lower and upper bound for the RTO in Linux are 200 ms and 2 min respectively.

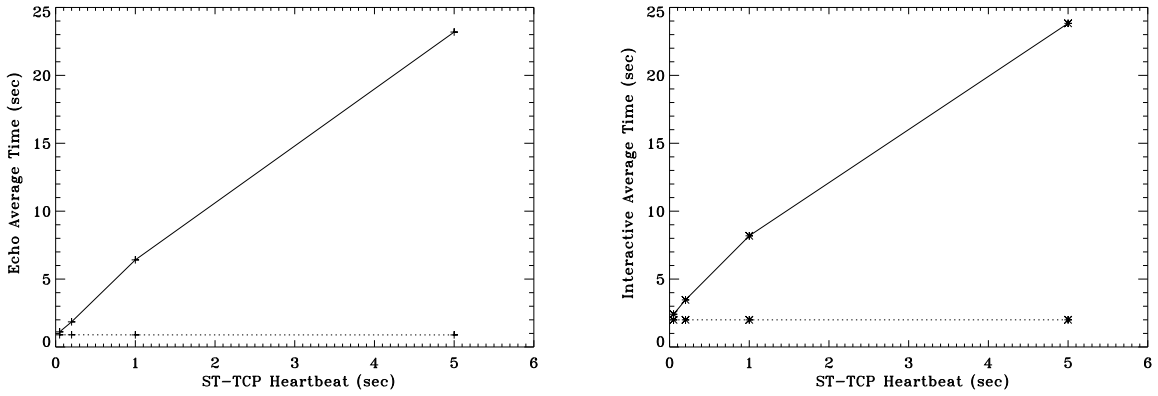
Figures 5.a, 5.b and 6 show the time taken by one run of each application when there is no failure and in the presence of a failure. The main observation here is that the failover time is directly dependent on the HB interval. The graphs show that the total time taken increases as the HB interval increases. This is because it takes longer to detect failure if the HB interval is large. In Figures 5.a and 5.b the upper curve depicts the failure case. The lower curve shows the failure free case. The failover time is the difference in the values of these two curves. Table 2 summarizes the failover time for the three applications. At a high HB frequency (i.e., at a short HB interval), the failover time is only a few hundred milliseconds, which makes it insigni£cant compared to the total time taken by the application in most cases. This is especially true of bulk transfer, Figure 6. Thus, by using an HB interval of about 50 milliseconds, ST-TCP ensures that there is no performance overhead during failure-free periods, and failover is quite short (less than 700 ms). This demonstrates that ST-TCP provides a fast failover.

7 Conclusions

This paper describes the design, implementation, and performance evaluation of ST-TCP, which is an extension of TCP to tolerate TCP server failures. This is done by using an active backup server that taps the TCP bytes exchanged between a client and primary TCP server, maintains a consistent state of the TCP connection, and takes over the TCP connection whenever the primary server fails. This migration of the TCP connection to the backup server is completely transparent to the client. Because no changes are required on the client machine, any TCP client can access any ST-TCP server.

	Average Total Time (in secs) without failure					
	Echo Application	Interactive Application	Bulk Transfer Application			
			1 MB	5 MB	20 MB	100 MB
Standard TCP	0.892	2.000	0.640	3.199	12.788	63.952
ST-TCP 5s HB	0.889	2.000	0.639	3.196	12.791	63.950
ST-TCP 1s HB	0.890	1.998	0.641	3.200	12.824	63.883
ST-TCP 200ms HB	0.898	2.000	0.640	3.203	12.794	63.964
ST-TCP 50ms HB	0.896	1.998	0.658	3.201	12.897	63.883

Table 1. Comparison of standard TCP with ST-TCP during failure free period.



(a) (b)

Figure 5. Performance of (a) Echo (upper line: with failure; lower line: without failure) and (b) Interactive (upper line: with failure; lower line: without failure).

ST-TCP has been implemented in Linux operating system by making a few changes in TCP/IP stack in the kernel. The modi£ed version runs on primary and backup server machines. A performance measurement from this prototype implementation shows that the performance overhead of ST-TCP over standard TCP is insigni£cant when there are no failures. In addition, the failover time exhibited by ST-TCP is quite low, less that 700 milliseconds when the primary and backup servers exchange heartbeat messages every 50 milliseconds.

The key distinguishing features of ST-TCP are complete transparency to the clients, no performance overhead during failure-free periods, and a fast failover that clients are unlikely to notice, especially in long-running applications. The design and prototype implementation of ST-TCP demonstrates that ST-TCP is a better alternative to tolerate TCP server failures than other approaches proposed earlier.

References

[1] L. Alvisi, T. Bressoud, A. El-Khashab, K. Marzullo, and Z. Zagorodnov. Wrapping server-side tcp to mask connection failures. In *Proceedings of Infocom 2001*, April 2001.

[2] M. Aron, P. Druschel, and W. Zwaenepoel. Ef£cient support for P-HTTP in cluster-based web servers. In *Proceedings of USENIX'99*, 1999.

[3] F. Baker. Requirements for IP version 4 routers. Request For Comments 1812, Internet Engineering Task Force, 1995.

[4] A. Bakre and B. R. Badrinath. Handoff and system support for indirect TCP/IP. In *Proceedings of the 2nd Usenix Symposium on Mobile and Location-Dependent Computing*, April 1995.

	Failover Time (in secs)					
	Echo Application	Interactive Application	Bulk Transfer Application			
			1 MB	5 MB	20 MB	100 MB
ST-TCP 5s HB	22.309	23.832	22.580	24.013	20.805	21.764
ST-TCP 1s HB	5.524	6.187	5.723	4.667	2.067	5.022
ST-TCP 200ms HB	0.953	1.468	1.245	1.255	1.036	0.485
ST-TCP 50ms HB	0.219	0.412	0.417	0.627	0.676	0.422

Table 2. ST-TCP failover time for the three applications.

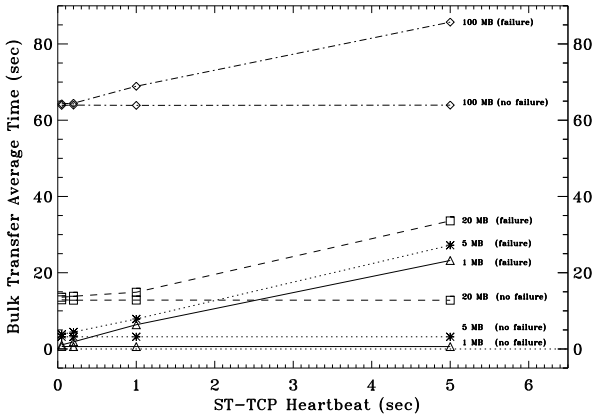


Figure 6. Bulk Transfer Application run with ST-TCP. Graph shows the total time taken with a failover and without failure (lower curve with same line type) during a run for data transfers of sizes 1 MB, 5MB, 20MB and 100MB.

[5] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the 1st ACM International Conference on Mobile Computing and Networking*, November 1995.

[6] P. Barret, A. Hilborne, P. Bond, P. V. D. Seaton, L. Rodrigues, and N. Speirs. The delta-4 extra performance architecture (xpa). In *In Proc. of 20th IEEE Symp. on Fault-Tolerant Systems (FTCS-20)*, pages 481–488, 1990.

[7] N. Budhiraja and K. Marzullo. Highly-available services using the primary-backup approach. In *Proceedings of the 2nd Workshop on Management of Replicated Data*, Monterey, CA, 1992.

[8] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of ACM*, 34(2):56–78, Feb 1991.

[9] C. Fetzer. Enforcing perfect failure detection. *IEEE Transactions of Computers*, Feb. 2003.

[10] D. A. Maltz and P. Bhagwat. MSOCKS: An architecture for transport layer mobility. In *Proceedings of INFOCOMM'98*, March 1998.

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

[12] B. Randell, P. Lee, and P. Treleaven. Reliability issues in computing system design. *ACM Computing Surveys*, 10(2):123–166, Jun 1978.

[13] G. Shenoy, S. Satapati, and R. Bettati. HYDRANET-FT: Network support for dependable services. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, Taipei, Taiwan, April 2000.

[14] 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.

[15] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking*, August 2000.

[16] R. Stewart and et. al. Stream control transport protocol. Request For Comments 2960, Internet Engineering Task Force, 2000.

[17] 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.

[18] H. Zou and F. Jahanian. Real-time primary-backup (RTPB) replication with temporal consistency guarantees. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, 1998.