Migratory TCP: Connection Migration for Service Continuity in the Internet [*]

Florin Sultan, Kiran Srinivasan, Deepa Iyer, and Liviu Iftode [†]
Department of Computer Science
Rutgers University, Piscataway, NJ 08854-8019, U. S. A.
{sultan, kiran, iyer, iftode}@cs.rutgers.edu

1 Motivation

Today's Internet services are commonly built over TCP [5], the standard Internet connection-oriented reliable transport protocol. The endpoint naming scheme of TCP, based on network layer (IP) addresses, creates an implicit *binding* between a service and the IP address of a server providing it, throughout the lifetime of a client connection. This makes a TCP client prone to all adverse conditions that may affect the server endpoint or the internetwork in between, *after* the connection is established: congestion or failure in the network, server overloaded, failed or under DoS attack. Studies that quantify the effects of network stability and route availability [4, 2] demonstrate that connectivity failures can significantly impact Internet services. As a result, although highly available *servers* can be deployed, sustaining continuous service remains a problem.

Service continuity can be defined as the uninterrupted delivery of a service, from an end user's perspective. The TCP's ability to support it is limited by its error recovery scheme based on retransmissions to the same server endpoint of the connection (bound to a specific IP address). In practice, the end user might be more interested in receiving continuous service rather than statically binding to a given server. As server identity becomes less important than the service, it is desirable for a client to switch servers during a service session, e.g., if a server cannot sustain the service.

We propose the *cooperative service model*, in which a pool of similar servers, possibly geographically distributed across the Internet, *cooperate* in sustaining a service by migration of client connections within the pool. The control traffic between servers, needed to support migrated connections, can be carried either over the Internet or over a private network. From client's viewpoint, at any point during the lifetime of its service session, the remote endpoint of its connection may transparently migrate between servers.

[*]This work is supported in part by the National Science Foundation under the ITR Grant Number ANI-0121416.

[†]Current address: Department of Computer Science, University of Maryland, College Park, MD 20742.

2 Migratory TCP (M-TCP)

To enable the cooperative service model for service continuity we have designed Migratory TCP (M-TCP) [10, 8], a reliable connection-oriented transport layer protocol that supports efficient migration of live connections. The protocol enables stateful servers to seamlessly resume service on migrated connections by transferring an application-controlled amount of specific state. Although fine-grained connection migration solutions have been proposed before by exploiting features of application-level protocols like HTTP [7, 11], to our best knowledge M-TCP is the first solution that provides generic migration support through a TCP-compatible transport protocol.

The M-TCP design assumes that the state of the server application can be logically split among connections by defining *fine-grained* state associated with each connection.

The M-TCP service interface can be best described as a *contract* between the server application and the transport protocol. According to this contract, the application must execute the following actions: *(i) export* a state snapshot at the old server, when it is consistent with data sent/received on the connection; *(ii) import* the last state snapshot at the new server after migration, to resume service to client. In exchange, the protocol: *(i) transfers* the per-connection state to the new server and *(ii) synchronizes* the per-connection application state with the protocol state.

The migration mechanism of M-TCP (Fig. 1) ensures that the new server resumes service while preserving the exactly-once delivery semantics across migration, without freezing or otherwise disrupting the traffic on the connection. The client application does not need to change.

A client contacts the service through a connection C_{id} to a preferred server S_1. At connection setup, S_1 supplies the addresses of its cooperating servers, along with migration certificates. The client-side M-TCP initiates migration of C_{id} by opening a new connection to an alternate server S_2, sending the migration certificate in a special option. (Fig. 1 (a)). To reincarnate C_{id} at S_2, M-TCP transfers associated state (protocol state and the last snapshot) from S_1.

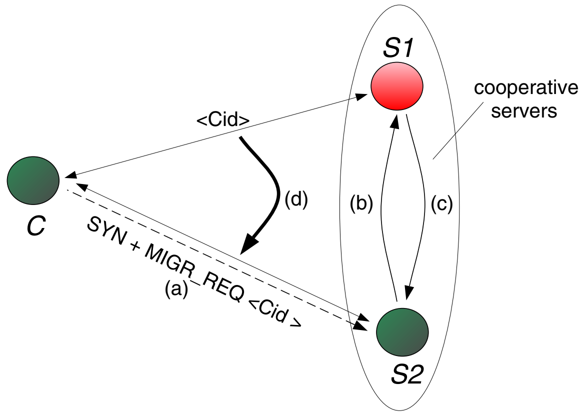

Figure 1. *Migration mechanism in M-TCP. Connection C_{id}, initially established by client C with server S_1, migrates to alternate server S_2.*

Depending on the implementation, the state transfer can be either *(i)* lazy (on-demand), i.e., it occurs at the time migration is initiated, or *(ii)* eager, i.e., it occurs in anticipation of migration, e.g., when a new snapshot is taken. Fig. 1 shows the lazy transfer version: S_2 sends a request (b) to S_1 and receives the state (c). If the migrating endpoint is reinstated successfully at S_2, then C and S_2 complete the handshake, which ends the migration (d).

Upon accepting the migrated connection, the server application at S_2 imports the state snapshot. It then resumes service using the snapshot as a restart point, and performs execution replay for a *log-based recovery* [3] supported by the protocol. The execution replay restores the state of the service at the new server and synchronizes it with the protocol state. To support the replay, M-TCP *logs* and transfers from S_1 data received and acknowledged since the last snapshot. It also transfers unacknowledged data sent before the last snapshot, for retransmission from S_2.

3 Implementation and Applications

We have implemented M-TCP in FreeBSD as an extension to the TCP/IP stack, compatible and inter-operable with the standard TCP [8]. M-TCP is decoupled from and can work with various migration policies [10].

We identify two classes of services that can benefit from M-TCP: *(i)* Applications that use long-lived connections, e.g., multimedia streaming services, applications in the Internet core [6], etc.; *(ii)* Critical applications from which end users expect both correctness and good response time, e.g., Internet banking, e-commerce, etc.

To demonstrate the potential of M-TCP in providing service continuity, we have implemented and evaluated two applications. The first one is a synthetic (generic) media streaming server, for which we use M-TCP in conjunction with a migration policy based on estimated inbound data rate. Migration is triggered when the throughput perceived on the client side falls under a fraction of the maximum observed. We show that, for the same profile of performance degradation at a server, M-TCP can sustain effective throughput close to the average server profile by migrating the connection between servers.

The second application is remote access over the Internet to a transactional database server. We have augmented a PostgreSQL [1] database back-end with support for migratory front-end contexts and used M-TCP between clients and front-end hosts. The resulting system allows a client to start a sequence of transactions with one front-end, then migrate and continue the execution on other front-ends if necessary. The system ensures that ACID semantics are preserved and that the execution is deterministic across migration. The design and implementation are described in detail in the extended version of the paper [9].

We plan to develop application-specific migration policies in the future. A software distribution of M-TCP will be available soon. More details about M-TCP can be found at our site: *http://discolab.rutgers.edu/mtcp*.

References

[1] PostgreSQL. http://www.postgresql.org.
[2] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-end WAN Service Availability. In *Proc. 3rd USENIX Symp. on Internet Technologies and Systems (USITS)*, Mar. 2001.
[3] E. N. Elnozahy et al. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. Technical Report CMU-CS-99-148, Carnegie Mellon University, June 1999.
[4] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Backbone Failures. In *Proc. 29th Symp. on Fault-Tolerant Computing (FTCS)*, June 1999.
[5] J. Postel. RFC 793: Transmission Control Protocol, Sept. 1981.
[6] Y. Rekhter and T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4), Mar. 1995.
[7] A. C. Snoeren, D. G. Andersen, and H. Balakrishnan. Fine-Grained Failover Using Connection Migration. In *Proc. 3rd USENIX Symp. on Internet Technologies and Systems (USITS)*, Mar. 2001.
[8] K. Srinivasan. M-TCP: Transport Layer Support for Highly Available Network Services. Technical Report DCS-TR-459, Rutgers University, Oct. 2001.
[9] F. Sultan et al. Migratory TCP: Highly Available Internet Services Using Connection Migration. Technical Report DCS-TR-462, Rutgers University, Dec. 2001.
[10] F. Sultan, K. Srinivasan, and L. Iftode. Transport Layer Support for Highly-Available Network Services. In *Proc. HotOS-VIII*, May 2001. Extended version: Technical Report DCS-TR-429, Rutgers University.
[11] C. Yang and M. Luo. Realizing Fault Resilience in Web-Server Cluster. In *Proc. SuperComputing 2000*, Nov. 2000.