

Position Summary: Transport Layer Support for Highly-Available Network Services

Florin Sultan, Kiran Srinivasan, Liviu Iftode
Department of Computer Science
Rutgers University, Piscataway, NJ 08854-8019
{sultan, kiran, iftode}@cs.rutgers.edu

We advocate a transport layer protocol for highly-available network services by means of transparent migration of the server endpoint of a live connection between cooperating servers that provide the same service. The current connection-oriented transport layer protocol of the Internet (TCP) reacts to what it perceives as lost or delayed segments only by retransmitting to the same remote endpoint of the connection. TCP provides no means to alleviate a performance degradation (low throughput, many retransmissions etc.) caused by adverse factors like server overload or failure, or network congestion on a given path. At the same time, TCP creates an implicit association between the server contacted by a client and the service it provides. This is overly constraining for today's Internet service models, where the end user of a service is concerned more with the quality of the service rather than with the exact identity of the server.

We propose a transport protocol that (i) offers a better alternative than the simple retransmission to the same server, which may be suffering from overload or a DoS attack, may be down, or may not be easily reachable due to congestion, and (ii) decouples a given service from the unique/fixed identity of its provider. Our protocol can be viewed as an extension to the existing TCP, and compatible with it. To start a service session, the client establishes a TCP connection with a preferred server, which supplies the addresses of its cooperating servers, along with authentication information. At any point during the lifetime of the session, the server endpoint of the connection may migrate between the cooperating servers, transparent to the client application. The current and the new server hosts must *cooperate* by transferring supporting state in order to accommodate the migrating connection.

We assume that the state of the server application can be logically split among the connections being serviced, so that there exists a well-defined state associated with each service session. Transfer of this state ensures that a new server can resume service to the client in the presence of other concurrent service sessions. In addition to the associ-

ated application-level state, transfer of in-kernel TCP connection state reconciles the TCP layer of the new server with that of the client.

Our proposed solution provides a minimal interface to the OS for exporting/importing a per-connection application *state snapshot* by a server. The origin server executes the export operation in order to (i) define an execution restart point for the stateful service on the connection in case of its migration, and (ii) synchronize the service state (reached as a sequence of reads/writes on the connection) with the in-kernel TCP state. The new server executes the import operation to reinstate the connection at the restart point, and resumes service on it, transferring data without altering the TCP exactly-once semantics.

We intend to integrate this mechanism in a general migration architecture in which the *client* side TCP initiates connection migration, in response to various *triggers* that can reside either at the client or at the server(s). Triggers are events like a degradation in perceived traffic quality (on the client side), failure, DoS attack, a load balancing decision etc. (on the server side).

The features of our protocol are: (i) It is general and flexible, in that it does not rely on knowledge about a given server application or application-level protocol. (ii) It allows fine-grained migration of live individual connections, unlike a heavy-weight process migration scheme. (iii) It is symmetric with respect to and decoupled from any migration policy.

We have implemented an operational prototype of our protocol in FreeBSD. We are currently building several applications that can take advantage of the protocol, including a transactional database application with migration support.

Issues that we plan to address in the future are: explore and evaluate various migration trigger policies, evaluate the two options for connection state transfer (eager vs. lazy), develop support to implement fine-grained fault tolerance, and explore the performance tradeoffs of our scheme. More details can be found at our site: <http://discolab.rutgers.edu/projects/mtcp.htm>.