DR-TCP: Downloadable and Reconfigurable TCP

Jae-Hyun Hwang, Jin-Hee Choi, Se-Won Kim, Chuck Yoo
Department of Computer Science and Engineering, Korea University
{jhhwang, jhchoi, swkim, hxy}@os.korea.ac.kr

Abstract

DR-TCP is a new TCP implementation structure that focuses on reconfigurability and extensibility. To make TCP reconfigurable, we re-implement Reno based on a state machine model so that reconfiguration can be done by just modifying state transition table. As a result, DR-TCP can support dynamic reconfiguration without loss of connectivity and its overhead is very low. It can also support binary-level protocol upgrade for extensibility by downloading a new TCP variant which the system does not have. This scheme is more suitable for mobile hand-held devices than existing source-level solution since not requiring compilation environment. To demonstrate the effectiveness, dynamic reconfiguration is performed over Internet, which successfully converts TCP Reno to Westwood at runtime.

1. Introduction

Advances in communication technology allow a variety of new network environments and services available very rapidly. Appearance of various network environments tends to enable a user with a mobile terminal to access among different network simultaneously. However, since new network environment affects performance of communication protocols, terminal systems should provide adaptation schemes for the protocols in order to keep the protocol performance high. A possible solution is to make the protocol reconfigurable to be adapted to current network environment. Unfortunately, most existing network systems are monolithic implementation impossible to support reconfiguration.

The purpose of our research is to propose a new TCP implementation model and show how TCP can be partially reconfigured based on the model. We name the implementation DR-TCP, which is the abbreviation for Downloadable and Reconfigurable TCP. A main usefulness of DR-TCP is its dynamic nature in which TCP functionalities can be replaced and/or extended without stopping TCP operation. In other words, DR-TCP is operational while reconfiguration (or an upgrade) is in progress, which can make the kernel non-stopping. Another usefulness of DR-TCP is for devices where a compiling environment (for upgrade) is not expected. One example is mobile phone, and compiling is not something that a mobile phone is supposed to do.

2. Structure of DR-TCP

It is well-known that the finite state machine can specify theoretically how TCP on one machine interacts with TCP on another [1]. Based on the state machine model, we re-structure TCP Reno so that each function unit is encapsulated by a state machine (e.g. Closed, Listen, Established, etc). A key advantage of applying state machine model to the functions of TCP is that the inter-dependency among TCP functions can be simplified. In other words, most functions share many data structures, which make dynamic reconfiguration extremely difficult. By using state machine, each function of TCP is clearly isolated from the other functions so that interface between states is clear and consistent.

2.1. Making TCP Reconfigurable

DR-TCP consists of one framework and several state machines. The framework contains state transition table, TCP global data such as TCB (Transmission Control Block), and active state indicator. State transition table is a hash table that has a pair of current state, event as a key and next state as a value. Active state indicator keeps track of the pointer of current state machine. Each state machine has an uniform execution interface, *start()*, and it performs incoming or outgoing packet processing. To help understanding of DR-TCP processing, let us consider three-way handshaking for example: when an user application tries to open a session actively, the framework invokes first *start()* function of Closed state machine that is pointed by active state indicator. Closed state machine, then, creates and sends a SYN packet generating SYN_SENT event. At the end of the function, current state is transited to Syn_sent by the transition table.

Since TCP functions are represented several independent state machines, partial reconfiguration can be done easily at runtime. The key idea is to modify state transition table. That is, if there is new ‘Established’ state for a TCP variant, all next state pointers that indicate old Established state in transition table should be change to point new one in order to reconfigure Reno into the new variant. This simple method leads our implementation to support dynamic reconfiguration without loss of connectivity. Note that reconfiguration can be performed with replacing only a few state machines, not whole states since most TCP variants are different from Reno in terms of a few specific functions such as congestion control. Therefore, its reconfiguration overhead is very low.

To show the effectiveness of the proposed scheme, we observed reconfiguration process from Reno to Westwood [3], one of TCP variants, over Internet. State machines that we have to reconfigure are only three: Established, Retransmit, and Fast-retransmit. In Westwood operation, Established state adds bandwidth estimation function to the basic facility, and Retransmit and Fast-retransmit states replace the original *ssthresh* calculation mechanism with new one based on the bandwidth estimation. Through this experiment, we confirm that DR-TCP is reconfigured into Westwood in runtime tracing *cwnd* size and *ssthresh* value, and the reconfiguration processing time is about 5ms.

2.3. Making TCP Extensible

Making TCP extensible is another main goal of DR-TCP. We consider TCP extensibility by downloading new TCP variants which an end-host does not have for new network environment. For the TCP update solution using mobile code, P. Patel et al. are already proposed Self-spreading Transport Protocol (STP), which is a protocol upgrade framework that downloads the entire protocol source code [2]. However, STP only supports source-level update requiring code compilation. This method would not be practical to hand-held devices since they usually do not have development environment for compiling the protocol source code. On the other hand, DR-TCP supports binary-level protocol downloading update, so it can reconfigure new protocol code immediately after downloading and loading the code into memory neither requiring compiling the code, restarting the application nor rebooting the system. This approach can also reduce downloading cost since the downloading code is much smaller than STP.

To actually support binary-level reconfiguration (or update) with the downloaded code (i.e. state machines), two system level supports are needed. One thing is a memory allocator to keep track of the memory address of state machines. Our allocator simply use block header algorithm, and, to identify the object type of allocated area, we only add a simple field to the formal block header. Using this identifier, DR-TCP is able to find the proper transition table of its framework. The other thing is reconfiguration loader that dynamically rearranges the symbols of the binary code. For instance, let us consider the case that the downloaded state machine references DR-TCP's *tcpsend()* function in the example of TCP Westwood. The reconfiguration loader loads the state machines into suitable memory area, and it rearranges the unresolved references. Based on proper equations, call parameter of functions and symbol address of variables can be rearranged.

Compared with STP, the total upgrade time of DR-TCP is significantly short. Actually, the compiling time of STP is about 87% of total upgrade overhead while DR-TCP does not have the overhead for compilation. This means that DR-TCP has two advantages: 1) reduce upgrade overhead significantly and 2) require no compiler (i.e. our solution is suitable to be deployed in mobile devices).

3. Conclusion

In this paper, we re-structure TCP Reno to enable reconfiguration of its partial functions at runtime. DR-TCP can also easily download each function of TCP, and the reconfiguration loader enables to dynamically update the functions. Through the experiment of reconfiguring TCP Reno into Westwood, we confirm that DR-TCP not only dramatically reduces the size of downloadable protocol functions but also makes protocol reconfiguration extremely simple.

DR-TCP is a result of our effort to make reconfigurable protocol stack. Perceiving that almost every protocol can be modeled as state machine, we apply the model to TCP implementation. Currently, DR-TCP is implemented in user-level using raw socket interface for the sake of convenience of testing and debugging[1], and porting to Linux kernel 2.6 is in progress.

Reference

[1] D. E. Comer, D. L. Stevens, "Internetworking with TCP/IP Vol II: Design, Implementation, and Internals – Third Edition," Prentice-Hall, Inc., 1999.

[2] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack, "Upgrading Transport Protocols using Untrusted Mobile Code," In Proceedings of the 19th ACM SOSP, Oct. 1993.

[3] R. Wang, M. Valla, M. Y. Sanadidi, and M. Gerla, "Adaptive Bandwidth Share Estimation in TCP Westwood," In Proceedings of IEEE Globecom, Nov. 2002.

[1] The user-level source code of DR-TCP is available from http://os.korea.ac.kr/network/dr-tcp.tar.gz.