Towards a Composable Transport Protocol: TCP without Congestion Control

Roland Kempter
Department Electrical and Computer Engineering
University of Utah
kempter@eng.utah.edu

Bin Xin
School of Computing
University of Utah
xinb@cs.utah.edu

Sneha Kumar Kasera
School of Computing
University of Utah
kasera@cs.utah.edu

1. INTRODUCTION

The reliable transport layer protocol of the Internet, TCP, provides a variety of features including error control, congestion control, flow control, and in-order delivery. Given the diversity of applications and the underlying network, some of these features are not required for certain applications and networks. In fact, their presence could result in reduced performance. We believe that these features should be offered as composable units rather than all tied together. Composability is also likely to help in adding new features or for upgrading. The eventual goal of our research is to design a fully composable TCP. In this poster, we focus on separating out the congestion control feature such that it could be added and removed without affecting the other features.

We first ask ourselves, where could we use a TCP without congestion control. TCP without congestion control could be useful in scenarios where packet losses are due to reasons other than network congestion, and where fair access to the network resources is achieved through other means. Wireless networks are good examples. Wireless networks have two interesting characteristics. First, they experience a fair amount of loss due to random channel errors[1]. Second, the fairness in accessing the wireless link is handled at the link layer. In these scenarios, TCP congestion control could become a burden.

2. EXPERIMENTATION

While the majority of efforts towards a composable protocol have been using specialized frameworks or APIs (e.g., [Kohler 99]), we investigate the feasibility of constructing a composable TCP based on the widely deployed, highly optimized TCP New Reno in FreeBSD 4.7. TCP New Reno in FreeBSD 4.7 has around 10K lines of code (LOC) with comments. The congestion control code is spread across several files and intermingled with the code implementing other features. We separate out the congestion control code using C preprocessors. In our modified version of TCP New Reno, congestion control amounts to about 250 LOC. We also investigate the use of the TCP Selective Acknowledgment (SACK) feature, that allows a TCP receiver to acknowledge out of order segments selectively rather than cumulatively acknowledging the last correctly received in-order segment (in the case of earlier version of TCP including TCP New Reno). TCP SACK amounts to about 1K LOC and is already available as a composable feature. We compose four different flavors of TCP: TCP New Reno without SACK (BASE), TCP New Reno with SACK (SACKBASE), our modified experimental TCP with neither congestion control nor SACK (EXP), and TCP with SACK but without congestion control (SACKEXP). Without congestion control, TCP send rate is determined by the receiver window and the bandwidth-delay product.

Instead of running synthetic simulations, we choose to follow the more realistic **emulation** approach to measure the performance of these four variants of TCP. We set up the test network, comprising four hosts, two routers and a single shared link, in Emulab [Emu]. The FreeBSD kernels with these variants of TCP are compiled and data for each variant is collected using `ttcp` and `tcpdump`. The data are then analyzed off-line using Ethereal [Eth]. We evaluate the four TCP variants by measuring goodput and efficiency, which is defined as the ratio of goodput to throughput.

Our results show that by simply excluding the congestion control feature from TCP New Reno, EXP improves goodput significantly over BASE up to 280%, but drastically reduces link efficiency down to 13%. This is because without the congestion control feature, the sender window is bigger resulting in higher goodput but at the same time the *Go-back-N* nature of the protocol wastes bandwidth. However, with SACK, we find that the **goodput** of SACKEXP over SACKBASE lies anywhere from **184%** (link loss rate p=0.01) to **730%** (link loss rate p=0.2), while **efficiencies** of SACKEXP over SACKBASE are between 60% and 95%. This interesting result shows that removing congestion control improves goodput but adding SACK also improves efficiency. In fact, we also observe that the goodput improvement of SACKEXP over SACKBASE can be much higher than that of EXP over BASE. We suggest using SACKEXP in the wireless scenario described above, although we need to examine this more.

In future, we will investigate the composability of other TCP features. Our initial efforts indicate that separating the error control feature is likely to be much harder (also corroborated by [Kohler 02]).

A full version of this poster is available at `http://www.cs.utah.edu/~xinb/poster-final.pdf`.

[1]It is possible to locally repair link errors at the link layer and there are several existing studies examining this issue. Here, we only point out that TCP is still necessary for end-to-end reliability.