Enhancing TCP for Networks with Guaranteed Bandwidth Services*

Yujie Zhu*, Oyebamiji Oladeji*, Kyu-han Kim†, and Raghupathy Sivakumar*
*School of Electrical and Computer Engineering
Georgia Institute of Technology
Email: siva@ece.gatech.edu
†Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor
Email: kyuhkim@eecs.umich.edu

***Abstract*— In this paper we consider TCP based applications that use bandwidth guarantees, but can also benefit from any additional best-effort service offered by the network. We show that default TCP cannot offer such applications the aggregate throughput offered by the two services. We propose an adaptation of TCP called GTCP that uses changes to TCP's congestion control mechanisms to provide the optimal aggregate throughput. GTCP does not require any additional implementation overheads, and does not change the TCP receiver. Through simulations we show that GTCP achieves significantly better performance than default TCP in the target environment.**

I. Introduction

The Internet is increasingly being seen as an infrastructure that will lead to *digital convergence*, supporting a diverse set of applications including streaming media, video conferencing, digital telephony, and plain data transfer. This has in turn resulted in the focus on providing applications with quality of service (QoS) assurances that cannot be provided in the current best-effort service model of the Internet. The integrated services (*intserv*) and differentiated services (*diffserv*) architectures are examples of approaches that can provide QoS assurances to applications.

In this paper, we consider applications that subscribe to bandwidth reservations in networks that are QoS enabled through either of the above-mentioned architectures and still require the reliable sequenced delivery services of the ubiquitous TCP transport protocol. We contend that such applications, while utilizing their reserved bandwidths, can further benefit from any best-effort bandwidth provided by the network[1]. In this context, we consider the following question: *How can the aggregate of the reserved and best effort bandwidths be delivered to the application effectively?*

To address this question, we profile the performance of the default TCP protocol over networks that provide both guaranteed and best-effort bandwidth services, and identify the key reasons for its non-optimal performance. Based on the analysis, we propose an enhanced version of the TCP[2] transport layer protocol called *GTCP*. GTCP successfully delivers to the application the aggregated data rates offered by the guaranteed and best-effort services. It adapts the slow-start, congestion avoidance, congestion control, and timeout reaction mechanisms in TCP to achieve the improved performance.

Through a comprehensive set of simulations, we evaluate the performance of GTCP and show that it achieves the desired performance over a variety of scenarios, and scales well across a variety of factors.

The rest of the paper is organized as follows: Section II discusses related work, identifies the key drawbacks of TCP and motivates the design of GTCP. Section III presents the design of GTCP. Section IV evaluates the GTCP protocol and Section V discusses some implementation issues and concludes the paper.

II. Background and Motivation

A. Network and Service Model

The proposed protocol is relevant to any architecture that supports bandwidth reservations. For the sake of clarity, we assume the guaranteed service model supported by the intserv architecture ([2]) in the rest of the paper. We make the following additional assumptions: (1) the marking of conforming traffic is performed at the access router that the source is attached to; (2) bandwidth reservations have been made out-of-band through a reservation protocol such as RSVP [3]; and (3) the specifications for the reserved bandwidth is conveyed by the application to GTCP through a special socket option.

B. TCP over Networks with Guaranteed Bandwidth

In this section we identify the key elements in TCP's congestion control mechanisms that contribute to its non-optimal performance when operating over networks with bandwidth guarantees. We classify these elements under the following two categories:

(1)*Congestion Window:* TCP uses a window based congestion control algorithm, where the congestion window *cwnd*[3] represents the instantaneous rate enjoyed by a connection normalized to its round-trip time. In other words,

$$cwnd = rate * rtt$$

where *rate* is the instantaneous available rate and *rtt* is the round-trip time. In an ideal setting, for a connection with a

*This work was supported in part by funds from the National Science Foundation(awards ANI-0117840, ECS 0225497, and CCR0313005), yamacraw(http://www.yamacraw.org), and the Georgia Tech Broadband Institute

[1]Note that providing such applications with their best-effort share of bandwidth is also fair if we assume realistically that the basic best-effort service level agreement with the network service provider exists.

[2]We assume TCP NewReno in this paper.

[3]Without loss of generality, we assume that cwnd is measured in packets, rate is measured in packets/second, and rtt is measured in seconds.

reserved bandwidth of $rate_G$ and a best-effort rate of $rate_{BE}$, TCP's *cwnd* should converge to

$$cwnd = (rate_G + rate_{BE}) * rtt$$

However, TCP is designed for an environment where all flows are best-effort flows without bandwidth assurances.

Default TCP will adapt its congestion window unaware of the available reserved bandwidth. For example, if the current *cwnd* is 100, of which the reserved bandwidth component is 50 and the best effort component if 50, and congestion is detected through receipt of DUPACKS, TCP will reset its congestion window to 50. However, an ideal congestion window adaptation should not change the reserved bandwidth component and should halve only the best-effort component, resulting in a congestion window of 75. Similarly, upon the expiry of the retransmission timer, instead of going down to a congestion window size of 1, the ideal congestion window adaptation should result in a window size of

$$cwnd = 1 + rate_G * rtt$$

Finally, TCP uses a constant slow-start threshold (*ssthresh*) during its initial slow-start period, and uses half of its current *cwnd* as the new *ssthresh* value upon loss detection. Ideally, TCP's ssthresh values should be set based on the reserved bandwidth for the connection.

(2)*Self Clocking:* Self-clocking is a technique used by TCP wherein the receipt of an ACK for a segment triggers the transmission of a new segment, and appropriately expands the congestion window. The use of such self-clocking reduces the timer overheads incurred by TCP, and ensures that TCP's rate of transmitting segments is highly conservative (unlike say in a rate based scheme). The ideal behavior for a connection with bandwidth reservations should transmit at least $rate_G * rtt$ worth of segments *every round-trip time*. TCP's default self-clocking is automatically maintained even for a connection with bandwidth guarantees as long as there are no losses. However, when losses occur, the left edge of the TCP *cwnd* cannot be shifted pending the arrival of the ACK for that first unacknowledged sequence number[4].

In section IV, we will substantiate the above discussions by evaluating default TCP's performance in an environment where connections have reserved bandwidths.

We use the insights drawn in this section in the design of GTCP that we present in Section III.

C. Related Work

Several related works have been proposed to improve TCP's performance in networks that provide bandwidth guarantees.[1],[4]

In [1], one of the main changes proposed to improve TCP's performance in a guaranteed bandwidth environment is to use delayed and timed transmissions. In delayed transmission, a packet is held back for a random amount of time until there are enough tokens to transmit it as a marked packet whereas in timed transmission each flow with a reservation maintains a periodic timer, transmitting a new packet at the expiration of the timer. This poses a significant overhead given that the granularity of the timers used is as low as 20ms. A key goal of GTCP is to avoid any additional timer overheads, and still achieve desired results. Furthermore, the approach in [1] is based on TCP Reno while GTCP's design is based on NewReno.

[4] proposes several schemes including limiting the number of out of profile packets transmitted, deploying an inverse-rate drop mechanism (where the drop probability at the router is inversely proportional to the reservation level of the flow), and using three drop priorities. These mechanisms require network support at the core as opposed to our approach which is applicable to any end-system mechanisms.

D. Design goals

In order to improve TCP's performance in networks with bandwidth guarantees, we propose an enhanced version of TCP called GTCP that address the previously mentioned problems. The following are the key design goals of GTCP: (a) *Reserved bandwidth awareness:* GTCP should recognize bandwidth reservations made by connections, and reliably deliver the reserved bandwidth to the application. (b) *Service aggregation:* GTCP should be able to use any best-effort bandwidth given by the underlying network, and deliver the aggregate of the reserved and best effort throughput to the application. (c) *TCP-friendliness:* The best effort service enjoyed by a GTCP connection should always be exactly the same as what a single TCP-flow without reservations would obtain, independent of the reservations made for the connection. (d) *No additional overheads:* GTCP should not incur any additional overheads such as timers to attain desired throughput.

III. The GTCP Design

In this section we describe the GTCP design. GTCP uses the same mechanisms as TCP for flow control, sequencing, reliability, and connection management (establishment and tear-down). It differs from TCP only in its congestion control mechanisms, and hence we elaborate only on these parts in this section.

A. Round-trip Time Estimation and Congestion Window

Since TCP uses a window based congestion control scheme, any bandwidth reservation done for a connection has to be translated into an appropriate window size. A simple approach to perform the translation is to set the reserved component of the congestion window as follows:

$$cwnd_G = rate_G * rtt_{base}$$

where rtt_{base} is the round-trip time for the connection. rtt_{base} differs from the default round-trip time used by TCP in

[4]Note that even if a loss occurs somewhere in the middle of the congestion window, the ACKs for the segments before the loss would have shifted the left edge of the congestion window to the sequence number corresponding to the segment that is lost.

that it approximates only the sum of the transmission and propagation delays along the forward and reverse paths used by the connection[5]. The rtt_{base} is calculated just as in TCP-Vegas [5] by keeping track of the minimum round-trip time sample experienced by the packets in the connection:

$$rtt_{base} = min(rtt_{base}, rtt_{sample})$$

Once the reserved bandwidth component of the congestion window is computed as above, GTCP's steady state congestion window *cwnd* is maintained at:

$$cwnd = cwnd_G + cwnd_{BE}$$

where $cwnd_{BE}$ is the best-effort component of the window. *Note that $cwnd_{BE}$ for a GTCP connection will always remain the same as the $cwnd$ it would have had if it were a default TCP connection without any reservations, thus ensuring the TCP-friendly nature of GTCP*. In the rest of the section we elaborate on how the congestion window is updated, and how the properties of GTCP are achieved.

B. Start-up Behavior

GTCP uses slow-start for both the reserved bandwidth and the best-effort components. This is to ensure that large packet bursts are not sent into the network causing buffer overflows. The initial congestion window of the connection is set to *two* (*one* for the best-effort component and *one* for the reserved bandwidth component). For every ACK received during slow-start, the *cwnd* is incremented. GTCP exits from slow-start when *cwnd* is larger than the sum of the default $ssthresh$ and the ideal reserved bandwidth component of the congestion window ($rate_G * rtt_{base}$).

C. Congestion Avoidance

In steady state, TCP performs congestion avoidance by increasing its congestion window by one every round-trip time. GTCP uses the same congestion avoidance mechanism of TCP. For every ACK received during congestion avoidance, $cwnd$ is incremented by $\frac{1}{cwnd}$. Note that although GTCP does not explicitly maintain the best-effort and reserved components of the congestion window separately, $cwnd$ will be the sum of $cwnd_G$ and $cwnd_{BE}$.

D. Congestion Control

In TCP, when a loss event is detected through the receipt of three DUPACKs, the $cwnd$ is halved. In GTCP, since only the best-effort component of the $cwnd$ is halved, and the window update is performed as follows:

$$cwnd = cwnd_G + \frac{cwnd_{BE}}{2}$$

This is similar to the approaches proposed in related work [1]. However, *GTCP differs from such approaches in its fast recovery and timeout handling mechanisms.*

(1)*Fast Recovery:* Fast recovery is critical to connections with bandwidth reservations as otherwise no new data will be sent during the loss recovery period. Ideally, if there are k losses and the congestion window at that stage is $cwnd$, then the ideal number of new segments that need to be sent every round-trip time should be

$$FR_{ideal} = max(cwnd_G, cwnd - k - \frac{cwnd_{BE}}{2})$$

The default fast recovery mechanism coupled with the window adaptation described earlier can sustain the desired self-clocking for small number of losses. However, if the number of losses k is larger than half of the best-effort component of the congestion window ($\frac{cwnd_{BE}}{2}$), TCP will not send the desired amount of data, as illustrated in Figure1.

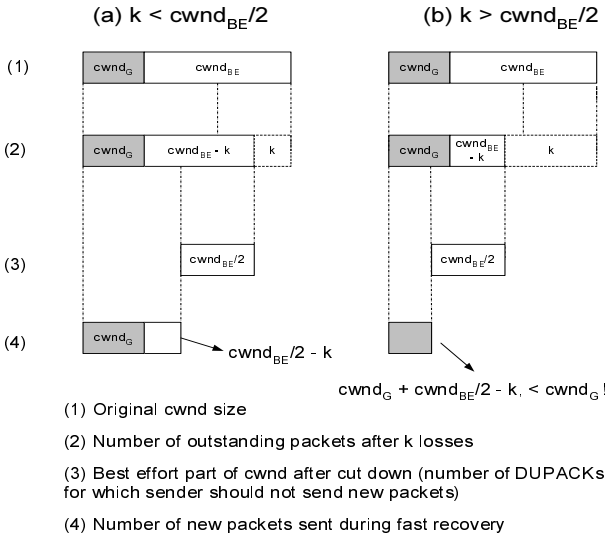

Fig. 1. Default TCP Fast Recovery Behavior

In order to solve this problem, GTCP uses a modified fast recovery mechanism that is described below:

- When a loss event is detected, GTCP does not adjust the congestion window immediately, but saves the updated window size $cwnd_G + \frac{cwnd_{BE}}{2}$ as $cwnd_{update}$.
- GTCP performs *forced data transmissions* for the first $cwnd_G$ DUPACKs immediately following a loss detection, and expands $cwnd$ for each transmission.
- After $cwnd_G$ transmissions, GTCP ignores the later $\frac{cwnd_{BE}}{2}$ (or $cwnd_{BE} - k$ if $k > \frac{cwnd_{BE}}{2}$) number of DUPACKs (no transmissions are triggered and $cwnd$ remains the same).
- GTCP transmits new data segments for further DUPACKS (if there are any)
- When an ACK (acknowledgment) arrives for the maximum sequence number at the time of loss detection (termed a Full ACK), GTCP collapses its window to $cwnd_{update}$.

The modified fast recovery algorithm is illustrated in Figure 2.

Because we assume that $cwnd_G$ number of packets are reliably transmitted by the network, it is guaranteed that there

[5]TCP's default rtt estimate includes the queuing delay.

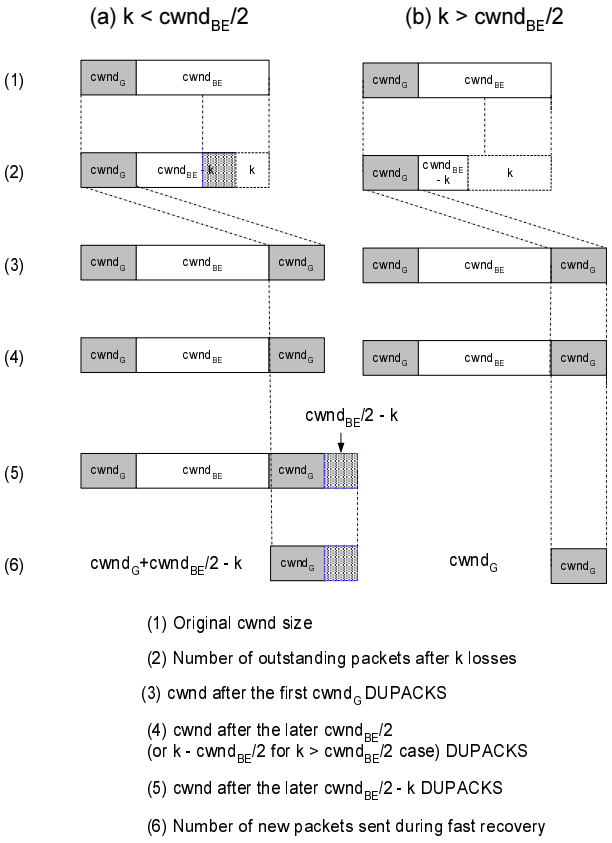

Fig. 2. GTCP Fast Recovery Behavior

are enough DUPACKs coming back to trigger at least $cwnd_G$ more number of packet transmissions after loss indication.

Using this algorithm, when the number of losses $k < \frac{cwnd_{BE}}{2}$, GTCP's behavior will be exactly the same as that of TCP NewReno, but the sequence in which the window is expanded, and $cwnd_G$ worth of new data packets is transmitted, is switched. However, when $k > \frac{cwnd_{BE}}{2}$, at least $cwnd_G$ number of packets are sent out, satisfying the FR_{ideal} identified earlier.

(2)*Timeout Recovery:* Timeouts can occur in default TCP due to two reasons - (a) less than four packets in a congestion window delivered successfully at the receiver and (b) a retransmitted packet lost again. However, for connections with bandwidth reservations, given our service model assumption that marked packets will not be dropped in the network, only the second reason can lead to a timeout as long as $cwnd_G$ is greater than four.

Hence, we change GTCP to successfully maintain its self-clocking during the timeout recovery period such that at least $cwnd_G$ worth of packets are transmitted every round-trip time. The design principle is similar to that of the modified fast recovery algorithm.

- When timeout occurs, the congestion window $cwnd$ is collapsed to one (not $cwnd_G + 1$), representing the true best-effort component of the congestion window, and the maximum sequence number transmitted thus far is saved as $ssexit_{thresh}$.
- For each of the first $cwnd_G$ number of DUPACKs, GTCP transmits a new data segment. The latter DUPACKS are not responded until the total number of incoming DUPACKS exceed the number of outstanding packets when timeout occurs. After that, a new data segment is transmitted for every incoming DUPACK.
- When the incoming ACK is not a DUPACK, and hence shifts the left edge of the $cwnd$ and expands the $cwnd$, GTCP retransmits all the data segments that lie within the $cwnd$ after the shifting and the expansion. This behavior is exactly the same as that of default TCP during slow-start.
- Finally, when the left edge of the $cwnd$ is shifted beyond $ssexit_{thresh}$, GTCP exits slow-start and collapses its window to $cwnd_G + cwnd_{BE}$, which is the size of the $cwnd$ when the last expansion occurs.

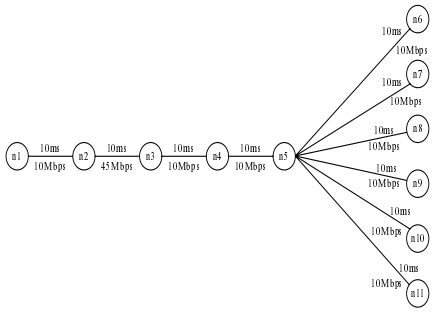

Fig. 3. Network Topology: Flows f0-f5 all originate from n1 and terminate at nodes n6-n11 respectively.

Note that for timeout recovery, GTCP performs a slow-start only for the best-effort component of the congestion window and the reserved bandwidth component is maintained at the reservation level. Hence, GTCP's start-up behavior is different from that of its timeout recovery behavior.

In Section IV we evaluate GTCP's performance and compare it with that of default-TCP and the ideal expected performance. In addition to the properties presented in this section, we show through our performance evaluations that GTCP scales well for a variety of factors.

IV. Simulations

In this section, we compare GTCP's performance against that of default TCP. We use the *ns2* network simulator for our simulations.

Bandwidth reservations are implemented in the ns2 simulator to emulate the intserv guaranteed service model mechanism. Flows with reservations are identified by their source and destination addresses. Node n2 serves as the edge router where metering and packet marking is done.

We use throughput as the main performance metric in the simulation, and differentiate between aggregate throughput,

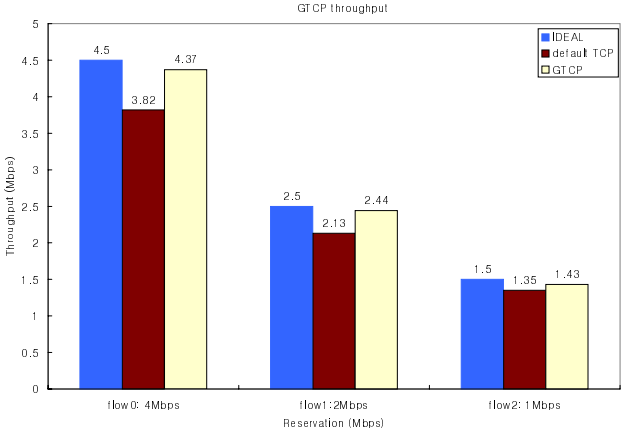

Fig. 4. GTCP Throughput (10Mbps Bottleneck Capacity): Flows f0, f1, and f2

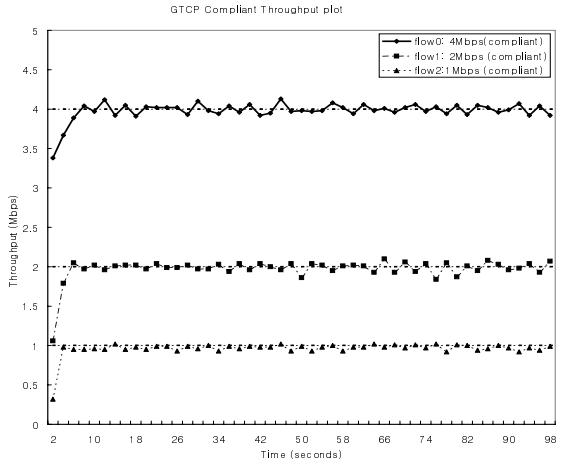

Fig. 5. GTCP Compliant Throughput (10Mbps Bottleneck Capacity): Flows f0, f1, and f2

best effort throughput, and compliant throughput (corresponding to reserved bandwidth) when necessary.

A. Basic Scenario

We present results for GTCP using the network topology described in Figure 3. There are 6 flows, 3 of them (f0-f2) have reservations of 4Mbps, 2Mbps and 1Mbps respectively, and the remaining 3 are best effort flows. Hence we expect to ideally achieve rates of 4.5Mbps, 2.5Mbps and 1.5Mbps for the three reserved flows respectively. Figure 4 shows that GTCP is able to achieve close to the expected throughput for the three reserved flows. The addition of $rate_G$ worth of packets to the congestion window at start-up allows the reserved flow to get their reserved rate from the start of the connection. Also the modification to TCP's congestion control mechanism after a loss occurs ensures that the flow can still enjoy its reserved rate when a loss occurs. These two mechanisms allow the reserved flows to achieve their compliant throughput, and enjoy a fair share of the excess bandwidth.

Figures 5 and 6 show the compliant and non-compliant(best-effort) components of the aggregate instantaneous throughput. It is observed that GTCP is able to achieve the sum of the reservation rate, and the fair share of the excess bandwidth as desired.

B. Scalability

First we investigate whether GTCP's performance scales as link capacity increases, and reservation levels increase proportionately. Figure 7 shows that as the link bandwidth increases from 10Mbps to 30Mbps, and the reservation level is increased by the same proportion, GTCP is still able to achieve close to the expected throughput. This is expected since the offset added to the congestion window is directly proportional to the reserved bandwidth. The advertised receive window size could potentially pose a bound on the sender's

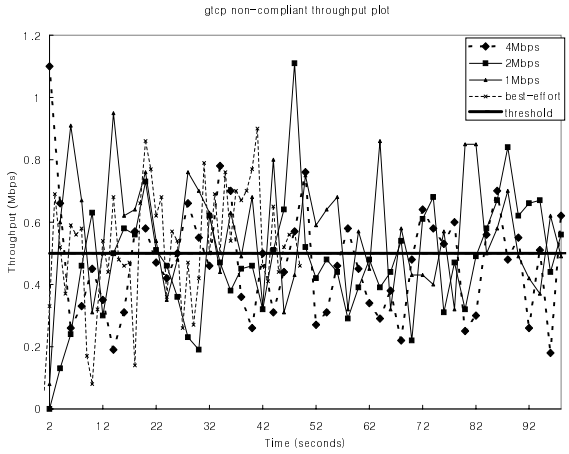

Fig. 6. GTCP Non-compliant Throughput (10Mbps Bottleneck Capacity): Flows f0-f5

transmission rate. However, TCP's window scaling option is used to overcome the problem.

In Figure 8, we present results to show that as the number of flows increases (and the number of flows with reservations is scaled up by the same proportion) GTCP still scales. In this scenario, the link capacity is kept constant and half of it reserved for the flows with bandwidth guarantees. Hence the the per-flow target rate reduces as the number of flows with reservations increases. However the realized rate still closely tracks the target rate as shown in Figure 8. As expected, TCP's performance is better when the target rate is lower, but is significantly worse when the target rates are higher.

Next we proceed to show that as the number of flows with bandwidth reservations increase (assuming total number of flows is held constant), GTCP scales. The number of

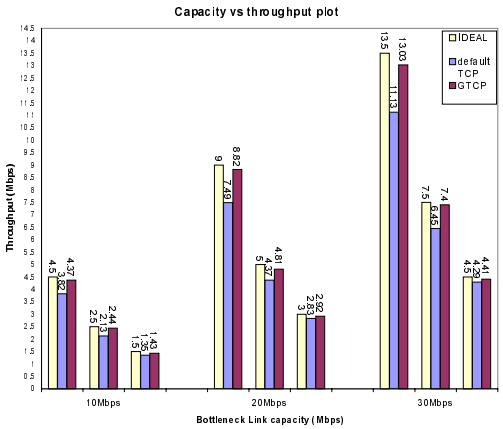

Fig. 7. GTCP Scalability with Capacity: Throughput - Flows f0, f1, and f2

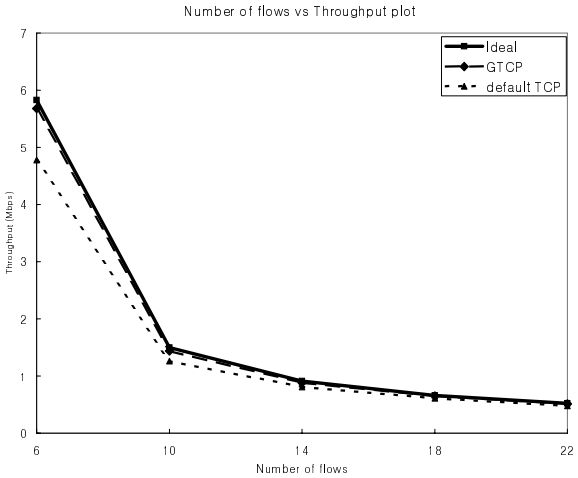

Fig. 8. GTCP Scalability with Number of Flows: Throughput

flows, reserved bandwidth, the ideal throughput and achieved throughput of each flow are demonstrated in Figure 9. As the number of flows with reservations increases, the proportion of reserved bandwidth when compared to the bottleneck capacity increases. Figure 9 shows that even when we have 5 reserved flows with a total reservation of 10Mbps, (same as link bandwidth) and 1 best effort flow, GTCP still tracks the target rate quite well for all the flows. However, as the number of flows increases, default TCP is unable to achieve the target rate.

V. Conclusions

In this paper, we address the problem of enhancing TCP's performance over networks with guaranteed bandwidth services. We show that default TCP cannot achieve the desired aggregate bandwidth which is the sum of the reserved and best-effort components. We propose a enhanced version of TCP called GTCP, that achieves the optimal aggregate performance without relying on additional overheads (when compared to default TCP). GTCP involves changes only to the congestion control mechanisms of TCP, and hence does not require any changes to the TCP receiver. Through simulations we show that GTCP achieves the desired performance in a variety of network scenarios.

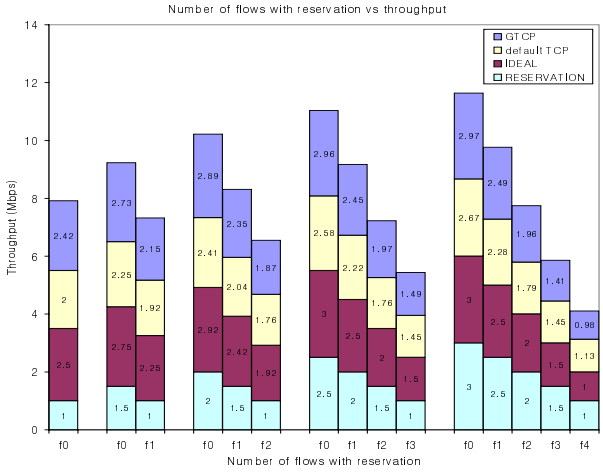

Fig. 9. GTCP Scalability with Number of Flows with Reservation: Throughput

References

[1] S.D. Shin, K. Feng W.and Kandlur D., "Understanding and improving tcp performance over networks with minimum rate guarantees," *transnet*, vol. 7, no. 2, pp. 173–187, Apr. 1999.

[2] R. Braden, D. Clark and S. Shenker, "Integrated services in the internet architecture: an overview," Internet Request for Comments RFC 1633, June 1994.

[3] R. Braden, L. Zhang, S. Berson, S. Herzog and S.Jamin "Resource reservation protocol (rsvp) - version 1 functional specification," Internet Request for Comments 2205 (rfc2205.txt), Sept. 1997.

[4] I. Yeom and A. L. Narasimha Reddy, "Realizing throughput guarantees in a differentiated services network," in *IEEE Int. Conf. on Multimedia Computing and Systems*, June 1999.

[5] L. Brakmo and L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM Symposium*, Aug. 1994.