Bandwidth Aggregation with SCTP

Antonios Argyriou and Vijay Madisetti
School of Electrical & Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA
Email: {anargyr, vkm}@ece.gatech.edu

Abstract— **In this paper we present a number of modifications to the recently adopted by IETF Stream Control Transmission Protocol (SCTP), that allow bandwidth aggregation over the multiple interfaces of a host. We show that it is possible to implement a number of algorithms for bandwidth aggregation, with only a small number of modifications to the base SCTP protocol. Our simulation results clearly depict the efficiency of our approach in terms of bandwidth utilization. Furthermore, we implement and evaluate a mechanism for identifying bottlenecks that are shared by flows from the same aggregate connection. Our purpose is to show that SCTP is a good candidate for building a practical protocol for bandwidth aggregation that is fair and supportive of TCP.**

I. Introduction

Bandwidth was always an immediate metric for the user perceived QoS. That is why end-user terminals are equipped today with a number network interfaces (LAN/WLAN) which in the near future will probably belong to various access technologies. Ideally, a user would like to use all the available aggregate bandwidth provided by the available interfaces. Moreover, service providers require better resource utilization for their equipment and one way to achieve this is by accommodating more users to the same infrastructure. Using concurrently multiple technology interfaces (aka multiple access networks) can be a source of increased revenue for them. Thus, the problem that formulates is how to use all theses interfaces in order to obtain the maximum possible bandwidth in a fashion friendly for the Internet and the dominant TCP traffic.

In this paper we propose a mechanism for efficient bandwidth aggregation (or load-sharing) across the multiple interfaces of a multi-homed host. The proposed approach consists of a number of modifications to the Stream Control Transmission Protocol (SCTP), recently adopted by IETF [1]. This protocol includes a number of novel features, including support for multi-homing. However, the current SCTP specification does not implement bandwidth aggregation due to the difficulty of the problem. We believe that due to SCTP's novel features, it has the potential of providing a robust protocol for practical bandwidth aggregation over the Internet. Our paper tries to give a first direction towards a practical bandwidth-aggregation solution based on SCTP that will be able to operate smoothly over the current Internet. Additionally, we evaluate a mechanism for solving the most important problem of a bandwidth aggregation system: TCP-friendliness/fairness of the aggregate connection even when the sub-flows from the same host converge to the same bottleneck router. Overall, our system functionality can be summarized into three key points: 1) shared bottleneck identification 2) recovery with a modified congestion control algorithm and 3) tackling out-of-order delivery of packets due to path asymmetries.

The rest of this paper is organized as follows: In section II we give an overview of work related to bandwidth aggregation and load-sharing. Due to the recent emergence of SCTP, we provide a brief overview of the protocol in section III. Section IV provides motivation for our work and analyzes in detail the problems we are dealing with. The next section describes in detail our algorithms and the proposed modifications to SCTP. The algorithm for shared bottleneck identification and recovery is presented in section VI. Finally we present simulation results in section VII, while section VIII concludes the paper.

II. Related Work

A number of application layer techniques for bandwidth aggregation have been presented [2], [3], [4]. The common practice at this level is to establish multiple TCP connections (each one mapped to a different interface) and then stripe application data according to the available bandwidth of each link. Bandwidth estimation is performed either passively (RTT measurements) or actively using probes. These approaches obviously tradeoff accuracy for wasted bandwidth. Another major problem is the need of large re-sequencing buffers in the case of significant mismatch in the available bandwidth of each link [5]. Recently in [4], was presented a application layer bandwidth aggregation system, that was built based on the analytical results presented in earlier work [6]. Their system performs well in identifying shared bottlenecks, except the case of sources that exhibit bursty traffic patterns.

Moving down to the protocol stack, we find the first transport layer bandwidth aggregation technique being reported in [7], in the form of a specialized protocol named RMTP. This protocol is based in the accurate bandwidth estimation of each path for proper data striping. However, since the bandwidth estimation accuracy depends largely on the frequency of probes sent to each path, significant bandwidth is given out for probing. Recently, Hsieh [5] proposed a modified version of TCP, called P-TCP, for bandwidth aggregation in mobile multi-homed hosts. However, the assumption of sub-flows that do not share a bottleneck limits the applicability of the protocol in the current Internet. Moreover, the additional requirement that applications must be built by being aware of the P-TCP protocol, poses an additional burden for practically deploying the proposed system.

A network layer approach was reported in [8]. The authors take a different approach to the problem by defining a bandwidth aggregation scheme at the network layer (IP). They claim that their system is transparent to the transport layer and it aggregates data of a single TCP session over the outgoing interfaces. The major problem with this approach is that of packet reordering at the receiver, when a connection spawns over multiple paths. To solve that, they propose a modification of the RTO calculation algorithm, which does not consist a transparent solution for TCP.

III. SCTP Overview

The Stream Control Transmission Protocol (SCTP) is a reliable transport protocol that operates on top of a connectionless packet based network such as IP. One of the most important new ideas that SCTP introduces is that of multi-homing. A single SCTP association (session) is able to use alternatively anyone of the available IP-addresses without disrupting an ongoing session. However, this feature is currently used by SCTP only as a backup mechanism that helps recovering from link failures. SCTP maintains the state of each IP-address (path) by sending heartbeat messages and it is thus able to detect a specific link failure and switch to another IP-address/interface. Another novel feature is that SCTP decouples reliable delivery from message ordering by introducing the idea of $streams$. The $stream$ is an abstraction that allows applications to preserve in order delivery within a $stream$ but unordered delivery across $streams$. This feature avoids Head-of-Line (HOL) blocking at the receiver in case multiple independent data streams exist in the same SCTP session. Congestion control was defined similar to TCP, primarily for achieving TCP friendliness [1].

IV. Problems in the context of SCTP

Generally, the challenge is how to send a stream of data over multiple outgoing interfaces in order to take advantage of the full aggregate bandwidth. The constraint that we have is that the total aggregate connection must not be more aggressive than a single connection (TCP-friendly). We identify the following problems that an SCTP-based bandwidth aggregation mechanism has to face:

A. Packet reordering

Out-of-order delivery of packets at the receiver is actually common in the Internet today [9]. In our case, the problem is compounded even more since we intentionally distribute data that belong to a single data flow over a number of outgoing interfaces which may actually correspond to highly asymmetric paths (in terms of delay or bandwidth). Given that a number of packets, stamped with a Transmission Sequence Number (TSN) [1], are sent over the interfaces it is clear that a larger than usual, re-sequencing buffer is needed at the receiver. Additionally, packet reordering at the receiver results in the reception of duplicate acknowledgments at the sender. After four successive duplicate acknowledgments [1], the sender fast retransmits the missing TSN which **may** actually still be buffered, or is in its way at the highest delay link. As explained in [10], in the case of simple changeover (primary interface change), the previously described situation may lead in the worst case to retransmissions equal to the number of TSN outstanding at the ”slow” link. One additional problem is congestion window ($cwnd$) behavior which may not only erroneously increase as pointed out in [10], but it can much worst, drastically decrease (halved successively) if the ”slow” link delays significantly the delivery of the corresponding TSN's. This of course has as a side-effect the under-utilization of the high bandwidth link.

B. TCP-friendliness

Spawning a data flow over a number of outgoing links can have severe effects over the TCP-friendliness of the protocol and the fairness of the aggregate connection. Assuming that we maintain one set of congestion control parameters for each transport address of the receiver then this may lead to under-utilization of the aggregate bandwidth since the ”slowest” link will set back the $cwnd$ of the whole aggregate connection. The naive solution to this problem which requires congestion control performed for each specific flow that corresponds to one interface at the sender, has a major problem. The flows may share the same bottleneck router and thus contribute to congestion and ”steal” bandwidth from other TCP flows. What we would ideally want in this case, is to perform unified congestion control for the flows that share the bottleneck, so that the aggregate connection is TCP-friendly. Moreover, there is another issue that has to be considered: We do not only require the location of bottleneck routers but also routers that are simply shared by two flows from the same SCTP association. This is necessary in order to prevent fairness problems between other, SCTP/TCP flows.

V. Proposed algorithms

In the next few sections we analyze the number of modification performed to the base SCTP protocol. All the subsequent algorithms are based in one crucial modification that we did: We do not apply congestion control for each transport address separately but rather perform a unified congestion control for flows that share bottleneck (i.e for each flow that corresponds to a pair of source/destination addresses).

A. Congestion window based data allocation

One crucial difference of our approach compared to related work, is that a single data flow is distributed to each outgoing interface, according to the congestion window value of each specific transport address. We do not use the bandwidth of each link in order to estimate the amount of data that should be assigned. The rationale behind this is to fill the bandwidth-delay product of each link. In this way, we can also solve the problems that occur when the used links are highly asymmetric

[1]Corresponds to TCP's sequence number.

in delay or bandwidth. In [5], the authors are following a similar approach for TCP but after they logically separate the sender in a number separate TCP ”sender entities” in order to be able to apply congestion control for each link. In our case we handle all this in a unified way at the base SCTP protocol. One additional difference is that we maintain data in flight for all destinations. However, data are assigned to an interface **after** we apply congestion control for a particular destination. In this way data can be sent immediately. By following this approach, we avoid the need for dynamic data reassignment in case we have stale data for a particular destination which has decreased its congestion window.

B. Fast retransmit algorithm modification

Fast retransmit of Gap reports is described in section 7.2.4 of the SCTP RFC [1]. According to this section, when there is a gap in the TSNs of the received chunks, the receiver will send a SACK chunk reporting the gap. After reception of four duplicate acknowledgments (SACKs) [1], the sender will retransmit the missing TSN. But as we said in an earlier section, the receiver will send the SACK over the high bandwidth link while the ”missing” TSN chunk can still be travelling through the low bandwidth link. So if the high bandwidth link is a few times faster when compared with the slower link, the threshold of four gap reports, will soon reach to an end and the sender will retransmit the chunk. The sender will also invoke slow-start [1] and reduce the congestion window, leading thus to under-utilization of the high bandwidth link [10].

In our case of concurrent link utilization, the problem becomes even worse: Receiving out of order TSNs will be the usual case since link delay mismatches is a common phenomenon. Applying data striping based on the congestion window value, will not solve the problem [5] and so the need of a large re-sequencing buffer is immediate. However, we can eliminate side-effects like spurious retransmission, and congestion window overgrowth or shrinking. Our system uses the following technique in order to overcome the above problems: Each flow that corresponds to a link consists a separate data pipe which monitors TSN order only for the packets transmitted in this pipe. This means that even if the receiver has not received successfully all the TSN’s from a specific address it will not send a gap report even it identifies a gap in the received TSN’s. It will only send a gap report to the original address from which the missing TSN was sent. Total chunk ordering is provided by the re-sequencing buffer. *Example*: if TSNs 10-20 were sent from $interface1$ of $host1$ to $interface1$ of $host2$, and we observe that TSN 13 is missing at the receiver, then a SACK with gap report will only be sent to $interface1$ of $host1$.

For someone familiar with SCTP, it might look that this mechanism could be implemented by exploiting the notion of SCTP $streams$ by mapping a stream to each interface because in-order delivery is maintained for each of them. However, we avoid doing that because: 1) SCTP does not provide a total order mechanism across streams, 2) $streams$ is an application layer facility which is used by applications to define more flexible delivery mechanisms, and 3) we might want to multiplex multiple applications streams at each path.

Implementation details: The sender maintains two variables that keep the lowest and highest TSN sent during the last congestion window round to the receiver. When the receiver replies with a SACK that contains a gap report for TSNs that do not belong to this range, the sender does not increase the Gap Ack reports and process the SACK chunk as a normal SACK that acknowledges the outstanding data for this transport address. On the meantime the receiver monitors its CumTSNack and when data are received from a different interface fill the gap then the receiver stops sending gap reports.

VI. Identifying shared bottlenecks

As we said in section IV, the most challenging problem in the bandwidth aggregation scheme, is how to locate a common congestion point for two flows that each one corresponds to one outgoing interface of the same host. This problem is orthogonal with what we have dealt so far. Moreover, even if the existence of shared routers does not imply a shared bottleneck, we still want to identify this case as we will later explain.

Our goal is to built a mechanism at the sender that can understand when a number of its flows share a common bottleneck and then apply congestion control in the aggregate of these flows. In this way the macro-flow that consists of flows sharing the same bottleneck will have a TCP-friendly aggregate traffic and the most important: **It will be fair with other flows sharing the same bottleneck**. In [11] the authors were trying to optimize the same problem at one layer above: They were grouping flows to a macro-flow according to the outgoing interface.

Given an SCTP association, we are able to define the flows of interest: Each possible source-destination pair of addresses/interfaces can be a separate data flow pipe. Our approach is simple: The sender observes the network packet delays (each RTT) and calculates correlation between packets that belong to different transport addresses. Additionally the receiver calculates auto-correlation from packets of the same flow. The general correlation test used is the one found in [4], [6]:

$$r_{xy} = \frac{\Sigma(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\Sigma(x_i - \overline{x})^2 \Sigma(y_i - \overline{y})^2}} \qquad (1)$$

The $\overline{y}$ and $\overline{x}$ variables represent the average value of packet delays over a period of RTTs (ten in our case). The correlation tests are performed in terms of delay and not packet losses, since it has been shown [6], that delay correlation tests converge much faster. If the cross-correlation M_x between packet delays of different flows, is larger that the auto-correlation M_a of one of the flows, then this means that the flows share a common bottleneck/router. For M_x packet samples must have difference $t > 0$ while for M_a it must be $T > t$ [6]. In addition, delay samples are obtained from the RTT measurements already available at the base protocol.

Even though timestamping would assure more accuracy, the relative gains are small as shown in [4], and for the sake of simplicity we used the RTT samples.

One additional feature of our algorithm is that it is reactive: flows that are initially assigned to different interfaces are assumed not to share a common bottleneck. However, after an amount of time the algorithm responsible for identifying the shared bottleneck will respond to a possible shared bottleneck by ”constructing” a macro-flow that consists of all the flows sharing the same bottleneck.

Decision for creation of macro-flows take place after a time T which depends on the number of necessary RTT samples that must be taken. N RTT samples must have been received for all the flows, number which is the minimum number of RTT samples that must be received by a flow so that its correlation tests are valid. We add a number of 10 more samples [4], for better accuracy and then the correlation tests are performed.

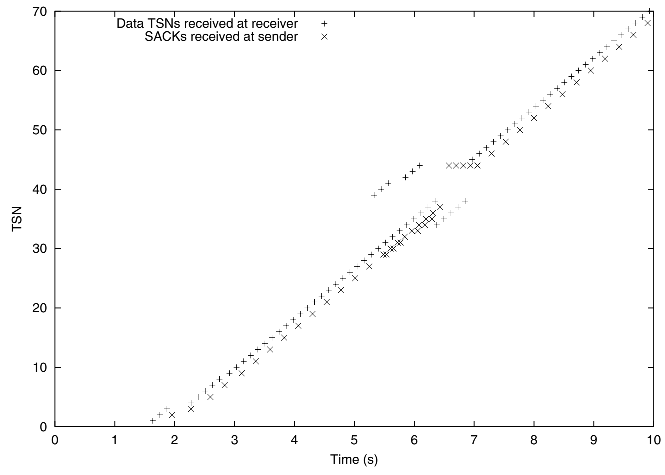
VII. Simulation results

The widely used ns-2 simulator was used for our experiments [12]. Additionally we used the SCTP ns-2 module available at [13]. Here we present representative results concerning the two operational modes of our system: One set in the absence of shared bottlenecks and one set of experiments when there is a shared bottleneck. Clearly, further experimentation is needed, but due to the lack of space, a small part of the available results is presented.

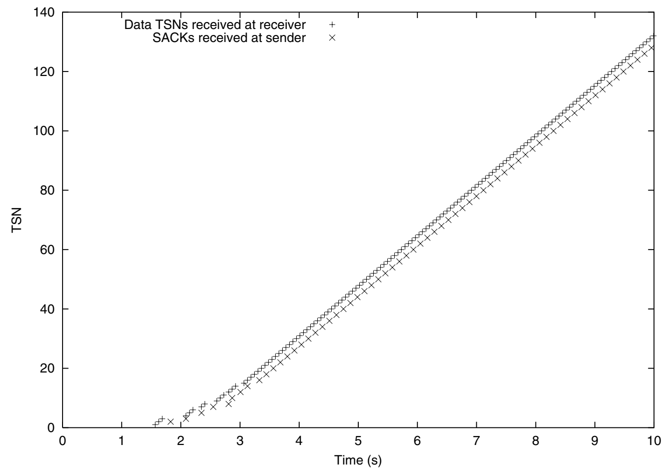
A. Without shared bottleneck

We assume the simple network topology of two hosts communicating through two non-overlapping paths. Both the links are 200Kbps, 200ms delay (i.e. they are symmetric). Fig. 1(a) presents TSN progression at the receiver and acknowledgements at the sender for the unmodified SCTP. In Fig. 1(b) we present the same network topology but with a sender enhanced with our bandwidth aggregation modifications. We can clearly observe that the receiver receives smoothly the TSNs. Note that in this case we do not use any large re-sequencing buffer which is possible due to the symmetric links. Moreover, we can see that the modified SCTP receiver receives nearly double number of TSNs in the same amount of time.

In the next couple of figures we present results concerning the same simple topology, but with asymmetric links. We observe that in this asymmetric case, unmodified SCTP (Fig. 2(a)) operates nearly smoothly with small perturbations. This is because after changeover it adapts to the delay of the new link. During our experiments we observed that even with small asymmetries, SCTP will still have problems resulting in spurious retransmissions. Our version, with bandwidth aggregation enhancements, also performs smoothly (Fig. 2(b)) with perfect aggregation efficiency compared to standard SCTP. We can see that despite the path asymmetries our version of the protocol is not affected at all.



(a) unmodified SCTP

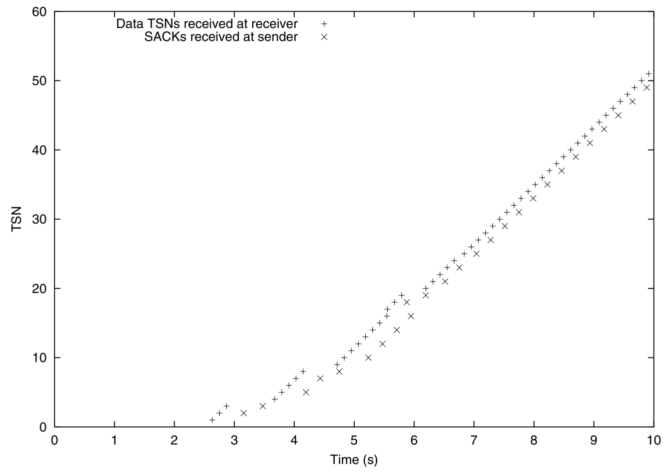


(b) SCTP with bandwidth aggregation support

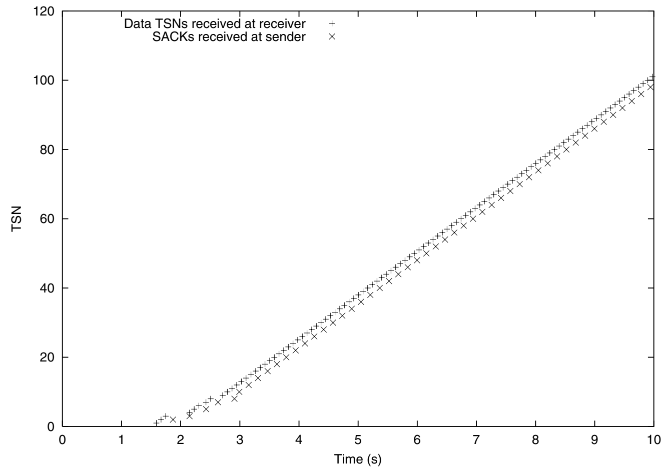
Fig. 1. Sequence number progression at the receiver with symmetric links (200kbps/200ms)

B. With shared bottleneck

In this sub-section, we present results concerning identification and recovery from shared bottlenecks. Due to lack of space we present a small set of a representative experiments that depict our approach. We used the topology of Fig. 3. All the links have parameters 100kbps for bandwidth and 200ms for delay. We used sources with infinite amount of data. Traffic routed from the second interface is mixed up with infinite background TCP traffic. We can see in Fig. 4(a) that packets coming from the same bottleneck router have similar delay variation or better: packet arrival times are closely correlated (lower part of Fig. 4(a) with interfaces 1 and 2). In figure 4(b) we mixed TCP traffic in the bottleneck router while we retained the TCP source at the second router. We can see that

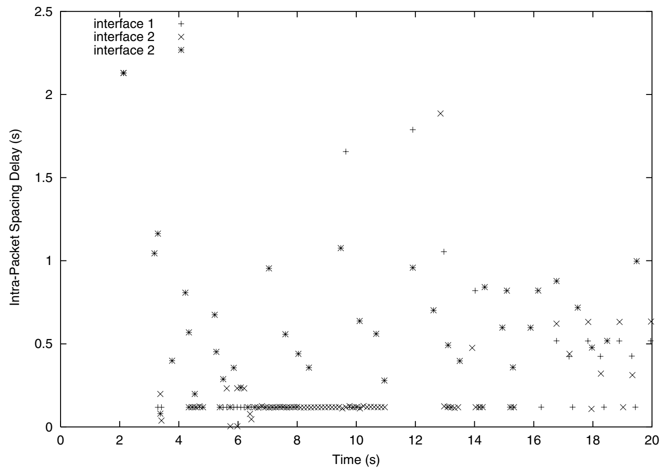


(a) unmodified SCTP (changeover happens at 5 sec)

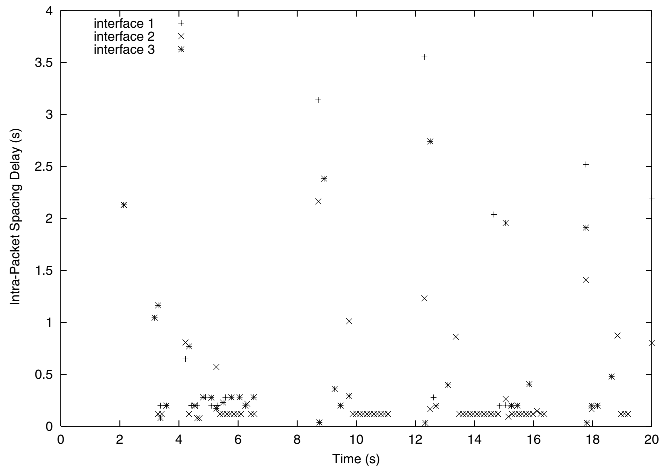


(b) SCTP with bandwidth aggregation support

Fig. 2. Sequence number progression at the receiver with asymmetric links (200ms/400ms,100/200kbps)



(a) No background TCP traffic at the bottleneck



(b) With background TCP traffic at the bottleneck router

Fig. 4. Intra-packet spacing delay

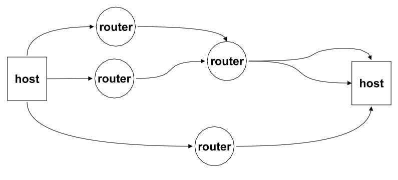


Fig. 3. Simple topology for simulation with shared bottlenecks

delay correlation is perturbed and as a result identification of the shared router becomes more difficult. However, our system is still able to identify the bottleneck. The time needed to identify the shared bottlenecks was 7 sec and 10 sec in Fig. 4(a) and Fig. 4(b) respectively.

Finally in Fig. 5 we present instantaneous throughput at the receiver in the second case (with background TCP traffic) after we activated the mechanism for identifying and recovering from shared bottlenecks. Additionally, in Fig. 5 we can see the fair bandwidth distribution among the the SCTP/TCP flows. The system is able to converge after nearly 10 sec as we previously said.

VIII. CONCLUSIONS

In this paper we proposed a bandwidth aggregation protocol that may be easily supported via SCTP. The modified protocol

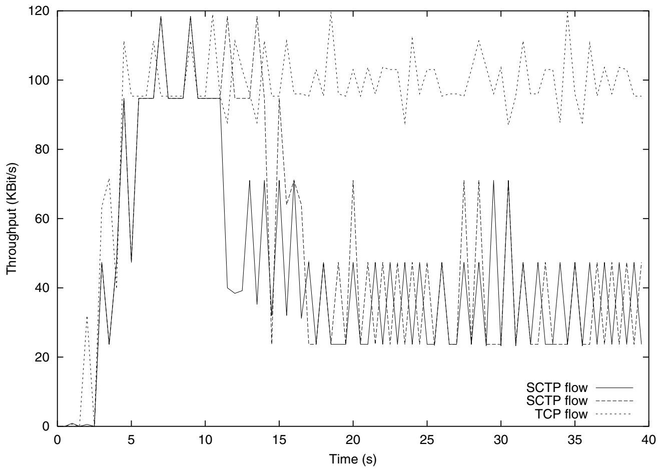


Fig. 5. Throughput at the receiver for the two SCTP flows and one TCP flow

allows seamless bandwidth aggregation for applications in the case where a host has number of interfaces. We also demonstrated the effectiveness of a well known method for identifying shared bottlenecks, when it is applied in a practical protocol. However, further study is needed in this area so that really complex topologies can be handled by the protocol.

REFERENCES

[1] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, “Stream Control Transmission Protocol,” RFC 2960, October 2000.

[2] H. Sivakumar, S. Bailey, and R. L. Grossman, “PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks,” in *Supercomputing*, 2000. [Online]. Available: citeseer.nj.nec.com/386275.html

[3] T. Hacker and B. Athey, “The end-to-end performance of effects of parallel tcp sockets on a lossy wide area network,” in *IEEE IPDPS*, Fort Lauderdale USA, 2002.

[4] O. Younis and S. Fahmy, “On efficient on-line grouping of flows with shared bottlenecks at loaded servers,” Tech. Rep., 2002. [Online]. Available: citeseer.nj.nec.com/younis02efficient.html

[5] H. Hsieh and R. Sivakumar, “A transport layer approach for achieving aggregate bandwidths on multi-homed hosts,” in *ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Atlanta, GA, September 2002.

[6] D. Rubenstein, J. F. Kurose, and D. F. Towsley, “Detecting shared congestion of flows via end-to-end measurement,” *IEEE/ACM Transactions on Networking*, 2002. [Online]. Available: citeseer.nj.nec.com/article/rubenstein02detecting.html

[7] L. Magalhaes and R. Kravets, “Transport level mechanisms for bandwidth aggregation on mobile hosts,” *Ninth International Conference on Network Protocols*, pp. 165–171, 2001.

[8] D. S. Phatak and T. Goff, “A novel mechanism for data streaming across multiple ip links for improving throughput and reliability in mobile environments,” in *IEEE INFOCOM*, vol. 2, 2002, pp. 73–781.

[9] J. C. R. Bennett, C. Partridge, and N. Shectman, “Packet reordering is not pathological network behavior,” *IEEE/ACM Transactions on Networking*, vol. 7, December 1999.

[10] J. R. Iyengar, A. L. Caro, P. D. Amer, G. J. Heinz, and R. R. Stewart, “Sctp congestion window overgrowth during changeover,” in *SCI 2002*, July 2002.

[11] H. R. Hari Balakrishnan and S. Seshan, “An integrated congestion management architecture for internet hosts,” in *ACM SIGCOMM*, Cambridge, MA, September 1999.

[12] S. McCanne and S. Floyd, “ns network simulator,” http://www.isi.edu/nsnam/ns/.

[13] A. L. C. et al., “SCTP module for ns-2,” http://www.pel.udel.edu.