SCTP Performance in Data Center Environments

N. Jani and K. Kant[1]

Intel Corporation

Abstract: Stream control transmission protocol (SCTP) is gaining more & more attention as a potential replacement for the aging TCP owing to several powerful features it provides. However, like TCP, much of SCTP work relates to WAN. In this paper, we examine SCTP from the data center perspective. In particular, we evaluate the most mature open source implementation of SCTP for Linux known as LK-SCTP. We find that the implementation is rather inefficient and can be improved substantially via a small number of targeted optimizations, many of which are approached from data center perspective. We also show that the protocol itself suffers from a number of weaknesses, which must be fixed in order for SCTP to become a preferred data center protocol.

Keywords: Stream control transmission protocol, data center, congestion control, hardware offload, selective acknowledgements.

1. Introduction

Data centers form the backbone of e-business and high performance computing. With the offering of sophisticated web-based services and increased complexity/size of the data to be manipulated, both performance and availability requirements of data centers continue to grow. This implies evolving requirements for the data center fabric as well. Unfortunately, much of the network research on data center related issues continues to focus on the front end. Our interest in this paper is on the fabric needs of *inside* the data center, i.e., all communications *except* direct client-server exchanges.

Although TCP/IP/Ethernet are well entrenched in data centers, this stack does not carry all of data center traffic. In particular, Fiber Channel is firmly entrenched in the storage area and specialized networks such as Myrinet, Infiniband (IBA), QSnet, etc. may be deployed for inter-process communication (IPC) for high-end systems. In particular, IBA was designed specifically as a universal data center fabric [1] and is beginning to gain acceptance in certain niche areas. However, Ethernet will continue to remain the technology of choice for the mass market because of a variety of reasons including entrenchment, incompatibility of IBA with Ethernet right down at the connector level, familiarity, commodity nature, etc. Furthermore, with a high data rate (10 Gb/sec already), HW protocol offload providing low overhead/latency [5,9], and maturing of IP based storage such as iSCSI, TCP/IP/Ethernet is likely to emerge as the *unified fabric* carrying all traffic types, possibly on the same “pipe”. Yet, although IP is expected to scale well into the future, there are legitimate questions about TCP for supporting data center

[1] Corresponding author: Krishna Kant, Email: Krishna.kant@intel.com. Address: Mail stop: JF1-231, Intel Corporation, 25111 NE 25th Avenue, Hillsboro, OR 97124, Phone (530)754-8579, Fax (503) 712-3754.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

applications demanding very low latencies, high data rates and high availability/robustness. Although TCP's weaknesses are well known, the force of legacy makes substantial changes to it almost impossible.

SCTP (Stream control transmission protocol) is a connection-oriented transport protocol designed to run over existing IP/Ethernet infrastructure [12]. Although it shares some features with TCP (particularly the flow and congestion control [2]), it is designed to be a lot more robust, flexible and extensible than TCP. SCTP was originally intended as the transport of choice for inter-working of SS7 and VoIP networks and was therefore designed with a view to support SS7 layers (e.g., MTP2 and MTP3) which have a number of unique requirements. Many of these features are useful in the data center environment which makes SCTP a better candidate for data center transport. Furthermore, the lack of entrenchment of SCTP makes it easier to change the protocol or its implementation in order to fine tune it for data centers. This paper provides a preliminary look at SCTP from the data center perspective along with the impact of some optimizations that we studied.

The outline of the paper is as follows. In section 2, we provide an overview of SCTP and discuss essential differences between data center and WAN perspectives on the protocol. One crucial difference in this regard is the efficiency of implementation. In section 3 we provide an "out-of-the-box" comparison between Linux TCP and SCTP implementations in terms of efficiency and discuss reasons for SCTP's poor performance. Section 4 discusses the results of several optimizations that we have already made. It also identifies several ways in which SCTP can be improved both in terms of implementation and in feature specification. Section 5 evaluates the relevance of SACKs in data center environment and concludes that they are actually detrimental to performance. Finally, section 6 concludes the paper.

2. SCTP features and data center requirements

Before discussing SCTP, let's first briefly review TCP. A TCP connection provides a reliable, *ordered byte-stream* abstraction between the two transport end-points. It uses a *greedy scheme* to increase flow of bytes at a rate commensurate with the round-trip times (RTTs) until it can't. The basic flow/congestion control algorithm used by TCP is the window based AIMD (additive increase and multiplicative decrease) where the window is increased slowly as the network is probed for more bandwidth but cut down rapidly in the face of congestion. AIMD schemes and their numerous variants/enhancements have been studied very extensively in the literature (e.g., see [5, 9] and references therein on analytic modeling of TCP). The continued "patching" of TCP over last two decades have made TCP flow/congestion control very robust in a hostile WAN environment. However, TCP lacks in a number of areas including lack of integrity/robustness checks, susceptibility to denial-of-service (DoS) attacks, poor support for quality of service, etc. Furthermore, since TCP was never designed for use within a data center, some of its weaknesses become particularly acute in such environments as we shall discuss shortly. Although SCTP also suffers from many weaknesses from data center perspective, it has the advantage of being an extensible protocol and can be modified by defining additional chunk types.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

2.1 Overview of SCTP

SCTP adopts congestion/flow control scheme of TCP except for some minor differences [2]. This makes SCTP not only "TCP friendly", but mostly indistinguishable from TCP in its congestion behavior except that the selective acknowledgement (SACK) mechanism is an integral part of SCTP. The negative side to this is QoS related issues and complexity as discussed later. However, SCTP does provide the following enhancements over TCP. We point these out primarily for their usefulness in a data center.

1. Multi-streaming: An SCTP association (or loosely speaking a connection) can have multiple "streams", each of which defines a logical channel, somewhat like the *virtual-lane* in the IBA context. The flow and congestion control are still on a per association basis, however. Streams can be exploited, for example, to accord higher priority to IPC control messages over IPC data messages and for other purposes [3]. However, inter-stream priority is currently not a standard feature.
2. Flexible ordering: Each SCTP stream can be designated for an in-order or immediate delivery to the upper layer. Unordered delivery reduces *latency;* furthermore, RDMA [11] based communication does not require ordering from the underlying transport.
3. Multi-homing: An SCTP association can specify multiple "endpoints" on each end of the connection, which increases connection level fault tolerance (depending on available path diversity). This feature will be particularly useful if it is possible to transmit data concurrently across multiple paths [4,13].
4. Protection against denial of service: SCTP connection setup involves 4 messages (unlike 3 for TCP) and avoids depositing any state at the "called" endpoint until it has ensured that the other end is genuinely interested in setting up an association. This makes SCTP less prone to DoS attacks. This feature may not be very useful *inside* the data center – however, since most connections within the data center are long-lived, the additional overhead of this mechanism is not detrimental.
5. Robust association establishment: An SCTP association establishes a verification tag which must be supplied for all subsequent data transfer. This feature, coupled with 32-bit CRC and heartbeat mechanism makes SCTP more robust than TCP. This is crucial within the data center at high data rates and can obviate CRC checks at higher protocol layers such as RDMA or iSCSI.

The key to SCTP's extensibility is the "chunk" feature. Each SCTP operation (data send, heartbeat send, connection init, …) is sent as a "chunk" with its own header to identify such things as type, size, and other parameters. A SCTP packet can optionally "bundle" as many chunks as will fit in the specified MTU size. Chunks are never split between successive SCTP packets. Chunks are picked up for transmission in the order they are posted to the queue except that control chunks always get priority over data chunks. SCTP does not provide any ordering or reliable transmission of control chunks (but does so for data chunks). New chunk types can be introduced to provide new capabilities, which makes SCTP quite extensible.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

2.2 Data center vs. WAN environments

Data centers have a number of requirements that are quite different from those for general WAN. We shall discuss some protocol performance issues in detail in the following. In addition, data centers require much higher levels of availability, robustness, flexible ordering, efficient multi-party communication, etc. which are crucial but beyond the scope of this paper.

In a WAN environment, the primary concerns for a reliable connection protocol are (a) each flow should adapt automatically to the environment and provide the highest possible throughput under packet loss, and (b) be fair to other competing flows. Although these goals are still important in data centers, there are other, often more important goals, that the protocol must satisfy.

Data centers are usually organized in tiers with client traffic originating/terminating at the front end server. The interior of the data centers portray multiple connected clusters, which is the main focus here. The key characteristics of these communications (compared with WAN) include: (a) much higher data rates, (b) much smaller and less variable round-trip times (RTTs), (c) higher installed capacity and hence less chances of severe congestion, (d) low to very low end-to-end latency requirements, and (e) unique quality of service (QoS) needs.

These requirements have several consequences. First and foremost, *a low protocol processing overhead is more important than improvements in achievable throughput under sustained packet losses*. Second, *achieving low communication latency is more important than using the available BW most effectively*. This results in very different tradeoffs and protocol architecture than in a WAN. A look at the existing data center protocols such as IBA or Myrinet would confirm these observations. For example, a crucial performance metric for data center transport is number of CPU cycles per transfer (or CPU utilization for a given throughput). It is interesting to note in this regard that most WAN focused papers do not even bother to report CPU utilization. Also, low latency and overhead means that packet losses should be actively avoided, rather than tolerated. In particular, delay based congestion control is much preferred in a data center than a loss based congestion control [14].

The data center characteristics also imply other differences. For example, the aforementioned fairness property is less important than the ability to provide different applications bandwidths in proportion of their needs, negotiated SLAs, or other criteria determined by the administrator. Also, the data center environment demands a much higher level of availability, diagonosability and robustness. The robustness requirements generally increase with the speeds involved. For example, the 16-bit CRC used by TCP is inadequate at multi-gigabit rates and has led to CRC-32 being implemented in other protocol layers (e.g., MPA, iSCSI, RDMA, etc.)

Protocol implementations have traditionally relied on multiple memory-to-memory (M2M) copies as a means of convenient interfacing of disparate software layers. For example, in the traditional socket based communications, socket buffers are maintained by the OS separate from user buffers. This requires not only a M2M copy but also a context switch both of which are expensive. In particular, for large data transfers (e.g., in case of iSCSI transferring 8KB or large data chunks), M2M copies may result in

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

substantial cost in terms of CPU cycles, processor bus BW, memory controller BW and, of course, the latency. Ideally, one would like to implement 0-copy sends and receives and standard mechanisms are becoming available for the purpose. In particular, RDMA (remote DMA) is gaining wide acceptance as an efficient 0-copy transfer protocol [8, 11]. However, an effective implementation of RDMA becomes very difficult on top of a byte stream abstraction. In particular, implementing RDMA on top of TCP requires a shim layer called MPA (Marker PDU Alignment) which is a high data touch layer that could be problematic at high data rates (due to memory BW, caching and access latency issues). A message oriented protocol such as SCTP is by comparison can interface with RDMA much more easily.

There are other differences as well between WAN and data center environments. We shall address them in the context of optimizing SCTP for data centers.

3. TCP vs. SCTP performance

A major stumbling block in making performance comparison between TCP and SCTP is the vast difference in the maturity level of the two protocols. SCTP being relatively new, good open-source implementations simply do not exist. Two *native-mode*, non-proprietary implementations that we examined are (a) LK-SCTP (http://lksctp.sourceforge.net/): An open-source version that runs under Linux 2.6 Kernel, and (b) KAME (http://www.sctp.org): a free-BSD implementation developed by Cisco. We chose the first one for the experimentation because of difficulties in running the latter, lack of tools (e.g., Oprofile, Emon, SAR) for free-BSD, and more familiarity with Linux. The choice of Linux implementation also enhances practical relevance of the work.

SCTP was installed on two 2.8 GHz Pentium IV machines (HT disabled) with 512 KB second level cache (no level 3 cache) – each running R.H 9.0 with 2.6 Kernel. Each machine had one or more Intel Gb NICs. One machine was used as a server and the other as a client. Many of the tests involved unidirectional data transfer (a bit like TTCP) using a tweaked version of the iPerf tool that comes with LK-SCTP distribution. iPerf sends back to back messages of a given size. iPerf doesn't have multi-streaming capability. Multi-streaming tests were done using a small traffic generator that we cobbled up.

Before making a comparison between TCP and SCTP, it is important ensure that they are configured identically. One major issue is that of HW offloading. Most current NICs provide the capability of TCP checksum offload and transport segmentation offload (TSO). None of these features are available for SCTP. In particular, SCTP uses CRC-32 (as opposed CRC-16). We found that checksum calculation is very CPU intensive. In terms of CPU cycles, CRC-32 increases the protocol processing cost by 24% on the send side and a whopping 42% on the receive side! Quite clearly, high speed operation of SCTP *demands* CRC32 offload. Therefore, we simply removed the CRC code from SCTP implementation, which is almost equivalent to doing it in special purpose HW.

TSO for SCTP would have to be lot more complex than that for TCP and will clearly require new hardware. Therefore, we disabled TSO for TCP, so that both protocols would do segmentation in SW. However, one discrepancy remains. The message based nature of SCTP requires some additional work

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

(e.g., message boundary recognition on both ends) which TCP does not need to do. Also, the byte stream view makes it much easier for TCP to coalesce all bytes together.

3.1 Base Performance Comparisons

Table 1 shows the comparison between TCP and SCTP for a single connection running over the Gb NIC and pushing 8 KB packets as fast as possible under zero packet drops. (SCTP was configured with only one stream in this case.) The receive window size was set to 64 KB and is more than adequate considering the small RTT (about 56 us) for the setup. The reported performance includes the following key parameters:

(a) Average CPU *cycles per instruction* (CPI),

(b) *Path-length* or number of instructions per transfer (PL),

(c) No of *cache misses per instruction* in the highest level cache (MPI), and

(d) CPU utilization.

(e) Achieved throughput.

Not surprisingly, SCTP can achieve approximately the same throughput as TCP. SCTP send, however, SCTP is 2.1X processing intensive than TCP send in terms of CPU utilization. The CPI, PL and MPI numbers shed further light on the nature of this inefficiency. SCTP is actually executing 3.7X as many instructions than TCP; however, these instructions are, on the average, simpler and have a much better caching behavior so that the overall CPI is only 60%. This is a tell-tale sign of a lot of data manipulation. In fact, much of the additional SCTP path length derives from inefficient implementation of data chunking, chunk bundling, maintaining several linked data structures, SACK processing, etc. On the receive end, STCP is somewhat more efficient (1.7X of TCP). This is because SCTP receive requires significantly less work beyond the basic TCP.

TABLE 1: 8 KB transfers, 1 CPU, 1 connection

Case	Total CPI	path length	2ndL MPI	CPU util	Tput (Mb/s)
TCP Send (w/o TSO, w/o Chksum)	4.93	16607	0.0285	41.6	929
SCTP Send (w/o TSO, w/o Chksum)	2.94	60706	0.0176	89.0	916
TCP Receive (w/o TSO, w/o Chksum)	3.89	20590	0.0543	40.3	917
SCTP Receive (w/o TSO, w/o Chksum)	3.92	35920	0.0334	69.4	904

The 8 KB data transfer case is presented here to illustrate performance for applications such as iSCSI where the data transfer sizes are fairly large and operations such as memory to memory copy substantially impact the performance. It is also instructive to consider performance w/ small transfer sizes (e.g., 64 bytes). In this case, packet processing overwhelms the CPU for both protocols (as expected); therefore, the

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

key measure of efficiency is the throughput rather than the CPU utilization. Again, TCP was found more efficient than SCTP, however the differences are very much dependent on receive window size and data coalescing as discussed below.

Since TCP is a byte-stream oriented protocol, it can accumulate one MTU worth of data before making a driver call to prepare the IP datagram and send. This is, in fact, the default TCP behavior. However, if the data is not arriving from the application layer as a continuous stream, this would introduce delays that may be undesirable. TCP provides a NO-DELAY option for this (turned off by default). SCTP, on the other hand, is message oriented and provides chunk bundling as the primary scheme for stuffing up an MTU. SCTP also provides a NO-DELAY option which is turned on by default. That is, by default, whenever the window allows a MTU to be sent, SCTP will build a packet from the available application messages instead of waiting for more to arrive.

Table 2: 64 B transfers, 1 CPU, 1 connection

Case	Tput (64 KB)	Tput (128 KB)
TCP Send (w/o TSO, w/o Chksum)	72	134
SCTP Send (w/o TSO, w/o Chksum)	66	100
TCP Receive (w/o TSO, w/o Chksum)	76	276
SCTP Receive (w/o TSO, w/o Chksum)	74	223

As expected, with default settings, TCP appears to perform significantly better than SCTP. However, in the data center context, latencies matter much more than filling the pipe, and keeping the NO-DELAY on is the right choice. The SCTP chunk bundling should still be enabled since it only works with available data. In this case, surprisingly, SCTP performance is only somewhat worse than TCP as shown by second column in Table 2. However, this is the case with the default window size of 64KB. With a window size of 128KB, TCP outperforms SCTP because of fewer data structure manipulations.

3.2 Multi-streaming Performance

One of the justifications for providing multi-streaming in SCTP has been the lightweight nature of streams as compared to associations. Indeed, some of the crucial transport functionality in SCTP (e.g., flow and congestion control) is common to all streams and thus more easily implemented than if it were stream specific. Consequently, one would expect better multi-streaming performance than multi-association performance for the same CPU utilization.

Since the streams of a single association cannot be split over multiple NICs, we considered the scenario of a single NIC with one connection (or association). However, to avoid the single NIC becoming the bottleneck, we changed the transfer size from the usual 8 KB down to 1.28 KB. Note that no

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

segmentation will take place with this transfer size. We also used a DP (dual processor) configuration here in order to ensure that the CPU does not become a bottleneck.

Table 3 shows the results. Again, for the single stream case, although both SCTP and TCP are able to achieve approximately the same throughput, SCTP is even less efficient than for the single connection case. This indicates some deficiencies in TCB structure and handling for SCTP which we confirmed in our experiments as well.

Now, with SCTP alone, the overall throughput with 2 streams over one association is about 28% *less* than that for 2 associations.[2] However, the CPU utilization for the 2 stream case is also about 28% lower than for the 2 association case. These observations are approximately true for both sends and receives. So, in effect, *the streams are about the same weight as associations; furthermore, they are also unable to drive the CPU to 100% utilization*. This smacks of a locking/synchronization issue.

On closer examination, it was found that the streams come up short both in terms of implementation *and in terms of protocol specification*. The implementation problem is that the sendmsg() implementation of LK-SCTP locks the socket at the beginning of the function & unlocks it when the message is delivered to the IP-Layer. The resulting lock contention limits stream throughput severely. This problem can be alleviated by a change in the TCB (transport control block) structure along with finer granularity locking. *A more serious issue is on the receive end – since the stream id is not known until the arriving SCTP has been processed and the chunks removed, there is little scope for simultaneous processing of both streams*. This is a key shortcoming of the stream feature and can only be fixed by encoding stream information in the common header, so that threads can start working on their target stream as early as possible.

TABLE 3: 1.28 KB transfers, 2 CPUs, all connections on the same NIC

Case	Total CPI	path length	2ndL MPI	CPU util	Tput (Mb/s)
TCP Send (2 conn)	6.68	4251	0.0456	40.9	896
SCTP Send (2 assoc w/ 1 stream)	5.74	11754	0.0438	100	888
SCTP Send (1 assoc w/ 2 streams)	6.18	10540	0.0495	72.5	635
TCP Receive (2 conn)	4.72	4739	0.0624	34.5	897
SCTP Receive (2 assoc w/ 1 stream)	6.2	7298	0.0579	64.5	890
SCTP Receive(1 assoc w/ 2 streams)	5.2	7584	0.0461	44	641

3.3 SCTP Performance Enhancements

In this section, we examine SCTP from the perspective of efficiency and identify several areas for performance enhancements. The discussion here includes both the implementation (via LK-SCTP) as well

[2] Given the zero error conditions and small RTT, the throughput should not be constrained by the usual AIMD behavior – if it were, two associate will be expected to behave better than one. The differences here are purely related to protocol overhead and locking/synchronization issues.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

as the protocol itself. We also show the performance improvements from the enhancements that have been carried out so far.

3.3.1 Implementation Issues

Figure 1 shows LK-SCTP's approach to chunking and chunk bundling. It maintains three data structures to manage the chunk list. The *data message* contains the list of chunks & it is dynamically allocated for each user message. It is reference counted, i.e. it is freed only when all the chunks belong to it are acknowledged by the remote endpoint. Each chunk is managed via two other data structures. The first one contains the actual chunk data along with the chunk header and the other contains pointers to chunk buffers & some miscellaneous data. Both of these structures are dynamically allocated & freed by LK-SCTP. In many instances, LK-SCTP repeatedly initializes local variables with values & then copies them to the final destination. The chunk itself is processed by many routines before it is copied to the final buffer. The implementation also maintains many small data structures/queues and does frequent allocation & de-allocation of memory.

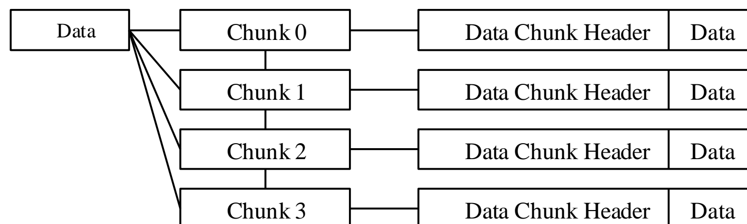

Fig. 1: Data Structures representing user message

The net result of the above scheme is a total of 3 memory to memory (M2M) copies before the data appears on the wire. These copies include (1) Retrieval of data from the user buffer and placement in the *data message* structure, (2) Bundling of chunks into a MTU sized packet (and passing control to NIC driver), and (3) DMA of data into the NIC buffers for transmission on the wire. These copies occur irrespective of prevailing conditions (e.g., even for one chunk per message and one chunk per SCTP packet).

It is clear from the above that the LK-SCTP implementation can be speeded up substantially using the following key techniques: (a) Avoid dynamic memory allocation/deallocation in favor of pre-allocation or use of ring buffers, (b) Avoid chunk bundling only when appropriate, and (c) Cut down on M2M copies for large messages. Although the issues surrounding memory copies are well understood, producing a true 0-copy SCTP implementation will require very substantial changes to both SCTP and application interface – not to mention the necessary HW support. The optimized implementation uses pre-allocation as far as possible and reduces number of copies as follows.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

- During fragmentation of a large user message, we decide whether a given data chunk can be bundled together with other chunks or not. If not, we prepare a packet with one chunk only & designate this chunk as a full chunk.
- Steps (1) and (2) above are combined together in order to eliminate 1 copy for messages larger than 512 bytes. For smaller message, the default 2-copy path actually turns out to be shorter. The current threshold of 512 bytes was arrived at via experimentation, and may shift as further optimizations are made.

3.3.2 SCTP Specification Issues

According to the SCTP RFC (2960), an acknowledgement should be generated for at least every second packet (not every second data chunk) received, and should be generated within 200 ms of the arrival of any unacknowledged data chunk. We looked at the packet trace using the Ethereal tool & found many SACK packets. In particular, it was found that *SCTP sends 2 SACKs instead of just one*: once when the packet is received (so that the sender can advance its window “cwnd”) and then again when the packet is delivered to the application (so that the sender would know about the updated receiver’s window “rwnd”). In effect, this amounts to one SACK per packet! Note that this is a problem with SCTP, not just with the implementation.

SACK is a very small control packet & SCTP incurs significant amount of processing on them at both the sender & receiver ends. Although conceptually SACK processing overhead should be similar for TCP and SCTP, the chunking and multi-streaming features plus an immature implementation make it a lot more expensive in SCTP. Also, since SCTP lacks ordinary acks (which are much lighter weight), the frequency of SACKs in SCTP becomes too high and contributes to very substantial overhead. After some experimentation, we set the frequency of SACKs to once per six packets and ensured that it is sent either on data delivery, or if delivery is not possible due to missing packets, on data receive.

SCTP defines the maximum burst size (MBS) which controls the maximum number of data chunks sent on any given stream (if cwnd allows it) before pausing for an acknowledgement. The LK-SCTP implementation uses MBS=4 which means that only 6000 bytes of data can be sent in one shot if cwnd allows. The initial value of the cwnd parameter is 3000 bytes. The implementation resets the value of cwnd to MBS*1500 + flight_size (data in flight on a given transport), if the cwnd value exceeds this amount. This condition severely limits the amount of data sender can send. While the intention of MBS is clearly rate control, specifying it as a constant protocol parameter or embedding a complex dynamic algorithm in the transport layer is not a desirable approach.

An important issue with respect to performance is the size and layout of connection descriptors, popularly known as transmission control block (TCB). In case of TCP, the TCB size is typically around 512 bytes. For SCTP, we found that the size of the association structure is an order of magnitude bigger at around 5KB. Since an association can have multiple end-points due to multihoming feature, the per endpoint data structure to handle is 200 bytes long for local IP addresses and 180 bytes for remote.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

Although these are mostly implementation matters, the specification also plays a big role in an outlandish TCB size. *Large TCB sizes are undesirable both in terms of processing complexity and in terms of caching efficiency.*

The multistreaming issue of SCTP was already discussed in section 3.2. There are a few others relating to features not explored in this paper (e.g., multistreaming) and hence are not discussed here.

4. Performance Impact of Optimizations

In this section, we show comparative performance of SCTP with and w/o optimizations against TCP. It should be noted that the current optimization work is an ongoing process and further performance improvements should be achievable. In particular, *we believe that a well optimized implementation along with certain minor protocol changes should be fairly close to TCP in performance.* Actually, for certain applications that exploit some of SCTP's features (e.g., RDMA or multi-streamed transfers), SCTP should be able to provide even better performance than TCP.

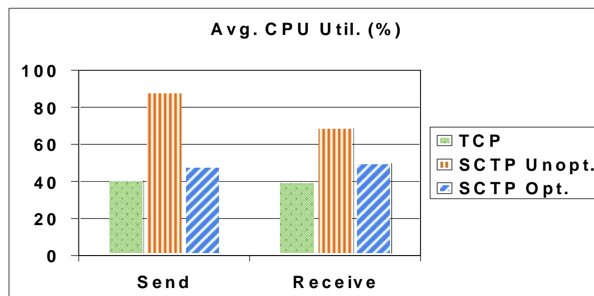

Fig. 2. Average CPU utilization for 8 KB transfers

Let us start with 8KB data transfer cases. Figure 2 compares SCTP CPU utilization against TCP w/ and w/o optimization. In each case, the achieved throughput was about the same (~930 Mb/sec); therefore, the CPU utilization adequately reflects the relative efficiency of the 3 cases. It is seen that the optimizations drop CPU utilization from 2.1x to 1.16x that for SCTP, *which is a very dramatic performance improvement considering that there have been no extensive code rewrites*. SCTP receive utilization also improves very substantially, from 1.7x down to about 1.25x.

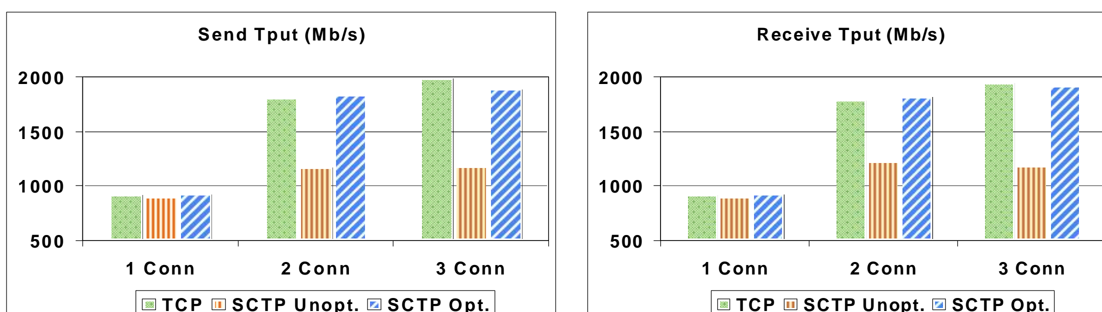

Fig. 3. Throughput scaling with multiple connections

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

Figure 3 shows SCTP scaling as a function of number of connections. Each new connection is carried over a separate GB NIC in order to ensure that the throughput is not being limited by the NIC. It is seen that the original SCTP scales very poorly with number of connections; however, the optimizations bring it significantly closer to TCP scaling. With 3 connections, the CPU becomes a bottleneck for both TCP and SCTP; hence the scaling from 2 to 3 connections is rather poor for both protocols.

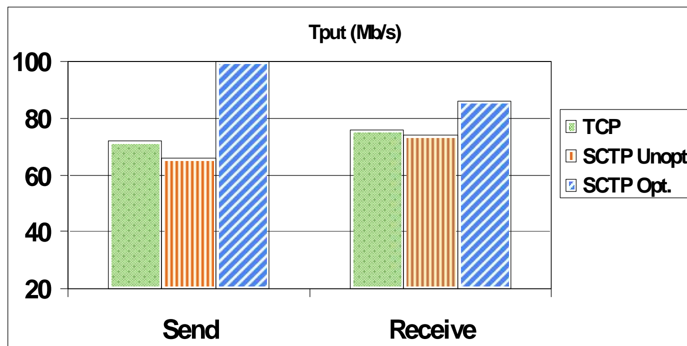

Fig. 4. Tput comparison with 64Bytes packets

Figure 4 shows the SCTP throughput for small (64B) packets. As stated earlier, the performance in this case depends on the NO-DELAY option and the receive window size. The results shown here are for NO-DELAY on and receive window size of 64 KB. In this case, SCTP and TCP throughputs were already comparable; the throughput improves it further, but not by much. In particular, optimized SCTP send throughput is actually higher than that for TCP. However, we note that with a receive window size of 128 KB, TCP continues to outperform optimized SCTP.

5. Performance under Errors

Figure 5 illustrates TCP and SCTP performance under regularly spaced packet losses. The case shown is the achieved throughput for a GB NIC for 8KB data transfers. It is seen that unoptimized SCTP actually performs better than TCP. This is perhaps due to several minor differences in the congestion control algorithm used by the two protocols [2] and other system issues. (These results do not conflict with earlier results showing poor SCTP performance – recall that with one connection both TCP and SCTP are able to achieve near-wire speed, except that the CPU utilization is much higher for SCTP.) With optimizations, the performance is somewhat better at low drop rates but degrades significantly at high rates. This is to be expected since *a reduction in SACK frequency is detrimental to throughput performance at high drop rates, but is desirable at lower drop rates.*

Figure 6 illustrates TCP and SCTP performance with the same packet loss probability as in Fig. 5 except that (a) the time between successive losses is now exponentially distributed, and (b) the SACK frequency is same in all cases. At very low drop probability (0.001), there results are similar, as expected.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

At drop probability of 1%, the achievable throughput is significantly higher across the board. This is perhaps a result of the fact that with exponential inter-drop times, drops are more clustered and thus separated by periods with very small loss rate (and hence high throughput) periods. The optimized SCTP stack yields slightly better performance than unoptimized stack here because the SACK frequency is the same for both in both cases (once every 2 packets). At very high drop rates, TCP somehow manages to perform significantly better than SCTP. We are not quite sure of why such a large discrepancy should exist – it could well have something to do with implementation details of the two protocol stacks.[3] Nevertheless, the optimized SCTP performance is still slightly better than unoptimized case because of same SACK frequency.

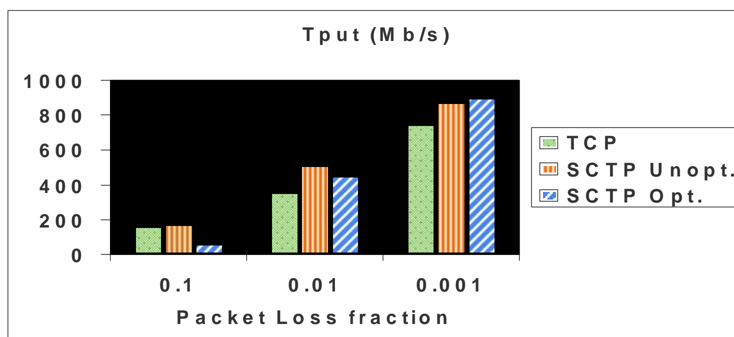

Fig. 5. Throughput under deterministic Packet loss

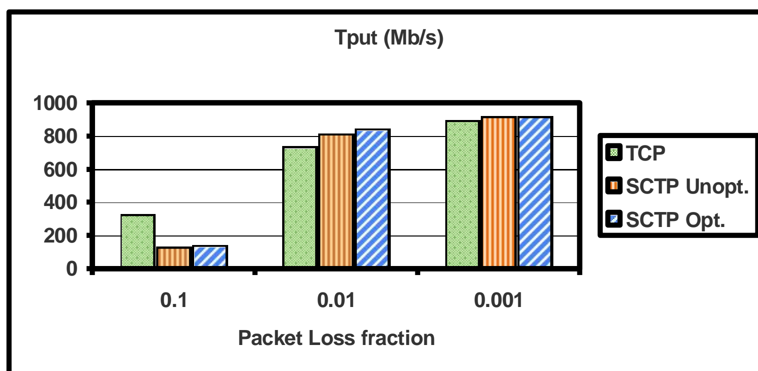

Fig. 6. Throughput under uniform packet loss & same SACK frequency

The net result, as expected, is that SACKs are desirable with medium to high drop rates, but may be a drag on performance at negligible drop rates. The primary attraction of SACKs is reporting of individual gaps so that only the missing packets are retransmitted. If SACKs are sent for every two packets, it will report at most two gaps, and usually no more than one gap. Yet, the SACK structure is designed for arbitrary lists of gaps and only leads to overhead. In a data center environment, the gap reporting is even

[3] A reminder to the reader: none of the results in this paper are from a simulation; they are from actual implementations and thus O/S and other issues could well affect performance in not so clear ways.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

less efficient since even a single gap will appear very rarely. A reduced SACK frequency is an obvious optimization for data centers. An even more significant point is that SACKs are actually undesirable in a data center. Firstly, the round-trip times (RTTs) within a data center being small, the amount of extra retransmissions w/o SACK can be made rather small. For example, consider a TCP-Reno like mechanism where 3 duplicate acks are considered as a signal for packet loss. In this case, with a RTT of 20 us, at 10 Gb/sec, the pipe has only about 25 KB worth of data which must be retransmitted. Second, without SACKs, there will be no need to allocate any buffers to hold undelivered data on the receive side. This could be a very substantial cost savings at high data rates. It is also important to note that SACKed data cannot be removed from the retransmission buffer according to the specification, so SACK does not help in reducing send buffer size.

6. Discussion and Future Work

In this paper we examined SCTP from a data center perspective. We exposed several essential differences between the WAN and data center environments and exhibited several issues with SCTP as applied to data centers. The important issues in this regard include both implementation as well as the protocol. On the implementation side, it is essential to reduce number of memory to memory copies, simplify chunking data structures, forego chunk bundling for larger messages, reduce SACK overhead, simplify TCB structure and implement finer grain TCB locking mechanisms. On the protocol side, the major findings include re-architecting of streaming feature to maximize parallelism and to provide a TCP like simple embedded ack procedure with SACK made optional. There are a few other protocol and implementation issues that we have found, which are under study currently. These relate to application centric coordinated window flow control, exploiting topological information within a data center for multihomed associations, and enhancing the protocol for multicast capabilities focused towards data center usage scenarios. In addition, there are a few other issues already addressed by others, e.g., need for stream priorities [3] and simultaneous transmission across multi-homed interfaces [4,13].

References

1. B. Benton, “Infiniband’s superiority over Myrinet and QsNet for high performance computing”, whitepaper at www.FabricNetworks.com
2. R. Brennan and T. Curran, “SCTP Congestion Control: Initial Simulation Studies,” Proc. 17th Int’l Tele-traffic Congress, Elsevier Science, 2001; www.eeng.dcu.ie/~opnet/.
3. G.J. Heinz & P.D. Amer, “Priorities in stream transmission control protocol multistreaming”, SCI 2004.
4. J.R. Iyengar, K.C. Shah, et. Al., “Concurrent Multipath Transfer using SCTP multihoming”, Proc. of SPECTS 2004.
5. K. Kant, “TCP offload performance for front-end servers”, in proc. of GLOBECOM 2003, Dec 2003, San Francisco, CA.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.

6. I. Khalifa & L. Trajkovic, "An overview & comparison of analytic TCP models", www.ensc.sfu.ca/~ljilja/cnl/presentations/inas/iscas2004/slides.pdf
7. S. Ladha and Paul D. Amer, "Improving File Transfers Using SCTP Multistreaming" Protocol Engineering Lab, CIS Department, University of Delaware
8. R. Recio, et. al. "An RDMA Protocol Specification", draft-ietf-rddp-rdmap-02 (work in progress), September 2004.
9. G. Regnier, S. Makineni, et. al., "TCP onloading for data center servers", Special issue of IEEE Computer on Internet data centers, Nov 2004.
10. B. Sikdar, S. Kalyanaraman, K.D. Vastola, "Analytic models and comparative study of latency and steady state throughput of TCP Tahoe, Reno and SACK", Proc. of Globecom 2001.
11. R. Stewart, "Stream Control Transmission Protocol (SCTP) Remote Direct Memory Access (RDMA) Direct Data Placement (DDP) Adaptation", draft-ietf-rddp-sctp-01.txt, Sept 2004.
12. R. Stewart and Q. Xie, Stream Control Transmission Protocol (SCTP): A Reference Guide. Addison Wesley, New York, 2001.
13. G. De Marco, M. Longo, F. Postiglione, "On some open issues in load sharing in SCTP", Proceedings of CCCT 2003, Orlando, FL.
14. J. Martin, A. Nilsson and I. Rhee, ``Delay based congestion avoidance in TCP'', IEEE/ACM transactions on networking, vol 11, no 3, pp356-369, June 2003.

Easy PDF Creator is professional software to create PDF. If you wish to remove this line, buy it now.