

Using CMT in SCTP-Based MPI to Exploit Multiple Interfaces in Cluster Nodes

Brad Penoff¹, Mike Tsai¹, Janardhan Iyengar², and Alan Wagner¹

¹ University of British Columbia, Vancouver, BC, Canada
{penoff,myct,wagner}@cs.ubc.ca

² Computer Science, Connecticut College, New London, CT, USA
iyengar@conncoll.edu

Abstract. Many existing clusters use inexpensive Gigabit Ethernet and often have multiple interface cards to improve bandwidth and enhance fault tolerance. We investigate the use of Concurrent Multipath Transfer (CMT), an extension to the Stream Control Transmission Protocol (SCTP), to take advantage of multiple network interfaces for use with MPI programs. We evaluate the performance of our system with micro-benchmarks and MPI collective routines. We also compare our method, which employs CMT at the transport layer in the operating system kernel, to existing systems that support multi-railing in the middleware. We discuss performance with respect to bandwidth, latency, congestion control and fault tolerance.

1 Introduction

Clusters are widely used in high performance computing, often with more money invested in processors and memory than in the network. As a result, many clusters use inexpensive commodity Gigabit Ethernet (GbE) rather than relatively expensive fast interconnects¹. The low cost of GbE cards and switches makes it feasible to equip cluster nodes with more than one interface cards, each interface configured to be on a separate network. Multiple network connections in such an environment can be used to improve performance, especially for parallel jobs which may require more high-bandwidth, low-latency communication and higher reliability than most serial jobs.

We investigate the use of Concurrent Multipath Transfer (CMT) [2,3], an extension to the Stream Control Transmission Protocol (SCTP) [4,5,6], to take advantage of multiple network connections. SCTP is an IETF-standardized, reliable, and TCP-friendly transport protocol, that was initially proposed for telephony signaling applications. It has since evolved into a more general transport protocol for applications that have traditionally used TCP for their requirements of reliability and ordered delivery. SCTP uses the standard sockets library and

¹ In the latest rankings of the Top 500 supercomputers, 211 of the systems use GbE as the internal network [1].

an application can interface with SCTP just as with TCP. We have recently released SCTP-based middleware in MPICH2 (`ch3:sctp`) that enables MPI-based parallel programs to use SCTP’s features [7].

SCTP introduces several new features at the transport layer such as *multihoming* and *multistreaming* that make it interesting for use in clusters. Of particular relevance to this research, SCTP’s multihoming feature allows multiple network interfaces to be used within a single transport layer *association*—a TCP connection can bind to only one interface at each endpoint. In standard SCTP, only a single path is used for data transfers; the alternative paths are only used as backups if the primary path fails. CMT is a recent proposed extension to SCTP that aggregates, at the transport layer, available bandwidth of multiple independent end-to-end paths between such multihomed hosts [3].

Our work is related to projects such as NewMadeleine [8], MVAPICH [9], and Open MPI [10] that support multihomed or multi-railed systems in the middleware. However, these projects focus on low latency network architectures such as Infiniband, and not just IP networks. NewMadeleine and Open MPI both support heterogeneous networks; CMT can be used to combine all IP interfaces, which in turn can be used in combination with these heterogeneous network solutions. MuniCluster [11] and RI2N/UDP [12] support multi-railing for IP networks at the application layer, and use UDP at the transport layer. Our approach is significantly different in that CMT supports multihoming (or multi-railing) at the transport layer², as part of the kernel, rather than at the application layer in user-space.

The remainder of the paper is organized as follows. In Section 2, we provide a brief overview of relevant SCTP and CMT features. We then discuss our approach, in comparison with other projects, with respect to configuration, scheduling, congestion control, and fault tolerance. In Section 3, we describe our experiments to evaluate CMT in a cluster for MPI programs, and discuss the performance results. We give our conclusions in Section 4.

2 Issues Related to Using SCTP and CMT in a Cluster

An example of a generic cluster configuration that can benefit from our approach is shown in Figure 1. This cluster has two nodes, each equipped with three Ethernet interfaces. On a node, each interface is connected to one of three IP networks through an Ethernet switch—two interfaces are connected to internal cluster networks, and the third interface is connected to a public and/or management network. In such a configuration, the internal networks can be used in parallel to improve cluster communication in several ways. A system that uses these separate networks simultaneously must also account for possibly varying and different bandwidths and/or delays of the different networks.

In this section, we discuss how SCTP and CMT enable this communication. To compare our approach to existing approaches, we discuss points that any

² We also investigated channel bonding, but did not observe any performance advantage in our setup.

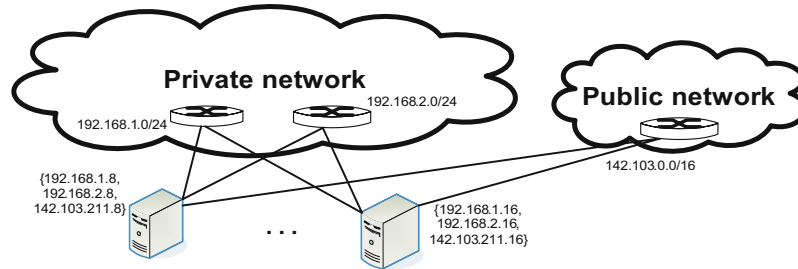


Fig. 1. Example Configuration

multi-railing solution must consider: configuration, scheduling, congestion control and fault tolerance.

2.1 Configuration

An SCTP association between two endpoints can bind to multiple IP addresses (i.e., interfaces) and a port at each endpoint. For example, in Figure 1, it is possible to bind ($\{192.168.1.8, 192.168.2.8\};5000$) at one endpoint of an association to ($\{192.168.1.16, 192.168.2.16\};4321$) at the other endpoint. By default, SCTP binds to all the interfaces at each endpoint, but the special `sctp_bindx()` API call can be used to restrict the association to a subset of the interfaces at each endpoint (i.e., the private network ones). We provide this configuration information to the MPI middleware in a configuration file that is loaded at run-time. This is the only interface configuration information needed by the system. Unlike related projects, the bandwidth and latency characteristics of the network are not provided statically but determined dynamically and adaptively as part of the transport protocol (see Section 2.3).

A final issue with respect to configuration is routing. CMT and SCTP determine only which destination address to send data to; at the transport layer, CMT and SCTP do not determine routes through the network. Routing is done by the routing component of the network layer using the kernel's routing/forwarding tables, and this component determines the local interface to use to get to a specified destination. For example, in Figure 1, the routing table at the host with the public address 142.103.211.8 must be configured so that the local outgoing interface 192.168.1.8 (and neither 192.168.2.8 nor 142.103.211.8) is used when the destination address for an outgoing packet is 192.168.1.16. This kind of routing configuration is straightforward and well understood by system administrators, who perform similar nominal configurations for all IP hosts on the network.

2.2 Scheduling

In our SCTP-based MPI middleware, MPI messages are split into one or more SCTP data *chunks* at the transport layer—data *chunks* are the smallest atomic schedulable unit in SCTP—which are then packed into a transport layer segment

and sent out the primary link, in the case of standard SCTP. However, CMT maintains a congestion window (`cwnd`) for each network path and *distributes* data segments across multiple paths using the corresponding `cwnds`, which control and reflect the currently allowed sending rate on their corresponding paths. CMT thus distributes load according to the network paths' bandwidths—a network path with higher available bandwidth carries more load.

CMT's scheduling results in a greedy scheduling of data segments to interfaces, which is similar to previous approaches. The main difference is that because CMT schedules data chunks, both short and long MPI messages are sent concurrently over multiple paths, which gives more flexibility in scheduling at a finer granularity than message-level granularity. A disadvantage of our approach is that it is difficult within CMT's scope to identify MPI message boundaries, making it difficult to schedule messages based on message size. For instance, CMT may have difficulty scheduling a small MPI message on the shorter delay path to minimize latency.

2.3 Congestion Control

Most of the existing work on multi-railing assumes dedicated resources on well-provisioned networks and do not consider the effect of congestion on network performance, using specialized protocols that lack congestion control. However, congestion is more likely in clusters with commodity Ethernet switches, especially for the bursty traffic in MPI programs. The problem of congestion is exacerbated in larger networks with multiple layers of switches and mixes of parallel and sequential jobs executing concurrently. In the face of congestion, senders must throttle their sending rate and avoid persistent network degradation.

SCTP, like TCP, uses congestion control mechanisms to minimize loss and retransmissions and to ensure that application flows are given a fair share of the network bandwidth. SCTP dynamically tracks a `cwnd` per destination address, which determines the current sending rate of the sender (recall that the source address is chosen by the routing component of the network layer). CMT extends SCTP's mechanisms to enable correct congestion control when sending data concurrently over multiple paths. Our system thus differs significantly from the previous approaches in that CMT is dynamic, not static, and will correctly adjust its sending rate to changing traffic characteristics in the network. CMT congestion control mechanisms ensure that (i) when a new flow enters the network, existing flows curb their `cwnds` so that each flow gets a fair share of the network bandwidth, (ii) when flows leave the network, continuing flows adapt and use the spare network bandwidth, and (iii) CMT can be used over network paths with differing bandwidths/latencies.

2.4 Fault Tolerance

Fault tolerance is an important issue in commodity cluster environments, where network failure due to switches or software (mis)configuration does occur. SCTP has a mechanism for automatic failover in the case of link or path failure. Similar

mechanisms have been implemented in MPI middleware [13], but these are unnecessary for SCTP-based middleware. SCTP uses special control chunks which act as *heartbeat* probes to test network reachability. Heartbeat frequency is user-controllable, and can be tuned according to expectations for a given network. However, at the currently recommended setting [14], it takes approximately 15 seconds to failover—a long time for an MPI application. Since CMT sends data to all destinations concurrently, a CMT sender has more information about *all* network paths, and can leverage this information to detect network failures sooner. CMT is thus able to reduce network path failure detection time to one second [15]. This improvement with CMT is significant, and future research will investigate if this time can be further reduced for cluster environments.

3 Performance

The goal of our evaluation was to see that CMT could take advantage of the available bandwidth, scheduling within the transport protocol. The focus of the evaluation was on bandwidth intensive MPI benchmarks. The hope was to demonstrate results comparable to MPI implementations that schedule within the middleware.

Test Setup. The potential benefits of using CMT with MPI were evaluated on four³ 3.2 GHz Pentium-4 processors in an IBM eServer x306 cluster, where each node had three GbE interfaces. The two private interfaces were on separate VLANs connected to a common Baystack 5510 48T switch, and the third (public) interface was connected to a separate switch, similar to the setup shown in Figure 1. We tested with an MTU size of 1500 bytes and with jumbo frames of size 9000 bytes.

For the tests with SCTP and CMT we used MPICH2 v1.0.5, which contains our SCTP-based MPI channel (`ch3:sctp`). We tested MPICH2 with and without CMT enabled and compared it to TCP from the same release of MPICH2 (`ch3:sock`). We also tested Open MPI v1.2.1, which supports message striping across multiple IP interfaces [16]. Open MPI was ran with one and two TCP Byte-Transfer-Layer (BTL) modules to provide a comparison between using one interface versus two interfaces⁴.

All SCTP tests were based on an SCTP-capable stack⁵ in FreeBSD 6.2. For all tests we used a socket buffer size of 233 Kbytes, the default size used by `ch3:sctp`, and we set the MPI rendezvous threshold for MPI to 64 Kbytes, matching `ch3:sctp` and Open MPI's default.

³ We did not have the opportunity to test our implementation in a large cluster. Since SCTP was designed as a standard TCP-like transport protocol, we expect its scalability to be similar to that of TCP.

⁴ Our network paths were not identical so Open MPI had the network characteristics set in the `mca-params.conf` file.

⁵ A patched version of the stack (<http://www.sctp.org>) was used. SCTP will be a standard part of FreeBSD 7— as a kernel option.

Microbenchmark Performance. The most important test was to determine whether MPI with CMT was able to exploit the available bandwidth from multiple interfaces. We used the OSU MPI microbenchmarks [17] to test bandwidth and latency with varying message sizes. We tested with MTUs of 1500 and 9000 bytes and found that using a 9000 byte MTU improved bandwidth utilization for both TCP and SCTP. Figure 2 reports our results with MTU 9000.

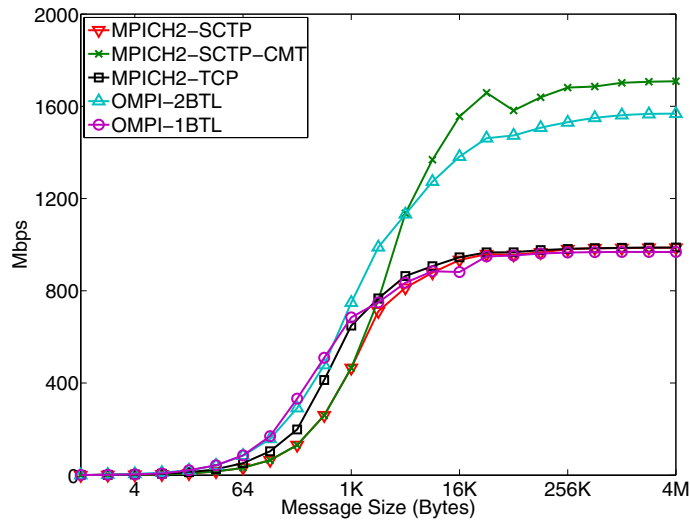


Fig. 2. OSU Bandwidth Test comparing MPICH2 with and without CMT and Open MPI with one and two TCP BTLs

As shown in Figure 2, MPICH2-`ch3:sctp` with CMT starts to take advantage of the additional bandwidth when the message size exceeds 2 Kbytes. At a message size of 1 Mbyte, MPICH2-`ch3:sctp` with CMT is almost achieving twice the bandwidth (1.7 versus 0.98 Gbps) in comparison to using one interface, and it is outperforming Open MPI (1.56 Gbps) which is scheduling messages in user-space across two interfaces within its middleware. All single path configurations are approximately the same, with a maximum of 0.98 Gbps; however for configurations using two networks, Open MPI is the best for message sizes 512 to 2048 bytes but once message size exceeds 4 Kbytes MPICH2-`ch3:sctp` with CMT is the best performing.

In the case of the OSU latency benchmark (figure not shown), `ch3:sctp` without CMT was between `ch3:sock`, which had the lowest latency at $77\mu\text{sec}$ (for zero-byte message), and Open MPI (one TCP BTL), which was consistently 20% higher than `ch3:sock` (for message size up to 1 Kbytes). Large message latency was more or less the same across all of these middleware configurations that used a single interface. When using two interfaces, CMT increased message latency by approximately 2.5% over `ch3:sctp` for small messages, but this increase was

smaller than the 10% increase we found with Open MPI with two TCP BTLs compared to Open MPI with one TCP BTL. We note that CMT is currently optimized for bandwidth only, and we have not yet explored the possibility of optimizing CMT for both bandwidth and latency.

The third component of performance that we investigated was CPU utilization. For any transport protocol there are trade-offs between bandwidth and the associated CPU overhead due to protocol processing, interrupts, and system calls. CPU overhead is important since it reduces the CPU cycles available to the application and reduces the potential for overlapping computation with communication. We used `iperf`, a standard Unix tool for testing bandwidth, together with `iostat` to investigate CPU utilization differences. CPU utilization for TCP and SCTP with an MTU of 1500 bytes were similar. However, with CMT, the CPU was a limiting factor in our ability to fully utilize the bandwidth of both GbE interfaces. For an MTU of 9000 bytes, the CPU was still saturated, yet CMT was able to achieve bandwidths similar to those obtained from the OSU bandwidth benchmark because more data could be processed with each interrupt. We conclude from our investigation that there is an added cost to protocol processing for CMT. Part of this cost may be due to the cost of processing *selective acknowledgement (SACK)* blocks at the transport layer receiver, which is higher in CMT. SACK processing overhead is actively being considered by developers in the SCTP community, and continued work should reduce this overhead.

Collective Benchmarks. The benchmarks in the previous section tested the effect of CMT and MPI point-to-point messaging on raw communication performance. We were interested in testing CMT's effect on more complex MPI calls like the collectives since they are also commonly used in MPI programs. The Intel MPI Benchmark suite 2.3 was used in our experiments. We tested several of the collective routines, but for brevity, we focus on the `MPI_Alltoall` four process results here since the results obtained for the other collectives showed similar benefits.

The results are summarized in Figure 3; only the multi-rail MPI results are shown (i.e., `ch3:sctp` with CMT and Open MPI with two TCP BTLs). Latency in the Alltoall test is always lower with CMT, except for message sizes of 64 Kbytes and 128 Kbytes. For large messages, CMT can have a considerable advantage over two TCP BTLs (e.g. 42% for 1 MByte messages). Although not pictured here, we note that without multi-railing, MPICH2 (both `ch3:sctp` and `ch3:sock`) is generally within 10-15% of Open MPI with one BTL. Multi-railing starts to show an advantage for message sizes greater than 8 Kbytes. We attribute the performance improvement of `MPICH2-ch3:sctp` with CMT over Open MPI with two TCP BTLs (up to 42%) to the advantages of scheduling data in the transport layer rather than in the middleware. We are currently implementing an SCTP BTL module for Open MPI that will be able to use CMT as an alternative for striping, and we hope to obtain similar performance improvements in Open MPI for multiple interfaces.

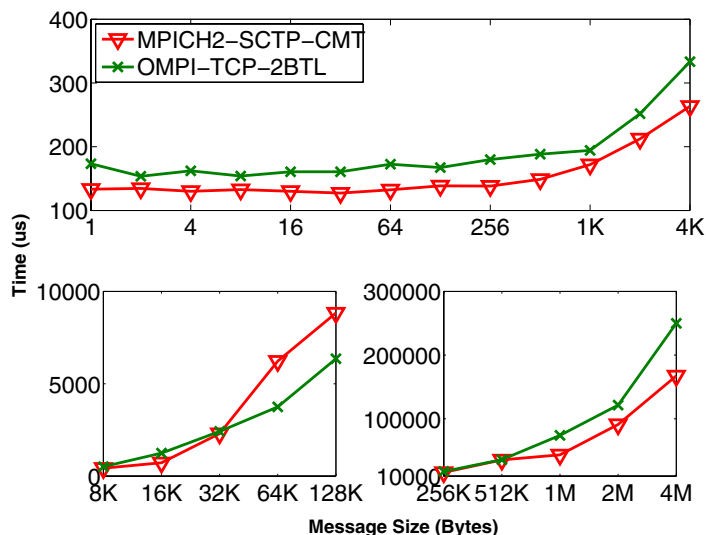


Fig. 3. Pallas Alltoall n=4 Benchmark MTU 1500

4 Conclusions

We have investigated a novel approach that uses multiple network interfaces on cluster nodes to improve the performance of MPI programs. Unlike systems that support multi-railing in the middleware, our approach uses CMT, an extension to SCTP, that supports multi-railing at the transport layer in the kernel. CMT dynamically adapts to changing network conditions, and therefore simplifies the network configuration process. We showed that CMT is able to take advantage of the additional bandwidth of an additional interface, and compares well with Open MPI's load balancing scheme. Experiments with MPI collectives demonstrate the possible advantages in scheduling messages at the transport layer (using CMT) versus scheduling messages in the middleware (using Open MPI).

CMT shows promise for bandwidth intensive applications. Future work will investigate optimization of CMT scheduling for lower latency and CMT refinements for fault tolerance. We expect that our research will lead to an easy-to-use system for efficiently using multiple interfaces in clusters with low-cost commodity networking.

Acknowledgments. We wish to thank Randall Stewart and Michael Tüxen for their extensive help with the experiments.

References

1. University of Mannheim, University of Tennessee, NERSC/LBNL: Top 500 Computer Sites (2007), <http://www.top500.org/>
2. Iyengar, J.: End-to-end Concurrent Multipath Transfer Using Transport Layer Multihoming. PhD thesis, Computer Science Dept. University of Delaware (2006)

3. Iyengar, J., Amer, P., Stewart, R.: Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths. *IEEE/ACM Transactions on Networking* 14(5), 951–964 (2006)
4. Stewart, R.R., Xie, Q.: *Stream Control Transmission Protocol (SCTP): A Reference Guide*. Addison-Wesley Longman Publishing Co. Inc. Reading (2002)
5. Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I.M.K., Zhang, L., Paxson, V.: *The Stream Control Transmission Protocol (SCTP)* (2000) Available from <http://www.ietf.org/rfc/rfc2960.txt>
6. Stewart, R., Arias-Rodriguez, I., Poon, K., Caro, A., Tuexen, M.: *Stream Control Transmission Protocol (SCTP) Specification Errata and Issues* (2006), Available from <http://www.ietf.org/rfc/rfc4460.txt>
7. Kamal, H., Penoff, B., Wagner, A.: SCTP versus TCP for MPI. In: *Supercomputing '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, IEEE Computer Society, Washington, DC, USA (2005)
8. Aumage, O., Brunet, E., Mercier, G., Namyst, R.: High-Performance Multi-Rail Support with the NewMadeleine Communication Library. In: *The Sixteenth International Heterogeneity in Computing Workshop (HCW 2007)*, workshop held in conjunction with IPDPS 2007 (2007)
9. Liu, J., Vishnu, A., Panda, D.K.: Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation. In: *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, p. 33. IEEE Computer Society, Washington, DC, USA (2004)
10. Gabriel, E., et al.: Open MPI: Goals, concept and design of a next generation MPI implementation. In: *Proc. 11th EuroPVM/MPI*, Budapest, Hungary (2004)
11. Mohamed, N., Al-Jaroodi, J., Jiang, H., Swanson, D.R.: High-performance message striping over reliable transport protocols. *The Journal of Supercomputing* 38(3), 261–278 (2006)
12. Okamoto, T., Miura, S., Boku, T., Sato, M., Takahashi, D.: RI2N/UDP: High bandwidth and fault-tolerant network for PC-cluster based on multi-link Ethernet. In: *21st International Parallel and Distributed Processing Symposium (IPDPS'07)*, Long Beach, California, Workshop on Communication Architecture for Clusters (2007)
13. Vishnu, A., Gupta, P., Mamidala, A.R., Panda, D.K.: Scalable systems software - a software based approach for providing network fault tolerance in clusters with uDAPL interface: MPI level design and performance evaluation. In: *SC*, p. 85 (2006)
14. Caro, A.: *End-to-End Fault Tolerance Using Transport Layer Multihoming*. PhD thesis, Computer Science Dept. University of Delaware (2005)
15. Natarajan, P., Iyengar, J., Amer, P.D., Stewart, R.: Concurrent multipath transfer using transport layer multihoming: Performance under network failures. In: *MILCOM*, Washington, DC, USA (2006)
16. Woodall, T., et al.: Open MPI's TEG point-to-point communications methodology: Comparison to existing implementations. In: *Proc. 11th EuroPVM/MPI*, Budapest, Hungary, pp. 105–111 (2004)
17. Ohio State University: *OSU MPI Benchmarks* (2007) <http://mvapich.cse.ohio-state.edu>