

A High Performance Configurable Transport Protocol for Grid Computing

Ryan X. Wu, Andrew A. Chien
Computer Science and Engineering and
Center for Networked Systems
University of California, San Diego
{xwu, achien}@ucsd.edu

Matti A. Hiltunen, Richard D. Schlichting, Subhabrata Sen
AT&T Labs - Research
180 Park Avenue, Florham Park, NJ 07932
{hiltunen, rick, sen}@research.att.com

Abstract

Grid computing infrastructures and applications are increasingly diverse with networks ranging from very high bandwidth optical networks to wireless networks and applications ranging from remote visualization to sensor data collection. For such environments, standard transport protocols such as TCP and UDP are not always sufficient or optimal given their fixed set of properties and their lack of flexibility. As an alternative, we present H-CTP, a high-performance configurable transport protocol that can be used to build customized transport services for a wide range of Grid computing scenarios. H-CTP is based on an earlier configurable transport protocol called CTP, but with a collection of optimizations that meet the challenge of providing configurability while maintaining performance that meets the requirements of such demanding applications. This paper motivates the need for customizable transport in this area, presents the design of H-CTP, and gives results from performance studies that compare H-CTP with both CTP and TCP. These show, for example, that H-CTP is able to achieve throughput of over 900 Mbps across Gigabit links. Three diverse Grid scenarios are used as example applications and for the H-CTP/TCP comparisons: remote visualization, fast message passing, and sensor grids.

1. Introduction

The emerging high-performance Grid [1; 2] encompasses a wide range of network infrastructures and communication patterns, as well as different types of scientific applications. The combination of all these factors highlights several new trends for data communication in Grid environments. First, network technology is evolving to incorporate very high bandwidth and long delay networks (e.g. LambdaGrid [3], OptIPuter [4], CANARIE [5]), as well as wireless and sensor networks [6]. Second, application communication patterns are shifting from point-to-point communication to multipoint-to-point and multipoint-to-multipoint communication; the former is used, for example, for fetching and visualizing data from multiple remote data repositories, while the latter is used for P2P networks [7; 8]. Third, each Grid application has unique communication needs across a diverse range of

transport characteristics, such as transmission rate, delay, and loss ratio.

Standard transport protocols such as TCP [9] and UDP [10] are not always sufficient or optimal for these emerging Grid computing scenarios. For example, traditional TCP reacts adversely to increases in bandwidth and delay [11], leading to poor performance in high bandwidth-delay product networks. Moreover, each Grid application has unique QoS demands related to communication. For instance, a remote visualization application may require unreliable fixed-rate data transfer, a fast message passing application may demand reliable data transfer with minimum end-to-end delay, and a sensor grid application may need to minimize data traffic to prolong battery life. However, since TCP is designed as a general reliable transport protocol and UDP is only a minimal transport protocol with essentially no guarantees, they cannot generally provide the optimal mix of communications attributes for each Grid application.

One way to address the issue of matching transport properties to applications is to use a configurable transport protocol such as CTP [12] that supports the construction of customized protocols from collections of software modules, each of which implements a given transport property. CTP in particular provides a high degree of flexibility in how transport protocols are constructed, including support for non-hierarchical module composition, an event-driven execution model, and fine-grain modules. These properties distinguish it from other systems for building configurable protocols [13; 14; 15].

While CTP offers competitive performance for general-purpose applications, it is not designed for applications of the type run on high-performance Grids. Such applications demand high-speed data transfer across 1 Gbps or even 10 Gbps links, and in these scenarios, the data handling bottleneck becomes CPU cycles rather than network bandwidth. As a result, *any* overhead in the protocol stack can be an issue, and it becomes critical in configurable approaches to reduce as much as possible the overhead associated with the framework that implements the configurability without losing flexibility. As one indication of the potential impact of such overhead, experimental results show that a minimum configuration of CTP can only achieve 240 Mbps over a 1 Gbps link when sending messages of regular packet size.

In this paper, we address this issue by presenting an optimized version of CTP called *High Performance CTP* (H-CTP). H-CTP preserves the flexibility and reusability of CTP, while achieving performance that can reach 900 Mbps over a 1 Gbps link. The primary contributions of this paper are as follows. First, we analyze the key research challenges that need to be solved to provide satisfactory communication service to grid applications, using three sample grid applications with significantly different demands on data transport as motivation. Second, we present H-CTP as a high-performance yet flexible framework for building customized protocols to achieve various optimization goals. Third, we show how H-CTP supports customization of transport service for our three example Grid applications, and illustrate how it supports module reuse. Finally, we demonstrate experimentally the performance of H-CTP versus both CTP and TCP. The former is done by comparing base configurations to highlight the difference in the overhead of the respective frameworks, while the latter is done in the context of the three example applications.

Other work has addressed the issue of providing high-performance communication services [16; 17; 18]. A number of TCP variants [19; 20; 21; 22; 23] have been proposed for shared, packet-switched networks, as have new UDP-based user level rate-based protocols (e.g., RBUDP [24], SABUL/UDT [25], GTP [26]) that target different network scenarios and communication patterns. Note that it usually takes a significant effort to design and implement new specialized protocols such as these from the ground up, something that is addressed by frameworks such as H-CTP.

Another related effort is Globus XIO, which explicitly addresses the use of configurable protocols in the context of Grid computing [27]. While the XIO library supports configurability and code reuse with the notion of a *configurable driver stack*, its dependence on Globus limits its applicability to those applications that use this platform. In addition, the approach supported by XIO organizes individual transport/transform drivers in a sequential (top-to-bottom) order, which limits the flexibility of the driver configuration and the interaction among drivers. In contrast, H-CTP is independent of any specific Grid platform and supports non-hierarchical composition of fine-grain modules for maximum flexibility.

The remainder of the paper is organized as follows. Section 2 focuses on data transport properties and the challenges in implementing these properties for high performance Grid applications. This is followed in Section 3 by a brief overview of CTP and a description of H-CTP, as well as the results from performance comparison experiments. Section 4 demonstrates how H-CTP can be customized for three sample applications, followed by preliminary experimental results comparing H-CTP with TCP for these applications in Section 5. Finally, Section 6 offers conclusions and future work.

2. Data Transport for Grid Applications

The emerging domain of Grid computing has yielded a wide variety of network infrastructures with very different capabilities. At one extreme, based on DWDM technology, the lambda-Grid [3] is able to couple traditional distributed Grid resources with dedicated lambdas or wavelengths, with 1-10 Gbps per link, and hundreds of lambdas per optical fiber. This allows the dynamic configuration of much higher network performance and QoS than traditional network architectures. At the other extreme, wireless sensor networks have been integrated with sensor Grids [6] consisting of both high-bandwidth wired nodes and much smaller, battery-powered nodes (or motes [28]) communicating over lossy and low bandwidth links.

The communication pattern between the components of the distributed application is also highly diverse in Grid environments. The advent of large-scale computation and data sharing in wide-area Grids and peer-to-peer applications is driving an evolution in communication patterns from point-to-point connections to multipoint-to-point and multipoint-to-multipoint structures. Examples include P2P content delivery networks, and data fetching based on erasure codes [29] in which multiple replicated data repositories are accessed simultaneously for remote visualization or computation at much higher aggregate speed.

2.1. Multi-dimensional Communication Requirements

The dramatic evolution of network infrastructures and communication patterns means that an increasing number of Grid applications are demanding enhanced and diverse transport-level functionality. The properties and QoS guarantees required can vary widely depending on the intended network environment (dedicated links or shared links), communication patterns (point-to-point or multipoint-to-point), and application requirements. Below we summarize a general set of transport attributes for Grid computing; a more general discussion of transport attributes can be found in [30].

Connection oriented vs. Connectionless. This describes the connectivity type of the transmission. Some Grid services (e.g., message broadcasting for resource discovery) may require connectionless communication similar to that offered by UDP rather than connection-oriented service such as found in TCP.

Reliability. Some Grid applications require reliable data service with retransmission of lost packets (e.g., large file transfer across Grids), while others may work with unreliable best-effort transmissions (e.g., real-time streaming and visualization).

Latency and Jitter. These are characteristics that describe the inter-packet QoS. For example, small message passing may require much lower end-to-end latency than large file transfers.

Sequenced vs. Unsequenced. Some applications (e.g., application-level message passing) may require ordered delivery, while others work without such ordering (e.g., file transfer).

Rate shaping vs. Best effort. This describes the throughput requirements from Grid applications. Some applications (e.g., real-time fixed-rate streaming, or multicast) may have specific needs for data transmission rates. Note that TCP can only provide best-effort service in this regard.

Fairness. This includes the requirements on intra-protocol fairness among multiple connections, as well as fairness with other TCP connections that share the same network infrastructure.

Congestion control. Applications running on a shared network infrastructure require congestion control schemes. This may be optional for data transfer over dedicated links or can be simplified when inter-network congestion is rare such as on lambda-Grids.

2.2. Three Example Grid Applications

To better illustrate the diverse transport demands of Grid applications, here we describe three example applications. For simplicity, we focus only on core functionality.

Real-time Visualization. Consider the case in which we need to fetch data from remote repositories and visualize locally in real time. This type of application may demand fixed-rate (usually specified by application/receiver) transfer without retransmission of lost packets. The optimization goal is to achieve *specified throughput with minimum jitter*. As a result, TCP is not the best candidate, since it includes loss retransmission, and since its rate and congestion control schemes are both unnecessary and a major source of jitter.

Fast Message Passing. Consider the passing of small messages among cluster nodes or over wide-area Grid resources. This requires *connectionless reliable transfer with minimized end-to-end delay*. TCP again might not be an optimal choice since it is connection oriented, has unnecessary rate and congestion control, and has much longer default timeout settings than necessary.

Sensor Grid Data Collection. Consider communication between battery-powered sensors and a base station. The goal in this case is to *minimize the traffic sent/received by each sensor* to prolong the battery life. Translated to transport properties, this means reducing outgoing traffic from each sensor by not sending out unchanged (or only slightly changed) readings (i.e., selective transfer), as well as minimizing incoming traffic by reducing the number of acknowledgments sent from the base station to the sensors. Moreover, while the base station needs to receive data with a certain frequency, it does not require that each packet be reliably transmitted.

These three example applications have clearly distinct optimization goals, leading to different combinations of properties making up the optimal combination for a

transport service (Table 1). Note that some of the properties are shared by two or all three application scenarios, which argues for the ability to do code reuse. For example, both message passing and sensor Grid data collection require a certain degree of loss retransmission and actions on timeouts, while all three applications could use a fixed-rate service that provides an upper bound on transmission rate. Configurable transport services such as CTP and H-CTP can address these requirements by making it easy to build customized protocols instead of having to design libraries from scratch for each individual scenario.

	Visualization	Msg Passing	Sensor Grid
Connection-oriented	X		X
Loss retransmission		X	X
Fixed Rate	X	X	X
Acknowledgment		X	
Sequenced transfer	X	X	
Selective transfer			X
Timeout action		X	X

TABLE 1: Transport Properties of Three Sample Applications

3. High performance CTP (H-CTP)

H-CTP is based on CTP, so in this section we start with a brief overview of CTP. We then present H-CTP and describe in detail the optimizations used to achieve high performance.

3.1. CTP Overview

CTP is built using Cactus [31], a design and implementation framework for constructing highly-configurable network protocols. Within CTP, each transport protocol property is implemented as a separate Cactus *micro-protocol*. A micro-protocol in Cactus is structured as a set of *event handlers*, which are executed when a specified event occurs. The Cactus framework provides a variety of operations for managing events and event handlers. For example, there are operations for binding an event handler to an event and for raising an event. When an event is raised, all the handlers bound to that event are executed, in a specified order if necessary. An event can be raised with blocking (synchronous) or non-blocking (asynchronous) semantics or with a specified delay, which is useful for generating timeouts. In addition to events, micro-protocols also interact with each other using shared data structures, such as messages.

The Cactus framework provides a message abstraction, *dynamic messages*, specifically designed for configurable protocols. A dynamic message consists of a message body and an arbitrary set of named message attributes. Micro-protocols can add, read, modify, and delete attributes independent of their physical location in the message. When a message is passed to a lower-level protocol, the message attributes of the current protocol are packed and become part of the message body for the next level. The

process is reversed when a message is passed to a higher-level protocol.

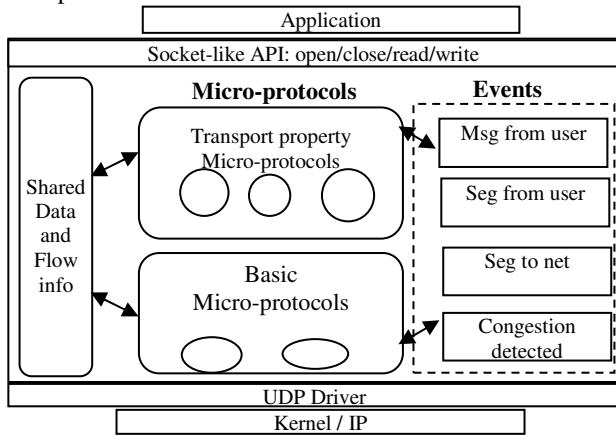


FIGURE 1: CTP Composite Protocol Framework

The CTP protocol is illustrated in Figure 1. Internally, CTP is a Cactus composite protocol with micro-protocols, events, and shared data structures. It consists of a small set of basic micro-protocols that are needed in every configuration and a set of micro-protocols implementing various transport properties that can be chosen based on the requirements of the application. CTP interacts with the application on top and with the UDP protocol provided by the underlying operating system kernel using simple adaptor layers that are implemented as simple protocols. The adaptor layer on top provides a socket-like API for applications to open and close connections, and to send and receive messages. The adaptor layer at the bottom, UDP Driver, provides an interface to UDP sockets provided by the underlying operating system.

The event interactions of the Negative Ack (NACK) micro-protocol are illustrated in Figure 2, where an arrow directed into the oval indicates an event that is handled by the micro-protocol, and an arrow directed out indicates the raise of an event. This micro-protocol sends a negative acknowledgment message if it detects a message missing from the incoming message sequence (SEGMENT FROM NET event) based on message sequence numbers or if its internal timeout event occurs (NACK TIMEOUT). It causes the message to be sent by raising the SEGMENT TO NET event. At the sender side, whenever the NACK micro-protocol receives a negative acknowledgment message (SEGMENT FROM NET event), it raises the RETRANSMIT TIMEOUT event, so that other micro-protocols (e.g., a retransmission micro-protocol) can handle the message loss.

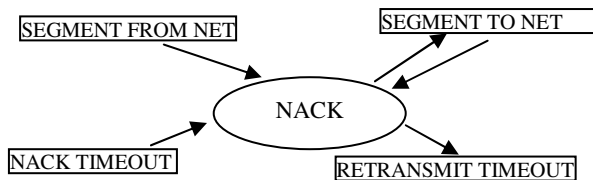


FIGURE 2: NACK Micro-Protocol Event Handling

3.2. H-CTP Description

A key research challenge for highly configurable protocol frameworks is balancing the trade-off between the inherent flexibility provided by the framework and its performance overhead. While the C version of CTP on Linux provides satisfactory performance on 100 Mbps links, the overhead of the event-driven framework becomes a major performance limiter on Gigabit links. In particular, as noted above, only 240 Mbps out of 1000 Mbps is achieved by a minimum configuration consisting of only the basic micro-protocols when sending packets matching the Ethernet packet size (1.5 KB).

To address these performance issues, we have designed and implemented a comprehensive set of optimizations that significantly reduce the overhead and make the resulting H-CTP framework suitable for high-performance Grid applications. The optimizations fall into the following general categories.

Optimized event handling. A major issue with event-based frameworks is the execution overhead caused by the indirection between modules that raise events and those that handle them. In [32] we describe a profile-directed optimization approach that identifies and eliminates unnecessary overhead associated with event-based systems. H-CTP applies some of these techniques to reduce the cost of event activation and maintenance. For example, based on the knowledge that most of the data packets from Grid applications are handled by synchronous I/O, we eliminate asynchronous event handling and state maintenance cost such as locking for global variables and data structures.

Zero-copy throughout the protocol framework. For high performance transport protocols and applications, data or buffer copying consumes a tremendous number of CPU cycles and thus, represents a significant source of performance overhead. The goal of this set of optimizations is to eliminate unnecessary buffer copies. While copy avoidance of this type has been considered in the context of kernel-level protocol stack optimization (e.g., Zero-copy TCP [33], Fast Sockets [34], and TCP optimization [35]), this is a far more serious issue for user-level communication libraries where buffer copying may take places not only within the protocol stack, but also at the boundaries between the protocol and kernel, and between the application and protocol. Figure 3 shows all possible locations that buffer-copying may occur in or around CTP.

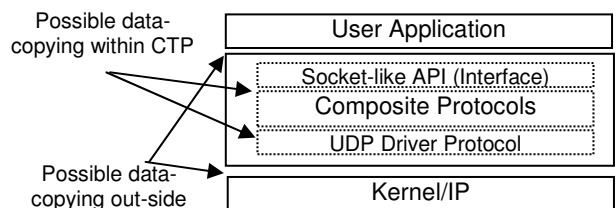


FIGURE 3: Possible Data-copying Around CTP Stacks

H-CTP effectively eliminates all the buffer copies within the CTP framework itself and attempts to minimize them elsewhere. This is done using several techniques. First, copying is avoided by only passing the pointers to data buffers through different layers of the framework, and among micro-protocols. Second, we implemented an IO-vector based packet packing function to avoid data copying when we pack multiple message attributes into one physical packet. Although the data within a physical packet will still be copied to a kernel buffer by the kernel UDP driver, all other copies within the stack are eliminated. Note that these changes do require more careful attention to be paid to which module is responsible for destroying a message.

To illustrate the performance impact of these optimizations, we added up to 3 additional buffer copies when handling each message to a basic configuration of H-CTP. Table 2 shows the achievable throughput as the number of copies changes. Note that introducing even a single copy drops the performance by close to 50%.

Number of copies	0	1	2	3
Throughput (Mbps)	901	495	330	248

TABLE 2: Throughput vs. Number of Copies

Memory reuse. Creating and destroying data structures can be very time-consuming. For example, in CTP it takes 4 μ s and 2.6 μ s to create and destroy a message, respectively. Note that in order to send data at 1 Gbps, each Ethernet-size packet needs to be handled within 12 μ s. Based on the observation that the data structures created for each message are very similar, H-CTP uses a *memory pool* similar in principle to a thread pool to reuse most of the data structures. Specifically, a memory pool containing a collection of data structures is created at the time when the H-CTP composite protocol is initiated. Then, whenever a new data structure is requested within the protocol framework, it will be allocated directly from the memory pool, thereby reducing the data structure construction time. When the task is complete, all relevant data structures are returned to the memory pool to be reused instead of being destroyed. Therefore, a relatively longer memory initialization time helps to significantly improve the run-time performance of H-CTP. For example, the performance of a basic configuration of H-CTP is reduced from 901 Mbps to 785 Mbps if the memory pool is disabled.

Re-implementing dynamic messages. H-CTP’s performance has also been improved by optimizing the implementation of dynamic messages. In CTP, both a hash table (for indexing) and a linked-list (for browsing) are used for organizing message attributes. The dynamic construction/destruction and maintenance time for these structures can be a significant overhead. For H-CTP, we implemented a statically linked hash table structure for both indexing and browsing, as shown in Figure 4. Together with the reuse of memory, the message access

performance has been dramatically improved. For example, the average message construction time has been reduced from 4 μ s to 0.25 μ s, and destruction time from 2.6 μ s to 0.22 μ s. Table 3 summarizes a performance comparison between H-CTP and CTP on message access time.

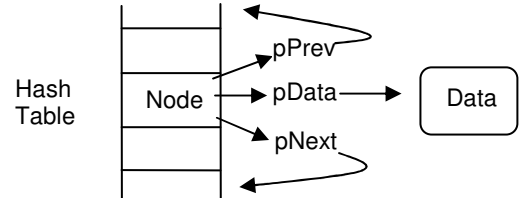


FIGURE 4: New Message Attribute Structure in Dynamic Message

	CTP	H-CTP
Single msg attribute access time	1.2 μ s	0.3 μ s
Msg initialization time	N/A ¹	8.0 μ s
Msg construction time	4.0 μ s	0.25 μ s
Msg destruction time	2.6 μ s	0.22 μ s

TABLE 3: Message Access Time Differences

3.3. Performance Comparison of CTP and H-CTP

A performance comparison between CTP and H-CTP was done using micro-protocol configuration consisting of only the basic support micro-protocols. We denote these configurations as CTP(Base) and H-CTP(Base), respectively. In addition to the micro-benchmarking results already presented in Table 3, we compared the two frameworks by measuring the sustained throughput and loss ratio over a 1 Gbps connection. These results are summarized in Table 4. We observe that H-CTP(Base) is able to achieve much higher throughput and lower packet loss than CTP(Base). The packet loss difference is due to relatively slow packet handling at the receiver in CTP(Base), which causes packets to be dropped.

	CTP(Base)	H-CTP(Base)
Throughput	240 Mbps	901 Mbps
Loss Ratio	16.9%	0.01 %

TABLE 4: Performance Comparison between CTP and H-CTP

To better illustrate the capabilities of H-CTP, we compare its performance with CTP across varying numbers of basic event and message handling operations. Figure 5 shows the throughput of H-CTP(Base) and CTP(Base) when extra event raises are added to the base configuration beyond the four in the original configuration. Each extra raise is handled by an event handler that reads one message attribute. Figure 5 indicates that the throughput of both H-CTP(Base) and CTP(Base) decreases when introducing extra event raises. However H-CTP(Base) can still achieve over 800 Mbps with 9 additional raises, and the throughput of H-CTP(Base) decreases at a much slower rate than for CTP(Base).

¹ CTP does not have a separate message initialization.

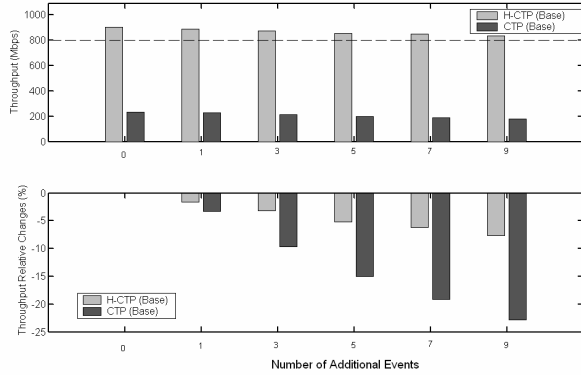


FIGURE 5: Top: Throughputs of H-CTP(Base) and CTP(Base) flows over 1 Gbps link with additional events. Bottom: The relative changes, compared with the throughput with 0 additional events.

Figure 6 depicts the throughput of H-CTP(Base) and CTP(Base) when additional message attributes are added to messages beyond the original three (data, source/destination address/port pairs). It shows that H-CTP(Base) can still reach 800 Mbps even when 7 additional attributes are added. It also has a lower rate of performance degradation compared to CTP(Base).

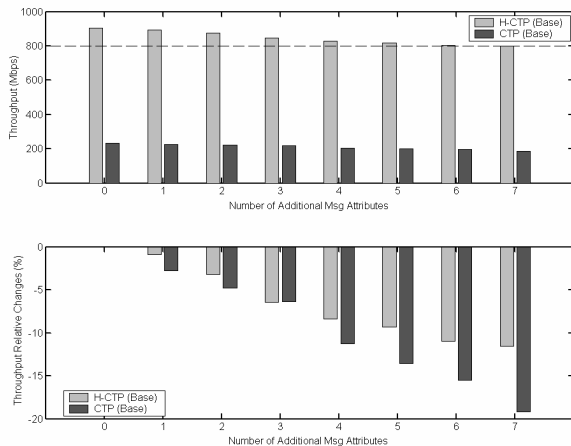


FIGURE 6: Top: Throughput of single H-CTP(Base) and CTP(Base) with additional message attributes. The packed message size is fixed as 1460 bytes. Bottom: The relative changes when compared with the performance with zero additional attributes.

Figure 7 compares the performance of H-CTP(Base) and CTP(Base) when sending messages with different sizes. Note that the standard Ethernet packet size is 1.5 KB. We observe that the performance of H-CTP(Base) is affected when the message size is between two to four Ethernet packet sizes; this is because the kernel needs to perform fragmentation for packets larger than 1.5 KB, which is also CPU consuming. As the message size keeps increasing, however, the throughput goes back up until it reaches the original level. This occurs because the per-packet handling overhead of H-CTP is almost a constant based on our zero-copy techniques, which means that the per-byte

handling time decreases as the message size grows. The performance of CTP(Base) also improves as the message size increases, albeit at levels far below H-CTP(Base). This increase is again attributable to the amortization of overhead over a larger message. The lack of a dip as seen with H-CTP follows because the kernel fragmentation has less relative impact given the lack of message handling optimization in CTP.

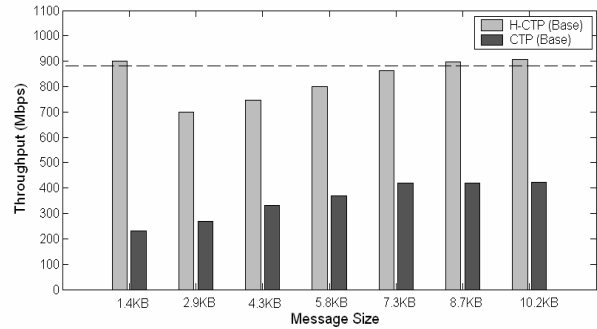


FIGURE 7: Throughput of H-CTP(Base) and CTP(Base) with different message sizes.

4. Building Transport Services with H-CTP

In this section, we highlight some of the micro-protocols we built to match the transport service requirements of the three example applications described above.

4.1. Basic Micro-protocols

Transport Driver is one of two basic micro-protocols that must be present in any protocol configuration. At the sender side it adds sender-receiver IP/port identifiers as message attributes to identify each active connection. At the receiver side, H-CTP uses these identifiers to do user-level message demultiplexing.

Fixed Size is the other required micro-protocol. Its handler is invoked when the MSG FROM USER event is raised to convert the message into a network segment. After it does so, it raises the SEG FROM USER event.

Sequenced Segment is an optional basic micro-protocol that adds sequence numbers to each message as an attribute to uniquely identify each outgoing segment. These numbers are used by other micro-protocols that deal with reliability and congestion control.

4.2. Micro-protocols for Three Grid Applications

Simple Remote Visualization, H-CTP(Viz): The key functionality is to transfer remote data with a fixed and receiver-specified rate. The Fixed Rate micro-protocol is designed to control the data sending rate to be no more than a specified value. Similarly, Rate Feedback is designed to send the receiver-specified rate to the sender as feedback, which is then used as the new rate for Fixed Rate. Rate Stats is an optional micro-protocol that maintains statistics (throughput, loss ratio, etc.) of the current connection. It works with Rate Feedback and

other flow and congestion control micro-protocols to provide receiver-based rate control.

Reliable Fast Message Passing, H-CTP(Msg): Besides using Fixed Rate to regulate the message sending rate, Positive Ack is used to send acknowledgments from the receiver to the sender when a message arrives. The Retransmit micro-protocol keeps track of outstanding messages and retransmits a message in response to a timeout event raised by Positive Ack.

Sensor Grid, H-CTP(Sensor): Instead of using Positive Ack for reliable message delivery, a Negative Ack micro-protocol is used in order to eliminate traffic sent from the base station to the sensors. With this approach, a NACK message is sent to a sensor only when the base station has not received any data from it for a certain period of time. The Retransmit micro-protocol is reused for data retransmission at the sensor side, and Fixed Rate and Rate Feedback described above are reused so that a sensor follows the data sending rate specified by the base station. A new optional Selective Transfer micro-protocol is provided to further eliminate data transfers from the sensor; it only passes data for transmission if the new reading from the sensor is significantly different from the previous one based on a given comparison criteria.

4.3. Reusability of Micro-protocols

We summarize the micro-protocols used in these three sample application scenarios in Table 5. From this, it can be seen that many micro-protocols can be used in two or all three scenarios, a pattern that matches the properties identified for the scenarios in Section 2.B as one would expect. In general, when designing a micro-protocol collection for a new application, one would first attempt to reuse available micro-protocols with similar desired functionality, and then develop new micro-protocols to fulfill the rest of the requirements.

	Visualization	Msg Passing	Sensor Grid
FixedSize	X	X	X
TransportDriver	X	X	X
SequencedSegment	X	X	X
FixedRate	X	X	X
RateStats	X		
RateFeedback	X		X
Retransmit		X	X
PositiveAck		X	
NegativeAck			X
SelectiveTransfer			X

TABLE 5: Reusability of Micro-protocols

5. Experimental Results

In this section, we compare the H-CTP configurations for the three example applications with untuned TCP under varying network conditions. The experiments were conducted in a local cluster environment of Intel Xenon 2.4 GHz machines with 2 GB memory, where the network

delays and packet losses were emulated by Dummynet [36].

5.1. Remote Visualization

In this scenario, we set the sending rate of the different configurations of H-CTP (*Viz*, *Msg*, *Sensor*) to be 200 Mbps. We compare their throughput with TCP over a 1 Gbps link with various round trip times (RTTs), as depicted in Figure 8. Note that a RTT of 50 ms or even more than 100 ms would be normal for many large grids, especially those that span multiple continents. It can be seen from the graph that TCP reacts adversely to increases in bandwidth and delay, which makes it difficult for TCP to provide high performance over links with long delays without tuning, assuming shared links. In comparison, the three customized H-CTP configurations are able to maintain high transmission rates across different RTTs. Among these, only the throughput of H-CTP(Msg) drops when the RTT increases, which is due to the overhead of reliable transmission handling (i.e., maintaining a sliding window).

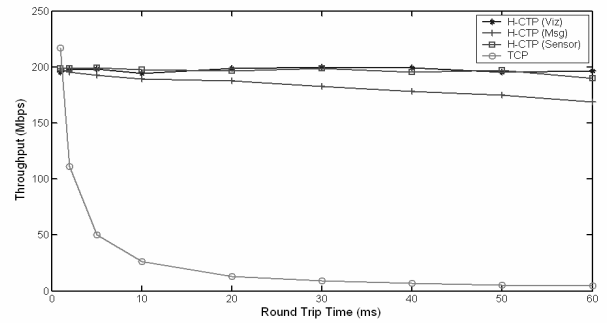


FIGURE 8: Throughput of three H-CTP configurations (*Viz*, *Msg*, *Sensor*), and TCP over 1 Gbps link with various RTTs.

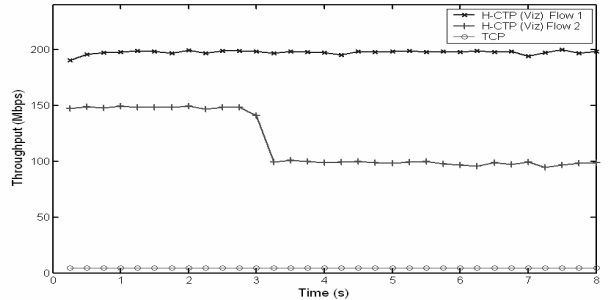


FIGURE 9: Throughput of single TCP and two H-CTP(*Viz*) flows over 1 Gbps link with 60 ms RTT. H-CTP(*Viz*) flow 1 is with a receiver-specified rate of 200 Mbps; H-CTP(*Viz*) flow 2 starts with a receiver-specified rate of 150 Mbps, and then changes to 100 Mbps at the request of the receiver.

Compared with TCP, the H-CTP(*Viz*) configuration also has the flexibility of adapting the transmission rate. Figure 9 shows the trajectories of three flows, two H-CTP(*Viz*) and one TCP, of which one H-CTP(*Viz*) flow maintains a receiver-specified rate of 200 Mbps. The other H-CTP(*Viz*) flow starts with a receiver-specified rate of 150

Mbps, followed by a receiver requested rate change to 100 Mbps after 3 seconds. In contrast, traditional TCP lacks the ability to perform receiver-specified rate adaptation, and can only provide best-effort service at a very low rate over high bandwidth-delay product links.

5.2. Fast Message Passing

Since H-CTP(Msg) is the only configuration with per-packet reliable transmission, we only compare it with TCP on round-trip delay. We measure the message turn-around time from the time the sender transmits a small message of 10 bytes until the sender receives a response. To offset the piggy-back effect of TCP acknowledgment, we let the receiver sleep for 2 seconds before sending back the reply; this constant offset has been deducted from the average value of 1000 measurements. Figures 10 and 11 show the round-trip delay with different random link packet losses on two connections with 60 ms and 1 ms RTTs, respectively. We observe that for both cases, H-CTP(Msg) has a much shorter delay than TCP, especially when there is significant packet loss on the link. TCP's poor performance is mainly due to its default long retransmission timeout.

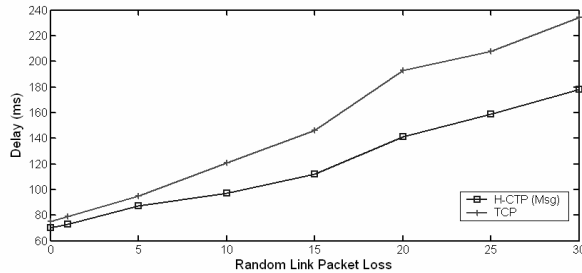


FIGURE 10: End-to-end message turn-around transmission delay over a link with 60 ms RTT with various introduced random packet loss

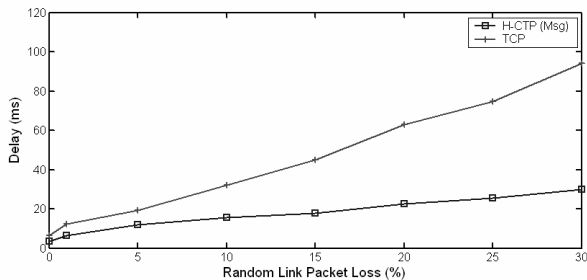


FIGURE 11: End-to-end message turn-around transmission time over a link with 1 ms RTT with varying random packet loss

5.3. Sensor Grid

The optimization goal in the sensor grid case is to minimize incoming and out-going traffic for each sensor. Among three H-CTP configurations, H-CTP(Msg) and H-CTP(Sensor) are the ones that implement retransmission timeouts and can perform all the required functions for this scenario. We compare the number of packets transferred by each protocol with TCP, as shown in Table 5. Since TCP and H-CTP(Msg) use positive acknowledgments, for

each new reading, one data packet is sent from the sensor to the base station and one ACK is sent in the other direction. In comparison, H-CTP(Sensor) uses a negative acknowledgment scheme, which does not send back any NACK packets from the base station when it is able to receive data readings with a certain frequency. Moreover, the Selective Transfer micro-protocol may even reduce the number of data packet being sent further when the new readings generated by sensors are close to the old ones.

	TCP	H-CTP(Msg)	H-CTP(Sensor)
Data Packet	1	1	≤ 1
Ack	~ 1	~ 1	0

Table 5: Number of packets needed for each data reading for TCP, H-CTP(Msg) and H-CTP(Sensor).

5.4. Discussion and Related Work

Through experiments we are able to compare H-CTP with TCP under different configurations, showing that H-CTP achieves much better performance than TCP for the three example scenarios. A number of frameworks other than CTP have also been proposed for constructing transport services that are configurable to some degree. Examples include Globus XIO [27], System V Streams [37], the *x*-kernel [13], CORDS [15], and Horus [14]. Most of these are not oriented towards high performance computing, and for this and other reasons, it is difficult to obtain comparable performance numbers. Further exploration of the best way to do such comparisons is part of our future work. In addition, as discussed in [12] none of these systems are able to construct transport services that are customizable to the same degree as CTP or H-CTP.

For the purpose of better demonstrating the capabilities of H-CTP, we have simplified the requirements for each of the three Grid application scenarios. We believe the same level of performance would still be possible when H-CTP is customized for more complicated and realistic usage scenarios. This belief is based in part the H-CTP experiments reported here, which demonstrate satisfactory performance under varying workloads, and across varying numbers of event raises, message attributes, and message sizes.

Since H-CTP has a socket-like I/O interface, the migration of Grid applications from TCP to H-CTP only involves limited effort. As future work, we are interested in building a Globus XIO wrapper for H-CTP. This will allow Grid application developers to switch transparently between H-CTP and other transport protocols. When configuring H-CTP for applications running in a public Grid environment in which most of the traffic is based on classical TCP, attention needs to be paid to avoid having H-CTP starve TCP traffic. We can either impose a sending rate threshold for H-CTP or implement a TCP-friendly traffic shaping micro-protocol to limit the outgoing H-CTP traffic.

6. Conclusions and Future Work

Standard network protocols such as TCP and UDP are not always a good match for emerging Grid applications, which can have widely varying transport service requirements as well as a need for good performance across high-bandwidth networks. Here, we have described H-CTP, a version of CTP that provides the ability to build custom transport services for such applications, while still providing high performance under a variety of conditions. Using a collection of optimization techniques, we are able to achieve the flexibility of protocol configuration, maximum reusability of micro-protocols, and high performance all at the same time. Experimental results show that H-CTP achieves much higher performance than CTP and significantly outperforms TCP for three sample Grid application scenarios.

We identify the following future work. First we intend to use H-CTP to construct more complex point-to-point transport services (e.g., SABUL/UDT) and group communication services (e.g., GTP) to further validate and test our approach. Second, we are interested in customizing the H-CTP interface to comply with the Globus XIO interface specification as a way to provide flexible high-performance transport services to a wider collection of Grid applications. Third, we intend to extend H-CTP to support run-time reconfiguration based on network conditions and application requirements. Last, we hope to explore H-CTP's performance over even higher bandwidth networks, such as those with 10 Gbps links.

Acknowledgments

This work is part of the collaboration between AT&T Labs-Research and the UCSD Center for Networked Systems, and it was done primarily while Ryan Wu was an intern at AT&T Labs-Research. This work was also supported in part by the National Science Foundation under awards NSF Cooperative Agreement ANI-0225642 (OptIPuter), NSF CCR-0331645 (VGrADS), NSF ACI-0305390, and NSF Research Infrastructure Grant EIA-0303622. Support from the UCSD Center for Networked Systems, BigBangwidth, and Fujitsu is also gratefully acknowledged.

References

- [1] I. Foster and C. Kesselman, "The Grid: Blue print for a new computing infrastructure." Morgan Kaufmann, 1999.
- [2] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the Grid: An open grid services architecture for distributed systems integration." Grid Forum white paper, 2003.
- [3] T. DeFanti, C. d. Laat, J. Mambretti, K. Neggers, and B. Arnaud, "TransLight: A global-scale LambdaGrid for e-science." *Communications of the ACM*, 47(11), Nov 2003.
- [4] L. Smarr, A. Chien, T. DeFanti, J. Leigh, and P. Papadopoulos, "The OptIPuter." *Communications of the ACM*, 47(11), Nov 2003.
- [5] "CANARIE." <http://www.canarie.ca>.
- [6] M. Gaynor, M. Welsh, S. Moulton, A. Rowan, E. LaCombe, and J. Wynne, "Integrating wireless sensor networks with the Grid." *IEEE Internet Computing*, special Issue on Wireless Grids, Jul/Aug 2004.
- [7] "Kazaa." <http://www.kazaa.com>.
- [8] "BitTorrent." <http://bitconjurer.org/BitTorrent/>.
- [9] J. Postel, "Transmission control protocol." RFC 793, Sept 1981.
- [10] J. Postel, "User datagram protocol." RFC 768, Sept 1980.
- [11] B. Jacobson, "TCP extensions for high performance." RFC 1323, May 1992.
- [12] G. Wong, M. Hiltunen, and R. Schlichting, "A configurable and extensible transport protocol." In Proc. INFOCOM 2001, pp. 319-328, Apr 2001.
- [13] N. Hutchinson and L. Peterson, "The x-kernel: An architecture for implementing network protocols." *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64-76, Jan 1991.
- [14] R. v. Renesse, K. Birman, R. Friedman, M. Hayden, and D. Karr, "A framework for protocol composition in Horus." In Proc. ACM PODC-14, pp. 80-89, Aug 1995.
- [15] F. Travostino, E. Menze, and F. Reynolds, "Paths: Programming with system resources in support of real-time distributed applications." In Proc. IEEE WORDS, Feb 1996.
- [16] "A survey of transport protocols other than standard TCP." GGF Data Transport Research Group.
- [17] A. Falk, T. Faber, J. Bannister, A. Chien, R. Grossman, and J. Leigh, "Transport protocols for high performance." *Communications of the ACM*, 47(11), Nov 2003.
- [18] X. Wu and A. Chien, "Evaluation of rate based transport protocols for lambda-Grids." In Proc. HPDC-12, Jun 2004.
- [19] S. Floyd, "Highspeed TCP for large congestion windows." Internet draft, Aug 2002.
- [20] S. Floyd, S. Ratnasamy, and S. Shenker, "Modifying TCP's congestion control for high speeds." Available from URL <http://www.icir.org/floyd/hstcp.html>.
- [21] C. Jin, D. Wei, and S. Low, "FAST TCP: Motivation, architecture, algorithms, and performance." In Proc. INFOCOM'04, Mar 2004.
- [22] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for high bandwidth delay product network." In Proc. SIGCOMM 2002, Aug 2002.
- [23] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks." *ACM SIGCOMM Computer Communication Review*, 33(2):83-91, April 2003.
- [24] E. He, J. Leigh, O. Yu, and T. DeFanti, "Reliable blast UDP: Predictable high performance bulk data transfer." *IEEE Cluster Computing*, p. 317, 2002.
- [25] Y. Gu and R. Grossman, "Experiences in Design and Implementation of a High Performance Transport Protocol." In Proc. SC2004, Nov 2004.
- [26] X. Wu and A. Chien, "GTP: Group transport protocol for lambda-grid." In Proc. CCGrid-4, Apr 2004.
- [27] "Globus XIO." <http://www-unix.globus.org/developer/xio/>.
- [28] J. Hill, "System architecture for networked sensors." Doctoral thesis, Dept of Computer Science, University of California, Berkeley, 2003.
- [29] L. Rizzo, "Effective erasure codes for reliable computer communication protocols." *Computer Communication Review*, vol. 27, no. 2, pp. 24-36, Apr 1997.
- [30] S. Iren, P. Amer, and P. Conrad, "The transport layer: Tutorial and survey." *ACM Comp. Surveys*, vol. 31, no. 4, pp. 360-405, Dec 1999.
- [31] M. Hiltunen, R. Schlichting, X. Han, M. Cardozo, and R. Das, "Realtime dependable channels: Customizing QoS attributes for distributed systems." *IEEE Transactions on Parallel and Distributed Systems*, Vol 10, no. 6, pp. 600-612, Jun 1999.
- [32] M. Rajagopalan, S. Debray, M. Hiltunen, and R. Schlichting, "Profile-directed optimization of event based programs." In Proc. PLDI'02, Jun 2002.
- [33] J. Chu, "Zero-Copy TCP in Solaris." *Usenix'96*, Jan 1996.
- [34] S. Rodrigues, T. Anderson, and D. Culler, "High-performance local area communication with fast sockets." *Usenix'97*, Jan 1997.
- [35] J. Chase, A. Gallatin, and K. Yocum, "End-system optimizations for high-speed TCP." in *IEEE Communications Magazine*, 39(4):68-74, Apr 2001.
- [36] L. Rizzo, "DummyNet: A simple approach to the evaluation of network protocols." *Computer Communication Review*, 27, pp. 13-41, Jan 1997.
- [37] D. M. Ritchie, "A stream input-output system." *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 8, pp. 311-324, Oct 1984.