An Efficient Communication Infrastructure for IBM Cyclops-64 Computer System

Ge Gan *Ziang Hu* *Juan del Cuvillo* *Guang R. Gao*

CAPSL Technical Memo 66
June 12, 2006

Abstract

This paper presents the design and implementation of a communication protocol for the IBM Cyclops-64 (C64) supercomputer system to enable reliable data transfer between the two major components of a C64 system: the C64 host system (also called C64 front-end) and the C64 compute engine (also called C64 back-end). The building block of C64 compute engine (C64 chip) employs a multi-core-on-a-chip architecture. A C64 computer system includes (in its compute engine) a large number of C64 nodes (chips) that are arranged in a 3D-mesh cellular structure. The compute engine is attached to the host system via Gigabit Ethernet links. The host system can be a Linux cluster that provides a familiar operating environment to the end users, plus special services targeted to the C64 compute engine, including system administration, job scheduling, file I/O, and remote memory operations, etc. Early in the design stage, a clear specification of the application requirements is presented to the design team: most of the communication is bulk data transfer, with little interactive requirement, and is mainly used to feed program code and huge amount of user data that need to be processed to the back-end of a C64 machine. Based on these requirements, this paper introduces CDP (Cyclops Datagram Protocol), a simple, reliable, and programmable communication protocol we designed for the communication between C64 back-end and the host system. Our CDP protocol has the following features: (1) creating a global name space on the C64 back-end and the host system; (2) providing a reliable communication channel between the C64 and host nodes; (3) providing a set of standard programming interfaces to system software developers; (4) it employes a simple design that are engineered to provide just enough capacity to achieve the specific application requirements. The CDP protocol has been fully implemented and tested, and a report of the experimental results is included that demonstrated that our design has met the application requirement.

Contents

List of Figures

List of Tables

1 Introduction

This paper presents the design and implementation of a communication protocol for the IBM Cyclops-64 (C64) supercomputer system to enable reliable data transfer between the two major components of a C64 system: the C64 host system (also called C64 front-end) and the C64 compute engine (also called C64 back-end). The C64 compute engine consists of tens of thousands of C64 chips, which are arranged in 3D-mesh cellular structure. For convenience, we also call it 3D-mesh network. The C64 compute engine is attached to a host system via Gigabit Ethernet links. The host system can be an off-the-shelf Linux cluster that provides a familiar operating environment to the users. All the system services like system administration, job scheduling, file I/O, and remote memory operations are processed through the host system. CDP, short for Cyclops Datagram Protocol, is developed to run over the 3D-mesh network and Ethernet to provide reliable communication links between the C64 nodes and host nodes.

Early in the design stage, a clear specification of the application requirements for the protocol is presented to the design team: most of the communication over the protocol is bulk data transfer, with little interactive requirement, and is mainly used to feed program code and huge amount of user data that need to be processed to the back-end of C64 machine. There is no stringent round-trip-latency requirement. But the protocol should retain at least half of the channel capacity under the Gigabit Ethernet environment, i.e. 500Mbps. In addition, the communication over the protocol should be reliable and the programming on the protocol should be easy. Regarding these requirements, we developed the CDP protocol, which serves as a general communication infrastructure in the C64 computer system. Its main features are:

- CDP creates a global name space on the C64 back-end and the host system. Every node in the communication domain is assigned a unique name. CDP users can use this name to identify or locate each node without the effort to differentiate between C64 nodes and host nodes. In this way, CDP hides the heterogeneous physical connections of the 3D-mesh network and Gigabit Ethernet from the system software developers and the end users.

- CDP provides reliable and efficient communication links between the C64 nodes and the host nodes. It deploys the slide window mechanism to support packet retransmission and flow control. CDP try its best to make sure that the transmission is correct, in-order, without lost, and without duplicate.

- CDP provides a set of standard programming interfaces for C64 system software developers. The programming interface is similar to BSD socket APIs that are widely used in network programming. All C64 system services like file I/O, job scheduling, system administration are built on top of CDP.

- CDP employes a simple design that are engineered to provide just enough capacity to achieve the specific application requirement. We came up the design of CDP by simplifying and customizing the functionalities defined in TCP and IP protocols. We merged these two protocol layers into one in CDP and implemented it as a user level protocol. The throughput of CDP when transferring large size packet is comparable with TCP and UDP, though CDP is running in user level.

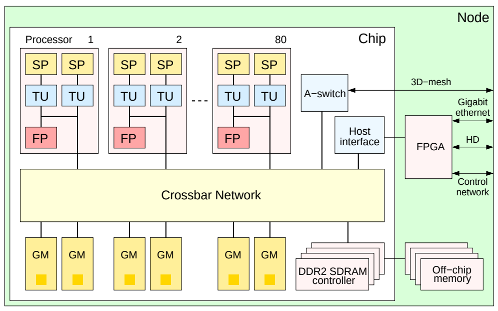
The paper is organized as follows. In section 2, we will introduce the details of the C64 chip architecture and system organization. In section 3, we will present the our motivations to develop CDP. In section 4, we will review the design and implementation details of CDP. In section 5, we present some performance data about CDP, as well as our analysis. The future work will be introduced in section 6.

2 Cyclops-64

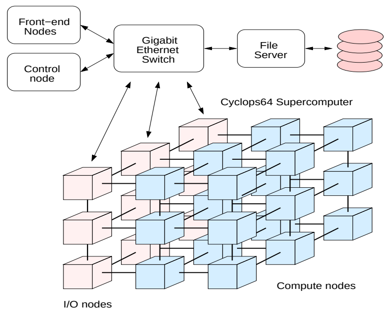
As we all know, with the number of transistors integrated on a chip doubled every 18 months in the last three decades, the power density of the chip becomes a big problem to continuously improving its performance. In addition, after the CPU clock rate reaches multiple Gigahertz, the long memory access latency becomes unbearable. Now, a cache miss may cost several hundreds or even nearly a thousand CPU cycles. It is increasingly clear that, following the traditional architecture design methodology, it is not possible to solve, or even alleviate these problems.

Under this background, "multi-core-on-a-chip" is emerging as a new architecture design methodology with very high potential for the next generation computer processor. It utilizes the huge number of transistors to integrate tens or hundreds of simple processor cores on a chip. Each "processor" on the chip can independently run a thread. If any thread was stalled because of accessing memory, other threads can still keep running to hide the memory latency. Usually, the chip operates at moderate clock rate to keep power consumption at low level. Multi-core now becomes the main trend in architecture design. Many companies have announced their multi-core plan or have already shipped out their multi-core products. Cyclops-64 (C64) [3, 4] is one multi-core architecture designed for high end computing. It is under development by IBM T.J. Watson Research Center, ET International Inc., and the University of Delaware. C64 supercomputer will be built out of tens of thousands of C64 processing nodes (chips), which are arranged in 3D-mesh cellular structure.

Figure 1(a) shows the internal structure of a C64 processing node. Each C64 node consists of a C64 chip, 1GB external DRAM, and a small amount of external interface circuitry. A C64 chip has integrated 80 "processors", which are connected to a 96-port crossbar network to form a tightly-coupled SMP structure. Each processor has two thread units, one floating point unit, and two SRAM memory banks, each 32KB. A thread unit is a 64-bit, single issue, in-order RISC processor core operating at clock rate of 500MHz. A 32KB instruction cache, not shown in the figure, is shared among five processors. There is no data cache in the processor, because it is very difficult to maintain cache coherence efficiently across so many processors. For this reason, a portion of each SRAM memory bank can be configured as scratchpad memory (SP), which is a fast temporary storage that can be used to exploit locality under software control. All of



(a) Cyclops-64 Chip Architecture



(b) Cyclops-64 Supercomputer

Figure 1: Cyclops-64 Cellular Architecture

the remaining part of the SRAM form the global memory (GM) and is uniformly addressable from all thread units.

The A-switch interface in the chip connects the C64 node to its six neighbors in the 3D-mesh network. In every CPU cycle, A-switch can transfer one double word (8 bytes) in one direction. Since the C64 chip operates at 500MHz, the bandwidth of each link in the 3D-mesh is 32Gbps. The 3D-mesh network may scale up to several ten thousands of nodes, which will form the powerful parallel compute engine of the C64 supercomputer.

A fraction of the C64 nodes in 3D-mesh, labeled as I/O nodes, use Gigabit Ethernet port, shown in Figure 1(b), to connect the C64 compute engine to the host system. Each I/O node will service a certain number of C64 nodes, called compute node. The I/O nodes relay requests and data between compute nodes and the host system. The I/O nodes and compute nodes in the 3D-mesh communicate only via the A-switch interface. On the other side, the host system consists of a set of front-end nodes, control nodes and file server nodes. It supports application program development and execution, system administration, and also hosts the file system of the C64 supercomputer.

The whole C64 system is designed to provide petaflop computer performance. It is targeted at applications that are highly parallelizable and require enormous amount of computing power.

3 Motivation

The C64 architecture design deviates from all mainstream general purpose processors. The C64 chip owns so many hardware resources - more than one hundred thread units and about 5MB on-chip memory - but it provides no resource virtualization mechanism. The thread unit is not preemptable and there is no virtual memory subsystem. Every single transistor on the C64 chip is integrated to contribute more computing power. The C64 chip is designed to run computation-intensive applications, not supposed to run a general purpose operating system. For these reasons, we decided not to port the "heavy weight" Linux kernel to C64, but develop a light weight runtime system, or kernel, for it. In addition, we also decided not to deploy

TCP/IP in C64 supercomputer. The reason is almost the same. TCP/IP stack is "heavy weight" and control-intensive. It is not suitable for it running on a computation orientated processor. Based on this thinking, we decided to develop our own communication protocol according to the practical C64 physical network environment.

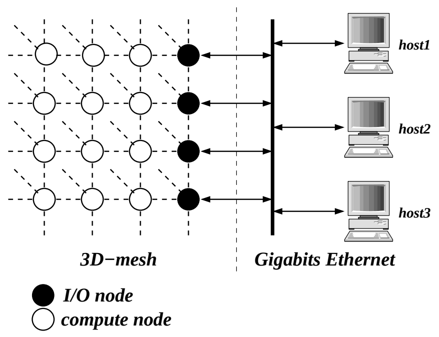


Figure 2: Network Topology of Cyclops-64 Supercomputer

The network topology in C64 supercomputer is simple. As Figure 2 shows, the compute nodes and I/O nodes are connected via 3D-mesh network. The correctness and reliability of the 3D-mesh communication are guaranteed by hardware, i.e. A-switch. The C64 I/O nodes are connected to host nodes through Gigabit Ethernet links. Our job is to build a communication framework over these two heterogeneous networks, i.e. 3D-mesh and Ethernet, to hide the physical connection details and provide reliable communication links between the C64 nodes and the host nodes.

Given the underlying physical network of C64 supercomputer, we designed CDP, short for **C**yclops **D**atagram **P**rotocol. CDP runs across the 3D-mesh network and Gigabit Ethernet. It transfers datagrams, or CDP packets, between C64 nodes and host nodes. It creates a global name space on the C64 nodes and the host nodes and provides a reliable connection between the two sides. All important C64 system services, such as system administration, job scheduling, remote memory operations etc., run on top of CDP [4]. We will present the implementation details of CDP in the next section.

4 CDP

Figure 3 shows the position of CDP in the protocol stack. Regarding the OSI reference model, CDP corresponds to the *Transport* and *Network* layers. Actually, its functionality is the convergence of these two layers. In the procedure of implementing CDP, we referenced a lot from the design of TCP/IP [10, 11], such as reliable data transfer, flow control, connection creation and termination, and addressing and routing, etc. We implement all these functionalities in a single protocol layer and provide a Berkeley-like socket API for the software developers.

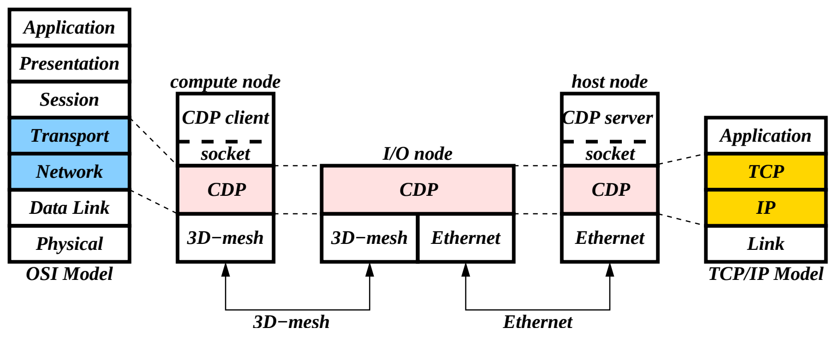


Figure 3: CDP Stack Model, OSI Model, and TCP/IP Model

4.1 Overview

CDP is connection oriented. It provides connections between the C64 nodes and the host nodes. A CDP client (usually a C64 node) establishes a connection with a given server(usually a host node), exchange data with that server, then terminate the connection. CDP connection is reliable. When CDP sends a packet to the other end, it requires to receive an acknowledgment in return. CDP automatically retransmits the packet if the acknowledgment is not received during a certain amount of time. After some number of retransmissions, CDP will give up and terminate the connection. CDP provides flow control. CDP always tells the peer exactly the amount of room currently available in the receiving buffer. The sender will stop sending data if the receiving buffer on the other end is full, guaranteeing that the sender can not overflow the buffer. CDP is a fully-duplex communication protocol. The CDP client and server can send and receive data in both directions on their connection at any time.

The implementation of CDP is asymmetric. On the C64 side, CDP is implemented in the C64 runtime system. While on the host side, CDP is implemented as a library running in user level instead of an implementation in the kernel. The reason for this design is that our customers don't want us to touch the Linux kernel that runs on the host system. Sacrificing certain amount of performance is fine for them to secure high-security of the system.

4.2 CDP Packet Format

The most important data object in CDP is the CDP packet. Its format is shown in Figure 4, which consists of two segments: the header segment and the user data segment. The CDP header contains a four-field tuple, <*source node,source port,destination node,destination port*>. Every node in the C64 supercomputer system, no matter it is a C64 node or a host node, is assigned a unique name, i.e. node number, to identify itself. CDP uses this node number to perform addressing and routing. In addition, different services on the same node use different port numbers to differentiate with each other. Therefore, the sub-tuple <*source node, source port*> identifies the source party of the communication; the sub-tuple <*destination node, destination port*> identifies the destination party of the communication. The whole tuple identifies

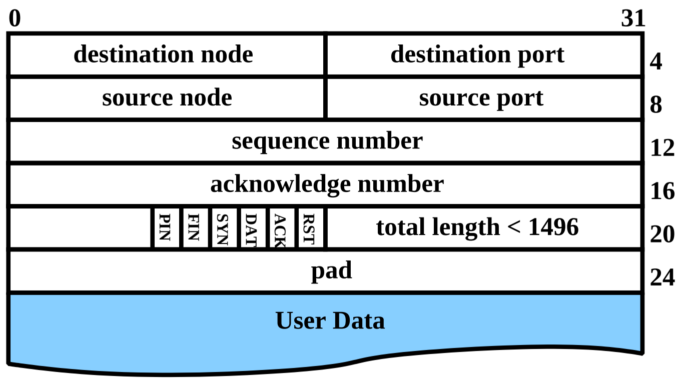


Figure 4: CDP Packet Format

a unique CDP connection in the system.

The *sequence number* field in the header is used to identify a unique CDP packet transmitted on a given connection. CDP associates each packet transmitted on a given connection with a sequence number. The sequence number increases monotonously from an initial sequence number (ISN), which is randomly chosen for every connection, to $2^{64} - 1$, and then round back to zero. It will take a very long time for the sequence number to round back to its old value. So, the sequence number on a given CDP connection is unique during a considerable period of time. Because of this reason, CDP use sequence number to detect duplicate packets, to sort the receiving queue if the packets arrived out of order, and to identify un-received packets.

The *acknowledge number* field is to acknowledge the packets that have been successfully received by the remote end. A value x in this field indicates that all packets with sequence number smaller than x have been successfully received by the remote end.

The *flags* field contains some control bits to direct the state transition of CDP connection. Currently, CDP support six control bits:

- RST: to tell the receiver that the sender has reset the connection;
- ACK: to tell the receiver that the *acknowledge number* field in the packet is significant;
- DAT: to tell the receiver that the user data segment contains useful data;
- SYN: to tell the receiver that the sender intended to establish a CDP connection;
- FIN: to tell the receiver that the sender closed the connection;
- PIN: to tell the receiver that the sender is still alive.

The *length* field stores the length (in bytes) of the CDP packet, including the header.

These fields in the header is to make sure the data transmission on a given connection is correct, in-order, without lost, and without duplicate. The rest of the packet is the user data segment. Its length (<1472) is limited by the MTU of Ethernet. Currently, CDP does not support Gigabit Ethernet jumbo frames [5], which has much bigger MTU (9000 bytes). We do not include a checksum field in CDP header. Because CDP is running on a local network with not many errors, the *CRC* field in Ethernet frame is good enough to detect most of the data corruption. This, on the other hand, avoids a lot of overhead.

4.3 CDP connection

CDP connection establishment and termination mostly follow the same way as TCP. For example, CDP deploys the 3-way handshake schema, the same as used in TCP, to establish a connection between a CDP client and a CDP server. CDP connection state transition diagram is shown in Figure 5. Regarding the TCP connection state transition diagram, CDP made two changes: the simplified 2-way handshake connection termination semantics; and the CDP keep-alive mechanism.

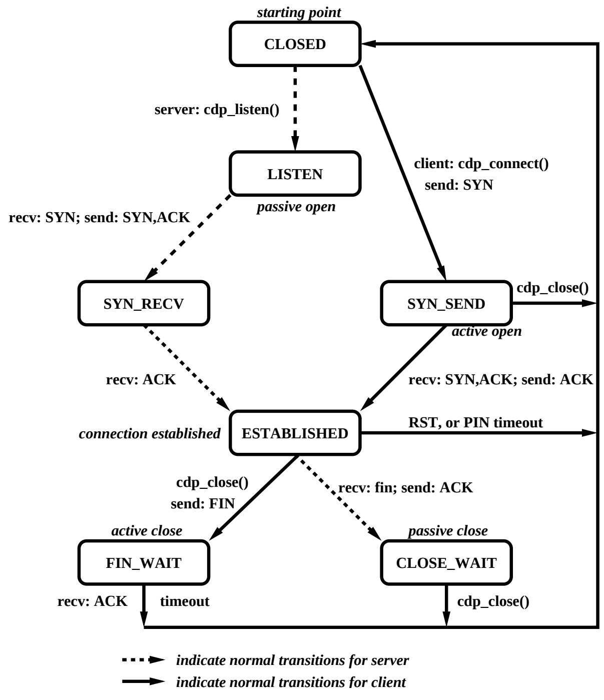


Figure 5: CDP Connection State Transmission

CDP does not support "half close" connection. As shown in Figure 5, the CDP end who proactively closes the connection will send a *FIN* packet to the other side and enter the *FIN_WAIT* state, waiting for an *ACK* packet in return. After receiving the *ACK* packet, it enters the *CLOSED* state. From this moment, the connection was closed on both directions. No data can be sent and received on it. This is different from TCP, in which the connection is not completely closed (in *FIN_WAIT_2* state), and the program can still receive data from the other side, though it can not send out data anymore.

On the other hand, the CDP end that receives a *FIN* packet will put the connection into *CLOSE_WAIT* state, waiting for the user to close it completely (by calling *cdp_close()*). In this state, users can still receive data from the receiving buffer (if there are still data there), but no new data will be sent out any more, because the other side has already pro-actively closed the connection. Figure 6 captures the details of CDP 2-way handshake connection termination scenario.

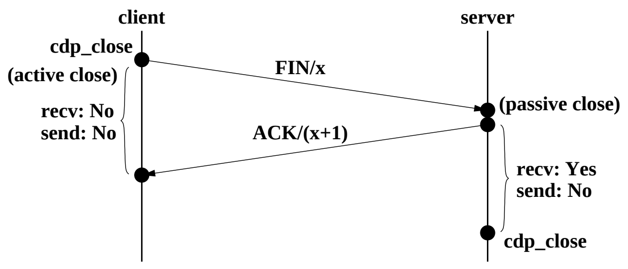


Figure 6: 2-way handshake CDP connection termination

In addition to the 2-way handshake connection termination schema, CDP has also designed the connection keep-alive mechanism in the protocol. This is used to differentiate the two cases that happen on a CDP client and CDP server. Suppose they have already established the connection. If the server has not received any packet from the client for a very long period, there may exist two possibilities:

- the client died a long time ago and did not get a chance to sent out any message;
- the application on the client side happened to have no data sent to the server during that period;

The symptoms of these two cases are the same when observing from the server side. If we have no way to differentiate these two cases, we can not release the resources occupied by that CDP connection. Gradually, the resources on the server machine will be exhausted. The solution to this problem is to make some "noise" when CDP client have no data sent to the server. This is what the *PIN* packet is supposed to do. A *PIN* packet is sent to the CDP server by a CDP client when the client finds that it has not sent any data to the server in the recent 60 seconds. The CDP server maintains a timer (expires in 300s) for the client. Every time the server receives any useful packet or a *PIN* packet from the client, the timer will be refreshed. If the timer expires, the server considers that the client is dead, therefore, reclaiming all resources that services that client.

4.4 CDP Socket and Programming Interfaces

A CDP socket is an end-point of a bi-directional communication link in CDP protocol. It is also the fundamental data structure used in the CDP socket programming interface, which is similar to the Berkeley sockets API. CDP socket maintains all the resources and information that are needed by CDP connection, including receiving buffer and sending buffer, the slide window, the connection state, etc. CDP socket programming interfaces provide a similar programming environment for the network software developers. Several of them support both *blocking* and *nonblocking* models.

4.5 CDP Routing and I/O Node Fault Tolerance

As shown in Figure 2 and Figure 3, if a compute node wants to send a packet to a host node, it is first sent to an I/O node and then forwarded by that I/O node to the destination host node. In the C64 system, every compute node has more than one I/O nodes as its *bridges* to the host side. one of these I/O nodes is treated by the compute node as its default I/O node. This information, i.e. the *I/O node set* and *default I/O node*, is configured at system boot-up time for every compute node and host node. For each CDP packet that need to be sent across the back-end and front-end is first sent to the default I/O node. If the default I/O node crashed, or if the packet is lost on the half way, the other I/O node in the *I/O node set* will be chosen for retransmission. CDP will try different I/O nodes for different retransmissions. Meanwhile, if retransmissions happened quite a lot of times, CDP will choose a new I/O node from the *I/O node set* as the default I/O node. This is a simple but effective method to automatically tolerate those malfunction I/O nodes.

5 Performance

The performance of CDP is very important to the overall performance of C64 supercomputer system. Because applications running on C64 supercomputer are supposed to process huge amount of data, and all these data are delivered to the compute nodes over CDP. In order to analyze CDP performance, we conducted several experiments on the C64 simulator, as well as on the real machines. All of these experiments were performed on the Gigabit Ethernet, not the 3D-mesh network. Because, as we mentioned in section 2, the bandwidth of the 3D-mesh that connects C64 nodes is 32Gbps, much bigger than the bandwidth of Gigabit Ethernet. So, CDP running on the Gigabit Ethernet segment has more possibilities to become the bottleneck of the CDP connection. Therefore, it deserves more attention.

In the following two subsections, we will present some performance data of CDP with comprehensive analysis. In addition, we will compare CDP's performance with other protocols like TCP and UDP to see its advantages and disadvantages.

5.1 Performance Measurement on Cyclops-64 Simulator

Since the C64 hardware is still under development, we use the C64 simulator to conduct our experiments. The C64 simulator, named as Functionally Accurate Simulation Toolset (FAST), is designed for the purpose of architecture design verification as well as early system and application software development and testing. Although not cycle accurate, it catches most of the important C64 architecture features [3].

Before we present the experiment results of CDP, we first introduce the model we used to estimate CDP performance. Figure 7 shows the simple schema of how CDP works.

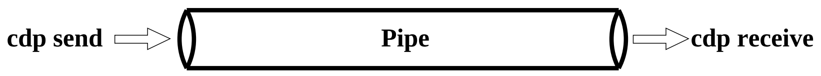


Figure 7: CDP Performance Analysis Model

The protocol layers under CDP can be viewed as a pipe which is transparent to the senders and receivers. The sender dumps its data into the pipe and the receiver gets data from the pipe. The **pipe** is just an abstraction of the cable (or switch, hub), NIC, driver, and everything else under CDP. CDP has no control over the pipe, let alone the pipe's performance. The only thing we can do is measure how many cycles it will take the sender to send data to the pipe by CDP, and how many cycles it will take the receiver to receive the data from the pipe through CDP. On the sender side, we assume when the sender wants to send out some data, the pipe is always empty. On the receiver side, we assume the pipe is always saturated with CDP packets. In other words, anytime the user wants to receive some data, there is always a CDP packet in the pipe. We believe these assumptions make analysis easier without losing any correctness.

It can be inferred from this analysis model that the **send** and **recv** operations are pipelined. Since the longest pipeline stage decides the upper bound of the maximum throughput of the pipeline, we are more interested in the slowest stage in transferring a CDP packet from the sender to the receiver. There are three stages that count:

- Send: in this stage, CDP on sender side creates a CDP packet for user datagram and deliver the CDP packet to the lower network layer.
- Recv: in this stage, CDP receiving daemon on receiver side gets the CDP packet from lower network layer and puts it into the receiver's receiving buffer.
- Copy: in this stage, the receiver obtains the CDP packet from the receiving buffer and copies the user data into the user buffer.

We measured the number of cycles used in each of these three stages. All the measurements were performed with a maximum CDP packet size of 1496 bytes (header + user data). The cycle numbers are shown in the table below:

Send	5644 cycles
Recv	4518 cycles
Copy	4584 cycles

We can see from the table that the slowest pipeline stage is the **Send** stage. It costs 5644 cycles when sending 1472 bytes of user data. As we have mentioned in section 2, the clock rate of the C64 chip is 500MHz. So, CDP's bandwidth, without considering the hardware and software limitations, is about 1Gbps ($0.5GHz * 1472Bytes * 8/5644$). This number does not mean that CDP can retain all the capacity of the Gigabit Ethernet link. It just indicates that CDP can not be a bottleneck in the C64 communication links.

5.2 Performance Comparison: CDP, UDP, and TCP

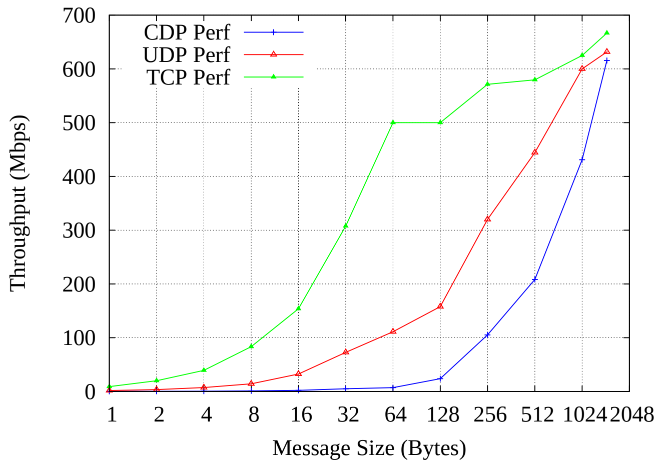


Figure 8: CDP, UDP, and TCP throughput under different message sizes: 1-1472 bytes

Besides the performance estimation derived from the C64 simulator on C64 side, we measured CDP throughput on real machines for host side, i.e. two Linux boxes connected via a Gigabit Ethernet link. In our experiment a client program tries its best to send CDP packets to a server running on the other machine as fast as possible. Meanwhile, the server program tries to receive all the packets coming from the client program. On the same test-bed, we also measured the throughput of UDP and TCP under different message sizes with the same program, but using the corresponding protocol.

Figure 8 shows the throughput of CDP, UDP, and TCP under different message sizes. According to the figure, CDP's maximum throughput is 615Mbps, which implements 61.5% of the physical channel capacity. While for the other two protocols, UDP's maximum throughput is 630Mbps, and TCP's maximum throughput is 666Mbps. All of these three protocols reach their peak number at message size 1472 bytes. We stopped at 1472 because, under the limitation of Ethernet MTU (1514 bytes), this is the biggest user datagram that can be sent by CDP. Compared with TCP and UDP, CDP's peak throughput is a little bit smaller. However, consider that CDP is a user level protocol, which means more memory copies and more kernel-user context switches, this is still a decent performance number.

At small message sizes (<1024bytes), CDP's throughput is far below UDP and TCP. This is because the overhead of CDP, when transferring small packets, is much bigger than UDP and TCP. But this is not a big problem. The applications running on C64 supercomputer are compute intensive, not I/O intensive or interactive intensive. There are not many small size messages exchanged between the C64 nodes and host nodes. Large portion of the communications are bulk data transfer. So, most of the time, CDP works under its peak performance configuration. This guarantees that CDP will not become the performance bottleneck of the whole C64 supercomputer system.

5.3 CDP round trip latency

As we mentioned in the previous section, we don't have a stringent round trip latency requirement for CDP. Therefore, we didn't perform experiments to compare the round trip latency of CDP, UDP, and TCP. The performance data in Figure 8 is enough as an indicator of the CDP performance.

6 Related Work

High performance communication networks for supercomputers has been an important research topic for quite a long time. Many researchers have performed significant amount of research on how to increase its bandwidth and speed, and have obtained great achievements. Nowadays, there are quite a lot high performance communication network products and standards, like InfiniBand [7, 8], HIPPI [14, 13], Myrinet [2], Quadrics [9], and SCI. A significant number of TOP500 [6] supercomputers have used one of these products as their communication framework. However, there are still some supercomputers just use off the shelf network products, instead of those special designs. One example is IBM BlueGene/L supercomputer [1, 12]. It uses a Gigabit Ethernet to connect its back-end and front-end, and runs a TCP/IP stack on top of it. This is because using off the shelf products makes the time-to-market faster. The building block of BlueGene/L is a PowerPC, a general purpose RISC processor. It is easy to run a Linux kernel on it. Therefore, supporting TCP/IP on it is not a big problem. While in the C64 supercomputer system, C64 chip is not suitable for running a full functional OS kernel, let alone a "heavy weight" protocol stack such as TCP/IP. Therefore, we decided to design a "light weight" protocol, CDP, as its communication infrastructure. CDP achieved the expected performance requirement. Its peak performance is comparable to that of TCP and UDP. There is still a lot of room for improvement in CDP. In our future work, we will try to make CDP support jumbo frame, which has 9000 bytes MTU. This will greatly improve the CDP throughput when performing bulk data transfer.

References

[1] George Almasi, Ralph Bellofatto, Jose Brunheroto, Calin Cascaval, Jose G. Castanos, Luis Ceze, Paul Crumley, C. Christopher Erway, Joseph Gagliano, and et al. An Overview

of the Blue Gene/L System Software Organization. In *Proceedings of the Euro-Par '03 Conference on Parallel and Distributed Computing*, 2003.

[2] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, 1995.

[3] Juan del Cuvillo, Weirong Zhu, Ziang Hu, and Guang R. Gao. FAST: A Functionally Accurate Simulation Toolset for the Cyclops-64 Cellular Architecture. In *Workshop on Modeling, Benchmarking and Simulation (MoBS'05) of ISCA'05*, Madison, Wisconsin, June 2005.

[4] Juan del Cuvillo, Weirong Zhu, Ziang Hu, and Guang R. Gao. Towards a Software Infrastructure for Cyclops-64 Cellular Architecture. In *HPCS 2006*, Labroda, Canada, June 2005.

[5] Phil Dykstra. Gigabit ethernet jumbo frames, and why you should care, December 1999.

[6] http://www.top500.org/. Top500 list for november 2005, November 2005.

[7] InfiniBand Technology Association. Infiniband architecture specification, release 1.0, October 2000.

[8] InfiniBand Technology Association. Socket direct protocol specification v1.1, 2002.

[9] Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46–57, / 2002.

[10] Richard Stevens. *TCP/IP Illustrated, Volume 1&2.* Addison Wesley University Press, 1994.

[11] Richard Stevens, Bill Fenner, and Andrew Rudoff. *Unix Network Programming: The Sockets Networking API, Volume 1.* Addison Wesley University Press, 2004.

[12] The BlueGene/L Team. An Overview of the BlueGene/L Supercomputer. In *Proccedings of ACM Supercomputing Conference*, 2002.

[13] D. Tolmie. High performance parallel interface (hippi). In A. Tantawy, editor, *High Performance Networks - Technology and Protocols*, pages 131–156. Kluwer Academic Publishers, Norwell, 1994.

[14] D. Tolmie and J. Renwick. HIPPI: Simplicity Yields Success. *IEEE Network - The Magazine of Computer Communications*, pages 28–32, January 1993.