A High Performance Communication Subsystem for PODOS

Sudharshan Vazhkudai Tobin Maginnis

Department of Computer and Information Science

University of Mississippi

chucha@john.cs.olemiss.edu
ptm@pix.cs.olemiss.edu

Abstract

PODOS is a performance oriented distributed operating system being developed to harness the performance capabilities of a cluster computing environment. In order to address the growing demand for performance, we are designing a Distributed Operating System (DOS) that can utilize the computing potential of a number of systems. Earlier clustering approaches have traditionally stressed more on resource sharing or reliability and have given lesser priority to performance.

PODOS adds just four new components to the existing Linux operating system to make it distributed. These components are a Communication Manager (CM), a Network Manager (NM), a Resource Manager (RM), and Global Interprocess Communication (GIPC). This paper addresses the communication mechanism in PODOS.

In any distributed environment, communication appears to be the performance bottleneck. Thus, in PODOS, we have implemented a high-speed communication subsystem that short circuits the network protocol stack, and further performs packet multiplexing (Transmission-Groups) across multiple network interfaces thereby achieving a two-fold performance gain.

In this paper we discuss the high-performance communication subsystem in PODOS and further analyze the performance gain achieved by comparing the variants of the PODOS protocol with traditional networking protocol.

1.0 Introduction

A distributed operating system (DOS) is basically the cooperation among a group of machines interconnected by a network such that the group of machines appear to the user as a single operating system. With distributed operating systems, users are not aware where their files are stored; nor are they aware that their programs may be executed by remote machines. All resources within the network are managed in a global fashion using global mechanisms rather than local mechanisms [1].

A group of machines could cooperate for a variety of reasons. A few of them are (1) Resource Sharing, (2) Performance Enhancement, (3) Reliability, (4) Fault Tolerance and (5) Transparency [1]. Tens of distributed operating systems have been designed and implemented with various goals. Most distributed system designs are willing to compromise on performance. On the other hand, systems that are designed to be performance oriented make no attempt to provide a single system image. They provide a high-performance computing environment (e.g., Beowulf [2], Condor [3], etc). High-performance computing environments are designed to solve one class of problems, whereas a PODOS is designed as a general high-performance computing solution.

With these issues in mind, we are designing a distributed operating system that is performance oriented (PODOS) [4]. As a secondary design goal, PODOS provides a resource sharing environment.

PODOS is the interaction of two or more **monolithic Linux** [5] machines. The PODOS design has a number of key performance benefits. A few of these are:

1. PODOS builds upon a highly robust and performance oriented monolithic Linux kernel.
2. PODOS adds very few components to the basic Linux operating system, thereby maintaining a simple design.
3. Each of these components is designed to achieve high performance. For example, the CM uses a custom high-speed protocol and the NM is tightly glued to Linux's filesystem to speed up remote file fetches.

In the following sections we give an overview of the PODOS design, the network topology that help building an efficient system. We then explain the communication subsystem in PODOS in detail and finally present a performance analysis with a study of the preliminary results.

We then explain the communication subsystem in PODOS in detail and finally present a performance analysis with a study of the preliminary results.

2.0 The PODOS Cluster

PODOS is an experimental Linux cluster being developed at the University of Mississippi. The primary project objective is to explore the performance capabilities of a clustering system, but at the same time incorporate the basic features of any distributed operating system. Furthermore, we try to minimize the additions to the basic operating system [6]. Each node in the PODOS cluster is a monolithic Linux kernel. (We have chosen a monolithic kernel against a microkernel because the monolithic paradigm suits our goal of high performance much better.) Our design involves adding four components to the basic operating system to support distributed features. These include

- Communication Manager
- Resource Manager
- Global IPC
- Network Manager

Let us look at these components briefly.

2.1 Communication Manager

The Communication Manager (CM) handles remote communication in the PODOS. Responsibility of the CM is to interact with peer CMs in the cluster. The CM transmits and receives frames to/from other hosts through a Local Area Network (LAN). The CMs implement a minimal send and receive protocol to the Network Manager (NM), Resource Manager (RM) and Global Inter Process Communication (GIPC). The CM accepts remote host addresses and messages from higher-level processes (e.g., local-to-remote GIPC, NM-to-NM and RM-to-RM communication). The CM will combine these addresses, messages, and RM status information with an error-free protocol to send the message to its remote counterpart on the target machine [6].

2.2 Resource Manager

The Resource Manager maintains global system state information. Each RM in the cluster maintains information about every other node. Although resource management implies decision making algorithms (processor allocation algorithms, etc.), the RM is designed to be as simple as possible for performance. Thus, the RM provides the primitives required to build unique resource management strategies. Typically, a program loader will query the RM concerning a remote machine. The purpose of this query is to discover the number of processes, willingness to accept remote queries, etc. and based upon these queries, assign processes to processors. The RM will make use of the CM to transmit and receive system information in a broadcast or a piggybacked fashion among its peers [6].

2.3 Global Interprocess Communication

The GIPC provides a mechanism with which processes can communicate in PODOS. It allocates a global pid (GPID) for every process in PODOS so that processes can be uniquely identified. The GPID of each process is the PID (given to the process in the local machine) and the IP address of the machine (to record the fact that the process originated from this particular machine). GIPC further provides communication primitives for processes to communicate among themselves [6].

2.4 Network Manager

The Network Manager will interface with two Linux basic operating system (OS) components: the file manager and the process manager. These basic OS concepts will be extended to support the notion of a PODOS. File and process management will be able to recognize non-local file names and invoke the NM. NM local-to-remote requests will be carried out by simply invoking the CM [6].

Let us look at the network architecture in PODOS.

3.0 The PODOS Network Topology

PODOS has a special network topology that aids in implementing an efficient and a high-performance communication mechanism [7]. Let us look at a model of the PODOS network topology. Figure 1 gives an overview of the PODOS network architecture.

PODOS uses an Ethernet network interface as the communication media. Each node represents a machine where node 1, node 2, ... is machine 1, machine 2, etc. PODOS can use as many interfaces as the system supports. Our current implementation has three Ethernet interfaces, namely `eth0`, `eth1`, and `eth2`. Each node is connected to the public subnet, W.X.Y.0, through the primary interface, `eth0` and thus gets an IP address, W.X.Y.n, where n is between 1 and 254. Further to this, each node has two more interfaces, `eth1`, and `eth2` which are connected to private subnets 192.168.1.0 and 192.168.2.0 respectively and thus get two more IP addresses, 192.168.1.n and 192.168.2.n. Thus each machine in PODOS is a multihomed host. Private subnets in PODOS are configured in such a way that machines can communicate only through the interface pairs `eth0-eth0`, `eth1-eth1`, and `eth2-eth2`. More specifically, packets on `eth1` can only go to `eth1` on another machine. This can be easily observed from Figure 1, since the network 192.168.1.0 connects only `eth1's` in all machines. Similarly, the network 192.168.2.0 connects `eth2's`. The Transmission-Groups feature uses these interfaces in a round-robin fashion [7].

4.0 Communication in PODOS

Communication in PODOS is handled by the CM. Higher-level DOS layers (RM, NM, GIPC, etc.) rely on the CM for packet transmission and reception. The CM in each node uses a specialized protocol to talk to its neighbors. The CM comprises of the following components:

1. PODOS-packet protocol
2. Communication Descriptor Table (CDT)
3. Transmission-Groups

Figure 2 demonstrates the relationship among these subsystems. To the left in Figure 2 is the traditional network protocol stack, the Open Systems Interconnection (OSI) [8] model of networking. To the right, in Figure 2, are the

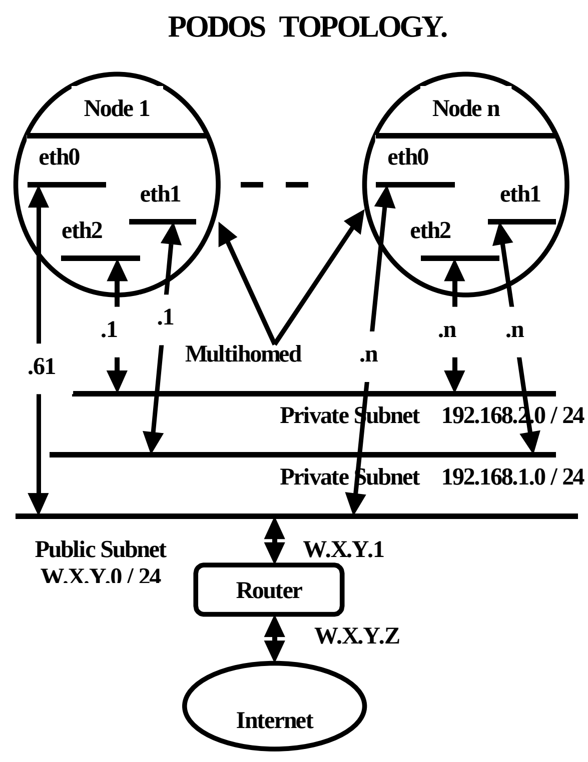

Figure 1

PODOS components. The communication subsystem is depicted in more detail. The CM comprises of three components namely the CDT, the Transmission-Groups and the PODOS-packet protocol. The CDT is an interface for higher-level PODOS layers. Higher-level layers typically make entries with the CDT. A

Transmission-Group algorithm is applied to each entry in the CDT. And finally the PODOS-packet protocol transmits the packet using the datalink layer of the OSI model. The RM, NM and GIPC use the CM to communicate with peer entities in the PODOS cluster. We will now look at the various components of the CM (Figure 2).

4.1 PODOS-packet Protocol

The PODOS-packet protocol is at the very bottom of the CM. It provides primitives for transmitting and receiving PODOS-packets. The PODOS packet protocol has evolved from a very rudimentary structure [9]. In this section we will describe the protocol briefly.

Since our primary goal is performance and the typical DOS bottleneck is network bandwidth, we needed an efficient communication mechanism that could speed up packet transmission and reception. We needed something other than the traditional networking protocol (left side of Figure 2). The traditional protocol consists of several layers and each layer has its own headers and error checking. However, traditional protocols also contain a lot of detail to accommodate many types of network configurations [8]. Thus, we designed and implemented a distributed operating system packet (PODOS-packet), which bypasses the traditional network protocol stack [4].

In Figure 2, we can see how the CM resides above the datalink layer (Ethernet driver). From Figure 2 it is also evident how the CM has moved away from the traditional network layers and has less overhead. Our approach here is to have the CM interact with the datalink layer (Ethernet driver) to transmit and receive packets. In short, the CM will have to transmit and receive packets that bypassed the network protocol stack. This would mean that the CM packets would have to have a separate protocol ID [9], one that is different from IP, ICMP etc.

Now let us look at the CDT.

4.2 The Communication Descriptor Table

The CDT is the CM's interface to the other PODOS layers. The CM shields the higher-level PODOS layers by performing all the intricate details involved with packet transmission. The higher-level PODOS protocols register their packets with the CDT. The CM picks up packets from the CDT and transmits the packets. The CM also uses the CDT to construct a virtual circuit, which helps in streamlining communication between peer components [10]. The CM also maintains a simple timeout mechanism by which it can keep track of errors and retransmissions. Thus, the CM maintains a simple and elegant protocol for packet delivery [6].

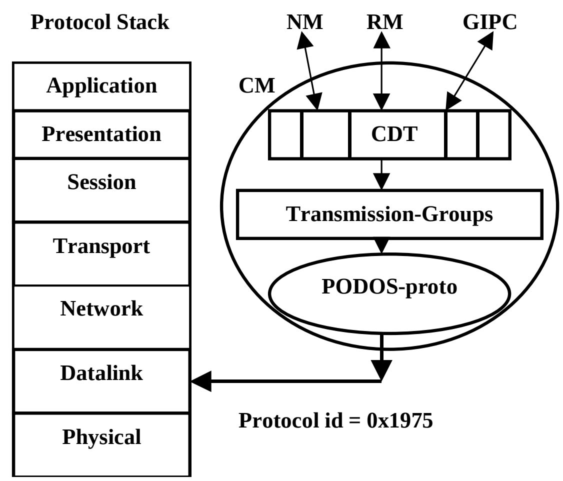

Figure 2

4.2.1 The CDT Structure

A fully functional communication descriptor table has marked a milestone in the evolution of the CM. The higher-level PODOS layers fill in a CDT entry and invoke the CM. The CM picks up the PODOS-packet from the CDT and transmits it. A typical entry in a CDT is described in Table 1 [10].

4.2.2 Classes of CDT entries

Communication descriptor table entries can be classified into two categories based on the time spent in the table. They are:

- **Ephemeral**

These entries are short duration entries and release the CDT slot once the packet has been transmitted. An example of such a request that is short-lived is the RM query. The RM periodically broadcasts system information to all the nodes in the cluster. It does not wait for the arrival of any packet. Whenever a broadcast message arrives the CM invokes the RM. Thus the RM would register its packet in the CDT and invoke the CM. Once the CM transmits the packet, it releases the CDT entry corresponding to the RM query [10].

- **Virtual Circuits (VC)**

When a higher-level PODOS protocol wishes to communicate with its peer for a longer duration, it usually requests the CM to establish a virtual circuit. The virtual circuit is nothing but a {cdt-active-index, cdt-passive-index} pair. When a higher-level PODOS layer (at the active end) requests such an entry, the CM makes a CDT entry and transmits the packet. The peer CM (at the passive end) receives the packet, makes a CDT entry and then invokes the appropriate layer. Henceforth, any communication between these two layers would go through this virtual circuit. This helps in streamlining the subsequent flow of packets [10].

Now let us look at the PODOS-packet structure.

4.3 The PODOS-packet Structure

Table 2 describes the PODOS-packet structure and also illustrates the importance of each field [10].

Now let us look at how PODOS handles Transmission-Groups.

5 Transmission-Groups

Transmission-Groups is a suite of algorithms that multiplex packets across multiple network interfaces. Transmission-Groups exploit parallel networks connected among distributed computers and thereby match the external aggregate network bandwidth with internal memory bandwidth. The CM employs Transmission-Groups to achieve further performance gains over the PODOS-packet protocol. The PODOS cluster is configured in such a fashion that the suite of Transmission-Group algorithms can exploit the network architecture. Each node in PODOS has been configured with multiple Ethernet interfaces (Figure 1). This increases the LAN bandwidth and effectively utilizes the communication media. Once the higher-level PODOS layers register their packets with the CDT, the CM decides which interface the packet should go through. This decision-making is Transmission-Groups [7].

Struct comm_desc_tab {	
struct PODOS_pkt pkt;	The PODOS-packet structure is embedded in the CDT
char *to_pid[8];	Higher-level layers specify the process id in the remote machine to whom the packet is sent. This could be a group of processes to support group communication.
int to_pid_cnt;	The count of the processes referencing this CDT entry.
int need_reply;	Whether the CDT entry should be held for a longer time. For example, RM requests are typically short lived as compared to NM or GIPC requests.
Char *hostname[20];	The name of the machine to whom the packet is being sent. Could be list of machine names (for group communication).
int host_cnt;	The count of machine names.
Struct interface if;	The interface structure (will be discussed later).
};	

Table 1

struct PODOS_pkt {	
char from_pid[8];	The global pid of the process from whom the packet is originating.
char to_pid[8];	The global pid of the process to whom the packet is being sent.
char ctrl_info;	This is a one byte field that is used by the CM to differentiate NM, RM and IPC packets. It uses the least significant 3 bits. Higher-level protocols use the most significant 5 bits. For example, the NM would use it to differentiate open, read, write, close, etc.
int cdt_active_ind;	The CDT index at the active end (the end that originated the connection).
int cdt_passive_ind;	The CDT index at the passive end (the end that is accepting the connection).
int wake_active_passive;	Since the CM code is the same for active and passive ends, it needs to know which process to wake up.
int length;	The length of the data being sent.
char data[1400];	The data.
};	

Table 2

Transmission-Group algorithms are applied only to PODOS-packets in the cluster. PODOS-packets can travel on any one of the three interface pairs (The following denote active end interface-passive end interface: `eth0-eth0`, `eth1-eth1`, and `eth2-eth2`). Whereas regular network traffic (IP, ICMP, IGMP packets) continue to go through `eth0` only. If we wish to multiplex those packets too, then we would have to employ an algorithm similar to Transmission-Group suite at a higher level in the network protocol stack.

Now let us look at Transmission-Groups in more detail.

5.1 Transmission-Group Suite

The Transmission-Groups is a suite of algorithms, each based on a different goal. We provided a set of routines (round-robin, load based) so that one may select the best algorithm that suits their requirements. The Transmission-Group routine is held as a function pointer in the interface structure (described in Table 3) in the CDT entry. When a higher-level layer wishes to transmit a packet it makes an entry in the CDT and invokes the CM. The CM initiates the Transmission-Group algorithm by invoking the function pointer. The function pointer is set during system initialization.

The following section describes a simple round-robin Transmission-Group algorithm that multiplexes virtual-circuits across multiple interfaces.

5.1.1 Round-Robin with VC multiplexing

PODOS employs a simple round-robin mechanism to multiplex packets across multiple network interfaces. But since, multiplexing packets would result in ordering and sequencing issues, PODOS multiplexes virtual circuits. This implies that all packets resulting from a virtual circuit would be transmitted on a particular interface pair. Inshort, PODOS employs a **"1-Interface-for-all-Packets-in-a-VC"** rule. This ensures that packets reach their destination in order and saves us the trouble of ordering them. Ephemeral packets (packets resulting from an Ephemeral CDT entry, a RM packet) would be transmitted on the current interface (maintained by the Transmission-Group routine). Let us look at this algorithm in more detail.

Implementation

The round-robin algorithm maintains an interface counter which it increments for every virtual circuit. This counter is reset once it reaches IF_MAX. The algorithm differentiates virtual circuits from ephemeral entries by looking at the control byte of the packet. Let us discuss the implementation of VC multiplexing with reference to a NM remote file-write. The local NM wishes to write a file to another node by contacting its peer entity in the other node. This NM file-write request is characterized by an open call, a sequence of write calls and then a close call. This is a typical virtual circuit. Once the NM decides to write a file to another node, it builds a PODOS-packet, makes a CDT entry and invokes the CM. Embedded in the CDT entry is the interface structure. Below is Table 3 describing the interface structure.

struct interface if {	
char active_interface[6];	The hardware address of the interface at the active end.
char passive_interface[6];	The hardware address of the interface at the passive end.
Int (*elect_interface)(struct device *);	Transmission-Group algorithm held as a function pointer. This is the function that round-robin's packets.
};	

Table 3

Having the interface structure in the CDT entry is the key to the algorithm. Each entry will have such a structure and packets can be transmitted to/from the address specified in the interface structure. Once an entry is made in the CDT, the Transmission-Group routine corresponding to that CDT entry is launched by calling cdt[i]->elect_interface(), where i denotes the particular entry. The Transmission-Group algorithm maintains a simple round-robin strategy with which it multiplexes virtual circuits.

Now, when the higher-level layer (NM) registers subsequent packets in the CDT, the CM just inspects the interface structure of the CDT entry and transmits the packet to the passive_interface address on the active interface address. Thus, with this approach, the Transmission-Group routine is invoked only once per virtual circuit (or CDT entry) and all packets belonging to one virtual circuit are transmitted on the same interface pair. Thus, the algorithm makes a basic assumption that **"virtual circuits have the potential to generate a lot of traffic on the interface"**. Hence, the algorithm attempts to uniformly distribute virtual circuits across the three interfaces. Ephemeral entries do not contribute much to the interface load and thus the algorithm does not worry about such packets [7].

6.0 Performance

In this section we analyze the performance capabilities of PODOS protocol and its variants. We further compare them with the traditional TCP/IP protocol and present the preliminary results based on this analysis.

Each experiment:

- Computes the average RTT of packets using 10 sets of 100 packets each.
- It discards the maximum and the minimum set and then computes the average.
- Transmits each packet with a 1 sec delay.

Let us look at each experiment in detail.

Experiment #1

In this experiment we compare the RTT's of a simple PODOS protocol with a typical socket read/write call of the TCP/IP. The experiment transmits 10 sets of 100 packets each. Each packet is 64 bytes long and is transmitted with a 1 sec delay. Table 4 depicts the results. Figure 3 depicts the performance gain achieved. From the graph it is evident that

PODOS protocol out-performs traditional networking protocol and is almost twice as fast. The RTT difference is substantial

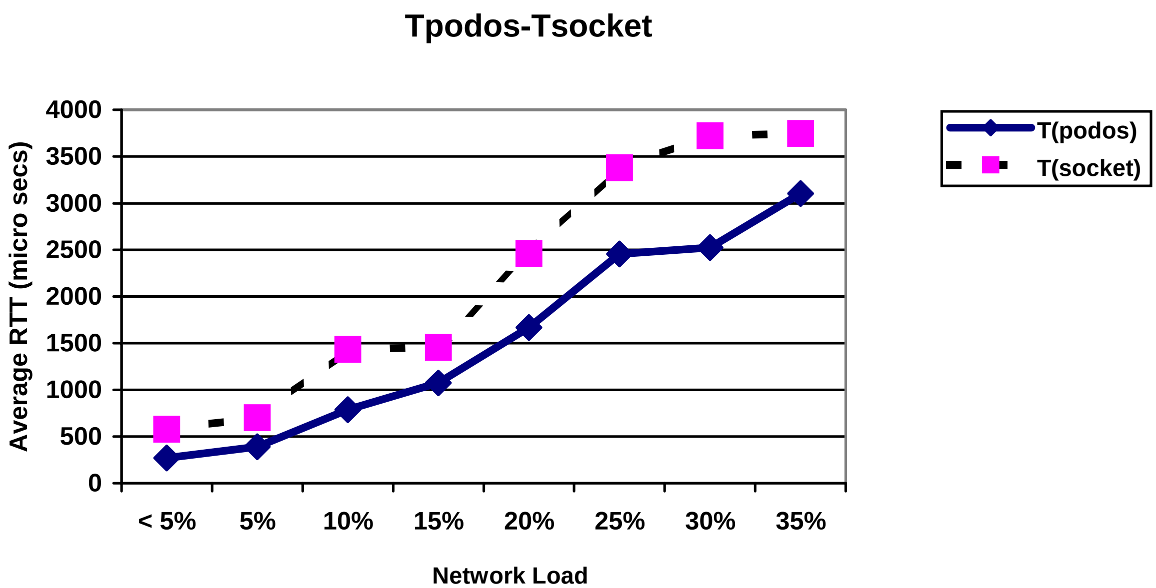

Figure 3

In order to study the behavior of these two protocols, we conducted a series of experiments, each with a different objective, and measured the average round trip time (**Average RTT**) in each case. All our experiments involved comparing variants of the PODOS protocol with the traditional networking protocol under various network loads [11].

at higher network loads. The graph depicts RTT's for loads upto 35% - 40%. This is because 35% - 40% network load is a substantial network load and the system is saturated at that load. In normal circumstances, the multiprogramming level would further increase the RTT thus making the load substantial.

Network Load	**T(podos)** Micro secs	**T(socket)** Micro secs
< 5%	271	578
5%	389	702
10%	787	1435
15%	1073	1456
20%	1669	2460
25%	2454	3381
30%	2524	3723
35%	3104	3746

Table 4

The RTT for the PODOS protocol is of the form:

$$T_{podos} = T_{build-podos-pkt} + T_{xmit} + T_{propagation}$$

$$T_{xmit} = T_{alloc-net-buff} + T_{build-Ethernet-pkt} + T_{drv-xmit}$$

$$T_{propagation} = T_{forward-propagation} + T_{backward-propagation} + T_{filter}$$

to filter out the PODOS packet and reply. The filter time is the time taken by the filter, installed in the datalink layer, to extract PODOS packets. Of these, the propagation time is entirely dependent on the network load and the rest depend on the multiprogramming level of the system.

Experiment #2

In this experiment we compare the RTT's of three protocols, the TCP/IP, the simple PODOS protocol, and PODOS with Transmission-groups. The PODOS with Transmission-groups is a variant of the PODOS protocol that transmits PODOS packets across multiple interfaces. In this case we vary the network load only on the primary interface and

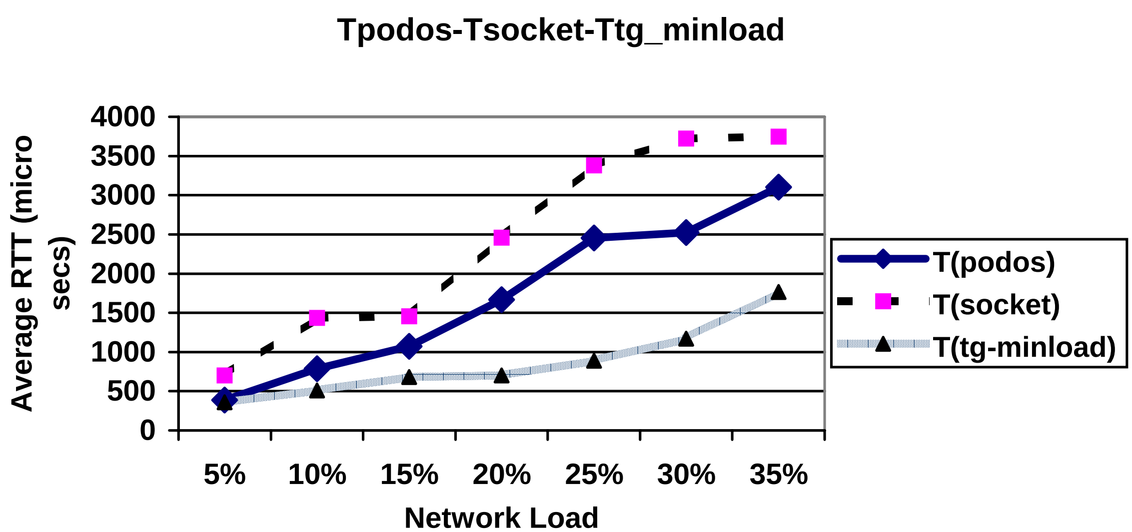

Figure 4

The RTT for a PODOS packet is the sum of the times taken to build a PODOS packet, build an Ethernet packet, the driver transmit time (the time taken by the driver to place the packet on the physical media after its transmit function has been invoked), and the propagation time. The propagation time includes forward and backward propagation and the filter time. The forward propagation is the time taken for the packet to reach its destination after it has been placed on the physical media. The backward propagation time includes the time taken by the passive end

maintain minimal load on the other two interfaces. The Transmission-Groups based communication is a simple divide-and-conquer strategy. It divides the tasks in hand equally among interfaces thereby sharing load and enhancing system throughput. Table 5 depicts the results.

Figure 4 is a graphical representation of the results of this experiment. The **T(tg-minload)** is the average RTT when transmitting packets across multiple interfaces, with varied load on the primary interface and minimal load on the other 2 interfaces. In Figure 4, we can observe the drastic performance gain with Transmission-Groups. We can see how the Transmission-Groups based PODOS protocol performs consistently at higher loads. It is almost 3 times faster than the regular networking protocol. Transmitting PODOS packets across multiple interfaces reduces the load on a particular interface and thus decreases the RTT of packets.

Net	T(podos)	T(socket)	T(tgminload)
Load	Micro secs	Micro secs	Micro secs
5%	389	702	359
10%	787	1435	510
15%	1073	1456	679
20%	1669	2460	700
25%	2454	3381	887
30%	2524	3723	1172
35%	3104	3746	1766

Table 5

The RTT is of the form:

$$T_{rtt} = T_{podos} + T_{tgcomm}$$

The RTT is the time taken to transmit a regular PODOS packet plus the time taken by the Transmission-Group algorithm. The Transmission-Group algorithm usually takes around 100-150 micro seconds, which is very less overhead when compared to the performance gain.

this case we vary the network load on all interfaces. Table 6 depicts these results.

Net	T(podos)	T(sameload)	T(tgminload)
Load	Micro secs	Micro secs	Micro secs
5%	389	386	359
10%	787	537	510
15%	1073	968	679
20%	1669	1105	700
25%	2454	1155	887
30%	2524	1239	1172
35%	3104	1935	1766

Table 6

We can observe the stability of the T(sameload) protocol at high loads, 20%, 25%, and 30%. The RTT varies by very meager amounts. Figure 5 depicts these results. From Figure 5 it is evident that even when the load on all interfaces is approximately the same, it is profitable to distribute packets among interfaces rather than transmitting them on a single highly loaded interface.

Let us consider the 30% load case. The primary interface was already 30% loaded and transmitting another 1000 packets resulted in a RTT of 2524 micro seconds. This loads further a highly loaded interface. Whereas dividing the 1000 packets among 3 interfaces and transmitting 333 packets on each interface resulted in an RTT of 1239 micro seconds which

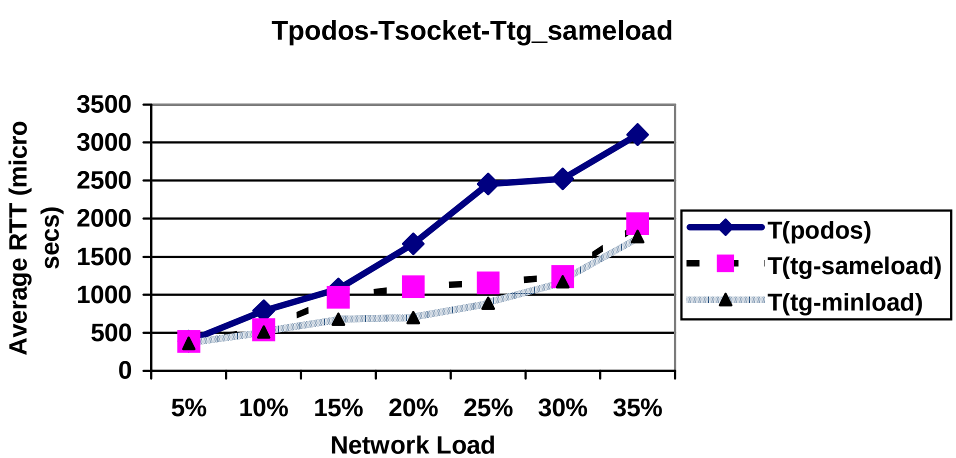

Figure 5

Experiment #3

This experiment is similar to the previous one except that we compare only the PODOS variants namely, the simple PODOS, the PODOS with Transmission-Groups (minimum load on secondary and tertiary interfaces), and the PODOS with Transmission-Groups (approximately same load on all interfaces). In

is less than half the RTT of a simple PODOS. Thus Transmission-Groups is beneficial even in a heavily loaded case.

From the above three graphs we can derive the following relation that holds under all network loads:

$$T_{tgcomm - minload} < T_{tgcomm - sameload} < T_{podos} < T_{socket}$$

The relation that is of more interest to us is the Transmission-Groups at approximately same load and the socket. From the above graphs it is obvious that the RTT of Transmission-Groups would be much better than that of a socket read/write.

Experiment #4

This experiment is to measure the connection establishment time of the PODOS protocol. We further compare this to traditional socket connection establishment in TCP/IP. Connection establishment in PODOS implies making a CDT entry at the active end, transmitting the packet, making a CDT entry at the passive end and responding back to the active end. This signifies a virtual circuit. We further time the read and write of the first 1500 byte packet through this circuit. In traditional socket connection this involves the following system calls: socket(), bind(), and connect(). Table 7 depicts the results of this experiment. From Table 7 we can observer that T(cdt) is very stable and consistent at network loads. Figure 6 depicts these results.

Net Load	T(cdt) Micro secs	T(socket-conn) Micro secs
< 5%	2457	3255
10%	2800	3378
20%	3041	4000
30%	3401	4742
40%	3653	5000

Table 7

The connection establishment in PODOS includes the server setup time too, whereas in typical socket connections, the server is assumed to be listening for a connection and thus server setup time is ignored. And server setup time typically takes around 500-800 micro seconds. Further, the results above depict connection establishment without Transmission-Groups. Thus, in reality, the performance gain is much more.

The RTT is of the form:

$$T_{rtt} = T_{podos} + T_{cdt}$$

A typical cdt entry creation, under an optimal multiprogramming level takes around 200 micro seconds which is lesser than standard connection setup time.

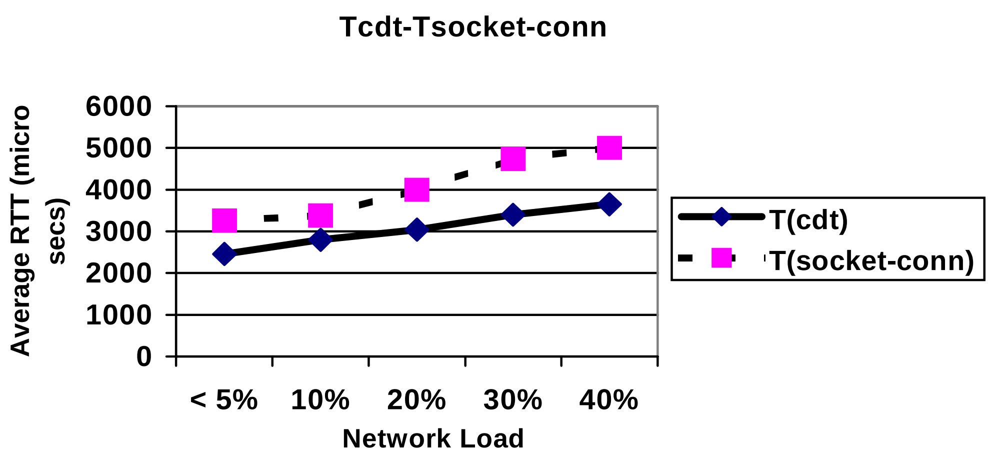

Figure 6

7.0 Conclusions

In this paper we have dealt with the communication aspect in PODOS. We have described a custom communication protocol that employs a round-robin Transmission-Groups mechanism to multiplex packets across multiple network interfaces and provides a communication descriptor table as an interface for other PODOS layers such as NM, RM and GIPC. We then presented a detailed performance analysis of the PODOS communication protocol. We analyzed the performance of the variants of the PODOS protocol (the simple Transmission-

Groups, Transmission-Groups under load) and further compared it with traditional networking protocol.

The CM is undergoing a major metamorphosis currently with the addition of error checking mechanism in the CDT. Variants of Transmission-Groups attempt to speed-up PODOS packets alone. A tremendous performance gain can be achieved if a similar packet multiplexing mechanism can be applied to other packets. For this purpose we could use the Channel-Bonding feature provided by the Beowulf that helps in multiplexing regular network packets. These features would make the CM more robust and reliable.

References

[1] A. Goscinski, "Distributed Operating Systems The Logical Design," Addison-Wesley Publishing Company, 1991.

[2] P. Merkey, "Beowulf Project at CESDIS," *http://beowulf.gsfc.nasa.gov*, 1994.

[3] M.J. Litzkow, M. Livny, and M.W. Mutka, "Condor-A Hunter of Idle Workstations," *Proceedings of the 8th International Conference on Distributed Computing Systems*, June 1988, pp. 104-111.

[4] S. Vazhkudai, P.T. Maginnis, "Distributed Linux: Evolutionary Steps," *Technical Report TR 1998-22, Computer Science Department, University of Mississippi*, December 98.

[5] "The Linux Documentation Project," *http://sunsite.unc.edu/LDP*, 1998.

[6] P.T. Maginnis, "Design Considerations for the Transformation of MINIX into a Distributed Operating System," *Proceedings of the 1988 ACM 16th Annual Computer Science Conference*, 1988, pp. 608-615.

[7] S. Vazhkudai, P.T. Maginnis, "Transmission-Group based Communication for PODOS," *Proceedings of the LinuxWorld Expo Conference*, San Jose, August 99.

[8] A.S. Tanenbaum, "Network Protocols," *ACM Computing Surveys*, vol. 13, no. 4, Dec. 1981, pp. 453-489.

[9] S. Vazhkudai, P.T. Maginnis, "The Design and Evolution of Communication in PODOS," *Proceedings of the 3rd ALS, USENIX Conference*, Atlanta, October 99.

[10] S. Vazhkudai, P.T. Maginnis, "Performance Oriented Distributed Operating System (PODOS)," *Technical report Computer Science Department, University of Mississippi*, May 99.

[11] D.R. Boggs, J.C. Mogul, C.A. Kent, "Measured Capacity of an Ethernet: Myths and Reality," *ACM SIGCOMM'88 Symposium*, vol. 18, no. 4, Aug. 1988, pp. 222-234.