# Low Latency Seamless Transport Interface

Mauli Gulati, Deepak Garg
*Computer Science and Engineering Department, Thapar University, Patiala*.

## Abstract

*In this paper we introduce a new concept of seamless interface. This seamless interface would help the businesses which are increasingly dependent on a connected world and the real-time flow of information across systems. In this paper we aimed at analyzing and combining the efficiency of shared memory and sockets for communication between processes on local host and on remote host. The notion here is to design and introduce the concept of an abstraction layer that encapsulates shared memory based communication interface and socket based communication interface into one seamless interface. In present market at this point of time, we do not have an equivalent solution. This innovative solution is all set to change the way industry communicates in between processes. The solution tries to give communication on local host its deemed advantage. The resultant system shall result in be extreme low latency, and would be used by the commercial organization.*
.

## Keywords

Seamless Interface, Shared Memory, Sockets, Inter-Process Communication, Local Host, Fast Communication.

## 1. Introduction

A communication can takes place between processes on local host or on remote host through number of ways. Since, there is a need for proper integration of business processes we require an efficient messaging system. In present market there are number of messaging solutions available. But all of them use sockets underneath for communication between processes on local host or on remote host which results in somewhat overkill of communication time and affects the performance of processes on local host. Shared memory is the fastest known inter-process mechanism and there are certain solutions available in market but all those solutions are customized. That is they are designed and used by few programmers to meet there own need. There is no integrated solution available in the market that could encapsulate the best features of both sockets and shared memory into one and use them for their designated tasks. Here we aimed at proposing an interface that would actually encapsulate shared memory based interface and socket based interface into one seamless interface. It would also let the user communicate without knowing the underlying complex architecture.

So the main question here is do we really need an interface that would encapsulate the best feature of both shared memory and sockets. And the answer is definitely yes, because this would lead to an interface that would allow programmer to use the shared memory for local communication and sockets for remote communication without knowing the underlying complexities. They would be able to use it like any other library they are presently using for communication. Programmers can focus on their designated task without being worried about the synchronization problems between processes using shared memory.

## 2. Hypothesis

The main objective of this paper is to analysis the present market. We have considered two big messaging solution providers in market which are TIBCO and 29WEST, for analyzing present market. They are using sockets for communication between processes on local host as well as remote host because of the portability and transparency provided by sockets. After analyzing we designed a seamless interface that would encapsulate the shared memory based interface and socket based interface. Our seamless interface would help us to marry the best features of both worlds.

## 3. Background

Shared memory has been used for the inter-process communication since it is the fastest known inter-process mechanism known so far but all these applications are customized. And similarly sockets are also used for communication between the processes on local host and as well as for the communication between the processes on remote host. But there is no integrated solution available in the market that could encapsulate the best features of both sockets and shared memory based inter-process communication. There is a requirement to design a seamless interface that would encapsulate socket based interface and shared memory based interface into one seamless interface to achieve maximum performance. In order to facilitate proper understanding of this paper let us throw some light on few two important terminologies used in this paper:-

### 3.1 Shared Memory

A shared memory is the part of memory that is mapped to the address spaces of the processes that wish to communicate. These processes share the same memory segment and have access to it. The following figure shows two processes and their address spaces. The shared memory is attached to both the address spaces and both process 1 and process 2 can have access to this shared memory as if the shared memory is part of its own address space.

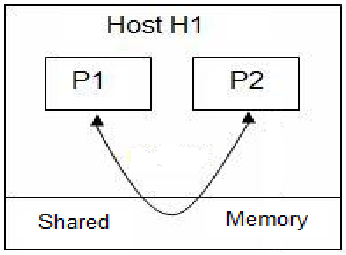

Figure 1. Shared Memory

While using shared memory as an inter-process communication mechanism, synchronization between the processes should be taken care off. Since there is no protection to a shared memory any process that knows it can access it freely. To protect a shared memory from being accessed at the same time by several processes, a synchronization protocol must be setup. In short, once the memory is being shared, there are no checks on how the processes are using it. They must rely on other mechanisms, for example System V semaphores, to synchronize access to the memory.

### 3.2 Sockets

Internet socket or network socket or socket is used for inter-process communication. A socket is one end of a two-way communications link between two programs running on the network. A socket address is the combination of an IP address and a port into a single identity.

The socket acts as a unique integer number called socket identifier or socket number within the operating system and the application that created it. Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP addresses and port numbers. Each socket is mapped by the operational system to a communicating application process or thread.

Sockets happen to be one of the most popular interfaces for communication over a network and even for communication on same host; the reason for this popularity is ease of usage and seamless connectivity, irrespective of location of target process.

## 4. Significance of Proposed Work

Communication via shared memory is fastest since it would not involve the interference of kernel for communicating information and data across processes on local host. Till date all the existing solutions are using sockets for communication between processes on local host or on remote host because of the transparency provided by sockets and extra overhead accompanied with shared memory in terms of synchronization. Here we aimed at designing a seamless interface which would reduce the inter-process communication time on local host by using shared memory and provide a seamless interface encapsulating shared memory based communication interface and socket based communication interface. Shared memory would be automatically used by an application if processes on the same host need to communicate. If communication between processes on different hosts is required than socket would be used automatically.

Sockets are the most robust and convenient whereas shared memory is fastest with a limitation that processes should be on the same host. We need to marry these two and come up with a solution which provides seamless interface on top providing best features of both worlds.

## 5. Communication via various methods

Communication can take place between processes by number of different mechanisms. But sockets are the most popular and widely used mechanism while shared memory is fastest for local host process communication. If processes P1 and P2 are on same host and if P1 wants to communicate with P2, though shared memory is available however currently, sockets still seems to be a better choice considering the complexity involved when dealing with shared memory and flexibility provided when using sockets. If we remove the complexity of shared memory usage from application programmer than it suddenly becomes a better choice.

However, we still have another limitation where we cannot communicate with a process on remote host using shared memory.

**5.1 Communicating via Sockets**

Communication through sockets facilitates flexibility which means that with the help of sockets a process can communicate with a process on another host within a same network or on different network with an ease. Though it affects the performance when processes on same host communicate via kernel resulting in slower communication speed because of extra copy operations and more context switches.

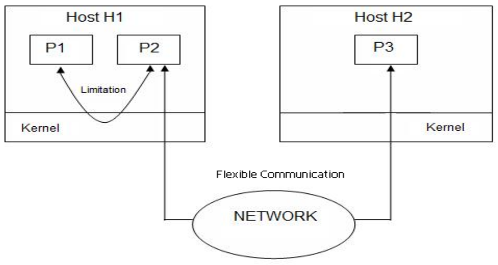

Figure 2. Communication via Sockets

**5.2 Communicating via Shared Memory**

Communication through shared memory transport suffers from one main disadvantage and one main advantage. Advantage is the fast speed of communication when processes on the same host communicate. And the disadvantage is that with the use of shared memory processes on different host cannot communicate.

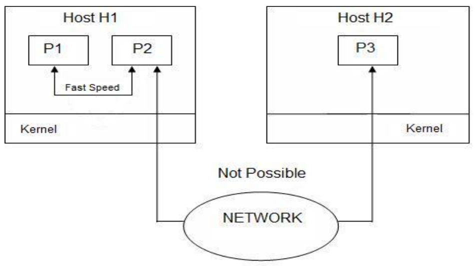

Figure 3. Communication via Shared Memory Transport

**5.3 Communicating via Seamless Interface**

Communication through seamless interface has two main advantages. First, advantage is the fast speed of communication when processes on same host communicate using shared memory transport. Second, advantage is that it offers high flexibility with the use of sockets for the communication between processes on different host. The seamless interface selects the shared memory transport or the socket interface seamlessly that is it's transparent from the user and the application program.

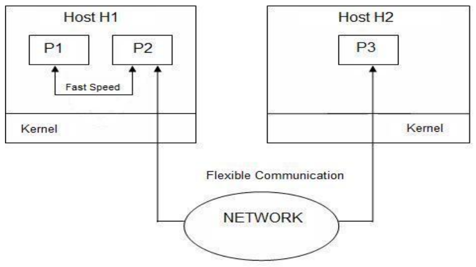

Figure 4. Communication via Seamless Interface

## 6. Conceptual Model of Proposed System

The conceptual model helps to analysis the complete system with an ease. The proposed system is designed keeping in mind the growing demand for fast communication of information and data between the processes on local host. The proposed seamless interface would result in a low latency system on local host which is the demand of current industry and as the name suggests "seamless", selection of sockets or shared memory based on locality of processes would be transparent to the user. Moreover, the complexity of using shared memory would be eliminated by providing simple send routine.

The conceptual design of the system represents the overall architecture of the proposed system and would help to understand and analyze the various modules with their proper arrangement and other various aspects of the proposed system.

It basically consists of generic "Shared Memory Transport Module" and "Socket Transport Module". Shared Memory Transport module should be designed for meeting growing demand for faster communication between processes. But the proposed system should take care of synchronization between communicating processes. This Shared Memory Transport should support multiple Reader and Writer Processes and should provide efficient synchronization support to allow all processes on local host to communicate without causing inconsistency.

The shared memory segment should consist of multiple lock-free queues created by Writer Process and Reader Process should connect to corresponding queue. The complete design of shared memory transport and socket transport are out of the scope of this paper and can be implemented based on application requirement.

This conceptual diagram doesn't express in detail the procedure used by existing messaging system since that is out of the scope of this paper. The architecture of the complete proposed model gives accurate information about our seamless interface encapsulating shared memory based interface and socket based interface.

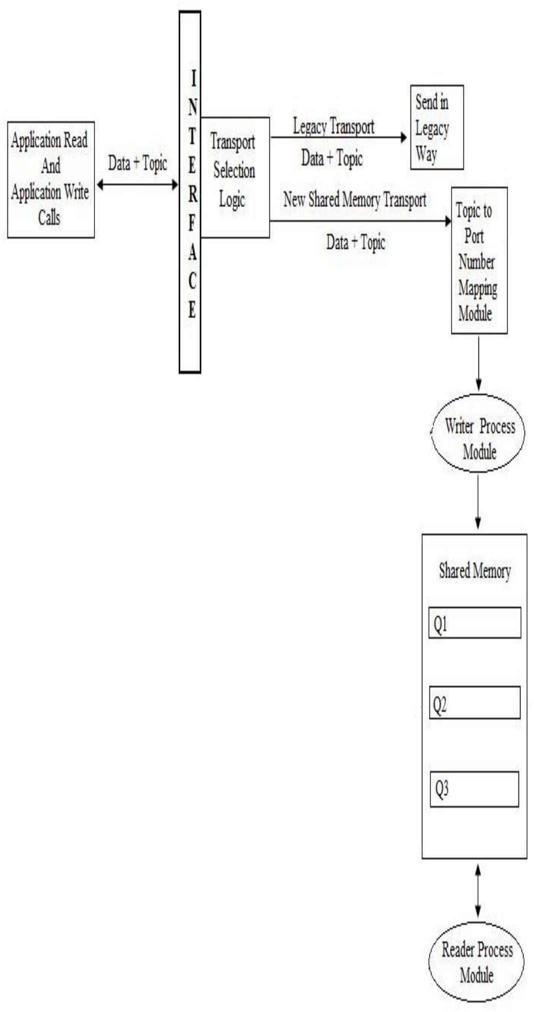

Figure 5. Architecture of the Proposed System

## 7. Algorithm of Seamless Interface

To implement seamless interface, let us assume a send routine accepting a 'DataBuffer' and a 'TopicName' from the User Application 1.

Topic name is basically a single or a set of multiple IP address and Port number pair. Here, we assume any of the available topic resolution mechanism would be used which would provide us with one or more IP address and a port number pairs associated with that topic. To simplify the matter for now we assume, it returns just one IP address and port number pair. Other scenarios would be trivial once we explain this use-case.

So now here we are with an IP address and port number pair given by topic resolution agent. The first step is to figure out if this IP address is same as our own IP address. Remember we can have multiple IP addresses for a same host. A simple parser which would parse output of ipconfig file and give us a set of IP addresses associated with our local host. So let us keep this set of IP addresses in a hash table which is initialized at process start-up. So we'll end up doing a lookup in this table. If address is found we would use shared memory as underlying transport otherwise we would use sockets as underlying transport.

```
Step 1: User Application 1 → send (DataBuffer, TopicName);
Step 2: ipAddr = TopicResolver ("TopicName");
Step 3: status = localIPaddrHashTable.lookup (ipAddr);
        if status is true
                sharedMemoryTransport.send
                (DataBuffer, TopicName);
        else
                socketTransport.send
(DataBuffer, TopicName);
```

The 'Topic to Port Number Mapping' module is used to map the Topic from the user to the port number associated with the shared memory segment queue. It is basically used to facilitate shared memory based communication in between publisher and subscriber.

## 8. Analysis and Results

The given conceptual solution was designed keeping in mind all the existing messaging solutions and the design goals of our paper.

This paper on **"Low Latency Seamless Transport Interface"** describes various concepts, arguments and applications related to the algorithm. And in section 3, we explained the basic concepts of Shared memory and Sockets as well. In this paper we analyzed in detail the advantages and disadvantages of using shared memory and sockets for communication between processes on local host and on remote host in addition to focusing on significance of this proposed system.

It is the well-known fact the shared memory is the fastest known inter-process communication mechanism and can result in extremely low latency system. Similarly it is also renowned that sockets are being used for communication across processes on local host and on remote host because of the flexibility and transparency that it provides.

But this seamless interface would definitely take the best from both sockets and shared memory and would help in marrying the best features of both to come up with more time-efficient and low-latency system which is the demand of current industry which is being depended on connected world and real-time flow of information.

## 9. Conclusion

Today all IT organizations or any business require communicating and exchanging data. And the focus is on saving time in processing and accessing data. All the available messaging solutions today are using sockets underneath for communication between local hosts or for communication on different hosts.

Shared memory is the fastest known inter-process communication for communication on local host. Hence, in order to increase the effectiveness in terms of CPU utilization and communication time between processes on local host we aimed at designing an algorithm. This effort would result in an end to end solution that would reduce the inter-process communication time on local host by implementing fast socket over shared memory. This implementation not only reduces the communication time but also reduces per message CPU utilization significantly.

That is our main notion is to provide a seamless abstract interface encapsulating shared memory based communication interface and socket based communication interface into one API.

The Shared memory based interface would be automatically used by API, if processes in question for communication are on the same host, while legacy Socket based interface will be used otherwise.

At present there is no integrated solution available to do similar job. And our solution would result in an API that’s time and CPU efficient, still generic enough to be used by a large variety of commercial applications. Be it a web server or a high availability real-time server. We target this solution to commercial organizations as their future messaging solutions for communication.

## 10. References

[1] W. Richard Stevens. UNIX Network Programming:Interprocess Communications, Volume – 2, Second Edition: Prentice Hall, 1995.

[2] Douglas E. Corner and Steven B. Munson. “Efficient Interprocess Communication Using Shared Memory”, February 1988.

[3] Sivaram; Rajeev (West Orange, NJ), Xue; Hanhong (Poughkeepsie, NY), “Facilitating communication within shared memory environments using lock-free queues,” U.S. Patent 7219198, May 15, 2007.

[4] Paul J. Christensen, Daniel J. Van Hook, Harry M. Wolfson, “HLA RTI Shared Memory Communication”, sponsored by the Defense Modeling and Simulation Office (DMSO) under Air Force Contract No. F19628-95-C-0002.

[5] Brain N. Bershad, Thomas E. Anderson, Edward D. Lazowska and Henry M. Levy, “User-Level Interprocess Communication for Shared Memory Multiprocessors”, in ACM Transactions on Computer Systems, Vol. 9. No. 2, May 1991.

[6] Jelica Protic, Milo Tomasevic and Veljko Milutinovic, “Distributed Shared Memory: Concepts and Systems”, by IEEE Computer Society Press Los Alamitos, CA, USA, in IEEE Parallel & Distributed Technology: Systems & Technology, Volume 4, Issue 2 (June 1996).

[7] (2002) The 29WEST website. [Online]. Available: http://www.29west.com/

[8] (2000) The TIBCO homepage [Online]. Available:http://www.tibco.com/software/messaging/default.jsp

[9] (2000) The TIBCO homepage [Online]. Available:http://www.tibco.com/software/messaging/smartsockets/default.jsp

[10] (2000) The TIBCO homepage [Online]. Available:http://www.tibco.com/software/messaging/enterprise_messaging_service/default.jsp

[11] (2002) The 29WEST homepage. [Online]. Available:http://www.29west.com/products/lbm/

[12] W. Richard Stevens, Bill Fenner and Andrew M. Rudoff. UNIX Network Programming: The Sockets Networking API, Volume – 1, Third Edition: Prentice Hall, 2004.

[13] Abraham Silberschatz, Peter Baer Galvin and Greg Gagne. “Operating System Principles”, Seventh Edition: WSE, 2006.