Accelerated Socket Communications in System Area Networks

Masanori ITOH, Tooru ISHIZAKI, Mitsuhiro KISHIMOTO
Fujitsu Laboratories LTD.
1-1, Kamikodanaka 4-Chome, Nakahara-ku, Kawasaki
Kanagawa 211-8588, Japan
{marc, tooru, hiro.kishimoto}@flab.fujitsu.co.jp

1 Introduction

In recent years, a great need for large scale servers has arisen. One practical solution is to construct such servers as cluster systems. For applications reside on them, inter process communications (IPC) among them must be high performance. The current generation of System Area Networks (SANs) have very wide bandwidths, and very low communication latencies, and they usually consume relatively little CPU time. Although such hardwares have quite good performance, we cannot utilize them fully with socket APIs that are used as a de facto standard IPC.

We can easily see the performance problem of socket APIs by comparing the results of benchmark test of socket APIs and new APIs such as VIPL [1, 2] over the same SAN hardware. For example, the result of our measurement of bandwidth says that we can get twice as good performance as socket APIs. In case of communication latencies and CPU utilization, we can get similar kind of results. This means that these performance problems come from the overheads of underlying network protocol stacks.

Indeed there are new APIs that are well optimized for SAN environment, none of them could not have replaced socket APIs. So, if we want to widely deploy an improved IPC over SANs, it is essential to retain compatibility with socket APIs.

Socket APIs are used to abstract the endpoints of communications via underlying network protocols as files. To be compatible with socket APIs, it is important to retain compatibility with the UNIX semantics for file descriptors without mentioning the semantics of the underlying network protocols.

2 Architectural Discussion

Socket APIs are constructed with layered structure. Ordinarily, a message from application (A) to application (B) will travel through all of the software component stacks shown in figure 1.

We can reduce the overheads involved in processing TCP/IP and UDP/IP protocol stacks and still retain compatibility with original socket APIs by making short cuts as depicted in figure 1.

The layers where short cuts are possible are (1) the user library, (2) the stream head, (3) the transport/network layer and (4) the data link layer.

The last means to implement an ordinary TCP/IP or UDP/IP protocol on System Area Networks.

Rodrigues et.al.’s FastSocket [3] was a pioneering work of this kind of acceleration. They adopted the user library short cut. Although this approach has the greatest potential in terms of performance, it suffers from a fatal problem in terms of compatibility with socket APIs and resolving this problem in itself has a high cost in terms of performance. This is thus not a realistic candidate of our solution.

The stream head short cut is viable in terms of both performance and compatibility. Our solution requires transparency when implemented in a multi-vendor system, so this is not the best solution.

The final approach is the transport/network layer short cut. This carries a small penalty in terms of performance, when compared with an approach based on the stream head.
layer short cut, but we can improve the performance by introducing new extended socket options. Furthermore, we can resolve the problem of transparency that arises in the stream head short cut approach.

Putting all of the possible approaches in perspective, the transport/network layer short cut is the best solution. In particular, the top border of the transport layer is best location for the short cut.

3 Implementation

Figure 2 shows the architecture of VIsocket.

![Figure 2. Architecture of VIsocket](image)

We chose VI Architecture for our System Area Network. That is the origin of the name: VIsocket.

The first target platform of VIsocket is Solaris of SUN Microsystems.

In figure 2, SCVA is an in-kernel VI Provider interface, and can be referenced from the web site of VI Developer Forum [4]. TPI, NPI and DLPI are specifications for the implementation of STREAMS module/driver interfaces by X/Open.

A short cut as described in section 2 is made within the VIsocket STREAMS module of figure 2. This module is pushed automatically by the autopush(1M) utility over /dev/tcp and /dev/udp to realize a short cut through the top transport layer.

Compatibility requirements for file descriptors are satisfied naturally by realizing the short cut within the kernel space, because the semantics of the file descriptors come from the vfs layer that resides above VIsocket’s STREAMS module.

The stream socket (TCP) and the datagram socket (UDP) are implemented within the VIsocket driver.

In our design, VI connections are established per node-to-node connection. TCP connections are established by exchanging control messages over the VI connection and TCP connections requested by user applications are multiplexed over those connections. UDP datagrams are multiplexed over a separate VI connection.

If the VIsocket STREAMS module receives a message that needs routing or which has a destination that cannot be reached by VI, the VIsocket STREAMS module will use the standard TCP/IP or UDP/IP protocol stacks. Applications can then transparently communicate with other applications both within and outside the area that VI is able to reach.

4 Conclusion

In this paper, we have outlined VIsocket, a communications interface which is compatible with socket interfaces and provides a low communication latency and a high bandwidth.

We discussed the problems of legacy socket APIs and offered several ideas on how the problems could be overcome, then surveyed them in terms of tradeoffs between performance and compatibility.

Currently, we are doing functional test of VIsocket and performance tuning. We have examined that VIsocket is compliant with the socket specification of UNIX98, and checked compatibility by using telnet, ftp, Apache, Netscape and WebSTONE. All of them worked normally.

We also estimated the performance of the final version of VIsocket. The round trip latency over Synfinity-2 (a SAN of Fujitsu that implements VI Architecture by hardware) is estimated $80\mu\text{sec} \sim 90\mu\text{sec}$. This is three times faster than the estimated value of TCP/IP latency over the same hardware.

References


