Failure Impact on OpenAIS: An Experimental Evaluation

MU-CHI SUNG, MING-CHUN CHENG, ZHI XIN FAN, PING-JER YEH, SHYAN-MING YUAN
Department of Computer Science
National Chiao Tung University
1001, Ta Hsueh Road, Hsinchu 300
TAIWAN

CHIA-YUAN HUANG, LO-CHUAN HU
Information & Communications Research Laboratories
Industrial Technology Research Institute
195, Chung Hsing Road, Section 4, Chu Tung, Hsinchu 310
TAIWAN

Abstract: The Service Availability Forum (SA Forum) was formed to foster an ecosystem as a building block for carrier-grade system development. It published a series of high availability specifications, commonly referred to as Application Interface Specification (AIS) and Hardware Platform Interface (HPI). In this paper, we try to evaluate recent implementation of the only one open source AIS-compliant middleware, OpenAIS, to see whether it satisfies the critical needs of failure impact in many carrier-grade applications. Our benchmark is conducted in two cases: processor leave and processor join, and evaluates several parameters crucial to the usefulness of the membership protocol. The result shows good performance for both cases. Finally we point out some possible directions to improve the worst-case time of OpenAIS.

Key-Words: Service availability, OpenAIS, Totem, failure impact evaluation, carrier-grade service

1 Introduction

As broadband booms and technology evolves, communication and data service providers are challenged to seek for cost-down while upholding carrier class services with high availability. When there is a need to deploy a new service, however, current practice of using proprietary and special-purpose hardware/software combinations makes design limited and maintenance expensive.

To survive in today's competitive marketplace, therefore, they head for other high availability platform solutions that have shorter time to market and are less expensive to maintain. As a result, the Service Availability Forum (SA Forum) was formed to push the delivery of next-generation service availability solution based on Carrier-Grade Linux (CGL). The SA Forum then published the Application Interface Specification (AIS) [1] and Hardware Platform Interface (HPI) [2] with works contributed from many industry-leading companies to enable providers to build their high availability hardware/software solutions under an agreed standard.

AIS only specifies the interface, not the implementation details. So far, the only one open source implementation for AIS-compliant middleware is OpenAIS [3]. The OpenAIS project was started in early 2002, and in 2003 it was morphed to develop an AIS-compliant middleware. After more than four years' development, OpenAIS successfully makes use of leading networking technologies to provide high availability complying with the AIS.

Since the official AIS is still under evolving, it is obviously impossible for OpenAIS to keep up-to-date to provide all defined services. Nevertheless current OpenAIS implementation still provides a good platform for building applications that maintain service during faults for the sake of well-designed architecture and effective group communication algorithms (to be discussed briefly in Section 2).

For service providers, however, they cannot adopt this OpenAIS solution without essential premises on service continuity [4][5], such as short system response time and, more importantly, short system recovery time from failure. Providing service continuity is not an easy job for various carrier-grade applications. Strict constraints such as different QoS requirements have to be achieved to make services acceptable from users' perspective. Furthermore, different criteria of acceptance may be expected by different groups of users. For example, the phone-to-phone delay in public telecommunication service should not exceed 150 ms [6], consisting of several components [7]:

- **Network Delay** is the delay during signal propagation on the network.
- **Codec-Related Delay** is the delay in packetization, look-ahead processing, and encoding.
- **Jitter Buffer Delay** occurs during smoothing out signal variation in network delay.

Undoubtedly service providers expect a middleware such as OpenAIS to perform its job well and at the same time minimize the delay incurred by all kinds of network events, since the middleware may take parts in all types of equipment and introduce a variety of delay. Therefore, this paper aims to explore the minimum delay of OpenAIS under the presence of processor failure and new processor installation, and to examine whether the current OpenAIS implementation satisfies the crucial needs in carrier-grade applications.

The rest of this paper is organized as follows. Section 2 introduces the OpenAIS architecture and its underlying network protocol. Section 3 describes the evaluation method and crucial adjustable parameters to be measured and fine-tuned. Section 4 presents the benchmark results. Section 5 discusses the findings and proposes possible improvements. Section 6 offers brief concluding comments.

2 OpenAIS Overview

In this section, we will take a closer look at the OpenAIS architecture and briefly explain the network protocol since they are essential to a proper understanding of the experiment and results discussed in the following sections.

2.1 Architecture

From the architectural perspective, the OpenAIS can be split into two main parts, one for the library part, and the other for the executive part, as shown in Fig. 1. The library part implements and maintains data structures for the AIS standards, while the executive part acts as a server program running on each node (or more precisely, processor) to complete client requests from the library part in the same context. When an application wants to use the standard interface, it needs to initialize an instance of the library first, and then the library instance will create several connections to the local executive server on behalf of the application itself.

All executive servers running on participating nodes (one executive server instance per node) will form a group and communicate with one another as a single cluster to provide AIS services, including:

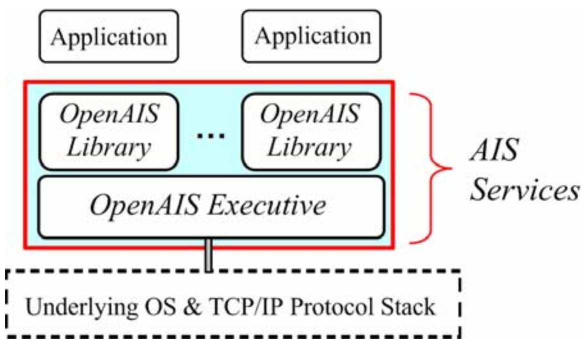

Fig. 1: OpenAIS architecture

− Availability management framework (AMF)
− Checkpoint service (CKPT)
− Cluster membership service (CLM)
− Event service (EVT)
− Message service (MSG)
− Distributed lock service (DLOCK)

Among them, AMF and CKPT are used to mask many types of faults in upper application and operating system, and to help developer to create redundant data against failures in a distributed fashion. On the other hand, CLM, EVT, MSG, and DLOCK provide communication and coordination features in distributed environment.

2.2 Reliable Multicast Protocol

AIS does not specify implementation details. To support distributed AIS services among multiple nodes, there must be a group communication layer inside the OpenAIS protocol stack as the basic infrastructure. Thus, OpenAIS adopted the Totem reliable multicast protocol [8] to satisfy such needs.

The Totem protocol was first proposed in 1995 and has been improved for several years. It is now widely accepted as a proven effective approach to achieving reliable group communication. Even though there are so many reliable multicast protocols proposed with quite different categories of purpose, Totem was designed mainly to reduce message latency and increase message throughput under the condition of group membership changes and unstable network. Totem's property of consistent membership has made it suitable for carrier-grade application domain, under which applications need to work as usual against processor failures.

To manifest related parameters used in our evaluation section, the Totem protocol is briefly described here. More detailed and formal definitions/proofs can be found in [9].

The Totem reliable multicast (formally termed Totem *single* or *multiple ring protocol*) employs one

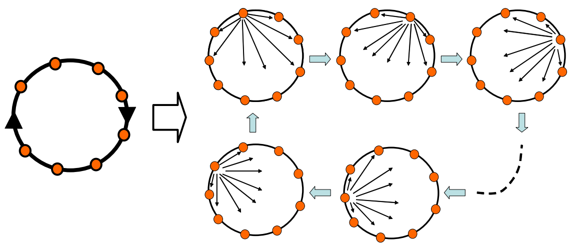

Fig. 2: Token passing on the Totem network

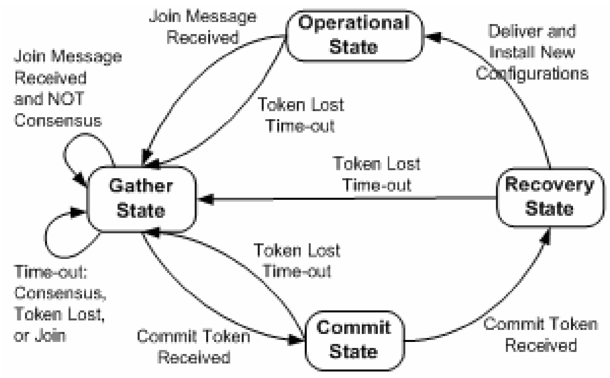

Fig. 3: State diagram in membership protocol

or more logical-passing rings within a broadcast domain, respectively. With four basic control mechanisms (total ordering protocol, membership protocol, recovery protocol, and flow control algorithm), all messages are delivered under the constraints of *extended virtual synchrony* [10] and consistency is maintained under the condition of membership changes. OpenAIS only uses the first three mechanisms of Totem, and this paper only tries to evaluate the *membership protocol*.

In Totem's single ring protocol, all processors form a logical ring, and a token continuously circulates around in a certain order. Whenever a processor receives a token passed from the previous processor, it is allowed to broadcast messages to all the other processors (see Fig. 2) and modifies the token accordingly. In this manner, messages can be delivered in an agreed or safe order. Once any fault occurs, which would stop some processors from responding, the logical ring will be broken and the membership protocol will be started within a given period of time slice to rebuild a new ring.

Inside the membership protocol, there are several states and transitions between them, as shown in Fig. 3. Normally every processor is in the operational state. If it receives a foreign join message, implying that one or more faults occur somewhere in the ring, it will turn into the gather state. In the gather state, each processor broadcasts join messages, perceives and revises a new temporary membership

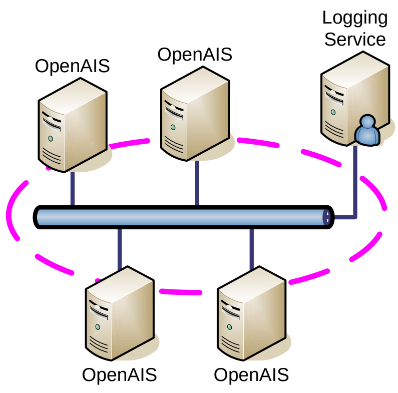

Fig. 4: Experiment Environment

set, gathers necessary information, and tries to reach a consensus about the new ring configuration. If all processors in the newest temporary membership set have the same view on the new ring, they will turn into another state called commit state. In the commit state, a new representative is elected and it will issue a special token to circulate around the new ring. After this special token is circulated, all processors are ensured to agree on the new membership and then shift to the recovery state. In the recovery state, messages will be recovered and redelivered according to Totem's ordering constraints, and all processors will go back to the operational state one by one after the next token circulation.

The whole process of membership protocol is not as simple as the one described here, but conceptually it can give some ideas about how the membership protocol works. As you can see, if we are trying to measure or to reduce the failure impact on service continuity, OpenAIS's implementation of the membership protocol dominates the overall performance.

3 Evaluation Method

3.1 Environment Setting

The experiment is conducted in a closed system with 4 nodes running OpenAIS 0.70 and 1 node as the log server, as shown in Fig. 4. The 4 OpenAIS nodes are of the same model with Pentium 4 2.4 GHz and 1 GB main memory, running RedHat 9.0 with 2.4.20-8 official kernel. All standard services are removed to keep the environment as simple as possible. The log server is VIA EPIA MII platform with 256 MB main memory, running RedHat 9.0 as well. The closed system is connected by a 100 M/bits Ethernet with a D-Link DES-1008D switch.

To measure the membership protocol execution time, we inject some code snippets into OpenAIS to inspect the time elapsed in each state and send notification to the log server. The time for every event is measured by the high resolution timer, which makes use of *rdtsc* assembly instruction to obtain current CPU clocks. Also, to achieve best performance, we remove extra time-out limits in OpenAIS since they were originally used to prevent unstable configuration.

3.2 Adjustable Parameters Relevant to the Gather State

The visible performance of OpenAIS under the condition of failure depends heavily on the internal parameter settings of the membership protocol. To investigate the failure impact on OpenAIS, several parameters have to be fine-tuned to find the lower bound of execution time required to rebuild a new ring in both cases of processor join and leave. The following text identifies relevant parameters that may affect the performance of the membership protocol (and consequently, OpenAIS): join, consensus, and token loss time-out.

Whenever a fault is perceived, operational processors will turn into the gather state and try to reach consensus on a new configuration (mentioned in Section 2.2). As one processor receives and perceives a temporary configuration inside any foreign join message, it will broadcast yet another join message containing a new configuration it revises after a small time span called *join time-out*. This time-out is used to prevent message burst that would unstabilize the whole network in the beginning of the gather process. There is a trade-off here: the shorter the join time-out is, the more quickly the gather process would run, but the more likely the network would be unstable.

All operational processors would broadcast join messages continually until they finally reach a consensus on the new configuration. Therefore, OpenAIS will never know exactly which processors have failed until the so-called *consensus time-out* expires. The consensus time-out was started immediately after entering the gather state. When it expires, all operational processors will add no-responding ones to the so-called *failed processor list*, re-enter the gather state, and try to reach consensus before the next consensus time-out expiration. Here comes another trade-off, too: the shorter the consensus time-out is, the more quickly the failed processors are identified, but the more likely an operational processor would be misunderstood as a failed one.

3.3 Adjustable Parameters Before Entering the Gather State

While the join time-out and the consensus time-out are used in the gather state, the *token loss detection time-out* and *token retransmission time-out* determine the time needed from the time when any failure occurs to the time when the gathering process is invoked.

Tokens in the logical ring could be lost due to processor failure or network partition. Both cases can be detected by the token loss time-out, and can be solved by the membership protocol discussed previously.

The token retransmission time-out is more implementation-oriented. In OpenAIS, tokens are transmitted between each node on the logical ring by UDP unicast, rather than UDP multicast or TCP. Since UDP unicast is not a reliable transmission facility, tokens could be lost due to various reasons such as switch buffer overflow. To resolve such circumstances, OpenAIS uses the token retransmission time-out to know when it should retransmit the token to the next hop. The default value for the token transmission time-out is about 1/4 of the token loss time-out; that is, all processors will retransmit the token for 4 times before invoking the gathering process.

As a processor receives a token from the previous hop, it will restart the token loss time-out and the token retransmission time-out. As a result, the token retransmission time-out decreases the probability of token loss between hops. It is also obvious that the next hop of the failed processor will be the first one recognizing the failure and initiating the gathering process.

3.4 Evaluation Procedure

In the series of experiments we try to adjust different parameters one by one to minimize the latency in configuration changes, and to observe the correlations among all these factors.

A single run for each parameter evaluation is automated by killing the in-memory process of OpenAIS and then restarting it on the specific node. As soon as the node leaves the ring, the log server will be notified (by the injected code) to record and measure the token loss detection time. Every case in our experiment is conducted for 100 runs to obtain the average performance.

4 Experimental Result

This section shows and discusses our experimental result. Since the situations for processor joins and

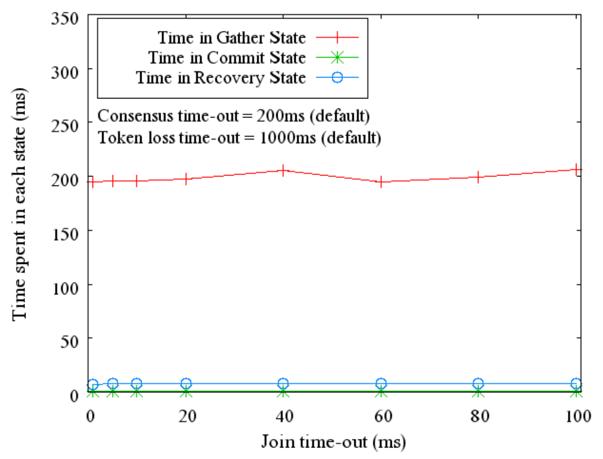

Fig. 5: Different join time-out: the processor leave case

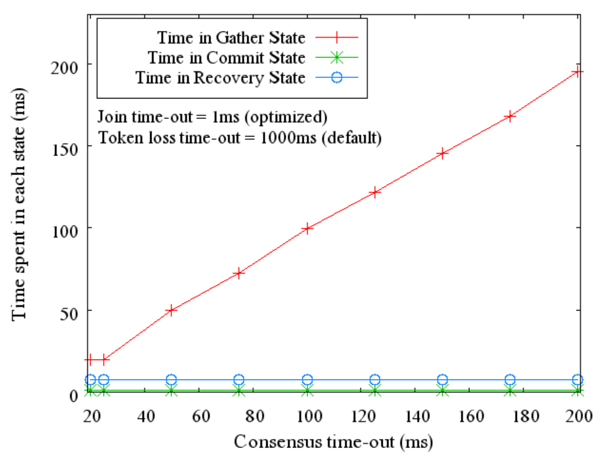

Fig. 7: Different consensus time-out: the processor leave case

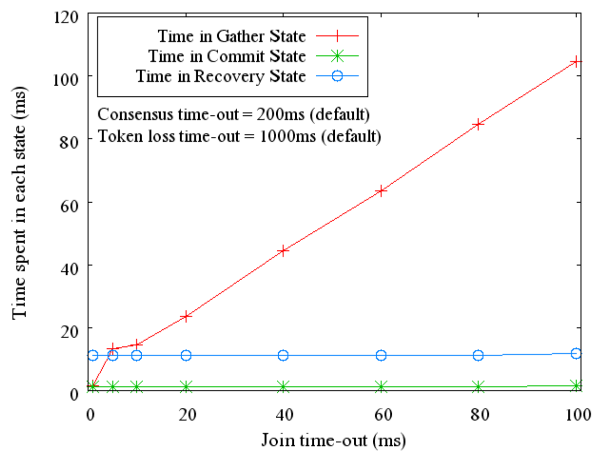

Fig. 6: Different join time-out: the processor join case

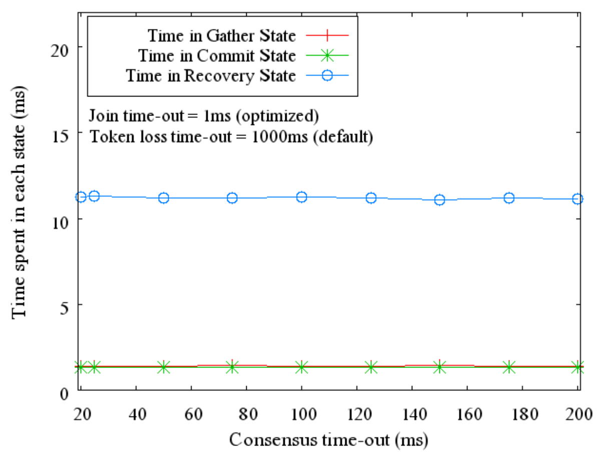

Fig. 8: Different consensus time-out: the processor join case

leaves behave quite differently in the membership protocol, we will show the results in different figures. Section 5 will go further to propose an alternative approach to improving worst-case failure detection time and to eliminating the destined consensus time-out.

4.1 Impact of Join Time-out

In Fig. 5, when some processors leave from current configuration, the time taken by membership protocol remains almost the same. The reason is that, by definition all operational processors must wait until the consensus time-out expires so that they can continue to add the leaving processors into the failed processor list. In this case, the membership protocol performance is dominated by the consensus time-out, which is 200 ms by default.

In Fig. 6, when foreign processors join in, different join time-out results in different performance in the membership protocol. The result is quite intuitive. In our experiments, with the default consensus time-out and token loss time-out, the join time-out can be shortened to 1 ms without unstabilizing the system, and the time spent in the gather state can be reduced to 1.4 ms on average.

4.2 Impact of Consensus Time-out

To shorten the time in the processor leave case, we move on to change the consensus time-out while keeping the optimized join time-out setting obtained previously since our goal is to minimize the latency.

In Fig. 7, the time in the gather state declines as the consensus time-out decreases. However, when the consensus time-out drops down to 15 ms or less, the system will never reach consensus. The reason is that the consensus time-out always expires before all processors ever have a chance to agree on a new membership.

In Fig. 8, since the consensus time-out is irrelevant to processor join, the performance of the

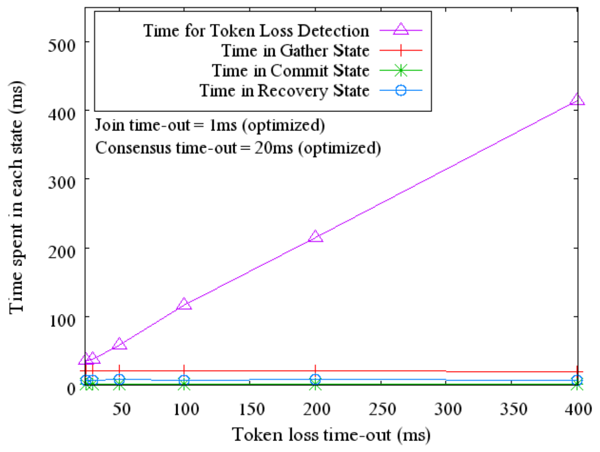

Fig. 9: Different token loss time-out: the processor leave case

membership protocol will remain the same. In fact this is the best performance OpenAIS can achieve in the case of processor join in our experiment.

As can be seen from Fig. 5 to Fig. 8, the time spent in the commit state and the recovery state remains almost unchanged for different join time-out and consensus time-out since it only depends on the token rotation time.

4.3 Impact of Token Loss Time-out

So far we have identified the best settings for the join time-out and the consensus time-out. However, it is the token loss time-out that controls the time required to detect any failure. The default value for the token loss time-out in OpenAIS is 1 second, but it may be too long for most carrier-grade application.

In Fig. 9, the time for the token loss detection is in proportion to the token loss time-out. However, there is a lower bound for this time-out because token rotation on the logical ring also takes time. In the conducted experiment, we observe the token loss time-out cannot be shorter than 30 ms in our system; otherwise the system would become unstable with the probability of 0.17 to enter the gather state without real configuration change events.

5 Discussion

So far, the best performance for processor join is about 13.84 ms, while for processor leave is as large as about 65.97 ms. There are two reasons for such differences.

5.1 Token Loss Detection

Because Totem detects network or processor failure by various time-out (mentioned in Section 3.3), the token loss time-out cannot be removed completely. Nevertheless, it could be lower than the current limit.

If we change the transport mechanism of token from the UDP unicast to multicast, we can relax the limit. In this manner every time a token is transmitted from one hop to another, all processors would recognize the fact that the source processor is alive and that the token is not lost, so the token loss time-out can be restarted. This allows us to shorten the token loss time-out to a multiple of the transmission delay between two hops. In such case, the token loss time-out will no longer be bounded by the number of nodes, which improves a lot in case of large number of nodes.

5.2 Destined Consensus Time-out

Whenever a processor leaves or fails, OpenAIS would initiate the gathering process of the membership protocol. By definition all operational processors must now wait for at least one consensus time-out (mentioned in Section 3.2) to determine which processors have failed and to exclude them from the new ring configuration.

This destined consensus time-out can be avoided if all operational processors know exactly the failed processor at the very beginning of the gathering process. It could be achieved by the same approach to lessening the token loss time-out. If we employ multicast in OpenAIS, all operational processors would receive tokens from all the others except the failed ones. As a result we could probably put those processors who did not broadcast tokens during the previous rotation into the failed processor list, reach consensus on the new membership quickly, and avoid the destined consensus time-out.

Of course, changing the transport mechanism of tokens requires more careful consideration from the design perspective. We propose the potential problems and possible solutions for readers interested in further improving the OpenAIS performance or in understanding the limit of the latest implementation of OpenAIS.

6 Conclusion

With the explosion in carrier-grade online services, an ecosystem to foster application development is becoming imperative to reduce the cost and the time-to-market. In this paper, we have evaluated the state-of-the-art implementation of an AIS-compliant middleware, OpenAIS, to see whether it is a practical and adequate platform for carrier-grade applications. Although these experiments were conducted under a simple scenario without external communication burst (high system load from other processes), and without real-world carrier-grade network equipments (carrier-grade IP switches with MPLS features), the proposed results are still useful for application developers to help assessments based on the concern of service continuity. They can assess whether OpenAIS fits their targeted application domain based on the time lower bound of the membership protocol during which services are interrupted.

Moreover, the proposed directions to decrease worst-case failure recovery time of OpenAIS makes it possible to further improve the service continuity without the need to buy faster transport media such as gigabit Ethernet.

To summarize, OpenAIS is still a good choice for application developers while selecting commercial-off-the-shelf (COTS) building blocks in creation of high availability applications.

References:

[1] Service Availability Forum, *Service Availability Interface Overview B.02.01*, 2005

[2] Service Availability Forum, *Hardware Platform Interface Specification B.01.01*, 2004

[3] OpenAIS, http://developer.osdl.org/dev/openais/

[4] Catherine Boutremans, Gianluca Iannaccone, and Christophe Diot, Impact of link failures on VoIP performance, *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ACM Press, Miami, Florida, USA, 2002, pp. 63-71

[5] Cornelis Hoogendoom, Karl Schrodi, Manfred Huber, Christian Winkler, and Joachim Charinski, Towards Carrier-Grade Next Generation Networks, *Proceedings of International Conference on Communication Technology*, IEEE Press, Beijng, China, 2003, pp. 302-305

[6] Yan Chen, Toni Farley, and Nong Ye, QoS Requirement of Network Applications on the Internet, *Information, Knowledge, Systems Management*, Vol. 4, No. 1, 2004, pp. 55-76

[7] Raj Kumar Rajendran, Samrat Ganguly, Rauf Izmailov, and Dan Rubenstein, Performance Optimization of VoIP using an Overlay Network, Technical Report, July 2006 http://www.ee.columbia.edu/~kumar/papers/icdcs06.pdf

[8] Y. Amir, L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, and P. Ciarfella, The Totem Single-Ring Ordering and Membership Protocol. *ACM Transactions on Computer Systems*, Vol. 13, No. 4, 1995, pp. 311-342

[9] L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, R.K. Budhia, and C.A. Lingley-Papadopoulos, A Fault-Tolerant Multicast Group Communication System. *Communications of the ACM*, Vol. 39, No. 4, 1996, pp. 54-63

[10] L.E. Moser, Y. Amir, P.M. Melliar-Smith, and D.A. Agarwal, Extended Virtual Synchrony, *Proceedings of the 14th International Conference on Distributed Computing Systems*, IEEE Computer Society, Pozman, Poland, 1994, pp. 56-65