A Transport Framework for Distributed Brokering Systems

Shrideep Pallickara[1], Geoffrey Fox[1], John Yin[2], Gurhan Gunduz[1,3], Hongbin Liu[1], Ahmet Uyar[1,3] , Mustafa Varank[1]
{spallick,gcf,holiu,mvarank}@indiana.edu, Community Grid Labs, Indiana University[1]
johnyin@earthlink.net, Anabas Inc., CA[2]
(ggunduz,auyar}@syr.edu, Department of Electrical Engineering and Computer Science, Syracuse University[3]

1.0 Introduction

Recent years have seen an increase in the number of devices with differing communication, compute and display capabilities. Increasingly, services need to interact with a wide spectrum of devices with varying networking capabilities. In most cases, services and the functions that they perform are independent of the transports deployed for communications. Furthermore, given the scale and the variety of devices that services need to interact with, services are usually hosted on a distributed messaging infrastructure. It is thus entirely conceivable that a message would traverse multiple hops (possibly over different underlying transports) en route to its final destinations.

Services hosted on a messaging infrastructure need to optimally utilize and exploit the conditions that exist within the local networks. Different transport protocols are suited for different tasks. Multicast works best within a domain where there is a high concentration of clients, most of which are interested in those events. TCP works best where reliable delivery is at a premium. UDP works best for applications that can sustain losses in delivery to clients but cannot afford the premiums – associated with error correction and out of delivery in TCP – that can lead to increased latencies.

Service protocol layers reside on top of the transport/networking layer. This layer may have constructed a view of the entire distributed network, but the routing algorithms may still continue to operate on an abstract representation of underlying communication links. The messaging infrastructure must manage the communication between external resources, services and clients to achieve the highest possible system performance and reliability. A lot of this decision making resides in the transport layers.

In this paper we suggest that the problem is an important one, and that, a transport framework needs to be incorporated into the messaging infrastructure hosting the services. We may enumerate the issues that need to be addressed within any transport framework designed for distributed brokering systems. These include –

1. Framework Design: Interfaces need to be general enough to abstract the communication requirements for most service protocol layers. At the same time, the interfaces should ensure that they are general enough over multiple transports, while not incorporating details pertaining to a specific transport into the framework.
2. Easy extensibility: An ability to incorporate support for new protocols easily. Each implementation of the interfaces might include support for any handshaking protocols that might be necessary.
3. Alternate Communications: Though communications between two nodes in the fabric would be over a transport protocol, there might be applications for which communications over that transport protocol might be unacceptable. The transport interfaces need to incorporate support for this need.
4. Performance Monitoring: The ability to incorporate support for measuring network performance over communication links. Performance monitoring is generally the pre-cursor to any remedial measures that might be deployed to assuage network conditions.
5. Migration Support: A lot of times the underlying transport of a communication link might become unsuitable for continued communications under certain network conditions. Links should thus be able to deploy other transports for communications. Link creators specify the conditions under which these migrations should take place.
6. Negotiation of best transports: Two nodes should be able to negotiate the best transport for communications.

Finally, a truly dynamic system would allow for adaptability in communications by responding to the changing network conditions. Though self-sustaining, responsive and self-healing systems are not within the scope of this paper, the underpinnings for such systems exist in those systems that provide a flexible transport framework, addressing the issues enumerated above.

There are also two other issues, which implementations of these transport interfaces need to address. First, it is inevitable that the realms, over which the nodes try to establish communication links, would be protected by

firewalls that would halt application channels dead in their tracks. The messaging infrastructure should thus be able to communicate across firewall, DHCP and NAT boundaries. Sometimes communications would also be through authenticating proxies.

Second, and more subtly, there are cases where the transport interfaces themselves would be used to process data received and routed from and to specialized applications. Implementations of transport interfaces would themselves be used to incorporate support for legacy applications, without the need to incorporate complicate logic in the higher layers at a given node. A similar strategy has been used by us to incorporate support for audio/video conferencing while interfacing with legacy clients. Work is also underway on a specialized implementation of the interfaces to incorporate support for PDA device. Note that data pre-processing is done over the transport interfaces.

In this paper, we address these issues in the context of our advanced research prototype, NaradaBrokering [1-7]. This paper is organized as follows. Section 2.0 provides an overview of the related work. In section 3.0 we provide an overview of the NaradaBrokering system, we then proceed to outline the transport framework in section 4.0. Section 5.0 presents issues related to implementing the framework with section 6.0 providing results from various transport implementations. Finally, in section 7.0 we outline the future work that we intend to do, along with our summary and conclusions (section 8.0) from the work discussed in this paper.

2.0 Related work

JXTA [8,9] from Sun is a set of open, generalized protocols to support peer-to-peer (P2P) [10] interactions and core P2P capabilities such as indexing, file sharing, searching, peer grouping and security. JXTA's end point layer abstracts communication details independent of transport protocols and can be implemented on top of a variety of transport protocols. Recent efforts from OMG to provide a transport framework for plugging in transports with sufficient predictability can be found in [11]. This effort seeks extensions to the Real-time CORBA 1.0 specification. The JMS [12] specification abstracts interactions in publish-subscribe environments. These interactions can be implemented on top of a variety of transport protocols. The specification itself however does not include a separately transport framework. Proteus [13] is a multi-protocol library for integrating multiple message protocols, such as SOAP and JMS, within one system while supporting the dynamic addition of protocols.

There are many efforts in the area of Internet performance measurement. IP Provider Metrics, which is a subgroup of IETF's Bench Marking Working Group (BMWG), is trying to develop a set of standard metrics that can be applied to the quality, performance and reliability of Internet data delivery services [14]. Cooperative Association for Internet Data Analysis (CADIA) [15], a collaborative effort in engineering and maintenance of the Internet, provides and analyses measurement tools currently available. The Network Weather System (NWS) [16,17] collects end-to-end throughput and latency information and uses that information to forecast future performance. Metrics are collected by sensors, which are organized as a hierarchy of sensor sets called cliques in order to prevent contention and also to provide scalability. NWS also accumulates CPU and available non-paged memory information from various nodes. Bprobe [18] measures the maximum possible bandwidth along the bottleneck link of a given path, while Cprobe [19] estimates the current congestion along the same path. All measurements are non-intrusive. Remos [20] provides a query based interface for applications to obtain information about their execution environment including network state.

In one of the efforts [21] to interface handheld devices to services, the approach involves a dedicated process, which interacts with the server. This server then communicates with specific PDA devices. In our approach this logic could reside in specialized links.

3.0 NaradaBrokering: Brief Overview

NaradaBrokering is a distributed brokering system, implemented on a network of cooperating broker nodes. Broker nodes are organized in a cluster-based architecture, which allows the system to scale to support an arbitrary number of clients. NaradaBrokering provides support for centralized, distributed and P2P interactions. NaradaBrokering has been tested in synchronous and asynchronous applications, including as a media server for audio-video conferencing. These features supported by NaradaBrokering, entail different and sometimes competing networking requirements. The issues enumerated, in the introduction (section 1.0), are thus very relevant to the NaradaBrokering system. Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events.

4.1 The Transport Framework

In the distributed NaradaBrokering setting it is expected that when an event traverses an end-to-end *channel* across multiple broker *hops* or *links*, the underlying transport protocols deployed for communications would vary. In this section we discuss the major components that make up the transport framework. The TransportHandler provides the interface between the transport and protocol layers at a node. The TransportHandler manages all registered LinkFactorys, which are responsible for enabling communications for a specific *type*, while managing the Links created in the process. Link implementations can monitor and report performance data in a specialized construct viz. LinkPerformanceData. Figure 1 provides a brief overview of the main components in the transport framework, we now discuss each of these in detail.

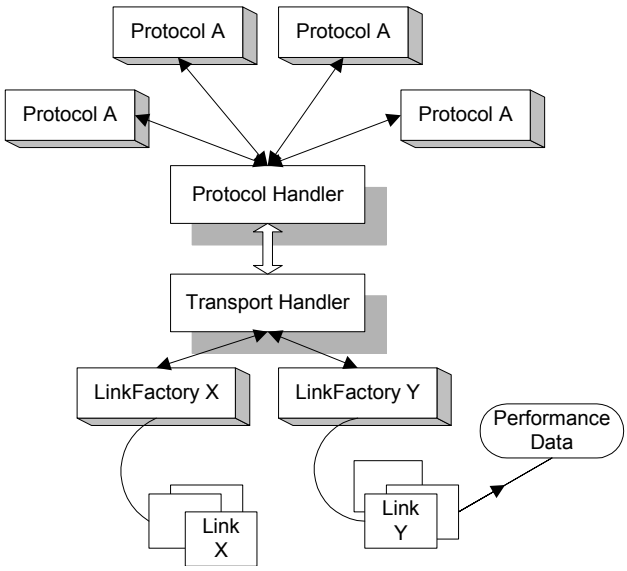

Figure 1: TransportFramework - Main components

4.1.1 Link

Operations that need to be supported between two communication endpoints are encapsulated within the Link primitive in the transport framework. A Link is an abstraction that hides details pertaining to communications. Implementations of the Link interface can incorporate transport-specific handshaking protocols to facilitate setting up of the communication link. Links encapsulate abilities to perform various functions such as

1. *Failure detection*: Links also contain methods, which allow for checking the status of the underlying communication mechanism at specified intervals while reporting communication losses to the relevant error handlers within the transport framework.
2. *Garbage collection*: This pertains to the collection of resources associated with the concept of alternate links, outlined in a subsequent sub-section.
3. *Performance measurements*: Each implementation of the Link interface can expose and measure a set of performance factors.
4. *Transport protocol migrations*: A Link allows the specification of a constraint (usually on the set of performance factors that it measures) and the Link type that communications migrate to, when the constraint is satisfied.
5. *Security Information*: A Link also includes methods to report whether communication over the link is secure, and if so, what the security/encryption mechanism is over the link.

4.1.2 Performance Metrics

Measurement of performance factors over a link requires cooperation, from the two nodes, between which, it is established. Link implementers for different transports have autonomy over the factors they measure, and the strategy they use to do so. Factors measured over a link include round trip delays, jitters, bandwidth, loss rates etc. Individual Links can enable/disable the measurement of a given performance factor or the entire set of performance factors measured for that link. Links expose the performance related information in the LinkPerformanceData construct. Using this construct it is possible to retrieve the list of factors being measured, the *type* of the parameter value, the *value* corresponding to a specific parameter or the complete set of performance data that is measured over the link.

Also important, is the ability of a link to deploy a different transport protocol, when communication using the current transport degrades substantially or is impossible to achieve. Links can specify a constraint on the performance factors measured over a link and specify the migration to another underlying transport protocol when this constraint is satisfied. For example in cases where communications using UDP is not feasible due to high loss

rates, one may consider switching to TCP for communications. Similarly, it is conceivable that while communicating using TCP, bandwidth and latency constraints force a switch to UDP communications.

4.1.3 Administrative Link and Negotiation of Optimal Transport

In the distributed NaradaBrokering setting, different broker nodes may incorporate support for link implementations with different underlying protocols. The framework places no constraints on the number of different implementations of the transport framework. Depending on the firewall, NAT and proxy boundaries that separate the nodes, communication will be possible over a subset of implementations of the framework.

As a pre-cursor to determining the possibility of communications over different transports, information needs to be exchanged between the nodes in question. Information regarding the availability of a specific link types could be encapsulated in an URI, which could then possibly be used to dynamically load services. This information is exchanged over the AdministrativeLink, which is different from Links in the methods that can be invoked on it. Communication over the AdministrativeLink will generally be HTTP based, to ensure the best possibility for communications between two nodes.

The AdmistrativeLink uses the information exchanged over it to determine the optimal transport for communications, between the nodes it is established over. Information exchange over the AdministrativeLink also includes information, pertaining to the supported protocols, such as host, port, multicast group etc.

LinkNegotiators are used by the AdminsitrativeLink to determine the best available link to deploy for communications between two NaradaBrokering nodes. LinkNegotiators are initialized based on information exchanged over the AdminitrativeLink. Initializations for LinkNegotiators are generally similar to those required for the creation of the corresponding Link. LinkNegotiators currently return whether communication is possible using the underlying protocol. It could also be used to return metrics that would enable the administrative link in arriving at a better decision.

4.1.4 LinkFactory

A LinkFactory is responsible for managing Links of a certain communications type, and provides three important capabilities. First, it facilitates the creation of inbound (and outbound) communication links from (and to) other nodes. For example in the case of the TCP communication link, the TCPLinkFactory needs to set up a ServerSocket that would allow TCPLinks to be set up based on the socket connections that are enabled by the ServerSocket.accept(). Second, it manages the migration of communications from a different underlying communications protocol. This is a very important function, and each implementation of the LinkFactory provides a list of communication types, for which, it can manage the migrations. Finally, the LinkFactory can enable or disable failure-detection and performance-monitoring over managed Links, while changing the measurement intervals associated with these important functions.

4.1.5 TransportHandler

Protocol layers use the TransportHandler interface to invoke methods for communications with other NaradaBrokering nodes. The TransportHandler manages all LinkFactorys and Links. Based on the LinkFactorys that are loaded at run-time the TransportHandler can expose the set of link types (generally corresponding to transport types) that it supports. A reference to the TransportHandler is passed on to every Link created by a LinkFactory.

Individual Links use the TransportHandler interface to report data streams that are received over the link, loss of communications and requests to migrate to a different communication protocol by invoking the appropriate methods within the interface. The TransportHandler deals with these notifications within the transport layer, and also propagates appropriate notifications and encapsulated data to the protocol layer.

The TransportHandler also facilitates the creation of *alternate* Links, an important feature to enable effieint communications. While routing events/messages between two NaradaBrokering brokers (over the established link) it should be possible for the event routing protocol to specify the creation of alternate communication links for disseminations. Support for this feature arises when routing handlers request the deployment of specific transport

protocols for routing content, for e.g. a RTP event router, in the protocol layer, could request that RTP-based Links be used for communication. Sometimes such links will be needed for short durations of time. In such cases we should be able to specify the time for which the link should be kept alive. Expiry of this timer should cause the garbage collection of all resources associated with the link. The *keepalive* time associated with a Link corresponds to the period of inactivity after which the associated Link resources must be garbage collected. Figure 2 depicts the issues that we discussed in this section.

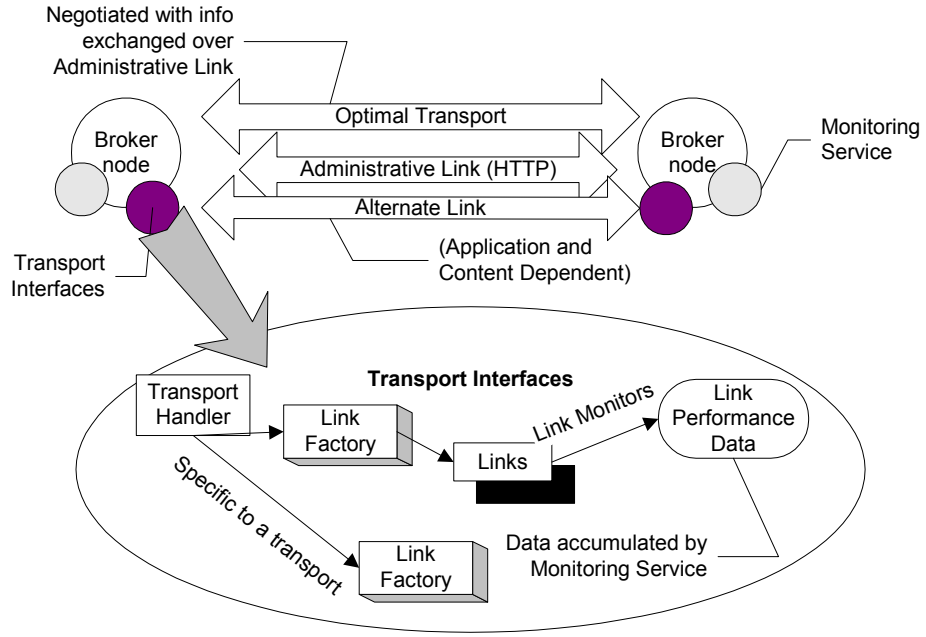

Figure 2: Transport Framework – The bigger picture.

5.0 Implementation Issues

TCP, UDP, Multicast, SSL and RTP based implementations of the transport framework are currently available in NaradaBrokering. It is also entirely conceivable that there could be a JXTA link, which will defer communications to the underlying JXTA pipe mechanism.

The SSL based implementation of the interfaces works with authenticating proxies and supports 3 different authentication mechanisms Basic, Digest and NTLM (a proprietary scheme from Microsoft). This implementation tunnels through firewalls that allow HTTPS traffic.

We provided support for legacy RTP applications by providing a specialized implementation of the transport interfaces. This implementation dealt with managing initializations (and assorted set of operations) mandated for every client connected to a broker. Raw data from RTP applications were packaged into appropriate NaradaBrokering events, with appropriate source and identifiers for intelligent routing within the system. While routing event data back to these applications, only the raw data is routed to the application, every else – the headers, distribution traces and other identifiers for computing destinations – is discarded. In this approach the transport implementation has insulated the both the brokering system and the RTP application from being tightly coupled to each other.

Addition of HTTP support is presently underway. NaradaBrokering can also tunnel through authenticating proxies and firewalls. Once the HTTP implementation is complete the negotiation of transport protocols between two nodes would be addressed.

6.0 Experimental Results

Figure 3 depicts results for the TCP implementation of the framework. The graphs depict the mean transit delays for native NaradaBrokering messages traversing through multiple (2, 3, 5 and 7) hops with multiple brokers (1, 2, 4 and 6 respectively) in the path from the sender of the message to the receiver. For each test case the payload associated

with the message was varied. The transit delay plotted is the average of the 50 messages that were published for each payload. The sender/receiver pair along with every broker involved in the test cases were hosted on different physical machines (Pentium-3, 1 GHz, 256 MB RAM). These machines resided on a 100 Mbps LAN. The run-time environment for all the processes is JDK-1.3 build Blackdown-1.3.1, Red Hat Linux 7.3 The average delay per inter-node (broker-broker, broker-client) hop was around 500-700 microseconds. Figure 4 depicts the standard deviation of the transit delays for message samples used in computing the mean transit delay in Figure 3.

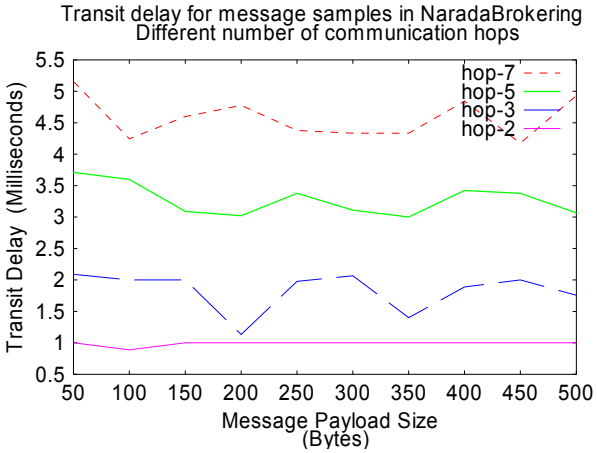

Figure 3: Mean Transit Delays for varying payloads

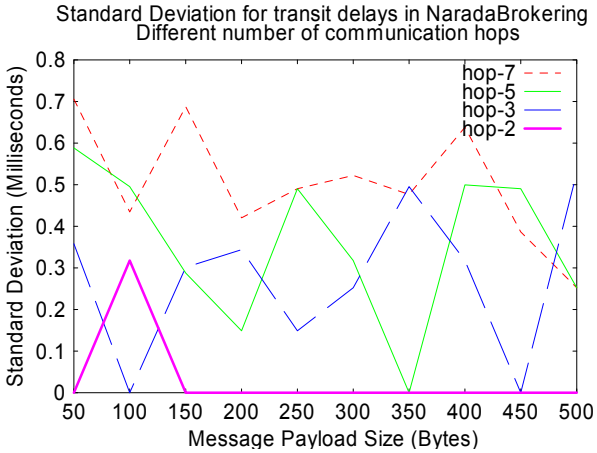

Figure 4: Standard Deviation for varying payloads

We now compare the performance of routing RTP audio packets using the Java Media Framework (JMF) and NaradaBrokering. The client machine (Pentium-3, 2.2 GHz, 1 GB RAM) runs the transmitter and 30 receiver clients (corresponding to the first 10, middle 10 and last 10 of the total 100 clients). The remaining 70 clients are hosted on another machine (Pentium-3, 1.2 GHz, 512 MB RAM). The JMF reflector server and a NaradaBrokering broker are hosted on another machine (dual CPU, Pentium-3, 1.2 GHz, 1 GB RAM). All processes involved in the experimental setup use the Blackdown-1.3.1, Java 2 JRE JVM. The machines reside on a 100 Mbps LAN. Our benchmark uses a ULAW format based audio file, with an average bit-rate of 600Kbps (Kilo bits per second) and a packet (492 bytes) being sent every 60 ms. The transmitter client reads this file from the disk and sends it to the server/broker machine. Then reflector server or the NaradaBrokering broker sends it back to the receiver clients which play it.

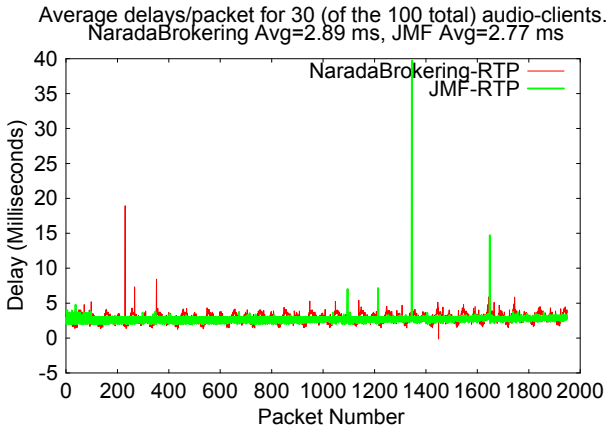

Figure 5: Transit delays comparing NaradaBrokering and JMF

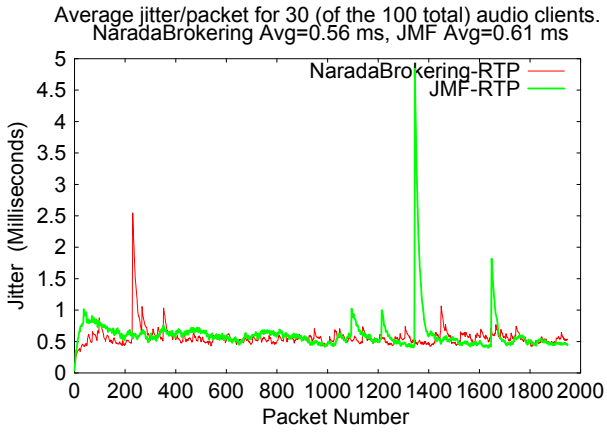

Figure 6: Jitter values, comparing NaradaBrokering and JMF

For every packet that is received we compute the average transit delay associated with the delivery of RTP packets, to *each* of the 10 receiver clients. We also measure the Jitter J, which is defined by the RTP RFC [22] as the mean deviation (smoothed absolute value) of the difference D in packet spacing at the receiver compared to the sender for a pair of packets. The Jitter J is computed based on the formula: $J = J + (|D(i-1, i)| - J)/16$, where $D(i-1, i)$ corresponds to the difference between the delay for i^{th} RTP packet and the delay for the $(i-1)^{th}$ RTP packet. For the

sample of packets that are received we also compute the mean delay and the standard deviation associated with the delays for individual packets. In both cases we ignore the first 50 RTP packets from our calculations since they correspond to application startup. Figures 5 and 6 depict the delays and jitter (up until that point) values associated with individual packets. The results demonstrate comparable performance between NaradaBrokering broker and JMF media server in routing RTP packets.

7.0 Future Work

There are some issues that need to be investigated further. Trade-offs in the language used to specify migration constraints, the evaluation of these constraints and whether it is practical in real time settings needs to be investigated further. We are researching a strategy where links report their constraints to a separate node which would evaluate these migration constraints. This eliminates any delays on links due to computations involved in evaluating constraints. Extending this strategy by incorporating support for specialized links for dealing with handheld devices is an area we plan to explore. Extending support further for RTP clients by including codec transformations in the links is another area that we intend to research to determine the complexities/trade-offs involved in achieving this.

8.0 Conclusions

In the paper we presented a transport framework that is appropriate for distributed brokering systems. The framework sufficiently abstracts operations that need to be supported for enabling efficient communications between nodes. The paper outlined the abstractions within the framework. We also discussed some issues pertaining to supporting different transports within this framework. We also presented some results from our preliminary implementations. The final version of this paper will also incorporate results from HTTP based implementations.

References

1. The NaradaBrokering System http://www.naradabrokering.org
2. A Middleware Framework and Architecture for Peer-to-Peer Grids. Shrideep Pallickara and Geoffrey Fox (To appear) Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.
3. NaradaBrokering: An Event Based Infrastructure for Building Scaleable Durable Peer-to-Peer Grids. Geoffrey Fox and Shrideep Pallickara. Chapter 22 of "Grid Computing: Making the Global Infrastructure a Reality". John Wiley April'03.
4. The Narada Event Brokering System: Overview and Extensions. Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, June 2002. pp 353-359.
5. A Scaleable Event Infrastructure for Peer to Peer Grids. Geoffrey Fox, Shrideep Pallickara and Xi Rao. Proceedings of ACM Java Grande ISCOPE Conference 2002. Seattle, Washington. November 2002.
6. "JMS Compliance in the Narada Event Brokering System." Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Internet Computing (IC-02). June 2002. pp 391-402.
7. "Integration of NaradaBrokering and Audio/Video Conferencing as a Web Service". Bulut et. al.. Proceedings of the IASTED IC Communications, Internet, and Information Technology, November, 2002, in St.Thomas, US Virgin Islands.
8. Sun Microsystems. The JXTA Project and Peer-to-Peer Technology http://www.jxta.org
9. The JXTA Protocol Specifications. http://spec.jxta.org/v1.0/docbook/JXTAProtocols.html
10. Oram, A. (eds) 2001. Peer-To-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly, Sebastapol, CA 95472.
11. Extensible Transport Framework for Real-Time CORBA. OMG document orbos/2000-09-12. Available from http://cgi.omg.org/docs/orbos/00-09-12.txt.
12. Java Message Service. M. Happner, R. Burridge and R. Sharma. http://java.sun.com/products/jms.
13. The Proteus Multiprotocol Library. K. Chiu, M. Govindaraju, and D. Gannon. Supercomputing, November 2002.
14. IETF Benchmark Working subgroup: http://www.ietf.org/html.charters/ippm-charter.html
15. CAIDA http://www.caida.org/tools/
16. R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The Network Weather Service. Tech. Rep. TR-cs97-540, University of California, San Diego, May 1997.
17. R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. Proceedings of the 6th IEEE Symp. On High Performance Distributed Computing, August 1997.
18. R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report TR-96-007, Boston University 1996.
19. R. Carter and M. Crovella. Measuring bottleneck link speed in packet-switched networks. Technical Report TR-96-006, Boston University 1996.
20. B. Lowecamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste and J. Subhlok. A resource query interface for network-aware applications. In Proc. 7th IEEE Symp. On High Performance Distributed Computing, August 1998.
21. G. Fox, et. al. on "Integration of Hand-Held Devices into Collaborative Environments" IC'02.
22. RTP: A Transport Protocol for Real-Time Applications (IETF RFC 1889) http://www.ietf.org/rfc/rfc1889.txt.