A Scalable Middleware Solution for Fast Data Transport

YINGJIN CUI
Department Computer Science
Virginia Commonwealth University
Richmond, VA
USA

ADE OLA
Learning Scope, Incorporated
PO Box 29215
Richmond VA 23242
USA

STEVE DAVIS
Department of Management
Clemson University
101 Sirrine Hall, Clemson, SC 29624-1305
USA

XUE BAI
Department of Computer Information Systems
Virginia State University
Petersburg, VA 23834
USA

Abstract - Multipoint distribution of data is a requirement for various types of applications in such areas as distance learning, multimedia conferencing and group collaboration. The economy of scale of multicast transport for multipoint distribution is not in doubt. The challenge is to develop technologies that will satisfy the varied requirements of these applications. However, IP Multicast has proven difficult to deploy on a large scale, and it can not satisfy the requirements of the plurality of multicast applications. Many multicast protocols have been proposed as alternatives to the open service model of IP Multicast. But no one set of protocols is capable of satisfying the various, often at odds, service requirements of multicast applications with heterogeneous multicast receivers. This paper proposes a middleware approach to provide core protocols, functions and interfaces, the combination of which can be selected and customized to meet the specific demands of various multicast-based applications. At its core, the proposed middleware solution will allow the routing of both datagrams and streams, using multiple channels and variants of the standard transport protocols, over clustered trees that scale well for large number of hosts in a multicast group.

Key-Words: - Data transportation, multimedia streaming, high performance data transfer, middleware, routing

1 Introduction
One of the essential requirements of multimedia conferencing and collaboration applications is an efficient multipoint data distribution technology. The requirements of these and similar applications are varied [1]. In addition to satisfying the varied service requirements of different applications, there is also the need to accommodate, within the same application and multicast group, multiple heterogeneous multicast receivers - with their diversity of bandwidth capacity and delivery delay.

The challenge is to develop multicast technologies that will not only resolve deployment issues of IP Multicast, but also satisfy the varied requirements of multicast applications. Many multicast protocols have been proposed as alternatives to the open service model of IP Multicast, including Source-Specific Multicast [2], reliable multicast projects [3] and application-level multicast protocols [3, 4, 5, 6]. However, no one set of protocols is capable of satisfying the various, often at odds, service requirements of multicast applications having heterogeneous multicast receivers. This paper proposes a middleware solution based on application-level multicast protocol and high-performance data transfer protocols through multiple-socket connection.

2 The Middleware Approach
Middleware is services above the TCP/IP level but below the application environment [7]. The proposed middleware consists of various data transport protocols, services overlay network construction, group management, multicast routing and secure multicast. The new approach solves the problem of satisfying the service requirements of varying multicast applications with heterogeneous multicast receivers through:

- middleware as a protocol and service software library from which multicast application services can be provided,
- clustered tree protocol as a basis for constructing routing structures that will scale well for large number of heterogeneous receivers,
- two types of parallelism through multiple sockets between two nodes and multiple sockets from one node each “connected” to a different node,
- TCP, UDP, and variants of TCP and UDP as transport to provide various levels of reliability, latency, and throughput, and
- populating the library with solutions to provide various degrees of secure multicast.

Each component of the approach helps provide a solution to the problem of the supporting various multicast requirements for groups of receivers with diverse bandwidth capacity and delivery delay:

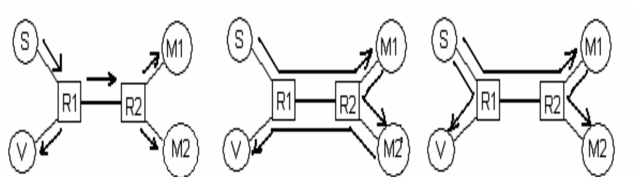
- The clustered tree routing protocol allows dynamic constitution of hosts into a multicast group. Hosts having the least capacities are located at the lower levels of the delivery tree.
- Parallel data delivery through the use of multiple sockets between any two nodes maximizes the throughput, subject to fair use of the network.. Allowing a node to open multiple sockets each “connected” to a different node reduces latency rather than increasing throughput.
- Data delivery through TCP, UDP, and variants of the two protocols provides varying degrees of throughput, latency, and reliability.
- Interfaces to security solutions must be reviewed for their impact on the performance of the multicast application.
- The middleware approach allows the multiple protocols and services to be developed and provided in a library, together with application interfaces for integration with other applications.

3 Middleware Components
This paper focuses on routing protocols and high performance data transfer functions, the core components. Multicast routing will be accomplished through a dynamic clustered tree structure capable of supporting large number of multicast participants within a specified delay constraint. We introduced the Clustered Tree protocol for implementing application-level multicast in [7]. It is based on unicast relay of TCP packets. However, in that protocol, for a host to distribute a packet to multiple (say N) child hosts the packet can only be sent to the Nth host after first N-1 hosts have received the packet. An enhancement to the Cluster Tree protocol will allow parallel packet distribution between a host and its direct children. A brief overview of this protocol is presented in Section 3.1. Section 3.2 describes our implementation of a high performance data transfer application and experimental results.

3.1 Multicast tree protocol

3.1.1 Overview of application-level multicast

In Application-Level Multicast (ALM), hosts participating in a multicast system share the responsibility for forwarding data to other hosts and all packets are transmitted as unicast packets. Thus, multicast-capable routers are not required. However, application-level multicast cannot perform as well as IP multicast, especially with regards to delay and scalability. An ALM tree is an overlay topology where each of its edges corresponds to a unicast path in the underlying structure. The structures in Fig. 1 depict, respectively, IP multicast, application-level multicast and a way of constructing a multicast tree to reduce link stress.



(a) IP (b) Application level overlay with high link stress (c) Application level overlay with low link stress

Fig. 1: Multicast examples

The data source labeled S sends data packets to recipients V, M1, and M2, respectively. Fig. 1(a) depicts the IP multicast protocol, where at most one copy of a packet is sent over any physical link. Each recipient receives data with about the same delay as if by unicast. Figs. 1(b) and 1(c) show application level multicast overlay topologies. In Fig. 1(b), S sends a packet to host M1; M1 sends it to host M2; and M2 sends it to host V. Redundant data transmission over the physical links R1-R2, R2-M1, and R2-M2 increases the link stress on some of the physical links and increases the delay from the source to recipients M2 and V. Fig. 1(c) is a better multicast overlay topology. The same data still appear multiple times over some physical links, but only a single copy of data is sent over the link R1-R2. Communication between two hosts is more reliable and more efficient if the hosts are connected to the same router because transmission will require fewer hops and less time. Therefore, the delay from source S to V is reduced, while the delay from the source to M2 remains the same.

3.1.2 Overview of the clustered multicast tree (CMT) protocol

The CMT protocol is designed to efficiently implement large-scale data distribution on the Internet. The objectives are to (a) construct dynamic multicast topologies to support large number of participants within a delay constraint, and (b) to provide the capability to reconstruct the structures in response to changing network conditions that affect delay and bandwidth utilization. CMT protocol design follows these principles:

- Minimize duplicate packets along router-to-router links which are more costly. Perform the bulk of packet redistribution within host clusters having short inter-host transmission latency.
- Put two hosts in the same cluster only if they share a router or the unicast latency between them is less than a specified threshold. Place hosts in a tree cluster to avoid accumulation of packets in the buffers caused by a host exceeding its capacity to consume packets.
- When data accumulation occurs, the host with the largest total latency among all its direct clients is moved to a lower level in the multicast tree. For a given bandwidth, this scheme forces a host with the longest latency to move to the bottom of its cluster where it will no longer adversely affect other hosts.
- One “dispatcher” host in the system assists a new host to join the clustered multicast tree and handles host departure or failure. It decides whether a host should disconnect its clients or should connect to another host. The use of multiple dispatchers localizes restructuring to each cluster.

A clustered multicast tree consists of a set of clusters, each of which is made up of end host systems such that the multicast latency of data transmission among members of a cluster is minimized. Ideally, no duplicate data packets will traverse the same physical link between routers. Thus this protocol tends to minimize link stress.

3.2 High performance data transfer

Often applications can utilize only a small percentage of the total available bandwidth. For instance, using FTP [8] on a network where the slowest hop from site A to site B is 100 Mbps one may achieve a throughput of only 4Mbps. Research has shown that tuning both the parameters of TCP communication and the TCP configuration of the communication devices themselves can lead to significant performance improvements [9]. In many circumstances, TCP connections require manual tuning to obtain respectable performance [10]. Many variants of TCP have been developed for high bandwidth network [11, 12], but the environments considered are often not characterized by low bandwidth connections.

To investigate how the TCP buffer size and the number of concurrent socket connections influence throughput we developed TCP/IP connection software in Java. We discuss first the impact of TCP window size and number of concurrent sockets on TCP performance followed by other techniques for improving TCP performance.

TCP window size and socket buffer sizes: TCP uses the congestion window (cwnd) to determine how many packets can be sent at one time. The larger the congestion window size, the higher the throughput would be. The TCP "slow start" and "congestion avoidance" algorithms determine the size of the congestion window that is usually less than 65535 bytes. Most operating systems support a much larger window size up to 1,073,741,823 bytes [13], which should be enough for a speed up to 1 Tbps. The maximum congestion window size is related to the amount of the TCP buffer space that an operating system allocates for each socket. For each socket buffer, there is a default value for the buffer size, which can be changed in a program. The buffer size can be modified for both the send and receive ends of the socket. To get maximum throughput, it is critical to use optimal TCP send and receive socket buffer size for the link connection. If the buffers are too small the TCP congestion window will never fully open, and if they are too large the sender can overrun the receiver, and the TCP window shuts down. Table 1 shows the result of transfer rate for varying socket buffer sizes. The experiment involves transferring over the Internet 23.3 megabytes of data between two computers located in Florida and Virginia, respectively. The 65536 byte buffer gives the best transfer rate.

Table 1: Socket buffer sizes vs. transfer time and transfer rate

Socket buffer size	Transfer time (ms)	Transfer rate	
		KBps	Mbps
8192	84750	281.9	2.2
32768	30422	785.2	6.1
65536	26719	894.0	7.0
524288	28641	834.0	6.5
1048576	47172	506.4	4.0

Concurrent multiple socket connections: Multiple sockets may be used to get a better use of the bandwidth capacity, often yielding linear speedup. Table 2 shows the increase in transfer rate with increasing number of connections when the buffer size is set to 8192 bytes.

Table 2: Number of connections vs. transfer time and transfer rate

Number of connections	Transfer time-(ms)	Transfer rate	
		KBps	Mbps
1	77328	308.9	2.4
2	45594	523.9	4.1
3	32188	742.1	5.8
4	26125	914.3	7.1
5	22469	1063.1	8.3
6	19094	1251.0	9.8
7	17219	1387.2	10.8
8	15844	1507.6	11.8
9	14407	1658.0	12.9

Interaction between socket buffer size and number of concurrent connections: To examine how socket buffer size interacts with the number of concurrent connections in impacting the TCP performance, we chose five socket buffer levels. For each buffer level, tests were performed for 1 through 9 concurrent connections. Eleven runs were conducted for each combination of the socket buffer and number of concurrent connections. For each number of concurrent connections, the throughput is maximized at the socket buffer size of 32768 bytes. Throughput increases with increase in buffer size within a certain range but begins to decrease after a certain buffer size level.

There is no optimal number of concurrent connections that generates maximum throughput for all the socket buffer sizes we considered. Nine concurrent socket connections each optimizes throughput when socket buffer is set at 8192 (default buffer size in Java API), 32768 or 524288. A transfer rate of 15.5 Mbps can be achieved with a socket buffer size of 32768 byte and three concurrent socket connections. Compared to the throughout without TCP tuning, this TCP tuning configuration can transfer data about 7 times faster.

When the socket buffer is set at 32768 bytes, increasing the number of concurrent socket connections from 1 through 10 improves throughput.

Since the buffer size at 32768 bytes outperforms other buffer sizes, this buffer size was used to test the effect of increasing the number of concurrent connections from 1 to 20 (Fig. 2). The best transfer times are achieved with the number of concurrent connections at between 8 and 13, after which it increases. Interferences among concurrent socket connections beyond 8 have adverse impact.

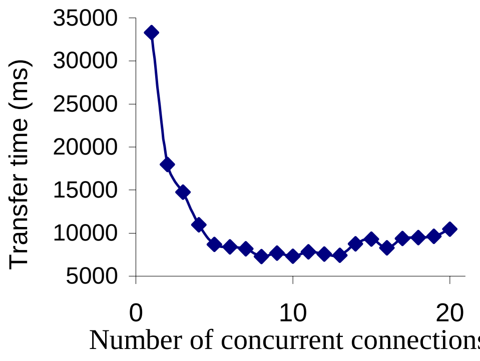


Fig. 2: Transfer time vs. number of concurrent connections

3.3 The middleware

The proposed enhancement to the Cluster Tree protocol will allow parallel packet distribution between a host and its direct children. Then, to construct a multicast system, any number of data transport protocol may be incorporated into the clustered tree routing protocol. Specifically, an instance of the middleware will consist of the following:

- The Clustered Tree routing protocol using TCP as the transport, instead of IP Multicast.
- An enhanced Clustered Tree routing protocol that allows multiple socket connections between a host and one of its direct descendants.
- An enhanced Clustered Tree protocol which allows multiple socket "connections" from a node with each connection to one of its direct descendants.
- One-to-many multicast distribution systems by transporting TCP data streams over each of the routing structures.
- One-to-many multicast distribution systems by transporting UDP data streams over each of the routing structures.

After a complete implementation, the main product will be a middleware to be offered as Software Development Kits and through Application Programming Interfaces to allow dynamic integration of multicast services with other applications. The software library will facilitate development of multicast products for many application areas.

4 Related Work

Several protocols for application-level multicast have been developed [4,5,6].

A repository of "Transport Protocols other than Standard TCP" is in [14]. Other parallel data distribution research and software includes Psockets [15], Java Parallel Secure Stream for Grid Computing [16] and GridFTP [17].

The Grid architecture emphasizes the identification and definition of protocols and services first to be followed by APIs and SDKs [18]. RelayCast suggests finding the common functions of ALM systems and incorporating them into a middleware [19].

What distinguishes our approach is the prospect of handling the plurality of multicast applications through a middleware that provides application-level multicast routing and high performance data transfer functions.

5 Conclusion

This new approach provides a framework for combining application level multicast with parallel data distribution to produce protocols and services as a middleware, to satisfy diverse multicast application requirements. The middleware services will then made available through either SDKs for constructing multicast applications or through APIs to allow dynamic integration of multicast services with other applications. The framework can lead to innovative implementations capable of meeting the diverse and conflicting requirements of multicast applications.

References:

[1] Yamamoto, M., Multicast Communications – Present and Future, *IEEE Transactions on Communications*, Vol. E86-B, No. 6, June 2003, pp.1754-1767

[2] Bhattacharyya, S., RFC 3569 - An Overview of Source-Specific Multicast (SSM), July 2003, http://www.faqs.org/rfcs/rfc3569.html

[3] Lucas, M., Dempsey, B. & Weaver, A., MESH-R: Large-Scale, Reliable Multicast Transport IEEE International Conference on

Communication (ICC '99), Vancouver, BC, June 1999, pp. 657--665

[4] Bai, X., Ola, A., Cui, Y. & Ikem, F., A Clustered Tree Method for Implementing Application Level Multicast, *3rd International Symposium on Information and Communication Technologies,* Las Vegas, Nevada. June 16-18, 2004

[5] Jannotti, J., Gifford, D. Johnson, K., Kaashoek, M. & O., J., Jr., 2000, Overcast: Reliable Multicasting With An Overlay Network, *Proceedings OSDI '00*, pp. 197-212

[6] Pendarakis, D., Shi, S., Verma, D. & Waldvogel, M., ALMI: An Application Level Multicast Infrastructure, *Proceedings of 3rd Usenix Symposium on Internet Technologies & Systems (USITS'01)*, San Francisco, March, pp. 49-60.

[7] Aiken, B., Strassner, J., Carpenter, B., Foster, I., Lynch, C., Mambretti, J., Moore, R. & Teitelbaum, B., RFC 2768 - Network Policy and Services: A Report of a Workshop on Middleware, 2000, http://www.faqs.org/rfcs/rfc2768.html

[8] Postel, J. & Reynolds, J., RFC 959 - File Transfer Protocol (FTP), 1985, http://www.faqs.org/rfcs/rfc959.html

[9] Leese, M., Grid-Mon – TCP Tuning, 2003, http://gridmon.dl.ac.uk/tcp_tuning.html

[10] Semke, J. Mahdavi, J. & Mathis, M., Automatic TCP Buffer Tuning, Computer Communication Review, *ACM SIGCOMM*, Vol. 28, No. 4, October 1998, pp. 315-323

[11] Survey of Transport Protocols Other than Standard TCP, http://www.evl.uic.edu/eric/atp/

[12] Chen, J., Akers, W., Chen, Y. & Watson, W. III, Java Parallel Secure Stream for Grid Computing, High Performance Computing Group, Thomas Jefferson National Accelerator Facility, http://www.jlab.org/hpc/papers/jparss.pdf

[13] Jacobson, V., Braden, R. & Borman, D., RFC 1323 - TCP extensions for high performance, 1992, http://www.faqs.org/rfcs/rfc1323.html

[14] Goutelle, M., Gu, Y., He, E., Hegde, S., Kettimuthu, R., Leigh, R., Primet, P., Welzl, M. (Ed.), Xiong, C. & Yousaf, M., Suvery of Transport Protocols Other Than Standard TCP, 2004, http://www.gridforum.org/Meetings/ggf10/GGF10%20Documents/Survey%20DT-RG.pdf

[15] Sivakumar, H., Bailey, S. & Grossman, R., PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks, *Proceedings of SuperComputing 2000*, Dallas, Texas. November 2000

[16] Chen, J., Akers, W., Chen, Y. & Watson, W. III, Java Parallel Secure Stream for Grid Computing, High Performance Computing Group, Thomas Jefferson National Accelerator Facility, http://www.jlab.org/hpc/papers/jparss.pdf

[17] Allcock, W. (Ed.), GridFTP: Protocol Extensions to FTP for the Grid, April 2003, http://www.ggf.org/documents/GWD-R/GFD-R.020.pdf

[18] Foster, C., Kesselman, C. & Tuecke, S., The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of Supercomputer Applications*, Vol. 15, No. 3, 2001, http://www.globus.org/alliance/publications/papers/anatomy.pdf

[19] Mimura, N., Nakauchi, K., Morikawa, H. & Aoyama, T., RelayCast: A Middleware for Application-level Multicast Services, *3rd International Symposium on Cluster Computing and the Grid*, May 12 - 15, 2003, p. 434