Exploiting Group Communication for Reliable High Volume Data Distribution

Jeremy R. Cooperstock
Department of Electrical & Computer Engineering
University of Toronto
Toronto, Ontario M5S 1A4
afdp@ecf.toronto.edu

Steve Kotsopoulos
Engineering Computing Facility
University of Toronto
Toronto, Ontario M5S 1A4
afdp@ecf.toronto.edu

Abstract - The design and implementation of a protocol to provide reliable and efficient distribution of large quantities of data to many hosts on a local area network or internetwork is described. By exploiting the one-to-many transmission capabilities of multicast and broadcast, it is possible to transmit data to multiple hosts simultaneously, using less bandwidth and thus obtaining greater efficiency than repeated unicasting. Although performance measurements indicate the superiority of multicast, we dynamically select from available transmission modes so as to maximize efficiency and throughput while providing reliable delivery of data to all hosts. Our results demonstrate that file-distribution programs based on this protocol can benefit from a linear speed-up over TCP-based programs such as rdist.

I. INTRODUCTION

In large workstation clusters, performing frequent file updates is often an expensive task. The update process can be automated with programs such as **rdist** or **track**, but their use of connection-oriented protocols such as TCP [9] can be very inefficient. This is because the data must be transmitted over the network multiple times, once to each target.

By exploiting the one-to-many capabilities of modern local area networks, it is possible for a system to send data to multiple hosts simultaneously, thereby greatly reducing network traffic, host load, and elapsed time. Since broadcast and multicast packets can be sent only as datagrams, it is necessary to forgo connection-oriented protocols and instead implement the system using connectionless protocols such as UDP [8]. We have designed and implemented such a system, called the *Adaptive File Distribution Protocol* (AFDP).

AFDP shares some ideas of CFDP [6], NETBLT [3] and Armstrong's multicast transport protocol [1], but generalizes them to provide a scalable, flow-controlled data distribution mechanism that capitalizes on available communication modes.

II. II. THE AFDP PROTOCOL

AFDP is intended for distributing large quantities of data to a group of machines, rapidly and reliably. This protocol is based on the publishing model [10], coupled with an adaptive flow control mechanism for efficiency. Any machine receiving data is called a *subscriber* while any machine sending data is called a *publishe*r. One special subscriber is designated as the *secretary*; this machine is responsible for managing group membership, authorizing publishers, and determining the appropriate transmission mode to be used. Publishing does not require explicit acknowledgments of received data. Instead, subscribers use negative acknowledgments as a means of requesting retransmission of missing or corrupted packets. To provide adaptive flow control, the transmission rate is increased while subscribers are receiving packets correctly. If any packets are dropped or incorrectly received, the necessary data is retransmitted, and the rate is decreased.

AFDP was developed as a protocol that would automatically determine the *best* transport mode, according to the number of subscribers, their capabilities, and support from the connecting networks. The protocol is implemented in two co-operating programs, similar in function to **rcp**, but capitalizing on the broadcast [7] and multicast [5] facilities of UDP. With its additional tuning features, AFDP allows the user control over the trade-off between speed (throughput) and host and network loading.

The interaction between subscribers, publishers, and secretary is depicted in Figure 1. Subscribers wishing to receive messages may join the group at any time by contacting the secretary. The identity of the secretary is either specified on the command line or found by broadcasting or multicasting a FIND_SECRETARY query to a well-known search port. If a secretary for this group exists, it will reply, providing the identity of other ports required for communication with subscribers and publishers. Otherwise, if no secretary replies, the

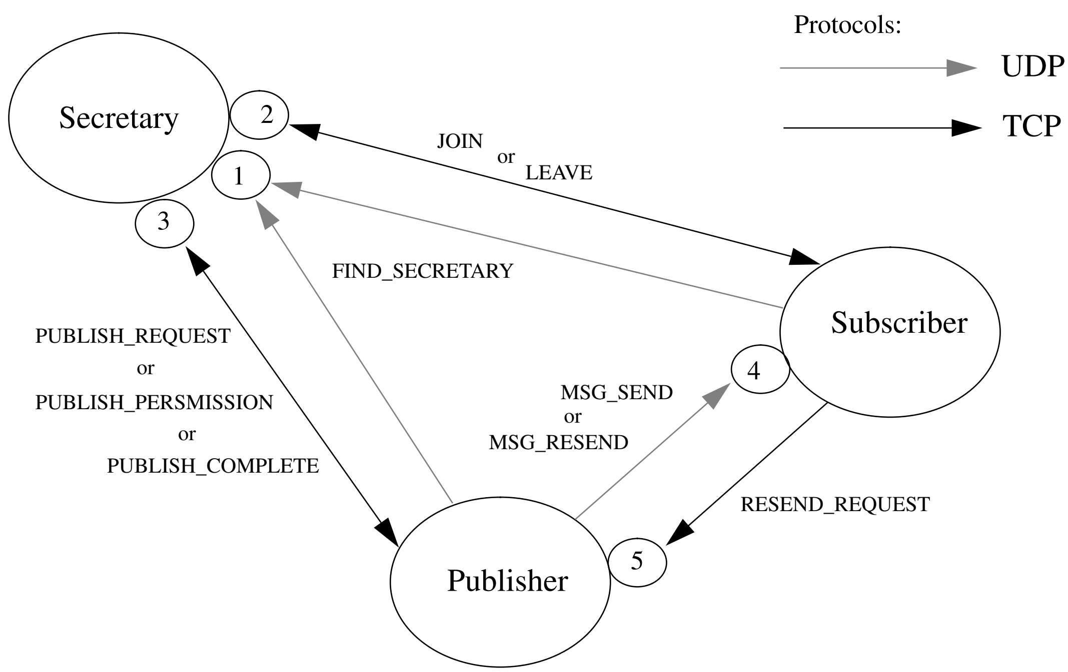

Figure 1. Interprocess communication in AFDP. The ports are defined as follows: 1. search port. 2. admin port. 3. publish port. 4. group data port. 5. control port. Note that a group may consist of multiple subscribers and publishers, although for simplicity, we only show one of each.

subscriber may become the group secretary, thereby creating a new group. The secretary may disband the group at any time by sending a SHUTDOWN message to each subscriber.

The secretary maintains information about the group in the form of a host table. It listens on three ports for messages; a *search* port for FIND_SECRETARY queries, an *admin* port for JOIN and LEAVE requests from subscribers, and a *publish* port for PUBLISH_REQUEST messages from publishers.

When a publisher wishes to transmit a message, it first sends a PUBLISH_REQUEST to the secretary, who in turn will either refuse the request[1] or grant permission. When a host is granted publishing permission, it is told which transmission mode to use, as well as an initial suggested inter-packet interval, T_p. If unicast will be employed, the secretary also provides the publisher with a current list of subscribers.

The secretary, in consultation with the publisher, is responsible for selecting a transmission mode that is appropriate for all subscribers. Since multicast is considerably more efficient than broadcast, this mode is chosen whenever all hosts support it. However, if some hosts do not support multicast, the secretary must choose between unicast and broadcast. Broadcast packets cannot travel beyond a LAN, so this option is only viable when all subscribers are on the same LAN as the publisher. If so, the secretary will specify unicast when the number of subscribers is below some threshold, and broadcast, otherwise. Note that when the group consists of two or more subscribers, unicast mode will require multiple transmissions of each packet. The threshold at which AFDP switches from unicast to broadcast mode can be controlled by the user, allowing one to influence the trade-off between network loading and CPU usage.

Once granted permission, the publisher transmits the message as a sequence of data packets at intervals of T_p milliseconds. Each packet contains the current sequence

1. The secretary will only refuse a PUBLISH_REQUEST if the number of active publishers exceeds some threshold. We are presently adding features to prevent all but a select number of hosts from publishing.

number, as well as the total number of packets in the message. Under the negative acknowledgment scheme, subscribers are silent unless they require the publisher to resend one or more packets. Subscribers perform a sequence check function at intervals of a fixed wait time, T_w, to determine if any packets are outstanding. This will be true if there is a gap in received sequence numbers[2], or if no packets have arrived since the last sequence check. If any packets are outstanding, the subscriber connects to the control port of the publisher via TCP, and issues a RESEND_REQUEST. Provided that the request is valid[3], the publisher first increases the value of T_p, then services the retransmission request before returning to normal publishing. The more RESEND_REQUESTS serviced, the more the publisher must slow down.

This rate-based flow control mechanism minimizes unnecessary retransmissions. As soon as one subscriber reports that it has missed a number of packets, the publisher is forced to decrease its transmission rate, thus relieving pressure on the network or the subscriber, whichever is suffering from overload. While this technique reduces publisher throughput, it keeps the number of retransmissions reasonably low.

Similar to VMTP [2], the publisher periodically decreases the value of T_p to ensure that it is transmitting at the maximum rate the subscribers can handle. In order to prevent AFDP from overloading the network, we typically enforce a reasonable minimum on T_p. However, a *rabid* mode that can be used to override this minimum, as well as a *nice* mode that raises this minimum to 100 msec, are both provided. Note that we provide very little fault tolerance. If a network partition arises, or a subscriber host fails, we make no attempt to recover from the error.

III. PERFORMANCE

An ideal solution to the problem of efficient file update in large workstation clusters would be to utilize AFDP as the transport layer protocol of existing file update programs such as **track** or **rdist**. This merge may be carried out as part of a planned rewrite of **rdist** [4]. In the meantime, we developed our own **rdist**-like program, **afdpdist,** as a proof-of-concept. This program can distribute files to a group of machines in time independent of group size,

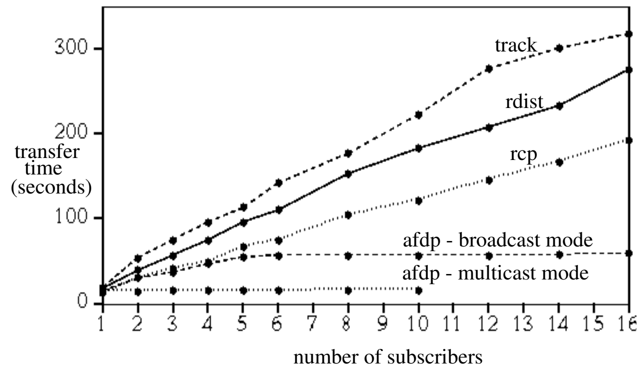

Figure 2. File distribution time vs. number of subscribers for various file-update programs. In each case, the entire contents of a 7.0 Mbytes directory were transferred to each host. The afdp-multicast plot stops at 10 subscribers since only 11 multicast-capable hosts were available. We did not consider the publisher to be a valid subscriber for these tests.

provided that subscribers are reasonably well-behaved and do not flood the network with retransmission requests. In our experience, retransmission requests have not been a problem, even for a group size of over 30 hosts.

Figure 2 compares the performance of **afdpdist** with various other file update programs. For each program, we distributed a directory containing 7.0 Mbytes of files of various sizes to subscriber hosts. The measurements were made on the University of Toronto's Engineering Computing Facility (ECF), a heterogeneous workstation cluster consisting of one MIPS RC6280, eleven SGIs, ranging from R3000 Indigos to R4400 multiprocessors, and 30 SUN Sparc10 systems. The machines are on a lightly-loaded Ethernet, connected to several other LANs. Similar results were obtained on various other clusters.

As can be seen, **afdpdist** offers a linear speedup, proportional to the number of subscribers, over any TCP-based file distribution program. For a large number of hosts, **afdpdist** distributes files far more efficiently than presently available programs such as **rdist** and **track**. While beta testing, John DiMarco **<jdd@cdf.toronto.edu>** reported:

> ''AFDP performance was impressive. Sending **/etc/passwd** to 68 machines took twenty seconds. The equivalent **rdist**[4] took 251 seconds.''

While high throughput is attractive, it is important that AFDP does not impact other uses of the network. To test this, **ftp** transfer times between a MIPS 6280 and a Sparc 10 were compared on the same network under the

2. Note that a packet is only considered successfully received if its data was not corrupted.

3. A RESEND_REQUEST is valid if the subscriber is an authorized group member and the packets it has requested have already been sent to the group.

4. John DiMarco is making use of the recent enhancements to **rdist**, provided by Cooper, which perform multiple TCP transfers in parallel.

following four conditions: relatively idle, with a large **rcp** transfer running, with a large AFDP transfer running in broadcast mode, and finally, with a large AFDP transfer running in unicast mode to three subscribers. The results of Table 1 indicate that the worst **ftp** performance was obtained while **rcp** was in use. Since **rdist** and **track** also use TCP, their performance impact would be similar. In contrast, AFDP puts less of an impact on other users of the network than currently popular methods.

Table 1. ftp throughput and network load under various conditions. The AFDP transfer was to three subscribers in both broadcast and unicast tests.

network	ftp throughput (Kbytes/s)	total network load (%)
idle	470	44
rcp	150	80
AFDP (broadcast)	430	50
AFP (unicast)	200	50

Results from our initial experiments using AFDP across multiple networks appear in Figure 3. The graph demonstrates that even in the worst case, when the publisher must unicast to all subscribers, the performance is still approximately linear in group size.

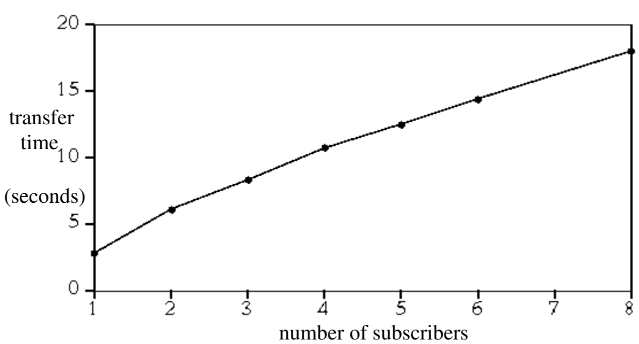

Figure 3. File distribution time vs. number of subscribers for AFDP running in internet mode. All subscribers were located on remote networks from the publisher, and had to receive by unicast mode. In each case, the transferred file was 1.7 Mbytes.

IV. CONCLUSIONS

The group communication capabilities of multicast and broadcast offer an appealing alternative to repeated unicasting of data to many machines. To achieve reliability and efficiency, a retransmission scheme and rate-based flow control mechanism must be added. AFDP provides these features, allowing large amounts of data to be distributed quickly to multiple hosts on a LAN. Changes in available network bandwidth as well as subscriber capabilities are dynamically accommodated.

Our initial implementation of **afdpdist** demonstrates the potential of file distribution that is independent of the number of participating hosts. Even for relatively small clusters, the improved performance of **afdpdist** over **rdist** or **track** is significant. Other applications making use of AFDP may benefit similarly.

AVAILABILITY

The source code for AFDP is now available via anonymous ftp from **ftp.ecf.toronto.edu:/pub/afdp/** as public domain software for research and educational purposes only. Companies or institutions wishing to make use of AFDP for commercial purposes should contact **afdp@ecf.toronto.edu**. AFDP has been tested under IRIX, SunOS, Solaris, RISC/os, Ultrix, and SVR4.2. Our sources include a library of convenience functions for writing network applications, which can be re-used in other programs.

REFERENCES

1. Armstrong, S., Freier, A., and Marzullo, K. Multicast Transport Protocol. RFC 1301. February, 1992.

2. Cheriton, David R. VMTP as the Transport Layer for High-Performance Distributed Systems. *IEEE Communications Magazine*, 27(6), June 1989.

3. Clark, David D., Lambert, Mark L., and Zhang, Lixia. NETBLT: A Bulk Data Transfer Protocol. RFC 998. March, 1987.

4. Cooper, Michael A. Overhauling Rdist for the '90s. In Proceedings of *Large Installation System Administrators Workshop Proceedings* (LISA VI). Long Beach, CA, 1992.

5. Deering, Steve. Host Extensions for IP Multicasting. RFC 1112. August, 1989

6. Ioannidis, J. and Maguire G. Jr. The Coherent File Distribution Protocol. RFC 1235. June, 1991.

7. Mogul, Jeffrey. Broadcasting Internet Datagrams. RFC 919. October 1984.

8. Postel, Jon. User Datagram Protocol. RFC 768. USC/Information Sciences Institute. August. 1980.

9. Postel, Jon. Transmission Control Protocol. RFC 793. USC/Information Sciences Institute. September, 1981.

10. Powell, M. L. and Presotto, D. L. Publishing: A reliable broadcast communication mechanism. Proceedings of the *9th Symposium on Operating Systems Principles*, pgs. 100-109, New York, 1983.