Why Use a Fishing Line When You Have a Net? An Adaptive Multicast Data Distribution Protocol [*]

Jeremy R. Cooperstock and Steve Kotsopoulos
Department of Electrical and Computer Engineering
University of Toronto

Abstract

The design and implementation of a system 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 our protocol can benefit from a substantial speed-up over TCP-based programs such as `rdist`. For example, our system has been used to distribute a 133 Kbyte password file to 68 hosts in 20 seconds, whereas the equivalent `rdist` took 251 seconds.

1 Introduction

Distributing data to multiple hosts using connection-oriented protocols such as TCP [1] can be inefficient because the data must be transmitted over the network multiple times, once to each target. Popular file distribution programs, including `rcp`, `rdist`, and `track` [2], are all based on this protocol. The time they require to distribute files to a group of machines is proportional to the number of machines in that group.[1]

Modern local area networks, such as Ethernet and FDDI, support one-to-many communication via broadcasting [4] and multicasting [5]. By exploiting these capabilities, 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 [6].

Unfortunately, the UDP protocol is unreliable: neither delivery nor ordering of UDP data packets is guaranteed. A further problem is that broadcast packets cannot travel outside of a local area network, and while multicast does not suffer this limitation, it is not supported by all hosts. To address these problems, we designed and implemented the *Adaptive File Distribution Protocol* (AFDP). AFDP provides reliable, rate-controlled delivery of data to multiple hosts, automatically determining the *best* transmission mode, according to the number of hosts, 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 connectionless 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.

Using the AFDP programs as building blocks, we have constructed a prototype file update application, called `afdpdist`. Although it is commonly held that a performance penalty must be paid for a large group, our experimental results indicate that `afdpdist` can distribute files to a group of

[*] Permission has been granted by the USENIX Association to reprint the above paper. This paper was originally published in the 1996 USENIX Technical Conference Proceedings. Copyright © USENIX Association, 1996.

[1] Recent improvements to `rdist` include provisions for parallel update [3], which involves sending files to a number of clients simultaneously. Using this method, file distribution time can be reduced by a factor of almost four. However, repeated unicasting is still used as the transmission mode, and thus, distribution time remains dependent on group size.

machines in time independent of group size. While `afdpdist` is rather crude, we hope that it can be used as a proof-of-concept to design and build a better `rdist`.

We now survey related work to AFDP, then describe the AFDP protocol. Following this, we discuss the performance of our system and summarize some applications of AFDP including `afdpdist`.

2 Related Work

The most relevant related work is that of the *Reliable Multicast Protocol* (RMP) [7], recently implemented by Whetten et al. Similar to the MBusI [8] and the Totem protocol [9], it provides reliable, ordered delivery of data. Like AFDP, RMP goes beyond these earlier systems by allowing multiple, simultaneous senders, and does not rely on hosts to provide multicast support.

Similar systems have been proposed and implemented, but these generally rely on a single transmission mechanism. For instance, Ioannidis et al. implemented CFDP [10], a one-to-many distribution system without any flow control facilities. Oki et al. implemented *The Information Bus* [11], which uses a retransmission protocol to provide reliable delivery semantics. Because these systems implement their group communications using broadcast, exclusively, they cannot be used to distribute data beyond a LAN.

Clark et al. describe the NETBLT protocol [12] for rapid transfer of large quantities of data between computers. To achieve high throughput, NETBLT uses a transmission rate-control algorithm similar to ours. However, since this is a connection-oriented protocol, it is not applicable to efficient group communications.

Other research has dealt with multicast transport protocols and flow control problems. For example, Cheriton describes the *V Distributed System* [13], which also utilizes multicast communication primitives, but only offers "best-effort," not reliable delivery. Armstrong et al. propose a *Reliable Multicast Transport* [14] protocol to provide a network service that could be used to implement a system such as ours. Unlike AFDP, though, it cannot be supported by hosts that do not have multicast capability.

Birman et al. constructed the ISIS system [15, 16], which offers reliable broadcast and multicast as part of its toolkit approach to distributed systems design. Unfortunately, ISIS requires separate acknowledgements from each destination, which limits performance and scalability.

3 The Adaptive File Distribution Protocol

AFDP was designed with the goals of efficiency and flexibility. We wanted to distribute large files[2], typical of operating system upgrades and the increasingly common image and MPEG files, to all group members reliably, efficiently, and as quickly as possible. To provide maximum flexibility, multiple hosts should be able to transmit data concurrently. The network, as well as all participating hosts, are assumed to support (unreliable) broadcast transmissions, and may also support multicast. It is also assumed that varying network and host loads will cause fluctuations in available network bandwidth as well as reception capabilities of the hosts.

These requirements motivated an approach that capitalizes on available communication modes and provides a low-overhead, rate-based flow-control strategy to ensure reliable delivery. Note that we presently provide very little fault tolerance. If a network partition arises between group members, or a host fails, we make no attempt to recover from the error. This is an area for future work.

3.1 Flow Control

Traditional flow control in transport protocols including *stop-and-wait* and *go-back-N* were designed with point-to-point connections in mind [17]. Because these techniques require positive acknowledgements from hosts receiving data, their performance tends to suffer as the group size, and hence return traffic to the sender increases. The *selective-repeat* technique addresses this shortcoming with a negative acknowledgement scheme: the sender transmits the entire message, as a series of message packets, to a group of receivers that collate the packets by sequence number. If a receiver discovers that it is missing a packet, by detecting a gap in the sequence numbers, it asks the sender to retransmit it. However, this technique may suffer if the network is frequently dropping data, thus requiring the sender to retransmit many packets.

AFDP combines selective-repeat with a rate-based flow control strategy to prevent unacceptable packet losses. Like Henriksen [17], we wish to

[2]The current range of sequence numbers allows AFDP to support up to 4 terabyte files.

avoid wasting bandwidth by unnecessary retransmissions. Therefore, our system attempts to maintain a transmission rate that is as high as possible without resulting in dropped packets.

3.2 Protocol Overview

AFDP is based on the publishing model [18], in which any host receiving data is called a *subscriber* while any host sending data is called a *publisher*. One special subscriber is designated as the *secretary* for each group; this host is responsible for managing group membership, authorizing publishers, and determining the appropriate transmission mode to be used. Publishing does not require explicit acknowledgements of received data. Instead, subscribers use negative acknowledgements as a means of requesting retransmission of missing or corrupted packets.

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 port. If no secretary replies, the subscriber may become the group secretary, thereby creating a new group. In the case of races, a voting procedure selects the secretary with the highest IP number. The secretary may disband the group at any time by sending a SHUTDOWN message to each subscriber.

Because the secretary is typically a long-lived process running on a reliable host, whereas subscribers may join and leave the group at any time, we separated the functionality of these two processes. The secretary maintains information about the group in the form of a host table. It listens on three ports for messages: one well-known 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.

JOIN and LEAVE requests may be issued by subscribers at any time. Each provides the identity of the subscriber, and the name of the group the subscriber wishes to join or leave. In addition, the JOIN request specifies whether or not the subscriber is capable of receiving multicast packets. The secretary replies to these requests with a JOIN_REPLY or a LEAVE_REPLY message, as appropriate.

While the secretary will allow hosts to join groups even while data is being published, these new subscribers may not influence the publishing rate until the start of the next message. If, in the event that data packets for the current message are being distributed in multicast or broadcast mode, all subscribers may receive these packets. To avoid confusion, new subscribers simply ignore packets from messages that were begun earlier than their JOIN_REPLY timestamps.

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 or grant permission. 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. If a host is granted publishing permission, it is told which transmission mode to use, as well as an initial suggested interpacket 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. (A planned enhancement to AFDP would allow the secretary to instruct the publisher to multicast to some hosts and unicast to others.) Since broadcast packets cannot travel beyond a LAN, this option is viable only when all subscribers are on the same LAN as the publisher. If so, the secretary will select 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 tradeoff 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 number, as well as the total number of packets in the message. Under the negative acknowledgement 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

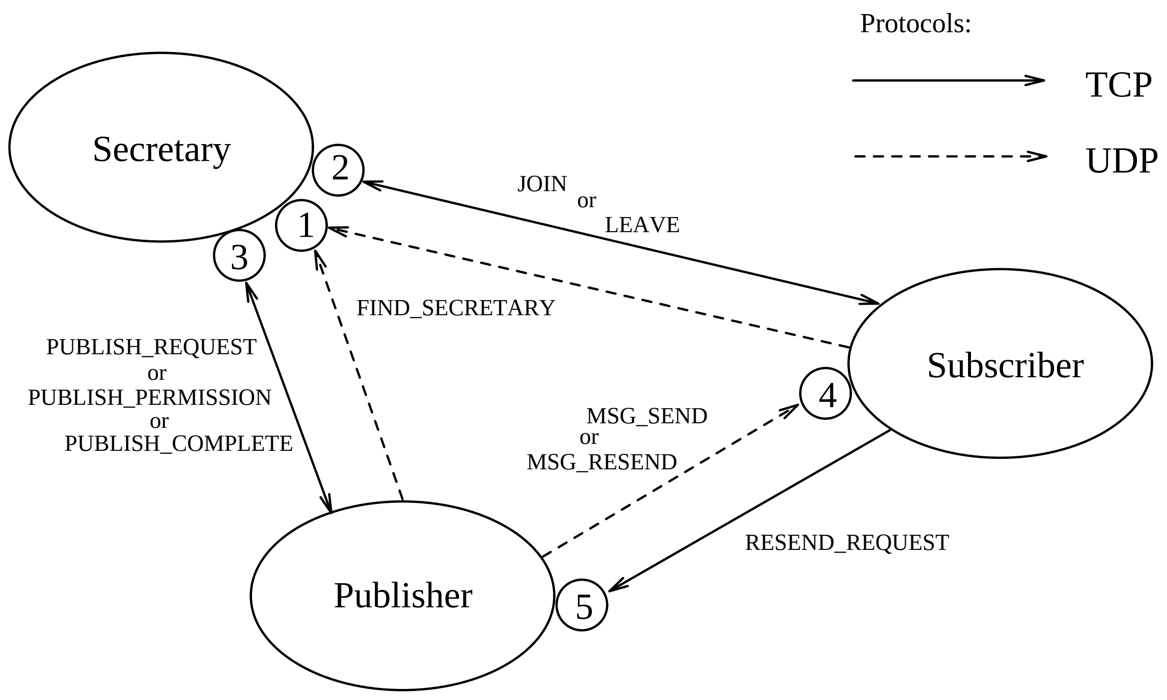

Figure 1: Interprocess communication in AFDP. The ports are defined as follows: 1. search port. 2. admin port. 3. publish port. 4. group data port. 5. control port. Note that a group may consist of multiple subscribers and publishers, although for simplicity, we only show one of each.

sequence numbers or if no packets have arrived since the last sequence check. Note that a packet is considered successfully received only if its data was not corrupted.

If any packets are outstanding, the subscriber issues a RESEND_REQUEST, which is simply a negative acknowledgement of the missing packets, to the control port of the publisher. This contact is handled via TCP, which incurs a minor penalty for the connection set-up time, but ensures reliable delivery of the NAK. This is of paramount importance, especially in light of the role this message plays in forcing the publisher to slow down its transmission. Provided that the request is valid (i.e. the subscriber is an authorized group member and the packets it has requested have already been sent to the group), 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 technique may present a danger in the case of temporary partitions, in which the publisher could potentially be flooded by an implosion of negative acknowledgements. To reduce the likelihood of this occurring, subscribers delay a random amount of time before requesting resends of missing packets. A RESEND_REQUEST is then issued only if some packets are still missing at the end of the delay. Furthermore, all retransmitted packets are sent to the entire group, based on the assumption that if one host has missed a packet, it is likely that others have too. We are presently experimenting with additional techniques intended to reduce further the chance of multiple subscribers requesting a resend of the same packet. One such method is for the publisher to inform subscribers in advance which packets it is about to resend.

Similar to VMTP [19], 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 provided as options.

Under ordinary circumstances, our rate-based flow control mechanism minimizes unnecessary retransmissions. As soon as one subscriber notices that packets have been dropped, the publisher is forced to decrease its transmission rate, thus relieving pressure on the network. While this tech-

packet size (bytes)	publisher throughput (Kbytes/s) unicast	multicast	broadcast
9000	820	818	N/A
4096	405	407	N/A
2048	203	204	N/A
1472	143	142	143
1024	101	102	102

Table 1: Publisher throughput vs. packet size for different transmission modes. Note that throughput for multicast and broadcast modes is independent of group size, provided all hosts are approximately the same speed and under the same load.

nique reduces publisher throughput, it keeps the number of retransmissions reasonably low.

3.3 Packet size

Table 1 demonstrates that using larger packets to transmit data provides greater throughput by reducing protocol overhead and thus increasing efficiency [19] [17]. Therefore, we are motivated to avoid the use of broadcast, as this transmission mode imposes a limit of 1472 bytes on packet size. This limit is due to the fact that broadcast packets cannot be fragmented by the IP layer, and so must fit into a single 1500 byte Ethernet frame [20]. Allowing for the UDP and IP headers, the maximum size of a broadcast UDP packet is 1472 bytes. However, on some MIPS machines, we found that the largest allowable broadcast UDP packet size was 1468 bytes, a value MIPS selected to work around a DMA problem, so we use this value for portability. Other architectures allow a broadcast packet size of 1472 bytes.

We observed that for the same packet size, throughput is relatively independent of transmission mode. As a result of the multicast data packet size limit being over six times that of broadcast packets, we obtain a corresponding improvement in peak throughput, as seen in table 1. Hence, we strongly favor multicast over broadcast whenever both are supported. Ioannidis' CFDP [10] uses a packet size of 512 bytes to avoid fragmentation but we did not observe any benefit in doing so.

3.4 Internetwork Extensions

There are two considerations in scaling this protocol to an internetwork environment. We note that multicast is not universally supported and that broadcast packets cannot travel beyond a LAN. To cope with these restrictions, we presently unicast to remote subscribers, unless all subscribers and intervening routers support multicast, in which case we use that transmission mode.

Where multicast is not supported, subscribers and publishers on remote LANs must specify the identity of the secretary host on the command line. This way, the exchange of information with the secretary can proceed with unicast packets.

4 Performance

The performance measurements of Figure 2 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 R4000 Indys to R4400 multiprocessors, and 30 SUN Sparc10 systems. The machines are on a lightly-loaded Ethernet, connected to several other LANs. Unless noted otherwise, all of our measurements were performed using the default AFDP parameters.

4.1 Distribution Time

For each test, a 7.0 Mybte file was transferred from the publisher to all subscribers. The peak publisher throughput observed for AFDP was between 800 and 900 Kbytes/s in unicast and multicast modes, and 140 Kbytes/s in broadcast mode. AFDP was also evaluated on various other clusters and across multiple clusters with similar results. When multicast is supported by all hosts and the intermediate networks, transfer time is determined by the network and the speed of the slowest machine, rather than the number of subscribers. As Cooperstock and Kotospoulos discuss in a previous paper [21], even in the worst case, when unicast must be used to all hosts, performance is still approximately linear in group size.

As can be seen from Figure 2, the performance of AFDP is primarily determined by the communication mode used. In unicast mode, data must be sent to each subscriber separately, so the transfer time is proportional to the number of hosts. However, in broadcast or multicast mode, data is sent only once, provided no subscribers request a retransmission, and received by all subscribers, so the transfer time remains constant, independent of group size. Using broadcast, AFDP distributed the file in 57 seconds, while with multicast, the time was 15 seconds.

When one or more subscribers are not capable of receiving packets at the current publishing rate, they communicate this to the publisher through

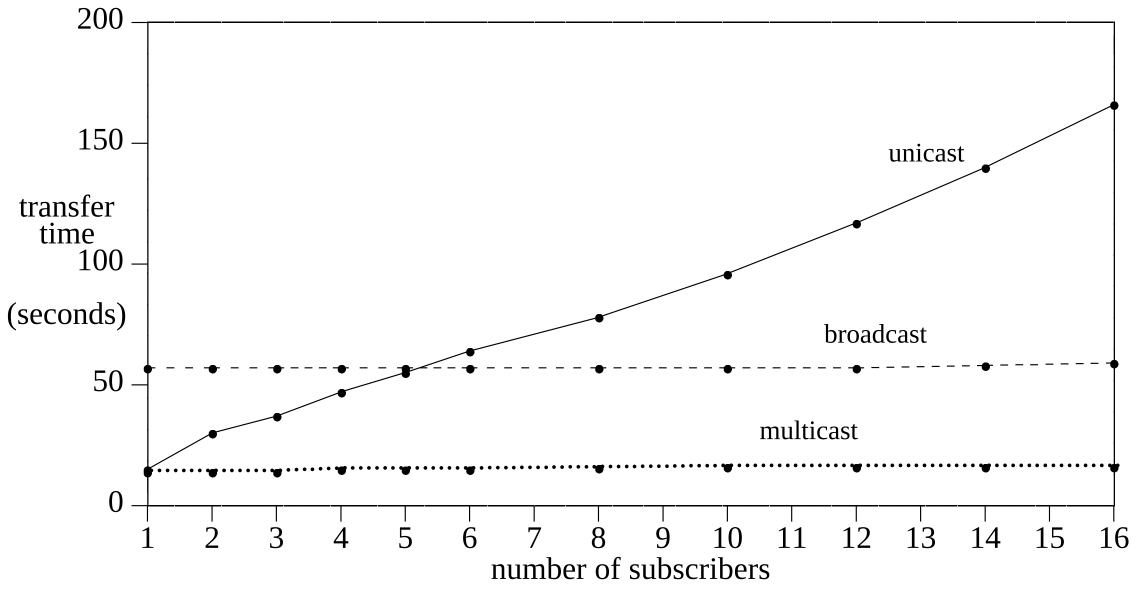

Figure 2: AFDP transfer time vs. number of subscribers using unicast, multicast, and broadcast modes. The transferred file was 7.0 Mbytes in each case. Note that because of the smaller packet size required by broadcast data, this mode takes approximately six times as long as multicast (or unicast to only one subscriber).

RESEND_REQUESTS. This means that the publishing rate can be dictated by the slowest subscriber in the group. While this may not be a desirable characteristic for all applications, our focus on reliable file distribution required such an approach.

We measured peak `ftp` performance at approximately 900-1000 Kbyte/s between two SGI R4000 and between two Sun IPC machines, although typical average throughput may be substantially less. AFDP consistently matches this performance.[3] on lightly loaded machines. While these numbers may seem unimpressive for our protocol, it must be noted that AFDP can maintain similar publisher throughput for many subscribers, provided that multicast is supported by all members. In this case, files may be distributed to an entire group as quickly as they could be to a single subscriber, whereas for any TCP-based protocol, the distribution time is proportional to group size. As pointed out by one reviewer, the linear relationship of distribution time to group size for TCP-based protocols could be beaten by arranging the receivers (subscribers) on multiple ethernet segments. However, we submit that such architectures are not always available.

[3]On the Sun IPC machines, this figure is attained in *rabid* mode. In order to be "network-friendly," we normally avoid this mode.

4.2 Adaptivity

As shown in the saw-tooth graph of Figure 3, AFDP dynamically adapts to current network conditions as determined by subscriber feedback. While subscribers are receiving data packets correctly, the publisher slowly decreases the inter-packet delay, T_p, to a minimum of 10 ms. However, for every RESEND_REQUEST issued by a subscriber, the publisher immediately increases the inter-packet delay by 10 ms. This balances the need for quick adaptation to problems with a conservative attempt to maximize throughput.

4.3 Transmission Mode

Since broadcast packets are received by every host on the LAN, they can waste CPU cycles on non-participating systems. For example, on an SGI Indy R4000PC, the processing of broadcast packets accounted for approximately 11% of CPU usage. For this reason, we allow the user to select the threshold at which AFDP switches from unicast to broadcast, via a command-line argument. Sites that wish to avoid broadcasts can set this threshold as desired, to tailor the tradeoffs for their environment. Based on the transfer times of unicast and broadcast modes in the previous graph, it would appear logical to select a threshold value of six.

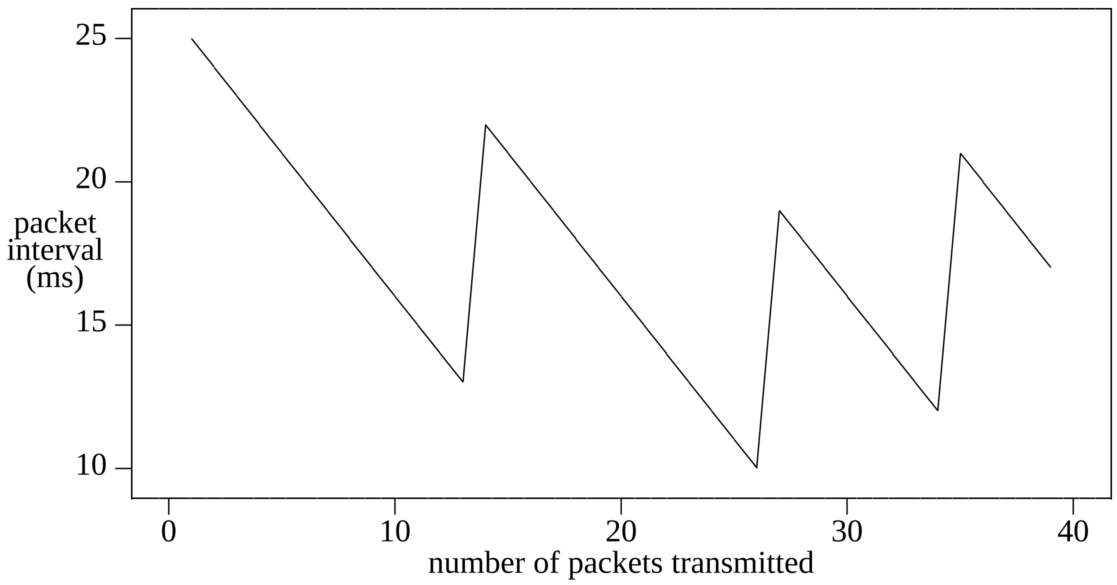

Figure 3: Inter-packet delay, T_p, as a function of packet number. Publishers decrease the delay by 1 ms/packet unless a subscriber issues a RESEND_REQUEST. In this case, the publisher increases the delay by 10 ms, resulting in the saw-tooth appearance of the graph.

network	ftp throughput (Kbytes/s)
idle	470
`rcp`	150
AFDP (broadcast)	430
AFDP (multicast)	200
AFDP (unicast)	200

Table 2: `ftp` throughput while `rcp` or AFDP is running. The AFDP transfer was to three subscribers in each case.

4.4 Network Load

While high throughput is attractive, AFDP, by default, does not attempt to consume all available network bandwidth. However, in the *rabid* mode, AFDP will attempt to utilize the full bandwidth available. To test the default mode, an `ftp` transfer between a MIPS 6280 and a Sparc 10 was run on the same network under the following conditions: relatively idle, with a large `rcp` transfer running, and finally, with a large AFDP transfer running in each of the three transmission modes. The results of Table 2 indicate that while AFDP impacts other users of the network, it is no worse in this regard than `rcp`, which, by contrast, consumes far more bandwidth.

The performance of our file update program, `afdpdist`, discussed in further detail in Section 5.1, was then compared 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 results, shown in Figure 4, demonstrate that `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`.

5 Applications

AFDP has been implemented as two co-operating programs, which may be run on most Unix platforms and without any kernel modifications. One program, `afdpjoin`, handles subscriber and secretary functions, and the second, `afdpsend`, handles publisher functions. The publisher can also specify an external program to be run by each subscriber after receiving the file. This allows AFDP to be used as a building block in the construction of larger distributed applications, such as the ones we now present.

5.1 The `afdpdist` Program

In large workstation clusters, there are often many files that must be updated frequently. This task

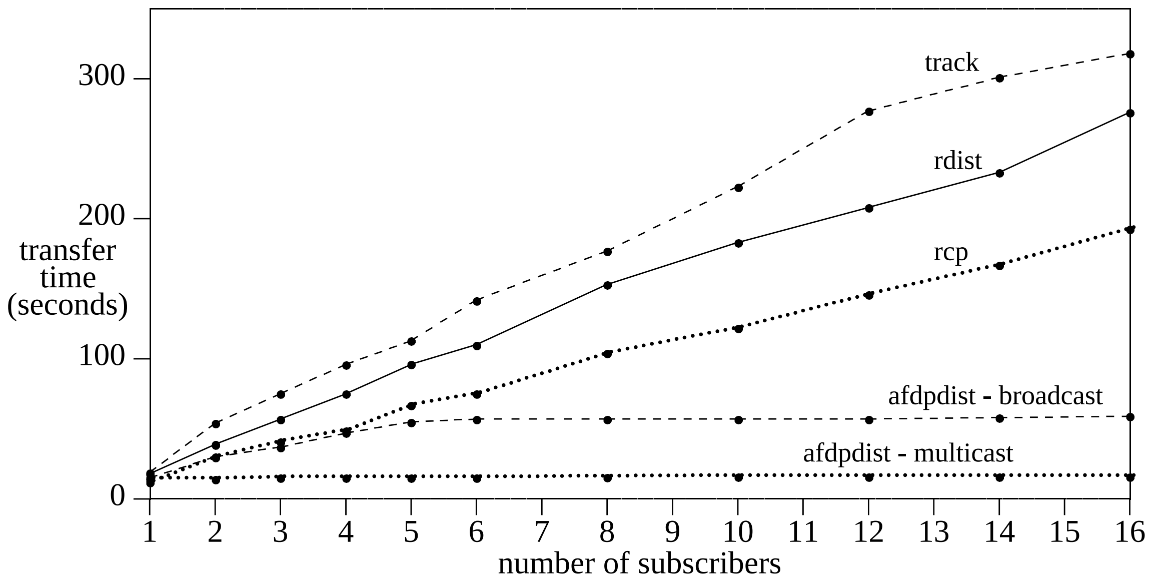

Figure 4: File distribution time vs. number of subscribers for various file update programs. In each case, the entire contents of a 7.0 Mbytes directory were transferred to each host. Peak Ethernet utilizations for the various programs were: `rcp` 95%, `rdist` 88%, `track` 74%, `afdpdist` (multicast mode) 90%, `afdpdist` (broadcast mode) 15%. The decreased ethernet utlization of `afdpdist` in broadcast mode vs. multicast and unicast modes is due to the limiting effect of system calls on the smaller broadcast packets. Also note that the original version of `rdist` was used for these tests.

can be automated with programs such as `rdist` or `track`, but their use of point-to-point communication can be very inefficient if many of the hosts are on the same local area network. For example, on the ECF cluster, an 800 Kbyte `/etc/passwd` file is tracked from the master server to 40 client systems every five minutes.

An ideal solution would be to utilize AFDP as the transport layer protocol of `track` or `rdist`, an idea we have discussed with Michael Cooper, who has been tuning the performance of the latter. Cooper agrees, but notes that this would require major changes to `rdist`. Therefore, we feel that this merge would best be accomplished as part of the re-write he is planning [3].

In the meantime, we developed our own `rdist`-like program, `afdpdist`, as a proof-of-concept. This allowed us to benchmark AFDP against other file distribution applications and to demonstrate the suitability of our protocol to such tasks.

While beta testing AFDP, John DiMarco `<jdd@cdf.toronto.edu>` reported:

> Performance was impressive. `afdpsend` of `/etc/passwd` to 68 machines took twenty seconds. The equivalent `rdist`[4] took 251 seconds.

For the following discussion, we adopt terminology from `track`. An *slist* is a subscription list, containing a list of files and directories to be distributed, and a *statsfile* contains file size and modification time information for each entry of the *slist*.

When `afdpdist` is invoked on the master machine, it will generate a *statsfile* for the specified *slist* and send this file to all subscribers using the AFDP protocol. Upon receipt of the master *statsfile*, each subscriber will generate its own local *statsfile* and compare the two to produce a list of requested files. This list is then sent back to the master over a TCP connection. When the master has received replies from all subscribers, it combines their request lists and generates a bundle of all the necessary files. The bundle is then distributed by AFDP to all subscribers, which unbundle the files they requested.

This scheme works well provided that most systems require the same files, such as after a software upgrade on the master system. If only one host needs files, then this can be very wasteful. In this case, it would be better to unicast files to the one host.

The current implementation of `afdpdist` is nei-

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

ther feature-rich, nor does it include as many configuration options as `rdist` or `track`. For simplicity, the program uses `tar` to bundle and unbundle files, and shell scripts to carry out operations whose performance is not critical. For example, the command, `'cat request.* | sort | uniq > bundle.list'`, is used to merge together the lists of files requested by subscribers. Also, the *slists* specification is presently very restrictive: only a single directory tree can be specified, with no provision to exclude subdirectories.

5.2 Conferencing

The ease with which AFDP supports group communication makes it ideally suited to distributed "slide-show" applications, in which files are simultaneously sent to all participants.

We have implemented an external front-end to AFDP, called `magic-cat`, which takes appropriate action based on the contents (as determined by the `file` utility) of any received files to develop sample applications of this sort. For example, `magic-cat` uses `xloadimage` to view GIF and JPEG images, and `ghostview` to display Postscript files. We have successfully used this application in Computer Science and Engineering tutorials, allowing the instructor to control the demonstration, just as in a real slide-show.

6 Conclusions

The group communication capabilities of broadcast and multicast 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 dependent only on the network and the speed of the slowest machine, rather than 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 available through the URL `http://www.ecf.toronto.edu/afdp` or via anonymous ftp from `ftp.ecf.toronto.edu:/pub/afdp/`. Questions should be addressed to `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] Jon Postel. Transmission Control Protocol. RFC 793, USC/Information Sciences Institute, September 1981.

[2] Daniel Nachbar. When Network File Systems Aren't Enough: Automatic Software Distribution Revisited. In *Proceedings of the Summer USENIX Conference*, Atlanta, GA, 1986.

[3] Michael A. Cooper. Overhauling Rdist for the '90s. In *Large Installation System Administrators Workshop Proceedings (LISA VI)*, pages 1–8, Long Beach, CA, 1992.

[4] Jeffrey Mogul. Broadcasting Internet Datagrams. RFC 919, October 1984.

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

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

[7] Brian Whetten, Simon Kaplan, and Todd Montgomery. A High Performance Totally Ordered Multicast Protocol. Submitted for publication, 1995.

[8] Alan Carroll. *ConversationBulider: A Collaborative Erector Set.* PhD thesis, Department of Computer Science, Univeristy of Illinois, 1993.

[9] D. A. Agarwal, P. M. Melliar-Smith, and L. E. Moser. Totem: A Protocol for Message Ordering in a Wide-Area Network. In *Proceedings of the First ISMM International Conference on Computer Communications and Networks*, pages 1–5, San Diego, CA, June 1992.

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

[11] Brian Oki, Manfred Pfuegl, Alex Siegel, and Dale Skeen. The Information Bus - An Architecture for Extensible Distributed Systems. In *Fourteenth ACM Symposium on Operating Systems Principles*, pages 58–68, Asheville, NC, December 1993.

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

[13] David R. Cheriton. The V Distributed System. *Communications of the ACM*, 31(2), March 1988.

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

[15] Kenneth P. Birman and Thomas Joseph. *Exploiting Replication.* Addison-Wesley/ACM Press Series, Sape J. Mullender, ed., June 1988.

[16] Kenneth P. Birman. The Process Group Approach to Reliable Distributed Computing. *Communications of the ACM*, 36(12):37–53, December 1993.

[17] Eva Henriksen, Gisle Aas, and Jan B. Rydningen. A Transport Protocol Supporting Multicast File Transfer over Satellite Links. In *Proceedings of Eleventh IEEE Phoenix Conference on Computers and Communications (IPCCC)*, pages 590–596, Scottsdale, Arizona, April 1992.

[18] M.L. Powell and D. L. Presotto. Publishing: A Reliable Broadcast Communication Mechanism. In *Proceedings of the 9th Symposium on Operating Systems Principles*, pages 100–109, New York, 1983.

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

[20] W. Richard Stevens. *UNIX Network Programming.* Prentice Hall, 1990.

[21] Jeremy R. Cooperstock and Steve Kotsopoulos. Exploiting Group Communications for Reliable High Volume Data Distribution. In *Proceedings of IEEE Pacific Rim Conference on Communications, Computers, Visualization, and Signal Processing*, Victoria, BC, May 1995.

Biographical Information

Jeremy Cooperstock received the B.A.Sc. in Computer Engineering from the University of British Columbia, Vancouver, Canada, in 1990 and the M.Sc. in Computer Science from the University of Toronto, Toronto, Canada, in 1992. He is currently working towards the PhD in Electrical and Computer Engineering at the University of Toronto. From 1987 to 1988, he worked at IBM Research in Haifa, Israel, and in 1989, at the IBM T.J. Watson Research Center in Yorktown Heights, New York. His research interests include reactive environments, learning in robotic and autonomous systems, communication in distributed systems, and competitive analysis of trading strategies. He can be contacted via email at `jer@dgp.toronto.edu`.

Steve Kotsopoulos is a Master's student in the Department of Electrical and Computer Engineering at the University of Toronto. Since 1988, he has been a systems programmer at the university's Engineering Computing Facility. His interests include networking, security and distributed systems. He received his B.A.Sc. in Electrical Engineering from the University of Toronto in 1988.

In his spare time, Steve enjoys snowboarding and skateboarding. He can be contacted via email at `steve@ecf.toronto.edu`.