Hybrid Reliable Multicast with TCP-XM

K. Jeacle & J. Crowcroft
University of Cambridge
Cambridge, UK
karl.jeacle@cl.cam.ac.uk

Marinho P. Barcellos
UNISINOS University
São Leopoldo, Brazil
marinho@unisinos.br

Stefano Pettini
European Space Agency
Frascati, Italy
stefano.pettini@esa.int

ABSTRACT

In recent years, much work has been done on attempting to scale multicast data transmission to hundreds or thousands of receivers. There are, however, many situations where an application might involve transmission to just ten or twenty sites. The European Space Agency, for example, carry out regular multi-gigabyte bulk data transfers to a handful of destinations.

Using multicast for this type of application can provide significant benefits including reduced load on the transmitter, an overall reduction in network traffic, and consequently shorter data transfer times.

In this paper we take a fresh look at the problem of deploying reliable multicast. So far, there has been no convincing solution to achieve this. We present a simple hybrid solution which has not been proposed before. The approach taken is to combine unicast with multicast by modifying TCP to support multicast transfers, and run this modified TCP engine over UDP as a userspace transport protocol.

Our goal is clear: reliable bulk data delivery to a moderate number of sites. Unlike some other multicast protocols, our work is complete: we have designed, implemented, deployed and evaluated a protocol which meets this goal.

Categories and Subject Descriptors

C.2.2 [**Communication Networks**]: Network Protocols

General Terms

Design, Experimentation, Reliability

Keywords

Multicast, TCP, TCP-XM

1. INTRODUCTION

Today, small scale group communication will typically take place using unicast transmission. Only when the network is capable, and the application demands it, will multicast be used.

Depending on the state of unicast and multicast connectivity between source and destinations, the right combination of unicast and multicast transmission can provide a large increase in efficiency at the cost of a small decrease in speed.

We believe that for large sender-initiated data transfers to a small group of receivers, not only does such a combined unicast and multicast transmission sweet spot exist, but that it can be quantified, and automatically found during the transmission process. Applications can then initiate fully reliable data transfers, while lowering network utilisation by taking full advantage of whatever multicast connectivity is available.

With a clear goal of reliable multicast data transfer to a moderate number of receivers, we propose a novel hybrid userspace solution combining unicast TCP and multicast transmission simultaneously. We have taken our work further than many other reliable multicast protocols as we have designed, implemented, deployed and evaluated our solution.

Part of the motivation for our work lies with addressing the practical problem of bulk data delivery faced by High Energy Physicists in a Grid environment, and by organisations such as the European Space Agency (ESA) [9].

A reliable multicast transport protocol could be used experimentally over the Internet, for the distribution across Europe of Earth Observation satellite data from European Acquisition Stations located in several European Countries (e.g. Rome in Italy, Kiruna in Sweden, Masplaomas in Canary Islands). The same data is distributed to a medium-high number of recipients, with a traffic volume greater than one terabyte per day.

Grid Computing is also an important technology that can benefit from a multicast file distribution system. Processing and archiving are often performed in different sites across Europe. Grid technology could allow the use of distributed computing power to improve the performance of data processing; but this requires the basic data, normally in the order of several hundreds of Megabytes, to be transmitted to all Grid nodes. A reliable multicast transport protocol can speed up file transfers between multiple Grid nodes significantly.

We would also like to see an increase in the number of practical tools available that exploit multicast. Most previous research work on reliable multicast has proposed new APIs and protocols that are not present outside of a lab en-

CoNEXT'05, October 24–27, 2005, Toulouse, France.

vironment. This has led to much theoretical analysis, but little in the way of practical deployment. We have designed, implemented, and deployed a publicly available reliable multicast file transfer tool that takes full advantage of our protocol work.

The rest of the paper is organised as follows. Section 2 describes the design of our protocol, including details of how we have modified TCP to support multicast transmission. Section 3 describes our implementation of the protocol, and the software we have built that makes use of it. Section 4 evaluates protocol performance on local and wide area networks, and provides comparisons with other reliable multicast protocols. Section 5 overviews related work and describes how we have taken a different approach. We close the paper with final remarks in Section 6.

2. PROTOCOL DESIGN

Today, applications use TCP for reliable unicast transfers. It is a mature and well-understood protocol. By modifying TCP to deliver data to more than one receiver at a time, and use multicast when available, an application can transparently send data reliably to multiple recipients. Using existing TCP mechanisms, the modified protocol ensures that data is delivered reliably. Multicast transmission is attempted for performance reasons, but fallback to unicast preserves backwards compatibility and reliability guarantees, and capitalises on more than a decade of experience that TCP implementations enjoy.

Network protocols are typically implemented in the kernels of hosts and network devices. Any proposals that require modifications to these protocols imply changes to kernel code. This immediately restricts deployment opportunities. By limiting changes to code that runs in userspace on end stations, new protocols can be developed and evaluated on live networks.

TCP is a reliable end-to-end unicast transfer protocol implemented in host kernels. It is possible to modify TCP behaviour between end stations without any changes to intermediate devices, however this requires kernel changes. If TCP is moved into userspace, changes can be made without modifying the kernel, but again, some form of privileged access is needed by the userspace TCP implementation to directly send and receive packets on a host's network interfaces. While not as significant a barrier to widespread deployment as kernel changes, in practice, this privileged access requirement limits the ease with which new code can be widely tested.

One solution to this problem is to implement a modified multicast TCP over UDP. Userspace applications can freely send and receive UDP packets, so a small shim layer can be introduced to encapsulate and decapsulate the TCP engine's packets into UDP.

We believe that the instant deployment potential offered by a userspace library, coupled with the efficiency provided by IP multicast, compensates for the loss of performance by running the protocol in userspace and using encapsulation. The key advantage of this approach is that any application can make use of the protocol by linking against the supplied library. Because the protocol is not tightly coupled to the application, should it become adopted for widespread use, a native implementation can be built in the kernel to improve performance.

Before discussing a mechanism for introducing multicast capability, some clarification is necessary with regard to the scope of the problem being addressed.

Transmitting data to multiple destinations via multicast will almost always be more efficient than transmitting via unicast, but an increase in transfer speed will be possible only if the source, or a link close to the source, is a bottleneck in the network. To achieve this increase, no additional application-level functionality is required, or indeed desired, at intermediary nodes in the network. The sole requirement is that the sender and receivers are reachable within the same cloud of end-to-end native network multicast connectivity.

The constraining factors are the extent of multicast deployment in the network, and the impact of ACK implosion. But assuming symmetric links, sending 1500 byte data packets and receiving 40 byte acknowledgements will constrain the group size to just under 40 receivers. For small group sizes, we do not believe that ACK implosion is a serious concern.

2.1 Modifying TCP

In this section we will describe our modifications to TCP to support multicast. We call our new protocol TCP-XM.

The key feature of TCP-XM is its focus on combining both unicast and multicast transmission simultaneously. Multicast transmission will always be attempted, but equally, fallback to unicast transmission will always be available.

The TCP-XM approach is a pragmatic one. It states some ground rules and initial assumptions about the target audience (small scale group communications). It lends itself to a prototype implementation over UDP, with no changes required in the network. Unlike most other proposed reliable multicast protocols, this allows the protocol to be implemented and evaluated on a Wide Area Network. This capability will serve as a useful feedback loop into the design of the protocol.

In order to describe TCP-XM functionality, we will walk through a typical TCP-XM session. Let us first state some rules or assumptions about how the protocol should operate:

- TCP-XM is primarily aimed at push applications;
- it is sender initiated;
- the changes to the API are minimal;
- the sender has advance knowledge of destination unicast addresses.

Given these assumptions, the "call process" for a TCP-XM sender is as follows:

1. The application connects to multiple unicast addresses (*not* to a multicast group address).
2. The user can specify a group address for multicast, otherwise a random group address will be allocated automatically.
3. Multiple TCP Protocol Control Blocks (PCBs) are created – one for each destination.
4. Independent 3-way handshakes take place.
5. The group address is sent as option in the SYN packet. Multicast depends on the presence of the group option in the SYNACK. A PCB variable then dictates the transmission mode for the PCB.

6. User data writes are replicated and enqueued on all PCB send queues.

7. Data packets are initially unicast and multicast simultaneously. Multicast packets have protocol type TCP, but the destination is a multicast group address.

8. Unicasting ceases on successful multicasting.

9. Multicasting continues for the duration of the session.

10. All retransmissions are unicast; each PCB has a copy of sent segments that remain unacknowledged.

11. Automatic fallback to unicast transmission for each single receiver takes place after 3 multicast loss events.

12. Unicast & multicast sequence numbers stay synchronised – minimum values across PCBs for congestion and send windows are used. There is one sliding window per receiver and the sender is only allowed to send the data in the intersection of all sliding windows.

13. Connections are closed as per TCP – closing a TCP-XM connection will cause a FIN to be sent to each receiver. The sender will wait until all individual closes are complete.

From the receiver's perspective:

- no API change is necessary;
- normal TCP listen takes place;
- IGMP group join on incoming TCP-XM connect;
- accept data on both unicast and multicast addresses.

When a TCP-XM connection is made, a SYN packet is sent that includes a TCP option specifying a multicast group address. If not specified by the user, the group address is automatically chosen by the protocol. The presence of this option means that the originating host has TCP-XM capability.

If the receiver is running TCP-XM, it can accept the group address offered or, in exceptional circumstances, return the SYNACK with an alternative proposed group address. On connection establishment, an IGMP join request is issued.

Each PCB in a sender's TCP-XM connection maintains the state or "transmission mode" for each particular receiver. The initial state depends on how the protocol has been invoked by the user. Its subsequent value can be one of those shown in Figure 1. The states shown correspond to the following "TX_" transmission modes:

- TX_UNICAST_ONLY- the sender has made an explicit request for unicast operation. This applies to all receivers.
- TX_UNICAST_GROUP - the default starting state. Data is unicast until multicast capability at the receiver has been confirmed.
- TX_MULTICAST_ONLY - the sender has explicitly requested multicast operation. This applies to all receivers.
- TX_UNICAST_GROUP_ONLY - there was no group option in the SYNACK. The receiver cannot receive multicast packets.
- TX_MULTICAST - the SYNACK contained the group option, and the receiver has confirmed its ability to receive multicast data from the sender.

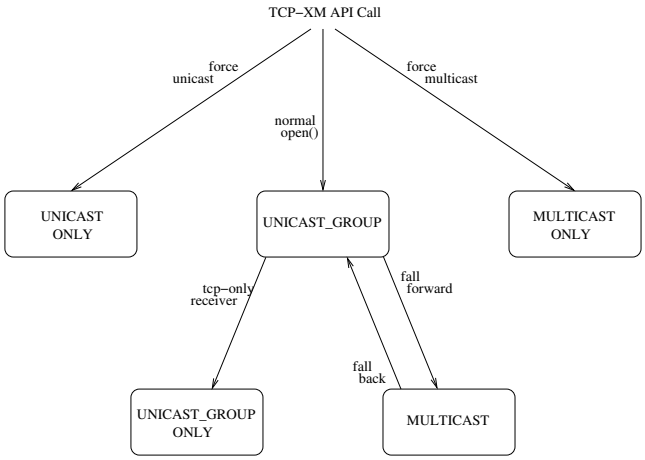

Figure 1: TCP-XM Protocol States

If multicast is working, TX_MULTICAST mode is used. Packet sequence numbers across all PCBs will be kept in synchronisation. If multicast is not possible to a destination, TX_UNICAST_GROUP_ONLY will be used. If multicast fails to a particular site, TX_UNICAST_GROUP mode will be used. Data segments will be unicast, but the sequence numbers will still be kept in line with those being multicast. The reason for this is to keep open the option of switching TX_UNICAST_GROUP connections back to TX_MULTICAST mode.

If the user has explicitly requested unicast-only operation, TX_UNICAST mode will be used. No attempt will be made to synchronise segment sequence numbers with other PCBs in the connection.

If the user has explicitly requested multicast-only operation, TX_MULTICAST will be used. No attempt will be made to fall back to unicast operation.

Note that the "GROUP" tag in the states listed above refers to a group of PCBs with synchronised segment sequence numbers. It does not refer to a multicast group.

2.2 Congestion Control

An important aspect of our work is the approach to congestion control. Because TCP-XM is an extension of TCP, the existing congestion control mechanisms used by TCP are also used by TCP-XM. Unlike other TCP equation based reliable multicast protocols, our protocol is truly TCP-like. The problem of TCP friendliness, therefore, does not arise for TCP-XM; it is, by definition, TCP friendly.

2.3 Multicast Detection

When TCP-XM transmits data via unicast or multicast, an acknowledgement packet will be returned on success. But a standard ACK does not indicate whether a segment has been received via unicast or multicast.

In order for a TCP-XM transmitter to make an informed decision about when to switch from unicast to multicast transmission, it needs to know what the state of the network is, or to be more precise, what the state of multicast reception is like at the receiver.

A feedback mechanism that allows the receiver to inform the sender of multicast reception is desirable. TCP-XM achieves this through the introduction of a TCP header option that indicates the percentage of the last n segments that have been received via multicast. The value of n defaults to 128.

Just a single byte is required to carry the percentage of recent multicast packets received. This option header is included on all acknowledgement packets from the receiver to the sender.

The receiver inspects every segment that arrives and records the sequence number and whether or not the segment was received via unicast or multicast. It then recomputes the percentage of recent packets received via multicast.

The sender simply parses the option on arrival and records the value contained for future processing.

2.4 Avoiding Duplicate Acknowledgements

Because the same segments may be sent via unicast and multicast to a given receiver, it is possible that the same segment arrives twice. To prevent duplicate segments from generating duplicate acknowledgements, the following rules are used:

- If a duplicate segment is received via multicast: drop it. Because segments are never sent via multicast twice (all retransmissions are unicast), if a duplicate segment is received via multicast, the segment must have originally been received via unicast. This means that the sender is unicasting to the receiver, but partial multicast connectivity has led to a multicast segment being received after the original unicast segment.

- If a duplicate segment is received via unicast: check if the original was received via multicast and if so, drop it. This is likely to be a case of network reordering of packets leading to a multicast segment arriving before a unicast segment. Otherwise, should the original segment be unicast, then this must be a retransmission, and a duplicate acknowledgement must be sent.

Checking if a previous segment has been received via unicast is achieved by scanning an array of recorded sequence numbers and checking how the segment was received.

2.5 Fall Forward & Fall Back

With continuous feedback from receivers regarding the state of their multicast reception (i.e. the percentage of last packets received via multicast), it is easy for the sender to decide when to stop transmitting unicast packets, and hence *fall forward* to multicast transmission only.

If a receiver has perfect multicast reception, and a sender is transmitting both unicast and multicast packets, the receiver should be receiving exactly 50% of packets via multicast.

Fall forward takes place when receiver feedback indicates that the percentage of multicast packets received has reached this figure. The PCB's transmission mode is then changed to TX_MULTICAST.

When to fall back to unicast could also be determined using the multicast feedback mechanism, but instead a simple retransmit count is used. When the sender detects three consecutive multicast losses for a given receiver, the PCB's transmission mode is changed to TX_UNICAST_GROUP.

Having fallen back to unicast, falling forward to multicast will happen once again when receiver feedback indicates that 50% of packets received at the remote end were delivered via multicast.

There is, of course, an overhead when unicasting and multicasting at the same time. The worst case scenario is where native multicast capability in the network exhibits intermittent behaviour; if everything is sent twice, then the cost for TCP-XM is $n + 1$ compared to TCP, whose cost is n. At best, the cost for TCP-XM is 1.

3. IMPLEMENTATION

Our implementation of TCP-XM has been built using lwIP [6] as the baseline TCP/IP stack. It has been compiled and tested on FreeBSD, Linux and Solaris.

As stated before, a design choice was to implement TCP-XM in userspace over UDP rather than build a kernel implementation. This facilitates our practical goals of widespread test and deployment.

3.1 lwIP

lwIP is a small independent implementation of the TCP/IP protocol stack that has been developed by Adam Dunkels at the Swedish Institute of Computer Science (SICS).

The original focus of lwIP was to reduce RAM usage while still having a full TCP implementation. This made lwIP suitable for use in embedded systems with limited memory resources. Now open source, these small scale goals still remain.

The size and simplicity of the implementation lends itself to ease of use as a userspace library. This was an important consideration in choosing lwIP over extracting a kernel implementation of TCP from Linux or FreeBSD.

The price paid for this compactness, however, is an inevitable limit on functionality. One key feature missing is TCP window scaling. While clearly not essential for the embedded applications where lwIP would normally be deployed, it does impact on our ability to compete with native kernel TCP on speed.

We make use of lwIP by creating applications that link to the lwIP code and comprise three threads. One thread runs our UDP driver to handle packets coming to and from other hosts; a second thread looks after the operation of the TCP-XM protocol; the third thread is the application mainline itself.

3.2 API

The only API changes necessary for TCP-XM are when the sender initiates a new connection. The user needs a mechanism for specifying multiple destination addresses, and an optional multicast group address.

lwIP applications typically use its "netconn" API, although a traditional socket API is also available. To open a new TCP-XM connection with the "netconn" API:

```
conn = netconn_new(NETCONN_TCPXM);
netconn_connectxm(conn,
  remotedest, numdests, group, port);
```

Instead of `netconn_connect()`, three argument changes are introduced for the `netconn_connectxm()` call: an array is used to specify destination address(es); the number of destination addresses is supplied; and an optional group address is supplied.

Again, note that all other API calls, such as `send()` and `recv()`, remain unaltered.

3.3 PCBs

TCP PCBs are stored on a linked list. This has not been changed, however some extra variables have been added to the PCB structure:

```
struct tcp_pcb {
    ...
    struct ip_addr group_ip;
    enum tx_mode   txmode;
    u8t            nrtxm;
    struct tcp_pcb *nextm;
}
```

The group address and transmission mode are dependent on the TCP group option in the SYN/SYNACK.

The `nrtxm` variable keeps track of how many times a unicast retransmission has been necessary for segments that have previously been multicast. Too many of these retransmissions will result in the transmission mode of the PCB falling back to unicast.

The `nextm` pointer references the next PCB that is part of a chain of PCBs associated with a single TCP-XM connection. Figure 2 shows how five PCBs would be represented in a kernel implementation – M1, M2 and M3 are all part of a single TCP-XM connection, while U1 and U2 are separate independent unicast connections.

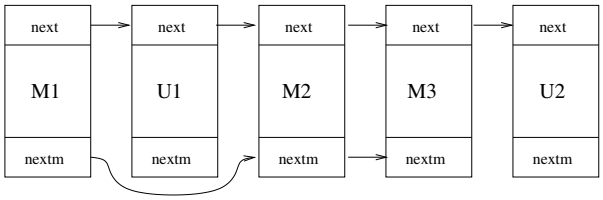

Figure 2: Multicast TCP PCBs

Some additional TCP PCB variables are also required to introduce the multicast feedback functionality:

```
struct tcp_pcb {
  ...
  struct segrcv msegrcv[128];
  u8_t msegrcvper;
  u8_t msegrcvcnt;
  u8_t msegsntper;
};

struct segrcv {
  u8_t ismcast;   /* received via mcast? */
  u32_t seqno;    /* sequence number     */
};
```

`msegrcv` is an array of `struct segrcv`. It is used to record details of the last n packets. The sequence number is stored in `seqno` while `ismcast` is true if the packet was received via multicast.

`msegrcvper` is the percentage of the last n packets that have been received by multicast. The content of this variable is placed directly into the ACK feedback option.

`msegrcvcnt` is a simple looping counter that indexes the most recent packet received in `msegrcv`.

`msegsntper` is the percentage of the last n packets that a remote receiver has received via multicast according to the ACK feedback option.

3.4 mcp & mcpd

As a vehicle for TCP-XM experimentation and deployment, we have created a simple file transfer application.

`mcp` is the client transmitter. It uses native TCP for control connections and TCP-XM (over UDP) for data transfers to `mcpd` receivers on one or more hosts. Both TCP and TCP-XM connections are opened in parallel to destination hosts.

To transfer a file '`myfile`' to hosts '`host1`' and '`host2`', `mcp` would be invoked as follows: `mcp myfile host1 host2`

On invocation, `mcp` begins by opening TCP connections to the destination hosts. It sends the name and size of the file to be transmitted, along with the UDP source port it will use for the subsequent TCP-XM data connection, and a suggested UDP destination port for the servers. The servers check for availability of this suggested port and report back to the client with confirmation or an alternative suggestion. `mcp` negotiates with the servers until an agreed destination port has been found.

With UDP ports agreed, `mcp` initiates its TCP-XM engine, and then opens a TCP-XM connection to the destination hosts. The file to be transferred is read from disk and sent on this connection. On completion, the servers send confirmation of the number of bytes received on the TCP control connection. `mcp` then closes both control and data connections.

Figure 3 shows how TCP control traffic and TCP-XM over UDP data traffic are combined to implement `mcp` & `mcpd`.

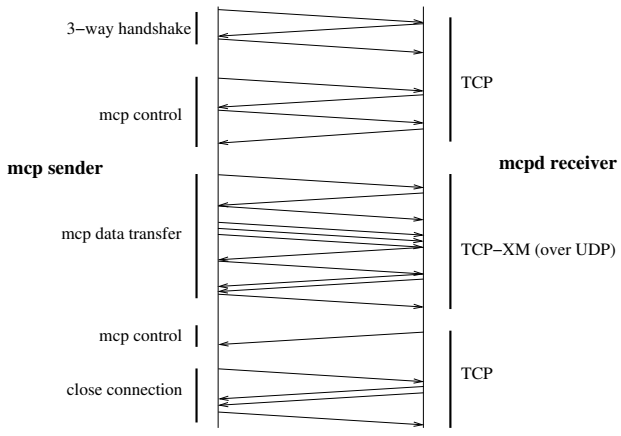

Figure 3: mcp control & data flow

3.5 Grid Support

As mentioned earlier, today's Grid environments demonstrate an application domain where bulk data transfer to a

small number of receivers takes place on a regular basis. For this reason, we have focused some of our efforts on working with Globus [10].

The Globus Toolkit is an open source software toolkit primarily developed by the Globus Alliance. It has become the de-facto standard for middleware used to build Grid services [8, 7].

Globus XIO is an eXtensible Input/Output library for the Globus Toolkit. It provides a POSIX-like API to swappable I/O implementations – essentially "I/O plugins" for Globus [3].

One of the main goals for Globus XIO is to provide a single user API to all Grid I/O protocols. There are many different APIs for many different protocols, and XIO should abstract this complexity for Grid developers.

Globus XIO provides an ideal mechanism for introducing new protocols to Grid users deploying Globus applications. We have wrapped our TCP-XM implementation with XIO to create a multicast transport driver for Globus that can benefit many applications.

4. PERFORMANCE EVALUATION

We have carried out experiments on both local and wide area networks. For local experiments, a selection of departmental workstations were used, along with a smaller bespoke testbed. For wide area experiments, shell accounts on machines at eScience Centres around the UK were obtained. These machines and locations were primarily chosen as they are representative of one of the target audiences for our work i.e. physicists requiring bulk data transfer to a relatively small number of regional sites.

Figure 4 illustrates the geographical connections.

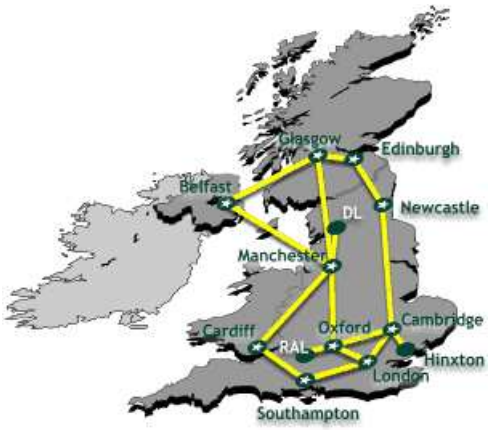

Figure 4: The UK eScience Network

Table 1 lists the eScience Centre hosts used. As the table shows, many of these sites have functional multicast connectivity.

The specifications, network connectivity and operating systems used by the hosts varies widely from site to site. Some hosts are high speed machines connected close to the WAN backbone. Others are smaller and older departmental machines with poorer connectivity.

Table 2 shows the average round-trip times and transfer rates seen from Cambridge to other sites around the network. The round-trip times vary in range from approximately 5 to 21 milliseconds. The transfer rates attainable on single TCP connections varied from as little as 1.5 Mb/s to over 50 Mb/s.

Table 1: UK eScience Testbed Hosts

Site	Hostname	Mcast
Belfast	gridmon.cc.qub.ac.uk	No
Cambridge	mimiru.escience.cam.ac.uk	Yes
Cardiff	agents-comsc.grid.cf.ac.uk	Yes
Daresbury	ag-control-2.dl.ac.uk	Yes
Glasgow	cordelia.nesc.gla.ac.uk	No
Imperial	mariner.lesc.doc.ic.ac.uk	Yes
Manchester	vermont.mvc.mcc.ac.uk	Yes
Newcastle	accessgrid02.ncl.ac.uk	Yes
Oxford	esci1.oucs.ox.ac.uk	Yes
Southampton	beacon1.net.soton.ac.uk	Yes
UCL	sonic.cs.ucl.ac.uk	Yes

While this mixture may not be conducive to optimal headline results, it allows a truly representative set of protocol performance results for a live wide area network.

Table 2: WAN RTTs & Bandwidth

Site	RTT (ms)	B/W (Mb/s)
Belfast	18.6	16.0
Cardiff	13.5	22.4
Daresbury	21.3	28.1
Glasgow	16.2	33.9
Imperial	17.1	51.0
Manchester	9.9	34.5
Newcastle	11.8	1.5
Oxford	7.0	4.0
Southampton	8.8	39.3
UCL	4.9	42.1

All experiments were conducted using `mcp` & `mcpd` and compare TCP-XM (in userspace over UDP) with native kernel-based TCP. Because our implementation of TCP-XM is in userspace, it is at an immediate performance disadvantage to native TCP. Nevertheless, the results provide a useful indicator of the protocol's worth.

Figure 5 shows a comparison of the TCP and TCP-XM data transfer rates to n non-dedicated hosts on a typical local departmental LAN. As would be expected, TCP's throughput declines as the host count increases. TCP-XM peaks at a much lower rate (the speed of the slowest receiver), but then consistently maintains this rate despite the introduction of more destination hosts. Eventually, as the bottleneck increases, TCP-XM outperforms TCP.

Figure 6 shows the number of bytes being sent on the wire for the same transfer. Because TCP-XM is multicasting, it naturally scales. TCP is sending more and more data as the host count increases, so performance inevitably suffers. Note that the TCP-XM data is split in two: unicast bytes and multicast bytes. The unicast bytes are barely visible at the bottom of the graph. These account for connection setup, close, and retransmissions. The majority of the TCP-XM data transfer is composed of multicast bytes.

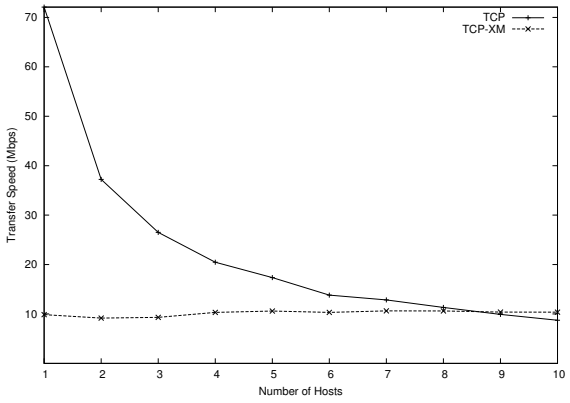

Figure 5: LAN Speed

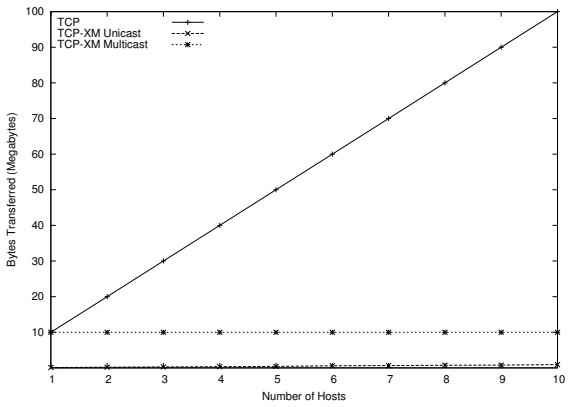

Figure 6: LAN Efficiency

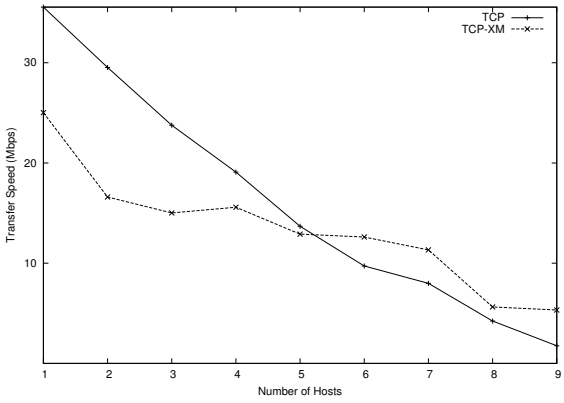

Figure 7: WAN Speed

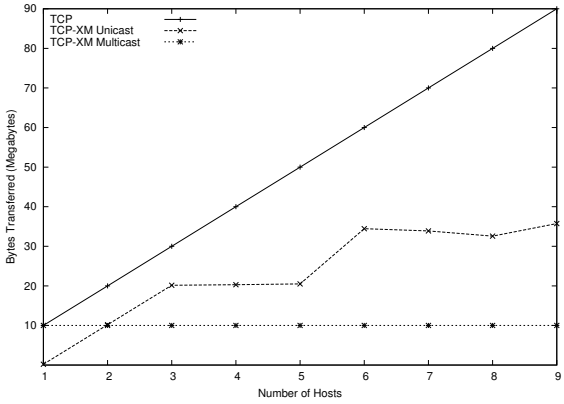

Figure 8: WAN Efficiency

Figure 7 shows how TCP and TCP-XM compare when a transfer to n hosts takes place using a wide area network. As in the local area, TCP can outperform TCP-XM, but less so than might be expected. The inherent bottlenecks present across the WAN, and the varied performance specification of receivers, prevent TCP from achieving the same strong results that are possible on a LAN. For the particular order and set of hosts used in this experiment, the increased bottleneck allows TCP-XM to match or outperform TCP for five or more hosts.

Note that the order in which hosts are added is important. In this experiment, faster hosts with multicast capability are added before slower hosts without multicast.

Figure 8 once again shows TCP's inefficiencies as the host count increases. More interestingly, we can see more clearly how TCP-XM is combining unicast and multicast. Unlike the LAN experiment above, not all destinations are multicast capable, so TCP-XM cannot quickly switch to multicast after connection setup. The number of bytes unicast by TCP-XM is therefore much more significant. There is an obvious step up in unicast bytes sent each time TCP-XM encounters a destination host without multicast.

We should mention again that lwIP currently lacks features such as TCP window scaling. Coupled with some optimisation in our implementation, we would expect considerable improvements in performance to be possible. And, of course, while kernel-based TCP will always have an advantage over a userspace implementation of TCP-XM encapsulated in UDP, the same would not be true of a kernel-based TCP-XM implementation.

In addition, the key benefit from TCP-XM and its multicast capability is not its raw data transfer rate, but its ability to reliably transfer large amounts of data in a far more efficient manner. In an appropriate application domain, end-users may find that the benefits offered by this improved efficiency compensate for any performance penalties incurred by running in userspace.

4.1 Multiple flows

To evaluate how multiple TCP-XM flows compete with each other to share a fixed bandwidth pipe, we have used a small testbed (Figure 9) with Dummynet [13] to perform three TCP-XM file transfers of 256 MB each: PC-C transmits simultaneously to the three receiver pairings of PC-A and PC-B, PC-A and PC-X, and PC-X and PC-B. Sessions are denoted as AB, AX, and XB, respectively.

The sessions run simultaneously, starting approximately at the same time. The network is configured with fixed static delays (5ms each queue) and a loss probability of 0.01%. The bandwidth is varied over time to verify that TCP-XM can

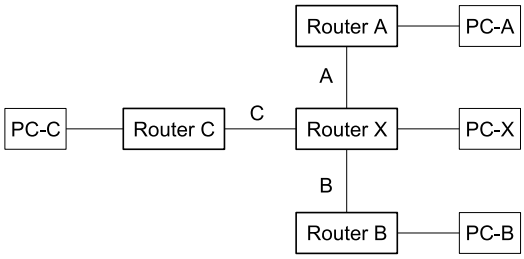

Figure 9: Multiple flow testbed

adapt its transmission rate dynamically. Figure 10 shows the result of the experiment.

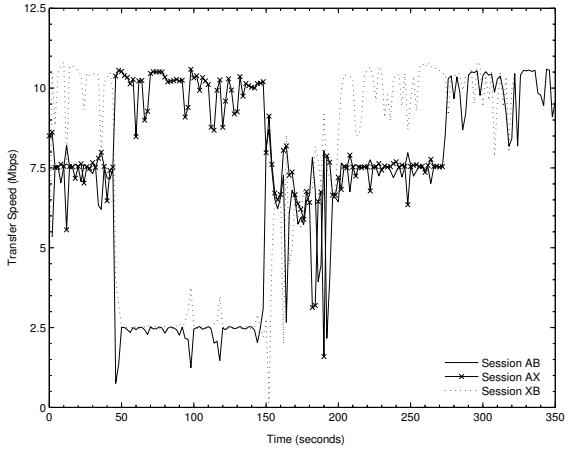

Figure 10: Multiple TCP-XM flows

The experiment is divided into six parts:

1. (t=0...60s) Link bandwidth: A=15, B=25, C=35
   Session bandwidth: AB=7.5, AX=7.5, XB=10.4
   Sessions AB and AX fairly share Link-A bottleneck (50%/50%). Link-B permits Session XB to run up to its maximum bitrate. Link-C is not a bottleneck.

2. (t=60...160s) Link bandwidth: A=15, B=5, C=35
   Session bandwidth: AB=2.4, AX=10.4, XB=2.4
   Link-B bandwidth is cut down to 5Mbps. In this time frame Sessions AB and XB fairly shared the bandwidth. Since Session AB runs at 2.4 Mbps, Session AX can raise its bitrate from 7.5 to 10.4 Mbps.

3. (t=160...215s) Link bandwidth: A=20, B=20, C=20
   Session bandwidth: AB=6.4, AX=6.4, XB=6.4
   Bandwidths are levelled to 20Mbps. All sessions are carried through Link-C: they fairly share the bottleneck and achieve the same bitrate.

4. (t=215...285s) Link bandwidth: A=15, B=25, C=35
   Session bandwidth: AB=7.5, AX=7.5, XB=10.4
   Network is reconfigured as part 1. Bitrates are as expected.

5. (t=285...335s) Link bandwidth: A=15, B=25, C=35
   Session bandwidth: AB=10.4, XB=10.4
   Session AX terminates at t=285s. Bandwidth of Link-A, the bottleneck, is now enough to run Session AB at its maximum bitrate.

6. (t=335...370s) Link bandwidth: A=15, B=25, C=35
   Session bandwidth: AB=10.4
   Session XB terminates at t=335s. The network does not have any bottleneck, so Session AB keeps on running at 10.4 Mbps until t=350s.

Similar results are obtained even if TCP-XM sessions do not originate from the same machine. The throughput has negative spikes since random loss probability is not zero. As we can clearly observe at t=110s and t=130s, when a loss affects the performances of one session (Session AB in our example), other competing sessions (Session XB) immediately saturate the bandwidth left unused by the first.

TCP-XM fairly and optimally shares the available bandwidth with other TCP-XM sessions.

To evaluate how TCP-XM performs in the presence of TCP, we have run a 64 MB file transfer from PC-C to PC-A and PC-B using TCP-XM. During the experiment, we used `iperf` to generate TCP traffic from PC-X to PC-A. The network configuration is static: 8 Mbps, 5ms delay and no packet loss on each queue. Figure 11 illustrates the result of the experiment.

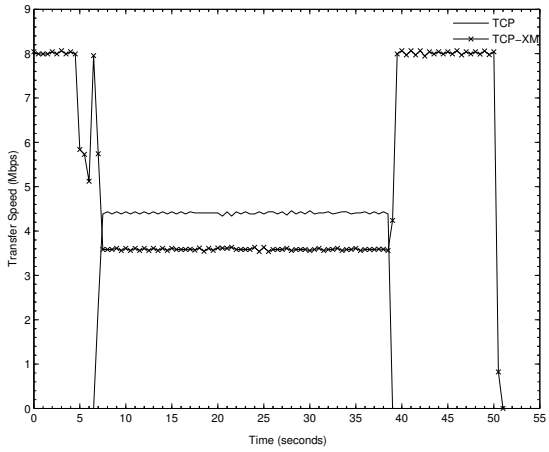

Figure 11: TCP-XM vs TCP flows

Given that TCP-XM is derived from TCP, we would expect this experiment to demonstrate TCP-XM's inherent ability to be TCP friendly. As can be seen, the link capacity is shared with both TCP and TCP-XM taking approximately 50% of the available bandwidth.

4.2 Other Protocols

MDP/NRL & NORM/NRL are two recent reliable multicast protocols with readily available implementations. We discuss these protocols in more detail in Section 5.

We have carried out a number of WAN experiments in order to evaluate how TCP-XM performs compared to implementations of MDP, NORM and multiple TCP streams.

As before, files were reliably transmitted by each protocol to a set of remote machines. The sending host was in BT Research, with receivers in Cambridge, Cardiff, BT Research, Imperial College, Manchester, Southampton (x2) and Newcastle.

Figure 12 shows the transfer speed achieved by each protocol. The performance of TCP drops quickly but predictably, due to the multiple redundant transmissions: a transfer speed of 71Mbps, for a group size of 1 (unicast), and 11Mbps, for a group size of 8. In contrast, the weak performance of implementations of reliable multicast is not predictable, and points to inefficiencies in their error control and congestion control mechanisms. In general, there is no clear winner with best performance, alternating among protocols according to group size. The only highlight is the difference between MDP and NORM, consistently in favour of the former for group size, which is in line with the fact that the congestion control algorithm of MDP is more aggressive than that of NORM.

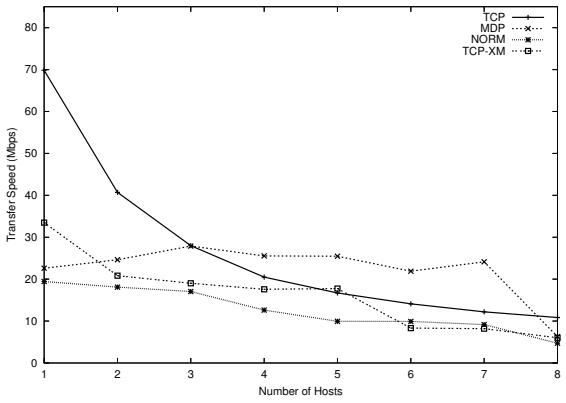

Figure 12: Protocol Performance

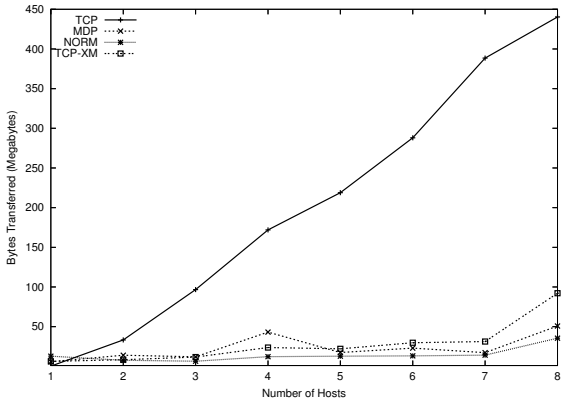

Figure 13: Protocol Efficiency

Figure 13 shows the efficiency of each protocol. The overhead of TCP grows rapidly, and linearly with group size, due to the multiple redundant streams. In comparison, the overhead of the other protocols is small. Recall that TCP-XM is capable of using multicast and unicast at the same time; a change in the transmitter, from multicast to unicast, in the eighth receiver explains why the cost is a little higher for a group size of 8.

The implementations of all three reliable multicast protocols are clearly at an early stage, hence their inability to compete effectively with TCP on performance. As stated earlier, we believe that there are some clear areas where we can improve the performance of our TCP-XM implementation. With this in mind, we are encouraged that our modified TCP approach performs comparably with current NACK-based protocol implementations.

4.3 Deployment Feedback

When deploying multicast-capable code in the wide area, there can be few assumptions about the state of multicast in the network. It is not just a case of checking whether or not multicast is available. Any number of network devices may be in the end-to-end path, and all of these must be properly configured.

In our experiments, we have found that very often multicast connectivity appears to be temporarily non-existent, only to find that some time later connectivity suddenly appears. Simple misconfiguration of devices and bugs in router code are the prime suspects, especially in their handling of multicast routing state timers. One clearly repeatable example of this kind is the frequent long latencies experienced on initially joining a group.

The extent to which these join latencies existed was tested on the WAN by continuously sourcing multicast data to random new group addresses and asking WAN hosts to join these groups. The time taken to receive data using multicast was recorded.

Table 3 shows the results of one such experiment. In this case, while the majority of hosts received multicast data almost immediately, some hosts took consistently longer to receive data. Again note that this says nothing about the hosts themselves (which would have issued IGMP join requests instantly) but rather the configuration of upstream network elements.

Table 3: Join latency (seconds)

Site	Median	Mean	Min	Max
Oxford	40	25	0	57
Imperial	59	31	4	59
Cardiff	49	31	2	60
Newcastle	1	0	0	1
Southampton	1	0	0	1
Manchester	0	0	0	1
Daresbury	0	0	0	1
All sites	1	1	1	1

Problems like these within a multicast-enabled network are notoriously difficult to troubleshoot and diagnose. From an end-to-end transport protocol perspective, we must adopt a black box approach. All we can do is take into consideration the fact that join latencies will vary and multicast connectivity may be prone to dropouts if configuration errors in the network allow state to be lost when timers expire.

This practical aspect of our work has fed into the development of our protocol's design. TCP-XM is smart enough to handle losses in multicast connectivity. It will fall back and

fall forward appropriately. The issue of long join latencies is a good example of how our design has been influenced. Unlike other protocols, we make no assumptions about multicast on initiating a connection; we neither assume multicast is present, nor blindly transmit packets and timeout on failure. We always unicast first. Only when we know multicast packets are getting through will we switch to multicast transmission.

5. RELATED WORK

Unlike many reliable multicast protocols, TCP-XM is sender initiated. This basic difference combined with per-receiver sender-side state means that much of the previous work in receiver-driven multicast and the associated congestion control issues, while relevant to a certain extent, is concerned with solving a different problem set.

There has been some precedent for attempting reliable multicast transmission by modifying TCP, but most research has focused on building new protocols. Probably the most obvious reason for not extending TCP is the ACK implosion problem. This is enough justification for most researchers to dismiss the idea, however, the significance of the ACK implosion problem depends entirely on the scale intended for the protocol. If a small or modest number of receivers are intended, ACK implosion is far less of an issue.

While few of the following have real implementations, all have combined (or have proposed to combine) multicast and TCP-like functionality in some way:

- UCL Extension [5] - UCL's Simple TCP Extension was a proposal to extend TCP to allow a connection from one application to many. SYN packets and data packets are sent to a group address. A number of mechanisms for dealing with internal data structures, API and dynamic window adjustment are suggested.

- Single Connection Emulation [15] - SCE is a sublayer between the unicast transport layer and the multicast network layer i.e. between TCP and IP. This new layer allows an application to open a reliable connection using TCP to a group address. The SCE intercepts this connection and provides the mechanism necessary to send and receive data over the underlying multicast IP layer while fooling the upper TCP layer into believing that a single connection exists.

- TCP-SMO [12] - The Single-source Multicast Optimization protocol extends TCP to allow senders and receivers to include multicast transmission when communicating. A single-source server can maintain a common multicast channel that is associated with n unicast channels. The work addresses problems such as connection management, ACK processing send window advancement, RTT estimation and packet retransmission, and congestion and flow control.

- M/TCP [16] - Multicast-extension to TCP is an extension to TCP that introduces multicast capability, however not conventional IP multicast, but small group multicast schemes such as SIM or Xcast where the sender attaches a list of receiver addresses onto the packet header. M/TCP is sender-initiated, requires no changes at the receiver, and provides multiple rates by classifying receivers into multiple groups.

- PRMP [2] - Polling-based Reliable Multicast Protocol is a TCP-like (but not based on actual TCP code) protocol that implements reliable multicast. It uses a polling system to avoid ACK implosion, and has its own form of session, error, flow and congestion control.

Other than TCP-SMO, none of the above have provided an implementation that combines the actual TCP protocol with native multicast as deployed in the network today. Functional reliable multicast code is thin on the ground.

TCP-SMO differs from TCP-XM in a number of ways. Unlike our sender-initiated "push" model, TCP-SMO adopts a server-side subscription model. The client makes a TCP connection to the server and then joins a specific multicast group. TCP-SMO matches the subsequent incoming multicast data with the existing TCP connection. TCP-SMO has been implemented as a modification to the Linux kernel. This has led to strong performance results, but at the cost of limited deployment potential. This contrasts with our userspace approach which is instantly deployable. Finally, TCP-XM's ability to switch between TCP unicast and UDP multicast by falling back and forward is unique among reliable multicast protocols.

With regard to non-TCP protocols, there is a family of protocols called NACK only, or NORM, which attempt to reduce the amount of feedback packets through NACK-based mechanisms and timer-based packet suppression. A NORM protocol [1], seeks to offer end-to-end reliable transfers of great amount of data from a sender to multiple receivers. Its instantiation combines NACK and FEC-based loss detection and recovery mechanisms, and either TFMCC [17] or PGMCC [14] for its congestion control.

MDP/NRL and NORM/NRL [11] are two protocols that follow this NORM approach (NORM is derived from MDP). Both use a selective NACK mechanism to obtain reliability, with potential help of FEC. MDP/NRL has a congestion control mechanism, but it is not friendly to TCP. The NORM/NRL congestion control mechanism, in contrast, is friendly; it dynamically selects the worst receiver, called CLR, and makes it send feedback information (including ACKs) to the sender. The feedback of the slowest receiver is used to feed an algorithm that is similar to those used by TCP. NORM/INRIA [11], part of the MCL library, is another implementation of a NORM protocol, but is not mature enough for deployment.

The performance of the NRL implementations of MDP and NORM was compared with our TCP-XM implementation in Section 4.2.

5.1 The Overlay Argument

An alternative to any transport or network layer protocol changes is to build an overlay or Application Level Multicast (ALM) network [4]. We believe this is side-stepping the issue. While alternatives to native network multicast are available, people will inevitably make use of these systems. This does nothing to improve the case for native deployment, as sceptics will argue that with application layer systems, there is no real requirement for network deployment.

By modifying the transfer protocol running on hosts, no changes are required in the network. Certainly, multicast should be enabled, but nothing *new* is required from the network. If applications make use of the new features of the protocol, they get a benefit, and users in turn have an incentive for more native multicast.

While the ALM approach does not demand changes to the network, a practical disadvantage is that to achieve efficiency approaching that of native network multicast, in certain topologies, the physical installation of new nodes at key points in the network will often be required. The logistics and financial expenditure required to procure co-location facilities for the installation of such nodes is a difficulty that is typically overlooked by ALM advocates. ALMs will not operate close to their stated efficiency without nodes in these key locations. We believe that this deployment cost outweighs that of enabling native IP multicast in the network.

In terms of driving multicast deployment, installing intermediate nodes is essentially the same as proposing to upgrade routers: it's an extra barrier. Certainly, in a Grid environment, this could be just another dæmon in a middleware release, but that confines such an approach to a restricted and controlled environment.

When building ALM networks, we would encourage the use of native network multicast. Using an approach such as ours can help achieve this, and can therefore be a useful building block in creating a "native multicast friendly" ALM. By using our protocol, such an ALM would provide upper-layer multicast functionality to its clients, while taking advantage of any deployed native network multicast.

6. CONCLUSION

We have described our design and implementation of TCP-XM, a modified TCP that supports multicast. It differs from other reliable multicast protocols in its novel hybrid combination of unicast and multicast transmission.

The protocol goals have been clearly stated, and the test environment extensive and realistic. Our practical approach has led to an implementation that has been evaluated over both local and wide area networks. These experiments have demonstrated its efficient use of network resources when compared to TCP.

We believe that our combined approach of theoretical design and practical network implementation are complementary, and create a valuable feedback loop. By continuing in this manner, we hope to further the state of reliable multicast research while delivering practical software to end-users.

7. REFERENCES

[1] B. Adamson, C. Bormann, M. Handley, and J. Macker. NACK-Oriented Reliable Multicast (NORM) Building Blocks. RFC 3940, IETF, Nov. 2004. ftp://ftp.rfc-editor.org/in-notes/rfc3941.txt.

[2] M. P. Barcellos, A. Detsch, G. B. Bedin, and H. H. Muhammad. Efficient TCP-like Multicast Support for Group Communication Systems. In *Proceedings of the IX Brazilian Symposium on Fault-Tolerant Computing*, pages 192–206, Mar. 2001.

[3] J. Bresnahan. Globus XIO. www-unix.globus.org/developer/xio/, Dec. 2003.

[4] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, pages 1–12, Santa Clara, CA, June 2000. ACM.

[5] J. Crowcroft, Z. Wang, and I. Wakeman. A Simple TCP Extension to Achieve Reliable 1 to Many Multicast. University College London, Internal Note, Mar. 1992.

[6] A. Dunkels. Minimal TCP/IP implementation with proxy support. Technical report, Swedish Institute of Computer Science, SICS-T-2001/20-SE, Feb. 2001.

[7] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[8] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[9] L. Fusco, V. Guidetti, and J. van Bemmelen. e-Collaboration and Grid-on-Demand Computing for Earth Science at ESA. *ERCIM News*, 61, Apr. 2005.

[10] The Globus Project. *Globus Quick Start Guide*, June 2001.

[11] J. P. Macker. The Multicast Dissemination Protocol (MDP) Toolkit. In *IEEE MILCOM*, volume 1, pages 626–630, 1999.

[12] S. Liang and D. Cheriton. TCP-SMO: Extending TCP to Support Medium-Scale Multicast Applications. In *Proceedings of IEEE INFOCOM*, 2002.

[13] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[14] L. Rizzo. pgmcc: a TCP-friendly single-rate multicast. In *SIGCOMM*, pages 17–28, 2000.

[15] R. Talpade and M. H. Ammar. Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service. In *Proceedings of 15th IEEE Intl Conf on Distributed Computing Systems*, Vancouver, June 1995.

[16] V. Visoottiviseth, T. Mogami, N. Demizu, Y. Kadobayashi, and S. Yamaguchi. M/TCP: The Multicast-extension to Transmission Control Protocol. In *Proceedings of ICACT2001*, Muju, Korea, Feb. 2001.

[17] J. Widmer and M. Handley. TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification. INTERNET DRAFT: draft-ietf-rmt-bb-tfmcc-04.txt, Oct. 2004.