TCP-XM: Unicast-enabled Reliable Multicast

Karl Jeacle and Jon Crowcroft
Computer Laboratory
University of Cambridge
firstname.lastname@cl.cam.ac.uk

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.

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 project, we are investigating how partial or incomplete multicast can be exploited alongside reliable unicast to improve both speed and efficiency of data transfers while maintaining reliability.

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. We describe the work to date on the design and implementation, and provide experimental results from our tests across both local and wide area networks.

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. As a result, large clouds of native IP multicast enabled networks are unnecessarily burdened with duplicate unicast data.

Depending on the state of unicast and multicast connectivity in the network, 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.

One way of implementing this combined unicast / multicast approach is to embed multicast capability within an existing unicast transport protocol such as TCP. The consequence of such an approach would be that programmers would see a familiar, but extended, API that supports one-to-many group communications. By using the group extensions, applications could take full advantage of any (partial) multicast coverage, without having to explicitly program for IP multicast.

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 user-space on end stations, new protocols can be developed and tested 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 user-space, changes can be made without modifying the kernel, but again, some form of privileged access is needed by the user-space 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, this privileged access requirement severely 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. User-space applications can freely send and receive UDP packets, so a small shim layer can be introduced to encapsulate the TCP engine's packets into UDP. While there are performance implications by running in userspace, the instant deployment potential of a userspace library, coupled with the scalability of multicast, mean that any such limitations are more than acceptable.

The key advantage of this approach is that any application can make use of this new protocol by simply linking against the supplied library. No changes are required in the network (other than enabling IP multicast).

Because the protocol is not tightly coupled to the application, should it become widely adopted, a native implementation can be built in the kernel to boost performance.

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 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 tested on a large scale. 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 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. It does *not* connect to a multicast group address (a Class D IP address in the range 224.0.0.0 to 239.255.255.255).
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. Native IP network multicast transmission capability is used
9. Unicasting ceases on successful multicasting.
10. Multicasting continues for the duration of the session.
11. All retransmissions are unicast.
12. Automatic fallback to unicast transmission takes place after n failed multicasts packets.
13. Unicast & multicast sequence numbers stay synchronised – $min(cwnd)$ & $min(snd\_wnd)$ used.
14. Connections are closed as per TCP.

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.

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

```
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 discussed below.

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 in a chain of PCBs associated with a single TCP-XM connection.

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 TCP-XM. 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.

The PCB `txmode` variable's initial value is dependent on the SYNACK. Its subsequent value can be one of:

TX_MULTICAST - the SYNACK contained the group option, multicasting will be attempted alongside unicast on this connection.

TX_UNICAST_GROUP - the SYNACK contained the group option, but multicast transmission failures have caused a temporary fallback to unicast.

TX_UNICAST_GROUP_ONLY - the SYNACK did not contain the group option. The destination cannot receive multicast packets.

TX_UNICAST - the sender has explicitly requested unicast operation.

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.

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

2.2 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.

The receiver inspects each segment that arrives and records the sequence number and whether the segment was received via unicast or multicast. It then recomputes the percentage of recent packets received via multicast.

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.

2.3 Avoiding Duplicate Acknowledgements

Because the same segments are being sent via unicast and multicast, it is possible for a receiver to receive the same segment twice. To avoid 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. If the original segment was also received via unicast, then this must be a retransmission, so a duplicate acknowledgement is required, but if the original was received via multicast, then this is likely to be a case of network ordering of packets leading to a multicast segment arriving before a unicast segment.

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

2.4 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 very 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, 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. After three consecutive unicast retransmissions have been necessary, the PCB's transmission mode is changed to TX_UNICAST_GROUP.

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.

We have chosen to implement TCP-XM in userspace over UDP rather than implement in the kernel. This facilitates our practical goals of widespread test and deployment.

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 incoming and outgoing packets; a second thread looks after the operation of the TCP-XM protocol; the third thread is the application mainline itself.

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.

4 Performance Evaluation

We have carried out tests on both local and wide area networks. For local testing, a selection of departmental workstations were used. For wide area testing, the JANET academic network was used.

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.

Round-trip times vary in range from approximately 5 to 21 milliseconds. Transfer rates attainable on single TCP connections varied from just 1.5 Mb/s to over 50 Mb/s.

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.

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

We have also conducted experiments that show TCP-XM providing comparable performance to other modern reliable multicast protocol implementations. Despite the availability of such multicast protocols, almost all data transfers use TCP. The purpose of the experiments described here, therefore, is to assess how TCP-XM performs against the most widely used data transfer mechanism.

Figure 1 shows a comparison of the TCP and TCP-XM data transfer rate to n hosts on a local departmental LAN. As would be expected, TCP's throughput declines as the host count increases. TCP-XM peaks at a much lower rate, but then consistently maintains this rate despite the introduction of more destination hosts.

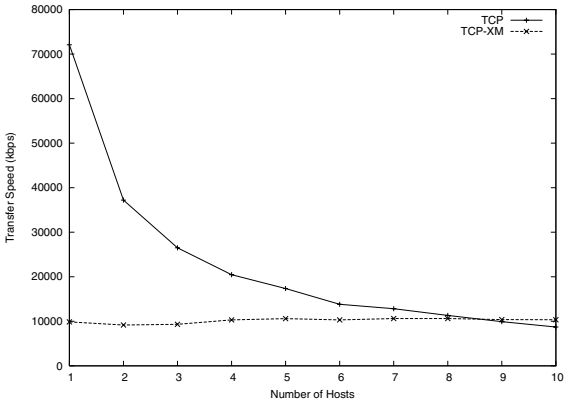

Figure 1. LAN Speed

Figure 2 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 3 shows how TCP and TCP-XM compare when a

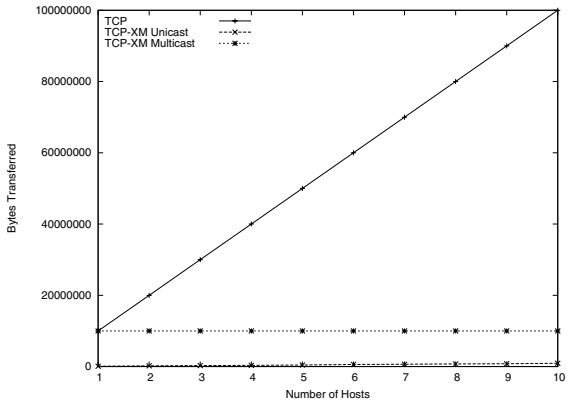

Figure 2. LAN Efficiency

transfer to n hosts takes place using a wide area network. As in the local area, TCP outperforms TCP-XM in raw speed, 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. TCP-XM finds its optimum transfer rate quickly and again manages to maintain this rate across the WAN while making use of both unicast and multicast simultaneously.

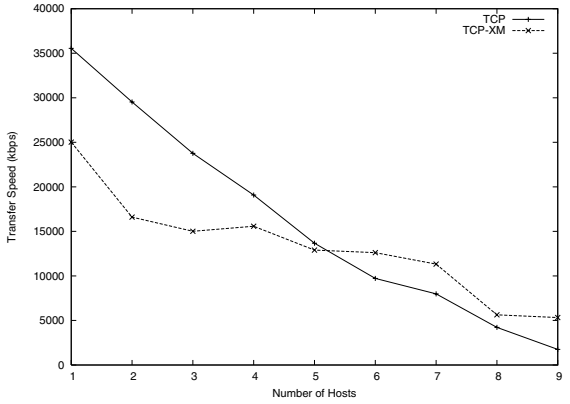

Figure 3. WAN Speed

Figure 4 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 test 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. With no multicast capability in the network, TCP-XM's efficiency level would decrease to that of TCP.

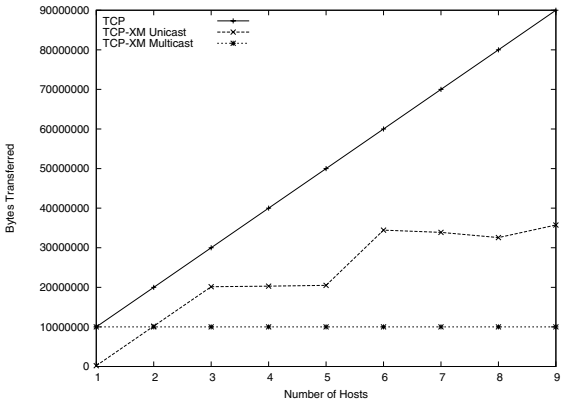

Figure 4. WAN Efficiency

We should mention that lwIP 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. Indeed, the performance of kernel TCP-XM should be almost identical to that of kernel TCP.

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, this feature will outweigh native TCP's data transfer rate so much that even a limited userspace implementation can be more desirable than kernel-based TCP.

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. The problem of TCP friendliness, for example, is irrelevant for TCP-XM; it is, by definition, TCP friendly.

There has been much work on making multicast reliable [7, 13, 14, 16, 11, 10], and it can be categorised in many different ways: sequenced or unsequenced delivery; fully or mostly reliable delivery; levels of receiver synchronisation; sender or receiver driven; and whether changes or ad-

ditional intermediary network elements are required. Most common across this work has been the focus on scaling to large numbers of receivers.

There has been some precedent for attempting reliable multicast by modifying TCP, but most research has focused on building new protocols. These include UCL's Simple TCP Extension [4], Single Connection Emulation [17], TCP-SMO [12], M/TCP [18], and PRMP [1].

All of the above combine (or have proposed to combine) multicast and TCP-like functionality in some way, but few have real implementations. And other than TCP-SMO, none 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.

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 combination of unicast and multicast transmission. Our practical approach has led to an implementation that has been tested over both local and wide area networks. These tests have demonstrated its efficient use of network resources when compared to TCP. They also indicate that there is potential to close the performance gap with TCP should the implementation be optimised.

We believe that our combined approach of theoretical protocol 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.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment Issues for IP Multicast Service and Architecture. *IEEE Network*, 14(1):78–88, 2000.

[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] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, Dec. 1997.

[8] 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.

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

[10] J. Gemmell, T. Montgomery, T. Speakman, N. Bhaskar, and J. Crowcroft. The PGM Reliable Multicast Protocol. *IEEE Network special issue - Multicasting: An Enabling Technology*, Jan. 2003.

[11] B. N. Levine and J. Garcia-Luna-Aceves. A Comparison of Reliable Multicast Protocols. *ACM Multimedia Systems Journal*, 6(5):334–348, Aug. 1998.

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

[13] J. C. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol. In *INFOCOM*, pages 1414–1424, San Francisco, CA, Mar. 1996.

[14] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven Layered Multicast. In *ACM SIGCOMM*, volume 26,4, pages 117–130, New York, Aug. 1996. ACM Press.

[15] P. Rajvaidya and K. Almeroth. Analysis of Routing Characteristics in the Multicast Infrastructure. In *INFOCOM*, San Francisco, CA, Apr. 2003.

[16] L. Rizzo and L. Vicisano. A Reliable Multicast data Distribution Protocol based on software FEC techniques. In *The Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems (HPCS'97)*, Sani Beach, Chalkidiki, Greece, June 1997.

[17] 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.

[18] 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.