A PROTOCOL FOR STREAMING LARGE MESSAGES WITH UDP

C. Briegel, Rich Neswold, Mike Slicznia
FNAL†, Batavia, IL 60510, U.S.A.

Abstract

We have developed a protocol concatenating UDP datagrams to stream large messages. The datagrams can be sized to the maximum of the receiver. The protocol provides acknowledged reception based on a sliding window concept. The implementation provides for up to 10 MByte messages and guarrantees complete delivery or a corresponding error. The protocol is implemented as a standalone messaging between two sockets and also within the context of Fermilab's ACNet protocol. The result of this implementation in vxWorks is analysed.

INTRODUCTION

The Fermilab control system [1, 2] required an alternative to providing large messages. Currently, the user must piece fragments together to support large messages. An atomic message needed to be transmitted guaranteeing delivery and provide the user with an atomic operation.

A working group was formed and discussed several alternatives. The resolution was to support existing protocols but with an extension protocol to concatenate messages with UDP datagrams. The user would not have to modify any code and realize a significantly larger message size. In comparison with other strategies, the implementation provided a solution with little impact to the control system, transparent to the user, minimized risk, and could be accomplished with minimal effort.

First, the implementation was accomplished as a standalone protocol. A simple interface was established to send large messages from one node to another with call-backs when the transmitter or receiver was complete. This minimal test solution would provide a base line for performance as well as a proof-of-principle for the protocol

Also, the protocol was used to increase the message size for ACNet [3], the Fermilab control system's messaging protocol. The added protocol for ACNet was implemented such that all existing protocols could be transparently supported by large messages. This included requests, replies to requests, and unsolicited messages.

PROTOCOL

The protocol in Figure 1 provides a mechanism to concatenate datagrams together. By acknowledging a sliding window of frames received, the message can be delivered reliably to the destination. The protocol abides by the following rules and recommendations:

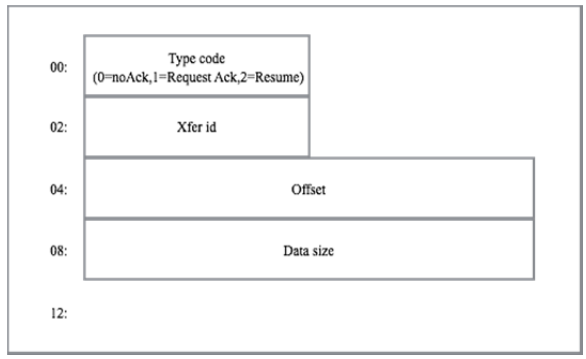

Figure 1: Large Message Protocol.

Rules

- If the offset is zero, then a new reply is arriving. The receiver can use the data size field to pre-allocate a buffer to hold the rest of the incoming data. After saving the data in the buffer, it sets the next expected offset to be equal to the size of data that was just received.
- If the offset is non-zero, it checks to see if the offset and transfer ID matches a reply that is in progress. If a match is found, the data is appended to the buffer and the next expected offset is updated.
- After appending the data, if the packet also asked for a response (type code 1 in the long message header), the task will send a resume message (Figure 2) with the current expected offset.
- If the offset is non-zero and a reply to a transfer ID is in progress but the offset is too high (a packet was dropped), the task waits for a packet that also wants a reply. When it arrives, a resume message is sent to the sender with the offset of the missing data.
- When the transfer is complete, the last packet will also require a response. The receiver returns the expected offset (which at this point will be the size of the data) or a previous offset, if a packet was dropped.

Recommendations

- The first segment **should** use type code 1, asking the receiver for a resume message. By doing this, part of the payload gets sent in addition to checking whether the receiver supports large messages (a timeout indicates no support.)
- The last packet of the message **should** use type code 1 to make sure the entire message was received.
- The sender **may** vary the interval between ACK requests to adapt to network conditions. For instance, the sender might begin the transfer with an interval of 4 packets before asking for an ACK. If there isn't an error, then 8 packets can be sent before the next

†Operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

ISBN 978-3-95450-148-9

ACK. If an error occurred, the sender reduces the interval of ACKs.

An alternative algorithm to the sliding window is also permitted. A "bit-map" algorithm allows the receiver to build its copy of the large message with the segments it receives and keeps track of the holes. When a packet arrives with the ACK request, the receiver can go back and specify the earliest hole. With each filler sent, the sender always asks for a reply. The receiver replies with each hole's offset until they are all filled [4].

STAND-ALONE IMPLEMENTATION

The protocol was implemented between two MVME5500s utilizing UDP socket communications. Both nodes ran vxWorks 6.4 operating systems. Repeated messages were sent for 10 Secs with selected large message sizes between 1,000 to 32,000,000 bytes. If a single datagram was needed for the large message, the protocol was still used to guarrentee messages arrived with acknowledgment. The maximum datagram sizes tested were 0x2100, 0x8100, oand 0xF100 bytes.

To establish a base-line, local messages were sent as well as messages over the a 1 Gbit Ethernet. This would provide a comparison of packet latency and associated problems with the network. The result of this local test is in Figure 2 for the various datagram sizes acknowledging every datagram. The maximum long message size displayed on the graph is 1,000,000 bytes since there was negligable change for larger message sizes.

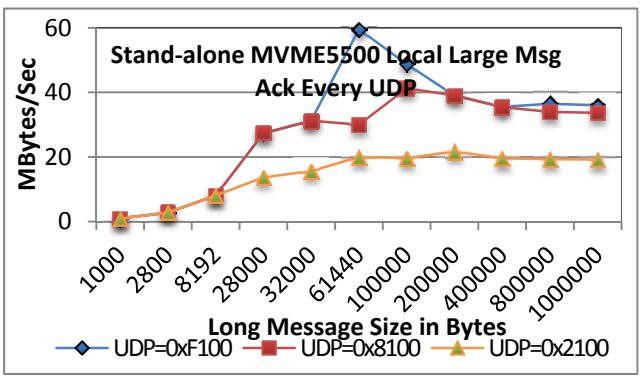

Figure 2: Stand-alone Local Msg Acking Every UDP.

Figure 3 displays the throughput sending messages of varying datagram sizes for a 10 second period. Messages was sent in only one direction over an operatonal 1 Gbit Ethernet acknowleging all datagrams.

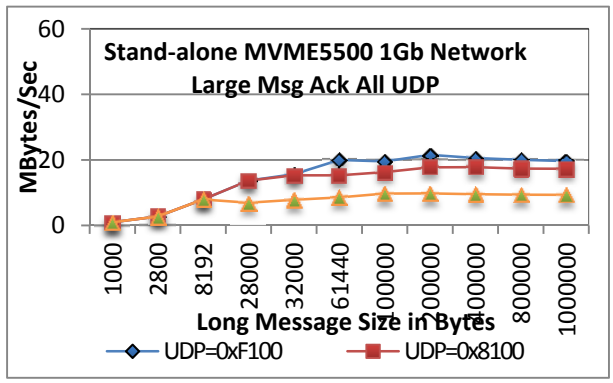

Figure 3: Stand-alone Network Msg Acking Every UDP.

Figure 4 displays the throughput from one node to another using a datagram size of 0xF100 while varing the acking rates.

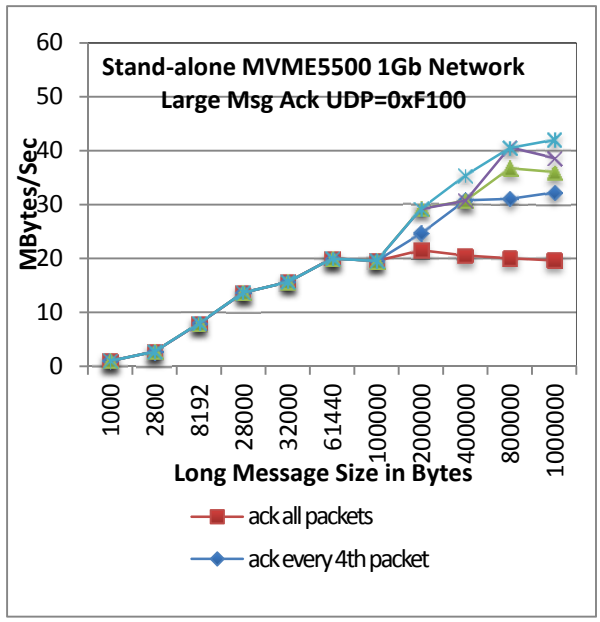

Figure 4: Stand-alone Network Msg, UDP=0xF100.

ACNET IMPLEMENTATION

ACNet is a peer-to-peer protocol which routes messages to connected tasks. ACNet has three distinct messages: unsolicited one-way messages, requests, and replies. Currently, ACNet uses UDP datagrams to implement this protocol and was limited to a single UDP datagram size for it's messaging. Historically, ACNet implemented packeting within the protocol but this was limited to a maximum of 16 packets. The goal was to provide at lease 10 mega-bytes of data in a single message.

The concept was to use a connected task called "LNGMSG" to implement the collection of UDP frames into a single message. If the task was not available on the destination node, then the node did not support the protocol. This would enable an adiabatic implementation across the complex.

ISBN 978-3-95450-148-9

Unsolicited messages were sent to the connected task "LNGMSG" followed by the large message protocol and the original ACNet header. The data portion of the message was specified in the large message protocol. The message received by "LNGMSG" would concatenate the data for the original message. After the protocol acknowledged receiving the entire message, the message would be delivered to the original task.

As before, an equivalent test was accomplished between two MVME5500 processors. The primary difference was the messages were sent as requests followed by a single reply echoing the original message. Comparing it with the original request validated the reply message.

Figure 5 is an ACNet local message sent as a request and single reply to itself. Since these messages result in memory moves acknowledging all datagrams was efficient.

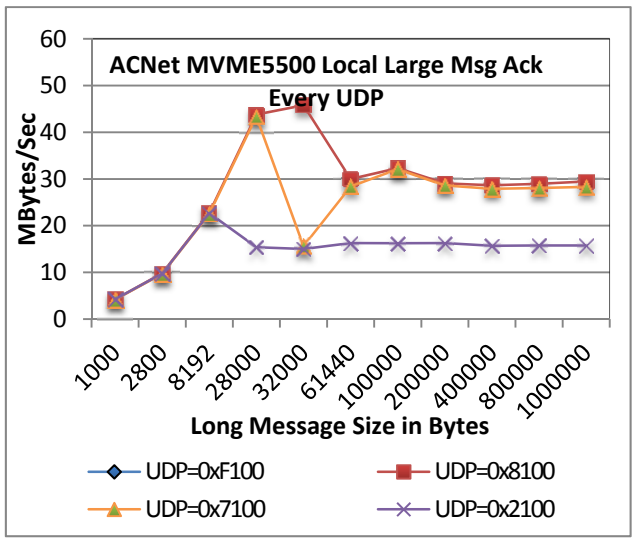

Figure 5: ACNet Local Msg Acking Every UDP.

Figure 6 displays the request/replies over the Ethernet between two nodes with various the datagram sizes acknowledging all datagrams.

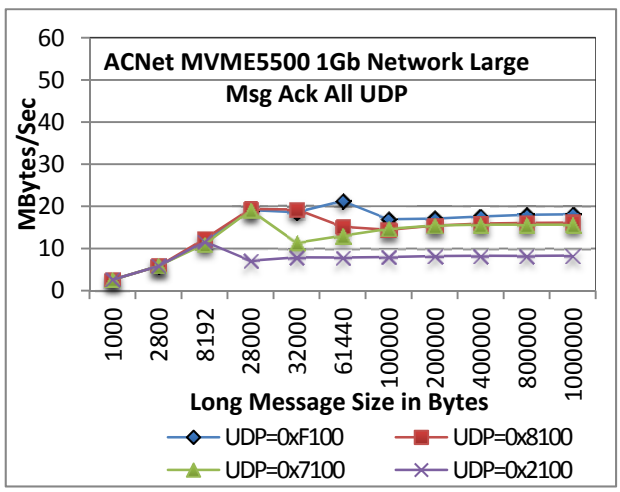

Figure 6: ACNet Network Msg Acking Every UDP.

Figure 7 displays the various acknowledgment intervals for a UDP size of 0xF100 bytes.

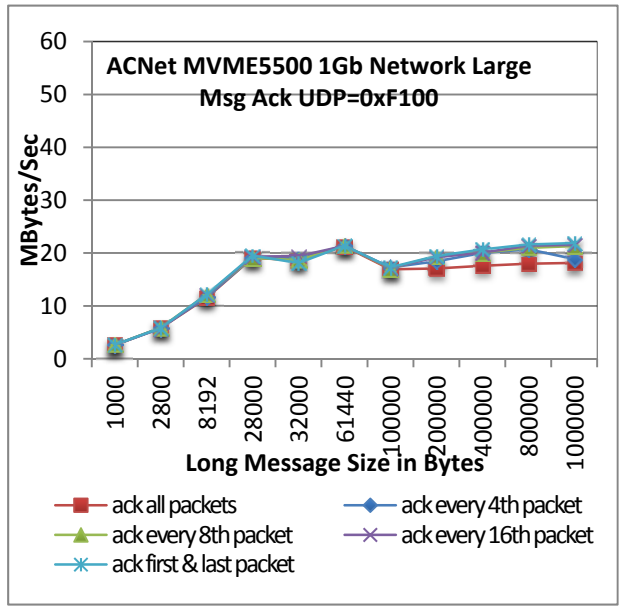

Figure 7: Stand-alone Network Msg, UDP=0xF100.

Figure 8 displays the various acknowledgment intervals for UDP size of 0x7100 bytes which appears to be optimal for the platforms tested.

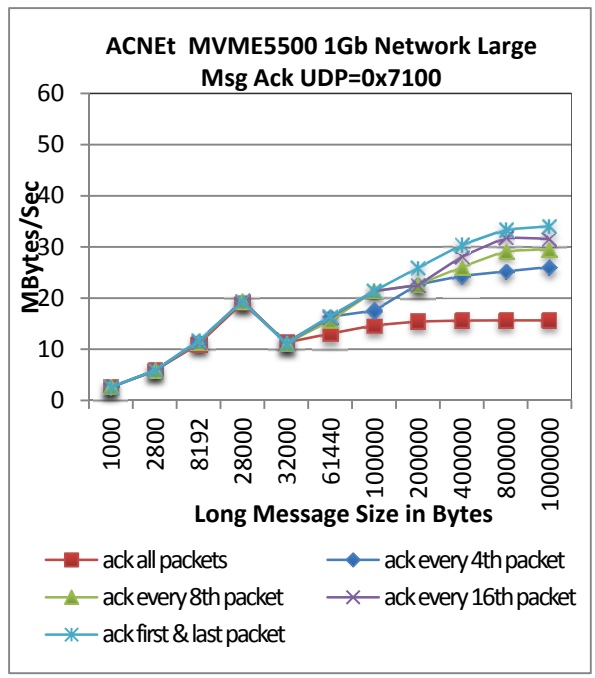

Figure 8: Stand-alone Network Msg, UDP=0xF100.

ISBN 978-3-95450-148-9

PERFORMANCE ANALYSIS

The protocol without ACNet managed to provide minimal processing to combine the frames into a large message. Thus, the sender did not over-drive the receiver. Mismatched nodes or networks will eventually cause buffers to be consumed and packets dropped within the socket software implementation. In vxWorks, the error is observed in the full socket counter returned from a call to udpstatShow.

The ACNet protocol produced the above error more readily since the processing of the ACNet received message requires more processing than the sending of frames. The acknowledgment process provides flow control since the next portion will not be sent until the resume type code is received. Only when the acknowledgment rate decreased did the receiver run out of buffers. For these tests, the transmitter was tuned with small delays for non-acknowledged frames. While the protocol retransmits dropped packets and the large message will arrive, the throughput suffered drastically.

Delay tuning is not an appropriate solution. Providing flow control by acknowlegment is not optimal for throughput. The transmitter of the protocol needs to be adaptive and adjust the acknowledgement rate as needed.

The performance graphs have consistently shown reasonable throughput. The difference between the standalone protocol and ACNet is as expected. The ACNet overhead of concatenating the datagrams is slightly greater to process the ACNet protocols. The resulting imposed transmitter delays in ACNet became the most significant factor in performance differences between these two solutions.

CONCLUSION

The ACNet throughput appears to be adequate for our immediate needs. The protocol as specified is sufficient for guaranteed delivery of large frames via UDP datagrams. Further tests with all nodes in the control system will give a better understanding of system performance.

The implementation on vxWorks required about one person-month of effort. The implementation was transparent to the user. The infrastructure only needed to be rebuilt with changes to the include files. The implementation on Linux is in progress to provide full functionality for the control system.

REFERENCES

[1] J. Patrick, “ACNET Control System Overview,” Fermilab Beams-doc-1762-v1.

[2] K. Cahill, L. Carmichael, D. Finstrom, B. Hendricks, S. Lackey, R. Neswold, J. Patrick, A. Petrov, C. Schumann, J. Smedinghoff, “Fermilab Control System,” Fermilab Beams-doc-3260-v3, http://beamdocs.fnal.gov/AD-public/DocDB/ShowDocument?docid=326

[3] C. Briegel, G. Johnson and L. Winterowd 1990 The Fermilab ACNET upgrade, *Nucl. Instrum. Meth. A* 293 p.235-238.

[4] C. Briegel, C. King, R. Neswold, K. Nicklaus, and M. Sliczniak, ”Large ACNET Mesages, ” 2015.

ISBN 978-3-95450-148-9