EXTENDING ACNET COMMUNICATION TYPES TO INCLUDE MULTICAST SEMANTICS

R. Neswold, C. King
FNAL†, Batavia, IL 60510, U.S.A.

Abstract

In Fermilab's accelerator control system, multicast communication wasn't properly incorporated into ACNET's transport layer, nor in its programming API. We present some recent work that makes multicasts naturally fit in the ACNET network environment. We also show how these additions provide high-availability for ACNET services.

INTRODUCTION

When multicasts were first used in Fermilab's ACNET control system, they were handled differently than normal ACNET communication. Rather than using the familiar ACNET API, programmers had to use specialty libraries. This meant that applications would communicate via ACNET for normal data acquisition needs, but would additionally manage multicast resources (even if hidden behind a library interface) for the few protocols requiring multicast reception.

As more services required multicast communications, more infrastructure was added – some of it resembling the connection management already done by the ACNET process. It became apparent that, with a few tweaks, multicasts could be cleanly incorporated into ACNET and, with it, some very interesting behaviour can be implemented.

MIGRATING TO MULTICASTS

For a system to become an ACNET node, it needs to be added to the ACNET node table. The node table maps an ACNET node name to its IP address. Each ACNET node carries a copy of the node table which it references when sending a message to another node.

Since the node table contains IP addresses, we can easily allow multicast transmissions by creating a ACNET node name and associating it with an IP multicast address. Applications using the ACNET communication libraries would then be able to send multicasts by using an ACNET node name, rather than using a separate set of functions tailored for a protocol.

Receiving Multicasts

Transmission was simple, but unfortunately we don't get reception of multicasts for free. Somehow ACNET needs to know when to join and drop the group address. One solution would be to join all the multicast addresses in the node table. As changes are made to the node table, groups can be joined and dropped, as appropriate. This approach seemed excessive, though, since most multicast traffic would then be received by the ACNET process, but thrown away if there were no clients interested in the packet. We chose to implement a solution that is more efficient with resources.

Historically, applications have always been able to connect to ACNET using more than one handle. We use this ability so that a client may connect with a unicast handle and one or more multicast handles. But what makes a handle able to receive multicasts? When the client connects to ACNET, it specifies a name to be its connection handle. The ACNET process looks in the node table to see if a node with the same name exists and if it is associated with a multicast address. If so, the ACNET process joins the multicast group and the client will begin receiving multicast messages.

As an example, Fermilab has a protocol called DbNews in which changes to our device database are broadcast to all interested clients. A multicast address is assigned to this service and is associated with the ACNET node name DBNEWS, through the node table. The task monitoring database changes sends its report to the DBNEWS “node”. A client wishing to receive these reports would create an ACNET connection using a handle of DBNEWS, which would make the ACNET process join the DBNEWS group. The client will receive DbNews reports only through this connection. When the client terminates, its connection gets closed and ACNET drops membership in the DBNEWS group.

Extending Requests with Single Reply

In addition to datagrams, ACNET supports two request/reply transports; one version expects a single reply and the other expects a series of replies. Extending the datagram transport was easy. But we wondered if we could cleanly extend our request/reply transports as well. It turns out we could.

In the single reply case, an ACNET client sends a request packet to a service. Making a request sets up a connection between the two tasks, the state of which is maintained by their respective ACNET processes. The connection is maintained until one of two events occurs. The normal resolution is for the service to respond with the reply packet, at which point the connection state is removed in the ACNET processes. The other, less

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

frequent possibility is that the requesting task sends a cancel message, which causes the participating ACNET processes to clean up the request connection (the remote service is also notified of the cancellation.)

How does this work with multicasts? The request is multicast and received by all tasks connected to the corresponding multicast handle. No matter how many services listen and reply, only the first reply will be seen by the requesting client because after receiving a reply, the request connection is invalidated.

All ACNET packets (datagrams, requests, and replies) have a common header which, among other things, contains the sender's node name. In the case of replies from a multicasted request, the sender's node name would be the generic multicast node name. This is less useful than having the client know who really responded to the request, so the ACNET process modifies the header to reflect the actual node name of the responder.

Extending Requests with Multiple Replies

The other request/reply transport supported by ACNET is one where multiple replies are sent. When a client asks for multiple replies, the server uses a flag field in the header to indicate whether the packet is a reply with more expected, or if it's the last reply. When marked as the last reply, the ACNET process will clean-up the connection's resources after delivering the last reply to the client.[1]

To support this mode of communication, more extensive changes were required. ACNET still modifies the header to indicate the node sending the reply. In addition, ACNET doesn't allow any of the repliers to close the connection by sending a “last reply” status. All replies will appear, to the client, to have more following. The only permitted way to end the request is for the client to multicast a cancel message.

USING MULTICASTED REQUESTS

Now that we have multicasted requests at our disposal, we found they have very useful applications. They give us further ways to reduce network resources as well as techniques that provide high-availability to our network services.

Reducing Resources

We have applications that poll a group of nodes. Some application monitor the node's health while others display a group of nodes' up-time. These sorts of application typically send a request to each node and expect a single reply. As long as the application runs, it cycles through its list, polling each node. They could be written to request multiple replies from each node. However, the network resources being used would grow proportionately with the number of monitored nodes, since receiving multiple replies requires a connection of its own. By multicasting a request for multiple replies, only one network connection is maintained and the application needs only to monitor the one connection – no matter how many nodes are participating in the replies.

High Availability: Service Discovery

Recent ACNET services have been developed that take advantage of multicasted requests that expect a single reply. A client interested in the service broadcasts a query request. The services receiving the query respond with an “I'm here!” notification. Only the first reply is passed on to the client. This retires the practice of setting aside certain nodes to be the “dedicated server” and of using hardcoded node addresses in software libraries. The system is much more flexible and dynamic (although admittedly, it now requires this initial discovery transaction.)

High Availability: Load Balancing

With a slight tweak to the ACNET service, load balancing can be added.

For example, a service is started on several ACNET nodes. A client needing the service broadcasts the request and only receives the first reply (service discovery). Each server is programmed to delay its reply to a discovery request. The delay should be bounded, but is scaled based upon how much work it's currently doing. Servers doing less work will respond quicker than busier servers. The client can check the reply's header to see which server responded first and, therefore, is the least busy. The rest of the service transactions can be done with this server.

If control system personnel find all servers are constantly loaded, more copies of the service can be started on other nodes to scale the resources according to the demand.

If a service is taken down (or crashes!) while a client is using it, the client can try to restore the connection, or more simply, make another service discovery request and start interacting with an alternate server.

CONCLUSION

We began this effort in order to make all network communications consistent – essentially cleaning up some code. But what we didn't anticipate, at the time, was how useful and powerful these changes would be for scaling our ACNET services. We currently have servers using the service discovery and load balancing techniques described in this document and plan on converting others.

[1] In fact, a request for a single reply is a special case where the first reply is marked as the last reply.