Replace UDP messaging for membership with TCP

To be Reviewed By: 7 April 2020

Authors: Dan Smith

Status: Draft | Discussion | **Active** | Dropped | Superseded

Superseded by: N/A

Related: N/A

Problem

Geode uses UDP messaging through JGroups for peer-to-peer messaging related to membership. Geode does not actually use jgroups group membership system, only it's reliable UDP messaging. Using UDP messaging through JGroups has a couple of issues:

- We implemented our own non-standard encryption and key exchange system on top of jgroups UDP messaging in order to ensure that all P2P messages are encrypted. See Secure UDP Communication in Geode. This adds complexity to the configuration by requiring a users to also set a separate UDP encryption property, and it also adds the risk of security and functional issues with our implementation. We recently discussed deprecating this property on the mailing list because of it's functional issues - See this thread.
- JGroups does not support rolling upgrades which makes it more difficult to upgrade JGroups.

Rather than continue with using UDP, we would like to replace the UDP messaging for membership with a TCP-based messaging system. This will allow us to use the standard encryption protocols for all peer to peer messaging, and it will remove our dependence on jgroups in the long term.

Goals

- Set us up for removing JGroups as dependency in future versions by moving to a protocol that does not require JGroups
- Support rolling upgrades from the old JGroups protocol to the new TCP-based protocol
- Use the existing SSL settings to control how membership messages are encrypted
- Be as reliable in the face of networking failures as the previous protocol

Anti-Goals

- It is not a goal to replace the existing and separate peer-to-peer TCP messaging system that is used for cache operations. The new messaging system is initially targeted only at replacing our use of jgroups UDP for membership messages.

Design

All of the messaging related to membership is handled by the JGroupsMessenger class, which implements the Messenger interface. We will create a new implementation of Messenger that uses TCP sockets, rather than JGroups and UDP sockets.

Our proposal for the new Messenger is to implement a TCP server and client using Netty. The Netty server and client will use the existing cluster SSL configuration. So if cluster SSL is enabled no additional properties will be required. See https://geode.apache.org/docs/guide/latest/managing/security/implementing_ssl.html for information on the relevant properties. The server socket will use the existing bind-address and membership-port-range properties to determine it's address and port.

The Netty based messenger will maintain one connection to each peer. When sending a message the messenger will create a connection to all destinations if no connection exists yet. Once a connection is established, the connection will remain open until the messenger is told to shut it down.

Messages received by the Netty server will be dispatched from Netty event loop threads. For this reason, it is important that message processing should not block, or it will prevent other messages from being received. The old JGroupsMessenger dispatched messages using a single jgroups receiver thread.

Backwards Compatibility and Upgrade Path

We need to be able to do a rolling upgrade from the old JGroups-based protocol to the new protocol. We will need to continue to support rolling upgrades for a certain range of versions before we can drop JGroups.

In order to accomplish this a member will actually need to be listening for connections on both protocols when it initially starts up. We will create a VersionAwareMessenger that contains both a JGroupsMessenger and a NettyMessenger. It can install handlers in both of them, and decide which Messenger to use when sending a message based on the version of the recipient. If a member receives a view that contains no old members that require t he old protocol it could shut down the JGroupsMessenger.

There is an issue here with the need to listen on two separate ports and distribute both ports to other members. Currently, we distribute the jgroups UDP server port as part of the InternalDistributedMember. InternalDistributedMembers are sent as part of view messages and find coordinator requests to the locator. That allows all members to discover the listening port of other members (see GMSJoinLeave message sequence diagrams). We need to distribute the new tcp port as well. There are a couple options we are evaluating:

1. Bind to the same port for both UDP and TCP traffic. Newer members would use TCP. This would require finding an available port with both protocols. This is the option we prefer to try first.
2. Add another port field to InternalDistributedMember

3. Pass the new membership port around outside of InternalDistributedMember. This would probably involve sending it part of the FindCoordinatorResponse, as well as including it in the GMSMembershipView.

For the last two options, we will need to version either InternalDistributedMember itself, or the FindCoordinatorResponse and GMSMembershipView classes. Old members will only receive the UDP port, newer members will receive both ports.

Handling TCP connection failures

The Messenger is used to send messages to destinations that may or may not be part of the current membership view. If a member is in the view, the Messenger needs to keep trying to deliver messages to that member as long as that member is still in the view. Because individual TCP connections can fail, this forces us to implement a reliability layer above TCP that will continue to retry messages until a member is removed from the view. This layer needs to be able to:

- Reestablish a connection to the destination to if the existing TCP connection fails.
- Retransmit messages that may not have been received.
- Prevent duplication of retransmitted messages on the receiver side

In order to accomplish these goals, we we implement a layer that attaches an increasing sequence number with each message, and save messages on the sender until their sequence numbers are acknowledged.

One concern about switching to a TCP-based protocol is that network outages may result in TCP sockets hanging on read or write operations. We need to ensure that, if a connection to one member is blocked, we still send messages to the other members. Each destination will have its own queue of messages to be sent, and adding to one queue should never block.

The Messenger is also used by membership to check for a quorum of members, after a network partition event. So we need to ensure that we retain the ability to send a ping message out to all members and check for responses within a certain time window.

Changes and Additions to Public Interfaces

Because we are eventually getting rid of udp messaging, the following udp related gemfire properties will be deprecated:

- udp-fragment-size
- udp-recv-buffer-size
- udp-send-buffer-size
- security-udp-dhalgo
- mcast-address
- mcast-ttl
- mcast-flow-control
- mcast-port
- mcast-recv-buffer-size
- mcast-send-buffer-size
- disable-tcp

Performance Impact

This proposal is only targeted at replacing the UDP messages used for membership events, which are fairly low traffic. Region operations like puts and gets go over separate TCP connections that are not going to be changed as part of this proposal. Therefore we don't anticipate any performance impact for region operations.

In the future we could consider switching all peer-to-peer messages to flow through this new TCP messaging system. We would only do that if the new system has similar or better performance than the old system.

Alternatives

Use DTLS

One option to address concerns with using custom encryption protocol would be to continue using jgroups, but remove our old security-udp-dhalgo property in favor of using DTLS. However, this approach is complicated by the fact the the JDK does not have DTLS support built in to Java 8, which would require us to backport and maintain an implementation. In addition this would not help us fix the problem of being tied to an outdated jgroups stack that cannot be upgraded.

Use the existing P2P tcp socket code

One the face of it, just using our existing TCP sockets for membership messages as well seems attractive. However this code as it is written is dependent on the membership system to bootstrap itself. For example it will not even send messages until the member has completely joined and some startup messages are exchanged. It would take some refactoring to be able to send membership messages over this system in order to bootstrap membership. The existing P2P socket code is also fairly old, and we feel a better path forward is to move towards a new system based on netty that is less entangled with the rest of the system.

FAQ

Errata