Active Connection Management

Previous Next

Active Connection Management (ACM) is enabled by default and helps to improve scalability and conserve application resources by closing idle connections.

On this page:

Configuring Active Connection Management

There are three components to an ACM configuration:

- Close
  Determines the situations in which Ice closes a connection

- Heartbeat
  Determines the situations in which Ice sends "heartbeat" messages to keep a connection alive

- Timeout
  Defines the time interval in which the Close and Heartbeat actions occur

You can configure these components using ACM properties that affect all of the connections created by a communicator or modify a single connection directly in your code:

- `Ice.ACM.Close`
  `Ice.ACM.Heartbeat`
  `Ice.ACM.Timeout`
  These properties establish the default settings for a communicator and affect both client (outgoing) and server (incoming) connections.

- `Ice.ACM.Client.Close`
  `Ice.ACM.Client.Heartbeat`
  `Ice.ACM.Client.Timeout`
  These properties override the default properties above for client (outgoing) connections.

- `Ice.ACM.Server.Close`
  `Ice.ACM.Server.Heartbeat`
  `Ice.ACM.Server.Timeout`
  These properties override the default properties above for server (incoming) connections.

- *`adapter`*`.ACM.Close`
  `adapter``.ACM.Heartbeat`
  `adapter``.ACM.Timeout`
  These properties override the `Ice.ACM.Server` properties above for connections to a particular object adapter.

- Programmatically by calling `setACM` on a connection object. These settings override all ACM property configurations for that connection.

Back to Top ^

ACM Timeout Semantics

That the Close and Heartbeat behaviors share a single Timeout setting may seem surprising at first glance. For instance, it's possible for a program to configure certain combinations of Close and Heartbeat settings such that the heartbeats prevent a connection from ever becoming eligible for closure. In general, however, one peer will configure Close and Timeout settings while the other peer will configure Heartbeat and Timeout settings, in a coordinated effort to meet the application's requirements for connection management.

For each connection configured for ACM, Ice checks its status approximately every (Timeout / 2) seconds. If heartbeats are configured for the connection, Ice also sends a heartbeat during every status check. If two peers use the same Timeout value, this strategy ensures that heartbeat messages are sent well before the connection would be considered idle by the other peer.

Back to Top ^

Creating a Connection Management Policy

The ACM settings give you a lot of flexibility for developing a connection management policy that meets the needs of your application. As application requirements can vary greatly, we provide recommendations for several common use cases in the subsections below. You should also take the following general guidelines into consideration when developing your policy:

- Connection semantics
  Is your application dependent on a connection remaining open? For example, there may be semantics attached to a connection, as in the case of a session in which a peer's identity, or some allocated resources, are valid only as long as the peer's connection remains active. Glacier2 sessions work this way. In this case you'd want to enable heartbeats and configure the timeout so that it is compatible with the session's expiration timeout.

- Network traversal issues
  Depending on your network topology, security requirements, and other application considerations, a connection may be a valuable resource that is expensive or time-consuming to construct and you may not want Ice to discard it lightly. If sending heartbeats is too costly, you may need to disable automatic closure altogether.

- Aggressiveness
  ACM's combination of heartbeats and connection closure options allows your application to detect when something has gone wrong with a peer. Furthermore, the connection closure options provide varying levels of response, from conservative to aggressive. Your application requirements will dictate the necessary behavior.

- Local networks
  When operating in a purely local network in which there are no firewalls to traverse and none of the other considerations listed above are a concern, you can usually allow Ice to open and close connections transparently. In fact, the default configuration is a reasonable starting point.

- Two sides to every connection
  It's important to consider the requirements of both clients *and* servers when designing your connection management policy. For example, if a client and server are using significantly different settings for their ACM timeouts, it can result in surprising and usually undesirable behavior.

To verify that the policy you've configured is behaving as expected, set `Ice.Trace.Network=2` and monitor your log output for connection-related activity.

Back to Top ^

Bidirectional Connections

A bidirectional connection allows a server to make callback invocations on a client, offering a simple solution to work around the limitations enforced by network firewalls in which the server would otherwise be prevented from establishing a separate connection back to the client. Given that the client's connection to the server represents the server's only path to that client, this connection must remain open as long as the client needs it.

In versions prior to Ice 3.6, our recommendation was to disable ACM completely in both client and server when using bidirectional connections to prevent unintended closure. As of Ice 3.6, there's no harm in enabling ACM connection closure as long as you also enable client-side or server-side heartbeats to prevent the connection from becoming idle during periods of inactivity. Consider the following settings:

```
# Client
Ice.ACM.Close=0      # CloseOff
Ice.ACM.Heartbeat=3  # HeartbeatAlways
Ice.ACM.Timeout=30

# Server
Ice.ACM.Close=4      # CloseOnIdleForceful
Ice.ACM.Heartbeat=0  # HeartbeatOff
Ice.ACM.Timeout=30
```

Here the client will always send heartbeats at regular intervals to keep connections alive. If a connection does become idle, which most likely would be due to a serious problem with the client, the server forcefully closes the connection regardless of whether any dispatch or invocations are pending at the time.

> You can also reverse the roles: configure the server to send the heartbeats and the client to close forcefully idle connections.

If your application manually configures bidirectional connections (as opposed to the automatic setup provided by Glacier2 connections, for instance), it's not much extra work to configure the ACM settings individually on the connection object if your requirements for bidirectional connections differ from the communicator-wide settings defined by the ACM properties. As an example, an application may not always need to use a connection for bidirectional purposes. As long as a connection is unidirectional, the application might consider it safe to be closed automatically by ACM. Once it transitions to a bidirectional connection, the program that initiated the connection should modify its ACM configuration similar to the client settings shown above.

Back to Top ^

Managing Sessions

The notion of a session is a common and very useful solution for managing resources associated with a particular client. Briefly, the idea is that a client creates a session, which usually includes an authentication process, and then allocates some resources. The server associates those resources with the client's session and requires the client to keep the session active, using an expiration timer to reclaim a session and its resources if a client abandons the session without formally terminating it. The strategy represents good defensive programming for a server; without such a solution, ill-behaved clients could continue allocating resources indefinitely.

In versions prior to Ice 3.6, we recommended that the client create a background thread that periodically "pinged" the server using an interval based on the server's session expiration time. The thread is no longer necessary as of Ice 3.6, since the ACM heartbeat functionality serves the same purpose. Note however that there are still a couple of considerations:

- The client needs to determine the server's session expiration time. An application might configure this statically, or the value may only be available at run time by calling an Ice operation. In the latter case, the client would need to transfer this value to a corresponding ACM timeout setting, most likely by calling `setACM` on the connection object.
- Using ACM heartbeats to keep a session alive is convenient for clients but presents a problem for server implementations: How does a server know that the connection is still alive? The solution is for the server to call `setCloseCallback` to get a notification of the connection's closure.

The following configuration settings should get you started:

```
# Client
Ice.ACM.Close=0      # CloseOff
Ice.ACM.Heartbeat=3  # HeartbeatAlways
Ice.ACM.Timeout=30

# Server
Ice.ACM.Close=4      # CloseOnIdleForceful
Ice.ACM.Heartbeat=0  # HeartbeatOff
Ice.ACM.Timeout=30
```

These settings assume that the application configures the ACM timeout statically. (For example, perhaps the ACM timeout also serves directly as the session expiration timeout.) The server's setting for ACM connection closure represents a design decision that assists the implementation: When Ice detects that a connection has become idle, likely due to a serious problem with the client, it forcefully closes the connection. This process involves calling the callback that the server previously set on the connection; presumably the server interprets this notification as an indication that the session can be destroyed. Without the connection callback, the server would have to implement some other strategy for periodically reaping expired sessions.

In this example, the client has the active role (it sends heartbeats) and the server has the passive role (it expects the client to send heartbeats). It's fine to also reverse these roles. You could for instance configure the client to forcefully close the connection when it becomes idle and configure the server to send heartbeats. The server will no longer receive heartbeats but it will still be reliably notified when the connection is closed if it has registered a close callback.

Back to Top ^

Detecting Peer Problems

ACM is an effective tool for detecting and recovering from catastrophic problems with a peer. Let's suppose that it's safe to enable ACM connection closure in an application for both the client and the server. Furthermore, in this application, the server can take a significant amount of time to dispatch a client invocation. The client is willing to wait as long as it takes for the invocation to complete, however, as a defensive measure, the client also wants the ability to recover in case the server becomes unresponsive. Consider these settings:

```
# Client
Ice.ACM.Close=3      # CloseOnInvocationAndIdle
Ice.ACM.Heartbeat=0  # HeartbeatOff
Ice.ACM.Timeout=30

# Server
Ice.ACM.Close=1      # CloseOnIdle
Ice.ACM.Heartbeat=1  # HeartbeatOnDispatch
Ice.ACM.Timeout=30
```

The server configuration causes it to close connections when they become idle. Since the client doesn't send heartbeats, this means the client hasn't actively used the connection during the timeout period. The server configuration also sends heartbeats to the client, but only while the server is dispatching invocations. As a result, no matter how long it takes to dispatch the invocations, the Ice run time in the server will continue sending heartbeat messages as an indicator to the client that the server is still "healthy". If the Ice run time in the client determines that a connection has become idle, it either means the connection is no longer being used, or if outgoing invocations are still pending, it means that the server has stopped sending heartbeats for some reason. In the former case, the connection is transparently closed without affecting the client. In the latter case, Ice forcefully closes the connection. Assuming the subsequent automatic retry behavior fails, all pending client invocations on that connection will terminate with an exception.

Don't use ACM timeouts as a mechanism for aborting long-running invocations. Ice provides invocation timeouts for that purpose.

Back to Top ^

Disabling ACM

An application may be interested in preventing ACM from closing connections. For example, oneway invocations can be silently discarded when a server closes a connection. One solution is to set the following property in the server:

```
Ice.ACM.Server.Close=0
```

This setting prevents the server from closing an incoming connection regardless of how long the connection has been idle.

Another solution is to enable heartbeats in the client so that the connection remains active and never becomes eligible for closure while the client process is operating normally. If the client should terminate unexpectedly, the server will still be able to close the connection within a reasonable amount of time without waiting for a low-level notification from the network layer.

Back to Top ^

See Also

- Ice.ACM.*
- Oneway Invocations
- Object Adapters
- Batched Invocations
- Using Connections
- Connection Closure
- Bidirectional Connections
- Glacier2

Previous Next