## Actor Bug:  Communication - Error Handle

Every robust system has some form of error handling and if it's not handled properly it can fail to catch or fix the problem that it's supposed to deal with. This section is specifically for when connection and message errors get mishandled.

Spary had an issue with handling a disconnection error which led to infinite attempts to reconnect. When a connection was closed unexpectedly the system didn't have a limit on how many times it would try to reconnect which could lead to dead lock as it tries to get responses that may never come. This was solved by adding a limit to the re-connection attempts. This is a subtle difference to the Connection error from Collossus seen previously as that was a manual disconnect not being registered properly compared to this error that was mishandled.

The code below shows how this issue was fixed by making use of a fixed number of times the connection attempt could be retried. Here they chose 5 for maxRetries and they added response handlers to deal with keeping track of the attempts made. If there are attempts left the failure is noted and the attempt repeated. If there are none, the error is logged and no more attempts are made.

**Link:** https://github.com/spray/spray/issues/78

**Code:**

spray/client/ConduitConfig.scala

```
+  maxRetries: Int = 5,

 object AkkaConfConduitConfig extends AkkaConfSettings("spray.client.") {
   lazy val ClientActorId = configString("spray-can-client")
   lazy val MaxConnectionsPerConduit = configInt(4)
+  lazy val MaxRetries = configInt(5)
 }
```

spray/client/HttpConduit.scala

```
-  protected case class Send(request: HttpRequest, responder: Either[Throwable, HttpResponse] => Unit)
-    extends HttpRequestContext
+  protected case class Send(request: HttpRequest, responder: Either[Throwable, HttpResponse] => Unit,
retriesLeft: Int)
+    extends HttpRequestContext {
+    def withRetriesDecremented = copy(retriesLeft = retriesLeft - 1)
+  }
```

```
     protected def receive = {
       case send: Send => config.dispatchStrategy.dispatch(send, conns)
-      case Respond(conn, Send(request, responder), result) =>
-        log.debug("Dispatching '%s' response to %s", result.fold(_.toString, _.status.value),
requestString(request), result)
-        if (result.isLeft || closeExpected(result.right.get)) {
+      case Respond(conn, send: Send, result@ Right(response)) =>
+        log.debug("Dispatching '%s' response to %s", response.status.value, requestString(send.request),
response)
+        if (closeExpected(response)) {
           conn.pendingResponses = -1
           conn.httpConnection = None
         } else conn.pendingResponses -= 1
-        responder(result)
+        send.responder(result)
+        config.dispatchStrategy.onStateChange(conns)
+      case Respond(conn, send: Send, Left(error)) if send.retriesLeft > 0 =>
+        log.debug("Received '%s' in response to %s with %s retries left, retrying..
          error.toString, requestString(send.request), send.retriesLeft)
+        conn.pendingResponses = -1
+        conn.httpConnection = None
+        config.dispatchStrategy.onStateChange(conns)
+        config.dispatchStrategy.dispatch(send.withRetriesDecremented, conns)
+      case Respond(conn, send: Send, result@ Left(error)) =>
+        log.debug("Received '%s' in response to %s with no retries left, dispatching error...",
error.toString, requestString(send.request))
+        conn.pendingResponses = -1
+        conn.httpConnection = None
+        send.responder(result)
```