Generic Session Layer to make reliable Sockets more reliable against ECONNRESET

Alexander Aring
RedHat, Inc.
Ottawa, Canada
aahringo@redhat.com

Abstract

This paper is about the technical background of the talk "ECONNRESET on a reliable socket and now?" presented at Netdev Conference 0x15 [8]. It's about how possible react on a connection oriented reliable socket if an "ECONNRESET" error occurs. As experience showed, most users will simple reconnect and let the data flow going on as nothing happened before. This requires to close the socket and create a new one to do the actual reconnect. Reacting on like such way will occur that the previous existing socket will lose it's state. Moreover pending data inside writequeue of the socket will be dropped which was declared on application layer as successfully transmitted before. Most users forgot about this behaviour and are not aware about that they leave the world of their reliable connection.

This paper will show why "ECONNRESET" can suddenly occur and how Distributed Lock Manager (DLM) filled the hole of reliability if "ECONNRESET" occurs. It was solved by introducing a transparent encapsulation header below the DLM application layer. This header introduced a session layer to be sure data has been arrived at the other end. If "ECONNRESET" occurs and a reconnect will be done, DLM will trigger a retransmit of pending messages again.

Furthermore this paper will discuss about a more generic solution which avoids application layer changes like it was done for DLM. DLM can switch in later versions back to such generic solution but there are other in-kernel candidates who can benefit switching to it.

Keywords

Distributed Lock Manager (DLM), Transmission Control Protocol (TCP), Stream Control Transmission Protocol (SCTP), Reliable Sockets, Transport Layer, Session Layer, CIFS, Samba, Half-Closed Socket, ECONNRESET

Introduction

This section will show the basic knowledge about "ECONNRESET" and why it could suddenly occur on your reliable socket. A quick look into DLM will be done to explain how "ECONNRESET" can effect the reliability of a DLM socket.

ECONNRESET

The errno "ECONNRESET" is a possible return value of "sendmsg()". As the manpage points out it stands for "Connection reset by peer.". But what does that actually mean? On a Linux TCP socket it can occur after a "TCP Reset" was received. If such a situation happens the Linux TCP stack will purge it's pending socket writequeue. Usually the socket becomes unusable and the application will close the socket and trigger a reconnect to the same peer again. RFC 3360 [3] points that the original reason why "TCP Reset" was introduced comes back to signal it's peer a receive of a noncomplaint TCP Header. Due malfunction network entities like "Firewalls" or "Load Balancers" [3] it could be that such network components trigger a "TCP Reset" to an endpoint. On the endpoint side the application will then end in the mentioned "ECONNRESET" error situation.

Session Layer

A session layer is settled in layer five of the OSI-model. Usually a session layer is not handled by the Linux kernel, it is part of the user space application to provide such a functionality. Therefore a session layer lifetime is not equal to the socket lifetime. An example for a session layer could be that it ensures that the socket communication is reliable as it will be shown in this paper.

One of the application decision to use a reliable connection oriented socket such as TCP or SCTP [10] was made to not drop any network communication to it's peer. Without additional handling such as the mentioned example to provide a session layer a drop can occurred when receiving an "ECONNRESET" error. If the application holds a peer per state, like DLM, a drop of any networking data can end in a cracked situation. A missing additional handling to resolves the side effects of "ECONNRESET" DLM can end in such a situation. The application layer was only aware that there was a problem, if a drop occurred or not is unknown. To know which data was dropped a session layer can be used to keep track which data was really received at the endpoint.

TCPkill

TCPkill [2] is a tool to trigger "TCP Resets" for a specific TCP connection. It collects flow information and uses a raw socket type to interfere the TCP connection and start to trigger a "TCP Reset". During this work TCPkill was used to trigger

"TCP Reset" for a DLM connection so the DLM application socket will fail with "ECONNRESET" which ends likely in a drop of DLM data.

Distributed Lock Manager Protocol

The Distributed Lock Manager Protocol is an Linux kernel upstream message based application layer protocol. As state of written this paper, the protocol implementation sits in "fs/dlm". A cluster manager like corosync [1] is required. One reason is that DLM is used inside the Linux cluster world to lock distributed resources like shared block devices e.g. iSCSI [4]. Such shared block devices will not control mutual access to the resource of all Linux cluster nodes. A cluster filesystem such as GFS2 [6] or OCFS2 [7] can be used to operate on such shared resource. One main user of DLM is GFS2 to control it's mutual access to the shared block device resource on all nodes.

It is important that DLM does not drop any of it's messages. It depends on the exact message type if DLM could ends in a cracked situation. However a message could contain an unlock operation to a specific lock, if such message gets drop a deadlock situation will occur. DLM has a built-in deadlock detection, but it should be avoided to run in any deadlock situation.

Cluster Manager

The cluster manager controls the membership of cluster resources, such as DLM. A cluster does usually have several nodes and can handle different shared resources within the cluster. In this paper it's important that every node has the possibility to do the following operation to the DLM cluster resource:

Join Membership Clean join of cluster resource

Leave Membership Clean leave of cluster resource

Fence (Close) Usually power off/on machine and rejoin

For DLM it is important that no drop can occur between "Join" and "Leave" membership cluster event. If a "Fence" (in this paper also named as "Close" event) occurs a hard disconnect is required and all current pending data must be dropped. When a node rejoins the cluster membership a special protocol handling will be executed to synchronize with other cluster nodes again.

Half-Closed Sockets

A special node about half-closed functionality will be mentioned in this paper. The reason is that SCTP does not support the functionality of half-closed sockets. DLM requires half-closed socket functionality due the fact that a leaving node from it's membership can still receive DLM messages up to the point that the peer receives the leaving event of the node. As side issue this paper will also solve to provide half-closed sockets for SCTP.

However during research it seems there are devices (Firewalls, NATs, Linux "net.ipv4.tcp_fin_timeout", ...) out there which breaks TCP half-closed socket behaviour. Some of the behaviours are justified to prevent DoS attacks, some other are related to a buggy implementation. For buggy implementations e.g. a NAT devince, a connection waits after the first TCP FIN [5] and will remove the NAT entry. If the NAT entry is removed the connection will stop working immediately.

Problem Detection

There exists ways about how to detect if your application is affected to ECONNRESET issues. This section will describe how an socket application can detect if such issues exists.

ECONNRESET

If an application does on a "sendmsg()" error an reconnect without additional handling to prevent drops it could be a first sign that the application is not able to provide reliability if such scenario occurs. As experience shows with DLM the easiest way to check if an application can handle it, "tcpkill" in the background can be run and observing the effects of large amount of "ECONNRESET" errors. This will probably not reflect the reality but it will trigger a reconnect at the right point that the socket write queues will be dropped. In case of DLM it will soon end in a deadlock.

Half-Closed Sockets

To detect issues with half-closed is more complicated and no tool such "tcpkill" exists yet. In case of a half-closed socket requirement and the application can either run on TCP or SCTP like DLM a problem definitely exists because SCTP does not support half-closed sockets.

Possible Solutions

This section will show possible solutions which was being discussed to provide a solution for the observed "ECONNRESET" issue. Also to a possible solution to solve the missing functionality of half-closed sockets in SCTP will be shown.

Ignore TCP Resets

A simple and harsh solution would be to simple ignore TCP resets. TCP resets would be only be ignored if the TCP connection is inside the "ESTABLISHED" state. During this work a fast hack was done to test if it's possible to just ignore TCP resets with success. However the socket gets not in a "reset" state in sense that a new TCP connection will be established and this is usually what a potential peer like a load balancer wants to provoke. It is TCP specific only as well.

TCP_REPAIR

TCP_REPAIR [9] is a technique to store and restore TCP states in Linux. It was introduced for migrating ongoing TCP connections to another Virtual Machine or Namespace. The idea is to use TCP_REPAIR to store the TCP state when an "ECONNRESET" occurs and restore the TCP state after creating the new socket. If both sides enables such behaviour the TCP socket can going on again like "ECONNRESET" was not happened before.

There are several disadvantages of this solution. One is that this solution is Linux specific only. It is not only Linux specific, it is Linux TCP specific as well. That means there is

currently no support for SCTP. It is required to add such feature for SCTP as well. Besides that it is required that the other peer is aware that such handling is enabled. Another problem which occurs is that the Linux TCP Stack will purge the write queue if a TCP reset arrives. At this point it is already too late to store the socket state. It might be possible to change this behaviour by introducing another socket option that the write queue will not be purged at arrival of TCP reset.

Session Layer

To prevents drops during "ECONNRESET" a session layer between the application and trasnport layer e.g. TCP can be introduced. This session layer will put the stream data into a message based structure and confirm that the endpoint received it. If a reconnect happens due "ECONNRESET" phase every non confirmed message will be resend to the peer in order as they was received. To provide such mechanism a simple sequence number and additional acknowledge message will be introduced on session layer level. It will guarantee that at peer side we will retransmit all unacknowledged messages, the peer can filter out all already received messages which was not acknowledged by the sender yet.

Half-Closed Sockets

To provide half-closed sockets or solve issues with half-closed regarding to buggy implementations an additional FIN message like in TCP [5] can be introduced. The advantage is that this message is not provided by TCP and common networking device (e.g. firewalls or load balancers) will not react on such messages and hopefully will never react on those in future. In case of SCTP introducing a FIN message on application layer will provide half-closed socket functionality. The TCP state diagram will therefore changed to provide only "ESTABLISHED" state until "CLOSED" to provide FIN state changes. This can also be provided by a session layer which takes care of reliability of transmitted messages.

DLM Solution

This section describes how DLM solved the issue of "ECONNRESET" which appeared suddenly. A session layer had been introduced on the DLM application layer. It acts transparently between TCP and DLM application layer. Each DLM messages are hold in memory by using a reference counter until it will be acknowledged from the other end. In case of reconnect unacknowledged messages will be retransmitted in order as transmitted originally. The receiving side will deliver messages according to their sequence number which is incremented by each message. If a message is received twice recognized by the sequence number, it will be dropped until the next expected message is received.

Figure 1 shows the layer diagram how DLM introduced the session layer. Important is to mention that the DLM socket acts underneath the session layer. The next section will discuss more about how to move the session layer underneath the socket handling.

For half-closed sockets a new message type "FIN" was introduced and acts like a TCP "FIN". The difference is that

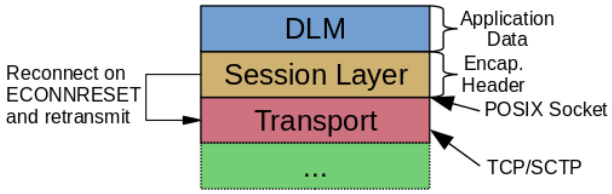

Figure 1: This figure shows the layer diagram of the DLM session layer. The application socket handling sits below the session layer.

the introduced "FIN" message works on top of TCP and is not part of TCP itself.

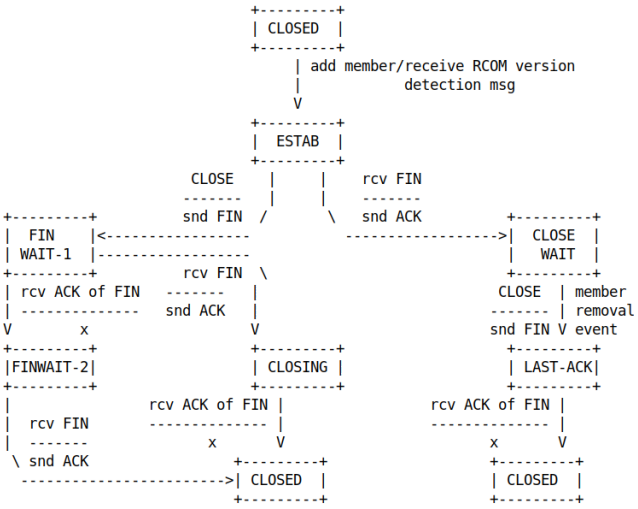

Figure 2: This figure shows the state diagramm like TCP for providing half-closed sockets on the DLM session layer.

Generic Solution

To implement the DLM solution an application layer change was required which had the effect to increase the DLM protocol version. This section will show a generic solution which does not affect the application layer layer. If it's being used, it is required that the other peer supports it as well. It is not being used, then it doesn't matter if the other peer supports it or not which means it is still backwards compatible but not forwards compatible. The generic solution therefore will simple move the session layer below the application socket handling.

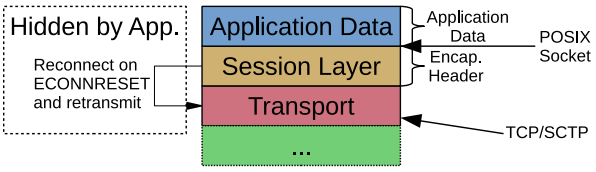

Figure 3: This figure shows the generic solution by moving the session layer below the application layer.

Figure 3 shows the layer diagram of the generic solution. The session layer becomes completely separated from the application layer and existing applications like DLM could easily switch to it without changing their application layer protocol. Such socket can operate on any transport layer, however it makes sense on reliable sockets only because the session layer will fix the gap of reliability on a reconnect. A reconnection because "ECONNRESET" will be handled completely transparently from the application. All socket related syscalls will be redirected to the related transport layer handling. It's an idea of a "redirected socket" type which handles the session layer for any transport layer underneath.

If an immediately close of the socket is required, such as in DLM for fencing, the "close()" syscall can be used. Otherwise syscall "shutdown()" can be used to invoke half-closed socket behaviour.

Potential Other Users

This section will show potential other users for such proposal session layer as described previously. As mentioned "ECONNRESET" can happened any time and the most users will do a reconnection and take possible drops as risk. It doesn't matter if the user is in user space or kernel.

Checklist

During this work a checklist was developed to indicate possible users:

1. Join/Leave Events - Session Layer Lifetime
2. Reconnect on Errors
3. Test with tcpkill

The final test for "tcpkill" requires observation of the protocol behaviours if actually a problem exists.

CIFS

I adapted the Checklist to CIFS. CIFS [11] is a network filesystem for Windows shares or a Linux based samba server.

1. Mount/Unmount - Session Layer Lifetime
2. Reconnect on Errors
3. Test with tcpkill

The session layer lifetime will be mapped with join/leave events to mount/unmount calls. During this time no drop should be occurred. It has the same reliability requirement like DLM for it's cluster manager events. Code observation resulted into that CIFS triggers a reconnect on "sendmsg()" error. Finally a "tcpkill" test on port sambas default port 445 with a simple file copy with "dd" and "/dev/zero" resulted into a stuck of the dd user application which was probably resolved about 30 minutes. Other filesystem calls like "open()", "close()" resulted into "EAGAIN". Due the fact that most filesystem applications (means user programs using POSIX "open()", "read()", "write()" and "close()") doesn't handle such error handling correctly a switch to such generic session layer should hide such errors and resolve them inside the session layer. That means that a filesystem application will not see such errors anymore and reduces the risk of failure in the user program.

To use the generic session layer it is required on the server and client side that such transparently handling of "ECONNRESET" will be done. If either doesn't support it the session layer will not be invoked and the old behaviour is being used.

Future Work

The DLM session layer was done and so far no problems was detected while running "tcpkill". The state before was most likely that DLM run into a deadlock state if "tcpkill" was running in the background. The DLM application layer can still switch from the existing DLM solution to the generic solution. To implement such generic solution would be one of the next steps. Therefore the generic solution with the special "redirection socket" is needed. Such implementation was not done yet but how it will act was shown in this paper.

The program "tcpkill" is a good testing tool to check if an application layer has problems with handling "ECONNRESET". However it is TCP specific only, but there exists ways to develop such tool for SCTP as well. The idea would be to send SCTP heartbeats [10] from another address than it's peer as this should end in an "ECONNRESET" error on SCTP.

References

[1] Corosync. Corosync - The Corosync Cluster Engine. https://corosync.github.io/corosync/.

[2] Dug Song. tcpkill - kill TCP connections on a LAN. https://www.monkey.org/ dugsong/dsniff/.

[3] Floyd, S. 2002. Inappropriate TCP Resets Considered Harmful. RFC 3360, RFC Editor.

[4] J. Satran and K. Meth and C. Sapuntzakis and M. Chadalapaka and E. Zeidner. 2004. Internet Small Computer Systems Interface (iSCSI). RFC 3720, RFC Editor. http://www.rfc-editor.org/rfc/rfc3720.txt.

[5] Jon Postel. 1981. Transmission Control Protocol. RFC 793, RFC Editor. http://www.rfc-editor.org/rfc/rfc793.txt.

[6] Linux Kernel Documentation. Global File System 2. https://www.kernel.org/doc/html/latest/filesystems/gfs2.html.

[7] Linux Kernel Documentation. OCFS2 filesystem. https://www.kernel.org/doc/html/latest/filesystems/ocfs2.html.

[8] NetDev Society. Netdev 0x15 - Home of THE Technical Conference on Linux Networking. https://netdevconf.info/0x15.

[9] OpenVZ. Checkpoint/Restore In Userspace. https://criu.org/Main_Page.

[10] R. Stewart. 2007. Stream Control Transmission Protocol. RFC 4960, RFC Editor. http://www.rfc-editor.org/rfc/rfc4960.txt.

[11] SAMBA Team. Common Internet File System. https://www.samba.org/cifs.