Multi-Rail Routing Requirements

Original Pre-Health Requirements

Router Requirements

A router is a node which has the routing feature turned on using `lnetctl set routing 1` or the equivalent modprobe configuration.

1. Track the last time stamp any message was received on a local NI
2. if the NI hasn't received any traffic for a period of `router_ping_timeout + MAX(live_router_check_interval, dead_router_check_interval)` then it's marked down
    a. This is done so that other nodes using the gateway can mark the route down, given that `avoid_asym_router_failure` is set to 1.
3. Do not send messages to a peer NI which is marked down.
4. Set the peer status to up when messages are received
5. For each peer NI that is marked down, when there are messages to forward to it, query it at least once per second to check if it is back up. If the query result determines that the peer NI is reachable, the peer NI state is set to UP. Messages can then be send to that peer NI.

Gateway Requirements

A gateway in this context is the peer NI created when adding a route on a node. For example: `lnetctl route add --net tcp --gateway <gateway-NID>.` Dealing with that peer-NI is somewhat of a special case.

1. Mark the gateway peer NI as down when the LND fails to send a message
    a. Note although the LND notifications happen for all peer NIs it is only pertinent on routers or for gateways.
2. Mark the gateway peer NI as up when we receive an unsolicited message or when we receive a REPLY for a PING sent from the router checker.
3. Mark the route as down if one of the gateway's interfaces, identified by the gateway peer NI, are down, provided the `avoid_asym_router_failure` is set to 1.

Peer Requirements

1. Do not check for peer aliveness when sending a message to a peer.
2. Pick a route which has its gateway peer NI marked as up.

Implementation Details

The routing infrastructure currently performs the following functionality

1. Keep track of the last time the peer was alive, `lpni_last_alive`
2. Keep track the last time the peer was notified that its state has changed, `lpni_timestamp`
    a. The peer can change state under the following conditions:
        i. The LND notifies that the peer is down when it fails to send a message to the peer.
            1. As an example in o2iblnd:
                a. kiblnd_peer_connect_failed() and kiblnd_disconnect_conn() call kiblnd_peer_notify() which calls lnet_notify() to set the peer to `dead` if there was an error
        ii. A message is received in `lnet_parse()`
            1. In this case the peer state is set to alive only for gateway peer NIs
        iii. When the router checker ping is responded to or it fails.
        iv. If the router checker ping times out.
3. This step only concerns routers: Only send the message if the peer is alive, determined as outlined above.
4. On the router if the NI hasn't received any traffic for a period of `router_ping_timeout + MAX(live_router_check_interval, dead_router_check_interval)` then it's marked down.

    a. This is done in order for the peers using the router to mark the peer down when the `avoid_asym_router_failure` is set to 1, which it is by default.

LNet Multi-Rail Routing

Multi-Rail introduced the concept of a peer and a peer NI. A peer can have multiple peer NIs. This changes the semantics of route configuration. Currently a route can be configured as:

```
lnetctl route add --net <remote net> --gateway <gateway-peer-NID>
```

The gateway-peer-NID refers to a specific interface on the router. However with MR enabled on the router, multiple interfaces can be configured on the same network. Therefore, the configuration semantics should be as follows:

```
lnetctl route add --net <remote net> --gateway <gateway-primary-NID>
```

The `gateway-primary-NID` must be on a local network. IE: a network which is reachable from the node the route is being configured on.

A router should be discovered on first use. The discovery process will determine all the interfaces available on the router. There could be multiple interfaces on the same network.

A route should only be marked down if it can not route message, which means that the route's remote net on the gateway has no active interface.

Nodes on different networks will use different primary NIDs to refer to the same router. IE a primary NID is only a representation of the router on the peer with the route configured.

Multi-Rail Router Requirements

1. Configure a percentage of the maximum health below which an interface will not be selected for use. This percentage value will be referred to as `router_sensitivity_percentage`
    a. 100% means that an interface which has less than MAX_HEALTH will not be selected for use
    b. 0% means that an interface will be selected for use as long as it has the best health value among the available interfaces.
2. Do not put message on the wire if the health of a peer_ni is below `MAX_HEALTH * router_sensitivity_percentage`
3. Attempt to recover an unhealthy peer_ni once per second by pinging it
4. LND shall notify LNet whenever it determines a peer_ni is alive or dead. The API will provide a parameter which will force LNet to fully recover the peer_ni's health.
    a. Currently the gnilnd is aware of the gni network health and therefore it can inform the LNet layer when a peer is alive or dead. In fact the gnilnd is the only LND which informs LNet when the peer is alive. All the other LNDs only tell the LNet when the peer is disconnected. Therefore the gnilnd can set the `fully_recover` parameter to true, while the other LNDs can set it to false.
5. LNet shall call an LND API to notify that a peer_ni is dead whenever the peer_ni's health goes below `MAX_HEALTH * router_sensitivity_percentage`
    a. This is derived from the current code and is only applicable to socklnd.

Multi-Rail Route Requirements

1. A route is considered down if there are no viable peer_nis on the remote net of the gateway
    a. EX: if a route is defined as: `lnetctl route add --net tcp2 --gateway 10.10.10.3@tcp`, then if the gateway defined as `10.10.10.3@tcp` has no healthy `peer_nis` on tcp2, then that route is dead
2. A gateway is consider down under two circumstances:
    a. All remote nets reported in the `REPLY` to the `PING` are down
    b. All local representation of the peer_nis on the remote net have a health value below: `MAX_HEALTH * rtr_sensitivity_percentage`

Configuration

A router can be configured as follows to utilize the new health infrastructure

1. `lnet_health_sensitivity >= 1 ## this will decrement the health of the NI by the value specified everytime there is a failure to send to that interface`
2. `router_sensitivity_percentage = 100 ## this will consider the route down if there is no NI on the remote net of the gateway with health == LNET_MAX_HEALTH_VALUE`
3. Optionally we can set `retry_count > 0 ## this will attempt to resend a message on a different NI if one is available`

Route Selection

Currently a route is selected based on the priority and hops value given to it, after that the credits for the peer NI are evaluated. With Multi-Rail there should be a two evaluation factors in the selection process.

1. Hops and priority
2. If multiple NI's exist on the desired network one must be selected based on the existing selection criteria, health, credits and finally round robin.

A mechanism should be created to restrict the selection to a group of peer NIs that could belong to different gateways that can reach the same remote network.

In this way the Mulit-Rail aspect of the gateway is considered.

Furthermore, with the LNet Resiliency feature the healthiest interface of the router or set of routers is selected.

LNet Resiliency

The LNet Health/Resiliency feature has added the following features:

1. Track a local NI or a peer NI health.
2. Re-send messages
3. Recover local NIs or peer NIs by pinging them every second.

The original route code which implements the requirements outlined above are no longer inline with the new mechanisms implemented. There needs to be an effort taken to bring the router code more inline with the new features implemented.

Some details were documented here: Routing and MR integration

Proposed Changes

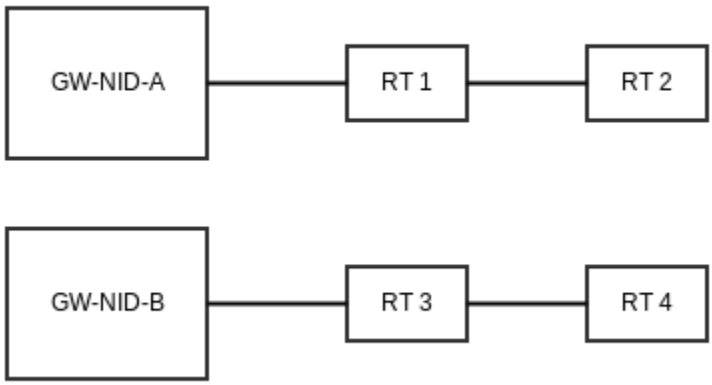
Router Discovery

There are two ways to discover a router:

1. when it's first added to the system.
    a. The problem with this is it doesn't work if the router is not up yet. You fail to discover it when its added, so when would you try to discover it again?
2. When it's first used - This is the option which will be implemented
    a. This is the most reasonable solution, since we check if the router has already been discovered when we first attempt to use and if it has not we initiate discovery. We will need to deal with this as we would discovery final destination in the sense that we'll need to queue the message to be sent once we finish discovery the router.

Different routes can be added using different NIDs of the same gateway. When the gateway is discovered on first use there will be a need to consolidate the routing information.

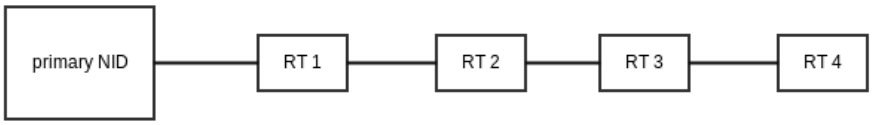
For example, let's take the scenario where SET-A of routes were entered through the gateway using GW-NID-A and SET-B of routes were entered through the gateway using GW-NID-B. This will create the following structure:



When the GW is discovered three scenarios are possible:

1. GW-NID-A is the primary NID
2. GW-NID-B is the primary NID
3. GW-NID-X is the primary NID

In all these cases we will need to consolidate the routing information as follows:

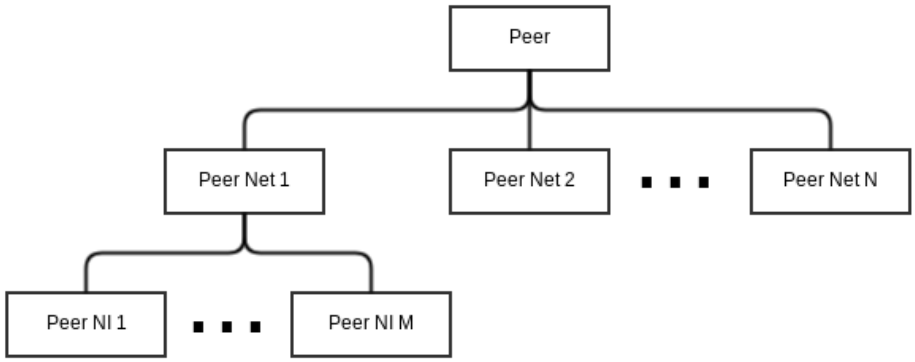


In the discovery code the consolidation of the peer information is driven from: `lnet_peer_data_present()`

The change is tracked under: LU-11292 - Getting issue details... STATUS

Router Peer instead of Peer NI

Multi-Rail considers that a Peer can have multiple interfaces, IE Peer NIs, on different networks.



Currently when a route is added the gateway is one of the leaves of the tree, IE a peer NI. To fully integrate routing with MR the gateway should be considered the tip of the tree, IE the peer.

This change will overhaul the current routing code. There are several fields in the `struct lnet_peer_ni` which are used for routing:

```
    /* messages blocking for router credits */
    struct list_head    lpni_rtrq;
    /* chain on router list */
    struct list_head    lpni_rtr_list;
    /* # times router went dead<->alive. Protected with lpni_lock */
    int             lpni_alive_count;
    /* # refs from lnet_route_t::lr_gateway */
    int             lpni_rtr_refcount;
    /* routes on this peer */
    struct list_head    lpni_routes;
    /* router checker state */
    struct lnet_rc_data    *lpni_rcd
```

These fields will need to be transitioned to the peer, and all areas of the code which use them will need to be modified.

Tracked under: LU-11298 - Getting issue details... STATUS

Router Ping

Gateways are pinged on a configured interval. If the Gateway is dead, then there is another configuration parameter which governs the frequency of the ping to determine if gateway is back up.

The ping requires somewhat of an extensive infrastructure, including MD/EQ, an event handler and router state management. Much of that can be consolidated with the Discovery code. The Discovery code currently implements all the infrastructure required to manage sending pings and receiving responses. The intent of this proposal is to use this existing infrastructure in place of the router pinger.

The monitor thread calls a function to check if the gateways should be pinged.

The function traverses the list of gateways and sends out a ping if it is required.

Code exists to handle receiving a reply for the ping. When a REPLY is received a function is called to analyze the NIs in the REPLY. This analysis revolves around checking the status of each of the peer interfaces. If the asymmetric router failure is set and one of the interfaces provided in the REPLY is down, then the gateway is marked down.

There are a couple of significant improvements/simplifications that can be done in this area:

1. Use the discovery mechanism instead of keeping the ping handling code.
    a. This will reduce the complexity of the router code significantly. All the lnet_rc_data, mdh, event handling will simply go away.
2. When a response is received to the ping, the discovery code can simply check if the peer is a router and if so call a routing function to further handle the response.
    a. This code will need to ensure that this gateway is viable for all the routes which use it.

Tracked under **LU-11299** - Getting issue details... STATUS

Router Aliveness and Health

Currently, there is some tricky code to determine the aliveness of a peer. The intent of the code seems to be divided into two categories: Router and Gateway

A router is a node which has routing feature enabled

A gateway is a peer_ni which references a router

Aliveness on a Router

1. If a peer NI is dead do not send messages to it
2. If a peer NI is dead query it every 1 second to see if it's back up when there is traffic.
3. If messages are received on a peer NI set it's aliveness to up
4. If a local NI does not receive a message for a configured period of time, then bring down the status of the local NI. That will be discovered when the router is pinged.

Aliveness for a gateway

1. Set the gateway peer NI to down if there is a failure to send a message
2. The gateway peer NI is pinged every configured period of time.
3. Set the gateway peer_ni to down if the routing is not possible through it
4. Mark the gateway peer NI as up when it receives a message

Most of these requirements can be consolidated with the Health feature. Here is the proposed solution:

1. Turning on sensitivity to > 0 will track the health of the peer NIs
2. When sending messages the healthiest peer NI is selected. This reduces the need to drop messages before they are passed to the LND.
    a. The current code has an UP/DOWN behavior. There is no granularity. Is that a requirement?
        i. After discussion with Cray this appears to have been introduced due to Cray's gni network, which has health baked into it. So when the gnilnd reports that a peer is alive then it's sure it's alive, similarly when it's dead.
        ii. This has been considered in the requirements outlined under the "Multi-Rail Router Requirements" section.
3. Unhealthy local and peer NIs are placed on their respective queues for recovery. This takes the place of querying peer NIs in the case of Router. Extra functionality can be added to remove the peer-NI from the recovery queue if it has not been used for a peer_timeout length of time
4. There will be no need to ping a gateway peer NI separately to determine if it's back up. This will be done by the health code.
5. When selecting a gateway peer ni the health of the interfaces will be considered. If the intent is to not use a gateway peer_ni if it's less than fully healthy a configuration parameter can be added to control that, `router_sensitivity_percentage` described above.
6. When messages are received or sent successfully on any peer_ni it's health value of the NI is incremented, making it more likely to be used.
7. If a router does not receive a message for a configured period of time on a local NI, then bring down the status of the local NI. That will be discovered when the router is pinged.
8. Instead of marking a gateway as up or down, mark a route as up or down. You can have multiple routes through the same gateway. Depending on which interfaces are up on the gateway a subset of these routes could function through the gateway.
9. Allow multiple routes to the same remote network over multiple gateways.

Using this proposal the router and gateway requirements will continue to be achieved, while at the same time, getting rid of lots of code which is currently used to keep track of the aliveness of the peer NIs.

Tracked Under: **LU-11300** - Getting issue details... STATUS