Fujitsu Channel Bonding

- Introduction
- Overview of Functionality
- Configuration
- Data Structures
    - Overview of Additions
    - kib_routes_t and kib_route_t
        - Route Recovery
    - kib_target_t
    - Object Relation Diagram
    - Object Relation Description
        - Logical Network Interface Group
            - ibn_routes
            - ibn_nifaces
        - Routes
            - kib_routes_t
            - kib_route_t
        - kib_peer_t
- Run time Behavior
    - Runtime Flow Diagram
        - Example One
            - Node A --> sends 3 messages --> Node B
        - Example Two
- DLC Configuration Interface
    - Network Configuration
    - Target Configuration
        - YAML Configuration
        - lnetctl commands
    - Network Status
        - YAML Configuration
        - lnetctl command
    - Configuration Sanity Check
        - lnetctl command

Introduction

This document describes the IB Channel Bonding design and implementation done by Fujitsu.

Overview of Functionality

IB Channel bonding is a feature by which multiple Host Channel Adapter (HCA) cards are combined to provide redundancy and/or throughput.
The Fujitsu implementation works by grouping multiple HCA cards into one **Logical Network Interface Group**, which is referenced by a NID. Thus this NID references a logical Network Interface which abstracts the use of the different HCA cards. This abstraction is presented at the LND layer. The LNET layer is only aware of a network interface (lnet_ni_t), essentially the Logical Network Interface Group. When configuring a Network from the modprobe or from the DLC interface multiple network interfaces can be associated with a single network; for example in lustre.conf: networks=o2ib(ib0, ib1). This configuration will make it down to the o2iblnd and it will configure two devices for ib0 and ib1 to be used in

round robin when sending messages over o2ib network. This information is stored in lnet_ni_t.ni_data, a void parameter, which abstracts the different devices discovered by the LND driver, given the names of the interfaces provided in the lustre configuration. When a connection is established to a peer a local HCA card and remote interface pair (termed a 'route') is selected. ***(Routes are discussed later on in the document)***. The selection is done using a round robin algorithm. Every time a communication with the peer is requested the next available route is selected using a round robin algorithm. This implementation ensures that all local HCA cards and remote interfaces identified in the configuration are utilized; thereby increasing the throughput. The implementation also ensures that if a local HCA card becomes inactive it is no longer used until it becomes active once more.
There is a mechanism to trigger the recovery of disconnected HCA cards and routes, described later in this document.

Configuration

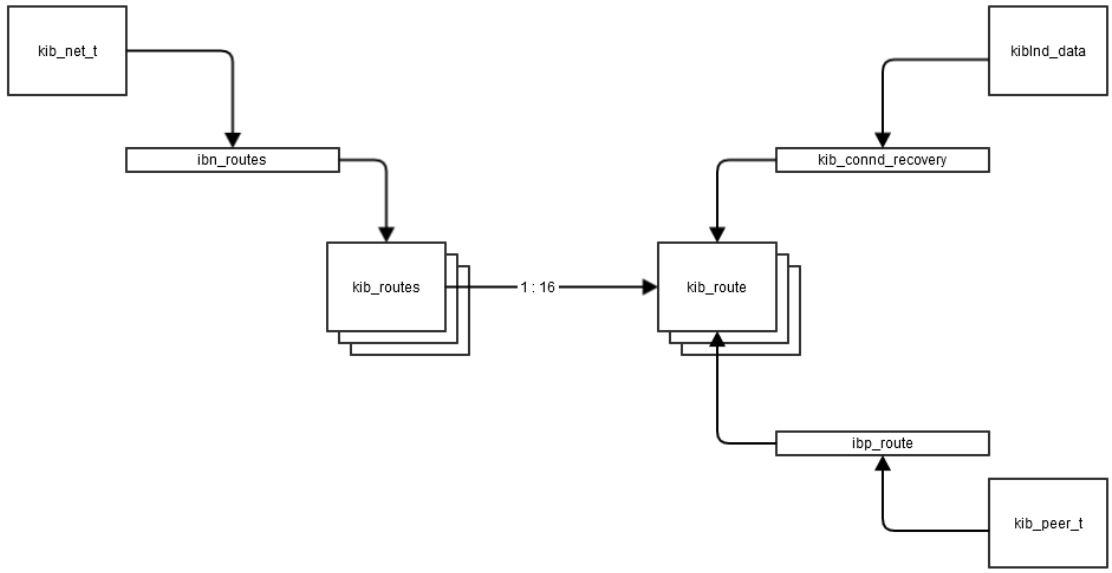
The following are the configuration steps that are needed to activate the Fujitsu IB Channel Bonding implementation.

1. Configure the networks module parameter. Depending on the number of HCA cards in the system, the configuration of this parameter will abstract the actual devices under one NID
    1. Ex: a system could have 4 HCA cards installed: ib0, ib1, ib2 and ib3. The networks parameter can identify two separate networks, each containing two devices:
        1. networks="o2ib0(ib0,ib2), o2ib1(ib1,ib3)"
    2. The same configuration can be done dynamically via DLC.
2. Target configuration. Fujitsu added a new construct called targets. These targets are parsed from the file: /etc/sysconfig/lnet_o2ibs.conf. This file is of the following format:
    1. NID <ip1> … <ipN>
        1. Each line basically says: A nid "NID" is reachable via the following IP addresses: <ip1> … <ipN>
    2. These targets can also be configured via the DLC interface using YAML config file, or through the lnetctl utility
    3. A proper configuration would at least have an entry for each local NID and the IP addresses it encapsulates (IE the IP addresses of the HCA cards in the Logical Network Interface Group). In our example, we have 4 HCA cards and thus 4 IP addresses
        1. Ib0: 10.1.1.2
        2. Ib1: 10.1.1.3
        3. Ib2: 10.1.1.4
        4. Ib3: 10.1.1.5
    4. Thus the configuration file must contain the following entries:
        1. 10.1.1.2@o2ib0 10.1.1.2, 10.1.1.4
        2. 10.1.1.3@o2ib1 10.1.1.3, 10.1.1.5
    5. From my examination of the code, this appears to be more of a sanity check, and has no real usage, unless a node is attempting to send messages to itself.
3. Furthermore, since other nodes would have similar setups, we'll need to identify how to communicate with them. This is achieved by adding target entries in the target configuration file. In our example if there is a node on the network which is trying to talk to the node identified above, it will have the following entries in its target configuration file:
    1. 10.1.1.2@o2ib0 10.1.1.2, 10.1.1.4
    2. 10.1.1.3@o2ib1 10.1.1.3, 10.1.1.5
        1. This means that you can reach the NID 10.1.1.2@o2ib0 via 10.1.1.2 and 10.1.1.4. When messages are being sent to 10.1.1.2@o2ib0, then both IP addresses (10.1.1.2 and 10.1.1.4) will be exercised in round robin.
        2. Same for 10.1.1.3@o2ib1

Data Structures

Overview of Additions

kib_routes_t and kib_route_t



kib_routes_t is an encapsulation of up to LNET_MAX_INTERFACES (16) kib_route_t entries. kib_route_t represents one route. A kib_routes_t is created, if one doesn't exist, when a connection is attempted with a peer NID. Before connecting to the peer, all possible routes to the peer must be determined. This is identified in the target configuration file via the following entries:
<peer NID> <ip1> … <ipN>

A mesh of routes are created using the following logic:

```
number_of_ips = get the number of peer NID IP address identified in the target file
number_of_interfaces = get the number of local interfaces
number_of_routes = the greater of number_of_ips and number_of_interfaces
for i = 0; i < numbe_of_routes; i++:
   create a new kib_route_t
   kib_route_t.rt_nid = peer nid
   kib_route_t.rt_addr = (number_of_ips) ? get peer target ip[i%number_of_ips ] : get
the ip encoded in the peer nid
   kib_route_t.rt_dev = dev[i % number_of_interfaces]
```

The newly created kib_routes is put on the kib_net_t.ibn_routes list, and looked up from there when sending a message.

Route Recovery

IB events are handled and the device status is maintained through these events. The three events that are currently maintained are:

1. IB_EVENT_DEVICE_FATAL
    1. The state of the device (ibd_state) is set to IBLND_DEV_FATAL
2. IB_EVENT_PORT_ACTIVE
    1. kiblnd_port_active_event() function is called to handle this event
        1. It querys the ib port for its attributes and determines the link speed. If the port link speed is less than the expected maximum, then the device is set to IBLND_DEV_PORT_DEGRADE
3. IB_EVENT_PORT_ERR
    1. device state is set to IBLND_DEV_PORT_DOWN

```
#define IBLND_DEV_LINK_SPEED(width, speed)    ((width) * (speed))
#define IBLND_DEV_LINK_WIDTH_4X   (4)
#define IBLND_DEV_LINK_SPEED_QDR  (4)
#define IBLND_DEV_LINK_SPEED_MAX(rate)  \
    IBLND_DEV_LINK_SPEED(IBLND_DEV_LINK_WIDTH_4X, (rate))
```

While finding a route (kiblnd_next_route_locked()), if a disconnected route is met, but the device's state is either IBLND_DEV_PORT_ACTIVE or IBLND_DEV_PORT_DEGRADE then that route (kib_route) is placed on the kib_connd_recovery list, and processed by the connd thread (kiblnd_connd()). That thread traverses the kib_connd_recovery list and reestablishes the connections on these routes.

kib_target_t

```
typedef struct kib_target {
    struct list_head tg_list;
    lnet_nid_t tg_nid; /* nid /
    int tg_naddr; / number of address entry /
    __u32 tg_addr[LNET_MAX_INTERFACES]; / target address */
};
```

kib_target_t instance is created per entry in the target configuration file.

For example if target configuration file contains the following:

```
10.1.1.2@o2ib0 10.3.3.3 10.3.3.4
10.1.1.3@o2ib0 10.3.3.5 10.3.3.6
```

Then two instances of kib_target_t are created and added on the global list kiblnd_data.kib_targets. This list is used to lookup the targets when creating routes. The peer NID is looked up in this list and the kib_route_t entries are created accordingly.

Object Relation Diagram

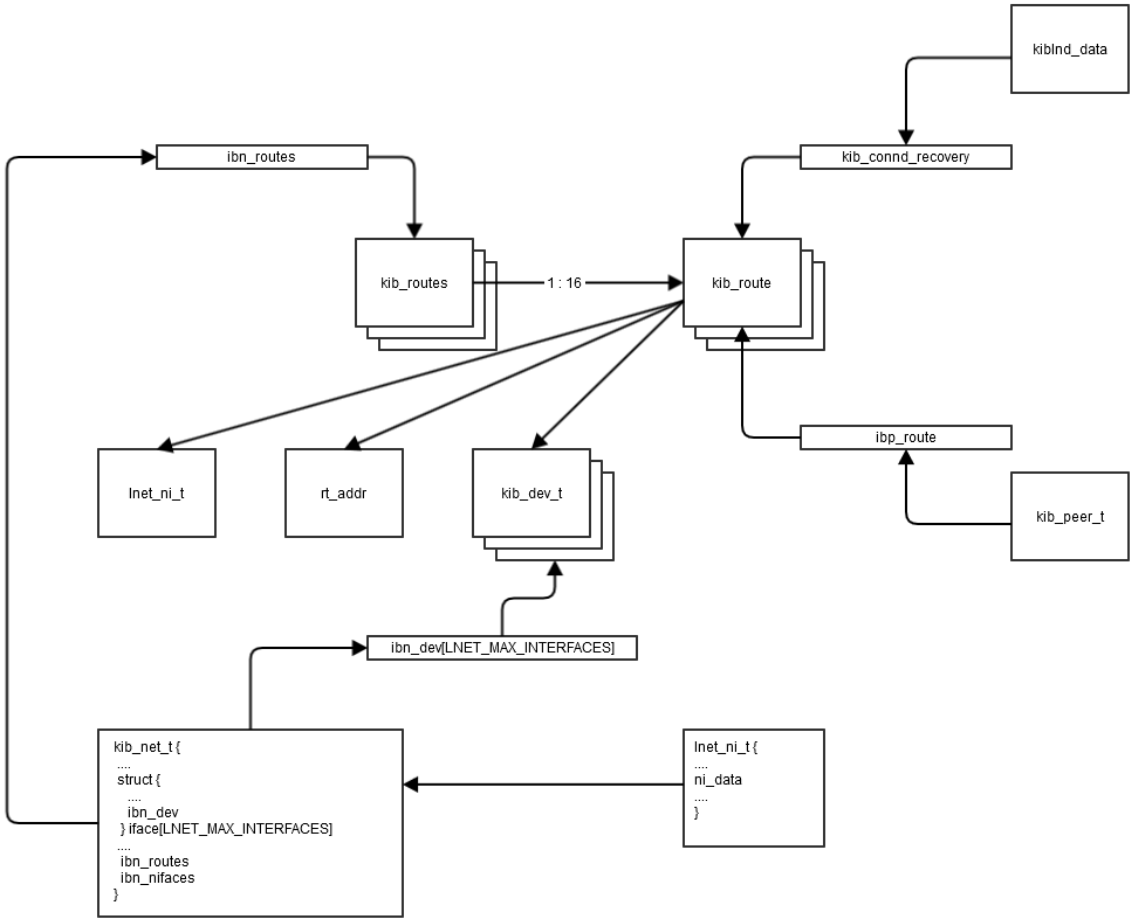


Figure 1: Channel Bonding Object Relation diagram

Object Relation Description

Logical Network Interface Group

An lnet_ni_t instance represents a Logical Network Instance Group. This representation is what the LNET layer deals with. lnet_ni_t.ni_data is a void pointer which is intended to be a pointer to the LND specific network instance representation.In the IB LND case, it is using kib_net_t, which has been modified to contain an array of size LNET_MAX_INTERFACES of physical interfaces. Each element in that array represents a physical IB interface. ibn_routes and ibn_nifaces have also been added.

ibn_routes

Is a list of kib_routes_t instances, each one representing the set of routes to a peer NID. As described above there are two ways these routes are derived:

1. Based on the target configuration file
2. Based on the number physical interfaces which are part of the logical network interface group.

These routes are created the first time the LND tries to connect to a peer through kiblnd_send().

ibn_nifaces

Counts the number of physical interfaces which is part of the logical interface group.

Routes

kib_routes_t

A kib_routes_t instance is created the first time the LND attempts a connection with a peer NID. Each kib_routes_t object has an array of up to LNET_MAX_INTERFACES kib_route_t objects. These kib_route_t are created based on the target configuration file and the number of physical interfaces in the group as described above.

kib_route_t

Each kib_route_t object defines which IP address to use to reach a NID and which local device (IE HCA card described via kib_dev_t) to use to send and receive messages.Henceforth, on every attempt to connect with a peer NID, a route (IE kib_route_t object) is picked in round robin, thereby, exercising all local devices, and all remote IP addresses identified by the peer. *NOTE: Explore the possibility of the peer identifying its own interfaces, rather than be defined by admin in a configuration file. This way peers would provide the list of IP addresses they can receive messages on through some protocol.*

kib_peer_t

A kib_peer_t object represents a remote peer that the local LNET is communicating with. The peer maintains two pieces of data which are pertinent to this discussion: NID and route. The NID is the target NID and route is the route (IE local device and remote IP address pair) messages going through this peer will traverse over.Thus it is possible to have multiple peers with the same NID, but different route*s.* The peer connection remains up until either side goes down. This is done to optimize performance. (Although it might be a good idea to consider striking a balance between performance and resources, by having some sort of a timeout on the peer connection. If it expires without any activity on the connection, the connection is brought down. If activity is detected, then the timeout is reset)

Run time Behavior

Runtime Flow Diagram

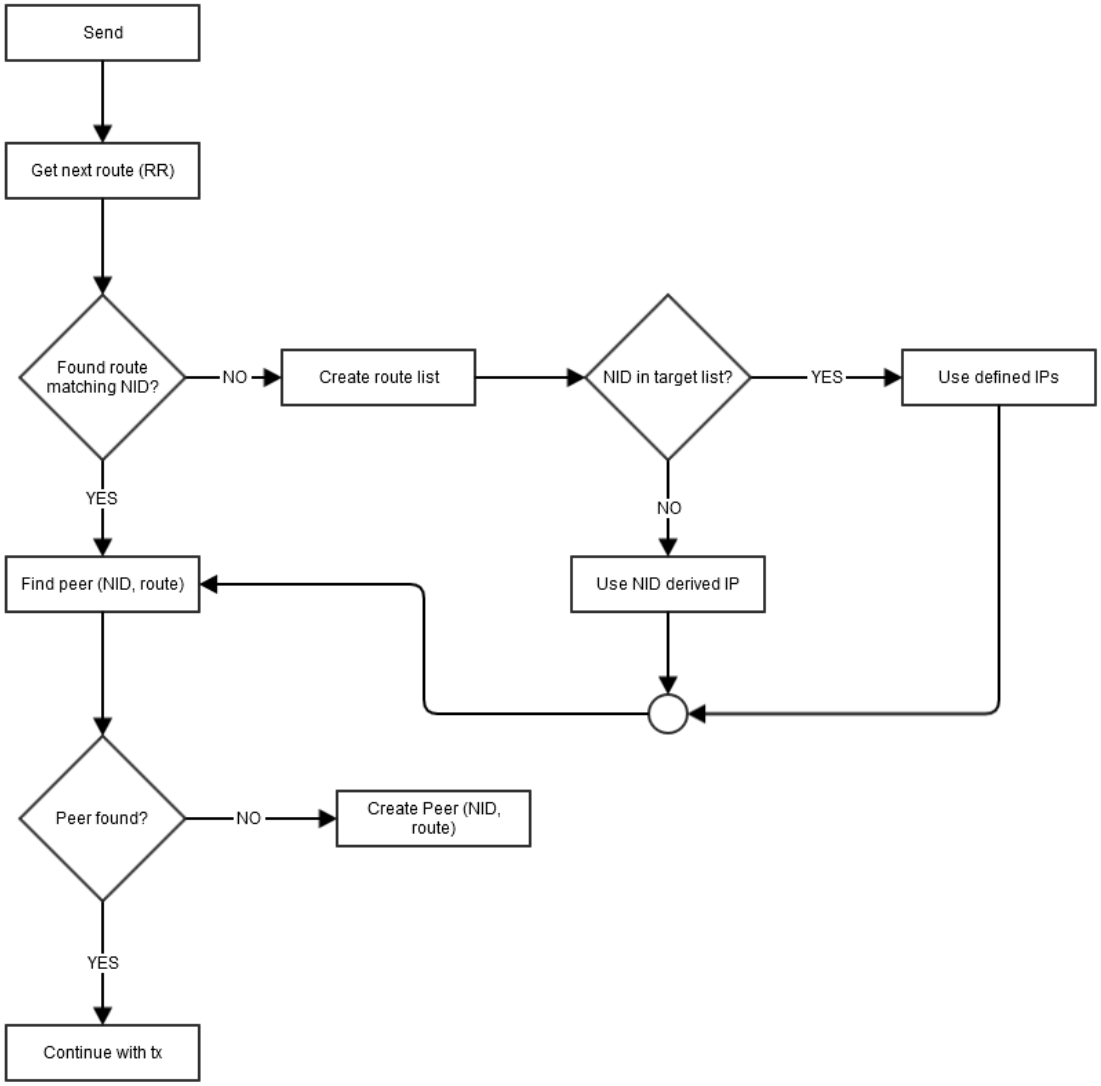


Figure 2: High-level IB Channel Bonding flow diagram

The system is configured to identify logical networks.
The system is then configured via user configured targets, to identify all the targets which this node will communicate with. Each entry in the target configuration file identifies the NID of a peer and all IP addresses which it can be reached by.The target are kept in a global list in the IB LND layer.When it comes time to send a message to a peer (kiblnd_send()), the LNET layer provides the NID of the peer to the IB LND. A kib_routes_t object is created, which will contain a list of kib_route_t objects, which is a local device/remote IP address pair. If no NID entry is found then the set of kib_route_t entries will depend on the number of devices in the logical network interface group (Refer to the above logic for more details).

Example One

Node A

3 HCA cards with the following information

1. lb0: 10.1.1.2
2. lb1: 10.1.1.3
3. lb2: 10.1.1.4

networks module parameter: options lnet networks="o2ib0(ib0,ib1,ib2)"

target configuration file will contain:

1. entry for local devices 10.1.1.2@o2ib0 10.1.1.2 10.1.1.3 10.1.1.4
2. entry for remote peer 10.1.1.5@o2ib0 10.1.1.5 10.1.1.6 10.1.1.7

Node B

3 HCA cards with the following information

1. lb0: 10.1.1.5
2. lb1: 10.1.1.6
3. lb2: 10.1.1.7

networks module parameter: options lnet networks="o2ib0(ib0,ib1,ib2)

target configuration file will contain:

1. entry for local devices 10.1.1.5@o2ib0 10.1.1.5 10.1.1.6 10.1.1.7
2. entry for remote peer 10.1.1.2@o2ib0 10.1.1.2 10.1.1.3 10.1.1.4

Node A --> sends 3 messages --> Node B

Three different routes to Node B are created, and represented by different peers. (NOTE: the connection via that route is not established until the route is picked)

1. Peer 1: NID: 10.1.1.5@o2ib0, Route: {Dev: 10.1.1.2, Remote: 10.1.1.5}
2. Peer 2: NID: 10.1.1.5@o2ib0, Route: {Dev: 10.1.1.3, Remote: 10.1.1.6}
3. Peer 3: NID: 10.1.1.5@o2ib0, Route: {Dev: 10.1.1.4, Remote: 10.1.1.7}

This means that message 1 will go over Route 1, message 2 over Route 2 and message 3 will go over Route 3

Example Two

Node A

3 HCA cards with the following information

1. lb0: 10.1.1.2
2. lb1: 10.1.1.3
3. lb2: 10.1.1.4

networks module parameter: options lnet networks="o2ib0(ib0,ib1,ib2)"

target configuration file will contain:

1. entry for local devices 10.1.1.2@o2ib0 10.1.1.2 10.1.1.3 10.1.1.4

Node B

3 HCA cards with the following information

1. lb0: 10.1.1.5
2. lb1: 10.1.1.6
3. lb2: 10.1.1.7

networks module parameter: options lnet networks="o2ib0(ib0,ib1,ib2)

target configuration file will contain:

1. entry for local devices 10.1.1.5@o2ib0 10.1.1.5 10.1.1.6 10.1.1.7
2. entry for remote peer 10.1.1.2@o2ib0 10.1.1.2 10.1.1.3 10.1.1.4

In this example the target file for Node A doesn't have any entries for Node B. Thus the routes which will be created will be as follows:

1. Peer 1: NID: 10.1.1.5@o2ib0, Route: {Dev: 10.1.1.2, Remote: 10.1.1.5}
2. Peer 2: NID: 10.1.1.5@o2ib0, Route: {Dev: 10.1.1.3, Remote: 10.1.1.5}
3. Peer 3: NID: 10.1.1.5@o2ib0, Route: {Dev: 10.1.1.4, Remote: 10.1.1.5}

Thus when three messages are sent from Node A to Node B, each message will go over a different local device, but will end up at the same remote device.

DLC Configuration Interface

Network Configuration

Network configuration is not new to the DLC interface and is already described in the Lustre manual

Target Configuration

Targets can be configured via YAML configuration file or through the lnetctl utility.

YAML Configuration

```
o2ibs:
    - nid: <nid>
      ips:
        - ip: <ip address 1>
        - ip: <ip address 2>
        - ip: <ip address N>
```

lnetctl commands

```
# add a target
lnetctl o2ibs add --nid <nid> --ips <ip1> <ip2> ... <ip N> # where N <= 16

# show targets for a specific network
lnetctl o2ibs show --net <net>

# import YAML configuration file
lnetctl import < o2ibs.yaml
```

Network Status

YAML Configuration

```
net_status:
    - net: <net>
```

lnetctl command

```
lnetctl net_status --net <net>
```

Configuration Sanity Check

lnetctl command

```
lnetctl o2ibs check --net <net>
```