10/29/2013 - Dynamic LNet Configuration User Guide

- 1 Introduction
  - 1.1 Overview
    - 1.1.1 LNET Utility
    - 1.1.2 LNET Configuration C language-based API Library (C-API)
    - 1.1.3 IML (Chroma)
    - 1.1.4 Startup Scripts/Shell Scripts
- 2 LNet C Configuration API Overview
- 3 General API Information
  - 3.1 API Common Input Parameters
  - 3.2 API Return Code
  - 3.3 API Out Parameters
    - 3.3.1 YAML Internal Representation
    - 3.3.2 Error Block
    - 3.3.3 Show Block
- 4 The LNET Configuration C-API
  - 4.1 Enabling and Disabling Routing API
    - 4.1.1 IOCTL to Kernel:
    - 4.1.2 Description:
      - 4.1.2.1 Enabling Routing
      - 4.1.2.2 Disabling Routing
      - 4.1.2.3 Enabling Routing on an already enabled node, or vice versa
    - 4.1.3 Return Value
  - 4.2 Adding Routes
    - 4.2.1 IOCTL to Kernel:
    - 4.2.2 Description:
    - 4.2.3 Return Value
  - 4.3 Deleting Routes
    - 4.3.1 IOCTL to Kernel
    - 4.3.2 Description
    - 4.3.3 Return Value
  - 4.4 Showing Routes
    - 4.4.1 IOCTL to Kernel
    - 4.4.2 Description
    - 4.4.3 Return Value
  - 4.5 Adding a Network Interface
    - 4.5.1 IOCTL to Kernel
    - 4.5.2 Description
    - 4.5.3 Return Value
  - 4.6 Deleting a Network Interface
    - 4.6.1 IOCTL to Kernel
    - 4.6.2 Description
    - 4.6.3 Return Value
  - 4.7 Showing Network Interfaces
    - 4.7.1 IOCTL to Kernel
    - 4.7.2 Description
    - 4.7.3 Return Value
  - 4.8 Adjusting Router Buffer Pools
    - 4.8.1 IOCTL to Kernel
    - 4.8.2 Description
    - 4.8.3 Return Value
  - 4.9 Showing Router Buffer Pools
    - 4.9.1 IOCTL to Kernel
    - 4.9.2 Description
    - 4.9.3 Return Value
  - 4.10 Showing LNET Traffic Statistics
    - 4.10.1 IOCTL to Kernel
    - 4.10.2 Description
    - 4.10.3 Return Value
  - 4.11 Adding/Deleting/Showing Parameters through a YAML Block
    - 4.11.1 IOCTL to Kernel
    - 4.11.2 Description

        - 4.11.3 Return Value
- 5 YAML Syntax
    - 5.1 Route Parameter
    - 5.2 Network Parameter
    - 5.3 Router Buffers Enable and Adjust Parameter
    - 5.4 Show Statistics
- 6 Command Line Syntax
    - 6.1 Adding Parameters
    - 6.2 Deleting Parameters
    - 6.3 Showing Parameters
- 7 Appendix A
- 8 Appendix B
    - 8.1 Command Line Parsing Infrastructure
    - 8.2 Infrastructure Job
    - 8.3 YAML Command Syntax Description

Introduction

Dynamic LNet Configuration (DLC) Project introduces, as the name implies, a more dynamic way of adding, deleting and showing LNet configuration. The DLC project manages the following configuration items:

- Enabling/Disabling routing
- Routes
- Networks
- Router Buffer Pools.

The DLC project introduces a C-API which is intended to be used to manage the above items. A Command Line Utility was added as well to provide an interface to the C-API. The Command Line Utility ends up calling the C-API introduced.

A YAML interface to the C-API was added as well to allow managing multiple items via YAML blocks. This provides an easier interface for IML. It also provides a way to configure multiple nodes by storing the configuration as a YAML file and then running it on multiple nodes.

DLC does not replace the ability to manage the above configuration items via module parameters. These two methods coexist.

Overview

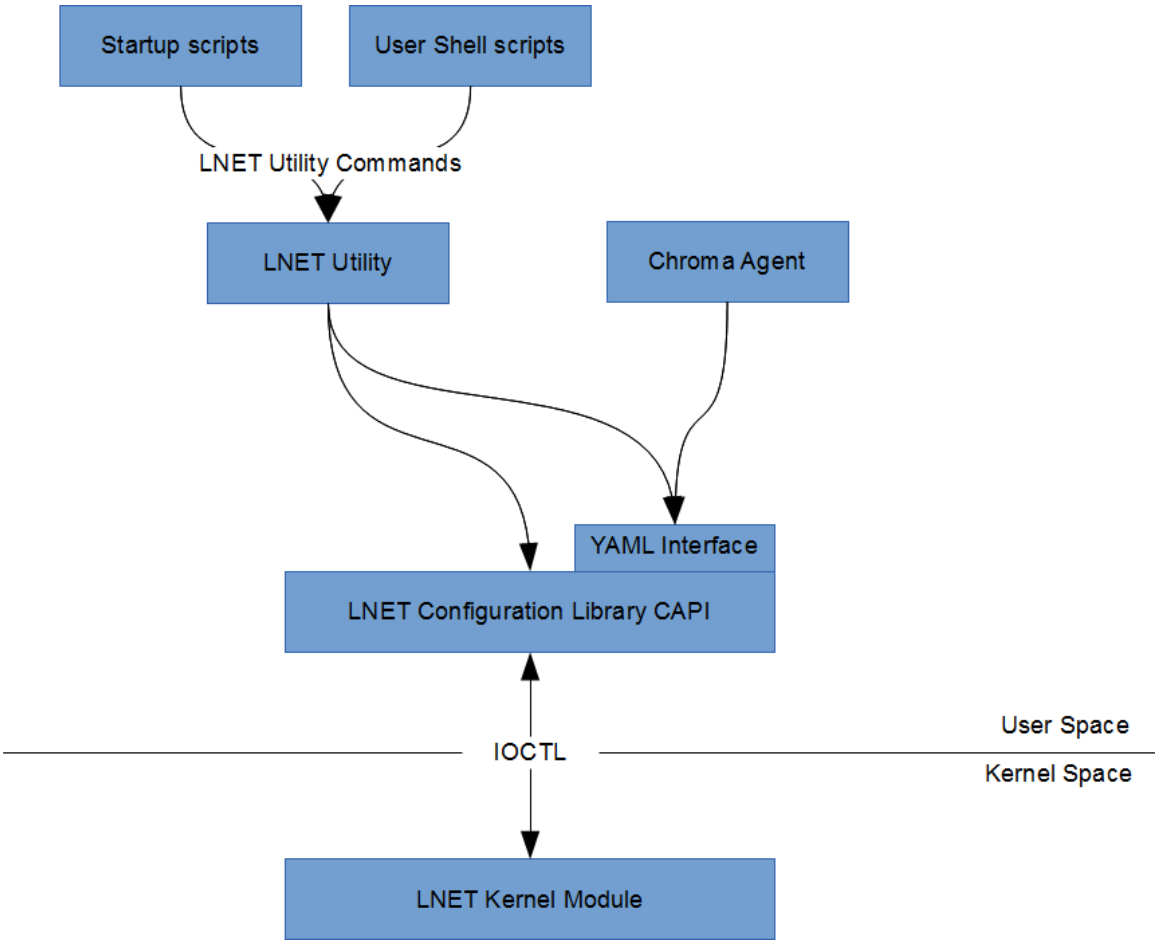


Figure 1: Block Diagram

The above diagram shows the high level logical blocks and their interaction.

LNET Utility

This is a utility that provides a command line interface to manage the supported configuration items. When configuration commands are matched

a corresponding action function is called with the parameters passed in the command line, which eventually will result in a call to the LNET Configuration Library C-API. The C-API will then do appropriate error checking and send an IOCTL to the kernel to do the actual configuration.

The LNET Utility is located in lnet/utils/lnetctl.c

LNET Configuration C language-based API Library (C-API)

The LNET Configuration Library C-API provides an abstraction layer to hide the Kernel IOCTLs and the structures used by the IOCTLs. This is a **userspace** API. The API provides a set of functions to manage the supported configuration items. It also provides a YAML interface. This interface is composed of three APIs that take a YAML file, parse it and end up calling the other APIs to manage the supported configuration items. The YAML block syntax is described later in the document. The result of the API is to call through an IOCTL into the Kernel, providing it with the parameters passed in by the user. The return code from the LNET Kernel Module is then encapsulated into a structural YAML representation hence forth referred to as cYAML, which is described later in the document.

The C-API is located in lnet/utils/lnetconfig

IML (Chroma)

This is not within the scope of the DLC project, but the LNET Configuration Library C-API, particularly the YAML Interface, was designed with IML in mind. IML's requirement is not to be exposed to the structure layout used for IOCTL communication with Kernel modules and not to spawn another process to configure LNET; thus the YAML interface is introduced. IML can represent what it needs to configure in a YAML block and then call the appropriate YAML Inteface API. Of course, since IML is written in Python there will need to be an intermediate layer that does the Python/C translation. There are multiple options to write this layer, one is using SWIG, which has been prototyped and proved that it works.

Startup Scripts/Shell Scripts

Since there is no easy way for the shell scripts to call into the C-API directly, shell scripts can call the LNET Utility which in turn calls the C-API.

In the rest of this document, the LNET Configuration Library C-API, the YAML syntax and the command line Utility are described in greater details.

LNet C Configuration API Overview

The DLC project introduces a C-API to configure LNET parameters. The LNet Parameters which are currently supported are:

1. Enabling/Disabling Routing
2. Adding/Deleting/Showing Routes
    a. network
    b. gateway
    c. hop
    d. priority
3. Adding/Deleting/Showing networks
    a. network
    b. Physical Interface
    c. peer timeout
    d. peer credits
    e. peer buffer credits
    f. credits
    g. CPU Partitions
4. Adjust buffer Pools
    a. tiny
    b. small
    c. large
5. Show various statistics
    a. Traffic statistics
    b. Peer credits
    c. peers
    d. Connection Queues

General API Information

API Common Input Parameters

All APIs take as input a sequence number. This is a number that's assigned by the caller of the API, and is returned in the YAML error return block. It is used to associate the request with the response. It is especially useful when configuring via the YAML Interface, since typically the YAML Interface is used to configure multiple items. In the return Error block, it is desired to know which items were configured properly and which were not configured properly. The sequence number achieves this purpose.

API Return Code

```
#define LUSTRE_CFG_RC_NO_ERR               0
#define LUSTRE_CFG_RC_BAD_PARAM             -1
#define LUSTRE_CFG_RC_MISSING_PARAM         -2
#define LUSTRE_CFG_RC_OUT_OF_RANGE_PARAM    -3
#define LUSTRE_CFG_RC_OUT_OF_MEM   -4
#define LUSTRE_CFG_RC_GENERIC_ERR   -5
```

API Out Parameters

YAML Internal Representation

Once a YAML block is parsed it needs to be stored structurally in order to facilitate passing it to different functions, querying it and printing it. Also it is required to be able to build this internal representation from data returned from the kernel and return it to the caller, which can query and print it. This structure representation is used for the Error and Show API Out parameters. For this YAML is internally represented via this structure:

```
typedef enum {
    EN_YAML_TYPE_FALSE = 0,
    EN_YAML_TYPE_TRUE,
    EN_YAML_TYPE_NULL,
    EN_YAML_TYPE_NUMBER,
    EN_YAML_TYPE_STRING,
    EN_YAML_TYPE_ARRAY,
    EN_YAML_TYPE_OBJECT
} cYAML_object_type_t;

typedef struct cYAML {
    /* next/prev allow you to walk array/object chains. */
    struct cYAML *next, *prev;
    /* An array or object item will have a child pointer pointing
       to a chain of the items in the array/object. */
    struct cYAML *child;
    /* The type of the item, as above. */
    cYAML_object_type_t type;
    /* The item's string, if type==EN_YAML_TYPE_STRING */
    char *valuestring;
    /* The item's number, if type==EN_YAML_TYPE_NUMBER */
    int valueint;
    /* The item's number, if type==EN_YAML_TYPE_NUMBER */
    double valuedouble;
    /* The item's name string, if this item is the child of,
       or is in the list of subitems of an object. */
    char *string;
    /* user data which might need to be tracked per object */
    void *user_data;
} cYAML;
```

Error Block

All APIs return a cYAML error block. This error block has the following format, when it's printed out:

```
error:
 <cmd>:
  <entity>:
   errno: <error number>
   descr: <error description>

Example:
    error:
        config:
            route:
                errno: -2
                descr: Missing mandatory parameter(s): network
```

Show Block

All Show APIs return a cYAML show block. This show block represents the information requested in YAML format. Each configuration item has its own YAML syntax. The YAML syntax of all supported configuration items is described later in this document. Below is an example of a show block:

```
net:
        nid: 192.168.206.130@tcp4
        status: up
        interfaces:
            intf-0: eth0
        tunables:
            peer_timeout: 10.000000
            peer_credits: 8.000000
            peer_buffer_credits: 30.000000
            credits: 40.000000
```

Refer to Appendix A for a description of the FSM used to build the parsed YAML.

The LNET Configuration C-API

Enabling and Disabling Routing API

```
/*
 * lustre_lnet_enable_routing
 *   Send down an IOCTL to enable or disable routing
 *
 *   enable - 1 to enable routing, 0 to disable routing
 *   seq_no - sequence number of the request
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_enable_routing(int enable,
                                      int seq_no,
                                      cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_ENABLE_RTR

Description:

Enabling Routing

The router buffer pools are allocated using the last set values. Internally the node is then flagged as a Router node. The node can be used to behave as a router from this point on.

Disabling Routing

The unused router buffer pools are freed. Buffers currently in use are not freed until they are returned to the unused list. Internally the node routing flag is turned off. Any subsequent messages not destined to this node are dropped.

Enabling Routing on an already enabled node, or vice versa

In both these cases the LNET Kernel module ignores this request.

Return Value

-ENOMEM: if there is no memory to allocate buffer pools

0: if success

Adding Routes

```
/*
 * lustre_lnet_config_route
 *   Send down an IOCTL to the kernel to configure the route
 *
 *   nw - network
 *   gw - gateway
 *   hops - number of hops passed down by the user
 *   prio - priority of the route
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_config_route(char *nw, char *gw,
                    int hops, int prio,
                    int seq_no,
                    cYAML **err_rc);
```

IOCTL to Kernel:

IOC_LIBCFS_ADD_ROUTE

Description:

The LNET Kernel module adds this route to the list of existing routes, if one doesn't already exist. If hop parameter is not specified (IE: -1) then the hop count is set to 1. If the priority parameter is not specified (IE: -1) then the priority is set to 0. All routes with the same hop and priority are used in round robin. Routes with lower number of hops and/or higher priority are preferred. 0 is the highest priority.

If a route already exists the request to add the same route is ignored.

Return Value

-EINVAL: if the network of the route is local

-ENOMEM: if there is no memory.

-EHOSTUNREACH: if the host is not on a local network

0: if success

Deleting Routes

```
/*
 * lustre_lnet_del_route
 *   Send down an IOCTL to the kernel to delete a route
 *
 *   nw - network
 *   gw - gateway
 */
extern int lustre_lnet_del_route(char *nw, char *gw,
                 int seq_no,
                 cYAML **err_rc);
```

IOCTL to Kernel

IOC_LIBCFS_DEL_ROUTE

Description

The Kernel will remove the route which matches the network and gateway passed in. If no route matches, then the operation fails with an appropriate failure code.

If a route is not there and a delete request is received then the request fails with error number -2, No such file or directory

Return Value

-ENOENT: if the entry being deleted doesn't exist

0: if the route delete is successful

Showing Routes

```
/*
 * lustre_lnet_show_route
 *   Send down an IOCTL to the kernel to show routes
 *   This function will get one route at a time and filter according to
 *   provided parameters. If no filter is provided then it will dump all
 *   routes that are in the system.
 *
 *   nw - network.  Optional.  Used to filter output
 *   gw - gateway. Optional. Used to filter ouptut
 *   hops - number of hops passed down by the user
 *          Optional.  Used to filter output.
 *   prio - priority of the route.  Optional.  Used to filter output.
 *   detail - flag to indicate whether detail output is required
 *   show_rc - [OUT] The show output in YAML.  Must be freed by caller.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_show_route(char *nw, char *gw,
                  int hops, int prio, int detail,
                  int seq_no,
                  cYAML **show_rc,
                  cYAML **err_rc);
```

IOCTL to Kernel

IOC_LIBCFS_GET_ROUTE

Description

The routes are fetched from the kernel one by one and packed in a cYAML block, after filtering according to the parameters passed in. The cYAML block is then returned to the caller of the API.

An example with the detail parameter set to 1

```
route:
        net: tcp5
        gateway: 192.168.205.130@tcp
        hop: 1.000000
        priority: 0.000000
        state: up
```

An Example with the detail parameter set to 0

```
route:
        net: tcp5
        gateway: 192.168.205.130@tcp
```

Return Value

-ENOMEM: if no memory

0: If successful

Adding a Network Interface

```
/*
 * lustre_lnet_config_net
 *   Send down an IOCTL to configure a network.
 *
 *   net - the network name
 *   intf - the interface of the network of the form net_name(intf)
 *   peer_to - peer timeout
 *   peer_cr - peer credit
 *   peer_buf_cr - peer buffer credits
 *       - the above are LND tunable parameters and are optional
 *   credits - network interface credits
 *   smp - cpu affinity
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_config_net(char *net,
                  char *intf,
                  int peer_to,
                  int peer_cr,
                  int peer_buf_cr,
                  int credits,
                  char *smp,
                  int seq_no,
                  cYAML **err_rc);
```

IOCTL to Kernel

IOC_LIBCFS_ADD_NET

Description

A new network is added and initialized. This has the same effect as configuring a network from the module parameters. The API allows the specification of network parameters such as the peer timeout, peer credits, peer buffer credits and credits. The CPU affinity of the network interface being added can also be specified. These parameters become network specific under DLC, as opposed to being per LND as it was previously.

If an already existing network is added the request is ignored.

Return Value

-EINVAL: if the network passed in is not recognized.

-ENOMEM: if no memory

0: success

Deleting a Network Interface

```
/*
 * lustre_lnet_del_net
 *   Send down an IOCTL to delete a network.
 *
 *   nw - network to delete.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_del_net(char *nw,
                  int seq_no,
                  cYAML **err_rc);
```

IOCTL to Kernel

IOC_LIBCFS_DEL_NET

Description

The network interface specified is deleted. All resources associated with this network interface are freed. All routes going over that Network Interface are cleaned up.

If a non existent network is deleted then the call return -EINVAL.

Return Value

-EINVAL: if the request reference a non-existent network.

0: if success.

Showing Network Interfaces

```
/*
 * lustre_lnet_show_net
 *   Send down an IOCTL to show networks.
 *   This function will use the nw paramter to filter the output.  If it's
 *   not provided then all networks are listed.
 *
 *   nw - network to show.  Optional.  Used to filter output.
 *   detail - flag to indicate if we require detail output.
 *   show_rc - [OUT] The show output in YAML.  Must be freed by caller.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_show_net(char *nw, int detail,
                int seq_no,
                cYAML **show_rc,
                cYAML **err_rc);
```

IOCTL to Kernel

IOC_LIBCFS_GET_NET

Description

The network interfaces are queried one at a time from the kernel and packed in a cYAML block, after filtering on the network (EX: tcp). If the detail field is set to 1, then the tunable section of the show block is included in the return.

An example of the detailed output:

```
net:
        nid: 192.168.206.130@tcp4
        status: up
        interfaces:
            intf-0: eth0
        tunables:
            peer_timeout: 10.000000
            peer_credits: 8.000000
            peer_buffer_credits: 30.000000
            credits: 40.000000
```

A none detailed show block is:

```
net:
        nid: 192.168.206.130@tcp4
        status: up
        interfaces:
            intf-0: eth0
```

Return Value

-ENOMEM: if no memory to allocate the error or show blocks.

0: if success.

Adjusting Router Buffer Pools

```
/*
 * lustre_lnet_config_buf
 *   Send down an IOCTL to configure buffer sizes.  A value of 0 means
 *   default that particular buffer to default size. A value of -1 means
 *   leave the value of the buffer unchanged.
 *
 *   tiny - tiny buffers
 *   small - small buffers
 *   large - large buffers.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_config_buf(int tiny,
                  int small,
                  int large,
                  int seq_no,
                  cYAML **err_rc);
```

IOCTL to Kernel

IOC_LIBCFS_ADD_BUF

Description

This API is used to configure the tiny, small and large router buffers dynamically. These buffers are used to buffer messages which are being routed to other nodes. The minimum value of these buffers per CPT are:

```
#define LNET_NRB_TINY_MIN     512
#define LNET_NRB_SMALL_MIN    4096
#define LNET_NRB_LARGE_MIN    256
```

The default values of these buffers are:

```
#define LNET_NRB_TINY         (LNET_NRB_TINY_MIN * 4)
#define LNET_NRB_SMALL        (LNET_NRB_SMALL_MIN * 4)
#define LNET_NRB_LARGE        (LNET_NRB_LARGE_MIN * 4)
```

These default value is divided evenly across all CPTs. However, each CPT can only go as low as the minimum.

Multiple calls to this API with the same values has no effect

Return Value

-ENOMEM: if there is no memory to allocate buffer pools

0: if success

Showing Router Buffer Pools

```
/*
 * lustre_lnet_config_buf
 *   Send down an IOCTL to dump buffers.
 *   This function is used to dump buffers for all CPU partitions.
 *
 *   show_rc - [OUT] The show output in YAML.  Must be freed by caller.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_lnet_show_buf(int seq_no,
                cYAML **show_rc,
                cYAML **err_rc);
```

IOCTL to Kernel

IOC_LIBCFS_GET_BUF

Description

This API returns a cYAML block describing the values of each of the following per CPT:

1. The number of pages per buffer. This is a constant.
2. The number of allocated buffers. This is a constant
3. The number of buffer credits . This is a real-time value of the number of buffer credits currently available. If this value is negative, that indicates the number of queued messages.
4. The lowest number of credits ever reached in the system. This is historical data.

An example Yaml block

```
pools[0]:
        tiny:
            npages: 0
            nbuffers: 2048
            credits: 2048
            mincredits: 2048
        small:
            npages: 1
            nbuffers: 16384
            credits: 16384
            mincredits: 16384
        large:
            npages: 256
            nbuffers: 1024
            credits: 1024
            mincredits: 1024
```

Return Value

-ENOMEM: if no memory to allocate the show or error block

0: if success

Showing LNET Traffic Statistics

```
/*
 * lustre_lnet_show_stats
 *   Shows internal LNET statistics.  This is useful to display the
 *   current LNET activity, such as number of messages route, etc
 *
 *     seq_no - sequence number of the command
 *     show_rc - YAML structure of the resultant show
 *     err_rc - YAML strucutre of the resultant return code.
 */
extern int lustre_lnet_show_stats(int seq_no, cYAML **show_rc,
                  cYAML **err_rc);
```

IOCTL to Kernel

IOC_LIBCFS_GET_LNET_STATS

Description

This API returns a cYAML block describing the LNET traffic statistics. The statistics include the following

1. Number of messages allocated
2. Maximum number of messages in the system
3. Errors allocating or sending messages
4. Cumulative number of messages sent
5. Cumulative number of messages received
6. Cumulative number of messages routed
7. Cumulative number of messages dropped
8. Cumulative number of bytes sent
9. Cumulative number of bytes received
10. Cumulative number of bytes routed
11. Cumulative number of bytes dropped

An example Yaml block

```
statistics:
        msgs_alloc: 0
        msgs_max: 0
        errors: 0
        send_count: 0
        recv_count: 0
        route_count: 0
        drop_count: 0
        send_length: 0
        recv_length: 0
        route_length: 0
        drop_length: 0
```

Return Value

-ENOMEM: if no memory to allocate the show or error block

0: if success

Adding/Deleting/Showing Parameters through a YAML Block

```
/*
 * lustre_yaml_config
 *   Parses the provided YAML file and then calls the specific APIs
 *   to configure the entities identified in the file
 *
 *   f - YAML file
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_yaml_config(char *f, cYAML **err_rc);

/*
 * lustre_yaml_del
 *   Parses the provided YAML file and then calls the specific APIs
 *   to delete the entities identified in the file
 *
 *   f - YAML file
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_yaml_del(char *f, cYAML **err_rc);

/*
 * lustre_yaml_show
 *   Parses the provided YAML file and then calls the specific APIs
 *   to show the entities identified in the file
 *
 *   f - YAML file
 *   show_rc - [OUT] The show output in YAML.  Must be freed by caller.
 *   err_rc - [OUT] cYAML tree describing the error. Freed by caller
 */
extern int lustre_yaml_show(char *f,
                cYAML **show_rc,
                cYAML **err_rc);
```

IOCTL to Kernel

Depends on the entity configured

Description

These APIs add/remove/show the parameters specified in the YAML file respectively. The entities don't have to be uniform. Multiple different entities can be added/removed/showed in one YAML block.

The entities can be divided by type (IE: net, route, buffer) and under each type would b a list of all entities of this type, as shown below.

An example YAML block

```
net:
 - nid: 192.168.206.132@tcp
   status: up
   interfaces:
       intf-0: eth4
   tunables:
    peer_timeout: 180
    peer_credits: 8
    peer_buffer_credits: 0
    credits: 256
    SMP: "[0]"
route:
   - net: tcp6
     gateway: 192.168.29.1@tcp
     hop: 4
     detail: 1
     seq_no: 3
   - net: tcp7
     gateway: 192.168.28.1@tcp
     hop: 9
     detail: 1
     seq_no: 4
buffer:
   - tiny: 1024
     small: 2048
     large: 4096
...
```

Return Value

Return value will correspond to the return value of the API that will be called to operate on the configuration item, as described in previous sections

YAML Syntax

Using the YAML APIs described above the supported configuration items can be managed. YAML is used to describe the configuration items that need to be operated on, and the API determines the actual operation: add, delete or show.

Route Parameter

```
route:
 - net: <network.  Ex: tcp or o2ib>
   seq_no: <integer.  Optional.  User generated, and is passed back in the YAML error
block>
   gateway: < nid of the gateway in the form <ip>@<net>.  Ex: 192.168.28.1@tcp>
   hop: <an integer between 1 and 255>
   detail: <This is only applicable for show command.  1 - output detailed info.  0 -
basic output>
```

Network Parameter

```
net:
   - nid: <network.  Ex: tcp or o2ib>
     seq_no: <integer.  Optional.  User generated, and is passed back in the YAML
error block>
  interfaces:
       intf-0: <physical interface>
     detail: <This is only applicable for show command.  1 - output detailed info.  0
- basic output>
     tunables:
        peer_timeout: <Integer. Timeout before consider a peer dead>
        peer_credits: <Integer. Transmit credits for a peer>
        peer_buffer_credits: <Integer. Credits available for receiving messages>
        credits: <Integer.  Network Interface credits>
        SMP: <An array of integers of the form: "[x,y,...]", where each integer
represents the CPT to associate the network interface with>
```

Router Buffers Enable and Adjust Parameter

```
buffer:
 - tiny: <Integer. Tiny buffers>
   small: <Integer. Small buffers>
   large: <Integer. Tiny buffers>
   enable: <0 - disable routing.  1 - enable routing>
   seq_no: <Integer.  Optional.  User generated, and is passed back in the YAML error
block>
```

Show Statistics

```
statistics:
 seq_no: <Integer. Optional.  User generated, and is passed back in the YAML error
block>
```

Command Line Syntax

Dynamic LNet Configuration introduces a Command Line Utility (lcmd) that can be used to configure LNET parameters. Below is a list of all the commands supported by the utility.

Adding Parameters

```
# Adds a route
add route --net <network> --gateway <nid> [--hop <integer>] [--priority <integer>]
# Adds a network
add net --net <network> --if <interface> [--peer_timeout <integer>] [--peer_credits
<integer>] [--peer_buffer_credits <integer>] [--credits <integer>] [--SMP <cpu
affinity list>]
# Adjusts the router buffers.  If none of the parameters are specified, the operation
is no-op.
add buffer [--tiny <integer>] [--small <integer>] [--large <integer>]
# Enables Routing
add routing
# gives a YAML file of entities to add.
add file <yaml file name>
```

Deleting Parameters

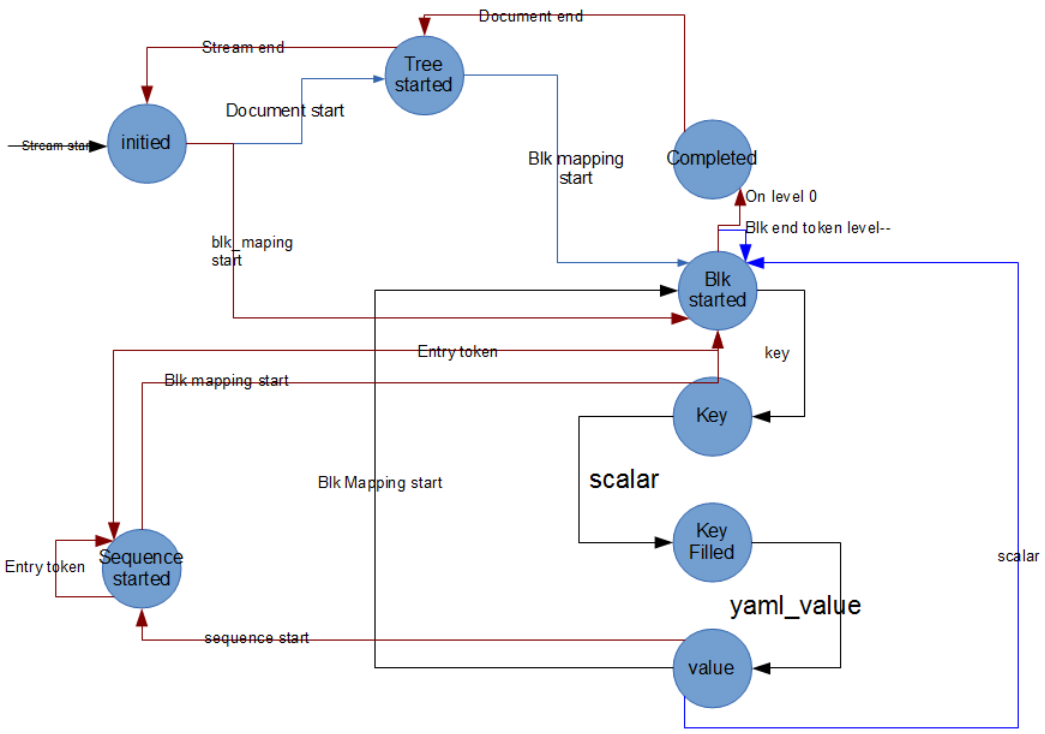
```
# deletes a route
del route --net <network> --gateway <nid>
# del a network
del net --net <network>
# Returns all router buffers to the system defaults
del buffer
# Disables Routing
del routing
# gives a YAML file of entities to be deleted.
del file <yaml file name>
```

Showing Parameters

```
# shows a set of routes.  The results are filtered by the parameters passed in.  If
the "detail" keyword is set then a more detailed output results.
show route [--net <network>] [--gateway <nid>] [detail]
# show a set of networks.  The results are filtered by the parameters passed in.  If
the "detail" keyword is set then a more detailed output results.
show net [--net <network>] [detail]
# Shows buffers for all CPTs
show buffer
# Shows the LNet traffic statistics
show statistics
# gives a YAML file of entities to be showed.
show file <yaml file name>
```

Appendix A

libyaml is used to parse YAML files. This is a stream parser, as it provides events of the parsed items. These events are then used to build an internal representation of the YAML block. The below FSM describes how the internal representation is built. The FSM doesn't support all YAML constructs, it only supports the supported constructs used to represent DLC configuration.



Appendix B

Initially I developed a new CLI infrastructure to be used. Eventually, this infrastructure was removed in favor of using the existing infrastructure. As a matter of historical record, I keep the description of this infrastructure in the appendix, and note that I think that this infrastructure is superior to the existing one, as it allows ease of expansion and addition of new CLI tools, without having to create a different binary. In fact you can have one binary, which can be fed different libraries and YAML command description files. The help system is also well integrated in the tool, so as we do not need separate man files to describe each command, which can easily fall out of sync with updates to the tool.

Command Line Parsing Infrastructure

The Command Line Parsing Infrastructure presents a way to define the supported commands in YAML format. This way the infrastructure can be reused to parse any command line syntax.

The infrastructure requires two main items

1. A YAML file describing the command syntax
2. A library that defines the action functions to be called when a command is matched.

The action function prototype and function parameters:

```
struct kvl_s {
        struct kvl_s *next;
        char *key;
        char *value;
};

typedef int (*cmd_cb_t)(struct kvl_s *);
```

The action function gets called with a linked list of key/value pairs. These are all the command line parameters.

As an example:

```
add route --net tcp --gateway 10.10.10.2@tcp1
```

This will result in a linked list as follows:

```
[[--net, tcp], [--gateway, 10.10.10.2@tcp1]]
```

This linked list will then be passed by the infrastructure to the action function for the "add route" command.

Infrastructure Job

1. Parse the YAML file describing the command syntax and store it in an internal cYAML representation
2. Match commands as they are typed in
3. Form a linked list of Key/Value pairs parsed from the command line
4. Call the action function with the Key/Value pair linked list once the command is matched properly
5. Detect and report any syntax errors

YAML Command Syntax Description

The command syntax can be represented in the following way using YAML. Comments are C-style

```
--- /* start of document */
<command name>: /* this is the primary command name */
   [lnet_cmd_help:  <help string>] /* lnet_cmd_help is a reserved keyword to describe
the command's help string */
   [lnet_cmd_value: /* tells the parser that this key requires a value */
        lnet_cmd_help: <value help> /* help on the format of the value if necessary */
        [lnet_cmd_back: <integer> ]] /* number of levels to go up if you want to
continue matching */
   [<sub command>: /* This is a sub command*/
      lnet_cmd_help: <help string>
      lnet_cmd_callback: <function to call if this command is matched>
      lnet_cmd_value: /* tells the parser that this key requires a value */
          lnet_cmd_help: <value help> /* help on the format of the value if necessary
*/
          lnet_cmd_back: <integer> ]/* number of levels to go up if you want to
continue matching */
... /* end of document */

As an example, below is the command syntax for adding a route:

    add:
        lnet_cmd_help: Dynamically configure LNET parameters
        route:
            lnet_cmd_help: Add or change an LNET route
            lnet_cmd_callback: lnet_config_route
            --net:
                lnet_cmd_help: route network
                lnet_cmd_value:
                    lnet_cmd_help: net name ex: tcp0
                    lnet_cmd_back: 2
            --gateway:
                lnet_cmd_help: route gateway
                lnet_cmd_value:
                    lnet_cmd_help: gateway address
                    lnet_cmd_back: 2
            --hop:
                lnet_cmd_help: integers number of hops
                lnet_cmd_value:
                    lnet_cmd_help: a between 1 to 255
                    lnet_cmd_back: 2
            --priority:
                lnet_cmd_help: integers number of hops
                lnet_cmd_value:
                    lnet_cmd_help: a value between 0 and 2^32
                    lnet_cmd_back: 2
```