
# Intel® Dynamic LNet Configuration

Amir Shehata

8 April 2014

# Dynamic LNet Configuration

Overview

- Purpose
- What can it do?
- What doesn't it do?
- Parameter Configuration
- C Configuration API

# Purpose

- Dynamically modifying LNet configuration capability is being built into the LNet module and will be landed in 2.7
- DLC aims at easing the process of fine tuning LNet without having to restart the LNet Kernel modules.  IE Configuration parameters are changed on a fully running system.
  - Streamlines setting and optimizing LNet parameters
- DLC makes configuration of key LNet parameters easier and more flexible

# What can it do?

- Adding/Deleting networks
- Adding/Deleting routes
- Configuring router buffer pools
- Enabling/Disabling routing.
- Showing routing information
- Importing/exporting configuration in YAML format

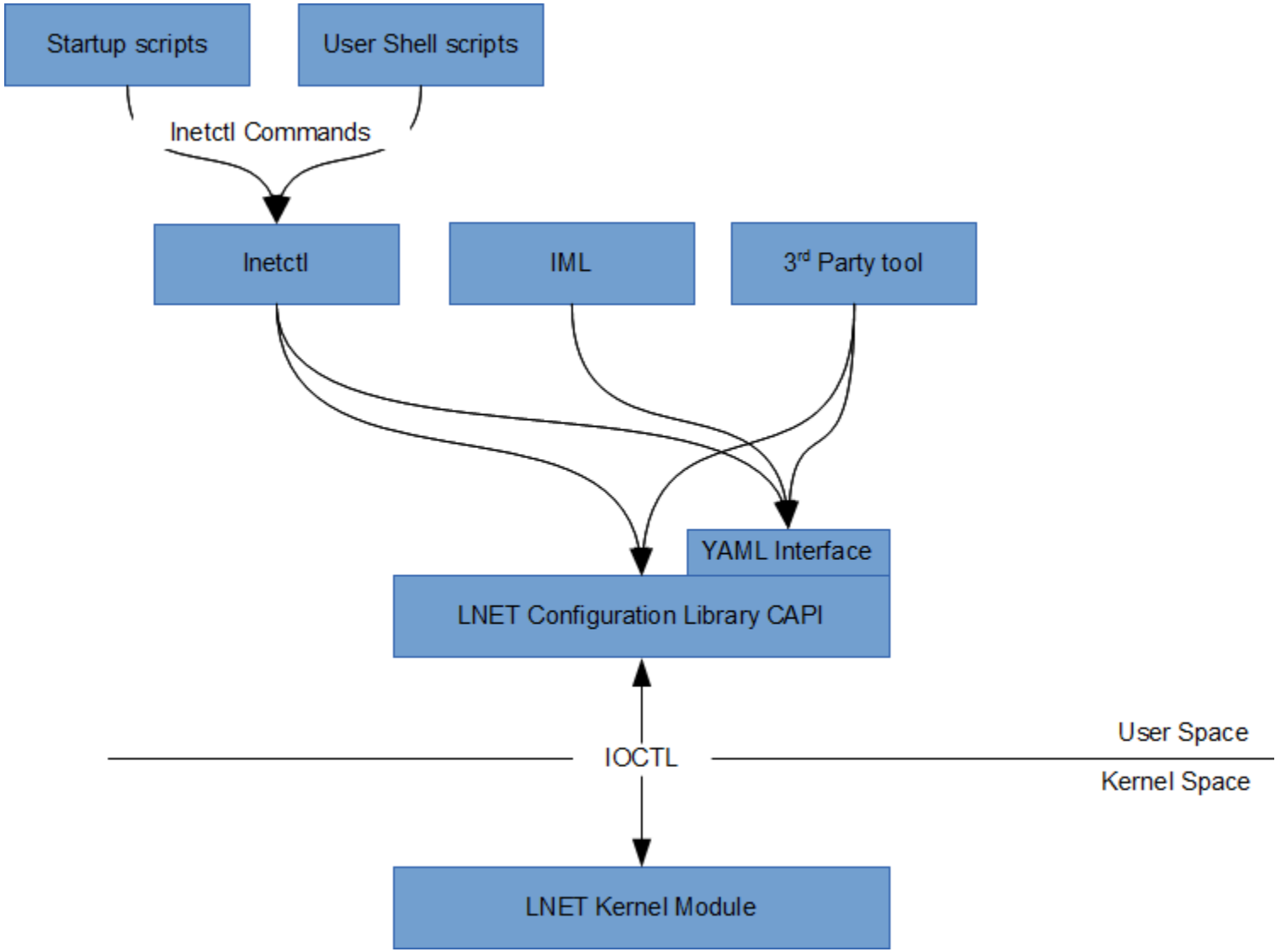
# What doesn't it do?

- For the first version of DLC, the most used parameters were picked to be configured dynamically.
- Currently DLC doesn't configure some of the LND parameters
  - Ex: map_on_demand
- Other examples of parameters not currently supported are:
  - Check_routers_before_use
  - avoid_asym_router_failure
  - dead_router_check_interval

# Parameter Configuration

- DLC Provides two ways of configuring LNet parameters
  - Via a Command Line tool, lnetctl
  - Via a C API

# Block Diagram



# Adding Networks

**From lnetctl:**

net add: add a network

--net: net name (ex tcp0)
--if: physical interface (ex eth0)
--peer-timeout: time to wait before declaring a peer dead
--peer-credits: define the max number of inflight messages
--peer-buffer-credits: the number of buffer credits per peer
--credits: Network Interface credits
--cpts: CPU Partitions configured net uses

Ex:

lnetctl net add –net tcp0 –if eth0 –peer-timeout 180 –peer-credits 8 --credits 1024

# Removing Networks

**From Inetctl:**

net del: delete a network

--net: net name (ex tcp0)

Ex:

Inetctl net del –net tcp0

# Showing Networks

**From Inetctl:**

net show: show networks

--net: net name (ex tcp0) to filter on

--detail: display detailed output per network

Ex:

# show all the networks

Inetctl net show

# show all the networks in detail

Inetctl net show –detail

# show a specific network

Inetctl net show –net tcp0

#show a specific network in detail

Inetctl net show –net tcp 0 –detail

# Show network sample output

- All show output is in YAML format

```
net:
    - nid: 0@lo
      status: up
      tunables:
          peer_timeout: 0
          peer_credits: 0
          peer_buffer_credits: 0
          credits: 0
    - nid: 192.168.205.130@tcp1
      status: up
      interfaces:
          0: eth3
      tunables:
          peer_timeout: 180
          peer_credits: 8
          peer_buffer_credits: 0
          credits: 256
```

# Adding Routes

**From lnetctl:**

route add: add a route

--net: net name (ex tcp0)

--gateway: gateway nid (ex 10.1.1.2@tcp)

--hop: number to final destination (1 < hops < 255)

--priority: priority of route (0 - highest prio

Ex:

lnetctl route add --net tcp0 --gateway 10.1.1.2@tcp1 --hop 1--priorit
0

# Removing Routes

**From lnetctl:**

route del: delete a route

--net: net name (ex tcp0)

--gateway: gateway nid (ex 10.1.1.2@tcp)

Ex:

lnetctl route del –net tcp0 –gateway 10.1.1.2@tcp1

# Showing routes

**From lnetctl:**

route show: show routes

--net: net name (ex tcp0) to filter on

--gateway: gateway nid (ex 10.1.1.2@tcp) to filter on

--hop: number to final destination (1 < hops < 255) to filter on

--priority: priority of route (0 - highest prio to filter on

--detail: display detailed output per route

Ex:

lnetctl route show –net tcp0

# Show route sample output - detailed

- All show output is in YAML format

```
route:
    - net: tcp2
      gateway: 192.168.206.133@tcp
      hop: 1
      priority: 0
      state: up
```

# Configuring Router Buffer Pools

- Configuring router buffer pools while routing is disabled, stores the configured values, which would take effect when routing is enabled.
- Disabling and enabling routing doesn’t reset router buffer pools values.
- Configuring router buffer pools while routing is enabled takes effect immediately
- Router buffer pool sizes adhere to min and max values.

# Configuring Router Buffer Pools

## From Inetctl

set tiny_buffers: set tiny routing buffers
VALUE must be greater than 0

set small_buffers: set small routing buffers
VALUE must be greater than 0

set large_buffers: set large routing buffers
VALUE must be greater than 0

Ex

set tiny_buffers 1024

# Enabling/Disabling Routing

- Enabling and disabling the routing feature on a node can be done dynamically

**From Inetctl**

set routing: enable/disable routing

0 - disable routing

1 - enable routing

Ex:

#enable routing

set routing 1

#disable routing

set routing 0

# Showing routing information

- All show output is in YAML format

```
routing:
    - cpt[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
    - enable: 1
```

# Importing YAML configuration

- It's possible to import a file describing LNet configuration in YAML format

**From lnetctl**

import FILE
import < FILE : import a file
    --add: add configuration
    --del: delete configuration
    --show: show configuration
    --help: display this help
If no command option is given then --add is assumed by default

Ex:
import < config.yaml # use config.yaml to add LNet configuration
import –del < config.yaml # use config.yaml to delete LNet configuration
import –show < config.yaml # use config.yaml to show LNet configuration

# Exporting YAML configuration

- It is possible to export LNet configuration in YAML format

**From Inetctl**

export FILE

export > FILE : export configuration

--help: display this help

Ex:

export > config.yaml

# YAML input/output example

```
---
net:
    - net: tcp3
      status: up
      interfaces:
         0: eth4
      tunables:
          peer_timeout: 180
          peer_credits: 8
          peer_buffer_credits: 0
          credits: 256
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
...
```

# C Configuration API

- Dynamic LNet Configuration introduces a C-library which can be used to configure LNet parameters.
- The library introduces two types of APIs
  - APIs to configure specific parameters
  - APIs which accept YAML input to configure a set of parameters.
- The C-library is the underlying infrastructure used by lnetctl.
- All APIs have an out parameter which is a YAML error block describing the errors
- Show APIs have an out parameter which is a YAML show block describing the show output
- YAML blocks can be manipulated and printed to a file stream.
- The C-library can be used in interpreted languages such as Python, which is useful when writing configuration scripts or possibly a front end for configuring LNet.

# C Configuration API – Python example

- Created a SWIG wrapper around the C Library
- This allows the C-API to be called from python scripts.

Ex: Configuring a network from a python script

```
import lustreconfigapi
rc, yaml_err = lustreconfigapi.lustre_lnet_config_net("tcp", "eth0",
            180, # peer timeout
             8, # peer credits
             0, # peer buffer credits
             256, # network interface credits
             None, # CPU affinity assignment
             -1) # sequence number used to identify the transaction

# print the YAML error to file
lustreconfigapi.cYAML_print_tree2file(f, yaml_err, 0)
```

# C Configuration API – YAML input

- The API provides a way to Add, Delete and show configuration parameters via YAML input
- The YAML input is as described above.

# C Configuration API - Note

- Note the YAML output from the show APIs can be fed directly into the APIs which take YAML input to add, delete or show LNet Parameters.
- This allows for scenarios such as
  - querying a node for its configuration
  - storing the YAML output in a file
  - Feeding that YAML file to configure the node on restart.
  - Or possibly to configure other nodes.

# Questions?