CC ONF DRIVER - HIGH LEVEL DESIGN

Data, threads, sockets

Static Behavior:

Global struct: cc\_of\_global\_t

Enum: ofdrv\_type //SWITCH or CONTROLLER

Enum: ofdev\_type //SERVER or CLIENT

Container: with cc\_ofdev\_info\_t nodes //hash table

Container: with cc\_ofchannel\_info\_t nodes //hash table

Callbacks: NET\_SVCS[MAX\_OF\_DRV\_TYPE][MAX\_L4\_TYPE]

Container: rw-threads //circular buffer

struct: cc\_ofdev\_info\_t

key: controller\_ip\_address + switch\_ip\_address + L4\_proto

number: controller\_L4\_port

Container: with cc\_ofrw\_info\_t //hash table

Int: main\_sockfd //only for controller

Struct: cc\_ofrw\_info\_t

Key: rw\_sockfd

Enum: State

Thread

Pipe to write to thread

Pipe to read from thread

Synchronization variables

Stats

Thread-specific data:

Each thread will need to maintain:

1. pipe\_read\_from\_lib
2. pipe\_write\_to\_lib
3. its rw\_sockfd
4. its dp-id and aux-id

Global:

Type – S/C

NET\_SVCS – full set of net service callbacks

Table - ofdevice instances

Table – ofchannels

Table – ofrw\_info

Table – ofdevice instances:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Key | | | C-L4-port | List: rw\_socket | Main Socket | Other |
| C\_ip\_addr | S\_ip\_addr | L4-proto |
|  |  |  |  | <?> |  |  |
|  |  |  |  |  |  |  |

Table – rw\_info:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Key | State | Thread | Async Queue | WR pipe | Ptr to ofdevice | Ptr to ofchannel | Other |
| Rw-socket |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Table – ofchannels:

|  |  |  |
| --- | --- | --- |
| Key | | Rw-socket |
| DP-ID | AUX-ID |
|  |  |  |
|  |  |  |

Dynamic Behavior:

Global data:

The Global data is initialized when application (controller or switch) registers with the library.

NET\_SVCS global table is statically generated by the init code – the callbacks for each ofdrv\_type and each layer-4 type (for example: client tcp, server tcp, …) are registered here.

3 hash tables maintain all necessary data.

Ofdevice\_instance:

Controller: An ofdevice\_instance is created for every IP address the controller wants to listen on for every Layer4 protocol

Switch: An ofdevice\_instance is created for every pair of <controller IP, switch IP> per layer4 protocol.

Every ofdevice\_instance maintains the port number for the controller.

Every ofdevice\_instance maintains a collection of ofrw socket entries in a hash table.

The main\_socket is applicable only for the controller, as this is the socket it will listen on.

Ofrw\_info:

An ofrw\_info entry is created every time a new connection is established.

Server: When accept returns a new socket fd for read/write upon establishing a new session, a new thead is started to rd/wr on this socket. This thread information is also stored in a newly created ofrw entry in global struct. The state of the connection is UP when this new socketfd is created by accept.

Client: When open creates a new socket and binds it to a port, a new thread is created. The state of this connection at this point is DOWN. Once connect returns without errors, the state of the connection is changed to UP making it ready for read/write.

Ofchannel\_info:

A global container with simple mapping between channel key (dp-id/aux-id) and its corresponding rw-socket. This rw-socket is the key to

Threads:

All the data listed so far – global, ofdev\_instance, ofchannel and ofrw\_info– is global for the threads. Synchronization variables will be used to read/write into these data structures.

The main library thread (ie, the application thread that links to library) will read/write to the openflow channels using pipes. Individual openflow channels are managed by the individual threads. Therefore, anytime data needs to be sent out on a specific openflow channel, the main thread writes to the specific pipe.

The data that is received on sockets on individual openflow channel threads is written into pipes that are read by the main thread.

TBD: A separate thread may be needed to manage the receiving of data from pipes.

The channel threads will be blocked on select unless:

1. data is received on the openflow channel: this data will be written into the pipe from which the main thread will read.
2. Data is received on the send pipe: this data will be written to the socket.

Pipes between threads for moving the read/write data:

Main thread

Rw thread #1

Rw thread #2

Rw thread #n

Pipe receive thread

Questions:

1. Can and does a specific controller communicate with switches over multiple IP addresses?
2. Backward compatibility with 1.0
3. Scope of this library – any additional functionality that needs to be absorbed
4. Which controller for integration
5. Performance, load balancing