# Tutorial on North Bound API Usage

# Contents

# 1. Introduction:

Controller architecture typically consist of "OF Controller Infrastructure" and multiple applications. "OF Controller infrastructure" is expected to provide a standard interface for applications to take advantage of OF infrastructure to program Data-Path entities. This interface is called "North Bound API", NBAPI in short.

North Bound Interface layer is a layer between controller and applications. It provides interface to applications to work on controller. This layer provides set of North Bound APIs to implement different applications on top of controller. These APIs also hides OpenFlow protocol functionality to the applications. Applications have impression that control and data plan are on the same device. NB interface layer has different types of North Bound APIs for applications to call based on requirement.

# 2. Scope

The Readers for this document are:-

Open Flow Application Software Developers.

# 3. Acronyms and Abbreviations

- DPRM – Database Resource Manager.
- NBAPI - North Bound API.

# 4. Types of North Bound APIs:

NBAPIs are divided into multiple types.

1. **Data path Resource Management(DPRM) APIs:**

   These APIs can be used by external applications such as topology discovery applications.

   These discovery applications inform the OF controller infrastructure layer via these NBAPI functions. DPRM APIs are again sub divided into the following types.

   - Switch Management APIs.
   - Domain Management APIs.
   - Datapath Management APIs.
   - Controller Management APIs.

2. **Openflow messaging APIs:**

   These APIs can be used by control plane and service plane applications. Applications can use this API to receive OF notifications generated by Openflow

switches, send commands to Openflow switches.

These APIs can be used to create flows, groups, and meters, inquire switch capabilities with respect to tables, ports and queues and register to get packet-in, port status and other asynchronous events generated by openflow switches. These APIs are again subdivided into the following types.

- Asynchronous Message (Notification) APIs.
- Controller-to-Switch APIs.
- Controller-to-Switch Multipart APIs.
- Symmetric Message APIs.

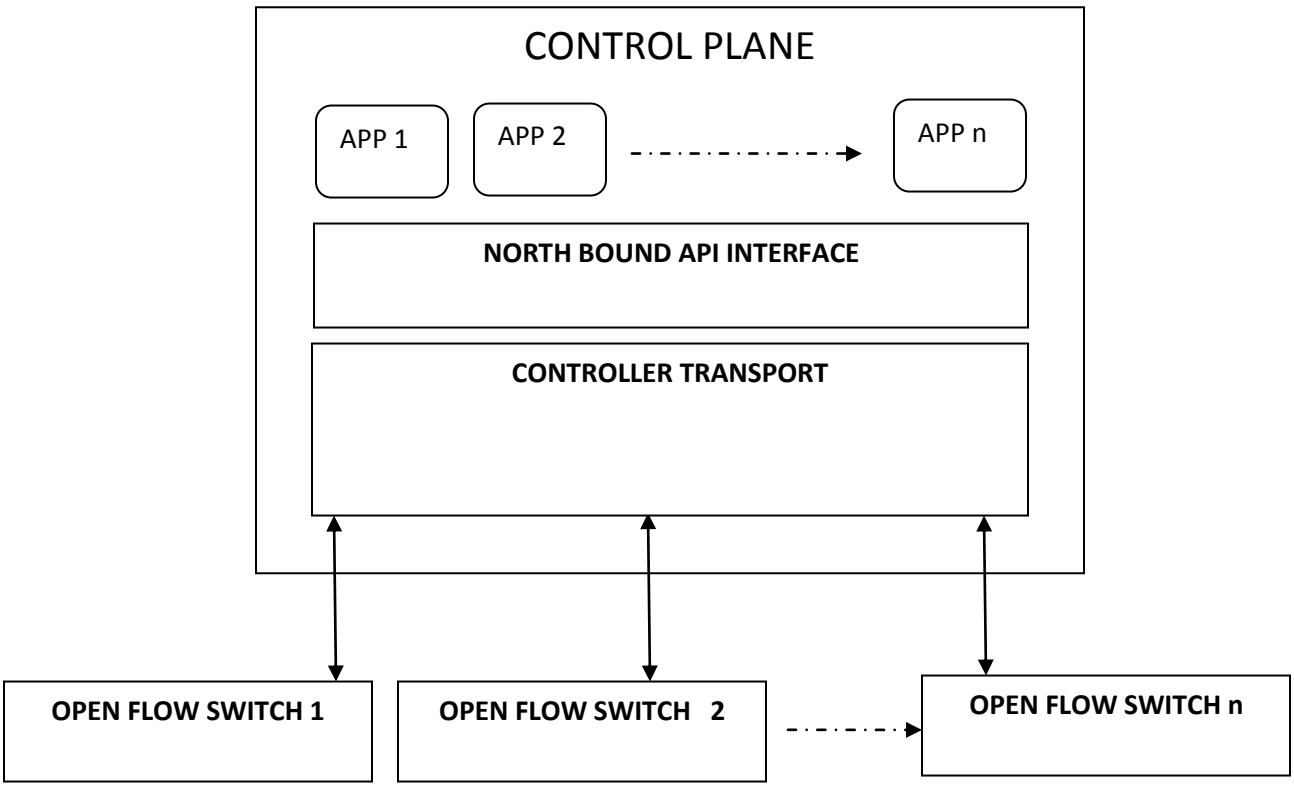
3. **Openflow utility APIs:**
   These API functions can be used by control plane and service plane applications to format openflow messages and get hold of parsed information from received messages. These APIs are again sub divided into to the following types.

- Set Match Field APIs.
- Push Instruction APIs.
- Push Action APIs.
- Group Utility APIs.
- Metering Utility APIs.
- Port Description APIs.
- Table feature Utility APIs.

4**. Generic utility APIs:**
These APIs set can be used by applications to some generic utilities such as Software timers, memory management and data structures such as hash lists. These APIs are again sub divided into the following types.

- Hash Table APIs.
- Memory Management APIs.
- Timer APIs.
- Of Message APIs.

# 5. Block Diagram:



# 6. Usage of North Bound APIs by using a Firewall Application:

1. Firewall application functionality will be created as shared library and will be Dynamically Loaded by Controller.

2. Firewall application, in its Application_Init() function, registers a callback function with DPRM for all domain events such as DOMAIN ADD EVENT, DOMAIN DELETE, etc by using **DPRM NBAPI**.

3. DPRM module calls all registered event call backs with domain handle, event type etc are as arguments whenever domain specific configuration event occurs.

4. Firewall application in its domain event callback function registers for DATA PATH READY event and as well as Table1( i.e Session Table ) PKT-IN asynchronous event by using **asynchronous NBAPI**.

5. Whenever a data path connects to the controller, all application's registered event callback functions are called.

6. Firewall application in its data path ready event callback function, adds a Session Table flow miss entry(i.e with no of match fields equal to zero) and action as forward to controller by using **Flow Mod NBAPI**. If no match is found in a flow table, the outcome depends on configuration of table miss flow entry. For example the packet may be forwarded to the controller over the OpenFlow channel, dropped, or continue to the next flow table. In our case, configuration is forward to the controller over the OpenFlow channel.

7. Whenever a packet comes to OpenFlow Switch, matching starts at the first flow table and may continue to additional flow tables. If no match is found in flow Table1(Session Table), as per Table1 flow miss entry action the packet is forwarded to Controller over OpenFlow channel by using PKT-IN asynchronous message.

8. Whenever PKT-IN comes to the controller, it calls all application registered asynchronous callback functions.

9. Firewall application in its asynchronous event callback function, parses the incoming packet and extract five tuple information and lookup for a matching policy rule. If no matching rule found it just drops the packet and return packet consumed return value to

the caller. If a matching policy rule found, based on policy action it either adds forward and reverse flow entries to the Session Table by calling 'sm_fw_app_add_session_entry' API or simply drops the packet. The Firewall's add session entry API internally uses **couple of NBAPIs** such as **message allocate**, **set match fields, set actions** and **set instructions** to form flow mod message and finally calls **Flow mod NBAPI** to add flow in the Session Table. After adding Session forward and reverse entries the packet is sent back from Controller to Switch by using **PKT-OUT NBAPI** with action as OFPP_TABLE by using **OUT PUT ACTION NBAPI** with OFPP_TABLE as an argument to the NB API, which submit the packet at Table0 level and the packet continue from Table0. This re-injected packet now matches Session flow entry at Table1 and go to the next table and so on.

## 6.1 Pseudo code:

The Firewall application sample Pseudo code shows usage of different types of North Bound APIs.

### 6.1.1 sm_fw_app_init

**Prototype:** int_t32 sm_fw_app_init(void)

**Description:** This function allocates and initializes application specific data structures and registers for domain events by using DPRM NB API.

**Pseudo code:**

```
void   sm_fw_app_init(void)
{

   1. Allocate and Initialize all application specific data structures.

    /* Registering to domain add/del/mod and other
       notifications  by using DPRM NBAPI*/
   2. if  (dprm_register_forwarding_domain_notifications(
                              DPRM_DOMAIN_ALL_NOTIFICATIONS,
                              sm_fw_app_domain_event_cbk,
                              NULL, NULL) != OF_SUCCESS)
      {
         i. of_debug("%s:Registering to forward domain  failed....\r\n",
```

__FUNCTION__);

ii. return;

}

3. return.

}

### 6.1.2 sm_app_domain_event_cbk

**Prototype:** static void sm_app_domain_event_cbk(
uint32_t notification_type,
uint64_t domain_handle,
struct dprm_domain_notification_data notification_data,
void *callback_arg1, void *callback_arg2)

**Description:** This function is called whenever a domain is added to DPRM data base. In this function application registers with controller for data path ready event and as well as Table1 PKT-IN asynchronous event.

**Pseudo code:**

```
static void sm_app_domain_event_cbk(
                    uint32_t notification_type,
                    uint64_t domain_handle,
                    struct dprm_domain_notification_data notification_data,
                    void *callback_arg1, void *callback_arg2)
{
    Switch (notification_type)
    {
       case DPRM_DOMAIN_ADDED:
       {
         /* NB API to register for data path ready event */
            if (of_cntlr_register_asynchronous_event_hook(
                                        domain_handle,
                                        DP_READY_EVENT,
                                        FW_PRIORITY,
                                        sm_fw_app_dp_ready_event,
                                        NULL, NULL) == OF_FAILURE)
            {
```

```
                of_debug("%s:%d Register DP Ready event cbk failed\r\n",
                                                    __FUNCTION__,__LINE__);
                Break;
            }

            /* NBAPI to register with Asynchronous PKT-IN message */
             if (of_cntlr_register_asynchronous_message_hook(
                                            domain_handle,
                                            SM_APP_SESSION_TABLE_ID,
                                            OF_ASYNC_MSG_PACKET_IN_EVENT,
                                            SM_FW_APP_PKT_IN_PRIORITY,
                                            sm_fw_app_pkt_process,
                                            NULL,NULL) != OF_SUCCESS)
            {
               of_debug("\n registering sample apps to Asyc message of
                            session table\r\n");
            }
            Break;
        }
        case DPRM_DOMAIN_DELETE:
        {
           /* NBAPI to deregister with domain event */
            of_cntlr_deregister_asynchronous_event_hook(
                                             domain_handle);

           /* NBAPI to deregister with PKT-IN async. event */
            of_cntlr_deregister_asynchronous_message_hook(
                                               domain_handle);
            Break;
        }
      }/* end of switch */
}
```

### 6.1.3 sm_fw_app_add_session_entry

**Prototype:** Int32_t sm_fw_app_add_session_entry(uint64_t datapath_handle,

uint32_t source_ip, uint32_t dest_ip,
uint16_t src_port, uint16_t des_port,
uint8_t protocol, uint16_t in_port,
uint8_t icmp_type, uint8_t icmp_code)

**Description:** This function uses couple of utility NBAPIs set match fields, actions and instructions and finally call flow mod NBAPI to add a flow to the given table.

**Pseudo code:**

```
Int32_t  sm_fw_app_add_session_entry(uint64_t datapath_handle,
                                uint32_t source_ip, uint32_t dest_ip,
                                uint16_t src_port, uint16_t des_port,
                                uint8_t protocol, uint16_t in_port,
                                uint8_t icmp_type, uint8_t icmp_code)
{

    /* calculated the flow mod message length for particular protocol */
    switch (protocol)
    {
       case OF_CNTLR_IP_PROTO_TCP:
       {
         msg_len =
                (OFU_ADD_OR_MODIFY_OR_DELETE_FLOW_ENTRY_MESSAGE_LEN+
                OFU_ETH_TYPE_FIELD_LEN+
                OFU_IPV4_SRC_FIELD_LEN+OFU_IPV4_DST_FIELD_LEN+
                OFU_TCP_SRC_FIELD_LEN+OFU_TCP_DST_FIELD_LEN+
                OFU_IP_PROTO_FIELD_LEN+
                OFU_GOTO_TABLE_INSTRUCTION_LEN);
         break;
       }
       case OF_CNTLR_IP_PROTO_UDP:
       {
          msg_len = (OFU_ADD_OR_MODIFY_OR_DELETE_FLOW_ENTRY_MESSAGE_LEN+
                    OFU_ETH_TYPE_FIELD_LEN+
                    OFU_IPV4_SRC_FIELD_LEN+OFU_IPV4_DST_FIELD_LEN+
                    OFU_UDP_SRC_FIELD_LEN+OFU_UDP_DST_FIELD_LEN+
                    OFU_IP_PROTO_FIELD_LEN+
                    OFU_GOTO_TABLE_INSTRUCTION_LEN);
         break;
```

```
            }
            case OF_CNTLR_IP_PROTO_ICMP:
            {
                msg_len = (OFU_ADD_OR_MODIFY_OR_DELETE_FLOW_ENTRY_MESSAGE_LEN+
                           OFU_ETH_TYPE_FIELD_LEN+
                           OFU_IPV4_SRC_FIELD_LEN+OFU_IPV4_DST_FIELD_LEN+
                           OFU_ICMPV4_TYPE_FIELD_LEN+OFU_ICMPV4_CODE_FIELD_LEN+
                           OFU_IP_PROTO_FIELD_LEN+
                           OFU_GOTO_TABLE_INSTRUCTION_LEN);
                break;
            }
        }/* end of switch */

    do
    {
        /* allocate buffer for flow mode message by using message NBAPI */
        msg = ofu_alloc_message(OFPT_FLOW_MOD, msg_len);
        if(msg == NULL)
        {
          of_debug("%s:%d OF message alloc error \r\n",__FUNCTION__,__LINE__);
          status = OF_FAILURE;
          break;
        }

        /* NBAPIs to set match fields to a flow entry. */
        /* start setting match field API must be called before calling set match field APIs */
        ofu_start_setting_match_field_values(msg);

        /* NBAPI to set incoming port as match field */
        retval = of_set_in_port(msg, &in_port);
        if(retval != OFU_SET_FIELD_SUCCESS)
        {
           of_debug("%s:%d OF Input Port set failed  \n",__FUNCTION__,__LINE__);
           status = OF_FAILURE;
           break;
        }
```

```
/*setting ethtype is mandatory before setting ip address*/

/* NBAPI to set ether type as match field */
uint16_t eth_type = 0x0800; /*Eth type is IP*/
retval = of_set_ether_type(msg,&eth_type);
if (retval != OFU_SET_FIELD_SUCCESS)
{
   of_debug("%s:%d OF Set IPv4 src field failed\n",__FUNCTION__,__LINE__);
   status = OF_FAILURE;
   break;
}

/* NBAPI to set source ip address as match field */
retval = of_set_ipv4_source(msg,&source_ip,FALSE,NULL);
if(retval != OFU_SET_FIELD_SUCCESS)
{
  of_debug("%s:%d OF Set IPv4 src field failed\n",__FUNCTION__,__LINE__);
  status = OF_FAILURE;
  break;
}


/* NBAPI to set destination ip address as match field */
retval = of_set_ipv4_destination(msg, &dest_ip, FALSE, NULL);
if (retval != OFU_SET_FIELD_SUCCESS)
{
   of_debug("%s:%d OF Set IPv4 dest field failed \n",__FUNCTION__,__LINE__);
   status = OF_FAILURE;
   break;
}

/*it is mandatory to set protocol match field before setting src and dest ports*/
/* NBAPI to set protocol match field */
retval = of_set_protocol(msg, &protocol);
if (retval != OFU_SET_FIELD_SUCCESS)
{
    of_debug("%s:%d OF Set ip protocol field failed\r\n",
```

```
                                        __FUNCTION__,__LINE__);
    status = OF_FAILURE;
    break;
}

switch (protocol)
{
   case OF_CNTLR_IP_PROTO_ICMP:
   {
     retval = of_set_icmpv4_type(msg, &icmp_type);
     if (retval != OFU_SET_FIELD_SUCCESS)
     {

        of_debug("%s:%d OF Set Src Port field failed\n",
                                    __FUNCTION__,__LINE__);
        Break;
     }

     retval = of_set_icmpv4_code(msg, &icmp_code);
     if (retval != OFU_SET_FIELD_SUCCESS)
     {
       of_debug("%s:%d OF Set Src Port field failed,err = %d\r\n",
                                        __FUNCTION__,__LINE__,retval);
       status = OF_FAILURE;
     }
     break;
   }
   case OF_CNTLR_IP_PROTO_UDP:
   {
     /*NBAPIs to set  UDP source and destination ports */
     retval = of_set_udp_source_port(msg, &src_port);
     if (retval != OFU_SET_FIELD_SUCCESS)
     {
        of_debug("%s:%d OF Set udp Src Port field failed \n",
                                        __FUNCTION__,__LINE__);
        status = OF_FAILURE;
        break;
```

```
        }

        retval = of_set_udp_destination_port(msg, &dst_port);
        if (retval != OFU_SET_FIELD_SUCCESS)
        {
            of_debug("%s:%d OF Set udp Dst Port field failed \n",
                                        __FUNCTION__,__LINE__);
            status = OF_FAILURE;
        }
        break;
      }
      case OF_CNTLR_IP_PROTO_TCP:
      {
        /* NBAPIs to set TCP source and destination ports */
        retval = of_set_tcp_source_port(msg, &src_port);
        if (retval != OFU_SET_FIELD_SUCCESS)
        {
           of_debug("%s:%d OF Set Src Port field failed \n",
                                    __FUNCTION__,__LINE__);
            status = OF_FAILURE;
            break;
        }

        retval = of_set_tcp_destination_port(msg, &dst_port);
        if (retval != OFU_SET_FIELD_SUCCESS)
        {
           of_debug("%s:%d OF Set Src Port field failed \n",
                                    __FUNCTION__,__LINE__);
           status = OF_FAILURE;
        }
        break;
      }
    }/* end of switch */
    If (status == OF_FAILURE)
    {
      If (msg != NULL)
         msg->desc.free_cbk(msg);
```

```
    return status;
 }
 /* This NBAPI is used to indicate end of setting match fields */
 ofu_end_setting_match_field_values(msg, match_start_loc, &match_fd_len);


/* The following NBAPI must be called before calling any instruction related NB
   APIs*/
ofu_start_pushing_instructions(msg);
/* NBAPI to push GOTO table instruction */
 retval = ofu_push_go_to_table_instruction(msg, SM_APP_L3ROUTING_TABLE_ID);
 if (retval != OFU_INSTRUCTION_PUSH_SUCCESS)
 {
     of_debug("%s:%d go to table instruction failed \n",
                                    __FUNCTION__,__LINE__);
    status = OF_FAILURE;
    break;
 }
 ofu_end_pushing_instructions(msg,inst_start_loc,&instruction_len);

 flags = 0;
 flags |= OFPFF_RESET_COUNTS;

 /* NBAPI to add flow entry to the given table with table id */
 retval = of_add_flow_entry_to_table(msg,
                                     datapath_handle,
                                     SM_FW_APP_SESSION_TABLE_ID, /*Table Id 1*/
                                     sess_entry_priority++, /*priority*/
                                     0, /*coockie*/
                                     0,/*Coockie amsk*/
                                     OFP_NO_BUFFER,
                                     flags,
                                     0, /*No Hard timeout*/
                                     60, /* Idle timeout*/
                                     NULL,
                                     NULL);
```

```
            if (retval != OFU_ADD_FLOW_ENTRY_TO_TABLE_SUCCESS)
            {
               of_debug("%s:%d  Error in adding miss entry\n",
                                        __FUNCTION__,__LINE__);
               status = OF_FAILURE;
               break;
            }
         }
         while(0);

         Return status;

}
```

### 6.1.4 sm_fw_app_pkt_process

**Prototype:**

```
         static int32_t

         sm_fw_app_pkt_process (int64_t controller_handle,
                                uint64_t domain_handle,
                                uint64_t datapath_handle,
                                struct of_msg *msg,
                                struct ofl_packet_in  *pkt_in,
                                void *cbk_arg1,
                                 void *cbk_arg2)
```

**Description:** This function parses and process the packet .As part of packet processing it lookup policy rule table and based on the policy action it either proceeds with packet or simply drops the packet. If a matching policy rule found it adds forward and reverse flow entries in the session table which part of open flow switch. After flow entries got added successfully it sends packet to the switch by using PKT-OUT NBAPI.

## Pseudo code:

```
static int32_t
sm_fw_app_pkt_process (int64_t controller_handle,
                       uint64_t domain_handle,
                       uint64_t datapath_handle,
                       struct of_msg *msg,
                       struct ofl_packet_in  *pkt_in,
                       void *cbk_arg1,
                        void *cbk_arg2)
{
```

1. Parse the packet and extract five tuple information: src-ip, dest-ip, src-port, dest-port and protocol.
2. Lookup policy rule table with five tuple information for a matching policy rule.
3. If (no rule matched)
   {
   a) Drop the packet;
   b) Return error code as packet consumed;
   }

```
/* Call API to add session flow entry in the given data path. This function
 * internally call couple of NBAPIs to add flow to the data path.
 */
```

4. sm_fw_app_add_session_entry(src-ip, dest-ip, src-port, dest-port, protocol, in_iface);

```
/* Add reverse flow entry for reply packets in reverse direction */
    sm_fw_app_add_session_entry(dest-ip, src-ip, dest-port, src-port,
                                protocol, out_iface);

/* calculate packet out message length, allocate buffer for open flow message, get
   packet in coming port, form packet out message by using set out action and other
   NBAPIs and send packet to the given datapath. */
```

```
msg_len =  (OFU_PACKET_OUT_MESSAGE_LEN+
                OFU_OUTPUT_ACTION_LEN+
                pkt_in->packet_length);

/* NBAPI to allocate memory for open flow message */
reply_msg = ofu_alloc_message(OFPT_PACKET_OUT, msg_len);
if (reply_msg == NULL)
{
    of_debug("%s:%d OF message alloc error \r\n",__FUNCTION__,__LINE__);
    status = OF_FAILURE;
    break;
}

/* NBAPIs to add actions */
ofu_start_pushing_actions(reply_msg);
retval = ofu_push_output_action(reply_msg,OFPP_TABLE,OFPCML_NO_BUFFER);
if (retval != OFU_ACTION_PUSH_SUCCESS)
{
   of_debug("%s:%d  Error in adding output action  err = %d\r\n",
                        __FUNCTION__,__LINE__,retval);
   reply_msg->desc.free_cbk(reply_msg);
   status = OF_FAILURE;
   break;
}
ofu_end_pushing_actions(reply_msg,action_start_loc,&action_len);

/* NBAPI to get in port id */
retval = ofu_get_in_port_field(msg, pkt_in->match_fields,
                        pkt_in->match_fields_length, &in_port);
if (retval != OFU_IN_PORT_FIELD_PRESENT)
{
   of_debug("%s:failed get inport ....inport=%d",__FUNCTION__,in_port);
   status = OF_FAILURE;
   break;
}
```

```
    retval = of_send_pkt_to_data_path(reply_msg,
                                      datapath_handle,
                                      TRUE,
                                      OFP_NO_BUFFER,
                                      in_port,
                                      0,
                                      pkt_in->packet_length,
                                      pkt_data,
                                      NULL, NULL);
    if (retval != OF_SUCCESS)
    {
       of_debug("%s:%d  Error in sending packet err = %d\r\n",
                                __FUNCTION__,__LINE__,retval);
       reply_msg->desc.free_cbk(reply_msg);
       status = OF_FAILURE;
    }
}
```

### 6.1.5 sm_fw_app_dp_ready_event

**Prototype:** int32_t

```
sm_fw_app_dp_ready_event(uint64_t  controller_handle,
                         uint64_t   domain_handle,
                         uint64_t   datapath_handle,
                         void      *cbk_arg1,
                         void      *cbk_arg2)
```

**Description:**

**Pseudo code:**

```
int32_t
sm_fw_app_dp_ready_event(uint64_t  controller_handle,
                         uint64_t   domain_handle,
                         uint64_t   datapath_handle,
                         void      *cbk_arg1,
                         void      *cbk_arg2)
{
```

```
    if (sm_fw_app_add_session_table_flow_miss_entry (
                                datapath_handle) != OF_SUCCESS)
    {
        of_debug("\n Failed to add session table flow miss entry”);
        Return OF_FAILURE;
    }
}
```

- **NBAPI usage for getting flow entries which internally uses multipart request message**:

### 6.1.6 cntlr_send_flow_stats_request

**Prototype:** int32_t cntlr_send_flow_stats_request (uint64_t dp_handle)

**Description:** This function gets flow stats by using multipart stats related NBAPIs.

**Pseudo code:**

```
{
   struct of_msg  *msg;
    /* Allocate message before using NBAPI */
    msg = ofu_alloc_message(OFPT_MULTIPART_REQUEST,
     OFU_INDIVIUAL_FLOW_ENTRY_STATS_REQ_MESSAGE_LEN);
     /* call NBAPI to get flow entries */
      retval = of_get_flow_entries(msg,
                                   dp_handle,
                                   OFPTT_ALL, //table_id,
                                   OFPP_ANY, //out_port,
                                   OFPG_ANY, //out_group,
                                   0, //cookie,
                                   0, //cookie_mask,
                                   cntlr_handle_flow_stats_msg,
```

```
                                NULL,
                                NULL);
}
```

### 6.1.7 cntlr_handle_flow_stats_msg

#### Prototype:

```
void
cntlr_handle_flow_stats_msg(
                uint64_t controller_handle,
                uint64_t domain_handle,
                uint64_t datapath_handle,
                struct of_msg *msg,
                uint8_t  port_no,
                void  *cbk_arg1,
                void  *cbk_arg2,
                uint8_t status,
                struct ofi_flow_entry_info *flow_stats,
                uint8_t more_ports)
```

#### Description:

This is a callback function to read flow entries from an multipart response

#### Pseudo code:

```
{
   Read flow_entry_stats from flow_stats;

   Send flow_entry_stats to management engine or print locally.


   If (msg != NULL)
      msg->desc.free_cbk(msg);
}
```