SwSS (entry-point), deep dive

Understanding, how the SwSS container works

Objectives

- Learn the SWSS container input options
- Why use ZMQ and Redis at the same time?
- Learn to configure input arguments
- Learn the initializations (SAI and SAI-Redis)
- Learn the DB connections
- Learn about SAI Attributes
- Learn the program flow

main.cpp

Argument parsing and config variables

- command line options
- for container initialization

b	batch size
i	asic instance string input
m	MAC-Address
r	enable/disable recording inputs
d	writing directory
h	help
s	enable synchronization
z	sai_deserialize_redis_communication_mode
f	file name
k	max bulk size limit in bulk mode
q	zmq server address

Why use ZMQ and Redis at the same time?

Redis	ZMQ (Zero M Queue)
Purpose (use case):	

- Data Store
    - Key Value pairs
- Benefits
    - **Language independent** Interface
    - **persistent** storage
    - **replication** and **inter-process** communication
	- **RTPS**: Real TIme Publish Subscribe, library
	    - No specific message format
	    - **low latency**
	- complex routing and filtering
	- **Distributed systems** / environments
	- **custom protocols**
	- Different operation modes (**IPC**, **TCP** and **Memory**)

Message Types:	
- Pub / Sub	- Pub / Subs - Request / Reply - Push / Pull - Pipeline (Buffer/Queue)
Storage Type:	
- In-memory	

Why ZMQ in SONiC and not any other RTPS or DDS

1) Telemetry and Monitoring
    - sonic **generates** a lot of **telemetry data**
        - network performance
        - traffic statistics
        - device health metrics (etc)
    - stream and deliver to visualization applications
1) Orchestration and Automation
    - Enabling network managers to **communicate** with **external** and **3rd party components** for **automation purposes**.
1) Example use case is,
    - Defining the **APIs** for **remote configuration** of the containers or the whole sonic environment. e.g in g**NMI** (g**RPC**, Network Management Interface) the commands from client get to the server and from there they are **communicated to SONiC** architecture through **ZMQ**

What is a Recorder

- Interface class to use recorder instances
- Affiliated to logging

```
namespace swss {
};

/* Interface to access recorder classes */
class Recorder {
public:
    static Recorder& Instance();
    static const std::string DEFAULT_DIR;
    static const std::string REC_START;
    static const std::string SWSS_FNAME;
    static const std::string SAIREDIS_FNAME;
    static const std::string RESPPUB_FNAME;

    Recorder() = default;
    /* Individual Handlers */
    SwSSRec swss;
    SaiRedisRec sairedis;
    ResPubRec respub;
};
```

Recorder Instance

- on input options, decision made if recorder has to be enabled or left disabled
  - swss,
  - respub,
  - sairedis
- Parameters set on initialization
  - setLocation
  - setFileName
  - startRec

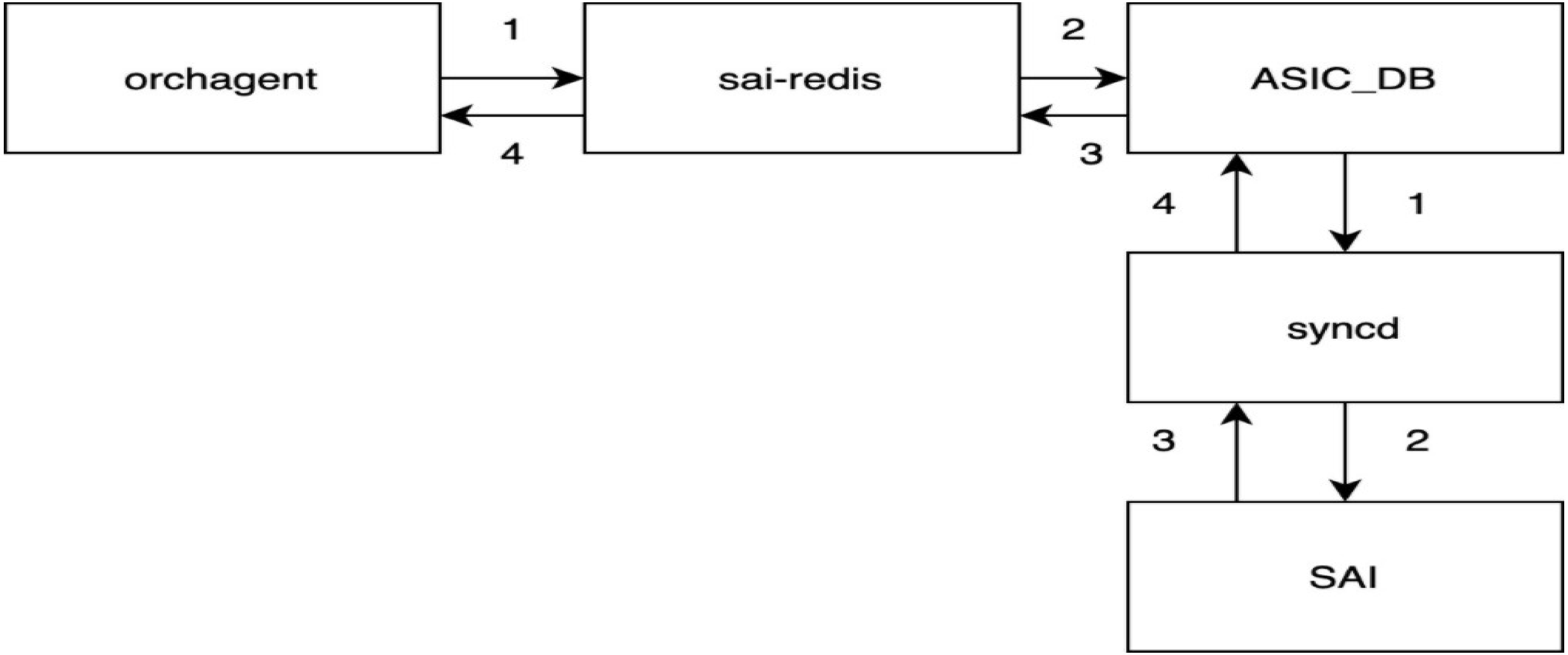
```
/* Initialize sairedis recording parameters */
Recorder::Instance().sairedis.setRecord(
    (record_type & SAIREDIS_RECORD_ENABLE) == SAIREDIS_RECORD_ENABLE
);
Recorder::Instance().sairedis.setLocation(record_location);
Recorder::Instance().sairedis.setFileName(sairedis_rec_filename);

/* Initialize sairedis */
initSaiApi();
initSaiRedis();

/* Initialize remaining recorder parameters  */
Recorder::Instance().swss.setRecord(
    (record_type & SWSS_RECORD_ENABLE) == SWSS_RECORD_ENABLE
);
Recorder::Instance().swss.setLocation(record_location);
Recorder::Instance().swss.setFileName(swss_rec_filename);
Recorder::Instance().swss.startRec(true);

Recorder::Instance().respub.setRecord(
    (record_type & RESPONSE_PUBLISHER_RECORD_ENABLE) ==
    RESPONSE_PUBLISHER_RECORD_ENABLE
);
Recorder::Instance().respub.setLocation(record_location);
Recorder::Instance().respub.setFileName(responsepublisher_rec_filename);
Recorder::Instance().respub.startRec(false);
```

Initialize SAI

Initialize SAI API
Initialize SAI Redis

```
/* Initialize sairedis */
initSaiApi();
initSaiRedis();
```

Orch-agent to ASICs: Information flow



Instantiate DB

- DB initializations
  - APPL_DB
  - CONFIG_DB
  - State_DB

```
int main(int argc, char **argv)

    // Instantiate database connectors
    DBConnector appl_db("APPL_DB", 0);
    DBConnector config_db("CONFIG_DB", 0);
    DBConnector state_db("STATE_DB", 0);

```


XFLOW RESEARCH

Initialize ZMQ communication (Conditional)

```cpp

    // Instantiate ZMQ server
    shared_ptr<ZmqServer> zmq_server = nullptr;
    if (enable_zmq)
    {
        SWSS_LOG_NOTICE("Instantiate ZMQ server : %s", zmq_server_address.c_str());
        zmq_server = make_shared<ZmqServer>(zmq_server_address.c_str());
    }
    else
    {
        SWSS_LOG_NOTICE("ZMQ disabled");
    }
```

- if **zmq_server** is specified in **option** then this **if command executes** and initializes zmq_communication.

switch_type and sai_attributes

- global variable

```
string gMySwitchType = "";
```

-	dpu	data processing unit
-	voq	virtual output queue
-	fabric	communication medium within multiple components inside a device
-	chassis-packet	end-to-end information flow

- chec swit hen configuring the sai_attribute_t object i.e attr
- vector<sai_attribute_t> attrs

```
// Get switch_type
getCfgSwitchType(&config_db, gMySwitchType);

sai_attribute_t attr;
vector<sai_attribute_t> attrs;

attr.id = SAI_SWITCH_ATTR_INIT_SWITCH;
attr.value.booldata = true;
attrs.push_back(attr);

if (gMySwitchType != "dpu")
{
    attr.id = SAI_SWITCH_ATTR_FDB_EVENT_NOTIFY;
    attr.value.ptr = (void *)on_fdb_event;
    attrs.push_back(attr);
}
```

sai_attributes : atrr -> (id, value(booldata, ptr))

Sr	SAI Attribute IDs	IF conditional on global variables	Reference Image

1.	SAI_SWITCH_ATTR_INIT_SWITCH		`attr.id = SAI_SWITCH_ATTR_INIT_SWITCH;`
			`attr.value.booldata = true;`
			`attrs.push_back(attr);`

2.	SAI_SWITCH_ATTR_FDB_EVENT_NOTIFY	gSwitchState != “**dpu**”	`attr.id = SAI_SWITCH_ATTR_FDB_EVENT_NOTIFY;`
			`attr.value.ptr = (void *)on_fdb_event;`
			`attrs.push_back(attr);`

3.	SAI_SWITCH_ATTR_PORT_STATE_CHANGE_NOTIFY		`attr.id = SAI_SWITCH_ATTR_PORT_STATE_CHANGE_NOTIFY;`
			`attr.value.ptr = (void *)on_port_state_change;`
			`attrs.push_back(attr);`

4.	SAI_SWITCH_ATTR_SHUTDOWN_REQUEST_NOTIFY		`attr.id = SAI_SWITCH_ATTR_SHUTDOWN_REQUEST_NOTIFY;`
			`attr.value.ptr = (void *)on_switch_shutdown_request;`
			`attrs.push_back(attr);`

5.	SAI_SWITCH_ATTR_PORT_HOST_TX_READY_NOTIFY		`attr.id = SAI_SWITCH_ATTR_PORT_HOST_TX_READY_NOTIFY;`
			`attr.value.ptr = (void *)on_port_host_tx_ready;`
			`attrs.push_back(attr);`

6.	SAI_SWITCH_ATTR_SRC_MAC_ADDRESS	gSwitchState != “**fabric**”
		&&
		getMACAddress
			`attr.id = SAI_SWITCH_ATTR_SRC_MAC_ADDRESS;`
			`memcpy(attr.value.mac, gMacAddress.getMac(), 6);`
			`attrs.push_back(attr);`

Redis Communication Mode at Initialization

- global variable

```
bool gSyncMode = false;
```

  - default **false**
  - set to **true** on **-s** option in cli. i.e **synchronization**
  - **deprecated,** use **-z** instead **(deserialize)**

```
        case 's':
            gSyncMode = true;
            SWSS_LOG_NOTICE("Enabling synchronous mode");
            break;
```

```
    if (gSyncMode)
    {
        SWSS_LOG_WARN("sync mode is depreacated, use -z param");

        gRedisCommunicationMode = SAI_REDIS_COMMUNICATION_MODE_REDIS_SYNC;
    }
```

OrchDaemon (init() and start() called in main.cpp)

```
int main(int argc, char **argv)
    shared_ptr<OrchDaemon> orchDaemon;
    if (gMySwitchType != "fabric")
    {
        orchDaemon = make_shared<OrchDaemon>(&appl_db, &config_db, &state_db, chassis_app_db.get(), zmq_server.get());
        if (gMySwitchType == "voq")
        {
            orchDaemon->setFabricEnabled(true);
            orchDaemon->setFabricPortStatEnabled(true);
            orchDaemon->setFabricQueueStatEnabled(false);
        }
    }
    else
    {
        orchDaemon = make_shared<FabricOrchDaemon>(&appl_db, &config_db, &state_db, chassis_app_db.get(), zmq_server.get());
    }

    if (!orchDaemon->init())
    {
        SWSS_LOG_ERROR("Failed to initialize orchestration daemon");
        exit(EXIT_FAILURE);
    }
```

```
    orchDaemon->start();
```

orchdaemon attributes

4 databases

- appl
- config
- state
- chassisApp

1 zmq server

orchagent > C orchdaemon.h > ...

```
class OrchDaemon
private:
    DBConnector *m_applDb;
    DBConnector *m_configDb;
    DBConnector *m_stateDb;
    DBConnector *m_chassisAppDb;
    ZmqServer *m_zmqServer;

    bool m_fabricEnabled = false;
    bool m_fabricPortStatEnabled = true;
    bool m_fabricQueueStatEnabled = true;

    std::vector<Orch *> m_orchList;
    Select *m_select;

    std::chrono::time_point<std::chrono::h
```

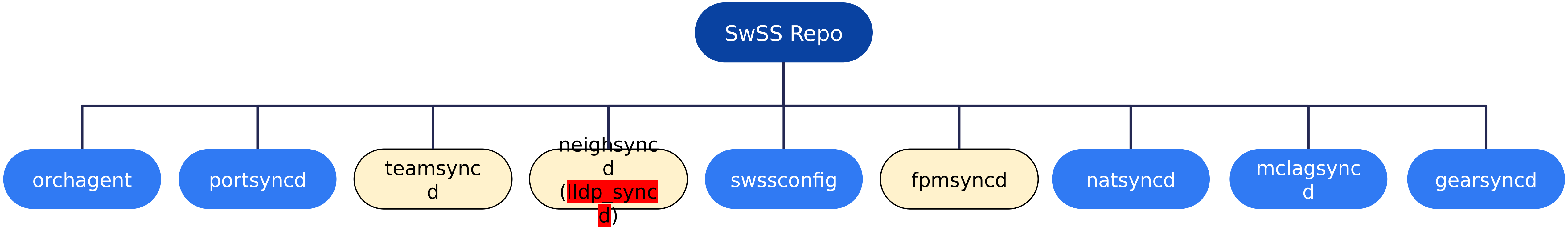
```
class OrchDaemon
{
public:
    OrchDaemon(DBConnector *, DBConnector *, DBConnector *, DBConnector *, ZmqServer *);
    ~OrchDaemon();
```

Child / Sub-Orch

Each of which inherits the **orchdaemon** class.

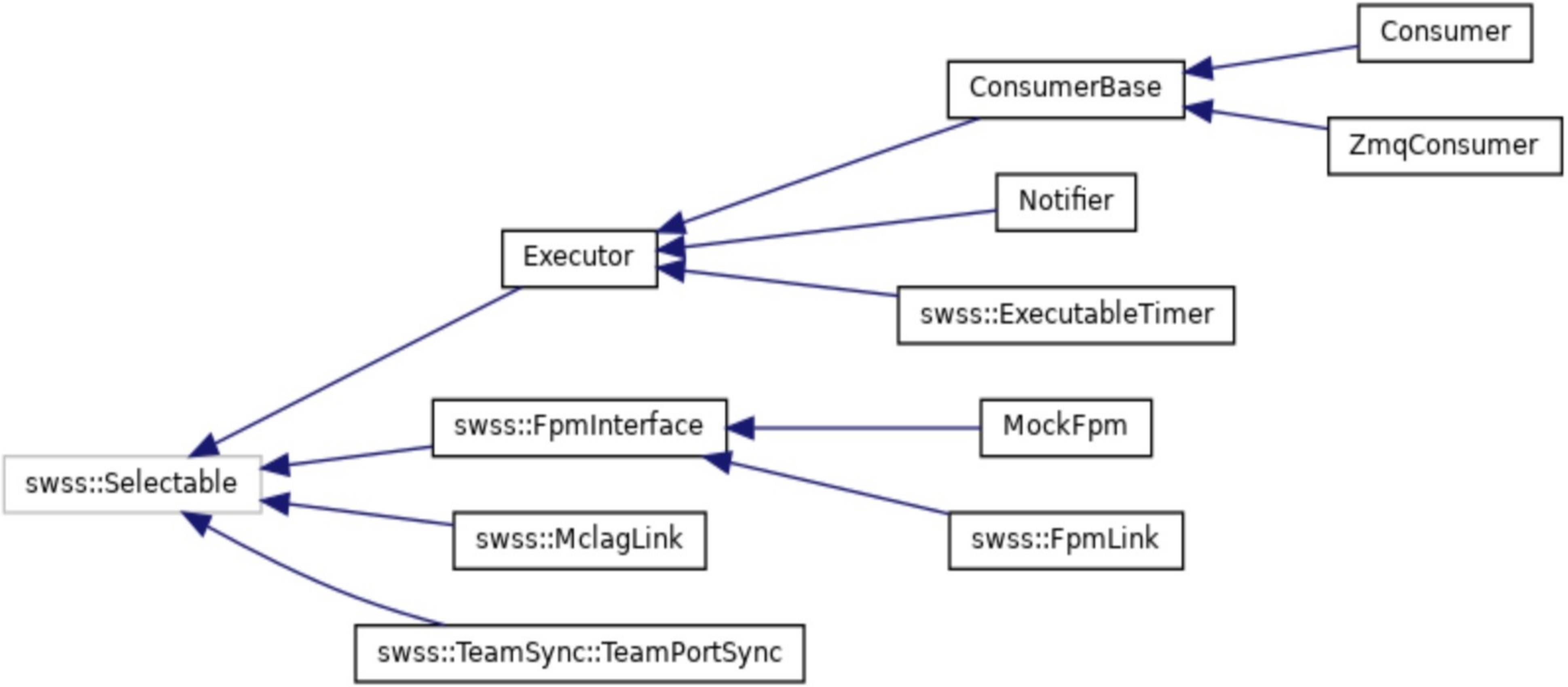
```
/*
 * Global orch daemon variables
 */
PortsOrch *gPortsOrch;
FabricPortsOrch *gFabricPortsOrch;
FdbOrch *gFdbOrch;
IntfsOrch *gIntfsOrch;
NeighOrch *gNeighOrch;
RouteOrch *gRouteOrch;
NhgOrch *gNhgOrch;
NhgMapOrch *gNhgMapOrch;
CbfNhgOrch *gCbfNhgOrch;
FgNhgOrch *gFgNhgOrch;
AclOrch *gAclOrch;
PbhOrch *gPbhOrch;
MirrorOrch *gMirrorOrch;
CrmOrch *gCrmOrch;
BufferOrch *gBufferOrch;
QosOrch *gQosOrch;
SwitchOrch *gSwitchOrch;
Directory<Orch*> gDirectory;
NatOrch *gNatOrch;
PolicerOrch *gPolicerOrch;
MlagOrch *gMlagOrch;
IsoGrpOrch *gIsoGrpOrch;
MACsecOrch *gMacsecOrch;
CoppOrch *gCoppOrch;
P4Orch *gP4Orch;
BfdOrch *gBfdOrch;
Srv6Orch *gSrv6Orch;
FlowCounterRouteOrch *gFlowCounterRouteOrch;
DebugCounterOrch *gDebugCounterOrch;
MonitorOrch *gMonitorOrch;
```

SwSS repo

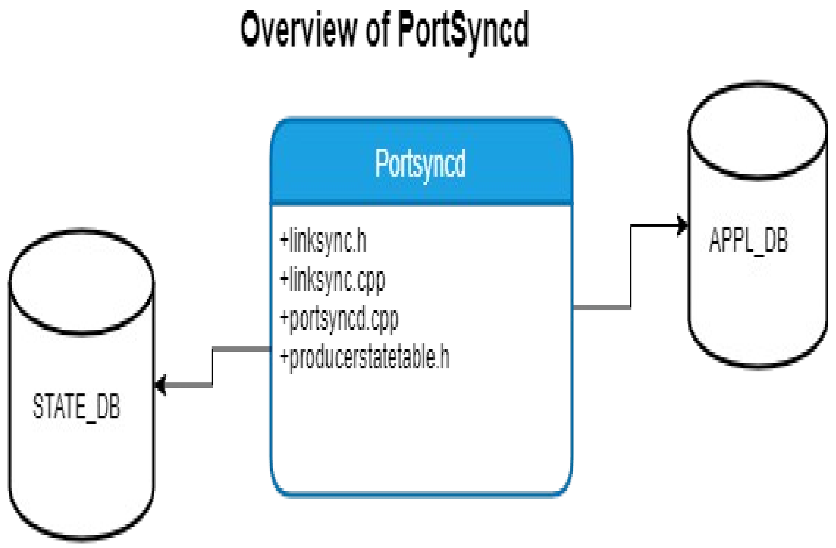

SwSS Repo
orchagent
portsyncd
teamsyncd
neighsyncd (lldp_syncd)
swssconfig
fpmsyncd
natsyncd
mclagsyncd
gearsyncd

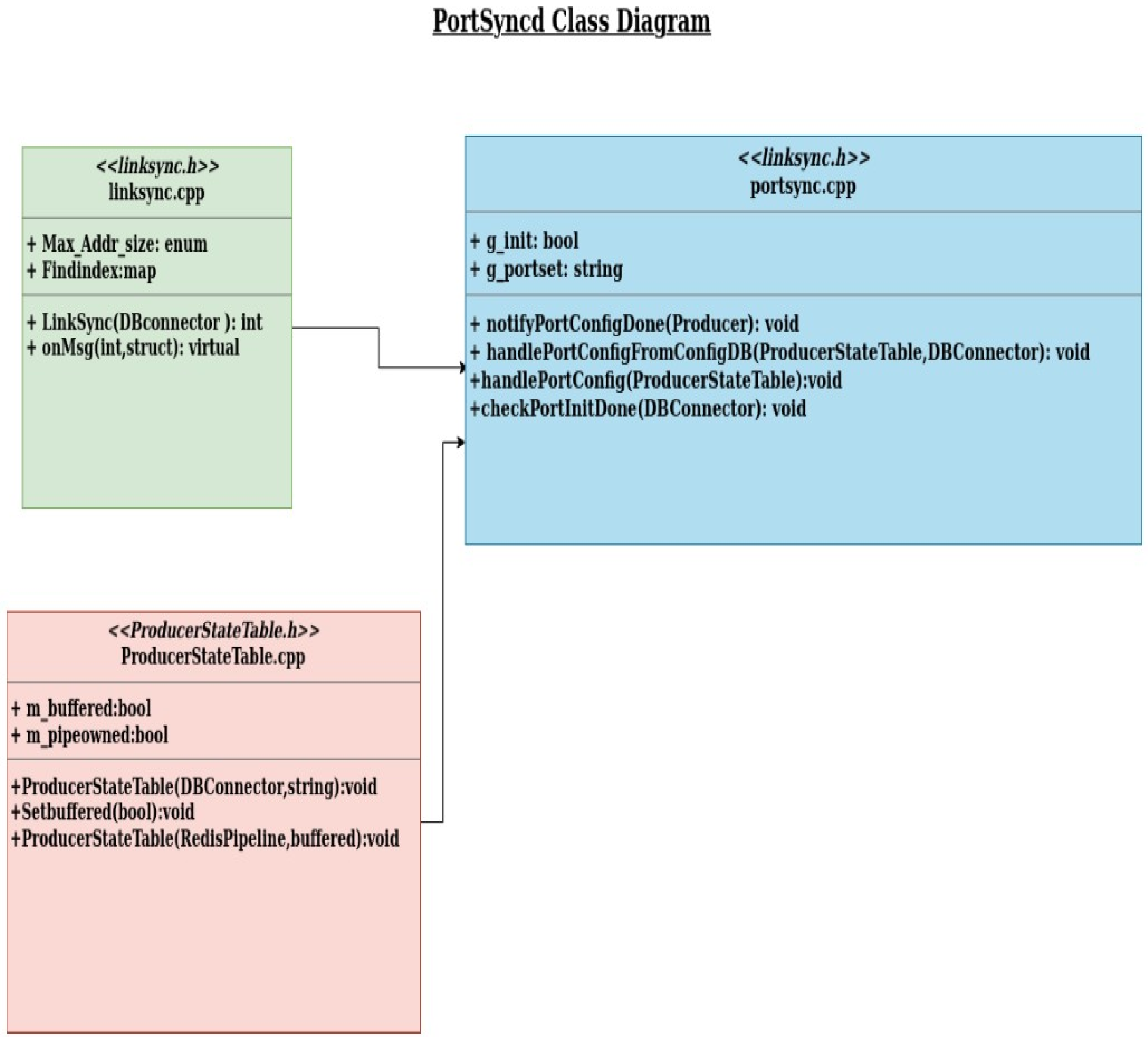
orchagent.cpp

SwSS Class Diagram



portsyncd class diagram

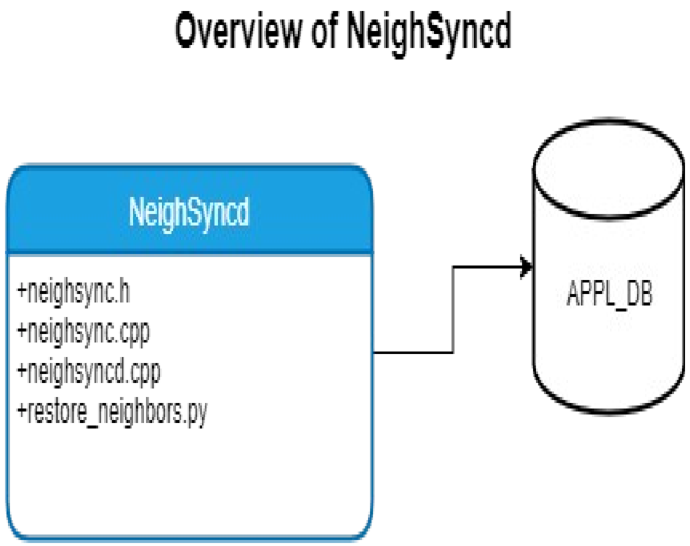


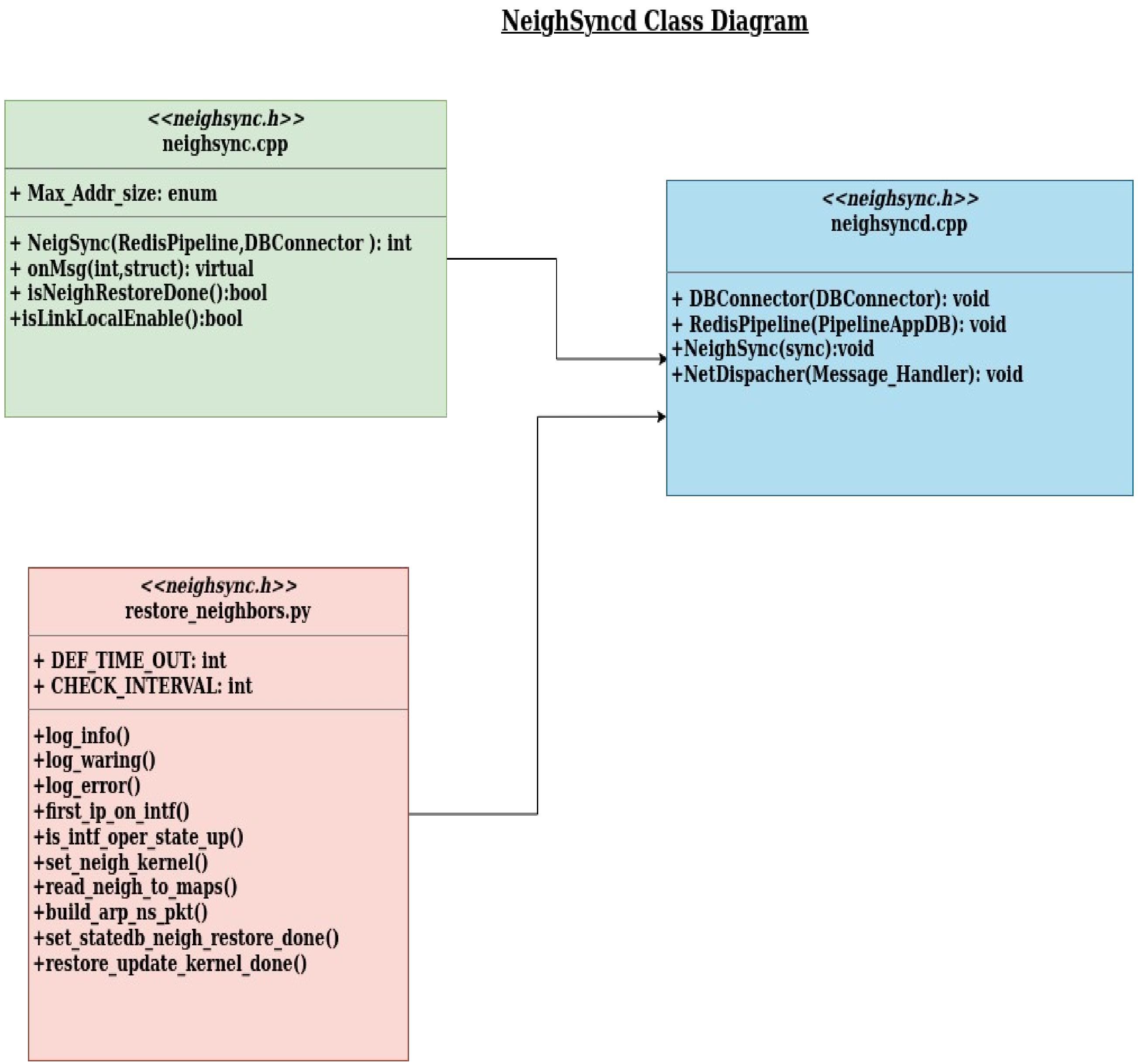


portsyncd **responsibilities**

- physical port information and synchronization with database
- Contains, **ProducerStateTable** which manages records of all **available ports** and their **states**
- **producerstatetable.h** is used by **linksync** file to manage the synchronization of ports with Database
- both these files are used by **portsyncd.cpp** to manage and notify port speed, lanes and mtu information to **APPL_DB** and **STATE_DB**

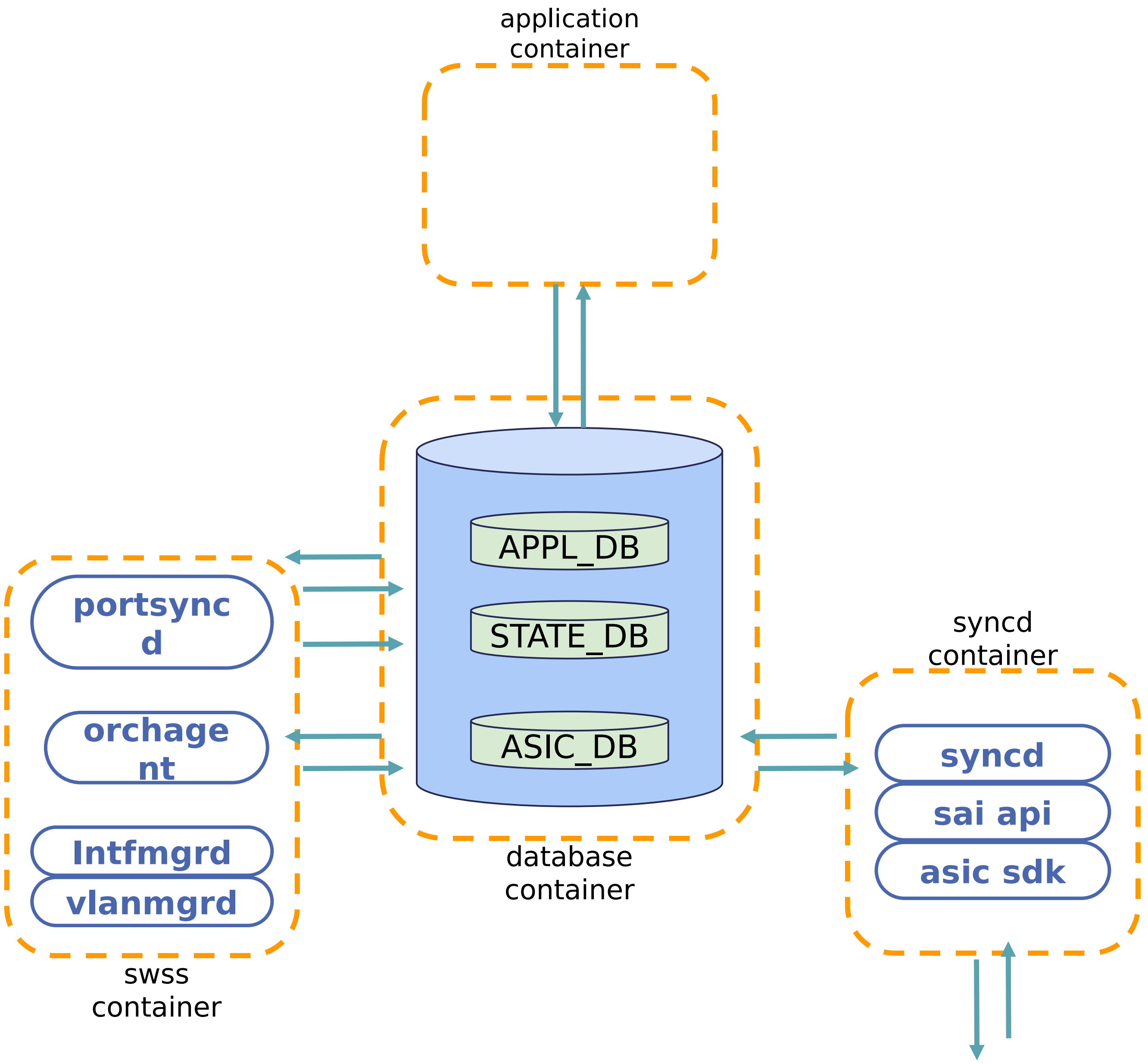
neighsyncd class diagram





neighsyncd **responsibilities**

- responsible for neighbour related activities like record management in data plane for L2 purposes.
- **neighsync.cpp** is there to discover and synchronize all the neighbors and maintain a **neighbors_table**
- Once all neighbors are synchronized it uses the **neighsyncd.cpp** file. In case of warm start the neighbor_table is read and cached onto the hashmap and keeps it updated from there on
- **restore_neighbors.py** is used for restoring the neighbor table from the **kernel** during system **warm reboot**. Agent **supervisord** in SwSS gets started on docker container startup. on warm reboot enabled it sets **stateDB flag** so neighsyncd can continue the reconciliation process.


application
container
APPL_DB
STATE_DB
ASIC_DB
database
container
portsync
d
orchage
nt
Intfmgrd
vlanmgrd
swss
container
syncd
container
syncd
sai api
asic sdk

Message Broker

Container **Process**	**Database** Container
***syncd**	

Container	**Container**
Redis_DB (Container**)**	

Application Container	**ASIC** hardware
SwSS	

Kernel Communication

Application Container		**SwSS** Container	
Receive from kernel	**Trans** to kernel	**Receive** from kernel /DB	**Trans** to kernel
Daemon process from **DB** (pushed by *syncd - SwSS)	_**syncd** process	__**synd** process (writes to DB and *mgrd - SwSS proceeds)	__**mngrd** process

XFLOW RESEARCH

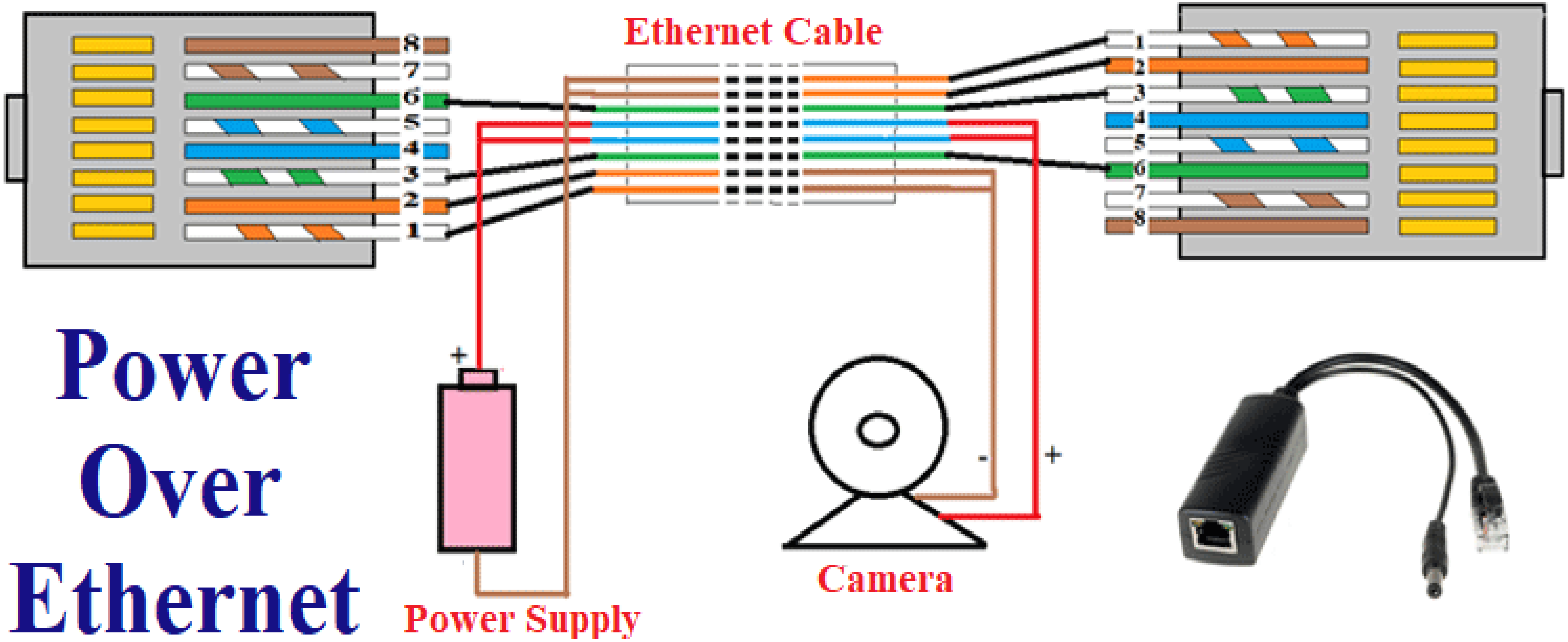
Key Terminologies

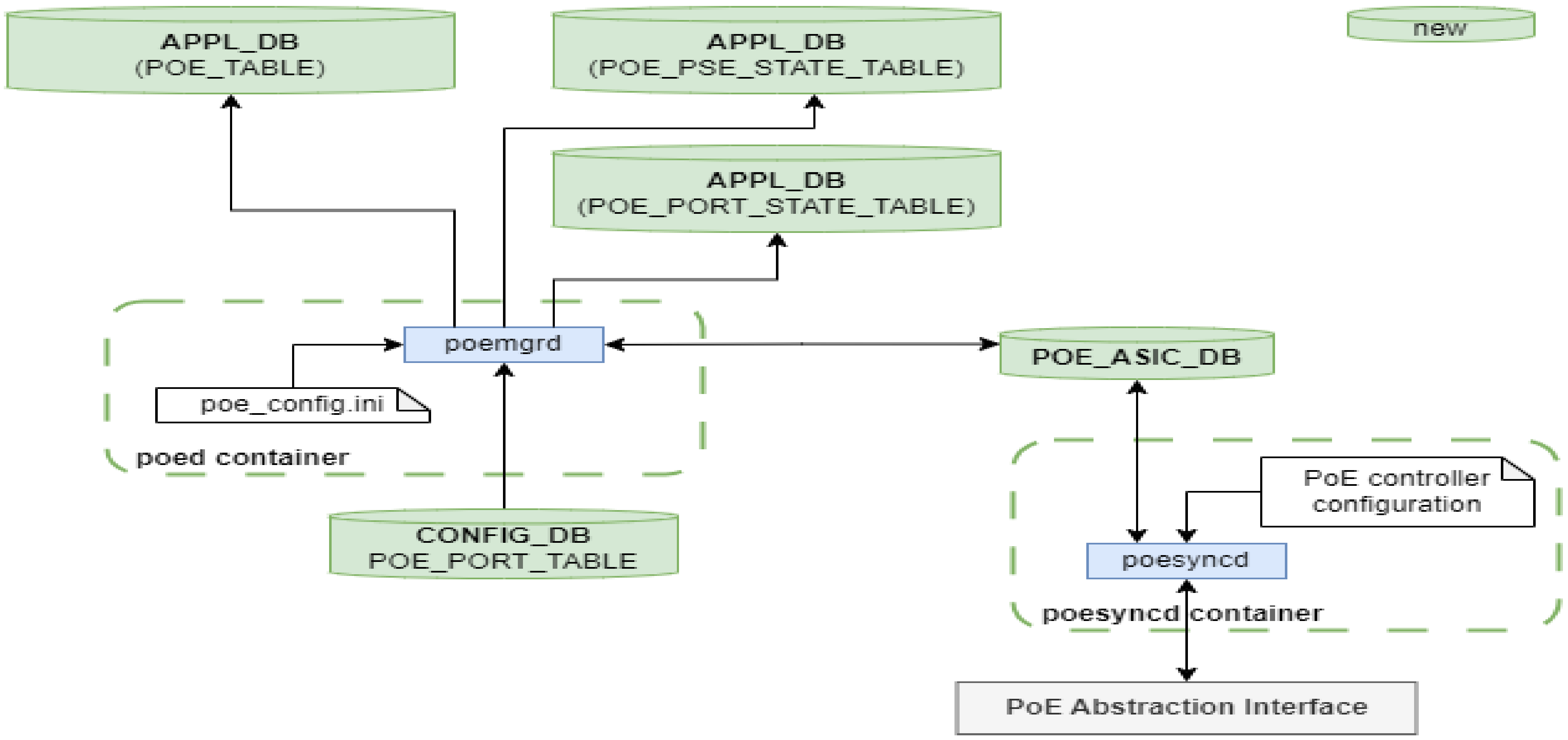
Application Container	Each application container, 1) Reads messages from kernel (netlink or socket or TCP) 2) Writes onto APPL_DB for other containers, through **syncd subprocesses**
Daemon	- Receives **netlink events** from the **kernel space** - **Shares back** the responses in **few cases**.
Syncd	- **Publish** to **DB**s, the current **state** of the **container** with the environment
Mgrd	- **Reads APPL_DB** and **sends back** the **response** from **SwSS** to the OS **kernel**. - **Example**: config interface in linux kernel etc
Orch	- **Reads** from **APPL_DB, CONFIG_DB and STATE_DB** - **Configure** the **AISCS** on start-up through **switch attributes** - **Dictates** to **ASICS**, south-bound forwarding devices
Cont -> Cont Communication	**Principle**: - The SONiC architecture follows **publish-subscribe** architecture. - **containers** only **interact** through the **DB** and **not directly** in any way.
South-bound communication Control-plane -> ASIC	1) orch-subprocess at SwSS 2) writes to ASIC_DB, through sairedisapi 3) Eventhandler at sync_d container receives and pasess to the ASIC
North-bound communication Application Cont -> SwSS	**1)** **Event** from **ASICS** through **driver** (kernel space), written to **ASIC_DB** by **syncd** **2)** **Orchagent**, reads the state and **notifies APPL_DB** (notify apps) and **ASIC_DB** (notify kernel) 3) reply comes back from the kernel to ***syncd.** e.g **portsyncd**

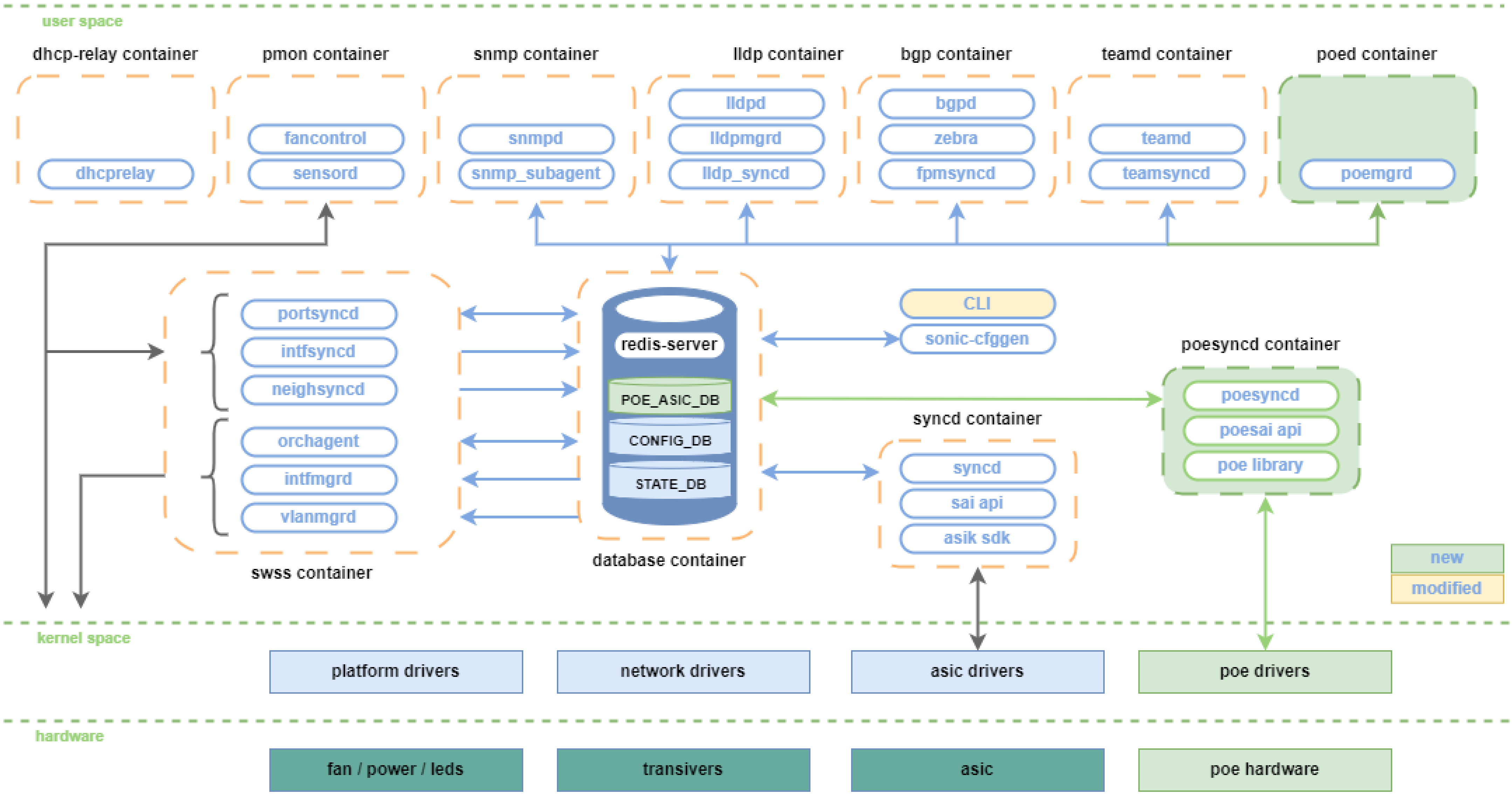
PoE proposed architecture

By NVIDIA

PoE: Power over Ethernet (Generally)

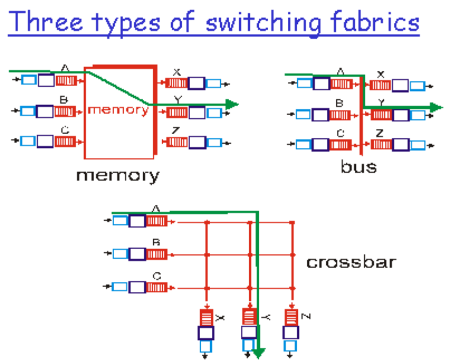



APPL_DB
(POE_TABLE)
APPL_DB
(POE_PSE_STATE_TABLE)
new
APPL_DB
(POE_PORT_STATE_TABLE)
poemgrd
POE_ASIC_DB
poe_config.ini
poed container
PoE controller
configuration
CONFIG_DB
POE_PORT_TABLE
poesyncd
poesyncd container
PoE Abstraction Interface



XFLOW RESEARCH

Switch Type

Sr #	Terminology	Details	Additional Reference
1 -	**dpu** **(**data processing unit**)**	- Involves tasks like switching and forwarding	- Another important aspect is that **DASH** (Disaggregated APIs for Sonic Hosts) **enabled DPU**s can be **used** in **SONiC infrastructure** out of the box.
2 -	**voq** **(**virtual output queue**)**	- Used to **control** / limit the flow of information on egress port as per network device capability	
3 -	**fabric** **(**communication within multiple components inside a device**)**	- Defines the **medium** and **protocol** on which **two** different **hardware** boards communicate (within one larger **system** or abstraction) e.g **cross GPU fabric** (common in **discussions** related to **hardware** in **AI-workgroup**)	
4 -	**chassis / modular** **(**dedicated rack type switches with extended functionalities**)**	- Rather than to **group** multiple switches through cables (**stackable** switch), we can opt for a chassis switch - switch **cards** can be swapped as per requirements - Most **flexibility**	 

XFLOW RESEARCH

DASH (Disaggregated APIs for SONiC Hosts)

- **Another project** by **Microsoft**
- Implements the **logic** for **hardware reconfigurability** through **software**
- Recently incorporated in SONiC
- **Extends** functionality of **SAI** for **DPU**s or **Smart NIC**s
- DASH enabled DPUs are compatible with SONiC deployed datacenter / infrastructure

SAI player (SwSS)

- It records every sai operation/command at every line
- And **saves** as a **recording file**
- Can be **played back** for debugging purposes
- Only **specific** to a certain **ASIC**
- Playback **not** applicable across **different ASIC**
- **Move** recording **file** to **syncd** and **replay**

change. Also it must also be used on the same ASIC with the same SAI version. Cross ASIC replaying is not supported.

Here is possible scenario:

1. bug is spotted, ASIC is not configured as expected
2. Remove comment lines from the recording file such as the ones which begin with `|#|`
3. copy recording file `sairedis.2017-04-27.02:47:15.674566.rec` from `swss` docker to `syncd` docker

```
docker cp swss:/sairedis.2017-04-27.02:47:15.674566.rec .
docker cp ./sairedis.2017-04-27.02:47:15.674566.rec syncd:/
```

4. stop all sonic processes and clean the redis DB for fresh start

```
docker exec -it swss killall orchagent
redis-cli flushall
docker exec -it syncd killall syncd
docker exec -id syncd service syncd start
```

5. replay recording in `syncd` docker

```
docker exec -it syncd saiplayer sairedis.2017-04-27.02:47:15
```

6. confirm that ASIC is in bad state as found in the first place
7. describe what the problem is and send recoding file to vendor for investigation

Any Questions

1) what is gear in swss? HLD was mentioned in a new proposal of PoE HLD community meeting. (It is a microservices open source framework incorporated in SONiC)
2) **fpmsyncd**, **teamsyncd, lldp_syncd** are exceptions. They are a **part of Application containers** but **code exists in SwSS** container as well. **Why**?
3) In the document shared about SwSS, what is the use of **restoreneighbors.py.** And how does this routine of **restore neighbor table** from **kernel** work in event of **warm reboot**?
4) What is the **broker** component in
   a) **across-containers** communication?
   b) **demon** and **database** communication?
   c) **database** and **ASIC** (Application plane and Control plane) communication?
      i) Is it bi-directional?
5) Do core devices also use only 4 wires(2 pairs) over and RJ-45 connector? There would be OFN, right? Is there a limit of how much bandwidth gets transmitted on copper wires?

Khappa Scene kab karna hai ....