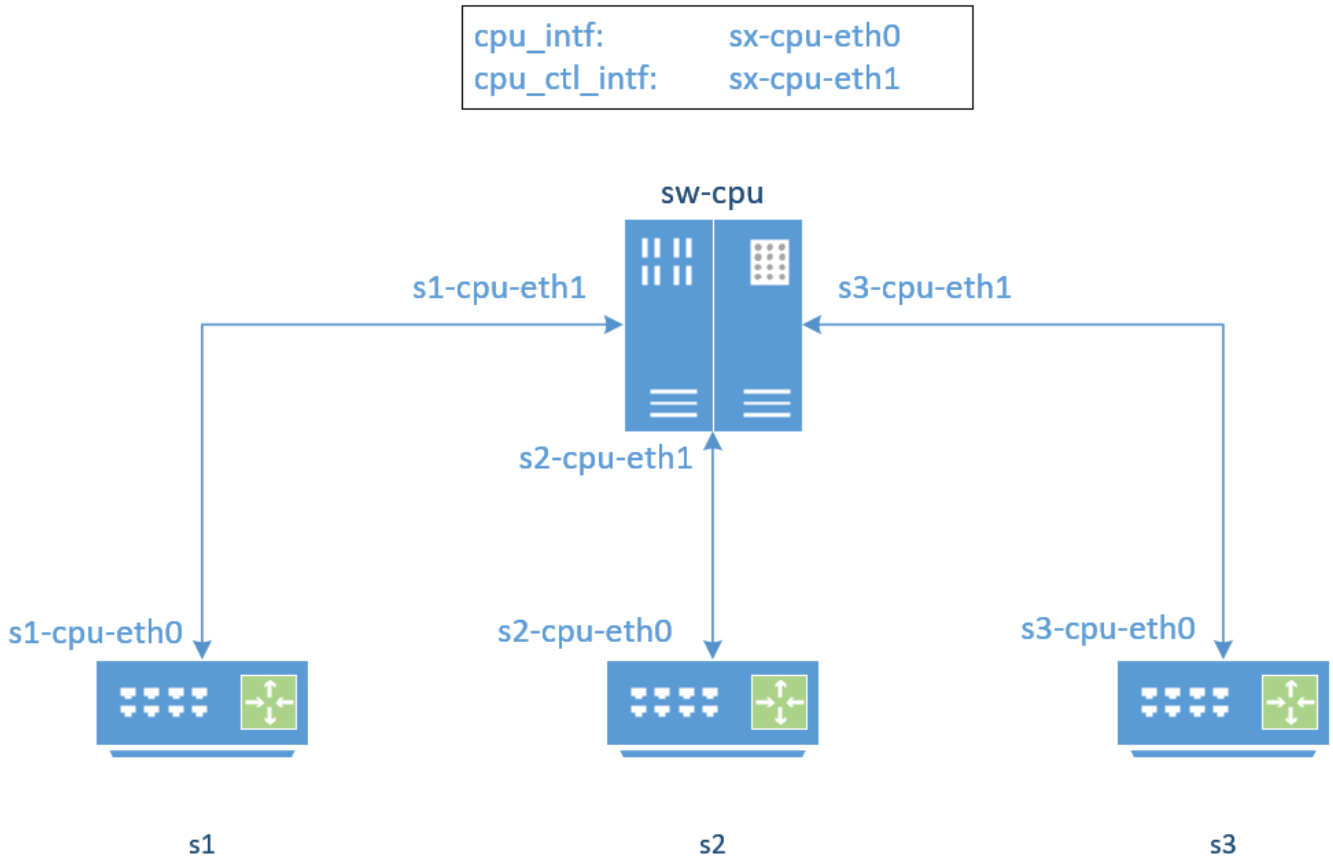
network_API.py

cpu port

- `enableCpuPort(name)`
  该方法使得指定交换机name获得cpu端口。
  - 该方法会创建一个新的交换机 `sw-cpu` ， 其class为 `LinuxBridge` 。
  - 将 `sw-cpu` 与交换机 `name` 绑定。因为 `LinuxBridge` 工作在l2，因此不涉及到ip地址的问题。
  - 交换机一侧的interface为 `name-cpu-eth0` ； cpu一侧的interface为 `name-cpu-eth1`
  - 交换机只能分到一个cpu端口。因为网桥的存在，所有转发到交换机这个cpu端口的pkt会被转发到sw-cpu的interface上
  - 如果存在多台交换机都开启了cpu port，那么该网络中始终只有一个 `sw-cpu`
  - 在controller中实现对某个交换机cpu监听pkt， 应该使用interface `name-cpu-eth1`
- `disableCpuPort(name)`
- `enableCpuPortAll()`
  开启网络内所有交换机的cpu port,如图为三个交换机开启cpu port



地址分配策略

- `l2()`
  对所有的设备ip都被分配在 `10.0.0.0/16` 的网络下
  前提
  - 每一个主机都只连接一个交换机
  - 网络中只有交换机和主机
  - 不存在平行链路
  - 交换机之间的链路不分配ip地址，只对主机分配ip地址
    分配细节
  - 只对host分配ip地址。取其最低字节为a, 次低字节为b，则该host分配到的ip地址为10.0.b.a。在编写网络拓扑时，如果存在两个标号相同的主机（如h1, h2）则会抛出错误，但还是会使用ip_generator为标号重复的主机分配ip。分配后主机IP的格式为ip/masklength
  - 对于mac地址的分配是按照链路，根据链路一侧已分配的host_ip进行。这也就意味着如果存在交换机之间的链路，那么这一段链路两段不会分配mac地址。mac地址的计算使用了 `utils.helper` 中的 `ip_address_to_mac` 方法，该方法将主机ip地址作为mac地址的四个低字节，首字节为0x00, 该方法返回的mac地址计算并不完全，其值为

    ```
    '00:%02x:0a.00.X.Y'
    ```

    次高位的字节在本方法中，根据结点的类型进行确定。主机次高位为00，交换机次高位为01

    ```
    host_mac = ip_address_to_mac(host_ip) % (0)
    irect_sw_mac = ip_address_to_mac(host_ip) % (1)
    ```

- `mixed()`
  每个 host 只能和一个交换机相连，连接在同一个交换机上的 host 属于同一个 /24 子网中，这个交换机于是就作为了 host 的网络层网关。
  - 首先根据交换机分配/24子网地址，依据是交换机sw_id（P4switch使用device_id, 其他交换机使用dpid），然后子网分配策略和l2分配IP地址稍微不同。

    ```
    upper_bytex = (sw_id & 0xff00) >> 8
    lower_bytex = (sw_id & 0x00ff)
    net = '10.%d.%d.0/24' % (upper_bytex, lower_bytex)
    ```

  - 然后按照链路，根据直接相连的交换机id对主机进行IP地址分配

```
sw_id = sw_to_id[direct_sw]
upper_byte = (sw_id & 0xff00) >> 8
lower_byte = (sw_id & 0x00ff)
...
host_ip = '10.%d.%d.%d' % (upper_byte, lower_byte, host_num)
```

- 为交换机相连的主机/交换机与内网直连的interface分配一个gateway（default route）/ip

```
host_gw = '10.%d.%d.254' % (upper_byte, lower_byte)
```

- 最后，为交换机之间的链路分配ip，依据是链路两端就交换机的id。

```
sw1_ip = '20.%d.%d.1/24' % (sw_to_id[node1], sw_to_id[node2])
sw2_ip = '20.%d.%d.2/24' % (sw_to_id[node1], sw_to_id[node2])
```

- `l3()`
  每个交换机都在网络层（layer 3）工作，所以每个 interface 都属于一个独立的子网。和mixed方式的区别在于，在mixed方式里，连接到同一个交换机的所有主机的gateway是相同的；但是在l3中，每一个主机都有一个自己的gateway。
- `is_multigraph()`
  用来判断网络拓扑是否是多重图

创建网络拓扑

- `startNetwork()`
  该方法创建并配置网络，基本上是网络拓扑构建代码的最后一行。在这个方法中，调用了一些本类中的方法。
  - `cleanup()`
  - `auto_assignment()`
    该方法更多是网络设备信息的完整性检查。其中IP和mac地址的分配可以在 `starNetwork()` 之前调用。具体可参考地址分配策略。
  - `compile()`
  - `printPortMapping()`
  - `save_topology()`
    - 默认的拓扑名为 `topology.json` ,同时回在 `/tmp` 下保存一个备份
  - `star_net_cli()`
    打开mininet控制台，需要我们在启动网络前调用 `enableCli()`

关于链路

- `addLink(node1, node2, port1=None, port2=None, key=None, **opts)`
  该方法为两个网络结点之间添加一条链路，返回值为key of the link(int)。opts有以下选项

        - intfName1(str)： 第一个结点的端口名
        - intfName2(str)： 第二个结点的端口名
        - addr1(str)：第一个结点的mac地址
        - addr2(str)：第二个结点的mac地址
        - weight(int)： 链路的权重，默认为1
    - deleteLink(node1, node2, key=None)
        - 这里的 key 用来区分多重图中两端连接了相同节点的边。
    - popLink(node1, node2, key=None)
- 返回值为 tuple:(link, key)
    - link： py:dict ;里面是链路的信息。
- links(sort=False, withKeys=False, withInfo=False)
  输出所有链路 list of (src, dst [, key, info])
- setBw(self, node1, node2, bw, key=None)
  bw in Mbps
- setBwAll(bw)
- setDelay(node1, node2, delay, key=None)
  delay in ms
- setDelayAll(delay)
- setLoss(node1, node2, loss, key=None)
  loss e.g. 0.5为50%丢包率
- setLossAll(loss)
- setMaxQueueSize(node1, node2, max_queue_size, key=None)
  设置队列规则（qdisc）可能同时持有的最大数据包数量
- setMaxQueueSizeAll(max_queue_size)

网络结点

- addNode(name, **opts)
- enableLog(name, log_dir='./log')
- enableLogAll(self, log_dir='./log')
- enableScheduler(self, name, path='/tmp')
- enableSchedulerAll(self, path='/tmp')
- addTaskFile(self, filepath, def_mod='p4utils.utils.traffic_utils')