# p4utils/utils/topology.py 源码阅读

The topology file can be imported in a Python script by using the utility :py:func: `~p4utils.utils.helper.load_topo()` , as shown in the following

```
from p4utils.utils.helper import load_topo
topo = load_topo('topology.json')
```

这里的 `topology.json` 是虚拟网络构建成功后由 p4utils 相关工具**自动生成的拓扑信息文件，无需手动编写**。load_topo() 返回的是topology中定义的NetworkGraph对象。

- **topology.json的结构**

```
{
    "directed": false/true,
    "multigraph": false/true,
    "graph": {},
    "nodes": [{...}, {...}, ...],
    "links": [{...}, {...}, ...]
}
```

  - `node` 的结构
    - host（属性不完全）
      - `log_enable`
        - `true`：允许日志记录
        - `false`：不允许日志记录
      - `log_dir`：日志文件夹路径
      - `cls`
      - `isHost`： `true`
      - `defaultRoute`： 默认路由
      - `mac`
      - `ip`
      - `id`：例如h1
    - switch
      - `p4_src`：配置交换机的p4文件
      - `pcapdump`
        - `true`：对经过该交换机的pkt进行dump

        - `pcap_dir`
        - `log_enable`
            - `true`：允许日志记录
            - `false`：不允许日志记录
        - `log_dir`：日志文件夹地址
        - `cli_input`：控制层命令输入文件(如果使用python应该是没有的)
        - `cls`
        - `isP4Switch`： `true`
        - `isSwitch`： `true`
        - `defaultRoute`： 默认路由
        - `devive_id`
        - `thrift_port`：控制层端口，默认9090
        - `json_path`：网络拓扑json的路径
        - `id`：例如s1
        - `cpu_port`
            - true: 拓扑中 net.enableCpuPortAll()
        - `cpu_port_num`：开启cpu port后，交换机的cpu 端口号
        - `cpu_intf`：交换机cpu端口名
        - `cpu_ctl_intf`:Controller一侧的cpu端口名
    - `link` 的结构

      `link` 是网络中两个节点之间的单项边，也就是说，在实际虚拟网络中两个结点之间的链路需要用两个相反的 `link` 表示。

# NetWorkGraph类

`NetworkGraph` 类继承自 `networkx.Graph` 。查询允许**检索有关网络拓扑的信息**的API。它还提供执行图形计算的有用方法（例如，使用Dijkstra算法获取最短路径。内部通过两个方法维护了三个字典字段。

> ## 三个字典字段
>
> - `edge_to_intf`
>
>   存储了边的接口信息，可以通过 `[node1][node2]` 检索，结果是node1与node2相连的接口信息，后面的两个字典都是由这个字典衍生出来的。这个字典是一个嵌套字典{{}}, 使用上述方法检索之后获得的其实是两个结点之间的link信息。
> - `node_to_intf`
>
>   存储了结点的接口信息，可以通过 `[node1][intfName]` 检索, 结果是node1指定接口intfName为源的link信息

- `ip_to_host`
  使用ip检索主机

## 内部方法

两个内部方法来维护

- `edge_to_intf()`
  在这个方法中，对link中的属性后的数字名作了替换。例如(port1, port2)->(port, port_neigh)。该边的源端的1去掉，2替换为_neigh。在后面的方法使用fields进行筛选时应该注意。
- `_populate_dicts()`

# 公有方法

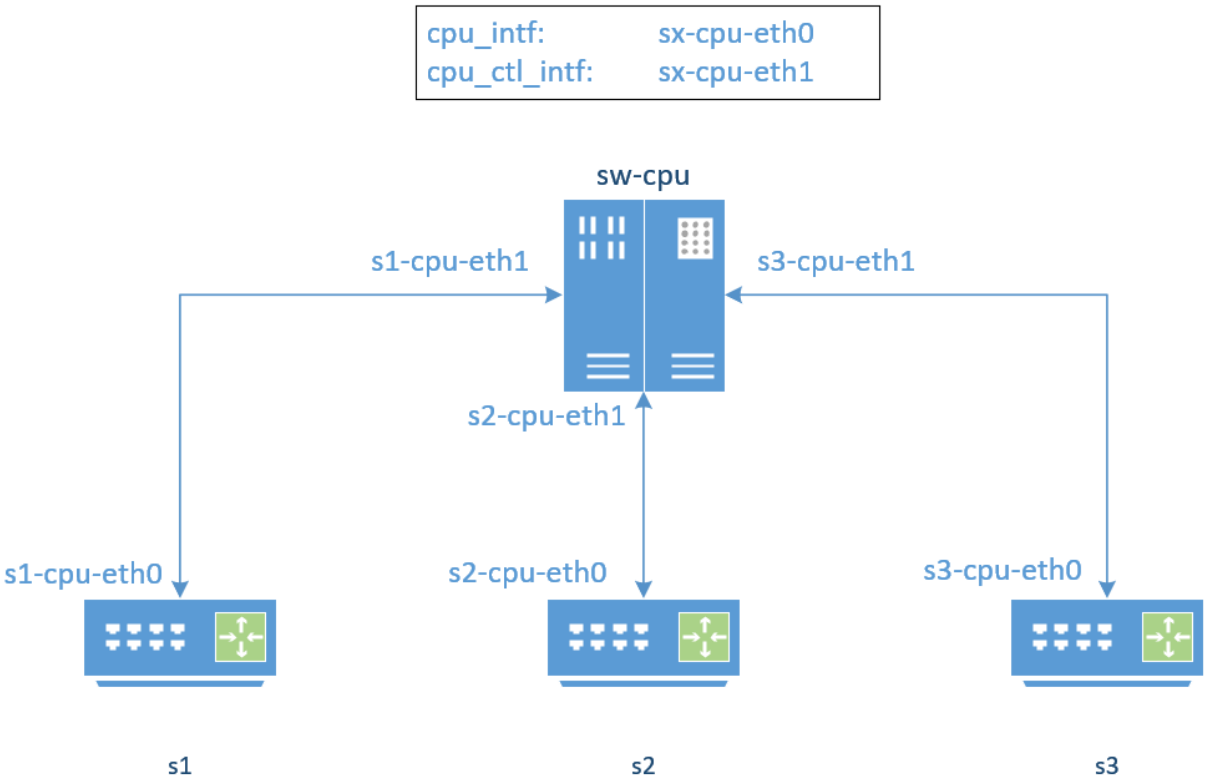
- `get_intfs(fields = [])`
- `get_node_intfs(fields = [])`
  - 获取每一个结点的接口信息，如果不指定fields，返回的就是 `node_to_intf` 字典字段
  - 如果要返回端口， 交换机的端口类型包括三种
    - 普通端口
    - `cpu` 端口 cpu端口单独可以通过 `get_cpu_port_index` 取得
    - `lo` 端口
- `get_nodes(fields = [])`
  返回的是nodes， 其中每个node元素的字段可以参考topology.json
- `get_switches(fields = [])`
  这个方法是对结点进行了筛选，依据是交换机的 `isSwitch` 属性为true，返回的是nodes的子集
- `get_P4switches(fields = [])`
  这个方法是对结点进行了筛选，依据是交换机的 `isP4Switch` 属性为true，返回的是nodes的子集
- `get_hosts(fields = [])`
  这个方法是对结点进行了筛选，依据是主机的 `isHost` 属性为true，返回的是nodes的子集
- `get_routers(fields = [])`
  这个方法是对结点进行了筛选，依据是主机的 `isRouter` 属性为true，返回的是nodes的子集
- `get_neighbors(name)`
  输入为node name，返回为邻居结点的name(str)
- `isNode()`
  输入为node name。如果节点存在，返回True；否则返回false
- `checkNode(name)`
  对于结点存在性的健全性检查，如果节点不存在，则抛出NodeNotExist
- `isIntf(node1, node2)`

    - `node1(str)，node2(str)`
    - 如果node1和node2之间存在接口连接，返回True；否则返回false
- `checkIntf(node1, node2)`
  健全性检查， 如果不存在抛出IntfDoesNotExist
- `isHost(name)`
  input: name(str), output: True if(name is a host); otherwise: false
- `isSwitch(name)`
- `isP4Switch(name)`
- `isP4RuntimeSwitch(name)`
- `isRouter(name)`
- `isType(name, node_type)`
  `node_tpye` 一共有五种
    - `'host'`
    - `'switch'`
    - `'p4switch'`
    - `'p4rtswitch'`
    - `'router'`
- `node_to_node_interface_ip(node1, node2)`
  返回 link<node1,node2> 源端node1接口的ip，格式为ip/maskLength
- `node_to_node_interface_bw(node1, node2)`
  返回 link<node1, node2> 的带宽bw(int)，若没有bw则返回-1
- `node_interface_ip(node, intf)`
  返回指定结点接口的ip， 格式为ip，没有给出子网掩码的长度
- `node_interface_bw(node, intf)`
  返回指定结点接口的带宽bw(int)，如果没有返回-1
- `subnet(node1, node2)`
  方法中使用了ipaddress中的ip_interface方法根据ip和subnet mask获得子网ip

```
from ipaddress import ip_interface
```

- `get_interfaces(name)`
  返回的是name结点各接口的名称(list)
- `get_cpu_port_intf(name, quiet=False)`
  返回的是p4交换机cpu 端口名(str)，用于CPU pkt
- `get_cpu_port_index(name, quiet=False)`
  返回p4交换机 cpu 端口号(int)
- `get_thrift_port(self, name)`
  返回p4交换机Thrift端口号(int)（使用Thrift API时使用）

- `get_thrift_ip(self, name)`
  返回p4交换机监听Thrift连接的ip，utils的源代码只会在交换机名正确时返回0.0.0.0(str)；否则抛出TypeError
- `get_grpc_port(self, name)`
  返回gRPC端口号(int)
- `get_ctl_cpu_intf(name)`
  返回controller side cpu interface name(str)
  - `cpu_ctl_intf` 与 `cpu_intf` 的联系&区别
    - 二者可以共用一个端口
    - `cpu_ctl_intf` 用于controller监听，图中的cpu就是controller




- `node_to_node_port_num(node1, node2)`
- `node_to_node_mac(node1, node2)`
- `total_number_of_paths()`
  计算网络中所有结点之间最短路径的数量(int)
- `get_shortest_paths_between_nodes(node1, node2)`
  计算两个节点间的所有简单路径（list）
  每个路径以 tuple 给出
- `keep_only_switches()`
- `keep_only_p4switches()`
- `keep_only_p4switches_and_hosts()`

以及几个需要networkx子类可视化时调整样式、颜色的方法。

```
def get_intfs(self, fields=[]):
        """Retrieves all the interfaces and their configuration parameters using
        the *nodes* as indexes.

        Args:
          fields (list): optional list of :py:class:`str` that specifies which
                         parameters need to be retrieved (and in which order)

        Returns:
            dict: the attribute ``edge_to_intf`` if ``fields`` is empty, or a dictionary of
            dictionaries of tuples, where the first two dictionaries are indexed by node and
            the tuples contain the values of the specified interface configuration parameters
            in the exact order they are placed in ``fields``.

        Note:
            If only one element is present in ``fields``, then no tuple is created and
            the value is directly put in the inner dictionary.

        Example:
            Let us consider the following example to clarify::

                >>> topo = NetworkGraph()
                >>> ...
                >>> print(topo.get_intfs())
                {'h1': {'s2': {'port': 0, 'port_neigh': 1, ...}, ...}, ...}
                >>> print(topo.get_intfs(fields=['port', 'port_neigh']))
                {'h1': {'s2': (0, 1), ...}, ...}
                >>> print(topo.get_intfs(fields=['port']))
                {'h1': {'s2': 0, ...}, ...}
        """
```

```
def get_node_intfs(self, fields=[]):
        """Retrieves all the interfaces and their configuration parameters using *node*
        and *interface name* as indexes.

        Args:
            fields (list): optional list of :py:class:`str` that specifies which
                           parameters need to be retrieved (and in which order)

        Returns:
            dict: the attribute ``node_to_intf`` if ``fields`` is empty, or a dictionary of
            dictionaries of tuples, where the first  dictionary is indexed by node, the second
            one by interface name and the tuples contain the values of the specified interface
            configuration parameters in the exact order they are placed in ``fields``.

        Note:
            If only one element is present in ``fields``, then no tuple is created and
            the value is directly put in the inner dictionary.

        Example:
            Let us consider the following example to clarify::

                >>> topo = NetworkGraph()
                >>> ...
                >>> print(topo.get_node_intfs())
                {'h1': {'h1-eth0':{'port': 0, 'port_neigh': 1, ...}, ...}, ...}
                >>> print(topo.get_node_intfs(fields=['port', 'port_neigh']))
                {'h1': {'h1-eth0': (0, 1), ...}, ...}
                >>> print(topo.get_node_intfs(fields=['port']))
                {'h1': {'h1-eth0': 0, ...}, ...}
        """
```

```
def get_nodes(self, fields=[]):
        """Retrieves all the nodes and their configuration parameters.

        Args:
            fields (list): optional list of :py:class:`str` that specifies which
                           parameters need to be retrieved (and in which order)

        Returns:
            dict: all the nodes configuration parameters if ``fields`` is empty, or
            a dictionary of tuples, where the first one is indexed by node name and
            the second ones collect the values of the selected parameters in the exact
            order they are placed in ``fields``.

        Note:
            If only one element is present in ``fields``, then no tuple is created and
            the value is directly put in the dictionary.

        Example:
            Let us consider the following example to clarify::

                >>> topo = NetworkGraph()
                >>> ...
                >>> print(topo.get_nodes())
                {'h1':{'isHost': True, 'isSwitch': False, ...}, ...}
                >>> print(topo.get_nodes(fields=['isHost', 'isSwitch']))
                {'h1': (True, False), ...}
                >>> print(topo.get_nodes(fields=['isHost']))
                {'h1': True, ...}
        """
```

```
def get_switches(self, fields=[]):
        """Retrieves all the switches and their configuration parameters.

        Args:
            fields (list): optional list of :py:class:`str` that specifies which
                           parameters need to be retrieved (and in which order)

        Returns:
            dict: all the switches configuration parameters if ``fields`` is empty, or
            a dictionary of tuples, where the first one is indexed by switch name and
            the second ones collect the values of the selected parameters in the exact
            order they are placed in ``fields``.

        Note:
            If only one element is present in ``fields``, then no tuple is created and
            the value is directly put in the dictionary.

        Example:
            Let us consider the following example to clarify::

                >>> topo = NetworkGraph()
                >>> ...
                >>> print(topo.get_switches())
                {'s1':{'isHost': False, 'isSwitch': True, ...}, ...}
                >>> print(topo.get_switches(fields=['isHost', 'isSwitch']))
                {'s1': (False, True), ...}
                >>> print(topo.get_switches(fields=['isHost']))
                {'s1': False, ...}
        """
```

```
def get_p4switches(self, fields=[]):
        """Retrieves all the P4 switches and their configuration parameters.

        Args:
            fields (list): optional list of :py:class:`str` that specifies which
                           parameters need to be retrieved (and in which order)

        Returns:
            dict: all the switches configuration parameters if ``fields`` is empty, or
            a dictionary of tuples, where the first one is indexed by switch name and
            the second ones collect the values of the selected parameters in the exact
            order they are placed in ``fields``.

        Note:
            If only one element is present in ``fields``, then no tuple is created and
            the value is directly put in the dictionary.

        Example:
            Let us consider the following example to clarify::

                >>> topo = NetworkGraph()
                >>> ...
                >>> print(topo.get_p4switches())
                {'s1':{'isHost': False, 'isSwitch': True, ...}, ...}
                >>> print(topo.get_p4switches(fields=['isHost', 'isSwitch']))
                {'s1': (False, True), ...}
                >>> print(topo.get_p4switches(fields=['isHost']))
                {'s1': False, ...}
        """
```

```
def get_hosts(self, fields=[]):
    """Retrieves all the hosts and their configuration parameters.

    Args:
        fields (list): optional list of :py:class:`str` that specifies which
                       parameters need to be retrieved (and in which order)

    Returns:
        dict: all the hosts configuration parameters if ``fields`` is empty, or
        a dictionary of tuples, where the first one is indexed by host name and
        the second ones collect the values of the selected parameters in the exact
        order they are placed in ``fields``.

    Note:
        If only one element is present in ``fields``, then no tuple is created and
        the value is directly put in the dictionary.

    Example:
        Let us consider the following example to clarify::

            >>> topo = NetworkGraph()
            >>> ...
            >>> print(topo.get_hosts())
            {'h1':{'isHost': True, 'isSwitch': False, ...}, ...}
            >>> print(topo.get_hosts(fields=['isHost', 'isSwitch']))
            {'h1': (True, False), ...}
            >>> print(topo.get_hosts(fields=['isHost']))
            {'h1': True, ...}
    """
```

```
def get_routers(self, fields=[]):
        """Retrieves all the routers and their configuration parameters.

        Args:
            fields (list): optional list of :py:class:`str` that specifies which
                           parameters need to be retrieved (and in which order)

        Returns:
            dict: all the routers configuration parameters if ``fields`` is empty, or
            a dictionary of tuples, where the first one is indexed by router name and
            the second ones collect the values of the selected parameters in the exact
            order they are placed in ``fields``.

        Note:
            If only one element is present in ``fields``, then no tuple is created and
            the value is directly put in the dictionary.

        Example:
            Let us consider the following example to clarify::

                >>> topo = NetworkGraph()
                >>> ...
                >>> print(topo.get_routers())
                {'r1':{'isHost': False, 'isRouter': True, ...}, ...}
                >>> print(topo.get_routers(fields=['isHost', 'isRouter']))
                {'r1': (False, True), ...}
                >>> print(topo.get_routers(fields=['isHost']))
                {'r1': False, ...}
        """
```