最新代码在：<https://gitlab.com/scut-underwater/performance_analysis/tree/test> （test分支）

[分析框架](https://note.youdao.com/ynoteshare1/iframe.html#6689-1547716443237)

[统计指标](https://note.youdao.com/ynoteshare1/iframe.html#6062-1547716482709)

[throughput()](https://note.youdao.com/ynoteshare1/iframe.html#1938-1547714358153)

[ave\_delay()](https://note.youdao.com/ynoteshare1/iframe.html#2044-1547714033893)

[mac\_detail()](https://note.youdao.com/ynoteshare1/iframe.html#7865-1547713979397)

[route\_detail()](https://note.youdao.com/ynoteshare1/iframe.html#8059-1547714992441)

[mac传送率](https://note.youdao.com/ynoteshare1/iframe.html#6635-1547715709413)

[其他工具函数说明：](https://note.youdao.com/ynoteshare1/iframe.html#7977-1547698291856)

**分析框架**

uwan\_performance.py

performance\_detail()：

-> self.find\_route\_table() #得到路由表 routelist, 路由表不为空时进行性能分析

-> self.performance() # 性能分析（实时性能中调用的是这部分得到的结果）

-> performance\_write\_to\_excel() #写入excel表

-> self.draw\_topo() return path\_list # 画出topo图，返回路径列表

-> self.end2end() return sheet\_route\_data # 分析端到端性能，并返回结果（需要path\_list）

-> self.point2point() return sheet\_mac\_data # 分析点到点性能，并返回结果

说明：

实时性能中调用的是 performance()的结果，end2end()和point2point()的结果没有使用

self.route\_data和self.mac\_data保存整个网络运行的性能结果，这里涉及到两种运行模式：

(1) 运行过程中路由表改变：只支持固定时间的路由表改变（比如每30min改变一次，需要在 split\_by\_route()中

设置INTERVAL\_TIME）

所以这里存在的问题是：实时性能分析时不支持路由表改变的情况，当运行过程中路由表变化，则会使端到端性能

分析有问题。

(2) 运行过程中路由表不变：实时分析中将固定路由时间改变的代码注释

# 性能分析

self.performance()：

-> self.throughput(loglist,3) 得到端到端吞吐量 e2e\_throughput

-> self.throughput(loglist,2) 得到点到点吞吐量 p2p\_throughput

-> self.delivery\_rate(loglist) 得到数据包传送率，以及相关的包数量统计，返回result

-> self.run\_time() 得到网络运行时间

-> self.ave\_delay() 得到平均传输时延（route层）

-> self.mac\_detail() MAC层的细节

mac\_data\_delay为mac\_detail中得到的数据包delay

mac\_ack\_delay为mac\_detail中的到的ack包delay

mac\_delay可以选择为它们二者的平均值，也可以选择为mac\_data\_delay，目前用的是后者

-> self.route\_detail() Route层的细节

实时分析中，取了6个指标，分别是：

['MAC-deli\_rate\_without\_broadcast', # delivery\_info['mac\_deli\_rate\_without\_broadcast'] 详见：mac传送率

'MAC-Throughput', # p2p\_throughput 详见：throughput()

'MAC-Delay', # mac\_delay 详见：mac\_detail()

'Route-deli\_rate\_clean', # e2eDeliRateClean 详见：route\_detail()

'Route-Throughput', # e2e\_throughput 详见：throughput()

'Route-Delay'] # aveDelay 详见：ave\_delay()

**统计指标**

**throughput()**

throughput(loglist, layerID)

当layerID==2时，统计点到点吞吐量；当layerID==3，统计端到端的吞吐量

吞吐量计算分成两部分：网络运行时间和收包大小

1. 网络运行时间
   1. 开始时间start\_time为log开始记录的第一条数据
   2. 结束时间end\_time为收到的最后一条数据包
   3. 运行时间为 end\_time - start\_time
2. 收包大小
   1. mac层收包大小：累加满足条件 layerID==2 && ptype==’data' && status==‘recv'的包大小
   2. route层收包大小：累加满足条件 layerID==3 && ptype==’data' && status==‘sendUp'的包大小

**ave\_delay()**

ave\_delay(loglist)

1. for log in loglist:
   1. 记录route层发送的数据包
   2. for log in loglist:
      1. 找到第一个对应的接收数据包的记录（判断条件为：packetID,srcID,destID相同，status=='sendUp',ptype=='data'
      2. 发包时间为a中记录的时间，收包时间为b-i中记录的时间

**mac\_detail()**

mac\_detail(loglist)

1. 将loglist按时间排序
2. 遍历排序后的loglist，以“mSrcID-mDestID:ackSeq"的格式记录收发包（所有包，包括ack、数据包等）。具体为：
   1. layerID==2 && status=='send'或者'resend'，记录为mac层发包，记录在mac\_send\_list
   2. layerID==2 && status=='recv'，记录为mac层收包，记录在 mac\_recv\_list
3. 记录mac层发数据包，具体做法为：
   1. 找到一条发送数据包的记录（条件为：layerID==2 && status=='send' && ptype=='data'） mac\_send\_data\_list
   2. 往后寻找，如果发现满足 layerID==2 && status==’resend' && ptype==‘data'，则将a中记录的第一次发数据包替换成resend 的这一次 （修改条件为： layerID==2 && status==’resend' && ptype==‘data' && ackSeq、mSrcID、mDestID相等）
   3. 往后寻找，如果发现满足 layerID==2 && status==’recv' && ptype==‘data' && ackSeq、mSrcID、mDestID相等，则说明找到了a（或者b）中的数据包收到的记录。计算延时delay，以’mSrcID-mDestID:ackSeq:Delay‘的格式记录在mac\_recv\_data\_list。回到步骤a，再找到下一条发送数据包的记录
4. 单独记录重发的数据包，记录在 mac\_resend\_data\_list
5. 记录mac发送的ack，具体做法为：
   1. 找到一条发送ack的记录（条件为：layerID==2 && status==’send' && ptype==‘ack'）mac\_send\_ack\_list
   2. 往后寻找，如果发现满足 layerID==2 && status==’recv' && ptype==‘ack' && ackSeq、mSrcID、mDestID相等，则说明找到了a中ack包收到的记录，计算延时delay，以’mSrcID-mDestID:ackSeq:Delay‘的格式记录在 mac\_recv\_ack\_list。回到步骤a，在找到下一条发送数据包的记录
6. 单独记录发送的广播包 mac\_send\_b\_list
7. 单独记录收到的广播包 mac\_recv\_b\_list

**route\_detail()**

route\_detail(loglist)

1. 遍历loglist，以格式 ‘nodeID-destID:packetID' 记录为route发包，记录在 route\_send\_list
2. 遍历loglist，如果发现满足 layerID==3 && status=='sendUp' && srcID,destID,packetID相等的记录，则记录在 route\_recv\_list

计算净传送率：e2eDeliRateClean

1. 将route\_detail得到的发包和收包的记录去重（原因：路由层只会发一次，但是mac层有重传机制时，可能会收到两个一样的包）
2. 统计去重后的个数，分别为 route\_send\_clean\_num， route\_recv\_clean\_num
3. 净收包数/净发包数，即为净传送率

**mac传送率**

delivery\_rate(loglist):

mac\_deli\_rate\_without\_broadcast = (float(mac\_recv\_data\_num + mac\_recv\_ack\_num) / float(mac\_send\_data\_num + mac\_send\_ack\_num))\*100.0

mac收到/发送数据包的数量：mac\_recv\_data\_num/mac\_send\_data\_num

mac收到/发送ack包的数据：mac\_recv\_ack\_num/mac\_send\_ack\_num

说明：mac层统计传送率时没有去重，路由层统计传送率时去了重

**其他工具函数说明**

performanceTool.py

str\_to\_list() # 返回包个数，延时

说明：上面的mac\_detail和route\_detail中分别以下面两种格式记录每个包的包id和延时

'mSrcID-mDestID:ackSeq:Delay'

'nodeID-destID:packetID'

str\_to\_list函数则将每个包的id计数得到包个数；将每个包的延时计数取平均值得到平均延时