

网络创新训练 新流量模型下的拥塞控制算法评估

刘般若 计05

1 实验背景

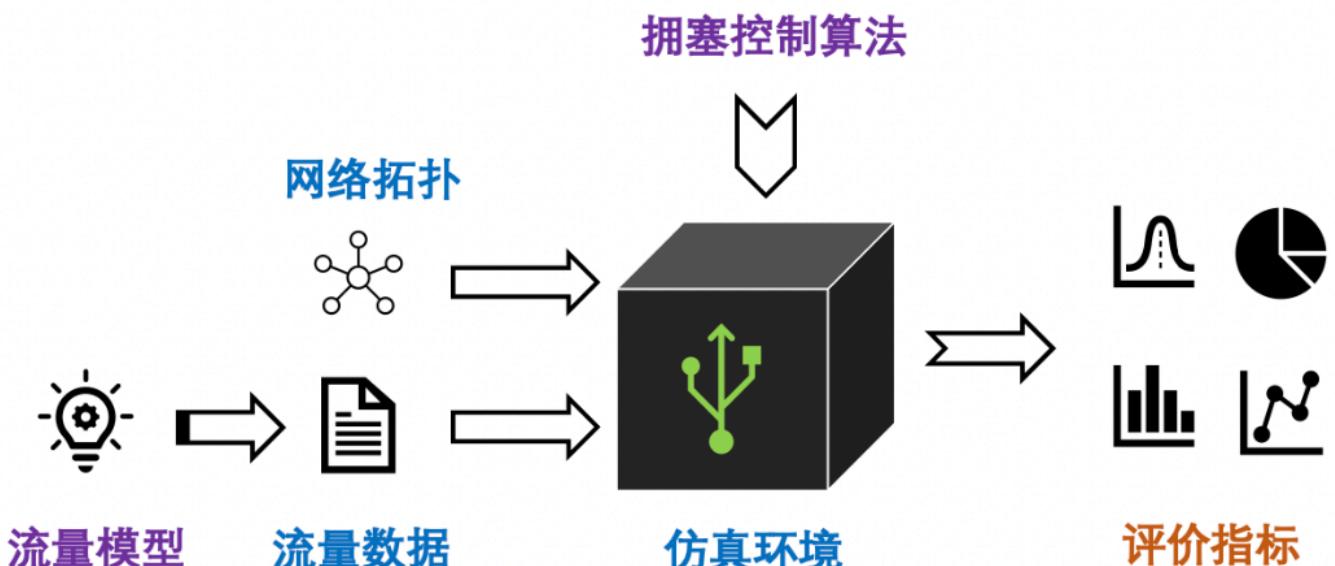
1.1 实验动机

数据中心网络比普通网络追求更高的吞吐和更低的时延。其中拥塞控制是提升性能的重要部分，在近年来也出现了很多CC算法，比如DCTCP，TIMELY，DCQCN，HPCC等。

在这些论文中，评价一个CC的好坏的方法主要是对比其在仿真环境下的一些性能指标，较为常见的是吞吐(throughput),流完成时间(FCT, flow completion time)等。显而易见的是，不同的CC面对不同的流量输入的性能可能会有所差别，而这可能会导致性能指标不同。在目前的论文中，解决方式一般是测试两种极端流量特征：理想均匀和incast（多个源往一个目的地）。

但从常理来讲，实际业务中的流量特征并不是极端到完全均匀情况，也不是incast的情况，而是一种介于两者之间的，有偏的流量特征。就好像我们无法根据函数的一个区间两个端点的取值，就断定其在中间部分的取值一样，我们也无法肯定的根据CC在两种极端流量下的性能，来断言其在中间情况下也能表现的如论文中那么好。

我所做的实验就是为了验证以上观点，我希望在不同的流量模型下测试不同的CC算法，并根据指标进行分析和评价。（虽然仿真结果并不能完全和实机测试相当，但也有一定意义）



2 实验设置

我的实验基于HPCC的论文提供的仿真代码 [<https://github.com/alibaba-edu/High-Precision-Congestion-Control>]，这是因为它除了实现了HPCC之外，也实现了先前提到的其他CC算法，较为容易。

这份代码使用NS-3进行仿真，需要提供流量输入数据以及网络拓扑。

2.1 流量生成模型

在接下来的实验中，我主要对比的是两个模型生成的数据，一个是论文中所使用的均匀数据的模型，即该代码原先所使用的。另一个是由刘志文学长提供的新模型，新流量模型从一个实际业务中提取建模获得，后文将其称作有偏模型。

在后文中，我使用 $\langle \text{src}, \text{dst}, \text{s-time}, \text{size} \rangle$ 四元组表示一条流，含义是源主机，目的主机，流开始时间以及流大小。

2.1.1 均匀模型

这是HPCC论文中数据生成的逻辑：

需要给定的参数有：主机数，链路带宽，平均负载，总时间，经验流大小的CDF

首先根据链路带宽和平均负载计算出一个参数 λ ，这将被用于计算流间隔时间。

使用以下方式生成：

- 枚举每一个源主机，对于每个源主机随机一个目的主机，作为本条流的 $\langle \text{src}, \text{dst} \rangle$
- 同时，对于每个源，维护上一条这个源发出的流的时间 $\text{time}[\text{src}]$
- 从Poisson(λ) 中随机采样，获得一个时间间隔 dt 将此条流的开始时间s-time设为 $\text{time}[\text{src}] + dt$
 - 如果 $\text{time}[\text{src}] + dt$ 超过了总时间，则将此源主机标记成完成
- 从经验流大小的CDF中随机采样，将采样到的大小作为此条流的size
- 不断重复，直到每一个源主机都被标记为完成

可以看到，以上的方法认为四元组相互独立，完全按随机的方法生成。

2.1.2 有偏模型

有偏模型中已经提前设定好了主机数为203，同时给定了一个经验流大小CDF和经验流时间间隔CDF。

需要给定的参数：总时间

使用以下逻辑生成：

- 首先获得若干个源-目的对。
- 对于每个源目的对，不断生成流，直到该源-目的对的时间超过总时间
 - 使用AI模型获取流大小的指标以及流时间间隔的指标，并从对应的CDF中采样。
- 直到所有源-目的对都生成完毕。

2.1.3 控制变量的方法

可以看到，两种模型的生成方法有一定不同，为了进行对比实验，我需要控制变量，保证在统计意义上两种流量模型生成的数据是一致的。为了控制变量，我采取了以下措施：

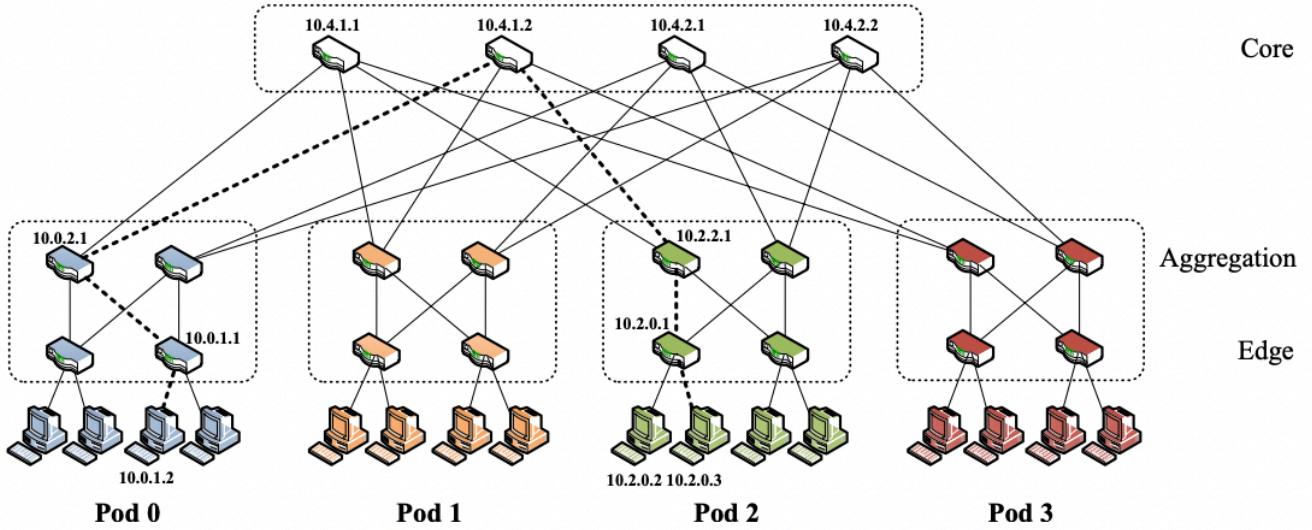
- 保证生成时总时间相同
- 由于有偏模型不能更改主机数，我将均匀模型的主机数设定为203。这项设定对后面的拓扑结构有影响

- 对于均匀模型中的 λ ，它是由带宽计算而来。我起初希望用有偏模型的时间间隔CDF数值积分的均值代替，但是后来发现两者在数量级上有差异。经过与刘志文学长的讨论，采用先对CDF进行放缩，然后确保其均值等于 λ ，具体而言：
 - 提前根据链路带宽和负载计算出 λ
 - 对CDF进行放缩，直到其均值为 λ 。对于时间间隔较大的部分，直接进行线性放缩，对于流较小的部分，使用一个二次函数进行光滑处理。
 - 由于两者的含义都是平均的流间隔，我这样做保证了在统计意义上的流间隔相同。
- 均匀模型使用与有偏模型相同的流大小CDF

可以看出，由于有偏模型的生成方式不同，控制变量有一定困难。但是经过与学长的讨论，这种方法是较为合理的。

2.2 网络拓扑

实验使用的是Fat Tree拓扑，其大小由参数 K 控制。一个 $K = 2$ 的示意图如下。



由于流量模型生成要求主机数为203，而Fat Tree拓扑所支持的主机数并不是整的，因此，我选择最小的满足其主机数大于203的参数 K ，这里为10。

对于 $K = 10$ 而言，其有10个pod，每个pod内有25个主机，我选择把203个主机放在前9个pod中，第9个pod未满，而前8个pod均为满载。

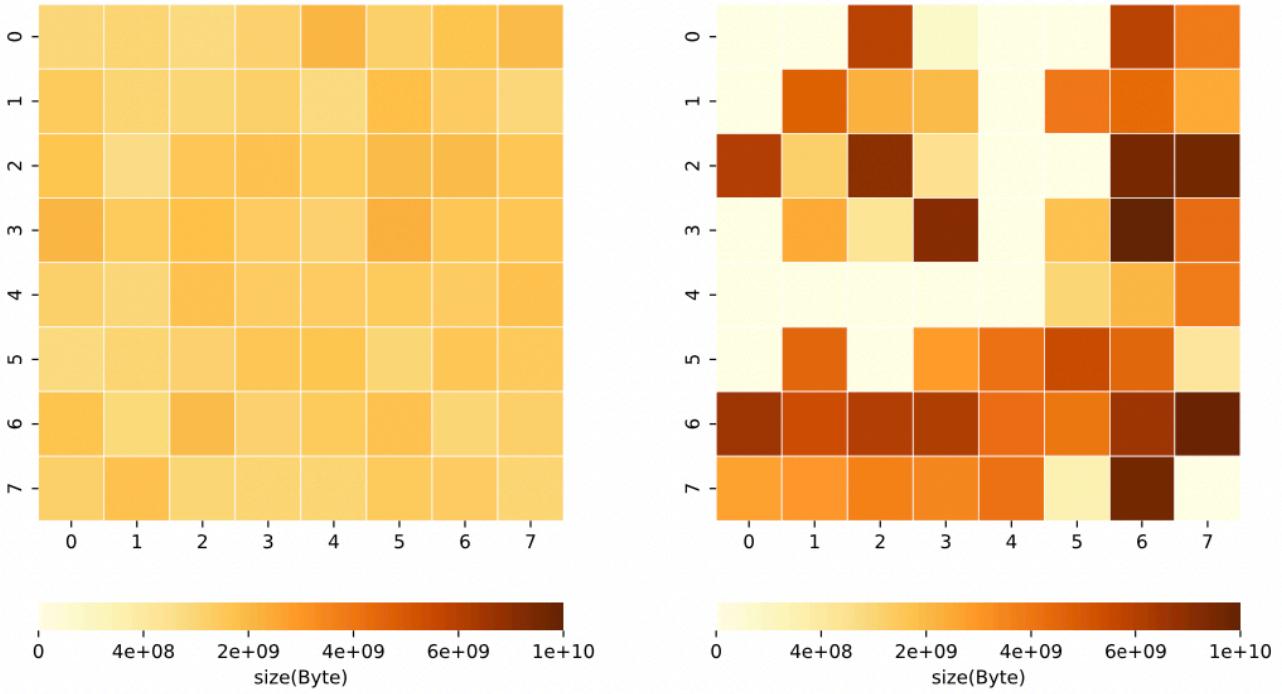
2.3 数据示意

这里设定所有链路带宽为100Gbps，无错传输。交换机基础延迟为0.001ms。

最终生成数据的统计意义下100Gbps 30% 荷载相同。

使用两种模型生成后，流量分布如下，左侧为均匀模型生成的流量，右侧为有偏模型生成的流量。

下图中以邻接矩阵的形式给出了流量大小， (i, j) 格子表示pod i 向 pod j 发出的流量。正如前文所提到的，只有8个pod满载，所以这里只画出了8*8



3 实验结果

在整体上，我的实验内容在于对比两种流量模型：一种是论文中使用的均匀模型，一种是由刘志文学长提供的有偏模型，两种流量模型的区别会在后文详述。

我所进行的具体实验内容由以下三部分组成。

- 学习不同CC算法，学习NS3仿真使用，复现HPCC中论文的结果。这部分工作没有什么创新性，因此不再赘述。
- 详细对比了HPCC算法在两种流量模型下的差异，对一些较细粒度的结果进行了分析。
- 对比了DCTCP，TIMELY，DCQCN，HPCC四种算法在两种流量下的差异。

我们测试的主要指标为：

- 流完成时间(FCT, flow completion time)
- FCT Slow Down: $\frac{\text{实测FCT}}{\text{理论FCT}}$
 - 理论FCT为流大小/带宽
 - 排除了流大小的影响，起到类似标准化的作用

除此之外，我还测试了吞吐，带宽利用率，队列长度，PFC频率等指标，但由于时间原因，没有进行深度分析。

3.1 对HPCC的测试

3.1.1 总体FCT配对比较

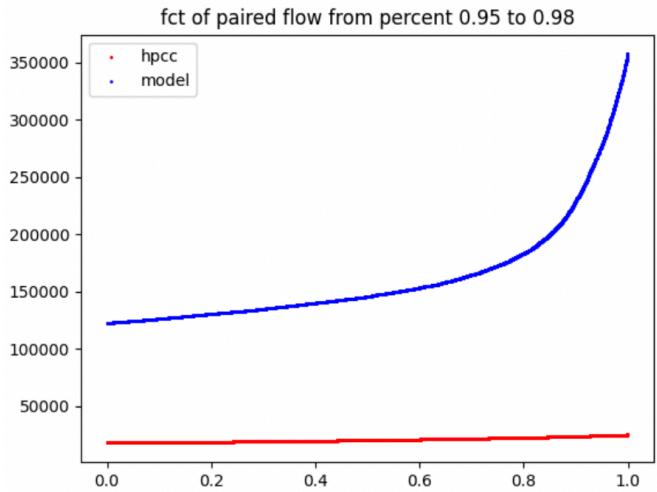
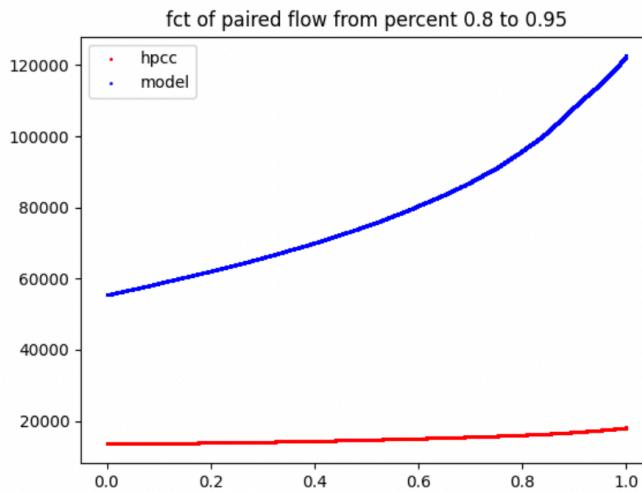
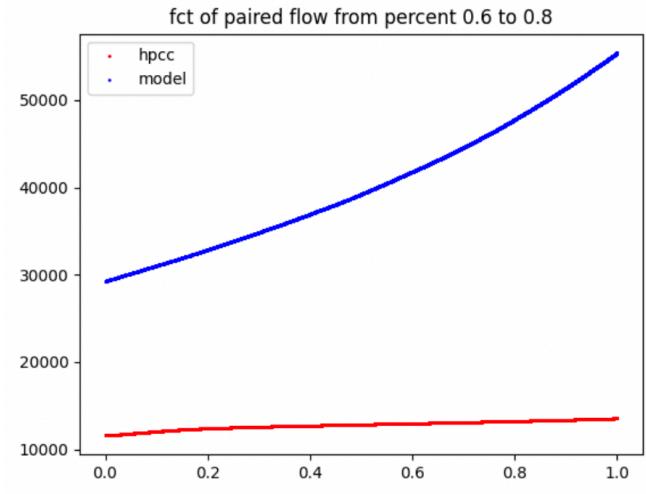
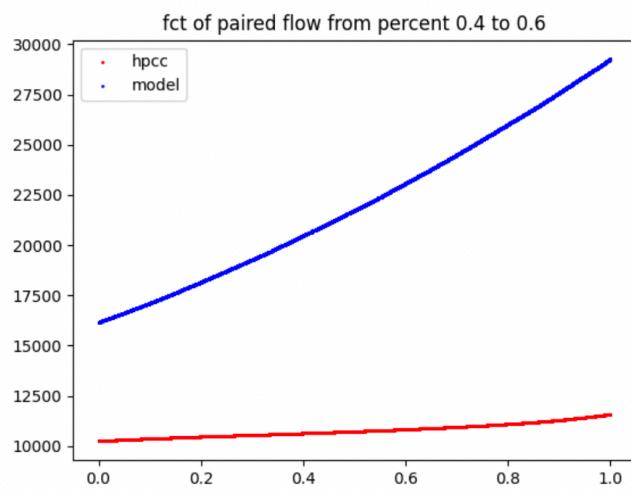
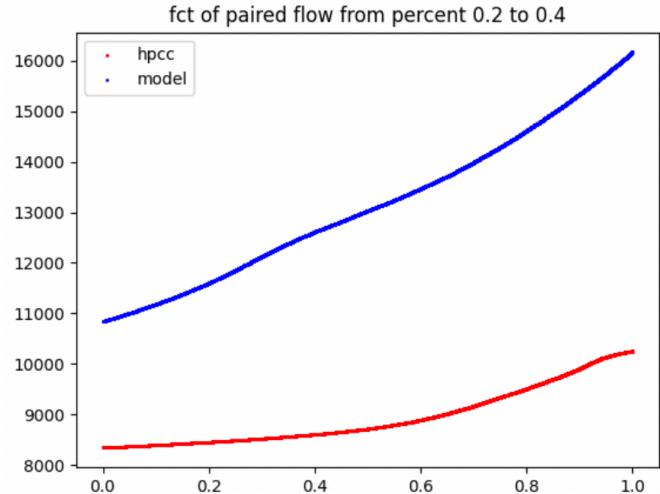
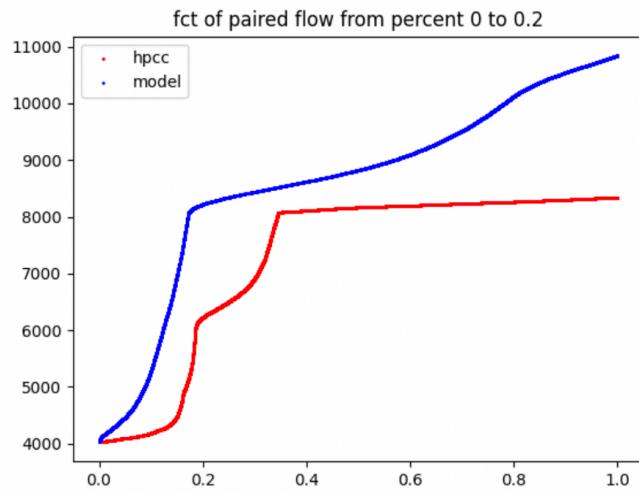
为了获得一个宏观上的理解，尝试计算出均匀流量和有偏流量下的FCT，并且将其排序，然后配对比较。由于流数较多，我将FCT从小到大分为6部分给出。

每个图的标题表示了其在总流中fct的占比，比如0.2to0.4表示这是将排名20%到40%的fct配对比较。

蓝色表示有偏模型数据的结果，红色表示均匀模型下的结果。

x轴表示内部排序，如果标题是0.2to0.4，且x轴为0.4，那实际上表示的是排名为 $20\% + 20\% * 0.4 = 28\%$ 的fct。

y轴表示时间，单位为ns



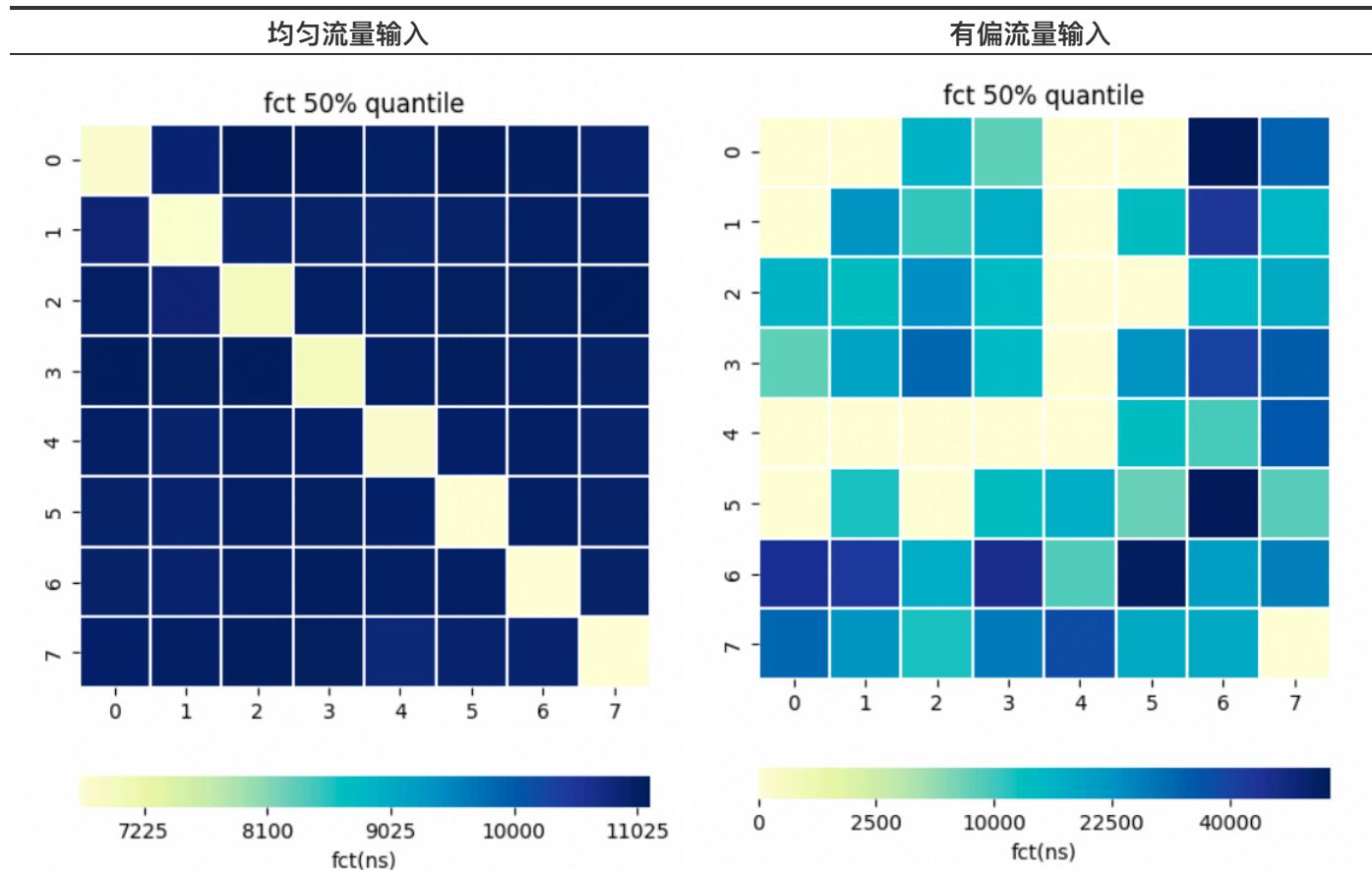
从图中可以看出，在有偏模型下，普遍FCT增长3~5倍。在短流(0~0.2)部分差距不明显，但随着流FCT增长，差距愈发明显。

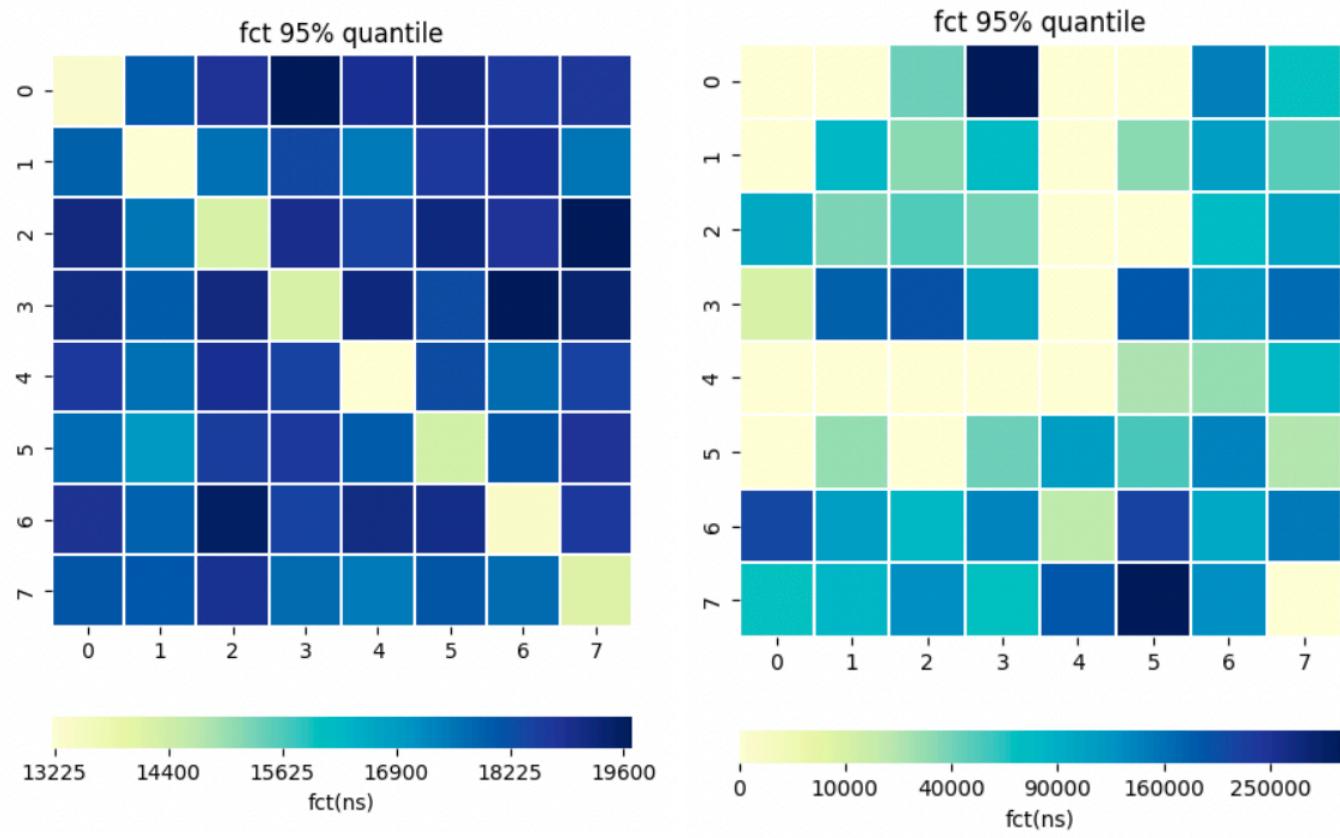
也就是说，从总体来看，有偏模型显著增大了FCT，带了拥塞。

3.1.2 FCT的空间分布

为了讨论拥塞形成的原因，我们给出FCT随空间的分布。

还是以邻接矩阵的形式给出。





分析结果：

均匀流量输入下的FCT分布符合预期。由于拓扑的高度对称，FCT的差异只体现在pod内流以及pod间流。而pod内不需要经过核心路由器，从而FCT较小，体现在图中就是对角线上的颜色较浅且一致，其他部分颜色较深且一致。

有偏流量输入下的FCT也符合预期，因为本身流量的分布具有空间特征，FCT的分布也体现出了空间特征。并且与2.3中的图对比，我们可以看到绝大多数pod都满足“流量越多，FCT越大”的趋势。除去少数几个对，比如(0,3), (7,5)等。这些pod的FCT较为反常，值得进一步分析。

因此，我固定源-目的对，进一步分析。

经过分析表明，这些反常的原因在流的数量太少，导致存在一些不均匀分配的情况。

比如 (0,3) 只有20条流，且其存在一条长度约为50m的大流，其余都是小流，这就体现出其95%分位点很大，但是总流量很少，50%分位点很小的原因。

比如 (7,5) 只有277条流，且其存在一条长度约为70m的大流，其余都是小流，这就体现出其95%分位点很大，但是总流量很少，50%分位点很小的原因。

对于其他的源-目的对进行分析，得到类似的结果，因此我们可以认为，流量分布的不均匀导致了拥塞，而流量越大则拥塞越严重，体现在FCT上。

3.1.3 FCT slow down 的空间分布

由于单纯FCT本身受到流大小的影响，使用FCT slow down 验证先前的结论。

这里是95%分位点的FCT slow down。

o -	1.6	1.4	1.5	1.5	1.5	1.5	1.5	1.5
↔ -	1.4	1.5	1.4	1.4	1.4	1.5	1.5	1.4
↷ -	1.5	1.5	1.7	1.5	1.5	1.5	1.5	1.6
↶ -	1.5	1.4	1.5	1.7	1.5	1.5	1.6	1.6
↶ -	1.5	1.4	1.5	1.5	1.5	1.5	1.5	1.5
↶ -	1.4	1.4	1.5	1.5	1.5	1.6	1.5	1.5
↶ -	1.5	1.4	1.6	1.5	1.5	1.5	1.6	1.5
↷ -	1.5	1.4	1.5	1.5	1.4	1.4	1.4	1.7



o -	0	0	3.9	1.4	0	0	11	5.1
↔ -	0	8.4	3.2	6.2	0	2.9	8.8	3.9
↷ -	8.4	3.4	5.9	3.6	0	0	6.6	9.3
↶ -	1.1	13	18	14	0	17	10	12
↶ -	0	0	0	0	0	1.9	3	6
↶ -	0	2.7	0	4	8.8	9.9	11	2.5
↶ -	16	8.6	6.9	11	1.9	17	18	12
↷ -	5.1	6.7	14	5.2	14	23	10	0



左图是均匀模型下的结果，可以看出，当流量较为均匀的时候，每对pod之间的拥塞情况也是大致均匀的，拥塞并不明显。而对比右图，在有偏的模型下，我们可以看到除去本身就没有流的地方外，基本上都要比均匀模型下的延迟更高。而这表明有偏的流量空间分布带来了更严重的拥塞。

可以看出，这验证了3.1.2的结论。

3.1.4 实例分析

我们希望能够得到这种拥塞对具体原因，而不是仅仅“流量多，就拥塞”这种显然的结论。

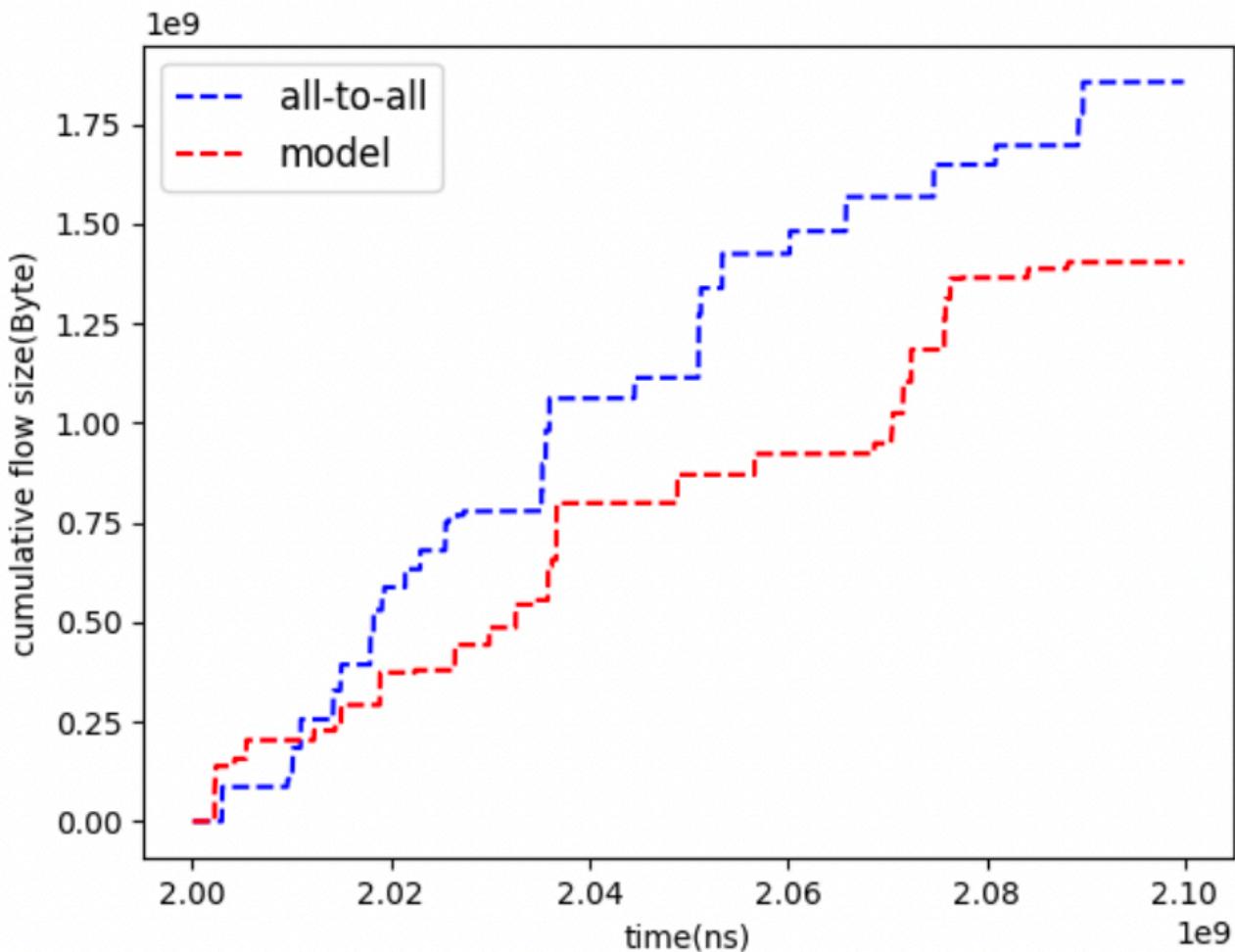
考虑(3,5)作为一个典型例子。

Pod3-Pod5	均匀输入	有偏输入
流量	1.77GB	1.34GB
50%FCTSD	1.0	2.2
95%FCTSD	1.5	16.9
出口带宽占比	9.8%	3.4%

可以看到，从流量来讲，在均匀模型中流量要比有偏模型多一些。然而从FCT slow down来看，有偏流量下显著的比均匀流量大。

我们分析，造成这种反常有两种可能

- 在时间上存在不均匀，即在某个很短的时间内存在流量的突发导致拥塞。但是下图画出了随时间变化的累积发出流量，蓝色表示均匀模型，红色表示有偏模型。可以看到两种流量输入随时间的变化是类似的，虽然存在一些跳变，但是并没有极端的跳变。这说明FCT slow down的差异不是由时间上的有偏导致的。



- 另一种可能是出口带宽被挤占。正如先前提到的，如果将fat tree 拓扑的pod看成一个点，由于其所有发往其他pod的流都要经过核心交换机，那么此时该pod通往核心路由器的带宽就会被挤占，导致最终的拥塞。从数据可以看出，在均匀模型下，3到5的流可以获得10%的带宽，而在有偏模型下，由于3发往其他pod的流量更多，它能够抢占的带宽只有3%，最终导致FCT显著增大。

因为我们在统计意义上控制了两种流量数据，所以流量总量应当是在同一量级的，此时我认为3.1.1中普遍的FCT增加是由于空间有偏，导致出口带宽相互挤占，最终导致的拥塞。

3.2 多个CC对比

我们对比了DCTCP, TIMELY, DCQCN, HPCC四种CC算法。同时，论文中还提供了DCQCN+win, TIMELY+win两个版本。

这里+win指的是在原本的算法下额外增加了一个窗口限制inflight bytes，即发出但没有收到ack的流量，希望能够减少PFC的出现，后面实验证明，这种控制是关键的。

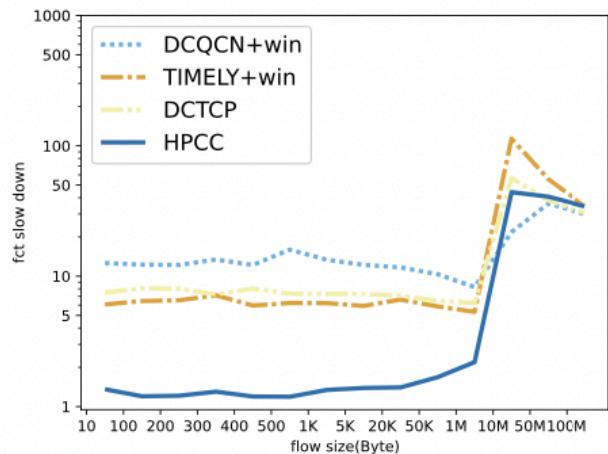
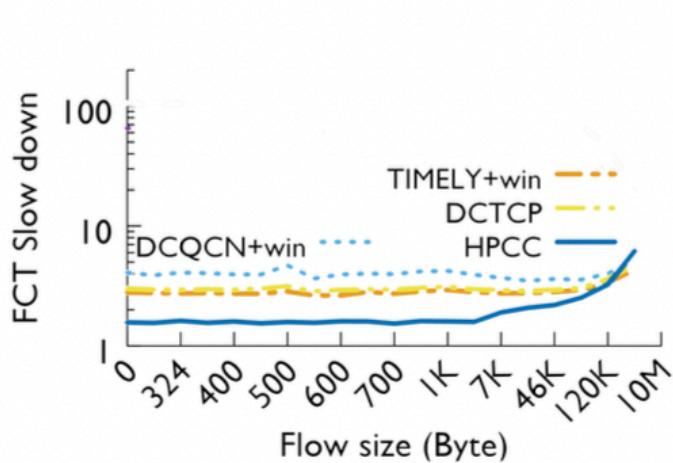
在测试时，我没有更改参数，均使用论文中测试使用的参数，这是为了控制变量。

3.2.1 有偏流量下的FCT slow down

我们测试了4种CC在均匀流量下95%分位点的FCT slow down。

左侧图摘自HPCC论文，右侧图是我的实验结果。横坐标为流大小，纵坐标为FCT slow down

均使用均匀流量模型生成数据，但是具体参数不同。



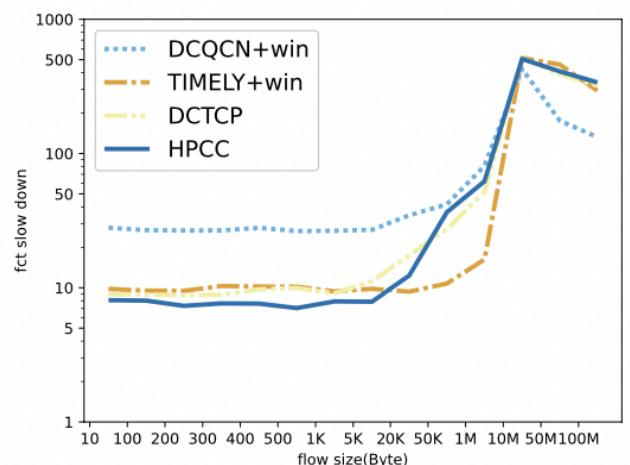
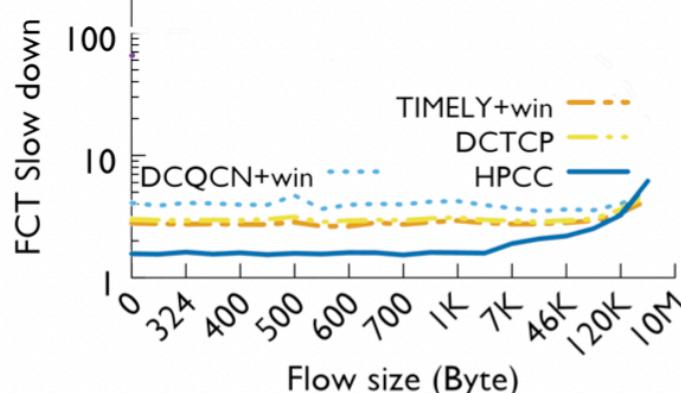
从上图可以看出，虽然我实验中的数据与HPCC论文中的数据不同，同时网络拓扑也略有不同（均使用fat tree，但是带宽设置和节点数不同）但由于均采用同样的模型生成流量，这导致最后的结果虽然在数值上有差异，但是在不同CC对比下，排序相同，并且CC间的差距也类似。

有一点值得指出的是，论文中流大小只到 $10m$ 就结束了，而从我测试的结果来看，当流大小显著增大时，HPCC的优势不再明显，甚至不如DCQCN+win。原论文中并没有测试到这部分。

这表明我所进行的实验是正确的，并且，如果在更换流量模型后不同CC表现出了显著差异，由于我们控制了流量数据的统计参数，此时就可以认为是由于流量特征的不同导致了结果的不同。

3.2.2 有偏流量下的FCT slow down

左侧图不变，右侧图是在有偏流量数据下的结果。



可以看出，流量特征改变后不同CC之间的相对顺序产生了明显差异。对于较小的流，HPCC不再有均匀流量下近乎于1的性能（近乎于1表明其已经达到了理论上限，不存在拥塞），并且当流大小变大后，其也已经不再是最优的CC。

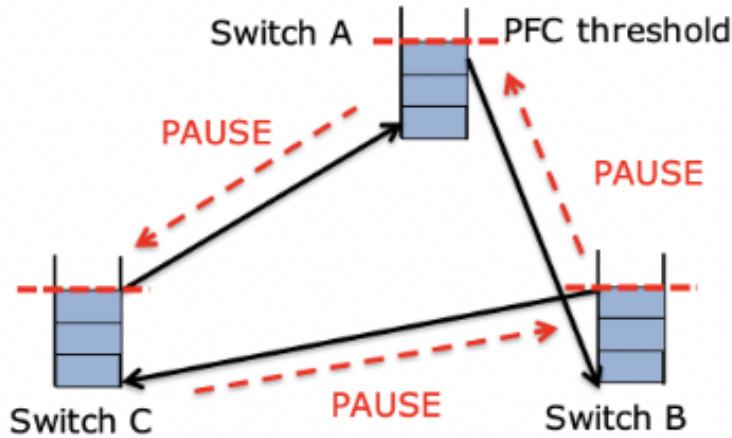
这表明，论文在均匀模型下测试的结果在不同的流量特征下不具有普适性。

3.2.3 朴素的DCQCN和TIMELY

注意，在以上对比中我没有测试朴素的DCQCN和TIMELY，这是因为这两种算法无法在有偏的数据下完成。

PFC指的是交换机当队列长度超过限制后，会向上游发送一个pause消息（在NS-3代码中，其直接广播pause），此时接到消息后就停止发包，然后队列长度恢复后会发送resume消息，允许继续发包。

但是如果依赖成环，PFC就会造成死锁，如下图：



朴素的DCQCN和TIMELY在有偏模型下，不断的触发PFC（如timely，经过测试发送了至少20w次PFC包），最终导致其无法在充分长的时间内完成。我在后台跑了超过24个小时，其时间超过了ns-3设定的time out自动退出了，但还是没有完成。

我跟踪了一下发出PFC包的交换机，看到最后一个包是pause，因此我怀疑是产生了死锁。即便没有死锁，也因为PFC路径过长导致无法完成有偏流量，最后的结果也没有意义。

```

471280    PFC send resume337
471281    PFC send pause337
471282    PFC send resume337
471283    PFC send pause337
471284    PFC send resume337
471285    PFC send pause337
471286    PFC send resume337
471287    PFC send pause337
471288    finish time elapsed: 104161

```

3.2.4 吞吐和带宽利用率

我还测试了吞吐和带宽利用率。

算法	总吞吐		带宽利用率	
	传统模型	新模型	传统模型	新模型
DCTCP	2175Gbps	795Gbps	30.14%	22.40%
TIMELY	1879Gbps	N/A	16.25%	N/A
TIMELY+WIN	2057Gbps	785Gbps	30.40%	20.87%
DCQCN	2085Gbps	N/A	21.62%	N/A
DCQCN+WIN	2165Gbps	1236Gbps	28.69%	20.30%
HPCC	1937Gbps	743Gbps	25.52%	19.59%

4 结论和讨论

4.1 流量特征对于CC表现的影响

- 在不同的流量模型下，CC的表现不同。
- 这是因为不同的CC存在设计上的trade-off，比如HPCC牺牲了吞吐来换取面对incast的稳定性。这里也在3.2.4中验证。
- 对于本实验中新模型的流量特征，由于数据有偏，需要额外控制。因此像DCTCP，HPCC等通过窗口限制在途数据的算法表现优秀，而朴素的DCQCN和TIMELY甚至无法完成。
- 由于学长所提供的流量模型是从实际业务中提取得来。不难推测，对于不同业务的流量特征，这些算法的表

现又会有所不同。

- 更进一步的，我认为一致最优的算法很难找到，需要根据流量特征调整算法中的参数，在有必要时针对目前业务的流量特征设计新的算法。

4.2 拥塞控制算法的选择

不同的算法部署的复杂度不同，这一方面体现在参数数量上，一方面体现在对于硬件的要求上。

- DCTCP, 2010
 - 只需要ECN，两个主要参数
- TIMELY, 2015
 - 需要准确的RTT时间测量，四个主要参数
- DCQCN, 2015
 - 需要硬件支持ECN和RED，约15个参数
- HPCC, 2019
 - 需要INT，三个参数

对于HPCC这种较新的算法，有一些硬件需求没有达到，需要自行对硬件进行修改，这无疑增大了部署成本。同时，我们从3.2.2中可以看出，HPCC在有偏流量下的FCT slow down的表现和DCTCP类似，而DCTCP已经内嵌Linux Kernel中，部署简单。此时数据中心应当进行相应的权衡。

4.3 下一步工作

- 使用不同的拓扑进行测试
- 调整CC参数，对比各个CC在最好参数下的表现
- 对于流的时间特征，队列长度等细粒度指标进行分析
- 考虑流量数据变动后，对其他方面如负载均衡等的可能变动

5 关于代码

我使用的代码基于HPCC论文的代码，链接如下（但是目前是private的，可能需要助教向刘志文学长或者我要一下权限）

<https://git.tsinghua.edu.cn/liubr19/network-advanced>

总体代码约2.5k行，绝大多数是python+shell脚本，C++代码不超过500行，主要是进行Debug和输出需要的信息，更改的内容并不多。下面我简单说明一下复现实验结果的过程

5.1 实验环境

需要g++4.8，要不然ns3没法编译。需要python2运行论文自带的python脚本。

复现我的实验结果（主要是画图）需要python3

5.2 目录框架

/analysis 是hpcc论文自己分析数据用的，没有使用

/model 是刘志文学长提供的有偏流量模型生成部分，其中generate2.py是我用来生成有偏模型的脚本，从generate.py更改而来。其余没有直接使用

/traffic_gen 是hpcc论文中生成均匀流量模型的部分，其中real_traffic_gen.py是我用来生成的脚本，从traffic_gen.py 更改而来。注意，需要将/model/stas/cdf_size.csv 转换成合适的cdf。

/my 是一些分析数据画图的脚本和shell自动跑的脚本

/simulation 是实际进行仿真的部分。/simulation/exp 中需要放入流量数据，拓扑，config文件等等，最后跑出来的结果fct，qlen等也会放在这里。/simulation/scratch/third.cc 是每次仿真的程序入口，是真正的main

我对于third.cc 以及 /simulation/src 中所有仿真代码的修改都用 // begin lbr 开头，可以搜到

5.3 复现流程

5.3.1 运行仿真

```
1 cd my
2 python3 gen_topo.py      # 生成拓扑， 默认带宽100g, K=10
3 mv topology ../simulation/exp/fat250.topo #将拓扑放在exp里面， 改名为fat250.topo
4 bash ./gen.sh # 生成数据， 其中有些写死的参数是为了保证统计意义上负载和相同
               # 生成的数据放在了 ./data/model.data ./data/hpcc.data
               # 前者是有偏， 后者是均匀
5 cd ../simulation
6 bash ./run.sh num # 运行hpcc算法， 最后结果会放在 /my/result/hpcc.traffic$num 和
                     /model.traffic$num 分别表示有偏模型和均匀模型
7 bash ./run_other.sh #运行多种CC， 具体见sh中ccs数组， 具体CC的命名请参考
                     ./simulation/myrun.py 中的一些参数， 结果存放于 run.sh 类似
```

需要若干个小时来运行得到结果。

运行时会输出一些.log文件到/simulation,用于调试等，可以自行调整。

5.3.2 分析数据

具体需要看一下代码，这里只列出文件和结果的对应关系。

```
1 cd my
2 python3 case_study.py num # 3.1.4
3 python3 cc2.py num=1 # 3.2.1 3.2.2
4 python3 draw_paired_fct.py num # 3.1.1
5 python3 fct_matrix.py num # 3.1.3
6 python3 throughput.py # 3.2.4
7 python3 traffic_matrix.py num # 3.1.2
8 python3 nodewise_analysis.py # 可以画出固定src, dst的fct分位图， 具体使用请见代码
```

这些代码本质都是读取 /my/result/xxx.traffic\$cc\$num/ 里面某些东西，如果运行错误时可以看下是否文件位置正确。

默认会在 /my/draw 里面生成图片，或者在命令行输出数据。

还有一些没有提到的文件，可以自行参考使用。

6 参考文献

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., ... & Sridharan, M. (2010, August). Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference* (pp. 63-74).

Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., ... & Zhang, M. (2015). Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review*, 45(4), 523-536.

Mittal, R., Lam, V. T., Dukkipati, N., Blem, E., Wassel, H., Ghobadi, M., ... & Zats, D. (2015). TIMELY: RTT-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4), 537-550.

Li, Y., Miao, R., Liu, H. H., Zhuang, Y., Feng, F., Tang, L., ... & Yu, M. (2019). HPCC: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication* (pp. 44-58).

Al-Fares, M., Loukissas, A., & Vahdat, A. (2008). A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4), 63-74.

Benson, T., Akella, A., & Maltz, D. A. (2010, November). Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (pp. 267-280).