**Fast Simulation of Communication Networks:**

**Collaborated Research with Huawei Theory Lab**

Area: Operations research, stochastic simulation, 5G network

Target: Develop **fast simulation algorithms** to evaluate **tail probability and percentile of delay**, known as service level agreement (SLA) in communication networks.

Tools: queueing models (simulator), importance sampling techniques (algorithm design)

Requirements:

* Implement simulation algorithms and carry out numerical experiments
* Derive simple theoretical results
* Write formal research reports
* Participate in group seminars and discussions
* Understand queueing models, good at programming and simulation techniques
* **Keep motivating yourself**

2.25

现在做到了什么地步？为什么要做这个？这是华为合作的一年期的工程，华为给的有simulator的模版，然后我们相当于在模版的一个叫ns的子区域去contribute，最后会变成一个子模块纳入他们的整个体系。他们的模版有完整的simulation框架，但是他们的问题就是不够accurate，不够fast，因为是直接暴力模拟蒙特卡洛，所以relative error会比较大，然而5G时代需求高速的模拟，准确的模拟。我们的目的就是使用各种technique减少relative error。当然现在主要关注的是importance sampling。

现在有直接的框架实现event based simulation，然后现在已经做的比较好的是简单MM1，以及一系列tandem MM1的情况，就是各种串联的，因为此时整个体系还是在**Markovian**的体系下进行的，有markov的背景就可以用什么markov chain monte carlo一类的东西，所以整体会容易一些。但是在现实生活中这种Markovian的假设基本就不存在，所以我们需要使用各类simulation technique去解决更复杂的模型，学长有提到先做non-preemptive priority的，然后往后会做一些比如**WFQ，DRR**的routing rule下的结果，最后可能会连接起来实现一个复杂的network。

目前的literature情况是基本上只有关于很简单的M/M/1的还有一些简单串联的tandem的模拟，像priority queue基本就没有，最多只有preemptive的priority queue，而non-preemptive的基本没有见到。这方面相关的literature已经很多年都处于停滞的阶段了，所以也是一个很cutting-edge的领域，而且是算是学界与工业界合作并行的一个领域，学术的研究真正的对业界产生影响，所以整个工作是很有价值的。

**我们的工作：**

1. **derive一些basic theoretical formula；**
2. **去开发尝试更加复杂模型的simulation algorithm；**
3. **去做performance optimization，也就是算法的优化设计**。

3.2

一些问题：

1. 为什么参数threshold必须对每个type要递增呢？
2. Mix\_p这个变量的目的是什么？
3. 这个reasons到底是什么东西？延伸到多类的时候这个reasons怎么弄？我记得之前print出来reasons是一大堆的二元组，而且全都是-1,0,1的数值，应该是每一次simulation都会产生一个reasons。那高维度怎么办？
4. 跑simulation出来的pid是什么东西？那个数字有意义吗，蓝色的数字，以及为什么后面那个数字永远都是1？那个simulation里面%10000 == 1的语句有什么意义吗？
5. 不同type的事件的change of measure之间是互相关联的吗？？这个里面policy=0, 1, 2有什么区别？我simulate出来的结果完全不一样！这个change of measure的参数怎么计算？这个real，real\_lower这种东西怎么算，公式是什么？

**Change of measure不会改变均值的期望，这本就是importance sampling的定义。Change of measure是减小方差的technique，而regenerative method是模拟稳态的方法。regenerative得到的estimator是biased，但是是strongly consistent。**

3.10

这里有个很巧的点：

我们之前自己写event simulation的时候，基本上都是waiting time到realize之后再去change of measure，因为只有realize之后才能判断是否大于那个临界值，从而决定是否要change of measure。但是这里的代码写的是还没有realize的时候，就先去做一个waiting time的预判，然后借此决定是否要change measure，所谓预判就是看看当前队列里面优先级比他高的包和优先级一样但是在前面的包以及当前正在serve而不能打断的包。

但是这样有个问题，就是这样预判的只是一个lower bound，因为忽略了在该包wait的过程当中后面来的优先级更高的包从而导致的waiting time的增加。那为什么要提前做预判，然后用这个lower bound的预判和gamma比较是否要change measure呢？学长说的是：

“如果之后再switch 会积累一堆change measure后的包 这些包贡献的likelihood会使得方差变大。大概high level就是说 change of measure 付出的代价是likelihood；但是得到的收益是event 不再rare 需要权衡二者；所以最好的权衡就是 刚刚能发生事件 就不再change 这样能使得后面额外的代价更小。likelihood其实是代价 （如果change of measure做得不好 其实方差也是会增大的。”

3.11

别忘了每轮cycle结束的依据是整个系统regenerate，因此等到出现了waiting time是零的情况就会停止，而不是说非要出现了target event，也就是超过了threshold才会停止。因此经常看到cycle里面reasons是-1的情况，此时对应的cycle\_sum\_probs一定是0，因为没有出现target event。

我好像突然懂了他那个policy是什么意思，policy有三种，在代码里为了方便是0,1,2，在pdf里面是1,2,3，这个三个policy是三种不同的change measure的办法。

第一种就是当且仅当目前的packet的type是我们的target type，也就是程序里面的stop的参数（stop by which type），那我们才会去预测他的lower bound从而决定是否要change measure。

第二种则是除了上面的packet的type就是target type的情况，还考虑了另一种可以change measure的情况，那就是如果目前的packet的type不是target type，而是比target type优先级更高的type，那此时我们的target type在目前这个packet的后面，他们相应的predicted waiting time也会变长（注意此时这个packet还没有serve，仅仅是刚刚arrive！！我们所有的waiting time都是在serve之前刚arrive就开始估计的lower bound），我们只考察在目前这个packet 后面当中type是我们target type的那些包，对那些包我们重新估计他们的waiting time（因为此时会延长）。

然后第二种change of measure是如果这些后面的第一个target type的reestimated waiting time (lower bound) 超过了threshold我们就提前把对应的phase从1改成0。

而第三种change of measure是如果后面全部的target type的reestimated waiting time中做一个for循环，只要有一个的estimated lower bound超过了threshold，那我们就把phase从1改成0。

一些想法：

总体给我的感觉就是很hurry，整个程序想尽快的去change measure回原来的measure，而其中显然是policy 3要快于policy 2要快于policy 1。当然这些效果也不一样，未必policy3就是最好的，policy1也未必。**不过既然guiyu说到change measure晚了会导致积累了一大把没有change measure的包导致贡献的方差变大，那我猜测会不会在K变大，也就是type比较多的时候，change measure应该发生的越快越好？相当于对于比如K=4，那么policy3的表现会不会比policy1要好？**

**然后广进给了一个问题：不同的policy是否会影响likelihood ratio的计算？**

**Likelihood ratio究竟是什么？**

两类的时候，likelihood是两序列joint probability的比值，而每个type of packet产生的interarrival time和service time都是iid的，因此joint probability是前面所有人的对应interarrival time, service time的乘积。

High level来讲，第n个packet的waiting time能否超过threshold，是有前面所有的packet的arrival和service序列共同决定的，是一个recursively defined的过程。

别忘了定义，在shelton ross的书里讲importance sampling，里面likelihood的定义是f(x)/g(x),是两个density的比值，那个是一元的情况，那现在变成一个序列就是density乘积的比值了。很好理解。那K个type也同样是这样的乘积的比值，然后在change measure结束之后，phase变成0，相当于likelihood ratio变成了1，就没有再change measure了。

我傻逼了，cross entropy里面每次更改test\_flow参数就是在对不同的队列在考察去得到CE下的参数，因此在做unbiased check的时候，除了把tilt的参数改成要测试的那个priority的class的CE算出来的参数，同时还要记得把里面test\_flow的参数改成对应要测试的那个参数！！！比如八队列，我们要针对其中的第四队列的tail probability在threshold为10的情况下进行测试，那么就应该在cross entropy里面把test\_flow设置为10然后跑出参数，然后，copy对应的参数到unbiased check，然后在那个里面改动test flow也是4，也就是做check的对象，然后跑。同时按理讲，unbiased check和cross entropy的policy应该对应上。

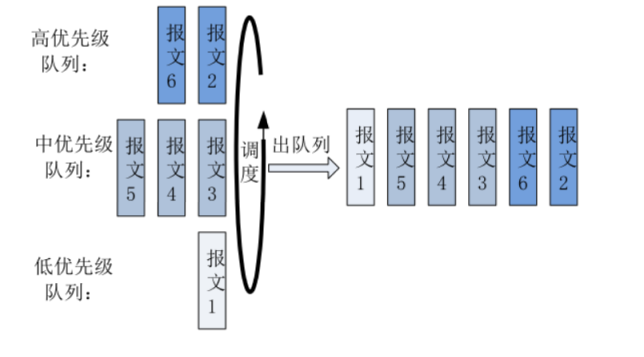
**QoS (Quality of Service) in Networking**

服务质量QoS指的是在[分组交换](https://zh.wikipedia.org/wiki/%E5%88%86%E7%BB%84%E4%BA%A4%E6%8D%A2" \o "分组交换)网络领域中指网络满足给定业务合约的[几率](https://zh.wikipedia.org/wiki/%E6%A9%9F%E7%8E%87" \o "几率)；或在许多情况下，非正式地指分组在网络中两点间通过的几率。QoS是一种控制机制，它提供了针对不同用户或者不同数据流采用相应不同的优先级，或者是根据应用程序的要求，保证数据流的性能达到一定的水准。QoS的保证对于容量有限的网络来说是十分重要的，特别是对于流[多媒体](https://zh.wikipedia.org/wiki/%E5%A4%9A%E5%AA%92%E4%BD%93" \o "多媒体)应用，例如[VoIP](https://zh.wikipedia.org/wiki/VoIP" \o "VoIP)和[IPTV](https://zh.wikipedia.org/wiki/IPTV" \o "IPTV)等，因为这些应用常常需要固定的传输率，对延迟也比较敏感。

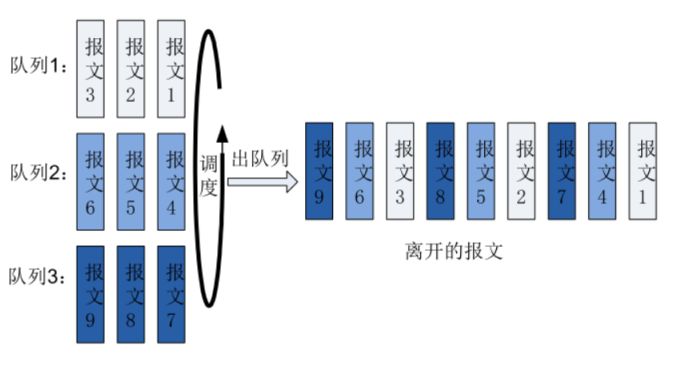
带宽（bandwidth）：对于[数字信号](https://zh.m.wikipedia.org/wiki/%E6%95%B0%E5%AD%97%E4%BF%A1%E5%8F%B7" \o "数字信号)而言，带宽是指单位时间内链路能够通过的数据量。

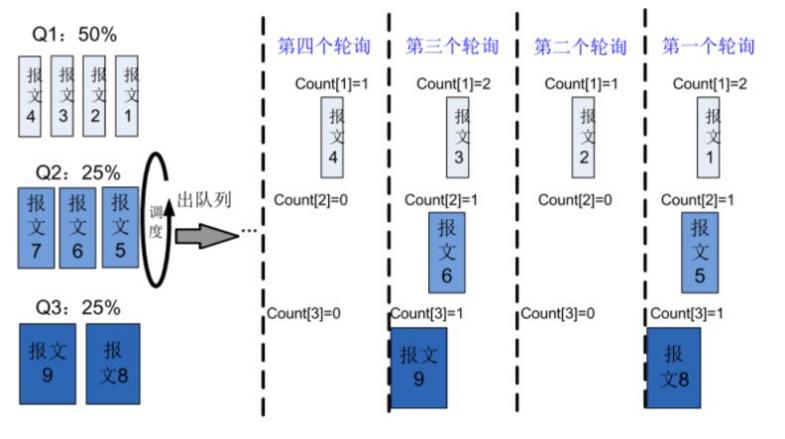
scheduler调度器，队列queue指的是在缓存中对报文进行排序的逻辑。当流量的速率超过接口带宽或超过为该流量设置的带宽时，报文就以队列的形式暂存在缓存中。报文离开队列的时间、顺序，以及各个队列之间报文离开的相互关系由队列调度算法（scheduling principle）决定。

PQ（Priority Queuing）调度，就是严格按照队列优先级的高低顺序进行调度。只有 高优先级队列中的报文全部调度完毕后，低优先级队列才有调度机会。 采用PQ 调度方式，将延迟敏感的关键业务放入高优先级队列，将非关键业务放入 低优先级队列，从而确保关键业务被优先发送。 PQ调度的缺点是：拥塞发生时，如果较高优先级队列中长时间有分组存在，那么 低优先级队列中的报文就会由于得不到服务而“饿死”。



RR（Round Robin）调度采用轮询的方式，对多个队列进行调度。RR以环形的方式轮询多个队列。 如果轮询的队列不为空，则从该队列取走一个报文；如果该队列为空，则直接跳过该队列，调度器不等待。RR调度各个队列之间没有优先级之分，都能够有相等的概率得到调度。RR调度的缺点是：所有队列无法体现优先级，对于延迟敏感的关键业务和非关键业务无法得到区别对待，使得关键业务无法及时得到处理。

  
加权轮询WRR（Weighted Round Robin）调度主要解决RR不能设置权重的不足。 在轮询的时候，WRR每个队列享受的调度机会和该队列的权重成比例。RR调度相当于权值为 1 的WRR调度。 WRR的实现方法是为每个队列设置一个计数器 Count，根据权重进行初始化。每次轮询到一个队列时，该队列输出一个报文且计数器减一。当计数器为 0 时停止调度该队列，但继续调度其他计数器不为 0 的队列。当所有队列的计数器都为 0 时，所有计数器重新根据权重初始化，开始新一轮调度。在一个循环中，权重大的队列被多次调度。



假设某端口有3个队列采用WRR调度，为每个队列配置一个权值，依次为50%、 25%、25%，详细的调度过程如下： 首先计数器初始化：Count[1]=2，Count[2]=1，Count[3]= 1。

− 第 1 个轮询： 从队列 1 取出报文 1 发送，Count[1]=1；从队列 2 取出报文 5 发送，Count[2]=0； 从队列 3 取出报文 8 发送，Count[3]=0。 − 第 2 个轮询： 从队列 1 取出报文 2 发送，Count[1]=0；由于Count[2]=0，Count[3]=0，队列 2 和队列 3 不参与此轮调度。 此时，Count[1]=0，Count[2]=0，Count[3]=0，将计数器重新初始化：Count[1]=2， Count[2]=1，Count[3]= 1。 − 第 3 个轮询：从队列 1 取出报文 3 发送，Count[1]=1；从队列 2 取出报文 6 发送，Count[2]=0； 从队列 3 取出报文 9 发送，Count[3]=0。 − 第 4 个轮询： 从队列 1 取出报文 4 发送，Count[1]=0；由于Count[2]=0，Count[3]=0，队列 2 和队列 3 不参与此轮调度。 此时，Count[1]=0，Count[2]=0，Count[3]=0，将计数器重新初始化：Count[1]=2， Count[2]=1，Count[3]= 1。

从统计上看，各队列中的报文流被调度的次数与该队列的权值成正比，权值越大被调度的次数相对越多。如果该端口为100Mbps，则可以保证最低权重的队列至少获得25Mbit/s带宽，避免了采用PQ调度时低优先级队列中的报文可能长时间得不到服务的缺点。WRR对于空的队列直接跳过，循环调度的周期变短，因此当某个队列流量小的时候，剩余带宽能够被其他队列按照比例占用。

WRR调度有两个缺点：

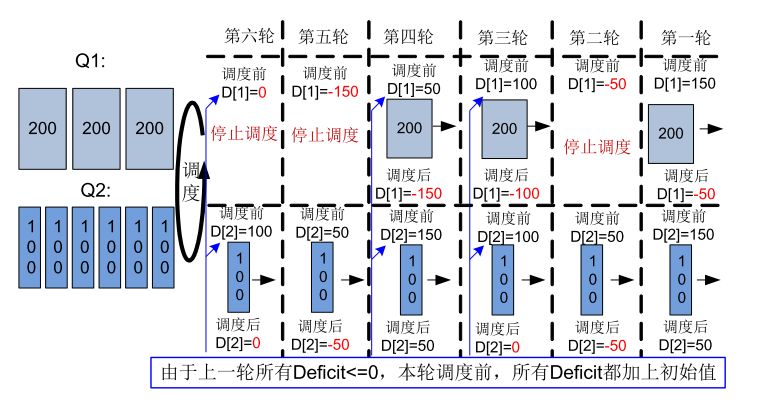
· WRR调度按照报文个数进行调度，因此每个队列没有固定的带宽，同等调度机 会下大尺寸报文获得的实际带宽要大于小尺寸报文获得的带宽。而用户一般关 心的是带宽。当每个队列的平均报文长度相等或已知时，通过配置WRR权重， 用户能够获得想要的带宽；但是，当队列的平均报文长度变化时，用户就不能 通过配置WRR权重获取想要的带宽。

· 低延时需求业务（如语音）得不到及时调度。

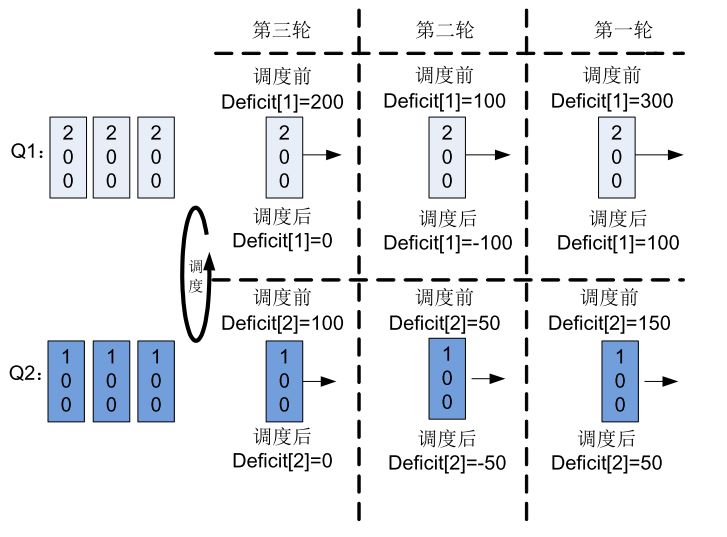
差分轮询DRR（Deficit Round Robin）调度实现原理与RR调度基本相同。DRR与RR的区别是：RR调度是按照报文个数进行调度，而DRR是按照报文长度进行调度。

DRR为每个队列设置一个计数器Deficit，Deficit 初始化为一次调度允许的最大字节数， 一般为接口MTU。每次轮询到一个队列时，该队列输出一个报文且计数器Deficit 减去报文长度。如果报文长度超过了队列的调度能力，DRR调度允许Deficit 出现负值，以保证长报文也能够得到调度。但下次轮循调度时该队列将不会被调度。当计数器为 0 或 负数时停止调度该队列，但继续调度其他计数器为正数的队列。当所有队列的Deficit 都为 0 或负数时，将所有队列的Deficit 计数器加上[初始值](https://www.zhihu.com/search?q=%E5%88%9D%E5%A7%8B%E5%80%BC&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"article","sourceId":"163812643"}" \t "_blank)，开始新一轮调度。

假设某端口MTU=150Bytes，有 2 个队列Q1和Q2采用DRR调度，Q1 队列中有多个 200Bytes 的长报文，Q2队列中有多个100Bytes 的端报文，则调度过程如图 4 所示。



由图4可以看出，经过第 1~6 轮DRR调度，Q1队列被调出了3个200Bytes 的报文， Q2队列被调出了6个100Bytes 的报文。从长期的统计看，Q1和 Q2 的实际输出带宽比 是 1:1，为公平的比例。DRR调度避免了采用PQ调度时低优先级队列中的报文可能长时间得不到服务的缺点。 但是，DRR调度不能设置权重，且也具有低延时需求业务（如语音）得不到及时调度的缺点。

差分加权轮询DWRR（Deficit Weighted Round Robin）调度主要解决DRR不能设置权重的不足。DRR调度相当于权值为 1 的DWRR调度。 DWRR为每个队列设置一个计数器Deficit，Deficit 初始化为Weight \* MTU。每次轮询到一个队列时，该队列输出一个报文且计数器Deficit 减去报文长度。当计数器为 0 时停止调度该队列，但继续调度其他计数器不为 0 的队列。当所有队列的计数器都为 0 时，所有计数器的Deficit 都加上Weight\*MTU，开始新一轮调度。  


假设某端口MTU=150Bytes，有 2 个队列Q1和Q2采用DRR调度，Q1 队列中有 多个 200Bytes 的长报文，Q2队列中有多个 100Bytes 的端报文，Q1和Q2配置权重 比为weight1:weight2=2:1。则DWRR调度过程如图 2-37。

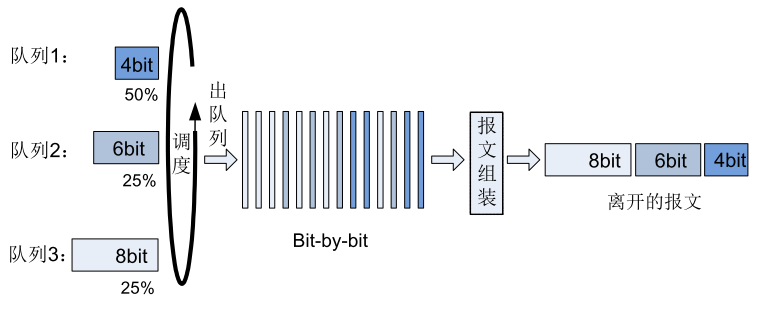
第一轮调度 Deficit[1] =weight1\* MTU=300，Deficit[2] = weight2\* MTU=150Bytes，从 Q1 队 列取出 200Bytes 报文发送，从Q2队列取出 100Bytes 发送；发送后，Deficit[1] = 100，Deficit[2] =50。

第二轮调度 从 Q1 队列取出 200Bytes 报文发送，从Q2队列取出 100Bytes 发送；发送后， Deficit[1] = -100，Deficit[2] =-50。

第三轮调度 此时两个队列都为负，因此，Deficit[1] = Deficit[1]+weight1\* MTU=-100+2150=200，Deficit[2] = Deficit[2]+weight2 MTU=-50+1\*150=100。 从 Q1 队列取出 200Bytes 报文发送，从Q2队列取出 100Bytes 发送；发送后， Deficit [1] = 0，Deficit[2] = 0。

由上图可以看出，经过第 1~3 轮DWRR调度，Q1队列被调出了3个200Bytes 的 报文，Q2 队列被调出了3个100Bytes 的报文。从长期的统计看，Q1和Q2的实际 输出带宽比是 2:1，与权重比相符。

DWRR调度避免了采用PQ调度时低优先级队列中的报文可能长时间得不到服务的缺点，也避免了各队列报文长度不等或变化较大时，WRR调度不能按配置比例分 配带宽资源的缺点。但是，DWRR调度也具有低延时需求业务（如语音）得不到及时调度的缺点。

加权公平队列WFQ（Weighted Fair Queuing）调度是按队列权重来分配每个流应占有出口的带宽。同时，为了使得带宽分配更加“公平”，WFQ 以 bit 为单位进行调度，类似于图 6的 bit-by-bit 调度模型  
  
Bit-by-bit 调度模型可以完全按照权重分配带宽，防止长报文比短报文获得更多带宽，从而减少大小报文共存时的时延抖动。但Bit-by-bit调度模型只是理想化的模型，实际上，华为交换机实现的WFQ是按照一定的粒度，例如 256B、1KB，或其他粒度，具体按何种粒度，与单板类型相关。 WFQ的优点主要有以下几点：

不同的队列获得公平的调度机会，从总体上均衡各个流的延迟。

短报文和长报文获得公平的调度：如果不同队列间同时存在多个长报文和短报文等待发送，让短报文优先获得调度，从而在总体上减少各个流的报文间的抖动。

从统计上看，权重越小，所分得的带宽越少。权重越大，所分得的带宽越多。

WFQ问题：

考虑steady state下的response time的quantile。K个class，不同权重，考虑每个class的quantile，不同权重，在WFQ的调度器下去计算tail probability。小概率事件，仍然是IS。

Regenerative还是把系统从非空到空去做切割。