# 数据中心网络关键技术项目研究进展报告

## 1 数据中心网络流量特征

### 1.1 细粒度的数据中心网络流量特征

分析了web、Cache、Hadoop三种业务的流量特征。在测试的过程中，数据中心使用的是传统的接入层、汇聚层和核心层的三层结构，每一个服务器都通过10Gbps的以太网链路与其所在rack上方的Tor-switch相连接，每一个Tor-switch相应地又与其上方的汇聚层交换机（“fabric”）通过40Gbps或100Gbps的链路相连接，然后，汇聚层交换机再与核心层交换机（“spines”）相连接，由此构成树状结构。本实验一共测量了30个rack上Tor-switch层在24小时内的流量数据，每个业务各占其中10个rack，并且每个rack上的所有服务器都只运行同一种业务，以确保采集到的数据能够代表相应业务的流量特征。实验取样的是所有rack上的Tor-switch连接的任意一个服务器端口的数据，每个小时任取其中的两分钟的数据进行记录，通过一个月的采集得到了720个这样的记录。

#### 1.1.1 细粒度的流量分布特征

首先对测量的数据以25µs的时间间隔进行采样，如果在采样间隔里链路的利用率超过了50%则我们称其在该时间间隔为Hot状态，如果一个链路在连续的采样间隔中出现了不间断的Hot状态则我们称其为一次burst。以下是关于Web、Cache、Hadoop三种业务burst的持续时间长度的CDF曲线，如图1所示。

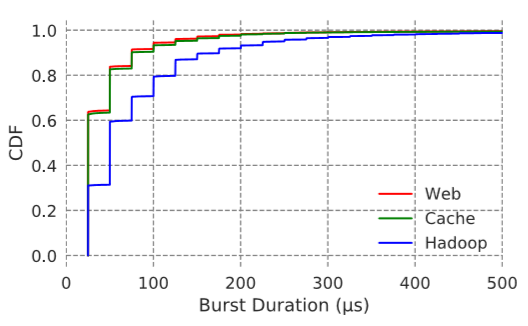


图1 突发的持续时间分布

由持续时间的CDF图线可以发现，三种业务超过90%的burst的持续时间长度都小于200µs，而且Web业务的机架的测得的burst的持续时间大概90%都要小于50µs，由此可以得出结论：链路处在高利用率的持续时间长度一般都很短，大概在200µs以内。

继续以25µs为时间间隔，再考虑相邻两个时间间隔是否出现burst是不是会相互影响。

令随机变量来表示在第t个时间间隔内是否出现Hot状态，表示第t个时间间隔内出现了Hot状态，表示第t个时间间隔内没有出现Hot状态。于是根据测得的结果得到以下数据，如图2所示：

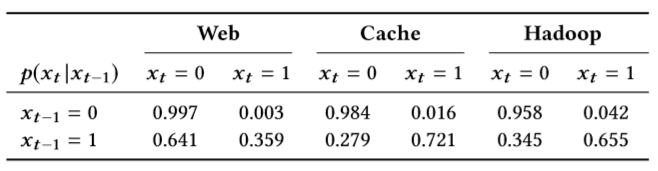


图2 突发间隔间的相关性

通过表格数据我们可以计算参数r=，如果则说明相邻两个间隔是否为burst状态没有相互影响，若或者则说明相邻两个间隔有联系。计算上述数据可得：

因为三种业务的，所以三种业务下，相邻时间间隔出现高利用率burst状态是相互影响的。

最后研究的是相邻两次burst之间的时间间隔大小规律，根据测得的数据可以得到25µs采样数据下相邻两次burst的时间间隔大小的CDF曲线，如图3所示：

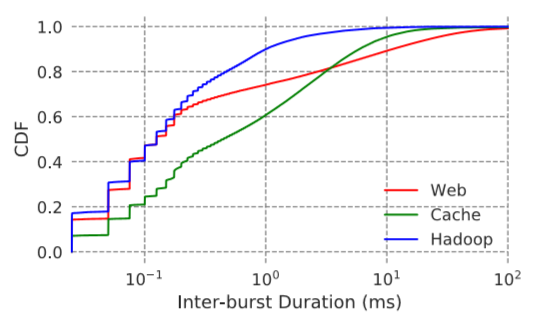


图3 突发间隔分布

根据曲线可以发现，相邻两次burst之间的时间间隔大多时候都是很小的，特别是对于Web业务和Cache业务来说，它们的间隔大小超过40%都是在0.1ms以下，但是之后的时间间隔大小的分布非常广，且随着时间间隔的增大其概率密度也不断变小。

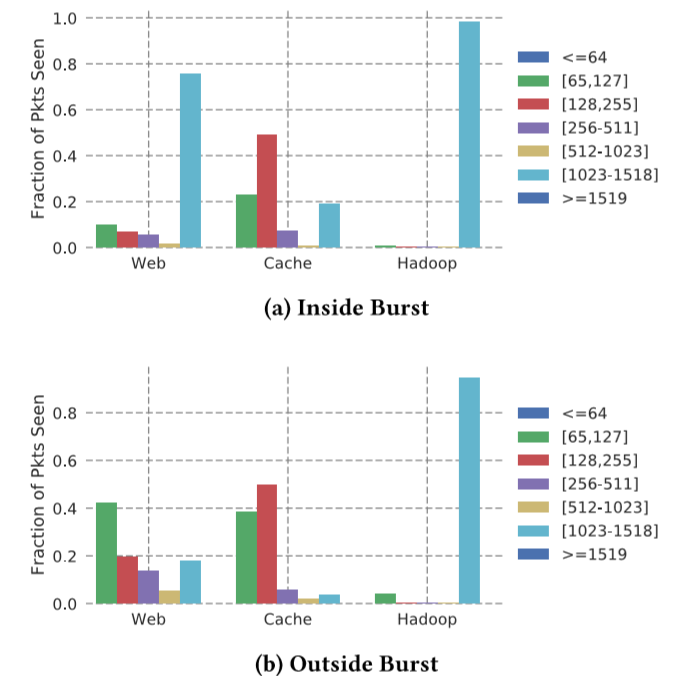


图4 三种业务的数据包大小分布

#### 1.1.2数据包大小特征

根据所测得的数据，得到在burst中和burst以外Web、Cache、Hadoop三种业务的数据包大小情况柱状图，如图4所示：

对比两张柱状图可以发现，大体上，burst以外的数据包大小要小于burst内的数据包，而在Hadoop业务下无论是burst内还是burst外主要都是大数据包（大概这与Hadoop业务本身的特点相关），对于Web业务来说在burst以内大数据包占主导，超过了数据包总量的75%，而在burst以内由中小尺寸的数据包主导，但对于Cache业务来说无论是burst以外还是burst内都是小数据包占大多数。

#### 1.1.3 链路负载均衡特征

在本测量环境中，每个Tor-switch都分别通过四个上和下行的链路与汇聚层交换机相连，Tor-switch采用Equal-Cost MultiPath（ECMP）的策略将负载分配至每一个链路上，原则上，这样的策略可以保证比较好的均衡效果，但是实际测试效果如下，用平均绝对离差（Mean Absolute Deviation）的CDF曲线表示，实线表示采样间隔为25µs的情况，虚线表示采样间隔为1s的情况，如图5所示：

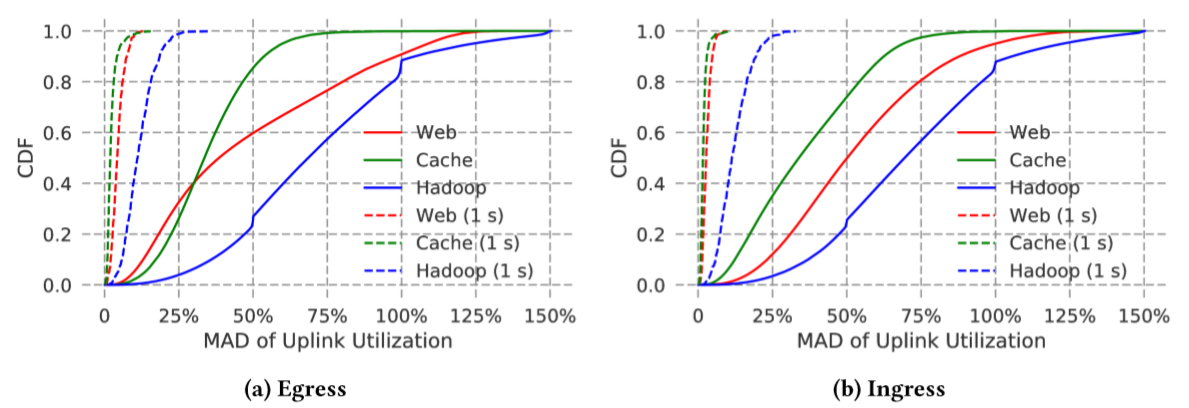


图5 链路利用率分布

有图可知在采样间隔为1s时，三种业务的负载相差并不大，基本做到了负载均衡，但是当用25µs的采样间隔进行采样时我们却发现，三种业务的负载都相差较大，特别是Hadoop业务，不同链路的负载情况甚至出现倍数关系，基本上可以判定从极短时间上看ECMP的负载均衡效果比较糟糕，其原因可能有两个：（1）ECMP的均衡对象是流的大小而非数据包；（2）它使用的是一致性Hash，不能使均衡效果做到最优。

#### 1.1.4 上行和下行链路繁忙情况

使用300µs采样周期进行测试，得到上行和下行链路在测试时间间隔中出现Hot状态的频数，并绘制以下上行和下行Hot状态比例柱状图：

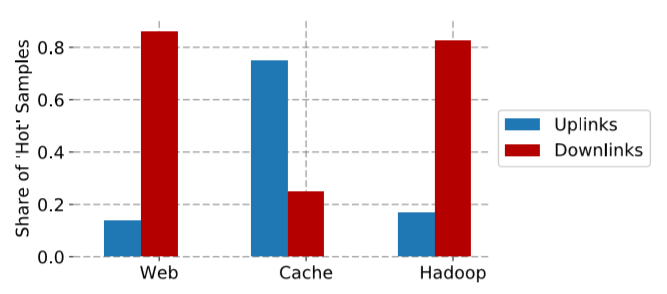


图6 上行和下行链路繁忙状态

根据图中信息，Web和Hadoop业务的Hot状态多出现于下行链路中，上行链路中出现的Hot状态的比例均不足20%。而Cache业务与之相反，在上行链路中更易出现Hot状态，而下行链路中出现Hot状态的比例不足30%。

#### 1.1.5 Burst对Buffer占用率的影响

使用50ms的抽样周期对测量的流量数据抽样，测量发生burst的端口的比例与buffer占用率的关系，这里对buffer占用率根据其最大值进行归一化处理，得到以burst端口比例为横坐标、buffer归一化处理后的占用率为纵坐标的箱型图，如图7所示：

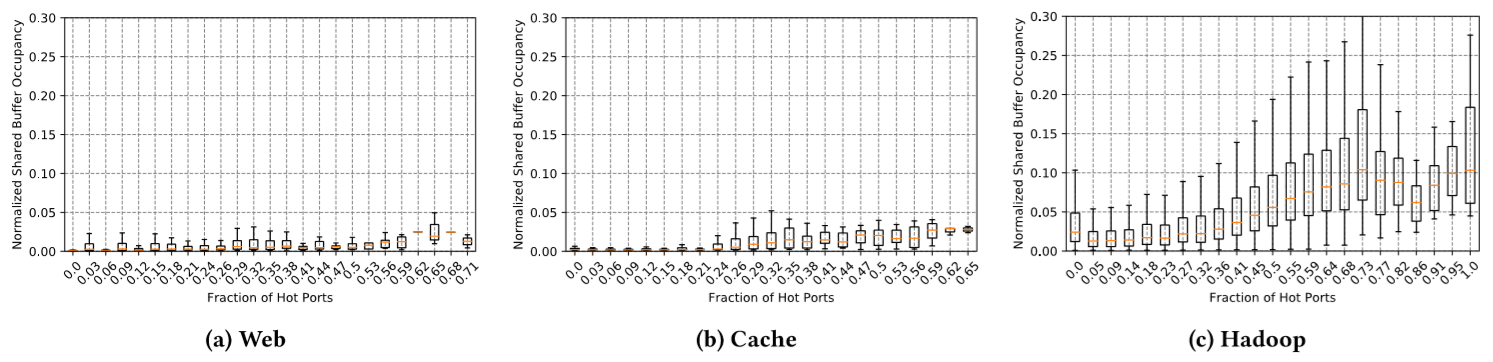


图7 Burst与端口buffer占用率

由图可知，Hadoop的burst的端口的比例最大值最高，而Web和Cache业务处于高利用率的端口比例在归一化条件下最高分别为71%和65%。而且相比于其他两种业务，Hadoop对buffer的占用率更高，而且占用率的分布范围也更广。同时，在burst端口比例较高时，三种业务的buffer占用率都保持持平趋势。

### 1.2 Facebook数据中心网络流量特征

#### 1.2.1 Facebook数据中心网络拓扑结构

Facebook的数据中心分布在多个地理位置，不同位置的数据中心可以成为一个Datacenter Site，由骨干网络将所有的这些Site连接起来。而在每一个Datacenter Site中可能包含了一个或多个建筑，每个建筑都相当于是一个独立的数据中心，各自都运营着独立的数据中心网络。每个数据中心网络都有类似于图8所示的拓扑结构，它由多个cluster构成，cluster可是视作数据中心网络中部署的一个个单元，承载某个独立的业务，如hadoop、cache、frontend等，每个cluster都采用类似clos架构，即完全二分图结构，其中一层为top-of-rack switch（RSW），另一层为cluster switch（CSW），它们之间通过10Gbps的链路连接起来。每个RSW会连接很多承载特定服务的主机，他们通过10Gbps的以太网链路与对应的RSW相连接，每个主机都只运行单一的服务，同一RSW下的主机运行的服务都相同，同一个cluster下的主机运行的服务可能都相同，也可能不同，这具体取决于cluster本身承载的业务。每个RSW下连接的主机数量不固定，因cluster而异。不同cluster中的CSW通过称为Fat Cat的另一层交换机连接起来，单独的CSW和FC之间通过四条10Gbps的链路相连接，实现40Gbps的可用带宽。除此之外，CSW还通过连接汇聚层交换机负责与同一个site的不同数据中心网络进行数据传输，以及通过连接路由器实现不同site之间的数据传输。

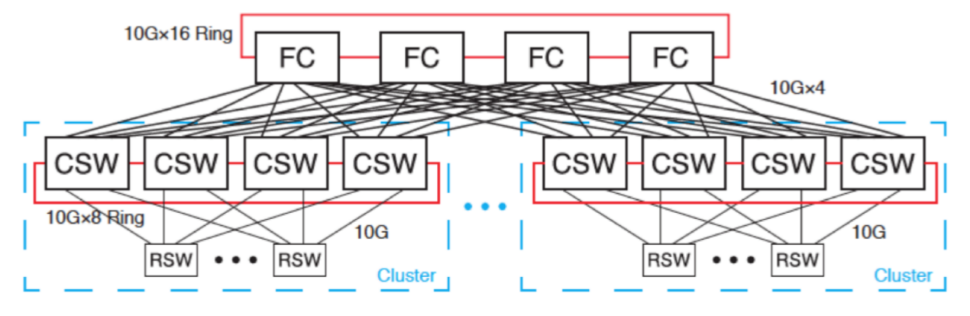


图8 Facebook数据中心网络拓扑结构

现今大多数的Facebook数据中心网络都还在采用4-post的clos架构，即每个数据中心网络中有4个FC，每个cluster中有4个CSW。而目前Facebook也已研发出新的Fabric架构，并开始在自己的数据中心中推广和应用。本文中所有的数据来源都是传统的图8所示结构。

#### 1.2.2 业务概况

本文将考虑五种运行不同服务的主机，包括：运行web业务的web服务器，运行数据库业务的MySQL服务器，用于临时存储请求结果的cache服务器（既包括了保证数据一致性的leader cache，还包括了处理大多数读取请求的follower cache），进行离线数据分析和数据挖掘的hadoop服务器，以及聚合并供给信息资源的Multifeed服务器。这些服务涵盖了Facebook数据中心大多数的数据流量。

Facebook数据中心中还有少数服务类型不固定的主机，他们的服务类型会根据其目的动态调整。同时，Facebook数据中心也没有运用虚拟机，每一个服务都是运行在真实的物理主机上。

#### 1.2.3 链路利用率特征

主机和它们对应的RSW之间的链路，1分钟内的平均链路利用率不足1%，如此低的链路利用率大概与Facebook不久前将主机的接入层链路由1Gbps升级为了10Gbps有关。即使但看负载最为严重的链路，99%的链路1分钟内的平均利用率也不足10%，而且不同cluster中的负载也由很大差别，负载最大的cluster（Hadoop业务）的平均链路利用率差不多是负载较小的的cluster（Frontend）的5倍。

对于RSW和CSW之间的链路，不同的cluster的链路利用率的中位数都在10%-20%之间，其中负载最重的前5%的链路利用率中位数在23%-46%之间，这个数据高出了之前大多数研究所得的结论。而不同cluster之间的差异减小了，负载最重的cluster的平均链路利用率是负载矫情的cluster的3倍。

而再上层，CSW和FC之间的额链路利用率就更高了，而且不同cluster的差异也更小了，大概是因为不同cluster中的链路会根据其特定的需求进行过超额的分配，而保证了cluster之间的链路利用率差别不大。

#### 1.2.4 流量传输的特征

通过以1秒为时间间隔，分别对Hadoop cluster的一个Hadoop服务器、Frontend cluster的一个web服务器和Cache cluster的一个cache follower和一个cache leader服务器的流量传输分布情况进行统计，得到了图2所示4台服务器的流量传输分布情况柱状图。

由图9可知，Hadoop业务的流量分布情况变化最为剧烈，显示出在某些时间间隔上由明显的流量传输过程，而还有一些时间间隔上没有明显的流量传输。从具体数据上看，该Hadoop服务器发送的99.8%的数据流量都发送给了其他的Hadoop服务器，其中，75.7%的流量发往了该服务器所在rack下的其他服务器，而且这些rack内流量分配相当均匀，剩下的几乎所有流量都发往了该cluster内其他rack的主机，只有极为少数的流量离开了该cluster。从具体分布上看，发往该cluster不同rack的流量的接收方主机占该cluster中所有主机的1.5%，分布在该cluster内的95%的rack中，然而，其中80%的流量都发往了17%的rack，此结论与此前研究观测到的数据保持一致。而且，尽管hadoop服务器发送的流量基本发往了分布在同一个rack内或者同一个cluster内的主机，但具体这两者各占比多少却不断变化。Hadoop这样的多变性大概是其处理的job的大小和job所处的不同阶段共同导致的结果。

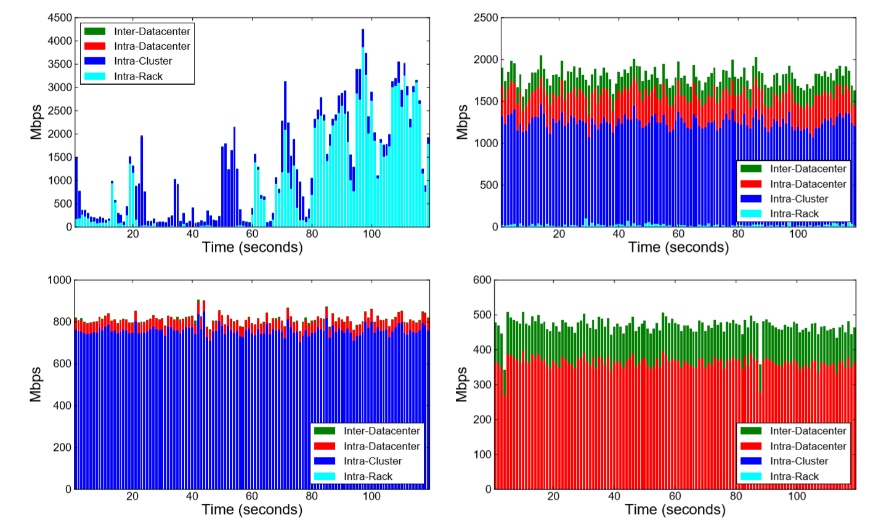


图9 流量传输特征（左上图为Hadoop服务器，右上图为Web服务器，左下为cache follower，右下图为cache leader）

除了Hadoop业务外，其他业务服务器的流量传输过程都较为稳定，并且只有很小一部分流量发送至发送方服务器所在rack内的主机，甚至数据中心网络之间传输的流量都比在该rack内传输的流量要多。

从具体数据来看，Frontend cluster中的web服务器发送的68%的流量都发往了其所在的同一个cluster内部的主机，而这其中的80%的流量都发往了cache服务器，而Multifeed服务器和SLB服务器均各接收了8%的数据。该web服务器发送的剩下的流量大多都是发往了其所在数据中心网络内的其他cluster的主机或者发送给了其他的数据中心网络，在同一个rack内传输的流量微乎其微。

由于cache follower主要是负责响应web服务器的请求，因此其发送的流量多发往了同一个cluster内的web服务器。具体数据来看，cache follower发送数据的88%都是发往了web服务器，而且其发送的流量的接收方覆盖了所在cluster内的75%的主机，其中也覆盖了90%的web服务器。

Cache leader负责维持cluster之间信息的一致性，并维护后台数据库，这就使得cache leader的流量传输多分布在数据中心内的不同cluster之间或者是不同数据中心之间，而具体的数据统计也证实了这一点。

图10列出了2015年1月在Facebook数据中心采集到的24小时间隔内流量传输分布比例，同时也分别列出了不同的业务的占比情况。

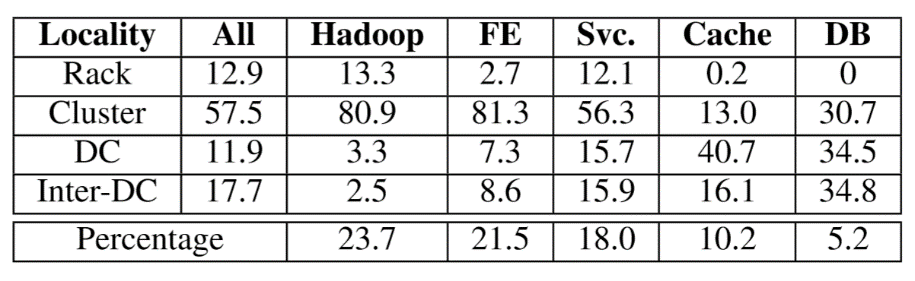


图10 不同业务的流量分布比例

很明显，大部分的流量都发往了发送方所在的cluster内的其他rack下的主机，约占总流量的57.5%，而在同一个rack内传输的流量占比约为12.9%，小于数据中心网络之间传输的流量比例。这与之前研究所说的流量主要在rack内传输的结论不同，即使是rack内传输的流量比重最大的Hadoop业务也只有13.3%的流量发送给了发送方所在rack内的主机。

通过图10也可以发现流量比例最高的五类业务分别为Hadoop业务（23.7%）、Frontend业务（21.5%）、Service业务（18.0%）、Cache业务（10.2%）、Database业务（5.2%），这五类业务流量约占总流量的78.6%。

#### 1.2.5. 流量矩阵特征

图11左侧两幅图分别为Hadoop业务和Frontend业务具有64个rack的cluster内的流量矩阵，其描绘的是cluster内rack之间传输的流量大小。

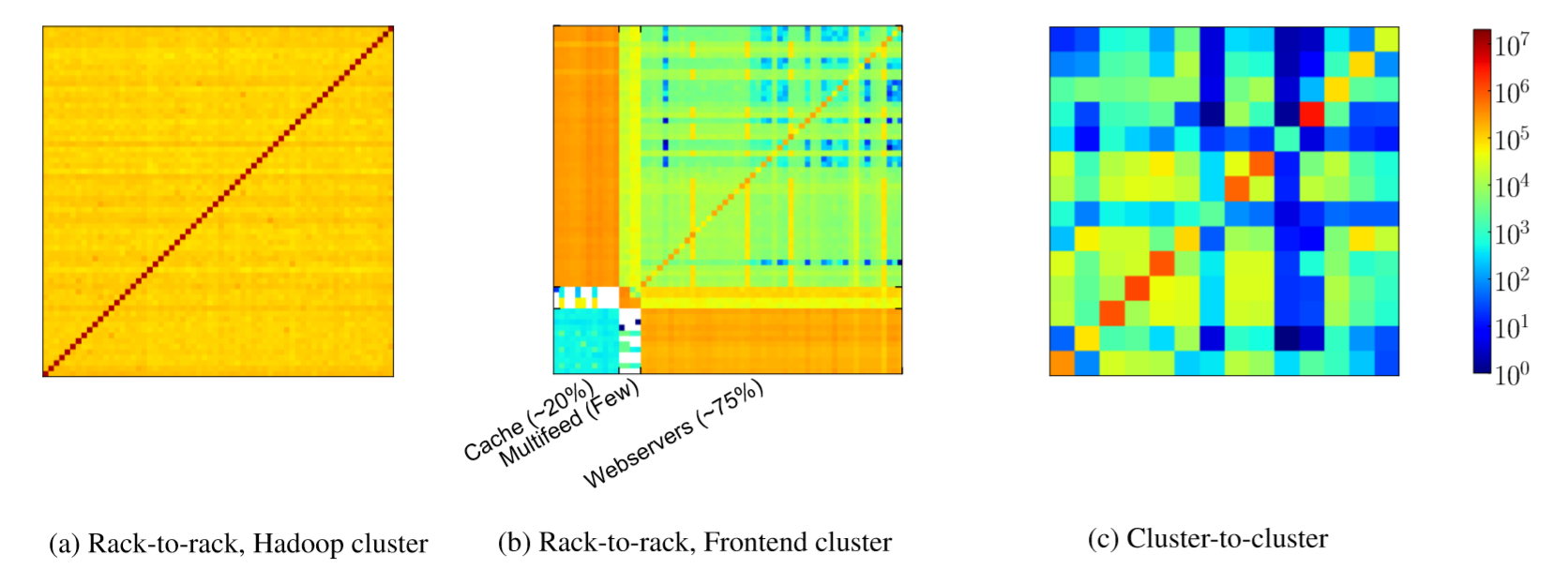


图11 流量矩阵特征

由左侧的Hadoop业务流量矩阵可知，Hadoop cluster内的不同rack之间的传输流量分布非常均匀，约占Hadoop cluster内传输的总流量的80%，而对角线上的值相对较大，说明rack内传输的流量相对于发送给其他rack下主机的流量具有更大的比例。这样的结果可能是因为map任务主要分配在本地，但是过量的任务本地rack无法提供足够的资源，因此，必须将一部分的流量发送给其他rack的主机进行处理，还有因为有一些任务并不要求在本地rack内的主机处理，因此这些任务也会使得本地rack内流量传输的比例下降。

图11中间的流量矩阵展示了Frontend cluster的三类不同服务的rack流量传输分布情况。由图数据可知，rack内部传输的流量并没有明显的优势，而web服务和cache follower的rack之间传输的流量占较大的分量，这说明web服务器主要向cache服务器发送流量，反之亦然，而不同类型的服务器存在于不同的rack中，导致了rack内传输的流量较少。

还有三种服务cluster的流量矩阵未在图中展示。Cache leader cluster的流量矩阵反映出在rack内流量传输的需求很小；Backend database的流量传输分布非常均匀，几乎cluster内所有的节点都平均地分配传输流量；Service cluster地不同rack支持的服务都有可能不同，因此它展示的流量模型介于这些极端情况之间。

图11右侧的流量矩阵展示的是在24小时的时间间隔里15个不同cluster之间传输流量的大小。由图可以发现由于cluster承载的业务不同，他们之间的流量大小会有较大差异。因为每个cluster内部都采用的是4-post的clos架构，而不同的cluster对之间传输的流量大小差距可能超过7倍之多，所以考虑在不同的cluster之间采用不同质量的通信链路可能会有一定的意义。

由于边缘接入链路具有很低的链路利用率，因此我们可以据此调整上层汇聚层和核心层链路超额分配的比例，但是这也需要观察边缘链路的利用率会不会随着使用年限的增加而逐渐增加。流量传输分布的差异性使得采用相同拓扑结构的不同cluster可能会出现在某些cluster中链路资源超额分配，在某些cluster中却出现了拥塞，或者两种情况都有，所以采用在特定位置分配更多带宽资源的非均匀fabric拓扑结构可能会提高网络传输性能。网络流量特征在时间上的稳定性也反映出网络状态的快速调整可能并没有太大必要。

#### 1.2.6 单个流的特征

图12和图13分别展示的是10分钟内（web服务是2.5分钟内）采集到的三类业务的flow（根据五元组区分）大小尺寸和持续时间情况统计：web服务的rack、单独的cache follower（cache leader特征相似，因此没有展示）和一个Hadoop节点。

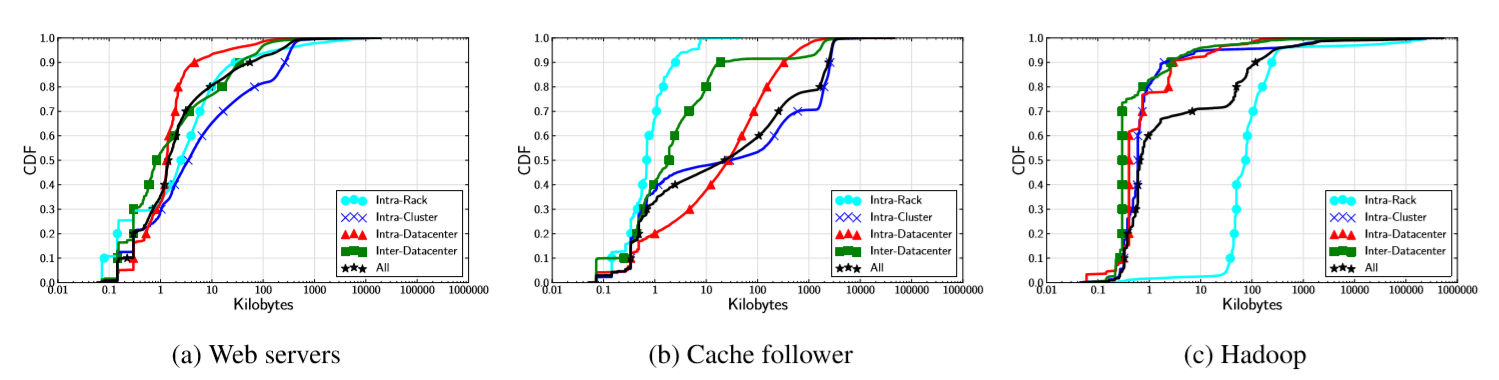


图12 单个流的大小分布

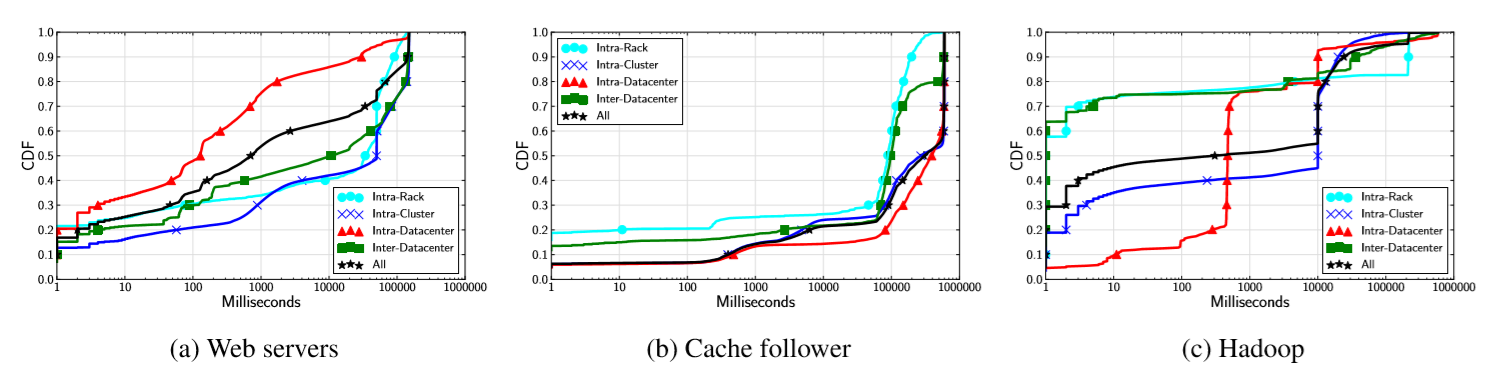


图13 单个流的持续时间分布

由于Hadoop的流量无论是在不同时间间隔上，还是不同的节点上，都会呈现出较大的差异，因此这里只测量了某个Hadoop节点在传输流量较大的10分钟间隔内的特征。从统计数据来看，Hadoop业务的flow约有70%长度尺寸都小于10KB，持续时间小于1s，只有不到5%的flow尺寸大于1MB或者持续时间超过了100秒，几乎没有flow的持续时间超过10分钟。

与Hadoop业务相反，其他服务的流量因为存在负载均衡的影响，要更具有代表性。而且许多Facebook内的服务都采用了一些connection pooling的机制，使得其flow在相对较小的吞吐量下却由较长的持续时间。

Cache follower和cache leader就是采用connection pooling机制的服务，大体上cache的flow尺寸要远大于Hadoop，follower中持续时间小于100秒的flow约占30%，leader中约占40%，而40%的follower的flow持续时间超出了这里10分钟的测量间隔，leader中有25%的flow持续时间超出了10分钟。这也就是说，大多数cache的flow都只在某些毫秒级别长度的时间间隔上较为活跃（在传输数据包），而这些时间间隔之间隔着较长的空窗期。换句话说，无论flow的尺寸大小还是持续时间是多少，flow内部都呈现突发式传输的流量特征。Web服务器的flow特征介于Hadoop和cache之间。

#### 1.2.7 负载均衡特征

负载均衡方法一般都是针对稳定性较差，不同时间上流量传输变化较大的情况，如果流量已经非常稳定，传输速率无明显波动，那么说明流量的传输已经较为通畅，负载均衡方法可以提升传输性能的空间就会很小。由此，可以检测某个主机发送的流量每秒的变化情况，即以1秒为时间间隔进行观察，绘制出如图14所示的每个时间间隔内该主机发往各个目的rack的flow速率的分布情况，对比各个时间间隔绘制的flow速率分布曲线，就可以了解流量传输的变化情况。

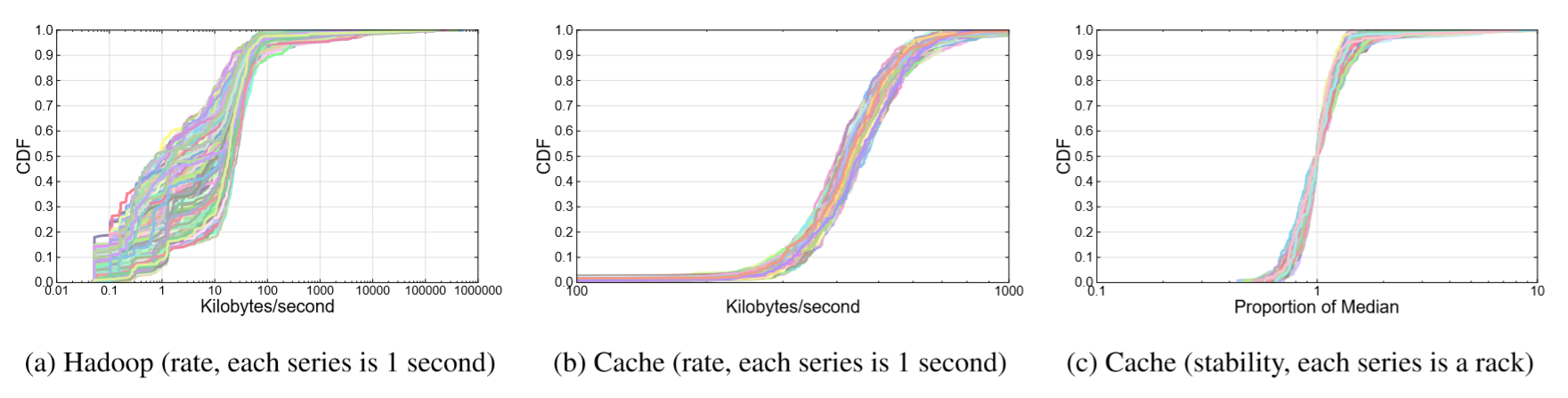


图14 不同业务

图14左侧两幅图分别是Hadoop业务和Cache业务每秒的flow到达速率分布曲线，一共检测了2分钟时间段上的变化情况，因此共有120条flow速率分布曲线。

由图14左侧图可知，Hadoop业务每秒的flow速率分布变化很大，变化大小超过3倍的情况很普遍。而由中间的图可知，Cache业务在120秒的测量期间内变化较小，曲线相对较为密集，差不多在中位数2Mbps处与横轴垂直。其他的业务与cache业务的曲线相似。

由上述测量结果可知，发往各个目的rack的Cache业务流量在时间上基本保持平稳，而图14右侧图还对单个目的rack接收flow时的速率分布做了统计，以检验单个目的rack的流量到达速率在时间上的变化大小，其数据按中位数进行了归一化处理。从图中曲线可知，曲线在中位数附近非常密集且接近垂直，所以可知单个目的rack的流量到达速率在时间上也较为平稳，且flow的到达速率分布非常集中，大约90%的flow到达速率都集中在中位数的2倍范围内。

#### 1.2.8. Heavy hitter流特征

传输速率最大的flow或这些flow的集合是负载均衡等网络工程需要关注的重点，因为它们最有可能对网络性能产生影响。这里定义heavy hitter为固定时间间隔内超过观测流量值总大小50%的flow（或host、或rack）最小集合。因此heavy hitter可以当作网络测量关注的重点。图15展示了1毫秒测量间隔内，四类服务，三种（flow、host、rack）heavy hitter在三种负载（负载在10%、50%、90%）下包含的元素的数量或者尺寸。因为瞬间传输的大流对网络影响更大，所以这里就用flow的到达速率来代替heavy hitter中包含的flow的尺寸大小。



图15 Heavy hitter中包含的流数目情况

图16展示了三种heavy hitter持续时间情况，分别在1毫秒、10毫秒、100毫秒三种测量间隔下，当前时间间隔的所有heavy hitter中可以持续到下一时间间隔的比例。

由图可知，Heavy hitter flow的持续性较差，无论检测的时间间隔大小为多少以及何种类型的服务，下一个间隔依然存在的heavy hitter在当前时间间隔中所占的比例中位数不超过15%，这说明内部flow的突发性特征。基于主机的heavy hitter情况会稍好些，除web服务外能持续到下一个时间间隔的heavy hitter host比例中位数不超过20%。Web服务在100毫秒测量间隔有较好的持续性，heavy hitter host持续到下一间隔的比例中位数约为30%。Rack级别的heavy hitter情况最好，100毫秒的检测间隔下，持续到下一间隔的heavy hitter比例的中位数均超过了40%，其中web服务甚至超过了60%，而1毫秒的测量间隔下持续到下一个间隔的heavy hitter比例中位数大约为32%，这说明web服务的heavy hitter大体上要更加的稳定。尽管如此，heavy hitter的持续性依然不足，能够持续至下一个测量间隔的heavy hitter比例依然较小，不是特别支持流量工程。

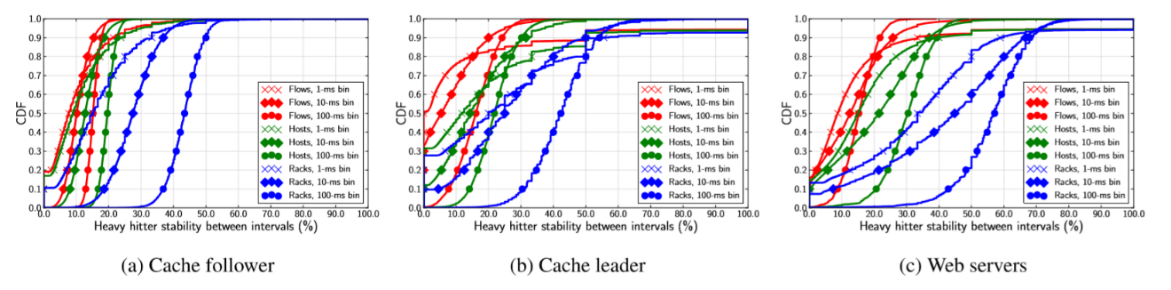


图16 不同业务Heavy hitter稳定性分布

由以上的检测结果可知，Facebook数据中心广泛应用的connection pooling机制导致flow的持续时间较长，这为流量工程提供了便利条件。应用层的负载均衡做到了较好的均衡效果，却限制了一些方法提升网络性能的空间。很多现有的技术是通过检测网络中的heavy hitter然后对他们特殊处理来改进网络性能，然而这些技术的前提是要先检测出heavy hitter，在heavy hitter持续性未知的各类cluster中检测出heavy hitter并不是一件容易的事情，并且即便是在大于100毫秒的时间间隔上检测rack级别的heavy hitter，也不能确定检测的结果能带来多大好处，因为heavy hitter在大多数的间隔上并不是始终都能持续。之前确实有工作指出，如果35%的heavy hitter都可以持续流量工程就有做的意义，然而，也只有web和cache服务的rack级别heavy hitter达到了这个要求。

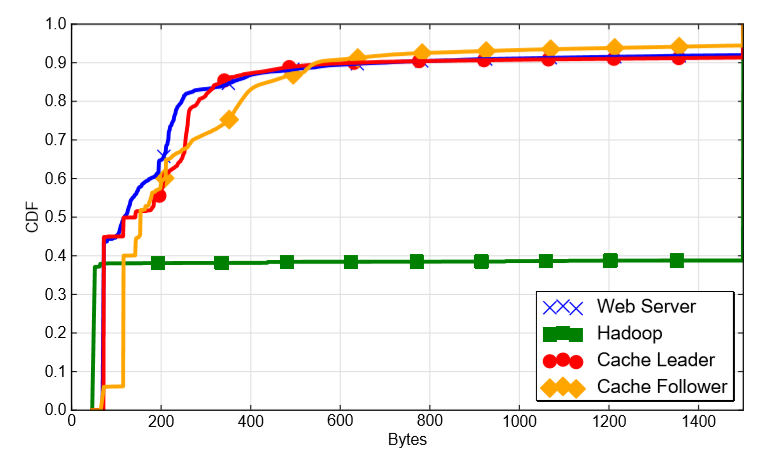


图17 不同业务的数据包大小分布

#### 1.2.9数据包特征

图17描绘的是四种服务主机发送的数据包大小分布情况。总体来看，数据包的平均尺寸在250字节左右，但是这个数字主要是因为Hadoop流量的影响发生了较大的偏移。

Hadoop服务的数据包尺寸呈现双峰分布的特征，即所有的数据包要么为MTU上限（这里是1500字节）的长度，要么为TCP ACK的大小，大概不到50字节。其他服务的书包尺寸分布较为宽广，但它们的中位数均没有超出200字节，即大概只有MTU上限的5%-10%。由此可见，尽管链路利用率很低，但是数据包的发送速率还是很高的。以cache服务器为例，假设服务器发往RSW的链路的利用率为10%，并且数据包平均尺寸为175字节，则该服务器每秒发送的数据包速率为发送MTU尺寸的数据包使链路利用率达到100%的数据包速率的85%。因此，对于单个数据包的操作，通过链路利用率可能无法立即看出发送过程承受的压力大小。

#### 1.2.10 数据包到达速率特征

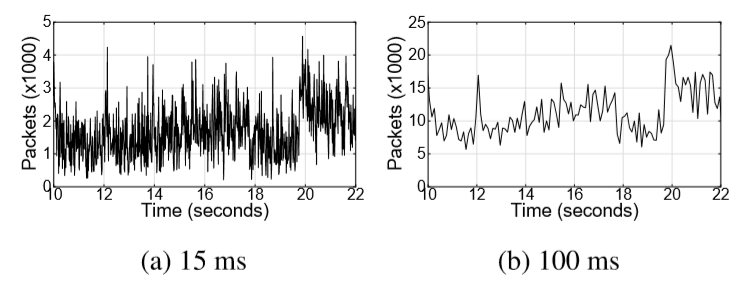


图18 数据包到达速率变化情况

之前的研究结果表明，在主机处观察到的数据包到达速率特征呈现开关模型。然而，对Facebook的数据中心的监测发现，即使是Hadoop业务也未能反映出这一特征，图18展示了以15毫秒和100毫秒为间隔统计的一个Hadoop主机发往对应RSW端口的数据包速率随时间变化曲线。而如果把发往不同宿点主机的数据包分离来看，便可观察到开关模型，这说明是发往不同宿点主机的数据包混杂在一起淹没了开关模型的特征。

图19展示的是四类业务的主机检测到的相邻tcp flow之间到达时间间隔的分布情况，即相邻TCP SYN数据包达到速率情况分布。虽然Facebook数据中心有很多服务采用了connection pooling机制，但依然存在一些持续时间较短的flow。通过图可以发现，Hadoop和web服务器flow到达间隔的中位数均在2ms左右，也就是说每秒约有500个flow到达。可能因为存在connection pooling机制的关系，两类cache服务器的flow到达时间间隔均高于web服务和Hadoop服务，而cache leader服务器的到达率又略高于cache follower，cache leader服务器flow到达时间间隔约为3毫秒，cache follower的约为8毫秒。

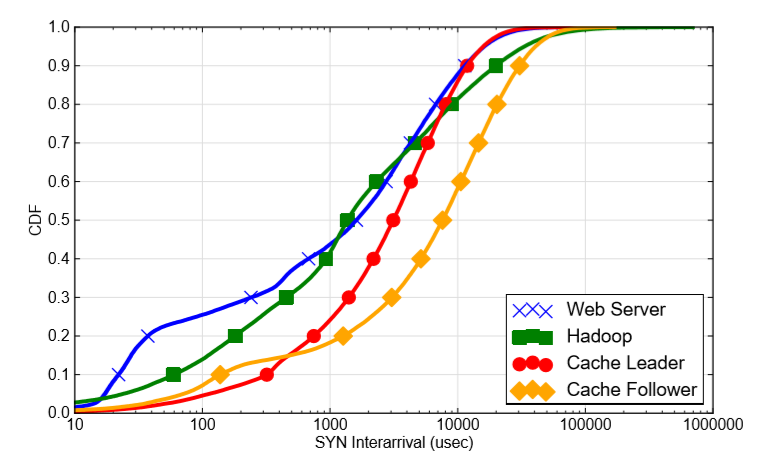


图19 不同业务的TCP流的到达间隔分布

#### 1.2.11交换机Buffer占用率

在24小时的时间段内，以10微妙为时间间隔，分别对连接web服务服务器和连接cache服务服务器的交换机的buffer占用率进行采样，并绘制每秒测得的占用率中位数和最大值，得到图20所示的结果。同时还分别对其在该测量时间段的链路利用率和丢包率情况进行统计，统计结果根据其最大值进行归一化处理，结果如图21和22所示。

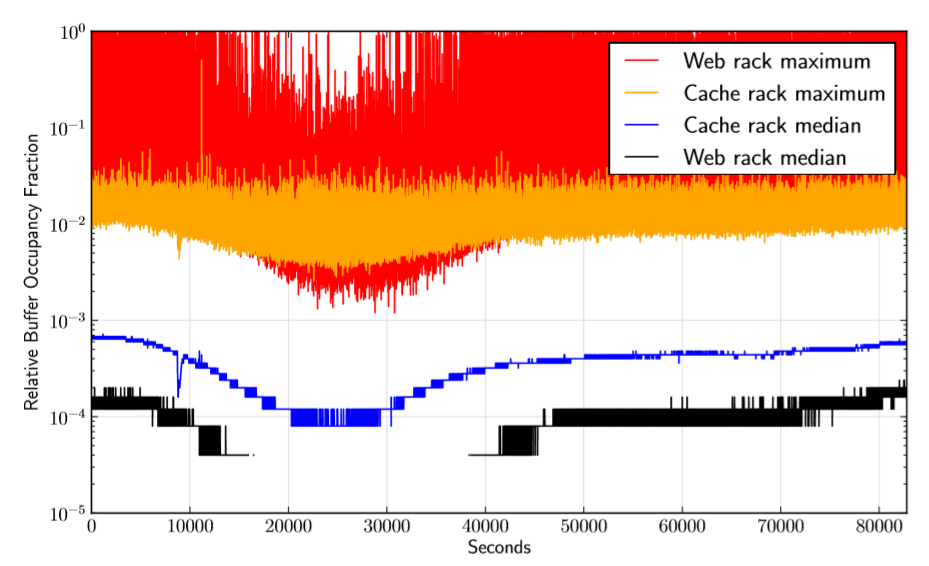


图20 承载不同业务的交换机上buffer的占用比例

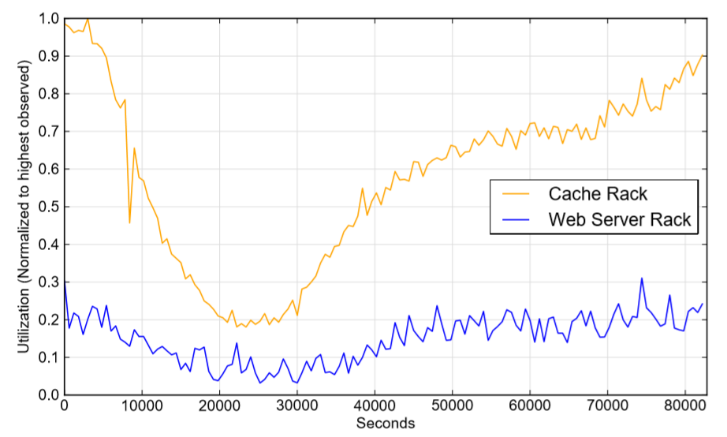


图20 承载不同业务的交换机上的端口利用率

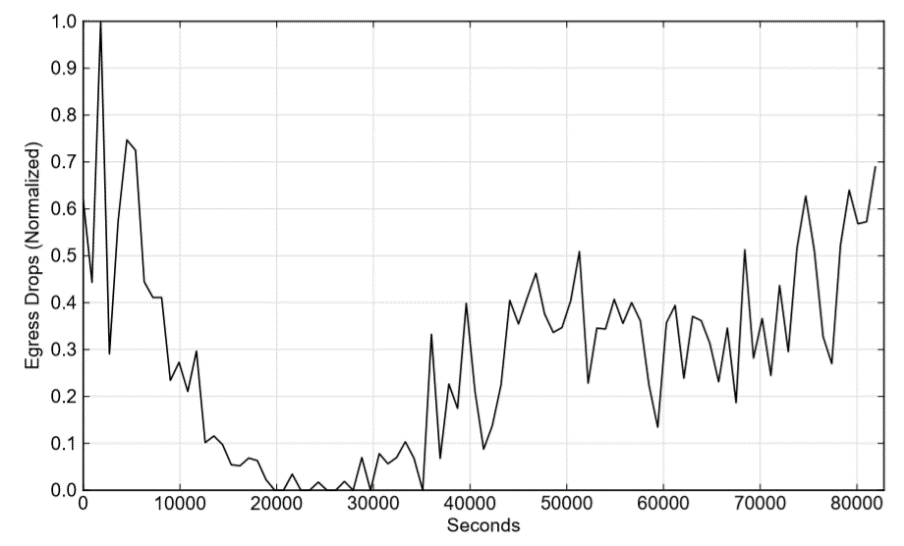


图21 交换机的丢包率

通过以上测量的结果可以发现，buffer占用率在测量的24小时时间段内持续较高，特别是web服务的交换机buffer占用率非常高，尽管其链路利用率大多数时间都在1%附近，但是在每个10微妙的采样间隔内，超过2/3的可用的buffer都被占用，且24小时内有3/4的时间段内buffer占用率最大值都有可能超出buffer占用的上限，而且web服务链路利用率和buffer占用率的变化过程大体一致。不过cache服务的链路利用率和buffer占用率变化关系没有那么明显，而且尽管cache服务的链路利用率要高于web服务，但是其buffer占用率和丢包率要远低于web服务。同时，这两种服务的buffer占用率、链路利用率和丢包率均与时间有关，24小时内不同时刻可能会出现较大差别。

持续较高的buffer占用率可能由于数据包到达率非开关模型、flow到达率较高、单个flow呈突发式传输等因素引起，因此如果数据包尺寸增大、flow的到达速率增加或者多个flow同时突发性传输流量都有可能对buffer占用率造成较大影响。

#### 1.2.12. Flow的并发性

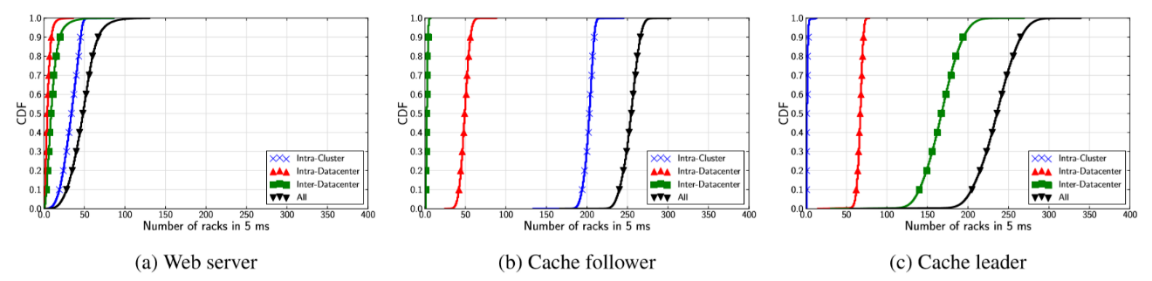


图22 不同业务流的并发特征

这里的并发性是指在5毫秒的间隔内同时传输的flow。因为之前已经验证rack内传输的流量很少，因此可以分别检测三类服务的单台主机在每5毫秒的时间间隔内与其传输数据的不同位置的rack的数量，绘制图22所示的rack数量统计图。

由图22中数据可知，与cache follower主机在5毫秒时间间隔内同时传输数据的rack数量在225-300之间，而与cache leader主机同时通信的rack数量在175-350之间，而他们的rack数量中位数都在250左右，但是他们rack分布的位置却有较大差别，前文测量结果也已验证过这一点。Web服务器同时通信的rack数量在10-125之间，中位数为50。

由于有一些交换机为大流有专门的设计，所以现单独考虑占大部分传输流量的heavy hitter rack的数量，测量结果如图23所示。

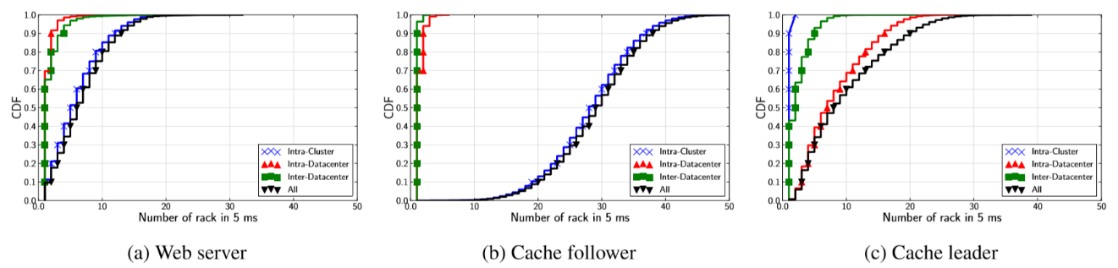


图23 Heavy hitter rack的数目分布情况

对于web服务，heavy hitter rack的数量在6-8之间，而cache leader的heavy hitter rack数量最大值在20-30之间，cache follower的heavy hitter rack数量中位数在29左右，最大值可以接近50上下。再看heavy hitter rack的分布，cache follower和web服务器在与其同一个cluster内的heavy hitter rack占大多数，而cache leader与之相反，cache leader服务器的同一数据中心网络内的heavy hitter rack占大多数。

### 1.3 手机云存储业务特征

手机云存储不同于传统的桌面云存储业务，它有许多新的特点：

手机云存储的传输流量特征遵循包含两个component的混合高斯模型；

上传至云端的文件数目要远高于下载的文件数；

本实验的数据采集自日活跃用户超过百万的一款中国的云存储服务商，用户可以通过其电脑或手机进行上传、下载、删除和分享操作，并且支持多文件的批量操作，上传文件时本地文件也不会丢失。其中数据传输使用的是HTTP服务，每个HTTP的报文都是固定大小512KB（除了每次HTTP传输的最后一个报文），如果超出了固定大小限制的文件则需要被拆分成多个符合大小要求的报文进行传输。对于一个文件，HTTP请求可以使用多个TCP连接进行传送，同时，一个TCP连接也可以传送多个文件的HTTP请求，这时需要按一定的序列逐个发送，并且只有在收到上一个报文的应答后下一个报文才会发送。

无论是上传和下载，用户设备首先会与metadata server联系。上传时，用户先发送一个包含文件名和文件MD5值的metadata至metadata server，metadata server先检查该文件是否已存在于云存储中，如果已存在，则直接将文件加入到该用户的云存储空间并通知用户设备无需上传，如果不存在，则发送给用户设备离其最近的front-end server的身份信息，通知用户设备与之建立连接。用户设备收到front-end server的身份信息后便向其发送一个file storage operation request，里面包含了文件名、文件大小、文件MD5值、待传的chunk数量、每一个单独的chunk对应的MD5值等信息，然后用chunk storage request初始化存储过程。下载时，用户设备先请求预设的metadata server下载文件对应的MD5值，然后使用这个值通过file retrieval operation request去从storage server请求关于该文件的信息，之后再通过chunk retrieval request请求一个一个的数据报文。

这里定义从一个file operation request的开始到下一次file operation request的开始的区间算作一个file operating interval，用T表示；给定时间参数τ，从某个的file operating interval的结束开始到下一个的file operating interval的最后一个报文的末尾作为结束，称其为一个session。

本实验采集的数据是2015年8月里该云存储业务的所有移动端用户的一周的HTTP请求日志，一共从1148640个活跃用户使用的1396494个移动设备中获得了349092451条日志记录，其中一个用户可能在使用多个移动设备。

#### 1.3.1流量大小和文件传输数量的时间分布情况

首先将得到的数据绘制成数据量随时间变化的曲线，如图24所示。

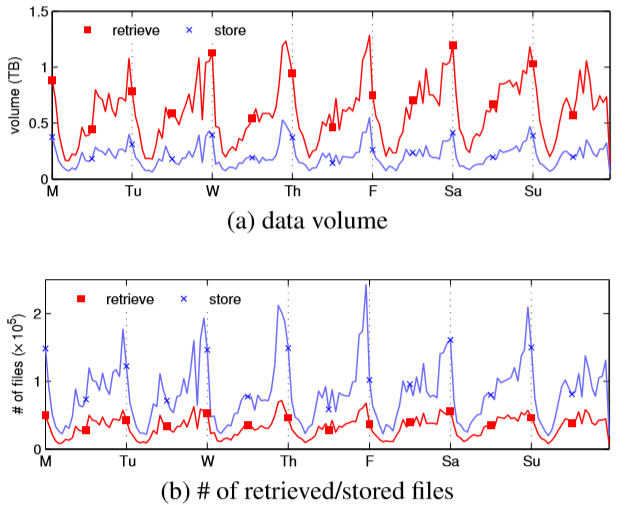


图24 获取和存储的数据量和文件数目随时间的变化

由图可以发现用户的数据流量大小和文件数目整体随时间呈现大致的周期性变化，在每天的23:00左右呈现出一个较高的尖峰，随后在3:00左右为一个低谷，这与手机用户的日常使用习惯相符。其次，对于流量大小，用户下载的流量大小始终都明显高于上传的大小，而对于文件数目，上传的文件数却又明显高于下载的文件数，由此可以推断，手机用户喜爱上传大量容量较小的文件，而下载时都偏向下载大容量文件，且上传的文件数目远高于下载的文件数。

根据分析结果可知，无论是storage server还是metadata server，为了能够应付数据传输的峰值情况，在大多数时候的实际资源利用率都很低，这时就需要使用可以弹性调整其资源大小的服务来减少这种浪费，其次，大容量数据传输对该云存储存储空间和带宽需求也是一个巨大的挑战。

#### 1.3.2. File storage/retrieval operating interval的大小分布情况

通过采集到的数据机绘制出file operating interval的大小分布情况，如25所示：

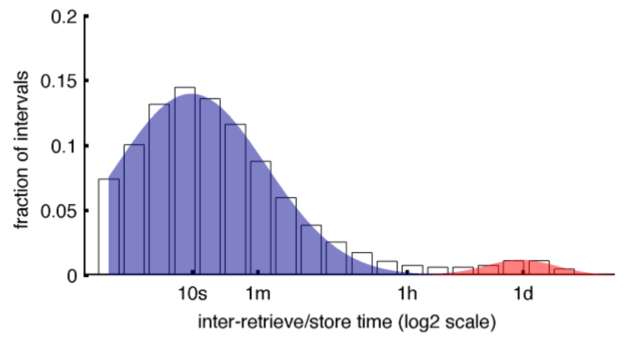


图25 文件操作间隔分布

由图可见，file operating intervals的分布是一条双峰曲线，其中一个波峰对应的interval大小为10s，另一个波峰对应的interval大小为1天，而中间频率几乎为0的波谷对应的interval大小为1h，由之前关于session的定义，这里就推荐令，这样，当file operating interval小于1h时，它与它之后紧邻的下一个file operating interval就同处在一个session内，否则，它传输的最后一个报文的结束就标志着其所在session的结束。具体关系见图26：

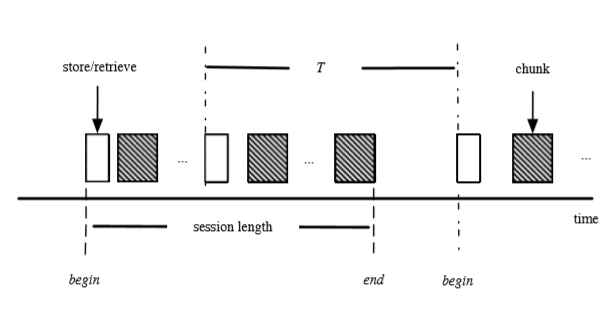


图26 文件传输过程示意图

如此一来，手机传输的流量变化特征就服从包含两个component的混合高斯模型，可以通过expectation maximization(EM)算法求得其各个参数，其中一个component相当于横坐标左边的部分，其均值约为10s，另一个compenent相当于右侧的部分，其均值约为1天。

#### 1.3.3. Session的大小变化规律

图27(a)绘制的是单个session所包含的file operating interval的数量的CDF曲线，图27(b)和(c)描述的是上传和下载两种情况下session的数据量大小随着其包含的file operating interval数量的变化情况：

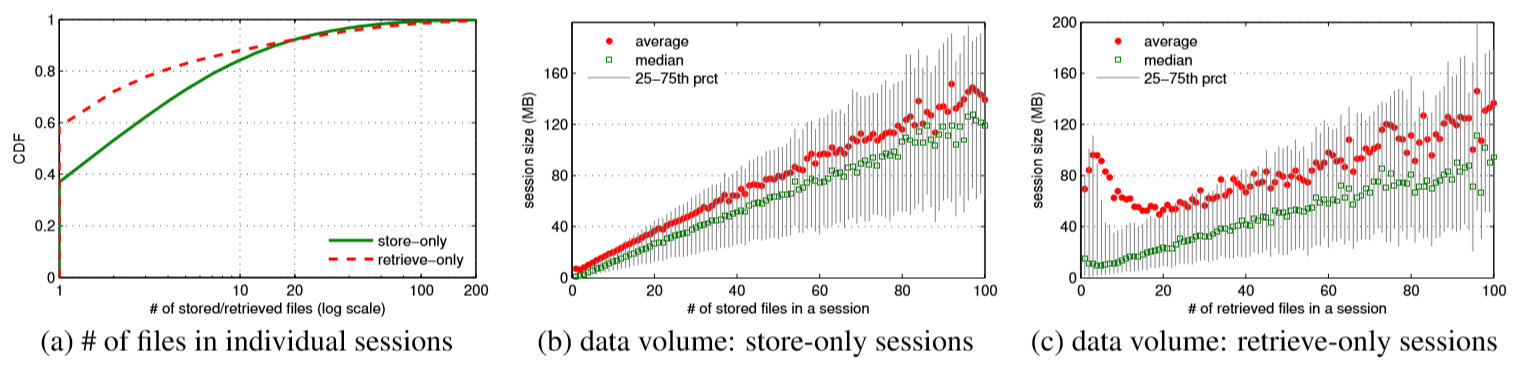


图27 Session中文件获取的特征

由图27(a)可以观察到，差不多半数的session都只包含单个file operating interval，还有大约10%的session包含20个以上的interval，并且下载过程中的session包含的file operating interval的数量总体上要小于上传时的Interval数量。

根据图27(b)可以发现，在上传过程中单个session的数据量大小和其所包含的file operating interval的数量呈线性关系，可以推断上传的每个文件数据量大小基本一致。而在下载过程中情况有所不同，单个session包含少量file operating interval时，其中的文件大小平均值较大，而且随着包含的interval数量的增多，session数据量大小波动也非常大，说明下载过程中传输的文件的大小不尽相同。

### 1.4 视频直播业务特征

Cisco VNI白皮书预测随着网络直播视频取代传统的广播观看时间，它有可能带来大量流量，直播视频流2017年已经占到网络视频流的百分之五，并在2022年其视频流流量将增长15倍，并且占到网络视频流的百分之十七。我们将以Periscope为例分析视频流在两种协议下的延时，流畅，资源消耗等特点。大多直播软件皆类似与Periscope。

Periscope能够让一部分使用者直播给其它使用者观看，并且观看者可以和直播者通过观看着的评论和爱心的方式实时交互。为了最小化直播评论的时延，Periscope只允许观看某一直播前一部分人发表评论，所有观看者都可以看到评论和发送爱心给直播者。在实际中，可以发表评论的用户使用延时较小的RTMP（实时消息传输协议，Real Time Messaging Protocol），其余的观看者使用更易扩展规模的HLS（HTTP 直播流，HTTP Live Streaming）。Periscope的传输模式如图28所示。

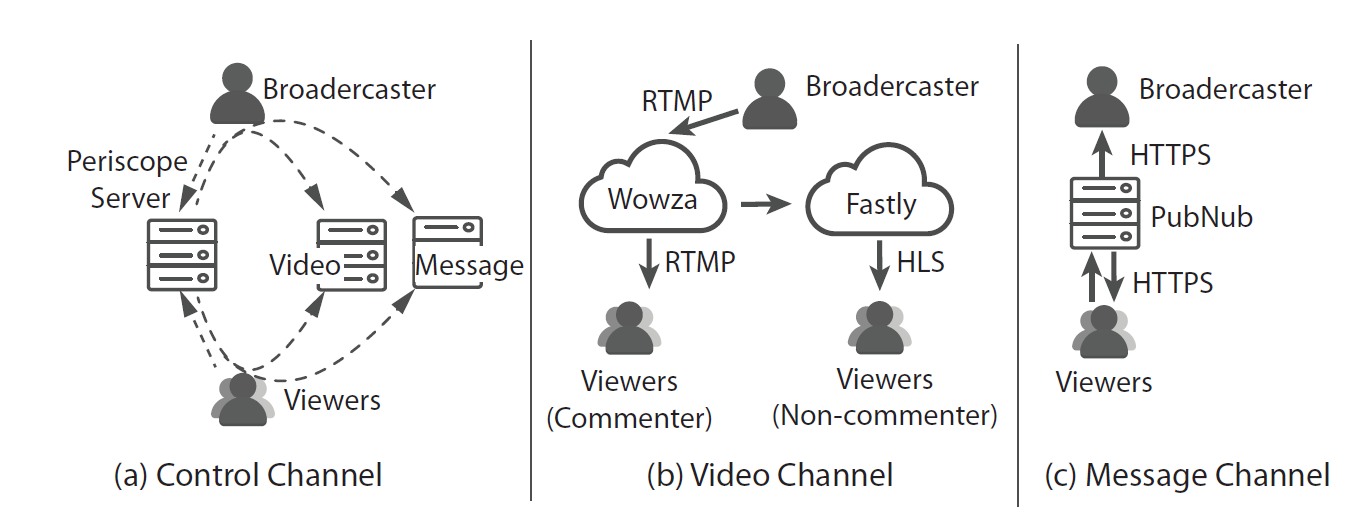


图28. Periscope CDN 结构

Periscope有两个独立的信道传输视频和消息（评论），并根据时间戳将两者结合。在视频信道方面使用两种CDN，Wowza和Fastly分别使用RTMP和HLS协议。RTMP通过push的方法传输视频流。直播者直播时视频以frame格式上载到Wowza，Wowza直接将视频流Push到观看者的终端设备上。HLS协议中，Fastly将获取在Wowza中被封装成chunk的视频流，用户通过poll的方式向Fastly sever来获得视频流，如图29所示。

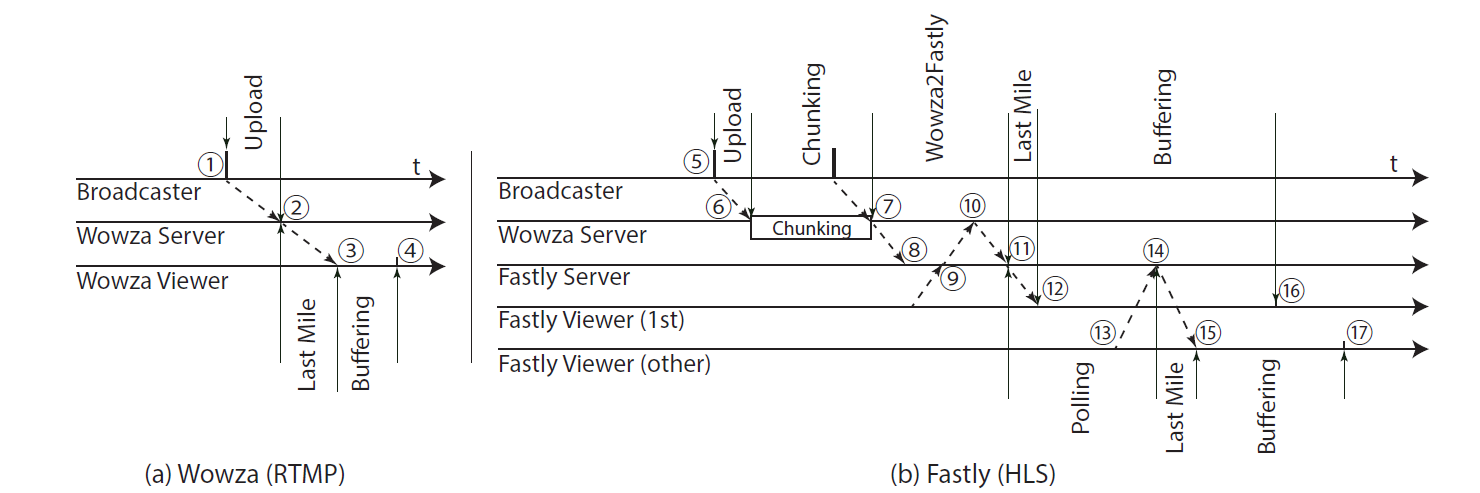


图29 .RTMP/HLS端到端传输

#### 1.4.1 时延

RTMP和HLS通过不同的数据格式传输（frame和chunk，在Periscope中1chunk大于等于75个frame）。并且分别通过push和poll的通信方式。两种协议方式下的时延如图30所示。

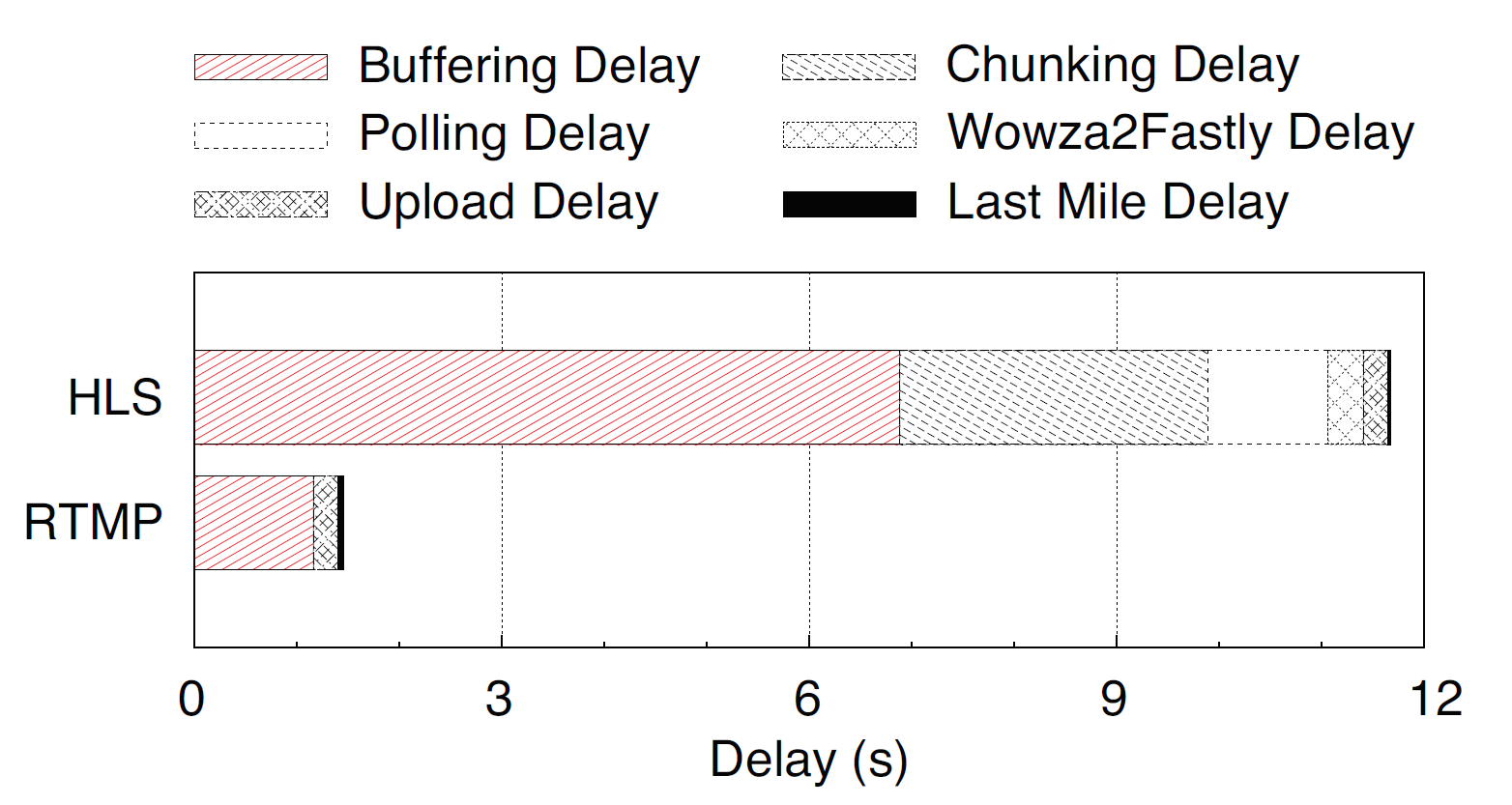


图30. RTMP/HLS时延

#### 1.4.2 流畅度

通过失速率（在观看视频中因为视频不流畅中断的时间与总的直播时间的比值）来定义观看者观看的流畅度。直观的说，在网络环境差不多的情况下，影响视频失速率的原因主要是在用户端缓存视频的时间大小不同。图31图32分别测得的RTMP与HLS失速率与缓存时间的对比（RTMP的往往缓存时间较小，是为了降低时延，以及使用frame格式的原因）。



图31. RTMP失速率与缓存时间

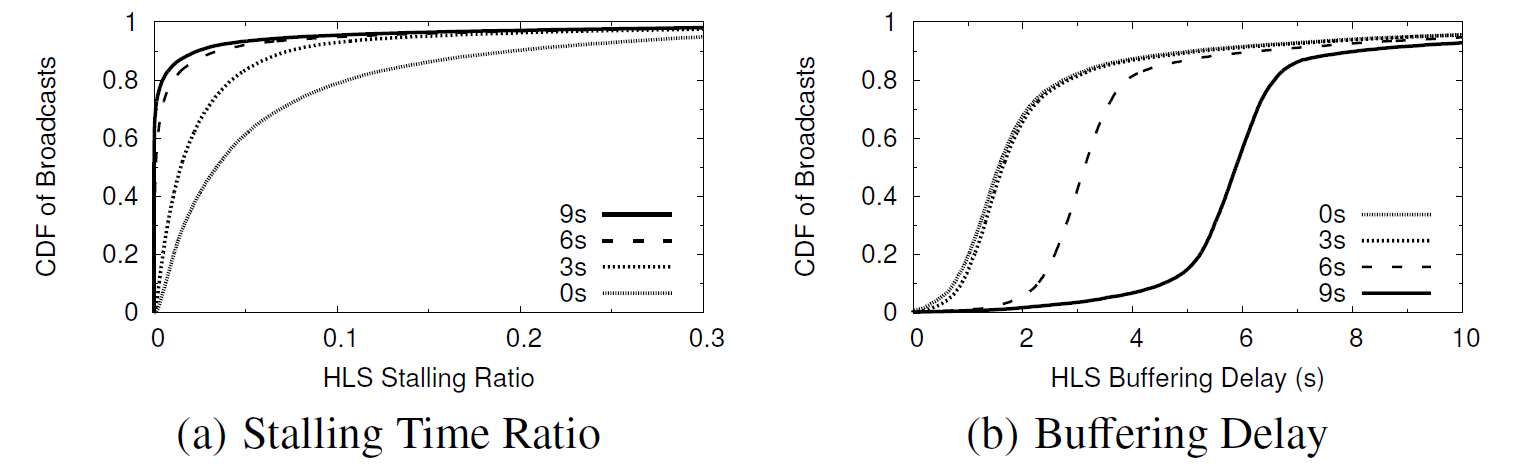


图32.HLS失速率与缓存时间

直播视频流在缓存时间和失速率之间有一个trade-off，通常来说缓存时间长，往往失速率较小，也就是直播视频不会突然中止。因为不追求较小延时，HLS通常缓存时间较长拥有比RTMP更好的流畅度。

#### 1.4.3 资源消耗

RTMP具有较小的延时为何不全部使用RTMP协议的方式来传输视频，因为RTMP以frame格式传输数据，导致了更大的数据处理负载。需要消耗更多的cpu资源。图33表示随着观看人数的上升，RTMP和HLS的cpu利用率。

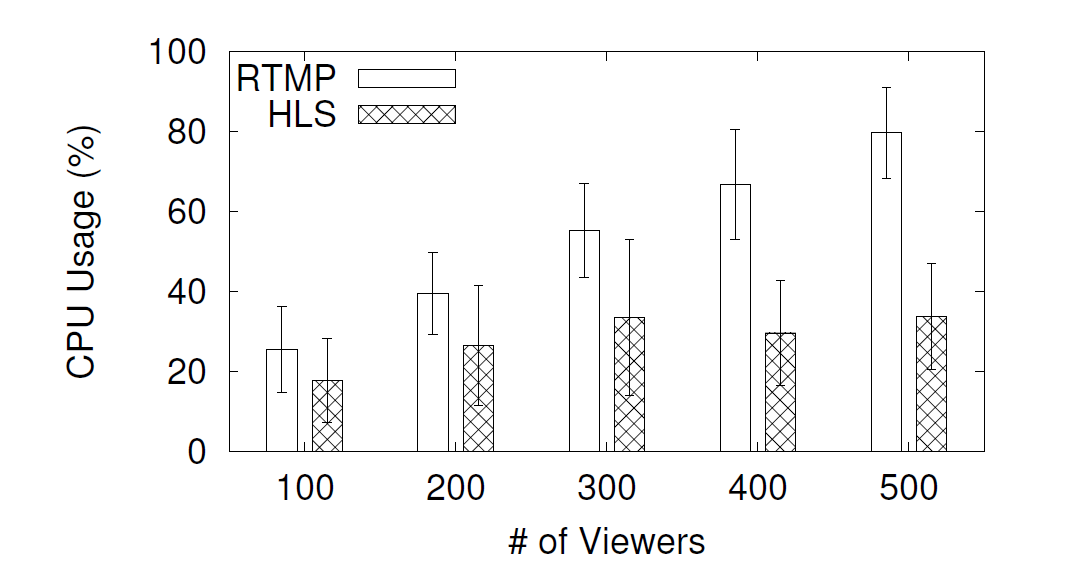


图33.人数上升cpu消耗

通过上图，虽然RTMP的延时很小，具有更好的实时性，但是却不利于规模的扩大。因此Periscope使用RTMP与HLS两种方式。

### 1.5 物联网流量特征

#### 1.5.1 数据来源

物联网应用的网络接入结构如图34所示，分析的数据在图中的支持OpenWrt的无线接入节点上收集。

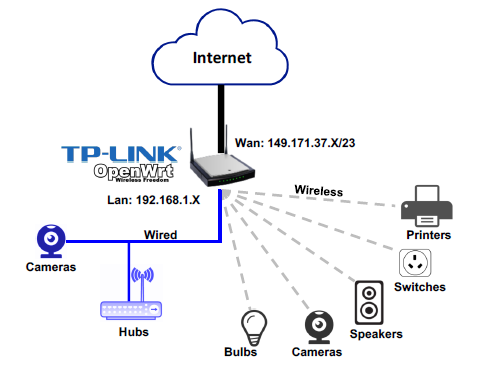


图34 物联网业务的接入网络结构

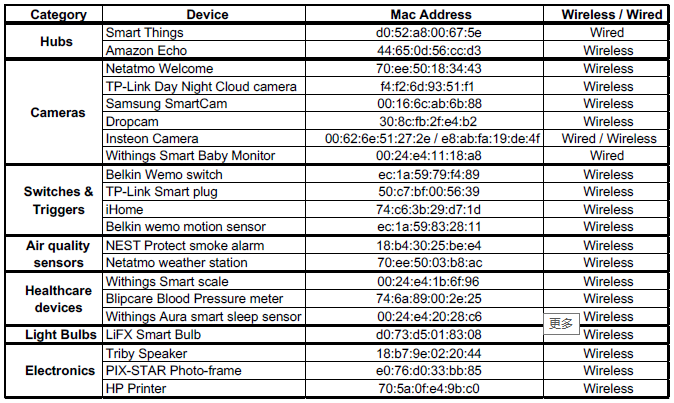
物联网设备通过路由器（AP）来访问外部的网络，在路由器上安装OpenWrt的程序包tcpdump (4.5.1-4)来收集通过该路由器的数据。

图35 物联网设备列表

物联网连接了21个物联网设备，如图35所示，并且除了以上的IoT设备连接到网络，还有手机、笔记本、安卓平板等非物联网设备。

#### 1.5.2 流量变化特征

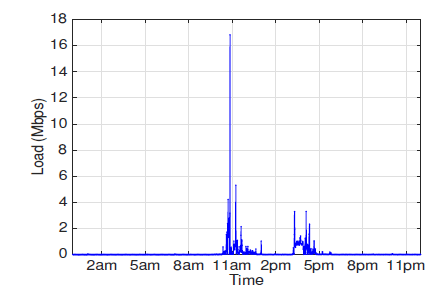


图36 物联网设备和非物联网设备的负载之和

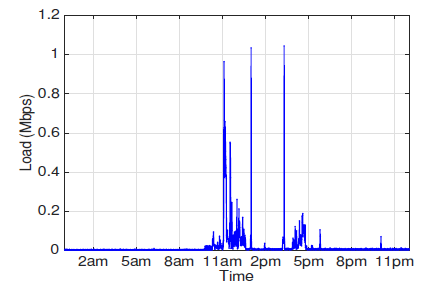


图37 物联网设备负载总和

图36和37给出了物联网设备和非物联网设备产生的流量随时间的变化曲线。由图36和图37可知，总体的网络负载峰值在 17 mbps 左右,而平均负载为 400 kbps。如果我们考虑到只有物联网设备的负载, 峰值负载为 1 mbps 和平均值负载为66 kbps。

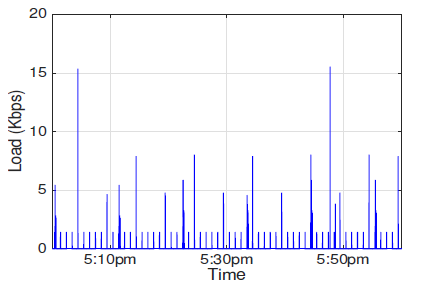


图38 LiFX light bulb物联网设备的流量变化情况

图38给出了LiFX light bulb物联网设备的流量变化情况。从图中，可以看出物联网设备产生的流量有活动和睡眠期的规律。

#### 1.5.3传输数据及包大小特征

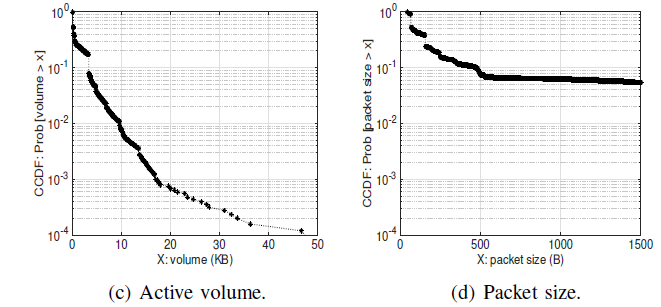


图39 物联网在活动期传送的数据大小和包的大小的分布

由图39（c）可以得出，超过75% 的物联网会话传输数据量少小于 1 kb, 并且只有少于1% 的会话交换超过 10 kb, 这表明大多数物联网设备产生一个小的burst。图39（d）表明大约只有10%的包超过500 Bytes。

### 1.6 Cisco预测的未来5年网络流量特征

Cisco公司预测了从2017到2022年的全球网络流量的变化趋势，主要得出以下结论：

2017年全球IP年总流量为1.5ZB/年或122EB/月，到2022年将达到4.8ZB/年或 396EB/月。

从2017年到2022年全球IP流量的复合年增长率（CAGR）为26%，全球IP流量在2017之后的5年内翻3倍，月IP流量从2017年的16GB/人，增长到2022年的50GB/人。

忙时网络流量比平均网络流量增长地更快。忙时（一天中最忙的60分钟）网络流量从2017到2022以4.8倍的速度增长，而平均网络流量是以3.7倍的速度增长。

#### 1.6.1总流量特征

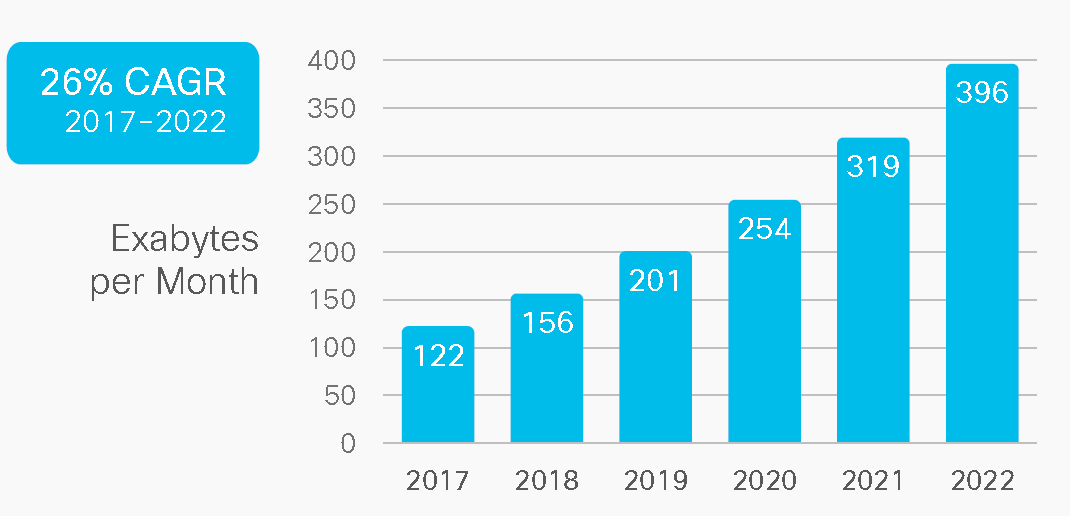


图40. 2017-2022年全球网络流量预测

如图40所示，2017年全球网络流量为122EB/月，到2022年达到396EB/月，5年的复合年增长率为26%。

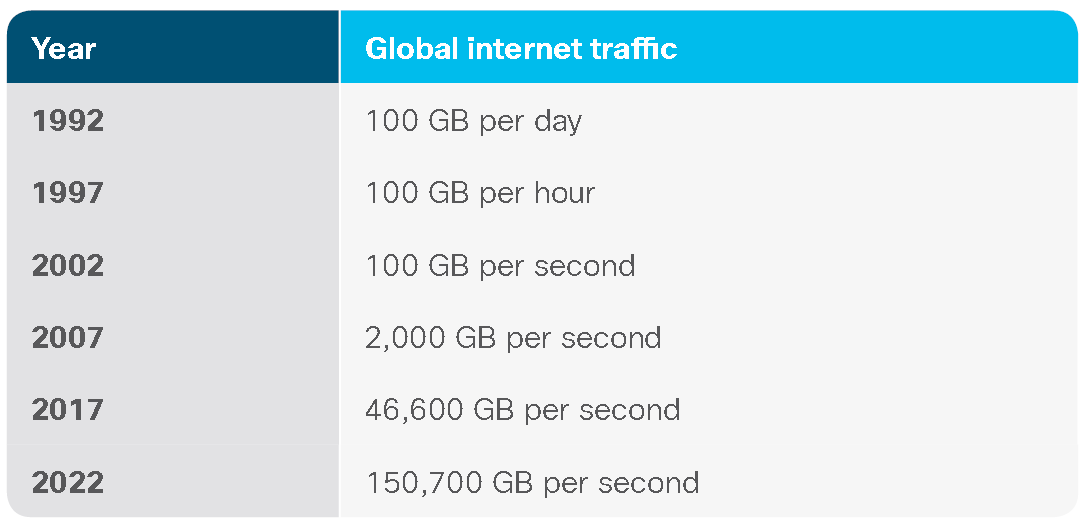


图41. 历史网络流量

在过去的二十年里，互联网的总流量经历了戏剧性的增长。20多年前，即1992年，全球网络每天大约承载100 GB的流量。十年后的2002年，全球互联网流量达到100GB/秒。2017年，全球互联网流量达到45000 GB/秒以上。图41提供了网络总流量的历史基准的视图。

过去10年，人均IP和互联网流量的增长也呈现出类似的陡峭曲线。从全球来看，到2022年，每月IP流量将从2017年的人均16GB增至50GB;到2022年，互联网流量将从2017年的人均13GB增至人均44GB。10年前，也就是2007年，中国的人均互联网流量远低于1 GB /月。2000年，中国的人均互联网流量为每月10兆字节。

#### 1.6.2不同应用类别的流量增长趋势

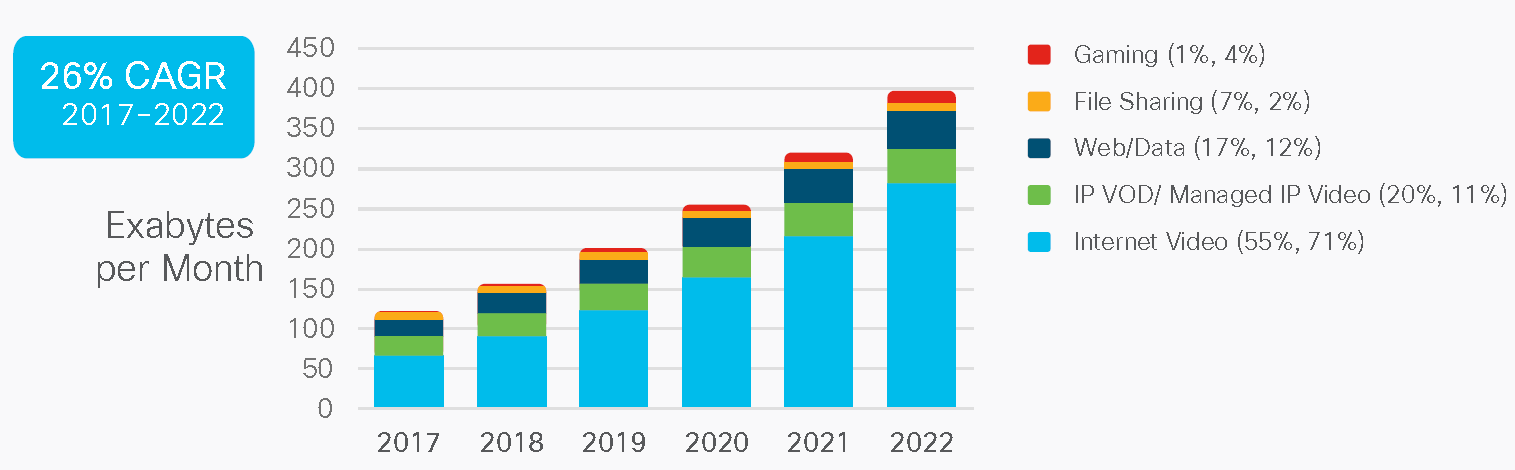


图42按应用类别划分的全球IP流量

如图42所示，所有形式的IP视频，其总和将继续在总IP流量的80%至90%之间。到2022年，全球IP视频流量将占流量的82%。另外，到2022年游戏流量将占总IP流量的4%，这些主要是因为现在的游戏设备提供了足够的空间让玩家可以在网络上下载游戏文件，所以主要的游戏流量是游戏文件的下载流量，而且这些流量主要发生在高峰期，2022年游戏下载流量将占忙时流量的8%。游戏是一种流量形式，它将限制视频流量到2022年超过预期82%的可能性。

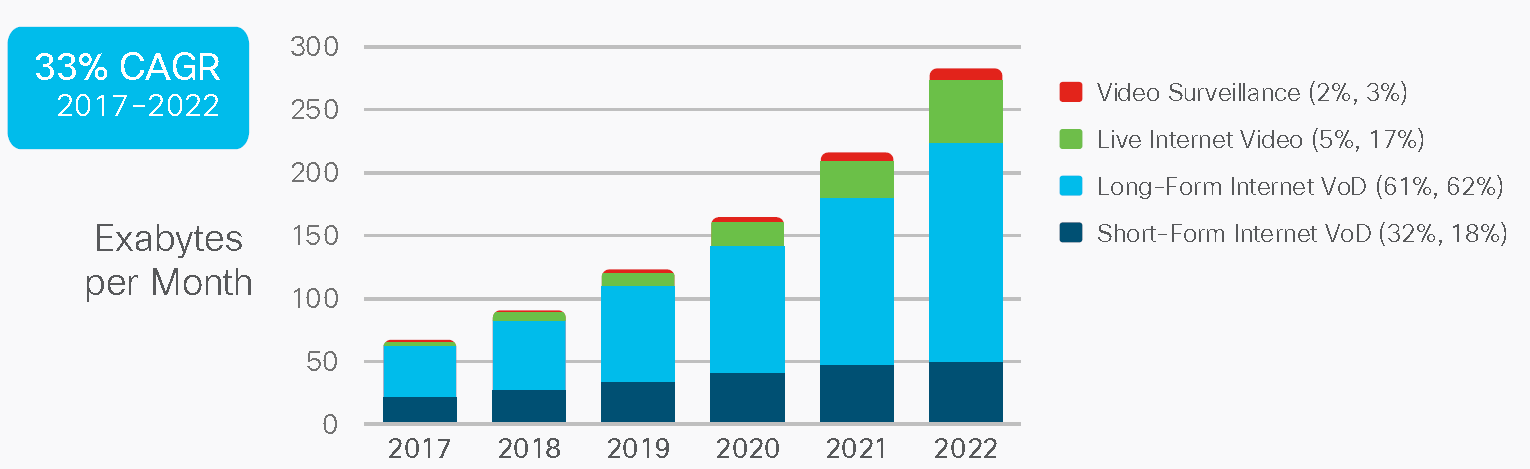


图43 全球网络视频流细分

如图43所示，互联网视频流量本身也发生了变化。特别是，网络直播视频取代传统的广播观看时间，有可能带来大量流量。直播视频已经占互联网视频流量的5%，到2022年将增长15倍，达到17%。此外，值得注意的是视频监控流量的增长。这种流量的性质与直播或VOD流截然不同，它代表着一种稳定的上游摄像机流量，不断地从家庭和小型企业上传到云端。

#### 1.6.3网络宽带速度的增长



图44不同地区的网络宽带（Mbps）

全球平均宽带速度继续增长，从2017年到2022年将翻一番，从39.0 Mbps增长到75.4 Mbps。图44显示了2017年至2022年的预计宽带速度。影响固定宽带速度预测的因素很多，包括光纤到户(FTTH)的部署和采用、高速DSL和有线宽带的采用，以及整体宽带普及率。在本研究涵盖的国家中，日本、韩国和瑞典宽带速度领先，这主要是因为它们广泛部署了FTTH。

#### 1.6.4 流量模式

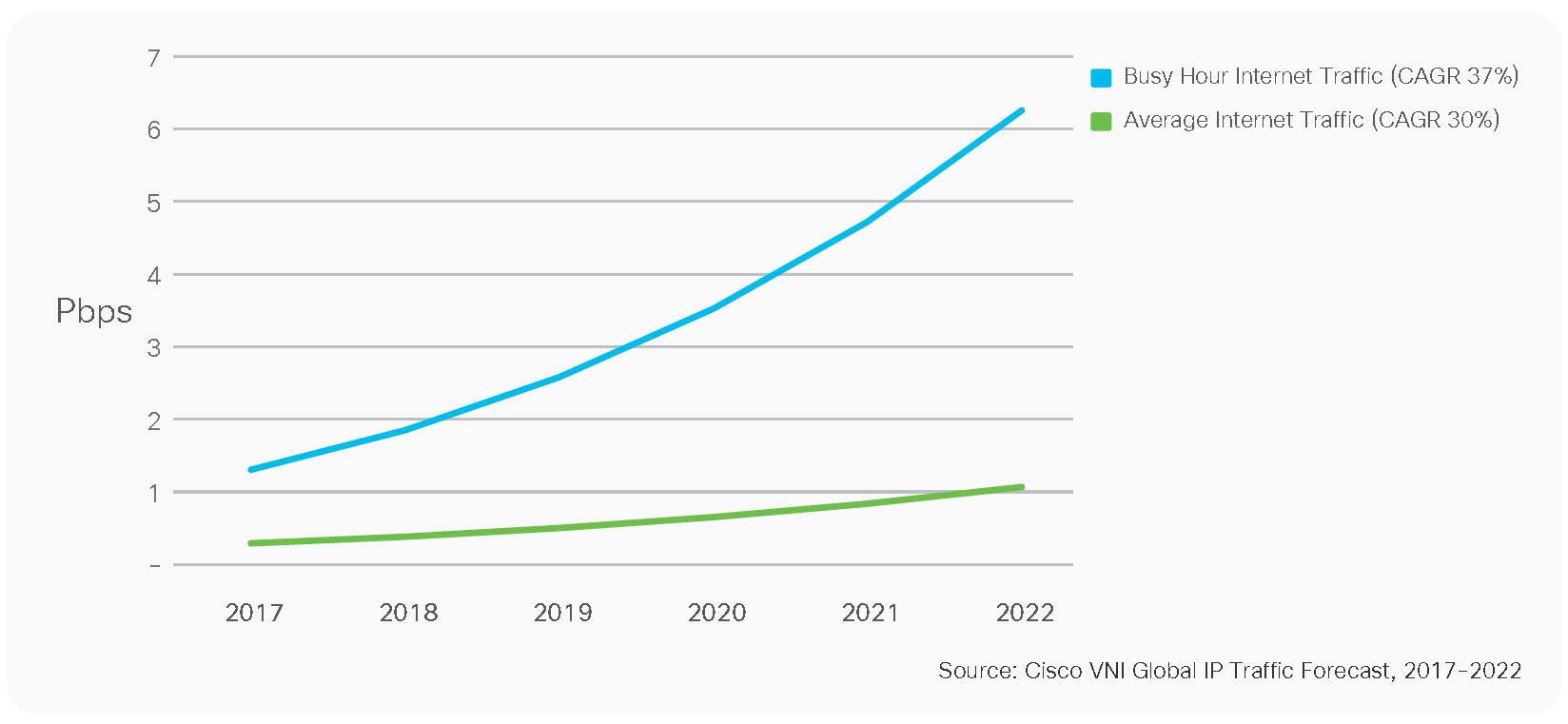


图45忙时流量和平均流量增长

虽然平均互联网流量已经稳定增长，但忙时(一天中最繁忙的60分钟)的流量继续以高于平均互联网流量的速度增长。服务提供商根据峰值速率而不是平均速率规划网络容量。从2017年到2022年，全球繁忙时段的互联网使用将以37%的年复合增长率增长，而平均互联网流量的年复合增长率为30%。

#### 1.6.5移动网络流量特征

从2016年到2021年，全球移动数据流量将增长7倍。从2016年到2021年，移动数据流量将以47%的复合年增长率(CAGR)增长，到2021年将达到每月49.0EB。

到2021年，移动网络连接速度将提高三倍。到2021年，移动网络平均连接速度(2016年为6.8 Mbps)将达到20.4 Mbps。

到2021年，5G将占总连接量的0.2%(2500万)，但占总流量的1.5%。到2021年，5G网络的流量将是4G网络平均流量的4.7倍。

到2021年，视频将占据全球移动数据流量的四分之三以上。从2016年到2021年，移动视频流量将增长9倍，到预测期结束时，将占到移动数据流量总量的78%。

到2021年，移动连接的平板电脑和个人电脑每月将产生8.0 GB的流量，比2016年平均每月3.4 GB的流量翻了一番。与PC和平板电脑相关的总流量将是目前的4倍，CAGR为33%。

到2021年，智能手机平均每月将产生6.8 GB的流量，是2016年平均每月1.6 GB流量的四倍。到2021年，智能手机的总流量将是现在的7倍，CAGR为48%。

到2021年，在所有的IP流量(固定和移动)中，50%将是Wi-Fi, 30%将是有线的，20%将是移动的。

中东和非洲将是所有地区中移动数据流量增长最快的地区，CAGR为65%。紧随其后的是亚太地区(49%)和拉丁美洲(45%)。

到2017年底，中国的移动通信流量将超过美国。到2017年底，中国的移动通信流量将达到每月1.9EB，美国的移动通信流量将达到每月1.6EB。

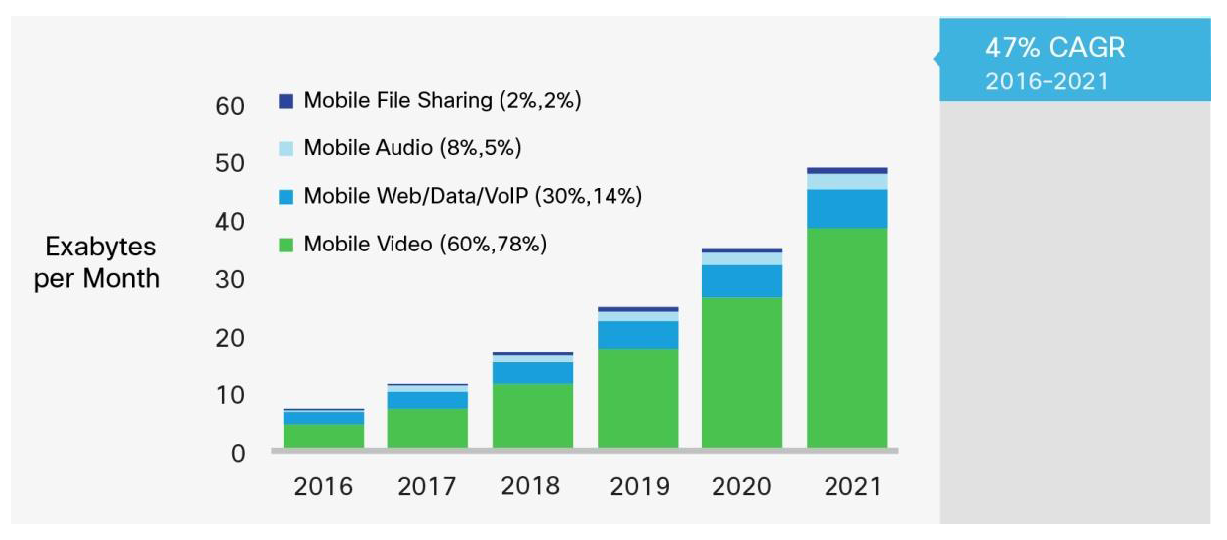


图46不同类型移动应用的数据流量占比

由于移动视频内容的比特率比其他移动内容类型高得多，到2021年，移动视频将为移动流量增长带来很大一部分。如图46所示，从2016年到2021年，移动视频的年复合增长率将达到54%，高于47%的总体平均移动流量年复合增长率。到2021年，移动网络每月传输49EB，其中38EB来自视频。从2012年开始，移动视频占全球移动数据流量的一半以上。

虚拟现实(VR)和增强现实(AR)都将成为移动技术的下一个最大趋势。从即将推出的5G到高效的移动连接解决方案，再加上智能移动和可穿戴设备的接入，边缘计算的发展和无线网络的进步，都为AR和VR的扩散和增长提供了一个丰富的环境。AR和VR的这些创新将对网络的质量和性能提出新的要求。对于高质量的VR和AR体验，带宽和延迟需求将变得越来越迫切，服务提供商将需要注意到这一新需求。

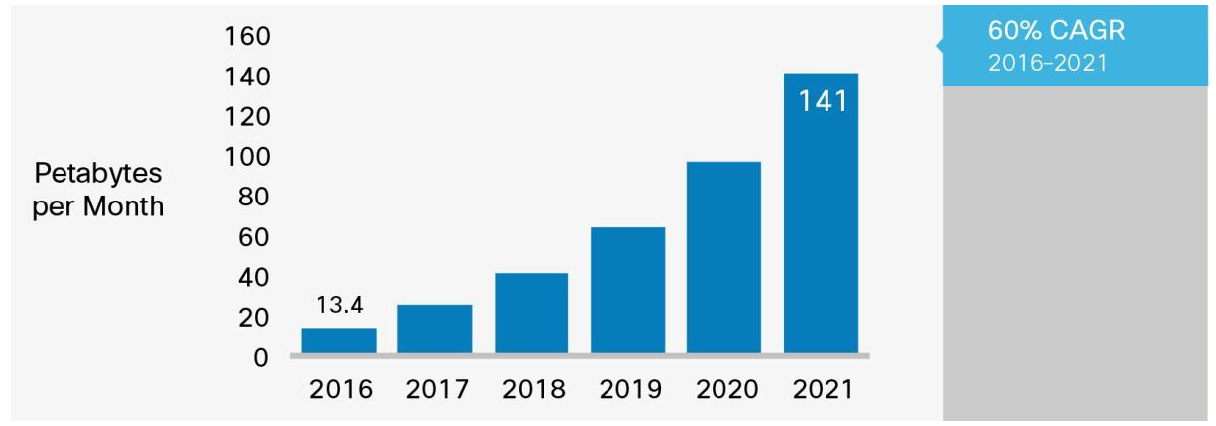


图47. VR移动数据流量

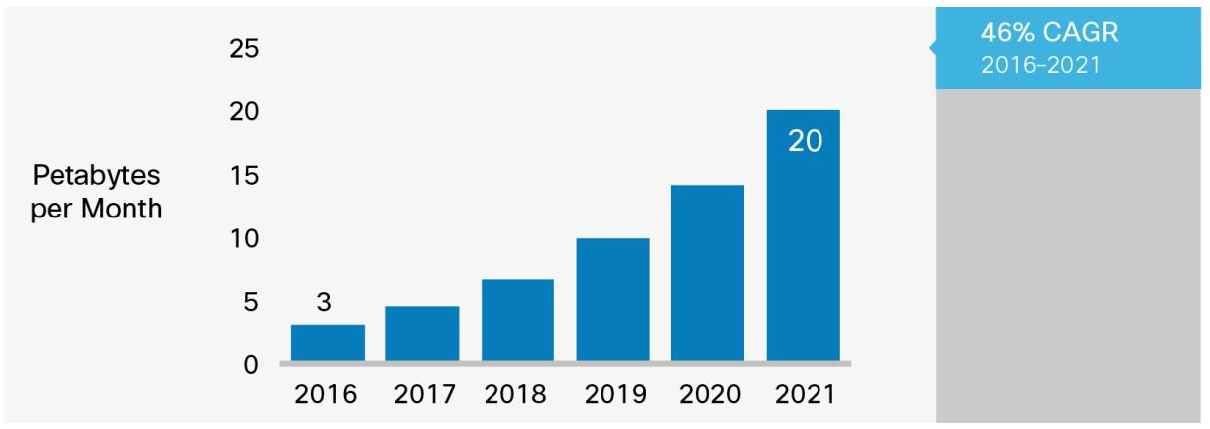


图48. AR移动数据流量

从全球来看，虚拟现实流量将增长11倍，从2016年的每月13.3PB增长到2021年的每月140PB(如图47)。从全球来看，增强现实的流量在2016年到2021年间将增长7倍，从2016年的每月3 PB增加到2021年的每月21 PB(如图48)。

## 2 数据中心网络细粒度负载均衡方法研究

### 2.1 研究背景介绍

通过对之前数据中心网络流量特征的调查研究，可以发现近年来，数据中心网络的性能得到了快速发展，传输性能大幅提升。然而，随着数据中心的不断改进，网络流量传输呈现出许多新的特性：

1. 数据包处理时延降低，交换机发送单个数据包的平均时延只有左右，而单个数据包穿越整个数据中心网络仅需；
2. 网络流量传输呈现短暂突发特性，即发送的流量由许多持续时间在的microburst组成，且不同microburst之间的时间间隔也为不等；
3. 现今商用数据中心网络中的大部分丢包都是由于microburst导致的交换机buffer溢出导致；
4. 商用数据中心网络交换机连接链路的利用率虽然不高，但是buffer占用率却一直保持较高的水平。

由于上述流量特征的存在，现有的绝大多数负载均衡方法都无法在做到有效均衡负载的同时，解决buffer溢出产生大量丢包的问题，一是因为现有大多数的均衡方法的处理时延都在都在几十个毫秒以上，甚至一些中心控制的均衡方法处理时间达到了秒级别，所以面对持续时间只有的突发不能及时均衡流量，从而导致交换机buffer溢出；二是由于负载均衡方法的均衡粒度较粗，很多方法处理的粒度是flow级别，有的是较细的flowlet，但是对于microburst来说依然无法有效分散其流量，使得流量在交换机的某个发送端口出累积至buffer溢出。

### 2.2 负载均衡初步方法设计

通过总结现有均衡方法的优势和不足，我们拟提出两种改进方法，在达到较好负载均衡效果的同时，也能对小时间尺度下突发式流量做出及时反应，减小丢包情况的发生。

#### 2.2.1基于Flowlet的负载均衡方法

现有的基于flowlet的常见均衡方法有CONGA和LetFlow等，但是现有的这些方法都存在着一些局限。例如CONGA，依赖于piggyback的方式来获取网络状态信息本身就会有较大的时延，甚至比数据包经历的RTT还要大，这对于级别的microburst来说太大了，而LetFlow依赖于flowlet的产生机制，即想要改变数据包的发送端口的条件是当前路径上发生了明显的拥塞情况，而我们想要减少microburst导致的丢包就必须在拥塞发生之前就做出处理，所以原生flowlet对于microburst无法有效应对。除此之外，CONGA和LetFlow均需要对网络中的Leaf Switch进行修改，使其能够支持flowlet的产生和发送，这样的方法部署起来并不方便。

我们的改进方法尽可能保留flowlet产生机制带来的网络状态自适应调节特性，同时，还要能够更加主动的应对microburst流量，并且不对现有的数据中心网络做过多的修改以方便其部署。

首先，我们的方法主要依靠数据中心网络中普遍使用的智能网卡，在服务器侧部署flowlet table控制flowlet的产生和发送，而网络侧则简单的使用现今绝大多数数据中心网络都在采用的ECMP方法。这样一来，就保证了不对现有网络做任何修改。

其次，如上所述，flowlet的产生机制过于被动，依赖于当前路径的拥塞情况，所以我们对flowlet table的产生机制进行改进：表项中原有的port number（该值在服务器侧无意义）将变换为counter，用于统计flowlet发送过程中每个内传输的所有数据包的总大小，设置counter的阈值max，如若单个内数据包尺寸之和不大于max，则如同原flowlet做相同处理；若在单个内数据包尺寸之和一旦超出max就立即结束该flowlet，重置该flowlet table表项，并且对新到达的packet作为新的flowlet的开始进行转发。该处理过程如图49所示。

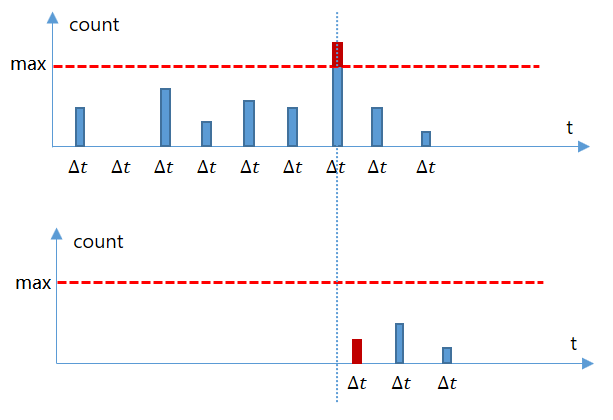


图49 服务器上flowlet的主动切分

发送出去的flowlet在数据中心网络中传输时都视作单独的一个flow（通过五元组定义），即服务器侧的处理对上层网络隐藏。在对同一flowlet的各个packet发送的过程中，修改其五元组中的源TCP port number，在目的主机处接受该packet时将其源TCP port number进行还原。如此一来，同一flowlet的所有packet都会在网络中被认定为一个独立的flow并沿相同路径发送。

除了上述的改进之外，因为我们将flowlet的产生和发送部署在服务器侧，其流量特征和交换机有较大差异，所以我们需要重新确定产生flowlet时规定的刷新间隔的大小，以保证既能够以较细粒度产生flowlet，又能有效减小目的主机处数据包乱序的发生。

#### 2.2.2 基于packet的方式

由于packet是网络中我们可以操作的最小粒度，所以基于packet的负载均衡方法在理论上可以获得最佳的负载均衡效果。现有基于packet的负载均衡方法有Drill和Hermes等。

Drill主要是部署在Leaf Switch上，且需要通过中心控制器为各个leaf switch下发子拓扑信息，这对其部署造成了较大的不便。其次，在对称性较差的网络中，由于每个flow是在某个子拓扑内传输，所以导致其对ECMP方法的提升非常局限。

Hermes方法部署在主机侧，它通过主动发送probe packet和接收piggyback回传的链路拥塞信息这两种方式来获取路径状态信息，并实时动态调整packet的转发路径。然而它调整packet转发路径的机制非常谨慎，尽管这可以减小路径调整对传输性能的影响，但是这却使得它对于microburst无法及时处理。

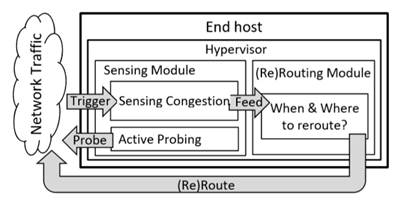


图 50 基于packet的负载均衡方法

现有方法的这些局限给了我们设计的方向，于是我们设计了新的基于packet的负载均衡方法。首先，我们设计的基于packet的负载均衡方法依然是部署在服务器侧，从而使得该方法在数据中心网络大规模部署情况下更加可行。图50给出了我们设计方法的系统架构。

在图50中，sensing module的sensing congestion部分是根据传输flow时对方回传的ACK报文来检测网络状态，而active probing部分是服务器周期性发送探测报文来检测网络状态，根据检测到的状态使用routing module来决定是否改变发送路径。

具体数据包发送过程需要分为两种情况，即对称网络和非对称网络。在对称网络下，发送的数据包将会尽可能平均的分配至各个可用路径，以保证各条路径负载的均衡。而在非对称网络中，可以先将源宿节点对之间的所有可用路径划分为一个个子拓扑，子拓扑即为可用路径的集合，使得在相同子拓扑内部的各条路径相互对称，同时，子拓扑划分时应保证子拓扑的数量尽可能少。Flow在转发时，会选择其中某个子拓扑，在选定的子拓扑内部按照对称拓扑下的发送方式发送packet。。我们初步设计的基于packet的负载均衡方法的具体过程如下所述：

首先，sensing module中的active周期性的检测各个子拓扑的状态，即通过发送probe packet的ECN和RTT来综合判断该子拓扑的传输质量，而每个rack只需要一台服务器作为probe agent来与其他rack的probe agent相互发送probe packet就可以，这样可以减小检测过程引入的网络资源开销。而在数据传输过程中sensing congestion也会根据回传的ack的ECN和RTT，实时更新当前传输的子拓扑的状态。

然后在有新的flow需要发送时会根据上述两种方法检测到的所有可用子拓扑的状态信息选择一条较好的进行发送，每个flow每次只能在一个子拓扑上传输。

在flow传输过程中，会根据其实时更新的子拓扑状态信息判断是否切换其他子拓扑进行发送，判断机制的设定较为谨慎，即防止频繁切换带来的性能损耗。

通过该改进方式可以实现，在网络较为对称时，所有可用路径划分的子拓扑数目较少，且每个子拓扑中的路径数较多，而在网络对称性较差时，子拓扑数目会变多，且每个子拓扑中的路径数会减小，最终均衡效果类似于Hermes。不过无论对称性如何，当遇到microburst突发流量时，子拓扑中多条可用路径会分散掉该突发带来的数据包，减小microburst导致丢包情况的发生。

### 2.3 性能验证

设计出以上两种方法后，下一步我们将着手通过仿真对比实验，对这些方法进行性能评估，以验证其是否能够在保证较好均衡效果的同时，减少microburst导致的丢包情况的发生。

我们的实验平台为ns3仿真平台，将通过模拟数据中心网络的拓扑，测试基于改进的flowlet的均衡方法和CONGA和LetFlow之间的性能差异，以及基于packet的方法和Drill和Hermes的差异，最终撰写实验报告验证我们的结论。

## 3 数据中心网络流测量方法研究

### 3.1 背景介绍

在数据中心网络中，网络测量在网络监控中起着非常重要的作用，而一个好的测量方法是提高网络性能的一个重要途径。网络测量解决的具体问题主要有两种：一种是大流(Heavy Hitter)的检测，或者说是top-k流的检测，这个问题要求能够通过某些算法从网络中的众多流中提取出大流进行存储，并能给出这些大流的流量值，而现在针对这种问题的测量的方法一般是基于Counter的测量方法，用有限的Counter统计网络中的大流，经典的基于Counter方法主要有Space Saving，Frequent，RAP等。另一种是流量矩阵估计问题，要求在给定一个流的ID，就能给出这个流的真实大小或者估计值，而这个估计值的误差需要保证在一定的范围中，解决这种问题主要是通过基于Sketch的网络测量方法，通过在数据平面安装Sketch数据结构，将网络流哈希到对应的桶中计数，查询时也是从哈希表中读取相应哈希值的计数值，经典的基于Sketch的方法主要有CM Sketch，C Sketch，CU Sketch等。现有的测量方法为了适应现在高速网络，都在最求常数时间复杂度，因此我们提出的测量方法也要求在常数时间内进行网络测量。但是现有方法还有许多不足之处，例如：不能同时解决Heavy Hitter和流量矩阵估计问题，不能适应复杂网络流量分布多变的情况，不能给出一个确切的误差范围等等。下面详细介绍了Space Saving和CM Sketch算法，并针对他们的不足之处，提出了两种数据中心网络流量测量方法。

### 3.2 基于Counter的测量方法

基于Counter的测量方法，主要是通过有限的计数器存储有限的大流，而小流则是忽略或是只给出一个大致估计值，所以基于Counter的测量方法给出的误差界一般都是确定的。目前基于Counter的测量方法主要有Frequent， Space Saving，RAP等，这些方法的主要思路都是在有限的计数器中只记录大流的数据包，而忽略小流的数据包，只是当计数器的个数达到上限时，处理的方法不一样。这些方法能够保存大流在有限的Counter中,并且采用hash操作能够使得运行时间在常数时间范围中。但是这些方法的缺点在于只能解决Heavy Hitter或top-k问题，对于流量矩阵估计问题解决得不够好，并且由于采用hash操作，不能很好的处理hash冲突而导致运行时间变长。

#### 3.2.1 Space saving算法

Space saving算法能够保证在Counter数足够的情况下，能够保证top-k条流都被记录下来，并且算法能够达到常数时间复杂度，对于Heavy Hitter的检测有非常好的性能。主要思想如下，T保存所有Counter，每次数据包i到达时，若没有空闲的计数器，它那么这个新到达的数据包将替换所有计数器中计数值最小的那一个j，而新的数据包的计数值为j的计数值加1，同时还将替换前的计数值记录到中。具体的算法如下：

**SpaceSaving**(m)

**1** n0;

**2** ;

**3 for each** i **do**

**4** n←n+1;

**5 if** **then**

**6 else if** **then**

**7** ;

**8** ;

**9**

**10 else**

**11**

**12** ;

**13** ;

**14** ;

#### 3.2.2 Space Saving实现方式

通常Space Saving可以采用链表形式的数据结构Stream Summary实现，可以达到常数时间复杂度。Stream Summary将计数值放在一个桶里面，然后每个桶用双向链表按计数值从小到大的顺序连接，记数值相同的counter用链表连接到相应计数值的桶中，每次查找最小的counter只需要在链表头中查找就行了，如图51所示为的Stream Summary实现Space Saving算法的示例：



图51 Stream Summary实现Space Saving

图51所示示例最大计数器个数为2，数据流为S={X,Y,Y,Z}，(a)(b)(c)分别为每个数据包到达时Stream Summary的状态。(a)当X和Y到达时，由于计数器个数没有达到上限，所以这两个包都被记录下来，并且指向了value=1的桶中；(b)当又到达一个Y时，由于Y已经记录在链表中，则增加一个value=2的桶，将Y移动到value=2的桶中；(c)当Z到来时，由于没有多余的Counter了，所以将最小的value的桶中的X替换为Z，记为X的计数值，再将这个Counter移动到value=2的桶中，删除value=1的桶。利用这种数据结构可以达到时间复杂度为O(1)，因为每次查找最小计数值时只需要在桶的链表头查找即可。

#### 3.2.3 Space Saving算法分析

显然Space Saving算法需要的空间为O(k)，如果利用Stream Summary数据结构可以达到运行时间为O(1)。对于没有记录在Space Saving的小流可以就用最小的那个Counter值来估计，但是这对于这些小流的估计值的误差是比较大的。

对于任意分布的数据流来说，要想确保检测到误差为松弛的top-k流(即假设表示第k大的流的值，表示流*i*的值，对于任意满足的流*i*，一定记录在Space Saving中)，必须保证Space Saving的Counter数为，其中表示流的个数。

对于常见的Zipf分布的数据流来说，若参数，只需要Counter数至少为，即可以保证准确地解决top-k的流的检测问题；若参数，只需要Counter数至少为，即可以保证准确地解决top-k的流的检测问题。

#### 3.2.4其他方法

除了Space Saving以外，经典的基于Counter的方法还有Frequent，Lossy Counting，RAP等。Frequent当Counter满的时候是通过每个计数器都减1的操作，然后将新流替换计数值为0的计数器，相比Space Saving来说测量误差更大；RAP与Space Saving不同的在于当新的流的数据包到来时，并不是所有的包都替换最小的Counter，而是按照最小计数值的倒数的概率来替换最小Counter，这样可以使得大流的计数值增长得越来越快，小流的计数值增长得越来越慢，这样可以减少大流被小流替换的概率，这样使得RAP需要的Counter数比Space Saving更少。

### 3.3 基于sketch的测量方法

这一部分介绍基于sketch的测量方法。3.3.1节以CM sketch为例，详细介绍基于sketch方法的基本流程，3.3.2节对CM sketch进行误差分析，在此基础上3.3.3节介绍C sketch与CU sketch的基本原理，3.3.4节对于sketch测量方法的优缺点进行讨论。

#### 3.3.1 Count Min sketch（CM sketch）的基本原理

Packet

h1

h4

图52 CM sketch的基本结构

CM sketch的基本结构如图52所示，直观上看，CM sketch数据结构的主体是一个由计数器组成的*d*（图中*d=*4）个长度为*w*（图中*w*=5）的哈希表组成二维数组，并且，这*d*个哈希表所选用的哈希函数彼此无关。

在收到数据报文时，CM sketch首先提取数据报文所对应的流ID。继而，将该ID同时在*d*个哈希表中所对应的计数器值增加该数据报文的权重（例如数据报文的长度）。

由于每个流ID所对应的计数器都至少记录了该流的流量，所以，给定流ID，该ID所对应的任一计数器值一定大于其真实值。因此，在查询流量时，CM sketch在其对应计数器数值中取最小值作为最终的查询值。在下一小节中，将会具体分析这一查询值的误差。

#### 3.3.2 CM sketch的误差范围

将所有流量之和记作。首先考虑单个哈希表的查询误差，对于任意一条流，它在该条哈希表上查询误差的期望为，并且该误差值非负。由马尔科夫不等式，该查询误差大于的概率至多为1/*e*，其中，*e*为自然对数。将*e*/*w*记作ε，对于单个哈希表，我们有：

现在综合考虑全部的*d*个哈希表，在所有哈希表中，查询误差全部大于的概率为,将记作*δ*，即*d*=ln(1/*δ*)，那么对于任意流，我们有：

即，CM sketch至少以1-*δ*的概率，查询误差小于。

#### 3.3.3 C sketch与CU sketch简介

C sketch与CU sketch均是在CM sketch的基础上修改得到的网络测量模型。C sketch在哈希表的更新策略上进行了更改。对于每一个哈希表，C sketch都额外维护了一个哈希函数gi(·)，该哈希函数的值域为{-1,1}。在进行计数时，C sketch会依照gi(·)的值决定在该计数器上执行相应增加或减少操作。而在查询时，C sketch将gi(·)与对应的查询值相乘，并返回这些查询值的中位数。

CU sketch直接根据CM sketch出现误差的原因进行了改进。CU sketch在进行更新时，仅对到达流ID对应的所有计数器中值最小的进行更新，在保证CM sketch优点的同时，减小了其可能出现的误差，但相对来说维护过程较为复杂。

#### 3.3.4 sketch的优缺点

sketch最大的优点就是它通过哈希存储避免了存储流ID（通常为104比特），解放了存储资源，使得其在相同存储空间下，计数器数量是基于counter的6~8倍。由于counter数量众多，因此sketch可以准确地测量小流的流量。

sketch的缺点主要体现在两方面：由于未存储流ID，因此，sketch只能被动地返回流量的查询值，而没有有效的方法单独维护大流计数，使得sketch无法及时返回异常流量；此外，由于交换机处理速度的限制，sketch的宽度*d*通常为4，无法保证单个流的测量精确程度。

### 3.4 基于Sapace Saving和FlowMap的混合测量方法

针对Space Saving算法不能够很好地估计小流的流量这一缺点，我们提出了一种基于FlowMap的检测方法。主要思想是先将数据包加入一个Space Saving中，然后将Space Saving替换出来的小流加入到一个FlowMap中，这样大流被留在了Space Saving中，小流被加入到FlowMap中。而小流加入FlowMap的过程主要是通过两级hash操作，第一级hash先将流hash到一个Counter分组中，然后第二级hash决定将它的值加入到哪一个Counter中。在估计时则根据两次hash的映射关系构建出汇聚矩阵A，求解Y=AX来估计流量矩阵X，其中Y是Counter上的值。

#### 3.4.1 FlowMap的主要步骤

如图53所示，首先当数据包到达时先进过一个BloomFilter确定这个包是否是已经记录过的流，如果不是，则将其ID记录到流ID表中，如果已经记录过，则直接进入Space Saving中；然后若从Space Saving中有被替换出来的流，则将它的ID和计数值经过第一级hash，选择一个Counter组；再在相应的Counter组中进行第二级hash，分别hash到相应的Counter中计数。

对于每一个counter组，我们可以根据流的映射关系，得到一个方程组AX=Y，其中A就是流的映射矩阵，X为流的大小，Y为counter值的大小。如果A的秩等与流的数目，那么我们可以唯一的确定每个流的大小，但是如果counter数目受限的情况下，我们无法准确的得到流的大小，为此我们可以采用流估计方法，设计优化目标。我们可以将流估计问题建模为如下的优化问题：



显然，上面的优化问题是一个线性规划问题。问题求解的复杂度主要由约束和变量的数目决定。为了降低求解的难度，我们将counter分成了若干的组，这样每组counter只对应少量的约束和变量，而且每组counter记录的流是完全不同的，因此可以并行求解。



图53 FlowMap步骤

#### 3.4.2 FlowMap分析

相对于Space Saving来说，FlowMap将小流也进行了统计，而且通过hash操作，在更新的时候可以达到常数时间复杂度。另外由于FlowMap进行了分组，每个Counter分组中保存的流是不同的，因此每个分组都可以构建一个相互无关的线性规划问题，减少了问题的规模，而且可以并行计算。

但是由于Space Saving的误差，有可能会出现大流被替换出来加入到FlowMap中的情况，并且由于BloomFilter也有误差，可能会时记录的流ID比真实情况少，这两种情况就可能使得流量估计误差变大。

#### 3.4.3 当前研究进度及下一步工作

目前正在进行实验仿真评估中，我们从网上下载真实的网络流量数据集进行仿真实验，并以估计流量矩阵的均方误差作为评估数据。在之后的工作中，我们希望能采取一些措施够减少FlowMap的误差，并且希望在NS3平台中进行网络范围的仿真实验。

### 3.5 sketch与counter的混合测量方法

Sketch与counter两类方法各有优劣，而我的第一部分工作就是尝试将sketch与counter两种测量方式进行混合，力求获得更为全面的测量性能。

#### 3.5.1 基本算法流程

Packet

Space Saving

Sketch

替换流量

图 53 sketch与counter的混合模型

混合模型的基本更新流程如图 53 sketch与counter的混合模型所示。在收到报文之后，混合模型首先尝试在Space Saving中进行更新。如果在这一过程中出现了替换操作，则将Space Saving中被替换掉的流ID及其对应的流量记录到sketch中。这样，就保证了所有到达的流量在Space Saving与sketch中记录且只被记录了一次，其算法如下所示。

**Function UPDATE（ID, v） #流ID及其流量**

**1**在SpaceSaving中更新

**2 if** SpaceSaving 中发生了替换：

**3** 在Sketch中更新被替换掉的ID及对应的c-ε

**4 *end if***

**Function QUERY（ID）**

**5*****if*** Space Saving中记录了ID：

**6** ***return***SpaceSaving.query(ID)

**7** **else**:

**8*****return*** sketch.query(ID)

#### 3.5.2 当前工作进展

现阶段已经搭建了一个算法验证平台，并且在UCLA数据集上测试了该混合算法，该测试结果显示在总存储资源消耗相同的情况下，混合算法对于流量测量的精确程度要比单纯的CM sketch与Space Saving都高，这是由于混合模型中的Space Saving截断了流入sketch中的大流流量，提高了其测量的精确性，使其在counter数量不占优的情况下反而比单纯的sketch测量的更准确。

#### 3.5.3 下一步工作内容

接下来的工作将在两个方向上进行：

一方面，下一步将在NS3中仿真实时的交换机环境，测试混合算法的吞吐量，并对其进行改进。

另一方面的工作将集中在将不同的sketch模型与counter模型进行混合，并分别进行算法验证，测试其测量的准确性。

## 在网计算应用研究

### 4.1 简介

SDN解决方案将控制平面与转发平面分离，并为我们提供了控制平面的可编程能力，目前人们提及最多的OpenFlow协议在逐渐的完善演化过程中，表字段和表类型不断的增加。白牌交换机在支持OpenFlow协议的版本更新上，面临着和传统交换设备厂商同样的困境——OpenFlow并不支持弹性地增加匹配域支持，协议新特性的支持所需要的成本大、时间周期长。同时，随着网络中新的协议不断出现，OpenFlow协议也必将变得越来越臃肿，表的扩展也必将变得越来越困难。转发平面编程语言P4和伴随它出现的新一代可编程交换机为运营商提供了转发平面的可编程能力，打破了硬件设备对数据转发平面的限制,让数据包的解析和转发流程也能通过编程控制，使得网络及设备自上而下地、真正地向用户开放。新一代可编程交换机的出现，它的灵活性和强大的能力为在网计算技术注入了新的血液，带来了很多新的思路。

网络拥有了计算能力，可以将分布式服务嵌入可编程交换机，这样可以充分的开发网络的潜力。原本需要在远端服务器中去争抢资源的分布式服务，可以在近端的交换机上直接解决。这提高了网络中的吞吐量，降低了包交换的延时。由控制器掌控全局，实现全局的管理功能，容错以及更新。再由P4编程语言自定义的灵活的包转发策略并且可以实现数据的key-value缓存。

发掘新的在网计算技术的应用也具有深刻的研究意义，希望通过本文研究进一步挖掘在编程交换支持下的在网计算技术的潜力。把在网计算技术应用到现在与人们生活息息相关的大规模数据的机器学习方向。把可编程交换机提供的缓存功能加入分布式机器学习架构，成为其中的重要一环。

现在已经确定将在网计算用于机器学习具体问题是利用因子分解机（Factorization Machine，FM）来解决广告预估模型。广告预估问题有数据规模大，并且数据稀疏的问题。在这类问题中，通常使用FM的方法来建模，然后使用分布式学习训练数据。在分布式机器学习中需要把节点分为worker nodes和sever nodes两类分别负责计算和存储全局数据。在每次迭代过程中，worker将自己通过随机梯度下降法（SGD，Stochastic Gradient Descent）的到梯度作为增量去更新server中存储的模型参数，而每次worker的迭代都需要从sever取得上一次更新的模型参数用于更新梯度。可以在这类问题中利用新一代可编程交换机的存储能力缓存每次更新的模型参数，以此将在网计算技术应用扩展到机器学习的领域中。

现阶段，正在将此问题通过mininet与P4语言进行软件仿真来验证其可行性与优势。在仿真中，我们使用p4c作为p4编译器，BMv2作为P4编译配置的软件交换机，PI（P4runtime的实现）实现控制平面对数据平面的控制。

### 4.2 在网计算在分布式机器学习方面的应用研究

我们希望新一代可编程交换机能在其它的应用上也能大放异彩，首先我们想到的是它在机器学习方面的应用。

#### 4.2.1 分布式机器学习及参数服务器

在现实应用中，机器学习训练数据的数量可能达到1TB到1PB之间，而训练过程中的参数可能会达到10的9次方到10的12次方。因此几乎不可能在单台设备中完成对这类大规模机器学习数据的训练。所以我们需要使用多台服务器训练存储参数，这就是分布式机器学习的由来。

李沐提出了使用参数服务器框架的分布式机器学习方法。如图54所示，集群中的节点可以分为计算节点和参数服务节点两种。其中，计算节点负责对分配到自己本地的训练数据（块）计算学习，并更新对应的参数；参数服务节点采用分布式存储的方式，各自存储全局参数的一部分，并作为服务方接受计算节点的参数查询和更新请求。



图54 参数服务器架构

* Worker节点： 每个worker都存储一部分训练数据，并计算本地参数（一般为全局参数的增量）。并push到参数服务器，在新一轮的迭代中从参数服务器中pull更新后的参数。Worker节点之间没有通信，只跟对应的sever通信；
* Task scheduler节点：任务调度节点，将任务分配给worker，并监控他们的进度。如果添加或者删除了worker，它将重新安排未完成的任务；
* Parameter sever：接受来自worker的参数，更新全局参数；
* Server manager：负责维护一些元数据的一致性，比如各个节点的状态，参数的分配情况等。

参数服务器采用key-value的存储方式，用链式复制方式来提高系统容错性（即使用多个参数服务器以链的形式存储相同的数据，来提高容错性，应付可能出现的服务器故障问题）。我们使用参数服务器训练数据，并不是等到所有worker都完成本次迭代再发往参数服务器的方法，这样会造成先完成数据worker需要等待很久。我们使用异步训练的方法，当其中1个worker完成了本次迭代，将数据传给sever并不立即开始新的任务，而是设置一个作为最大延时时间，只有大于之前任务都被完成了，才能开始一个新的任务。这样保证了等待时间不会太长，也保证了每次迭代训练的数据都是有效的。

#### 4.2.2 广告预估模型及因子分解机

在机器学习中，预测是一项基本的任务，所谓的预测就是估计一个函数

 （1）

该函数将一个维的实值特征向量映射到一个目标域中，例如回归问题中：，二分类问题中： 或者。在有监督学习中通常还有一个带标签的训练数据集

 （2）

其中表示输入数据，对应样本的特征向量，对应标签，为样本的数目。

在现实中很多应用问题会应用到预测模型，因为我们关注大规模机器学习，所以选定其训练数据规模较大的广告预估模型，通过个人特征以及浏览历史预估是否会点击当前广告（预测一个人对什么广告感兴趣，做广告推送）。但这类通常会有太过稀疏的问题。我们通过FM来建模。

回顾线性回归模型： 

 （3）

在此模型中，各个特征时独立考虑的并没有考虑特征之间的相互关系，如女生喜欢看化妆品类，而男生喜欢看球类。可以将上述模型推广到二阶：

 （4）

这样将任意两个（互异）特征分量的关系也考虑进来了。对上式来说，只有和都非0式，组合特征才有意义，并且从上面来看组合特征有个，如果特征很多组合特征的量也会特别多，并且非零的项会非常少，导致训练样本的不足，很容易导致不准确，为了克服这个缺陷，改变的表现形式：引入辅助向量

 （5）

并将改写为

 （6）

这样公式（4）就变成了

 （7）

上式还可以改写为：

 （8）

上面实质上的变换，对应与一种矩阵分解，于是最终得到的模型方法就叫做Factorization Machines方法。通过FM不仅解决了数据稀疏的带来学习问题，还降低了模型的复杂度。

#### 4.2.3 随机梯度下降法

通过上面的预估模型，可以得到相应的损失函数。我们使用随机梯度下降法来使得损失函数最小，得到需要估计的参数。

梯度是指函数增加最快的方向，所以求损失函数的最小值，只需要将变量（模型参数）沿着梯度的反方向，一次次迭代，便可以得到损失函数最小值，损失函数最小值时的模型参数便是训练出的最优模型参数。

但是梯度下降法每次更新都需要遍历所有的data，当数据量太大或者以此无法获取全部数据时，这种方法并不可行。解决这个问题的基本思路是：只通过一个随机选取的数据来获取梯度，以此对模型参数进行更新。这种优化方法叫做随机梯度下降法。随机梯度下降法对比梯度下降法来说虽然单次迭代的梯度方向可能会有很大偏差，但足够多次迭代方向的期望与一次遍历所有数据得到的方向是一致的。SGD相对来说要快很多，但是也存在问题，由于单个样本的训练可能会带来很多的噪声，使得SGD并不是每次迭代都向着整体最优方向。因此在刚开始训练时可能收敛的很快，但是训练一段时间后就会变得很慢。再次基础上又提出了小批量随机梯度下降法（我们说的随机梯度下降其实指的是小批量随机梯度下降法），它是每次从样本中随机抽取一小批进行训练，而不是全部数据。这也契合参数服务器框架训练参数的思想。



图55 交换机缓存的参数服务器模型

#### 4.2.4交换机缓存的参数服务器模型

结合之前的参数服务器与当前在网计算技术的发展，我们希望能通过交换机在参数服务器的框架中扮演缓存服务器的角色，将一部分worker pull参数从应用层转到第二层的交换机，以此提高大规模机器学习的效率。并且减轻参数服务器的带宽压力。起到负载均衡的作用。新的框架如图55所示。

在以往的参数服务器框架中新加了switch与controller，switch是用于缓存每一轮新的参数，switch以key-value方式存储latest值，初始值为0，Latest为1表示当前switch的值是最新的，0表示当前switch缓存的参数非最新值。controller用于switch的故障恢复。

新的参数服务器工作逻辑（以随机梯度下降法举例）：

1. 首先Task Scheduler将training data分配给多个worker，worker计算出第一轮梯度，将新得到的梯度push给参数服务器。之后等待之前开始工作的worker工作完成。才可以进行新一轮迭代。
2. 参数服务器得到worker发来的梯度更新模型参数，并存储下来。
3. 当第一个worker可以进行新一轮迭代了，便去pull模型参数。第一个worker经过switch发现switch中latest的值为0，便向参数服务器去pull参数，并在返回时将latest设1，并在switch出缓存一部分参数。其余worker需要pull参数发现latest为1，便可以在switch中pull一部分参数。
4. 当参数服务器发现参数被更新便发送指令把switch中的latest的值设为0。直到新的worker需要pull参数。循环这个过程直至训练结束。

上述过程保证，每一轮最新的参数值只有第一个需要的worker需要从参数服务器pull全部参数。

### 4.3软件仿真

我们现在的工作是通过mininet软件在虚拟机上做一个demo验证我们的想法。软件环境抽象如图56。



图56 仿真环境结构

如图，我们写好xxx.p4代码，通过 p4c 这个 p4 compiler 将p4代码编译成为p4交换机可以理解的各种”机器代码”。如果目标交换机是 bmv2 , 那么p4c将生成 .json文件

* p4c是一款 p4编译器；
* BMv2是支持P4编程的软件交换机；
* PI是P4 runtime的实现，用于Control Plane对数据平面的控制。
* mininet的功能是构建一个虚拟的网络拓扑。 它通过linux内核的一些特性(net命名空间)，在一个主机上划分出多个虚拟网络空间，各个网络空间之间相互隔离，有自己的端口, ip等等。mininet让一个或者多个vhost（虚拟主机）, 软件交换机(如ovs, bmv2)等。以进程的状态分别绑定在这些网络空间之中，共同构成一个进程级别的虚拟网络拓扑。需要注意的是这些进程级别的主机和交换机他们只是网络上的隔离，而文件系统则是共享主机的文件系统。