数据中心网络下基于 SDN 的 TCP 拥塞控制机制研究与实现

陆一飞[1),2)] 朱书宏[1)]

[1)](南京理工大学计算机科学与工程学院 南京 210094)

[2)](东南大学计算机网络和信息集成教育部重点实验室 南京 210096)

摘 要 当前的数据中心中大量存在多个发送端向一个接收端同时发送数据的通信模式,但是,这种多对一的通信模式会造成 TCP incast 问题.当数据中心网络发生 TCP incast 时,网络整体吞吐量将急剧下降甚至崩溃.软件定义网络(SDN)下的集中控制方法和网络全局视角是解决这一问题的有效途径.文中提出一种基于 SDN 的 TCP 拥塞控制机制,称为 TCCS.当 OpenFlow 交换机检测到网络拥塞,将产生拥塞消息并发送至控制器,控制器将通过调整背景数据流 ACK 报文的接收窗口来限制相应数据流的发送速率.利用 SDN 的全局视角,TCCS 可以精确地降低背景数据流的速率来保证突发数据流的性能.TCCS 机制聚焦于网络侧解决 TCP incast 问题,因此对端系统是透明的.最后的实验表明,TCCS 机制能够容纳更多的突发数据流,而且能够保证突发数据流的吞吐量.

关键词 数据中心网络;SDN;TCP;拥塞控制;接收窗口

中图法分类号 TP393 **DOI 号** 10.11897/SP.J.1016.2017.02167

Research and Implementation of TCP Congestion Control Mechanism Based on SDN in Data Center Network

LU Yi-Fei[1),2)] ZHU Shu-Hong[1)]

[1)](*School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing* 210094)

[2)](*Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, Nanjing* 210096)

Abstract TCP incast usually happens when a receiver requests the data from multiple senders simultaneously. This many-to-one communication pattern constantly appears in the data center networks (DCN). However, when TCP incast problem happens in DCN, DCN may suffer from hundreds of milliseconds delay and up to 90% throughput degradation, severely affecting application performance. With Software Defined Networks (SDN) as a new paradigm for networking, the centralized control methods and the global view of the network can be an effective way to handle the TCP incast problem. In this paper, we propose a TCP congestion control mechanism based on SDN, referred to as TCCS, to solve the TCP incast problem. TCCS leverages the features of SDN to accurately assign sending rate for background flows and burst flows so as to accommodate more burst flows and improve the overall network performance and utilization. In particular, TCCS contains four modules including network congestion trigger module, elephant flows selection module, receive window estimation module, and receive window regulation module. We first design network congestion trigger module over an OpenFlow enabled switch. Once network congestion is discovered by assessing queue length, the congested switch will trigger a congestion notification message to our SDN controller. Subsequently, elephant flow selection module exploits

收稿日期:2016-04-14;在线出版日期:2017-06-23.本课题得到江苏省博士后基金、国家自然科学基金(61602243)资助.**陆一飞**,男,1982 年生,博士,讲师,中国计算机学会(CCF)会员,主要研究方向为数据中心网络传输协议、软件定义网络、存储网络.E-mail:luyifei@njust.edu.cn.**朱书宏**,男,1991 年生,硕士,主要研究方向为软件定义网络和流量工程.

the controller to differentiate the background flows from burst flows according to different flow traffic characteristics in DCN. After that, receive window estimation module at the controller side estimates the current bandwidth of these chosen background flows and then degrades their bandwidth to the desired one. We assess our desired bandwidth in terms of the network congestion level. Then, our controller generates a notification message containing new flow table entries that is used to regulate the background flow bandwidth to our desired one and sends them to the switch. Upon receiving the notification, the congested switch can deliberately manipulate the advertised window of TCP ACK packets of the corresponding background flows so as to effectively degrade the bandwidth of the background flows. TCCS is a centralized approach that does not revise the legacy TCP stack. Thus, it is transparent to the end systems and easy to deploy. By carefully deploying the location of our controller, our extensive analysis results show that the control delay between the switch and controller can be virtually ignored. Besides, the fairness of TCCS is discussed in our paper. We implement TCCS by revising the OpenFlow protocol. During TCP connection establishment, we can generate a Global-View flow Table (GVT) that includes all of the TCP flows information in the network. GVT consists mainly of BackGround flows Table (BGT). We also extend the standard OpenFlow protocol to support the congestion notification message, TCP ACK flag match function and ACK advertised window adjust action. Our experimental results show that TCCS is effective in improving the performance of burst flows, e.g., reducing transmission time, increasing the number of concurrent flows, and providing high tolerance for burst flows while guaranteeing the bandwidth of background flows. Therefore, TCCS is an effective and scalable solution for TCP incast problem.

Keywords data center network; software defined networks; transmission control protocol; congestion control; receive window

1 引 言

云计算的蓬勃发展得到了产业界和学术界的广泛关注,已经成为信息领域的热点和未来发展趋势,其中,数据中心网络成为了实现云计算、云存储的最重要核心基础设施.已有的研究指出[1,2],TCP 协议是数据中心网络传输的基础,TCP 流量占数据中心内部流量的主导地位.TCP 协议的拥塞控制策略与方法直接影响数据传输的效率.但是传统 TCP 协议是基于互联网环境设计的,在数据中心网络下遇到了很大的挑战.数据中心网络中除了背景数据流(称为大象流)外,还大量存在一对多和多对多的网络通信流量,多条数据流竞争同一交换机的出口缓冲队列使得缓冲区溢出,产生丢包,导致吞吐量的急剧下降,这种现象称为 TCP incast(也称为 TCP 吞吐量崩溃),如图 1 所示.数据中心的突发数据流(称为老鼠流)是造成 TCP incast 的最主要原因[3,4].TCP incast 作为数据中心网络实际存在的严重问题,引起了越来越多的关注.本文在数据中心网络场景下,针对 TCP incast 问题,设计新的 TCP 拥塞控制协议.

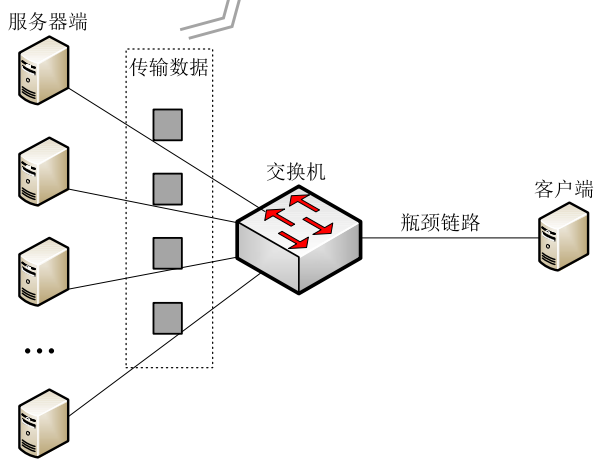


图 1 TCP incast 场景

当前,针对 TCP incast 问题已有不少研究.这些研究的主要思路是通过一定的方法了解网络拥塞程度,并通告发送端,以此调整发送速率,缓解或解决网络拥塞.但是这些方法要么存在拥塞程度估计不准确,如显示拥塞通告(ECN)技术,要么技术方

案部署困难，如需要修改交换机、发送端和接收端三者协议栈. 而且，当前的解决方法没有考虑数据中心网络数据流本身的特征，如没有区别对待老鼠流和大象流. 综上所述，必须引入新的研究思路和技术手段来解决数据中心网络拥塞和 TCP incast 问题.

近年来出现的 SDN 思想，特别是 OpenFlow 协议[5]的提出，为数据中心网络的拥塞控制研究提供了全新的解决思路. SDN/OpenFlow 技术可以收集网络信息，包括网络流量信息和交换机队列信息等，通过这些信息能够更加精确、更加快速的进行拥塞判断和拥塞调整. 本文基于 SDN 思想，提出了一种基于 SDN 的 TCP 拥塞控制机制(TCP Congestion Control based on SDN，TCCS). TCCS 机制的核心思想是当网络拥塞时，通过临时降低大象流的发送速率，以满足老鼠流的数据传输，避免 TCP incast 的发生. TCCS 机制通过 OpenFlow 技术收集网络数据流信息和 OpenFlow 交换机(下文简称为交换机)队列信息，当交换机队列长度超过阈值时，触发拥塞通告信息至 SDN 控制器(简称为控制器)，随后控制器从全局 TCP 数据流表中选出大象流，根据大象流应有的传输速率，估算大象流 ACK 报文的接收窗口值，并通过控制器下发修改 ACK 接收窗口的流表. 交换机根据该流表自动匹配大象流的 ACK 报文并修改相应 ACK 报文的接收窗口，降低大象流传输速率，解决网络拥塞.

2　相关研究

TCP incast 作为数据中心网络实际存在的一个严重问题，引起了越来越多的关注，迄今为止，研究者已经提出了很多解决 TCP incast 问题的方案. 总的来说，根据解决手段的不同，可以分为以下 4 部分：基于窗口的解决方案、基于恢复的解决方案、基于应用的解决方案和基于 SDN 的解决方案.

(1) 基于窗口的解决方案. 基于窗口的解决方案的核心思想是通过修改 TCP 协议的拥塞窗口或接收窗口，使发送端调整 TCP 的发送窗口(也即发送速率)，减少网络中的数据报文，最终减少交换机的队列长度，解决 TCP incast 问题. DCTCP 协议[1]是第一个专门针对数据中心网络的拥塞控制机制. 该协议在交换机队列长度超过阈值时，使用 ECN 标记超过阈值的报文，发送端根据被标记数据包的比例调整拥塞窗口大小，被标记的比例越大，降低的幅度越大. 随后的 D^{2}TCP 协议[6]基于截止期限和拥塞程度来调节发送端发送窗口，远期截止期限数据流降速多，近期截止期限数据流降速少. D^{2}TCP 具有较好的稳定性和收敛性，但是 D^{2}TCP 协议不能根据全局网络信息来调整发送窗口，具有较大的优化空间. 文献[7-10]通过网络延迟的方法来表示网络拥塞的程度，以此调整发送窗口大小，这种方法的问题在于网络延迟或者 RTT 时间的计算精确度不高，或者算法复杂度较大. PAC 协议[11]在接收端判断网络拥塞程度，然后通过回复 ACK 报文的间隔来控制发送端的发送速率. 修改 TCP 接收窗口是解决网络拥塞问题的另一种思路. ICTCP 协议[12]提出通过修改 TCP 接收窗口来解决 TCP incast 的方法. ICTCP 根据接收端的 TCP 连接情况，判断拥塞程度，主动调整每一个 TCP 连接的接收窗口大小，从而避免 TCP incast 的发生. SAB[13]协议利用交换机统计流经的数据流数量，并将端口缓存平均分配给每一条数据流，以此解决 TCP incast 问题. 但是 SAB 协议也存在一些不足：① 需要修改每一个 ACK 报文头部的接收窗口，开销较大；② 由于 SAB 对流数量的统计不准确，需要保留一部分缓存空间(一半缓存空间)给未被统计的流；③ SAB 需要修改交换机、发送端和接收端的协议栈.

(2) 基于恢复的解决方案. 基于恢复的解决方案主要在拥塞发生时能快速的恢复至正常状态，其主要的解决手段有降低重传定时器的时间和快速重传等. 在文献[14,15]中提出了多种拥塞快速恢复的算法，如将 TCP 的 RTO_{min} 降低到 1 ms 以此来降低由于 TCP incast 问题导致的超时影响；将 TCP 快速重传中的重复 ACK 数从 3 降为 1，使得发送端能够快速重传；关闭 TCP 的慢启动过程，使得发送端的速率更快增长；通过不同的 TCP 算法来解决 TCP incast 问题. 以上的方案较为简单，仅需要修改当前 TCP 协议栈的某些参数设置或者增加高分辨率的计时器，但是这些方案并没有阻止交换机队列中数据的累积，也没有解决传输延迟和 TCP incast 问题. 国内的清华大学在这方面进行了深入的研究. 文献[16]提出了基于准确丢包通知的 CP 协议，该协议在交换机过载时，会通过一种类似 TCP SACK 的方法精确地告知发送端丢失的报文，发送端根据该报文重传丢失报文和调整发送速率. CP 协议能够快速地重传丢失报文，不会造成重传超时，但是 CP 引入了新的报文格式，需要端系统和交换机的支持，协议的复杂度和开销也比较高. 文献[17]针对数据中心的两种超时(满窗口报文丢失超时和缺 ACK

超时),提出了 GIP 协议,该协议能感知发送窗口的边界并使用该边界信息调整报文发送机制,以此避免以上的两种超时,解决 TCP incast 问题.

(3)基于应用的解决方案. 基于应用的解决方案核心思想是通过应用层的方法降低数据包的丢失,如降低并发传输数据的端系统数量. 文献[14,18]提出了几种有效解决 TCP incast 问题的应用层方案,包括增加 SRU(Server Request Unit)大小、延时数据传输、全局规划数据传输和限制并发通信服务器的个数等. 文献[19]在应用层上构造一个传输分组模型,每一组通过延时一个确定时间 T_{deter} 或者随机时间 T_{rand} 分批发送数据块的请求,这样就使得数据传输具有一定的间隔,避免 TCP incast 问题的产生. 文献[20]中假定了应用可以知道发送端信息(如发送者的数量),提出从应用业务角度限制发送端并发流数量和数据大小,以及精确的调度策略,来解决 TCP incast 问题. 文献[21,22]从应用层的角度出发,设计了数据流的调度策略,使服务器响应请求时避免同步执行,以此解决 TCP incast 问题. 这种策略能够避免数据包的丢失,而且不需要修改 TCP 协议和网络交换机.

(4)基于中心控制器的解决方案. 近年来,基于 SDN 技术的 TCP 拥塞控制研究也得到广泛的关注. 文献[23]第一次提出了基于 SDN 思想的 OpenTCP 架构,该架构利用全局的网络视角和流量情况,使系统能够自适应的动态调整 TCP 参数甚至切换不同的 TCP 协议,更快更精确的进行拥塞控制. OpenTCP 并不是一个新的 TCP 协议,而是一种开放架构,该架构可以与传统的 TCP 协议兼容. 文献[24]提出了通过 SDN 技术的全局网络能力,设置新建 TCP 数据流的初始窗口和重传超时定时器. 这种方法能够减少老鼠流的传输时间. 文献[25]提出了基于 SDN 的速率控制机制,该机制通过发送端与网络侧交互信息,获得可分配速率和已占用速率,使得发送端可以精确的控制发送速率,避免网络拥塞的发生. OTCP 协议[26]在控制器端通过全局网络信息计算 TCP 协议的参数,并把参数传递给相应的 TCP 协议栈,使得当前的 TCP 协议能获得最佳的传输速率. 以上的方案主要是通过 SDN 的全局视角协商 TCP 协议的最优参数,需要修改 TCP 协议栈和交换机端代码,而 TCCS 机制基于控制器的全局化技术降低大象流速率以此满足老鼠流的传输需求,缓解 TCP incast,并且 TCCS 只需要简单的修改交换机端,不需要修改 TCP 协议栈.

文献[27]文中提出一种用于 OpenFlow 网络的快速流调度策略(Nimble). Nimble 由网络设备自主监测设备状态,并在网络出现拥塞时通过扩展的 OpenFlow 消息主动向控制器通告拥塞信息. 该策略能够以近于零的时延检测网络链路拥塞,从而有效提高网络性能. Nimble 在结构化拓扑下能得到较好的性能,但是非结构化拓扑下计算新路径的时间开销较大,网络性能下降. 另外,Fastpass[28]也是一种基于全局的拥塞解决方案,Fastpass 通过全局仲裁者为发送端-接收端分配不同的时间槽和选择不同路由,以此解决网络拥塞问题. Fastpass 需要端系统的硬件支持,特别是网卡的支持,同时需要修改终端用户的协议栈,最后 Fastpass 是一种新的网络架构,并不是专门针对 TCP 协议进行优化.

文献[29]提出了 IQM 算法,该算法在队列长度超过阈值时,通过 SDN 控制器将数据流 ACK 报文的接收窗口设置为 1*MSS*,以此解决 TCP incast 问题. IQM 算法需要修改每一个 ACK 报文,消耗较大的计算资源,同时,IQM 算法直接将发送窗口降至 1*MSS*,会降低网络的吞吐量,特别是在数据流数量较小时. 文献[30]通过软件交换机为不同流设置不同的 DSCP 值,以此来标识大象流;同时,当拥塞发生时,中心控制器不对大象流做 CE 标记,使得大象流不受拥塞控制的影响. 这种方法试图不调整大象流的速率,但是这种保证大象流的方法会影响老鼠流的吞吐量,另外,该文献并没有自动识别大象流,而是通过约定的端口号和 TCP 协议判别,不利于实际部署. 本文提出的 TCCS 机制与以上两种技术有较大的区别,TCCS 会自动的识别大象流和老鼠流,并根据拥塞的程度调整大象流的发送速率,以此缓解网络拥塞.

从上面 4 个方面的 TCP 拥塞控制方案可以看出,这些方案要么需要修改 TCP 协议,难以兼容现有 TCP 协议;要么需要实际应用的支持,实用性很差;要么通过 ECN 标记、RTT 或者重复 ACK 判断拥塞程度,这些方法估计拥塞程度不准确. 利用 SDN 技术可以从网络侧直接获取交换机当前队列长度信息,可以较为准确的表示拥塞程度,并以此解决 TCP incast 问题.

3 TCCS 机制

3.1 基本思想

TCCS 机制的基本思想为:控制器对所有数据

流信息进行记录，当网络发生拥塞时，控制器计算大象流的发送速率并下发流表至交换机，使交换机修改大象流 ACK 报文中接收窗口字段，降低大象流的传输速率，保证老鼠流的传输速率，达到拥塞控制的目的. TCCS 整体架构如图 2 所示.

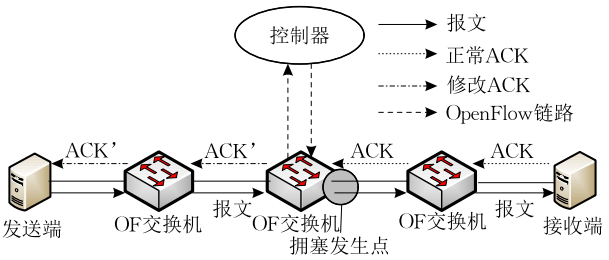


图 2　TCCS 总体架构

TCCS 机制整体工作流程如图 3 所示，包括四个模块：拥塞触发模块、大象流选择模块、接收窗口估算模块和接收窗口调整模块. 拥塞触发模块实时监视交换机端口的队列长度，当队列长度超过阈值时，交换机向控制器发送拥塞通告消息. 当控制器收到拥塞通告消息后，控制器从全局 TCP 数据流表(Global-View flow Table，GVT)中选出大象流，并根据拥塞程度估算大象流 ACK 报文的接收窗口大小，最终下发修改 ACK 接收窗口的流表项(简称为修改窗口流表)到发生拥塞的交换机. 当进入交换机的报文匹配修改窗口流表时，将会修改大象流 ACK 报文的接收窗口.

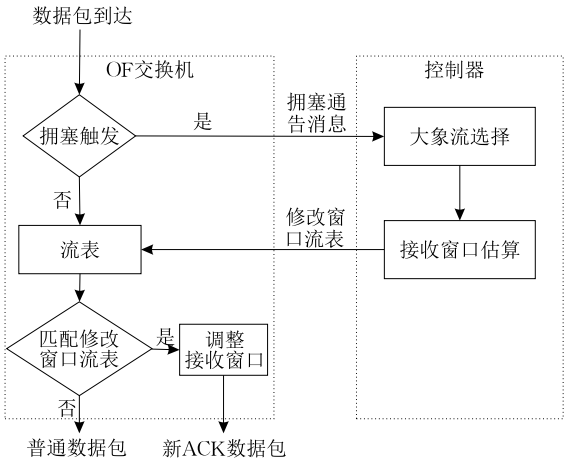


图 3　TCCS 工作流程

TCCS 机制不修改终端侧(发送端和接收端)的 TCP 协议栈，因此，TCCS 机制可以兼容现有的 TCP 协议，并且部署灵活方便. TCP 发送端的发送窗口 W_S 还是由拥塞窗口 W_C 和接收到的 ACK 报文中的接收窗口 W_R 的最小值决定，也即满足：

$$W_S=\min(W_C,W_R) \tag{1}$$

3.2　拥塞触发

拥塞触发模块通过尾丢弃队列管理方式实时监控队列长度，当交换机中的某个端口的队列长度超过了阈值 L 时，产生拥塞通告信息(Congestion Notification Message，CNM)，并将该消息传送到控制器，预警拥塞发生. CNM 是扩展的 OpenFlow 协议消息，包含交换机的标识、发生拥塞的端口号和队列长度. 该交换机进入拥塞状态，并周期性的监视队列长度. 如果之后每个周期监视队列的长度仍超过阈值 L，则持续向控制器发送 CNM. 当连续三个周期监视到的队列长度都小于阈值 L 时，交换机向控制器发送拥塞恢复信息(Congestion Recovery Message，CRM)，交换机恢复到正常状态. 我们设置周期时间为 T，该 T 值为流经交换机的所有数据流 RTT 的平均值. 详细的拥塞触发的算法如算法 1 所示.

算法 1.　拥塞触发算法.

输入：报文 P

输出：无

```
1.  state=0, count=0
2.  CurrentLength=CurrentLength+size(P);
3.  IF (CurrentLength>=L)
4.   IF (state==0)  //in the normal state
5.    //a first congestion notification message is generated
6.    Message=GenerateOFMessage(CNM);
7.    sendMessage(Message);    //send message to
                                controller
8.    state=1;      //enter into congestion state
9.    //generate TIMER
10.   START COUNTDOWN TIMER;
11.  ELSE
12.   IF (TIMER EXPIRED)
13.    //if queue length is bigger than L in every period,
14.    //then generate congestion notification message
15.    Message=GenerateOFMessage(CNM);
16.    sendMessage(Message);
17.    START COUNTDOWN TIMER;
18.   END IF
19.   count=0;
20.  END IF
21. ELSE //CurrentLength<L
22.  IF (state==1&&TIMER EXPIRED)
23.   count++;
24.   START COUNTDOWN TIMER;
25.  END IF
26.  //if queue length is smaller than L for three periods,
27.  //then generate congestion recover message
```

```
28. IF (count==3)
29.   Message=GenerateOFMessage(CRM);
30.   sendMessage(Message);
31.   STOP TIMER;
32.   state=0;  //recover to normal state
33. END IF
34. END IF
```

3.3 大象流选择

在 SDN 网络架构下,网络中所有数据流的建立都需要通过控制器,因此,所有 TCP 数据流信息会被记录在控制器的全局数据流中(GVT),根据数据中心流量特征,我们可以将 GVT 的数据流分成大象流和老鼠流,生成相应的大象流表(BGT). GVT 和 BGT 的生成过程将在 4.1 节中介绍.

当控制器收到拥塞通告信息时,从中提取队列长度(Q_i),发生拥塞的交换机(S_i)和端口(P_i). 随后,控制器根据大象流选择算法(算法 2)从 BGT 中选出经过交换机 S_i 的端口 P_i 的数据流.

算法 2. 大象流选择算法.

输入:交换机 S_i,端口 P_i,大象流表 BGT

输出:匹配流 *MatchFlows*

```
1. FOR EACH flow f_i∈BGT
2.   IF f_i.In_Port==P_i && f_i.SwitchID==S_i
3.     MatchFlows.add(f_i); //f_i add to the MatchFlows
4.   END IF
5. END FOR
6. RETURN MatchFlows
```

3.4 大象流速率调整

选出大象流后,控制器需要限制大象流的发送速率来保证老鼠流的发送速率. 在此,我们通过调整大象流 ACK 报文头部的接收窗口降低其发送速率. 大象流速率调整模块包含两个功能:(1)估算当前大象流发送窗口;(2)根据拥塞程度,计算大象流 ACK 报文的接收窗口.

3.4.1 大象流发送窗口估算

如图 1 所示,有 n 条数据流经过交换机的某端口,假设其中大象流集合为 $\boldsymbol{A}$,集合大小为 a,老鼠流集合为 $\boldsymbol{B}$,集合大小为 $b(n=a+b)$,则在该端口以最大速度转发数据报文时,满足如下式(2):

$$\sum_{i\in \boldsymbol{A}\cup \boldsymbol{B}} W_i(t) = C\times RTT_{\text{avg}} + Q(t) \tag{2}$$

其中,$W_i(t)$ 是 t 时刻数据流 i 的发送窗口大小,C 为端口的最大传输速率,RTT_{avg} 为 n 条数据流的平均往返时延,$Q(t)$ 为 t 时刻端口的队列长度.

根据传统 TCP 协议的公平性原则,在稳定状态下,每条数据流发送窗口的平均值为

$$W_i = \frac{C\times RTT_{\text{avg}} + Q(t)}{n},\ i\in \boldsymbol{A}\cup \boldsymbol{B} \tag{3}$$

当网络拥塞刚发生,也即收到 CNM 报文时,我们可以从 CNM 报文中提取当前拥塞队列长度 $Q(t)$. 数据流数量 n 可以通过 GVT 表获得. 平均 RTT 值可以通过式(4)获得.

$$RTT_{\text{avg}} = RTT_e + Q(t)/C \tag{4}$$

其中,$mRTT_e$ 表示网络非拥塞状态下的平均 RTT 时间,该时间是一种预先测量值,可以通过网络部署时获得. $Q(t)/C$ 表示报文在队列中的排队时间,其中队列长 $Q(t)$ 从 CNM 报文中获取.

因此,根据式(3),可以计算得出 W_i,也即是大象流的发送窗口值,在此记为 $SWND_A$.

3.4.2 ACK 接收窗口计算

当控制器估算出大象流的发送窗口值 $SWND_A$ 后,可以根据网络拥塞程度,调整大象流的 ACK 报文的接收窗口大小. TCCS 机制使用拥塞队列长度($Q(t)$)来表征网络拥塞的程度. 我们设置两个阈值将网络拥塞程度分为三种情况,这两个阈值分别记为 $Low\_threshold$(记作 Q_L)和 $High\_threshold$(记作 Q_H),满足 $L<Q_L<Q_H$.

(1) $L<=Q(t)<Q_L$:表明当前已有轻微拥塞,但并不严重,只需较小的限制大象流发送速率. 此时,限制 ACK 报文的接收窗口大小为

$$W_r(i) = \max\left(\frac{2}{3}SWND_A, 1MSS\right),\ i\in \boldsymbol{A} \tag{5}$$

(2) $Q_L<=Q(t)<Q_H$:表明拥塞持续产生,并且超过 $Low\_threshold$ 阈值,需要更多的降低大象流的发送速率,因此大象流的 ACK 报文的接收窗口值为

$$W_r(i) = \max\left(\frac{1}{2}SWND_A, 1MSS\right),\ i\in \boldsymbol{A} \tag{6}$$

(3) $Q_H<=Q(t)$:这种情况下,队列长度超过 $High\_threshold$ 阈值,需要采取较为极端的措施来避免拥塞,此时大象流的 ACK 报文的接收窗口值为

$$W_r(i) = 1MSS,\ i\in \boldsymbol{A} \tag{7}$$

这种情况下,我们也限制老鼠流的发送速率,此时,老鼠流的 ACK 报文的接收窗口值设置为

$$W_r(i) = \max\left(\frac{n\times SWND_A - a\times 1MSS}{b},\ 1MSS\right),\ i\in \boldsymbol{B} \tag{8}$$

在计算出大象流(或老鼠流)ACK 报文的接收窗口后,控制器会生成修改大象流 ACK 报文接收窗口的流表,并且将接收窗口值($W_r(i)$)附在流表报文的数据部分. 该流表头部匹配 ACK 报文,动作

为修改 ACK 的接收窗口，被称为修改窗口流表.

3.5　接收窗口调整

交换机收到修改窗口流表后，当收到匹配的 ACK 报文后，将修改 ACK 报文头部的接收窗口字段为

$$RWND(i)'=\min(W_r(i),RWND(i)) \quad (8)$$

其中，$RWND(i)$为数据流 i(主要是大象流)的 ACK 报文中原有的接收窗口字段的大小.

3.6　TCCS 分析与讨论

通过上述的描述，我们了解了 TCCS 机制的基本原理和详细的工作流程，但是 TCCS 机制还存在一些问题需要进一步探讨.

(1)公平性问题. TCCS 机制的核心思想是降低大象流所占带宽，给予老鼠流更多的带宽，以此容纳更多的老鼠流，解决 TCP incast 问题. 因此 TCCS 机制会造成大象流和老鼠流间的不公平，但是从 TCCS 机制中我们也可以看到，大象流之间和老鼠流之间是公平的. 另外，已有的很多研究[6]都采用类似的方法，给予某些流更多的带宽来解决 TCP incast 问题.

(2)控制延迟. TCCS 机制检测到交换机队列长度大于阈值 L 时，触发拥塞通告消息，控制器收到该消息后，选择大象流，计算其发送窗口和相应 ACK 报文的接收窗口，下发修改窗口流表至交换机，这个过程消耗的时间称为控制延迟. 因此，控制延迟(T_{cd})由交换机和控制器间的往返传输时间(T_{oc})和控制器处理时间(T_p)组成，即 $T_{cd}=T_{oc}+T_p$. 控制器处理时间又由大象流选择和大象流速率调整两个模块的处理时间组成. 这两个模块的时间复杂度与大象流数量相关，它们的时间复杂度都为 $O(a)$(a 是大象流的数量). 由于数据中心网络内，发生拥塞的节点处大象流数量较小，其数量有 90% 的概率小于 4，因此，控制器处理时间的时间复杂度为为 $O(1)$，可忽略不计. 此时，控制延迟可近似等于控制器与交换机间的往返时延，即 $T_{cd}\approx T_{oc}$. 而 T_{oc} 又跟控制器和交换机的距离有关，当控制器和交换机直接相连或者只有 1 跳距离时，T_{oc} 的时间也可以忽略. 综上所述，当控制器与交换机间的距离很短，TCCS 的控制延迟时间很小，可以忽略.

(3)无大象流场景. 当网络发生拥塞时，同样也会触发拥塞通告消息，但是由于此时网络中没有大象流，因此只有在队列长度大于 $High\_threshold$ 时，才会限制老鼠流的发送速率.

4　TCCS 机制实现

4.1　全局 TCP 数据流表生成

在 TCP 协议中，客户端和服务器通过三次握手建立 TCP 连接以及四次挥手来断开连接. 因此，通过在交换机中添加特殊流表项来匹配带有 SYN 以及 FIN 字段的数据包，并将这些数据包上交控制器，控制器可以记录所有 TCP 数据流信息并生成全局数据流表(GVT).

根据图 4 所示，建立 TCP 连接的第一握手时，客户端发出 SYN 报文，交换机中并没有该数据流的流表项，所以交换机通过 OpenFlow 协议发送包含 SYN 报文的 OF_PacketIn 数据包到控制器. 控制器将该数据流信息增加到 GVT 中.

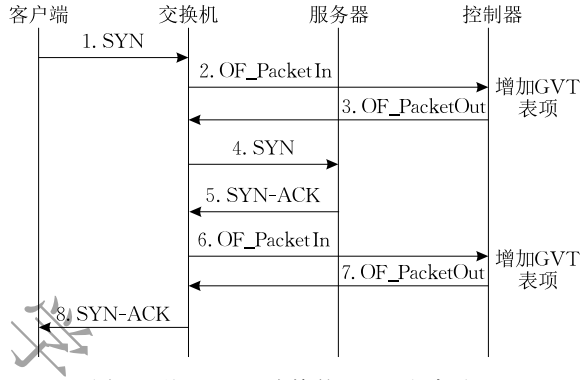


图 4　基于 TCP 连接的 GVT 生成过程

GVT 结构如图 5 所示，其中 FLOW_ID 用于标记一条数据流，SRC_IP 和 DES_IP 分别是该数据流的源和目的 IP 地址，LifeTime 表示该数据流的生存时间，Size 表示该数据流的发送数据报文大小，Flow_Path 记录着该数据流的路由信息，该路由信息记录着所有途径交换机上的出入口. 同样的，接收端回复的 SYN-ACK 消息也会被交换机上送至控制器进行记录. 生成 GVT 后，我们会对 GVT 中的数据流进行分类，将生存时间大于 1 s 且数据流大小超过 1 MB 的数据流加入大象流表(BGT).

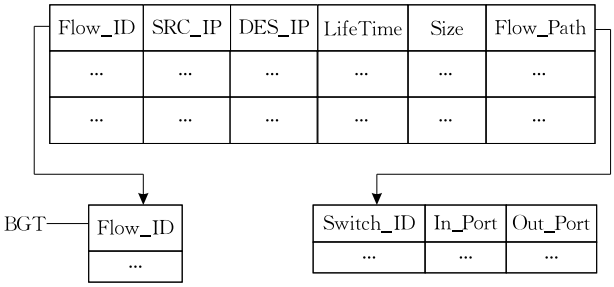


图 5　全局数据流表

与 TCP 的三次握手相似，当 TCP 终止连接时的四次挥手也会通过交换机上交信息至控制器，从而更新 GVT 表项. TCP 数据流删除该过程如图 6 所示.

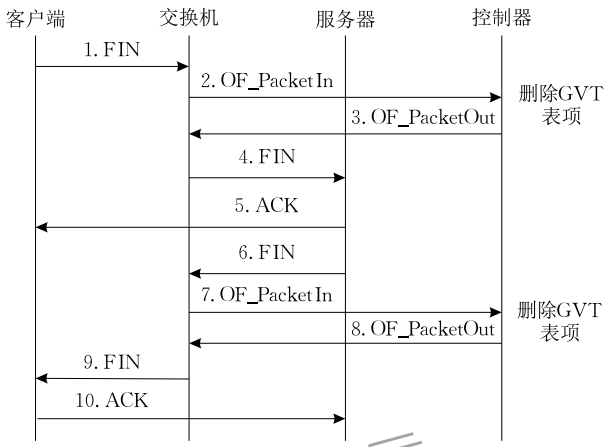


图 6 TCP 数据流删除流程图

4.2 OpenFlow 协议扩展

为了实现 TCCS 机制，需要扩展 OpenFlow 1.3 版本协议，以支持新的拥塞控制信息、新的匹配域 TCP_FLAGS 和窗口修改动作.

4.2.1 拥塞控制信息

在拥塞触发模块中，交换机产生两种消息来通知端口的状态. 通过扩展 OpenFlow 1.3 协议，在端口队列长度超过阈值 L 之后，产生拥塞触发消息 CNM；在拥塞消除后，产生拥塞恢复消息 CRM. CNM 报文格式如图 7，它包含 OpenFlow 标准报文头 ofp_header，拥有所有端口信息的 ofp_port，表明端口队列长度的 port_buf，优先级和 cookie 字段. 报文头部 ofp_header 中的 type 字段设为 OFPT_BUF_CN，表示拥塞触发消息，类似的，type 字段为 OFPT_BUF_CR 时，表示拥塞恢复消息.

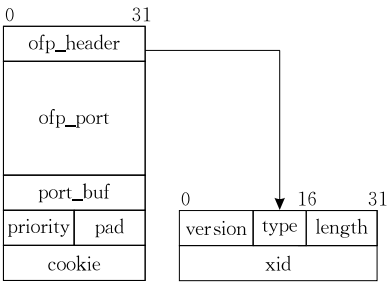


图 7 OFPT_BUF_CN 格式

4.2.2 TCP_FLAGS 匹配

TCCS 机制需要流表匹配 TCP 的 ACK 报文，SYN 报文和 FIN 报文，但是在 OpenFlow1.3 版本中，无法准确匹配这些 TCP_FLAGS 字段，也即无法自动匹配 ACK 报文. 虽然在 OpenFlow1.5 版本协议中，增加了 TCP_FLAGS 匹配字段，然而目前并没有支持 OpenFlow1.5 协议的交换机和控制器.

在 Open vSwitch 2.3.0 中，TCP_FLAGS 字段作为 NXM(Nicira extensible match，Nicira 私有扩展匹配域)被添加至交换机中，所以控制器只需要根据 NXM 要求的特定 OXM(OpenFlow Extensible Match)匹配报文格式来对 TCP_FLAGS 字段进行匹配. OXM 头格式如图 8 所示. OXM_CLASS 字段固定为 0x0001，OXM_FIELD 字段值为 34，HM 字段为表示 OXM 格式是否带掩码，如果该匹配域字段带掩码，则值为 1，反之为 0，OXM_LENGTH 为整个 OXM 结构体的长度.

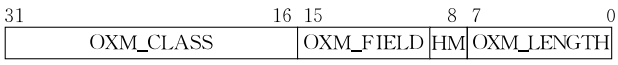


图 8 OXM 头格式

4.2.3 接收窗口修改动作

在 OpenFlow1.3 版本协议中，拥有一个动作集合 OFPT_SET_FIELD，该集合包含的动作能完成修改源和目的 IP、源和目的端口、VLAN 标签等一系列目标. 通过扩展该集合动作，建立修改接收窗口(MOD_WINDOW)新动作. 在数据报文被匹配后，MOD_WINDOW 动作会先检查流表项中要求修改的接收窗口大小是否小于原数据包的接收窗口. 若小，则修改，否则不进行操作.

5 实验和结果

该部分我们首先介绍实验设置，随后通过简单网络和复杂网络分别测试 TCCS 的性能，并与 TCP NewReno 和 DCTCP 的性能进行比较，最后对 TCCS 的控制器进行压力测试.

5.1 实验设置

由于缺少物理 OpenFlow 交换机，而且需要对交换机协议进行扩展，以支持 TCCS 机制，因此，本文使用 Mininet 工具建立网络拓扑并对 TCCS 机制评估. Mininet 中的软件交换机为 Open vSwitch v2.3.0(OVS)，OVS 支持 OpenFlow 协议. 控制器采用 Floodlight.

部署该实验平台的实验机为台式机，配置 2.4GHz、Intel i7-4700 四核八线程 CPU，8GB 内存，1T 硬盘，操作系统为 Ubuntu 14.04.2(内核为 3.16.0-30-generic). 部署控制器的实验机为笔记本，配置 1.9GHz 和 Intel I5 四核八线程 CPU，4GB

内存，500 G 硬盘，操作系统为 Ubuntu 14.04.2(内核为 3.16.0-30-generic).

实验中交换机的端口队列缓存大小一致，且队列缓存大小设为 150 个报文，也即 225 KB(150×1.5 KB). TCCS 机制中的队列阈值 L，*Low_threshold* 和 *High_threshold* 分别设为 40，70，135 个报文长度. 为了与 TCCS 机制做比较，我们也在 Mininet 上部署 TCP NewReno 协议(为了书写方便，后面实验都以 TCP 代称)和 DCTCP 协议. 实验中的最大数据大小(MSS)为 1460 B，TCP 的最小重传超时时间(RTO_{min})为 30 ms. 本实验中的结果都是重复 10 次后的平均值.

5.2　简单网络实验

该部分实验采用的拓扑结构如图 9 所示，包括 n 个发送端和 2 个接收端，两台交换机，Floodlight 控制器. 交换机与控制器直接相连. 所有链路的带宽为 1 Gbps，传输延迟为 50 μs，因此发送端与接收端之间的 RTT 为 300 μs.

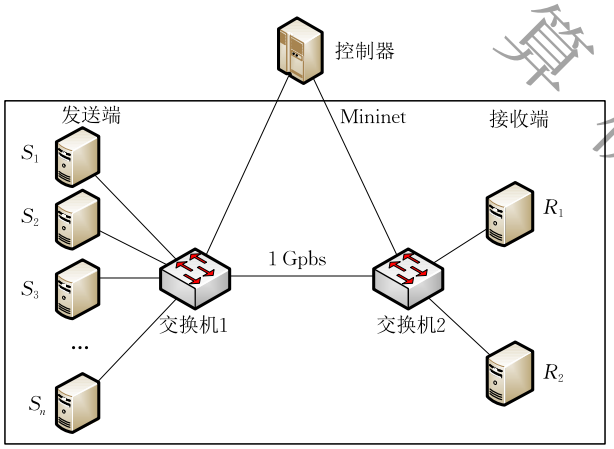


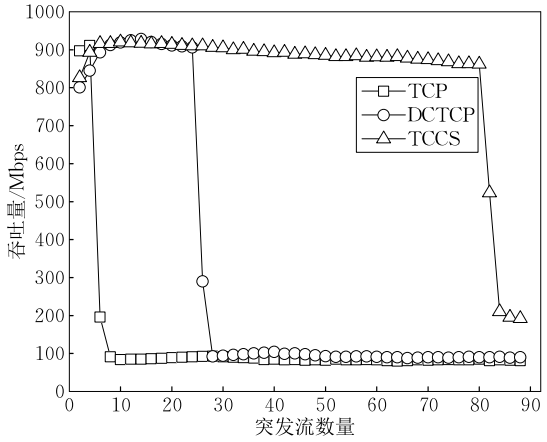
图 9　简单网络拓扑结构

(1) 网络性能实验

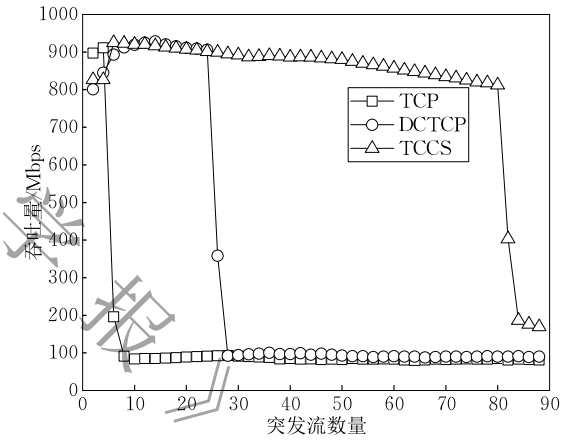
该部分实验目的为测试 TCP、DCTCP 和 TCCS 机制在老鼠流和大象流并存场景下，网络吞吐量和丢包率的变化情况.

首先，我们分析不同老鼠流数量情况下，网络吞吐量变化情况. 该实验共有 90 条数据流，其中 88 条老鼠流和 2 条大象流. 大象流在实验中持续发送数据. 老鼠流分为两种流量特征：① 固定每条老鼠流大小为 120 KB，此时老鼠流数量越大，老鼠流总量越大；② 固定老鼠流总量大小为 20 MB，此时老鼠流数量越大，每条数据流大小越小.

图 10 为 TCCS 与 TCP 和 DCTCP 在不同老鼠流数量(从 1 到 88)情况下吞吐量的变化情况. 可以看出，无论在固定每条老鼠流大小还是在固定老鼠流总量大小的情况下，TCP 都在接近 6 条老鼠流时吞吐量下降，而 DCTCP 在大约 26 条老鼠流时吞吐量下降. 在老鼠流数量增加至 82 时，TCCS 机制才出现了吞吐量明显下降，其它时候 TCCS 机制在保持了 90%左右的吞吐量，这说明 TCCS 可以很好的处理 80 条以内的老鼠流.



(a) 固定每条老鼠流大小

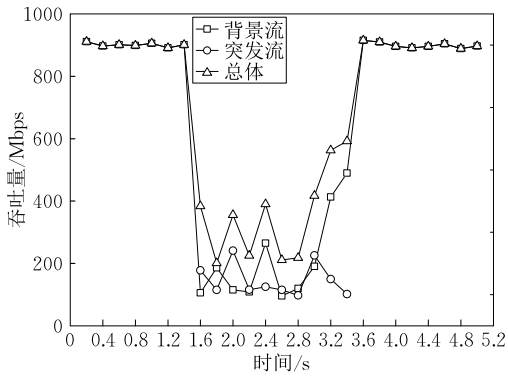


(b) 固定老鼠流总量大小

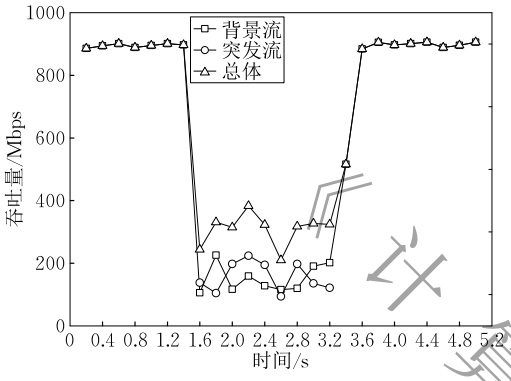
图 10　吞吐量随并发数据流数量变化情况

随后，我们观察连续时间下，老鼠流、大象流和网络总体吞吐量的变化情况. 实验中有 25 条老鼠流和 2 条大象流，其中老鼠流一共传输了 30 MB 的数据. 实验结果如图 11 所示.

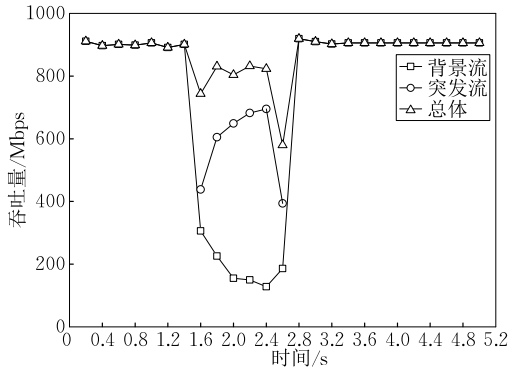
从图 11(a)和(b)可以看出，在老鼠流出现之前，TCP 和 DCTCP 中的大象流的吞吐量大概为 910 Mbps. 当老鼠流到达时，TCP 和 DCTCP 的吞吐量急剧下降，这是由于多条数据流争抢瓶颈线路，造成大量的 TCP 超时重传，形成 TCP incast 问题，不仅大象流发生了吞吐量的急剧降低，老鼠流的吞吐量也保持在 150 Mbps 左右，总吞吐量在 200～400 Mbps 之间震荡. 图 11(c)中为 TCCS 的吞吐量情况，在老鼠流到达网络后，控制器通过修改接收窗



(a) TCP吞吐量



(b) DCTCP吞吐量



(c) TCCS吞吐量

图 11 TCP、DCTCP 和 TCCS 整体吞吐量变化

口动作字段,使大象流的吞吐量持续降低. 同时,大象流让出的带宽被老鼠流使用,随着大象流的吞吐量持续走低,老鼠流的吞吐量持续提高,直到数据流传输完成. 在传输完成之后,控制器需要一定时间来撤销原来修改接收窗口的流表项,这时大象流吞吐量仍然较低,导致此时总吞吐量会有一个低极值,之后整个网络恢复到高吞吐量状态. 整个过程中,除了老鼠流在传输完成的一瞬间,总体吞吐量的平均值保持在 850 Mbps 左右,有效地证明了 TCCS 机制的拥塞控制效果.

最后,本实验观察老鼠流数量增加时,TCP、DCTCP 和 TCCS 三者间的超时重传情况. 实验结果如图 12 所示.

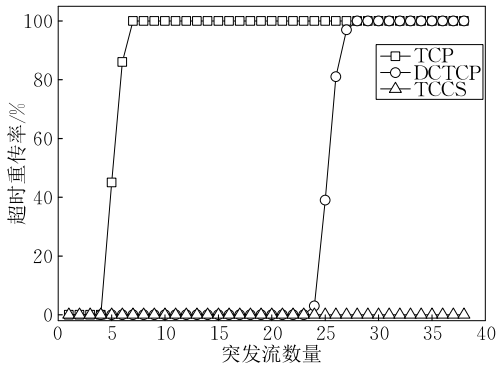


图 12 超时重传率

从实验结果可以看出,TCP 和 DCTCP 分别在并发数据流数量为 7 条和 27 条时,所有数据流都有超时重传发生,而老鼠流数量增加到 40 时,TCCS 没有发生超时重传,这主要是由于 TCCS 机制下,降低大象流发送端的发送速率,从而使得网络能够容纳更多的老鼠流,而且 TCCS 能有效地降低交换机的端口队列长度,达到拥塞避免的效果.

(2) 交换机队列缓存变化实验

本实验主要检测 TCP、DCTCP 和 TCCS 三种协议对交换机队列缓存占用情况. 本实验场景开始有 2 条大象流传输数据并持续至实验结束,28 条老鼠流在 1.5 s 时开始传输数据,老鼠流共传输 10 MB 数据. 实验结果如图 13 所示.

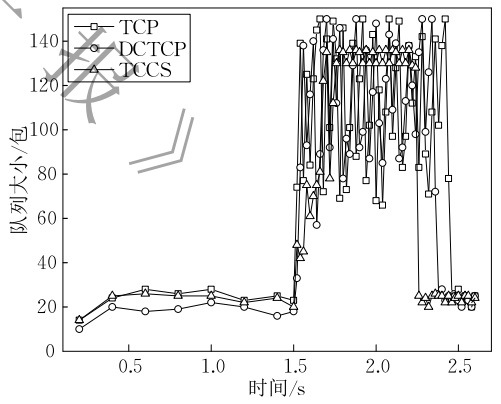


图 13 队列缓存变化

从图 13 可以看出,在 1.5 s 之前,TCP 和 TCCS 占用的队列缓存较为一致,队列缓存大小在 14～28 个报文间震荡,而 DCTCP 占用队列缓存稍微较小,在 10～22 个报文间震荡. 在这段时间三种协议未发生丢包现象. 25 条老鼠流加入之后,三个协议占用队列缓存都较大增加,但是 TCP 和 DCTCP 协议占用队列缓存都达到 150 个报文,也即占满队列缓存,因此会产生丢包现象,而 TCCS 虽然占用缓存也增加但是尚未占满队列缓存. 这主要是由于数

据流在突破交换机阈值之后，修改接收窗口，降低大象流传输速率，而随着队列缓存占用持续走高，甚至超过了控制器上设置的 *High_threshold* 后，TCCS 对整个数据流进行大规模调整，因此队列缓存大小维持在 135 个报文左右，保证了数据报文的不丢失.

5.3　复杂网络实验

该部分实验采用传统的树形网络拓扑结构. 如图 14 所示，服务器与边缘交换机的链路带宽为 1 Gpbs，边缘交换机与汇聚交换机，汇聚交换机与核心交换机的链路带宽都是 10 Gpbs. 所有链路间传输延迟为 20 μs. 交换机与控制器直接相连.

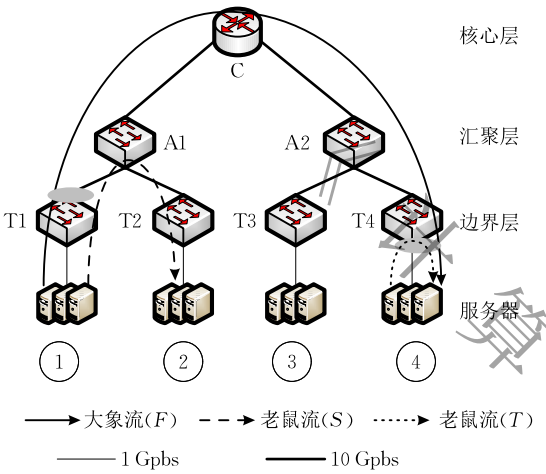
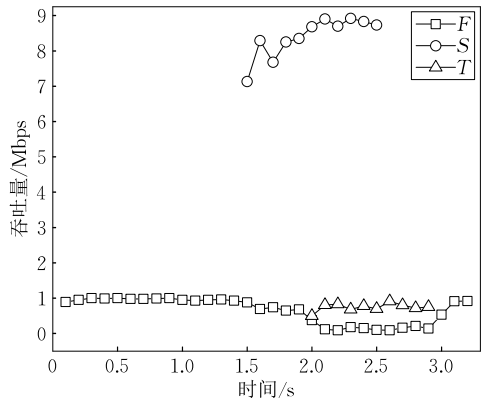


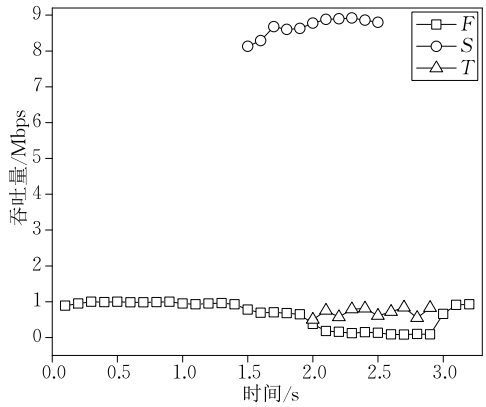
图 14　树形网络拓扑结构

该部分实验中我们生成 3 组较为典型的数据流：(1) 跨核心交换机的大象流(记为 F)，从服务器①到服务器④，有 2 条大象流；(2) 跨汇聚交换机的老鼠流(记为 S)，从服务器①到服务器②，有 28 条老鼠流；(3) 边界交换机内的老鼠流(记为 T)，服务器④内部，有 38 条老鼠流. 大象流 F 首先开始传输数据，1.5 s 时老鼠流 S 开始传输，2 s 时老鼠流 T 开始传输，2.5 s 时老鼠流 S 结束传输，3 s 时老鼠流 T 结束传输. 实验结果如图 15 所示.

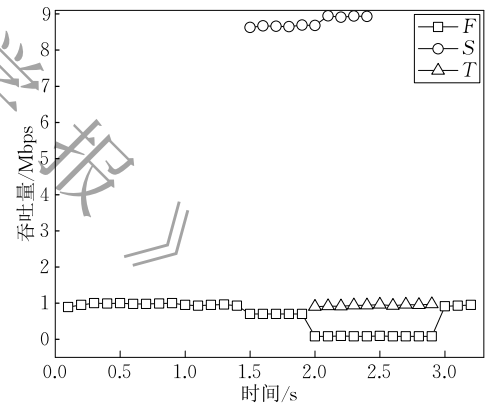
在 TCCS 机制下，大象流吞吐量比较平坦，这主要是由于 TCCS 机制直接限制大象流的发送速率. 当老鼠流 S 加入网络后，我们发现大象流会被进一步降低速率，从而缓解 T1 交换机上的拥塞，导致老鼠流 T 的吞吐量上升. 从实验中我们可以看出，TCCS 机制能够较好地解决网络多拥塞点的情况. 而 DCTCP 和 TCP 却没有这种特性. 另外，从整体吞吐量来看，TCCS 机制也要明显优于 TCP 和 DCTCP. DCTCP 次之，这主要是由于 T4 交换机上产生拥塞并丢包，整体吞吐量下降. TCP 吞吐量最差，且波动较大.



(a) TCP



(b) DCTCP



(c) TCCS

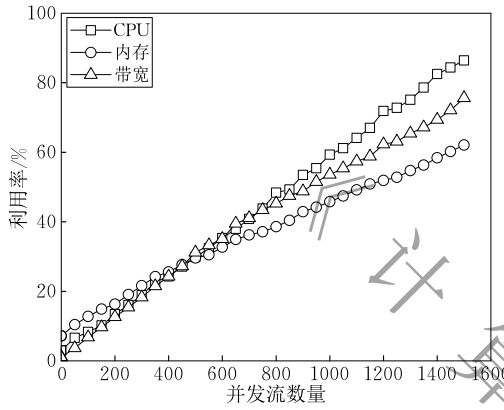
图 15　树形拓扑结构下吞吐量

5.4　控制器压力测试

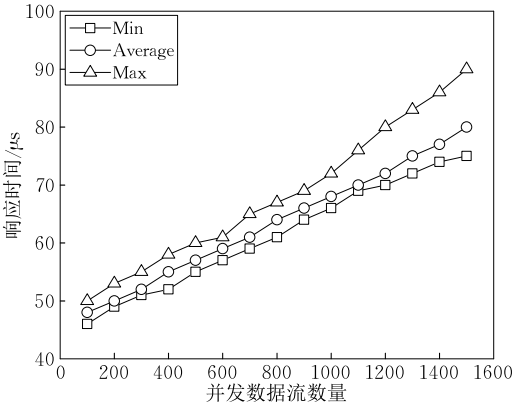
在 SDN 网络中，当一条新的数据流的第一个数据报文到达交换机时，交换机会将该数据报文发送到控制器. 此时，控制器会计算该数据流的路由并下发流表. 在本系统中，控制器不仅要进行路由计算，还要记录数据流信息并生成 GVT. 本实验为了测试大量并发数据流时，控制器能够及时处理并发数据报文，对控制器进行压力测试. 该实验采用图 14 所示拓扑，4 组服务器一共产生 1500 条数据流，实验过程中记录控制器的 CPU、内存和带宽利用率，以

及数据流第一个报文的应答时间(交换机与控制器间上报数据和收到流表项的时间).

如图 16(a)所示,随着并发数据流数量的增加,系统各项资源开销随之提升,在 0～800 个 TCP 数据流并发连接情况下,控制器的 CPU、内存和带宽的利用率都保持在 50%以下,然而随着并发数据流数量的持续增长,CPU 和带宽资源消耗较快,在一定程度上影响控制器系统的使用.但是正如文献[4]中所说,通过分布式控制器的方式,能够满足现有数据中心的性能需求.



(a) 控制器资源利用率



(b) 控制器应答时间

图 16 控制器资源利用率和应答时间

图 16(b)中,随着并发数据流数量的增长,控制器应答时间也线性增加,控制器应答的平均延迟保持在 42 至 72μs 间波动,该时间包括链路延迟时间(2×20 μs=40 μs)和控制器对报文的处理时间.我们知道控制器应答延迟主要影响 TCP 数据流的建立时间(也即 TCP 的三次握手时间),对于数据传输延迟没有影响.另外,我们知道网络同时发生大规模拥塞的概率较小,也即不存在同时大规模触发拥塞通告消息的场景,因此控制器对拥塞通告消息的处理会比较及时,几乎不影响网络性能.

综上所述,整个控制器对数据流的处理速度以及抗压能力都是可以接受的.

6 结论和展望

在当前的数据中心网络中既存在着大量的大象流,也存在着老鼠流,而老鼠流是造成 TCP incast 问题的最主要原因.本文借助 SDN/OpenFlow 技术的优势,提出 TCCS 机制.TCCS 机制首先利用 OpenFlow 协议生成全局 TCP 流表,其次,通过拥塞触发模块监控队列并产生拥塞通告信息至控制器,然后控制器控制大象流 ACK 报文的接收窗口来降低相应数据流的发送速率,解决 TCP incast 问题.文中我们通过实验比较 TCCS 和 TCP NewReno、DCTCP 的网络性能.实验结果表明,TCCS 机制通过降低大象流的发送速率,满足更多的老鼠流而不产生网络拥塞,网络性能得到极大的提高,与此同时,在大量老鼠流产生时,控制器性能也能得到保障,不会影响 TCCS 机制的响应时间.

本文下一步希望进一步完善 TCCS 机制,主要表现在两个方面:(1)当前的 TCCS 机制中的队列管理采用是最简单的尾丢弃方式,我们下一步针对 RED 队列管理方式进行深入探讨;(2)TCCS 机制并没有考虑数据流的截止期限要求,我们在之后的工作中将重点考虑该问题;(3)我们将采用当前主流的数据中心拓扑结构——FatTree 结构,构建更加真实和复杂的测试环境,获得更加真实的结果.

致 谢 审稿专家和编辑为本文提出了许多宝贵的意见和建议,作者在此表示衷心的感谢!

参 考 文 献

[1] Alizadeh M, Greenberg A, Maltz D, et al. Data center TCP (DCTCP). ACM SIGCOMM Computer Communication Review, 2010, 40(4): 63-74

[2] Greenbei A, Hamilton J R, Jain N. VL2: A scalable and flexible data center network. ACM SIGCOMM Computer Communication Review, 2009, 39(4): 51-62

[3] Benson T, Anand A, Akella A, et al. Understanding data center traffic characteristics. ACM SIGCOMM Computer Communication Review, 2010, 40(1): 92-99

[4] Benson T, Akella A, Maltz D A. Network traffic characteristics of data centers in the wild//Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC 2010). Melbourne, Australia, 2010: 267-280

[5] Mckeown N, Anderson T, Balakrishnan H, et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74

[6] Vamanan B, Hasan J, Vijaykumar T N. Deadline-aware datacenter TCP. ACM SIGCOMM Computer Communication Review, 2012, 42(4): 115-126

[7] Wang G, Ren Y, Dou K, et al. IDTCP: An effective approach to mitigating the TCP incast problem in data center networks. Information Systems Frontiers, 2014, 16(1): 35-44

[8] Zheng F, Huang Y, Sun D. Designing a new TCP based on FAST TCP for datacenter//Proceedings of the IEEE International Conference on Communications (ICC 2014). Sydney, Australia, 2014: 3209-3214

[9] Zhang J, Wen J. RETCP: A ratio estimation approach for data center incast applications//Proceedings of the IEEE International Conference on Communications (ICC 2015). London, England, 2015: 338-343

[10] Wang J, Wen J, Li C, et al. DC-Vegas: A delay-based TCP congestion control algorithm for datacenter applications. Journal of Network and Computer Applications, 2015, 53: 103-114

[11] Bai W, Chen K, Wu H, et al. PAC: Taming TCP incast congestion using proactive ACK control//Proceedings of the IEEE 22nd International Conference on Network Protocols (ICNP 2014). Raleigh, USA, 2014: 385-396

[12] Wu H, Feng Z, Guo C, et al. ICTCP: Incast congestion control for TCP in data center networks//Proceedings of the ACM CoNEXT. Philadelphia, USA, 2010: 13-13

[13] Zhang J, Ren F, Yue X, et al. Sharing bandwidth by allocating switch buffer in data center networks. IEEE Journal on Selected Areas in Communications, 2014, 32(1): 39-51

[14] Vasudevan V, Phanishayee A, Shah H, et al. Safe and effective fine-grained TCP retransmissions for datacenter communication//Proceedings of the ACM SIGCOMM. Barcelona, Spain, 2009: 303-314

[15] Phanishayee A, Krevat E, Vasudevan V, et al. Measurement and analysis of TCP throughput collapse in cluster-based storage systems//Proceedings of the USENIX FAST. San Jose, USA, 2008: 175-188

[16] Cheng P, Ren F, Shu R, et al. Catch the whole lot in an action: Rapid precise packet loss notification in data centers//Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014). Seattle, USA, 2014: 17-28

[17] Zhang J, Ren F, Tang L, et al. Taming TCP incast throughput collapse in data center networks//Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP 2013). Goettingen, Germany, 2013: 1-10

[18] Chen Y, Griffith R, Liu J, et al. Understanding TCP incast throughput collapse in datacenter networks//Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN 2009). Barcelona, Spain, 2009: 73-82

[19] Dou K, Ren Y, Li J. Avoiding TCP incast through scheduling data requests//Proceedings of the 4th International Conference on MINES. Nanjing, China, 2012: 453-457

[20] Krevat E, Vasudevan V, Phanishayee A. On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems//Proceedings of the 2nd International Petascale Data Storage Workshop (PDSW 2007). Reno, USA, 2007: 1-4

[21] Podlesny M, Williamson C. Solving the TCP-incast problem with application-level scheduling//Proceedings of the MASCOTS. San Francisco, USA, 2012: 99-106

[22] Podlesny M, Williamson C. An application-level solution for the TCP-incast problem in data center networks//Proceedings of the IWQoS. San Jose, USA, 2011: 1-3

[23] Ghobadi M, Yeganeh S H, Ganjali Y. Rethinking end-to-end congestion control in software-defined networks//Proceedings of the HotNets. Seattle, USA, 2012: 1-6

[24] Jouet S, Pezaros D P. Measurement-based TCP parameter tuning in cloud data centers//Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP 2013). Goettingen, German, 2013: 1-3

[25] Gruen J, Karl M, Herfet T. Network supported congestion avoidance in software-defined networks//Proceedings of the 19th IEEE International Conference on Networks. Singapore, 2013: 1-6

[26] Jouet S, Perkins C, Pezaros D. OTCP: SDN-managed congestion control for data center networks//Proceedings of the Network Operations and Management Symposium. Istanbul, Turkey, 2016: 1-9

[27] Li Long, Fu Bin-Zhang, Chen Ming-Yu, et al. Nimble: A fast flow scheduling strategy for OpenFlow networks. Chinese Journal of Computers, 2015, 38(5): 1056-1068(in Chinses)
(李龙，付斌章，陈明宇等. Nimble：一种适用于 OpenFlow 网络的快速流调度策. 计算机学报，2015，38(5)：1056-1068)

[28] Jonathan P, Amy O, Hari B, et al. Fastpass: A centralized "zero-queue" datacenter network. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 307-318

[29] Abdelmoniem A M, Brahim B. Incast-aware switch-assisted TCP congestion control for data centers//Proceedings of the IEEE Global Communications Conference(GLOBECOM 2015). San Diego, USA, 2015: 3-10

[30] Lee C, Nakagawa Y, Hyoudou K, et al. Flow-aware congestion control to improve throughput under TCP incast in datacenter networks//Proceedings of the IEEE 39th Annual Computer Software and Applications Conference Computer. Taichung, China, 2015: 155-162

LU Yi-Fei, born in 1982, Ph. D., lecturer. His current research interests include transmission protocol in data center network, software-defined networking and storage network.

ZHU Shu-Hong, born in 1991, M. S. His current research interests include software-defined networking and traffic engineering.

Background

In today's data center networks (DCN), TCP has been used as the de facto transport layer protocol to ensure reliable data delivery. However, the unique workloads scale and environments of the data center work violate the WAN assumptions on which TCP was originally designed. A reported open problem is TCP incast. TCP incast problem was initially identified in distributed storage cluster and has nowadays become a practical issue in data center networks. TCP incast, which results in gross under-utilization of link capacity, occurs in synchronized many-to-one communication patterns.

Ghobadi et al. proposed an OpenTCP that is presented as a system for dynamic adaptation of TCP based on network and traffic conditions in Software Defined Networks (SDN). OpenTCP is not a new variation of TCP. Instead, it complements previous efforts by making it easy to switch between different TCP variants automatically (or in a semi-supervised manner), or to tune TCP parameters based on network conditions. For instance, one can use OpenTCP to either utilize DCTCP or CUBIC in a data center environment. The decision on which variant to use is made in advance through the congestion control policies defined by the network operator. Jouet S et al. proposes an SDN approach to tune TCP initial window and retransmission timers for newly created flows based on a network-wide view. This method allows the online tuning of these parameters in order to improve response time for mice flows by designing a measurement-based control loop with an SDN controller as the feedback source.

Previous works focus on how to tune TCP parameters or choose between different TCP variants automatically, while our work is aim to avoid packet loss by designing a new TCP congestion avoidance mechanism, which makes our work complementary to previous work.

In this paper, we take this advantage of SDN to design and implement an innovative TCP congestion control mechanism based on SDN (TCCS). The key design principle behind TCCS is that we avoid network congestion by adjusting the received window of TCP ACK packet at controller to reduce sender's transmission rate. We deploy a very simple queue management scheme in OpenFlow-switch that triggers a congestion notification to controller when queue occupancy is greater than threshold. When controller receives this signal, it will select background flows and push a new flow table entry to modify advertised window of ACK packet at OF-switch automatically. TCCS does not need to modify the TCP stack of end system.

This project was partially supported by the Jiangsu Provincial Postdoctoral Science Foundation and the National Natural Science Foundation of China under Grant No. 61602243.