# 第5章

# 数据中心网络的拥塞控制协议

随着云计算技术在大数据、人工智能等领域的应用和发展，作为云计算核心支撑平台的数据中心网络（data center network，DCN）近年来也得到了深入的发展。数据中心将网络资源、计算资源、存储资源以及其他辅助资源（如 FPGA、ASIC 等）整合为一个统一的整体，已经成为云计算、人工智能、移动计算等计算密集型业务的基础设施，其运行效率直接决定了这些业务的服务质量。而在现有的数据中心网络中，大部分应用仍使用 TCP 进行数据传输[1-2]。数据中心网络高带宽、低时延等特性与传统广域网的特性差异较大，传统 TCP 在数据中心网络中的运行效率较低，会引发 TCP Incast、TCP Outcast 等一系列问题。

本章主要针对数据中心网络中的 TCP Incast（也称为 TCP 吞吐量崩溃）问题进行研究，该问题是指当多个发送方同时向一个接收方发送数据时，与接收方相连的交换机缓冲区发生拥塞，从而导致网络吞吐量崩溃。TCP Incast 问题的产生将使网络吞吐率急剧下降，并可能导致任务错过截止完成时间而被丢弃，影响计算结果的质量和用户体验。因此，如何缓解该问题以便为分布式存储、Web 搜索等云计算任务提供高效率的数据传输服务是具有实际应用价值的重要课题[3]。

本章从两个方向着手研究 TCP Incast 问题的解决方案，一是针对 TCP 机制在 DCN 中不适应的问题，改进 TCP 的拥塞控制机制，及时探测网络拥塞，快速缓解拥塞，尽量减少丢包，改善传输性能的同时降低数据流完成时间；二是基于近年来出现的 SDN 思想，利用 SDN/OpenFlow 技术收集网络信息，包括网络流量信息和交换机队列信息等，并通过这些信息更加精确、更加快速地进行拥塞判断和拥塞控制。

## 5.1 引言

随着云计算、大数据、物联网、人工智能等新一代信息技术快速发展，数据呈现爆炸式增长，使得互联网企业对数据中心基础设施的需求不断增长。据最新研究报告显示，2019 年中国数据中心数量约有 7.4 万个，约占全球数据中心总量的 23%，数据中心机架规模达到 227 万架，在用 IDC 数据中心有 2213 个。数据中心大型化、规模化趋势仍在延续[4]。数据

中心是云计算系统的主干，由数千个相互连接的提供云服务的计算机节点组成。这些服务器与交换机通过高速网络互相连接形成数据中心网络。数据中心网络通过视频流和云计算等集中基础设施向订阅用户提供多样化的网络服务和大量大规模计算[5]。

为了保证提供可靠的传输服务，数据中心的大部分应用仍使用 TCP 作为传输协议。而 DCN 有一些不同于其他网络（如互联网或局域网）的特点：首先，与传统 Internet 网络相比，数据中心网络具有高带宽、低时延的网络特性。DCN 是为数据密集通信设计的，用于提供大规模计算服务，如网络存储、电子邮件和网络搜索。这些服务和应用程序需要网络具有高带宽和低时延的特点，以实现 DCN 分布式组件之间数据的高速传输。近年来，随着数据中心的发展和互联网技术的提高，数据中心网络的带宽已经从 1Gbps 提升到 10Gbps 甚至 100Gbps。其次，DCN 有数千台紧密相连的服务器，链路的超额订阅率通常为 1∶1，不可避免会出现链路拥塞的情况。图 5.1 显示了一个典型的数据中心结构框图，其中服务器被放置在机架上，它们通过架顶式（ToR）交换机连接起来，再连接到汇聚交换机。汇聚交换机可与其他汇聚交换机互连，所有这些汇聚交换机最后连接到核心交换机，通过核心交换机连接到互联网。

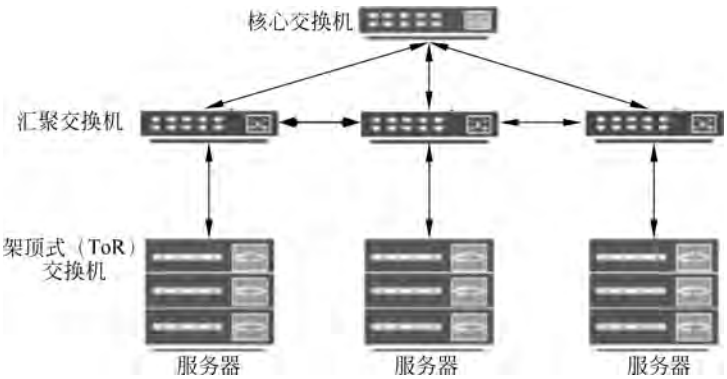


图 5.1　数据中心结构框架

DCN 不仅支持通信，还提供网络中所有服务器之间的容错。DCN 在任意两台服务器之间部署多条路径[6]。DCN 中发送方和接收方之间的 RTT 是微秒级的，而不是传统互联网中的毫秒级[7]。数据中心网络中常见的数据流量类型有两种，一种是背景数据流，称为大象流（也称为长数据流），大象流主要是传输大型文件的数据流，对于时延不太敏感；另一种是老鼠流（也称为短数据流），主要是网络中突发的网络请求/响应数据，老鼠流对时延十分敏感，其响应时间影响着在线服务的用户体验。因此，在 DCN 中既要减少老鼠流的时延又要保证大象流的高吞吐率[8]。

正如前面提到的，数据中心的网络特性和流量特性与传统广域网差异较大，传统 TCP 在数据中心网络中运行会引发诸如 TCP Incast 等问题。

TCP incast 问题是由多对一流量模式和在线查询的高时延造成的。图 5.2 显示了一个典型的 TCP Incast 场景。DCN 中的客户端以多对一通信模式向一个或多个服务器请求数据。这些数据从服务器传输到客户端，需要经过交换机到客户机的瓶颈链路。在集群文件系统内（如网络搜索或批处理环境中），客户端应用请求某个逻辑数据块（通常情况下一个数据块大小是 1MB），该数据块以条带化方式分别存储在几个存储服务器上，即采用更小的数

据片存储(32KB、256KB 等),这种小数据片称为服务器请求单元(SRU)。只有当客户端接收到所有的服务器返回的其所请求数据块的 SRU 后才继续发送出下一个数据块请求,即客户端同时向多个存储服务器发起并发 TCP 请求,且所有服务器同时向客户端发送 SRU。随着并发发送者数量的增加,这种多对一的服务器向客户端并发传输数据的模式,很容易造成与客户端相连接的交换机端口的缓冲区溢出,从而导致丢包及随后的 TCP 重传。在丢包严重的情况下,TCP 将启动超时重传,此时传输需要经历一个最少持续 200ms 的超时,这是由 TCP 最小重传超时参数(RTOmin)确定的。在并发传输过程中,当某个服务器发生了传输超时而其他服务器已完成了传输,则客户端在接收到剩余的 SRU 之前必须等待至少 RTOmin,而等待期间客户端的链路很有可能处于完全空闲状态,这就导致在应用层的可见吞吐量与链路容量相比较显著下降且总的请求时延将高于 RTOmin[9]。

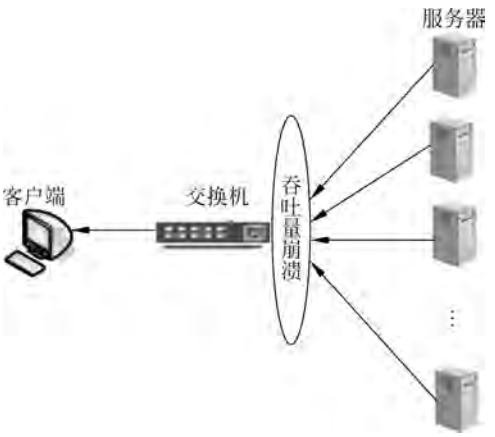


图 5.2 典型的 TCP Incast 场景

如前所述,数据中心中多对一的传输模式常常会导致 TCP Incast 问题,这会降低短数据流的完成时间和长数据流的吞吐量。针对 TCP Incast 问题,研究者已经提出了很多解决方案,包括调节参数的方法、基于 ECN 的方法、基于时延的方法和主动拥塞控制方法。

目前比较有效的解决方法是基于 ECN 的方法以及主动拥塞探测方法中基于 SDN 的方法。本章针对这两类方法进行研究,并提出了基于 ECN 的快速 DCTCP 和基于 SDN 的自适应拥塞控制方法。

## 5.2 快速 DCTCP 协议——FDCTCP

本节基于 DCTCP,提出一种快速 DCTCP,称为 FDCTCP。其目标是得到准确的拥塞反馈并进行快速的速率控制,以实现数据中心网络的高利用率和低排队时延。FDCTCP 与 DCTCP 一样,使用 ECN 来推断网络拥塞并计算拥塞因子。然而,当发送方收到带有 ECN 标记的 ACK 时,与 DCTCP 不同的是,FDCTCP 通过监测观察到的 ECN 标记的比例和拥塞梯度来检测网络拥塞。FDCTCP 根据拥塞级别降低拥塞窗口。如果网络严重拥塞,FDCTCP 将拥塞窗口重置为一个较小的值。如果根据 ECN 标记判断网络中已经没有拥

塞，则 FDCTCP 会直接将拥塞窗口恢复到拥塞发生前的值，而不是像 DCTCP 那样逐渐恢复窗口。

### 5.2.1　FDCTCP 的主要思想

FDCTCP 由三部分组成：网络拥塞检测方法、快速拥塞缓解机制和快速恢复机制。图 5.3 给出了 FDCTCP 框架。在网络拥塞检测方法中，发送方一旦接收到带有 ECN 标记的 ACK，就开始根据拥塞因子和拥塞梯度周期性地预测网络拥塞程度和趋势。当检测到网络正在变得越来越拥挤时，快速拥塞缓解机制会根据拥塞程度的不同而适当减小拥塞窗口。若随后接收到的 ACK 没有 ECN 标记，则快速恢复机制将把拥塞窗口恢复到拥塞发生前的值。

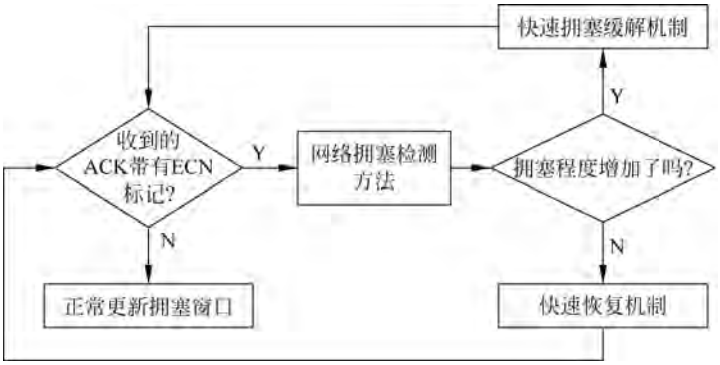


图 5.3　FDCTCP 框架

### 5.2.2　FDCTCP 的详细实现

图 5.4 是 FDCTCP 的流程图，具体实现描述如下。

1. 网络拥塞检测方法

为了对网络拥塞进行快速可靠的检测，网络拥塞检测方法同时利用拥塞因子和拥塞梯度定期估计网络拥塞。启用了 ECN 的路由器实时监测队列长度，若瞬时队列长度超过预先定义的阈值，则后面到来的分组中的 ECN 位将被标记为 1。拥塞因子是发送方收到的带有 ECN 标记的 ACK 分组占总的 ACK 分组的比例。拥塞因子在一定程度上可以用来推断网络拥塞情况。但这还不够，为了提高网络拥塞检测的可靠性，还需要了解拥塞的变化趋势。因此，采用拥塞梯度来估计网络拥塞的变化趋势。网络拥塞检测方法每隔一段时间（记为 period）估计一次网络拥塞状态，并计算自上次估计以来的拥塞因子（记为 current_F）和拥塞梯度（记为 diff）。拥塞因子和拥塞梯度计算公式如下：

$$\text{current\_F} = \text{ECE\_num}/\text{total\_packet} \tag{5.1}$$

$$\text{diff} = \text{current\_F} - \text{prev\_F} \tag{5.2}$$

式中，ECE_num 是在检测周期内观察到的 ECN 标记数，total_packet 是周期内接收到的数据包数。由于估计周期相同，拥塞梯度（diff）即为相邻周期的拥塞因子（即 current_F 和 prev_F）之差。

若计算出来的拥塞因子较大，则表明发送方观察到的 ECN 标记的比例增加，也就是说

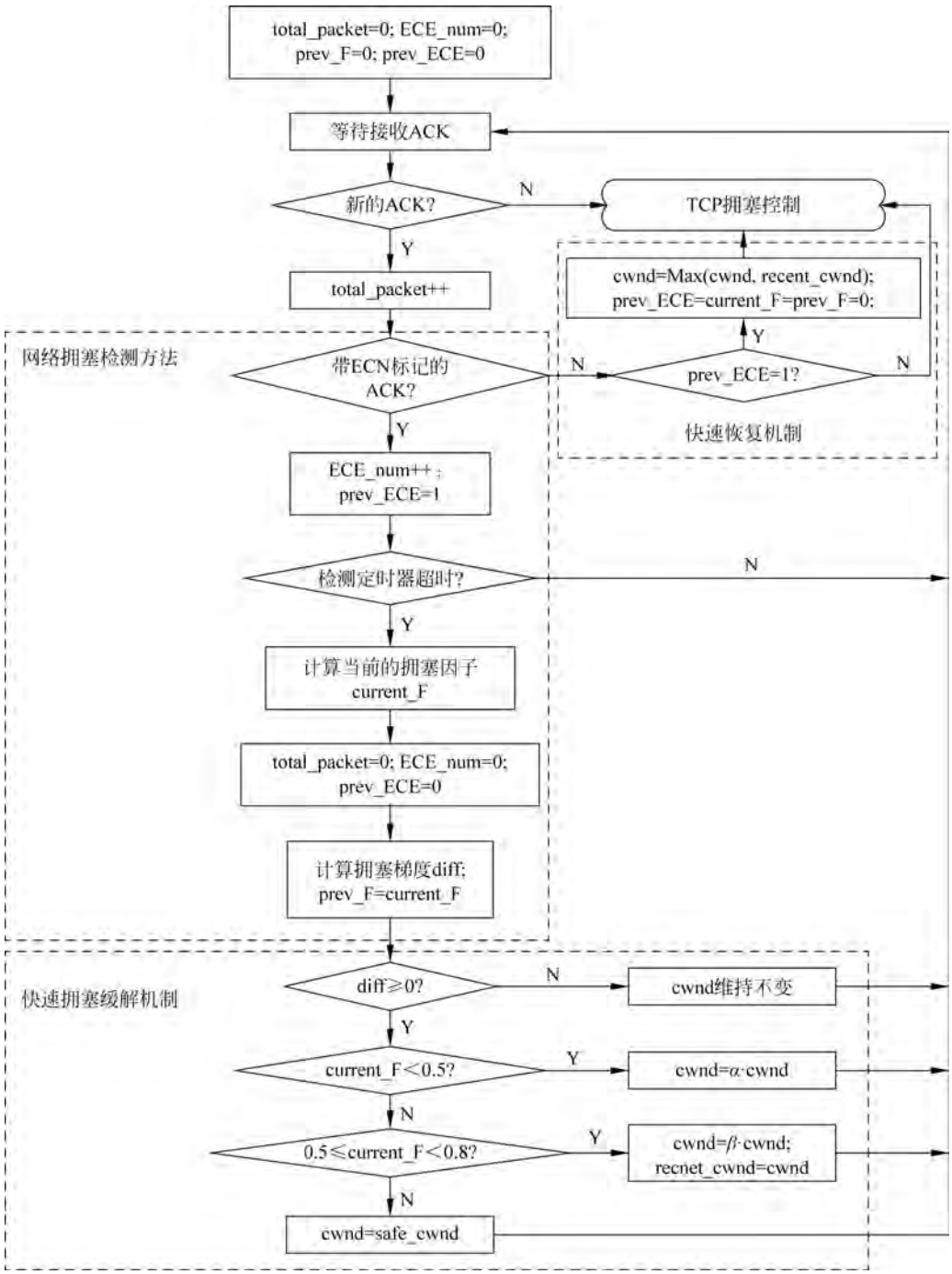


图 5.4　FDCTCP 的流程图

存在潜在的网络拥塞。因此，拥塞因子可以用来估计拥塞程度。若拥塞梯度 diff＞0，则表明拥塞因子增大，网络拥塞变得越来越严重；若 diff＜0，则表明拥塞因子减小，网络拥塞得到缓解。因此，可以利用拥塞梯度变化来检测网络拥塞的变化趋势。

2. 快速拥塞缓解机制

FDCTCP 利用拥塞因子和拥塞梯度按以下规则估计拥塞程度：

(1) 若 diff≥0 并且 current_F＜0.5，则可能是轻微拥塞；

(2) 若 diff≥0 并且 0.5≤current_F＜0.8，则可能是持续拥塞；

(3) 若 diff≥0 并且 current_F≥0.8，则可能是严重拥塞；

(4) 若 diff＜0，则可能是拥塞正在缓解。

当探测到网络拥塞时，为了尽快缓解网络拥塞，FDCTCP 使用快速拥塞缓解机制，根据拥塞程度减少当前的拥塞窗口。若认为网络是轻微拥塞，则拥塞窗口适当减小为原来的 $\alpha$ 倍；若认为网络是持续拥塞，则将拥塞窗口减小为原来的 $\beta$ 倍，并将此拥塞窗口值记为 recent_cwnd；若认为网络严重拥塞，即使没有丢包，拥塞窗口也会被减小到一个预定的安全值(记为 safe_cwnd)；若网络拥塞正在缓解，则拥塞窗口保持不变。

3. 快速恢复机制

FDCTCP 利用快速拥塞缓解机制，实现对 Incast 拥塞的快速激进反馈，从而有效缓解拥塞。一旦 Incast 结束，发送方将不会收到带有 ECN 标记的 ACK。此时拥塞窗口应迅速恢复，以实现高吞吐量。若发送方在一段时间内没有再收到任何带有 ECN 标记的 ACK，则快速恢复机制会将该窗口恢复到之前保存的窗口值(即 recent_cwnd)。

### 5.2.3　仿真实验结果

本节使用 OPNET Modeler 对当前数据中心里常见的叶脊拓扑网络进行仿真。叶脊拓扑网络由上层的脊交换机和下层的叶交换机组成，其中，脊交换机充当汇聚交换机的角色，叶交换机充当接入交换机的角色。在叶脊拓扑网络结构中，所有的叶交换机都和每一台脊交换机连接，因此，任何一台服务器和另一台服务器间的数据传输只需要经过一台叶交换机和一台脊交换机。本节的实验使用 4 台叶交换机，每台叶交换机连接 10 台服务器，如图 5.5 所示。所有链路带宽为 10Gbps，在没有负载的情况下链路时延为 20μs。

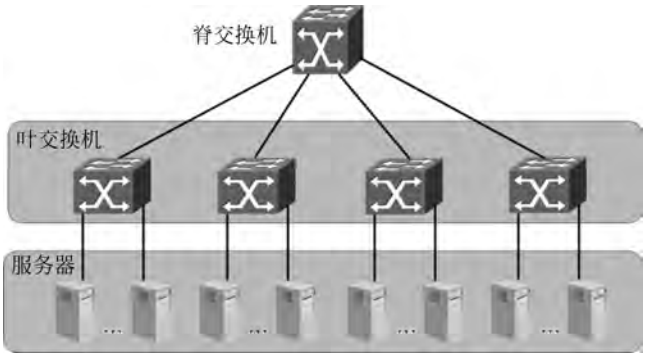


图 5.5　叶脊拓扑网络图

实验使用文献[10]中给出的流量模型，模型中混合了短数据流和长数据流。假设数据流的到达服从泊松分布，其中短数据流大小在 8～32KB 随机选择，长数据流大小设置为 1MB。长数据流的数量占所有数据流数量的 30%。仿真从 0s 开始，在所有数据流传输完成时停止。在 0～2s 期间，所有的数据流每 10ms 被请求一次。

本节通过实验比较 FDCTCP 和 DCTCP 在不同服务器数量和不同缓存大小下的数据流完成时间、吞吐率和队列长度。DCTCP 的实现使用了协议推荐的参数设置（如 ECN 的阈值）。FDCTCP 基于 DCTCP 在终端主机中实现拥塞控制。其余参数设置如下：safe_cwnd＝4MSS，$ECN_{threshold}=20$，$\alpha=0.8$，$\beta=0.5$，period＝40$\mu$s。

1. 数据流完成时间

下面比较 DCTCP 和 FDCTCP 的流完成时间。图 5.6 和图 5.7 分别表示长数据流和短数据流全部结束时的完成时间。DCTCP-120KB 表示缓冲区大小为 120KB 时 DCTCP 的数据流完成时间，可以看出，无论是短数据流还是长数据流，DCTCP 的数据流完成时间都会随着服务器数量和缓冲区大小的增加而增加。然而，在不同的服务器数量和缓冲区大小下，FDCTCP 保持较稳定的数据流完成时间，且明显小于 DCTCP 的数据流完成时间，其中长数据流的完成时间仅为 DCTCP 的 60%左右，短数据流的完成时间约为 DCTCP 的 78%。这是由于：首先，FDCTCP 采用可靠的网络拥塞检测方法，准确检测网络拥塞程度和变化趋势；其次，FDCTCP 采用快速拥塞缓解机制来降低发送速率，以避免出现 Incast 和大量分组丢失。

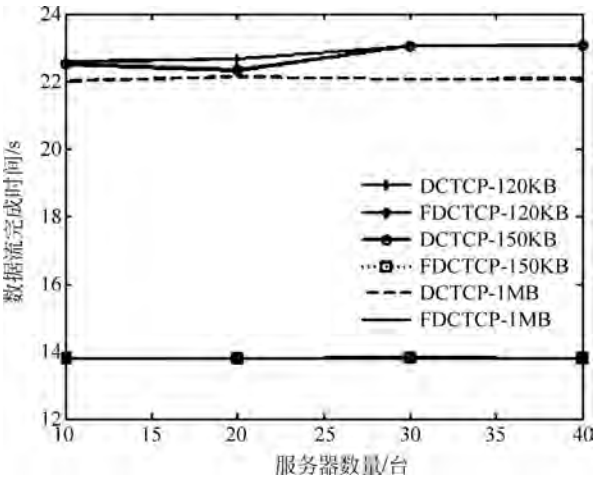


图 5.6　不同服务器数量和不同缓存大小下长数据流的完成时间

图 5.8 显示了一条长数据流中拥塞窗口的变化情况，从图中可见，当检测到 Incast 发生，FDCTCP 及时对网络拥塞进行反馈并将拥塞窗口减小到安全窗口（safe_window），而 DCTCP 不断收到带有 ECN 标记的 ACK，然后逐渐减小窗口。由于 FDCTCP 中长数据流的发送方及时进行窗口回退，所以使得队列长度减小，从而有助于减少短数据流的完成时间。

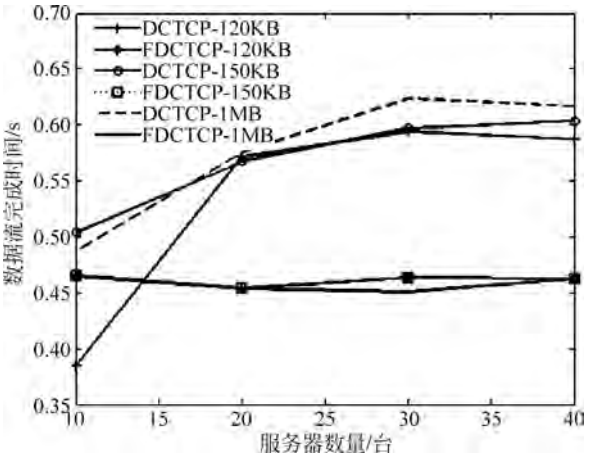


图 5.7　不同服务器数量和不同缓存大小下短数据流的完成时间

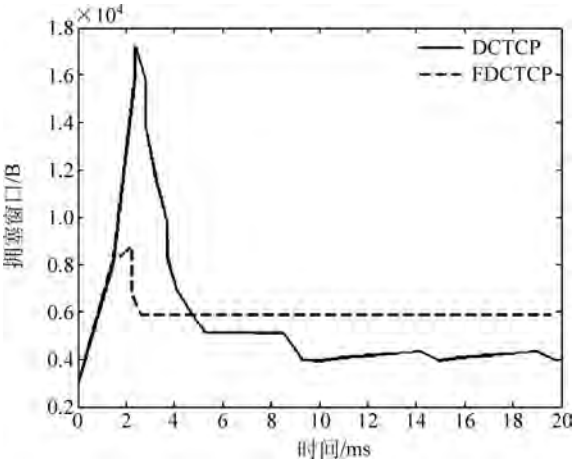


图 5.8　一条长数据流的拥塞窗口变化

2. 吞吐率

下面将比较 FDCTCP 和 DCTCP 的吞吐率。从图 5.9 可以看出，FDCTCP 和 DCTCP 的吞吐率随着缓冲区大小的增加而增加，而服务器的数量对它们的吞吐率的影响很小。即使 ECN 阈值被设置为一个较小的数值(如 20)，FDCTCP 在不同服务器数量下都比 DCTCP 实现更高的吞吐率，具体来说，FDCTCP 实现了比 DCTCP 高 67%的平均吞吐率。出现这样的结果的主要原因包括：首先，与 DCTCP 相比，FDCTCP 使用快速拥塞缓解机制减少了那些共享拥塞链路的背景数据流的丢包数量；第二，当 Incast 结束时，FDCTCP 采用快速恢复机制，使用 Incast 发生之前的最后一个拥塞窗口作为新的拥塞窗口，而不是采用逐渐增加窗口的方式(如慢启动)。这些机制有助于发送方快速地恢复到拥塞前的速率，从而获得良好的吞吐率性能。

3. 队列长度

下面将分析 FDCTCP 的队列长度。图 5.10 显示了一个脊交换机输出端口上的平均队列长度。可以看出，DCTCP 和 FDCTCP 的队列长度随着服务器数量的增加而增加，大部分

情况下 FDCTCP 的队列长度比 DCTCP 的队列长度要短(除了服务器数量为 40 以及缓冲区较小时)。当数据开始传输时,随着服务器数量的增加,会有更多的数据包进入网络。因此,FDCTCP 和 DCTCP 的队列会在启动时快速建立。在传输过程中,DCTCP 会由于较小的缓冲区而导致数据包丢失,从而减少了队列长度,而 FDCTCP 可以保持稳定的拥塞窗口和队列长度。因此,在服务器数量为 40 以及缓冲区较小的情况下,FDCTCP 的平均队列长度略大于 DCTCP。当缓冲区大小增加到 1MB 时,DCTCP 不会很快检测到丢包,而其使用的窗口更新策略较迟缓,所以无法有效控制队列长度。FDCTCP 由于具有快速的拥塞缓解机制,可以保持较小且稳定的队列长度。从图 5.10 可以看出,FDCTCP 相比 DCTCP 减少的队列长度高达 69%,因此可以大大缩短短数据流的完成时间。

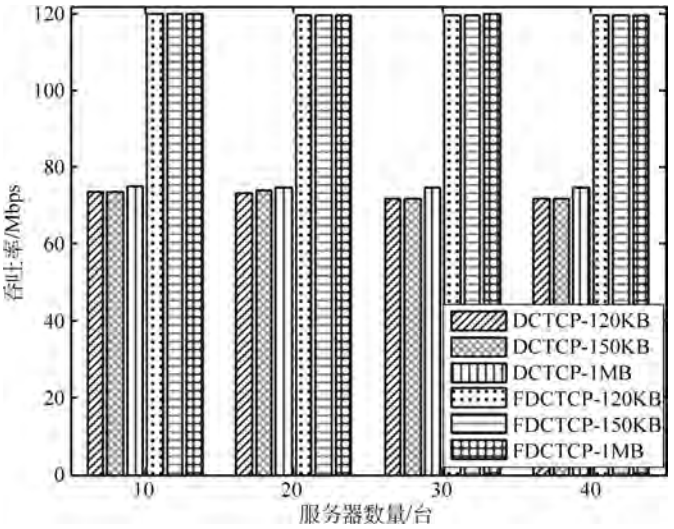


图 5.9 不同服务器数量和不同缓存大小下长数据流的吞吐率

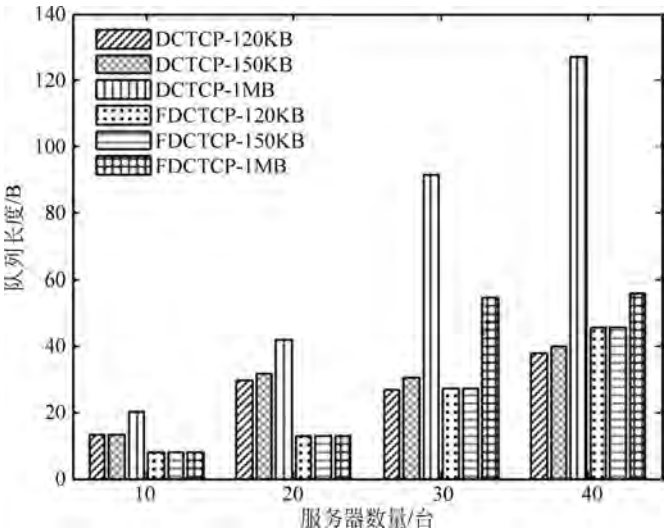


图 5.10 不同服务器数量和不同缓存大小下的平均队列长度

## 5.3 一种基于SDN网络的自适应可靠数据传输方法

传统互联网中使用TCP实现数据的可靠传输。但TCP仅通过端到端的网络参数估计网络状态,无法直接感知报文在中间路径上传送的状态,因此不能及时、准确跟踪网络状态的变化并动态调整传输参数,在数据中心网络中极易导致严重的网络拥塞,造成丢包率高、网络带宽利用率低等问题。当前研究的主要方法是通过对网络参数的估计了解网络拥塞程度,并通告发送端,以此调整发送速率,缓解或解决网络拥塞。这些方法虽然能够在一定程度上缓解拥塞,但是传输性能的提升仍然有限,因此研究者考虑引入新的研究思路和技术手段来解决数据中心网络拥塞问题。

软件定义网络(SDN)[11]是一种新型的基于软件的网络架构及技术,SDN将交换机控制平面(control plane)与数据平面(data plane)分离,利用控制器通过南向接口向数据平面上的流表下载规则(rule),从而指导数据包转发。由于TCAM① 查找速度快且支持通配符查询,因此当前大量商用硬件SDN交换机都采用TCAM存放流表[12]。目前应用最广泛的南向接口是OpenFlow[13],一条规则可以包含1个或多个字段(field),并伴有匹配到的数据包进行操作的动作(action)。

SDN/OpenFlow技术可以统计网络信息,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、快速地进行网络状态的判断,为数据中心网络的拥塞控制和数据传输研究提供了全新的解决思路。然而在基于SDN的TCP传输协议研究中,一些解决方案仅通过SDN控制器实现部分初始传输参数的修改,并未优化传输过程中的传输速率,另一些解决方案虽然利用SDN控制器实现了拥塞控制,但是传输速率的调整未充分考虑当前的网络状态和数据中心网络的数据特性。

本节提出一种基于SDN的自适应可靠数据传输机制(adaptive reliable transmission control based on SDN,ARTCS),针对当前网络状态和数据中心网络的数据特性,增强数据传输对网络状态的自适应性,在兼顾当前的网络状态和数据中心网络的数据特性的同时,优化传输过程中的传输速率。在连接建立时,根据SDN控制器获得的网络状态统计信息,设置TCP流的初始传输窗口,有效减少数据中心网络中老鼠流的传输时间。传输过程中自动检测拥塞,并根据网络拥塞程度调整TCP流传输速率,有效缓解拥塞,提高网络带宽利用率,实现数据中心网络中数据的高效传输。该机制根据网络状态动态调整TCP流的初始窗口和传输速率,减少拥塞丢包的同时充分利用网络带宽资源,利用SDN网络环境,直接获取网络的状态信息,增强了网络状态参数估计的准确性,提升了数据传输效率。

### 5.3.1 SDN的工作模式

SDN支持2种工作模式:被动模式(reactive)和主动模式(proactive)。

1. 被动模式

所谓被动模式,是指任何在交换机流表中找不到匹配项的数据包,都以Packet-In的形

① TCAM(ternary content addressable memory)是一种三态内容存储器,支持通配符查找,可以在1个时钟周期内给出查找结果。

式发送给控制器，由控制器根据策略生成流表项后，通过 Flow-Mod 下发到交换机。在被动模式下，SDN 的网络管理将会更加灵活。但是交换机的性能很容易受到控制器的处理速度、交换芯片与 Local CPU 之间的带宽等因素的影响[14]。因此在当前网络基础设施下，工业界普遍偏向于主动的工作模式。

2. 主动模式

控制器将所有流表项计算出，并提前下发到交换机流表中(尽管该交换机可能还未转发过该流表项对应的数据包)。受 TCAM 大小所限，若当前已知的流表项全部提前下发到交换机，则需要存放在交换芯片的片上存储。由于以上原因，TCAM 一般充当高速缓存(cache)。数据包到达后，先在 TCAM 中查找，若没有匹配则从片上存储中进行查找转发。相比被动模式，主动模式能在一定程度上解决控制器与交换机之间通信时延较大的问题。

图 5.11 展示了 SDN 的数据包处理流程。数据包从端口 1 进入交换机，交换芯片将分组头送入 TCAM 上的流表进行查找，找到则根据相应的 action 进行操作。若在 TCAM 中没有相应的匹配项，则交换芯片将数据包上传到 Local CPU，Local CPU 将数据包封装成 Packet-In，通过交换机上的 OpenFlow Agent 上传到控制器。控制器通过其处理逻辑下发 Flow-Mod 到交换机上的 OpenFlow Agent，进而传到交换芯片并将其插入流表，指导后续数据包转发。

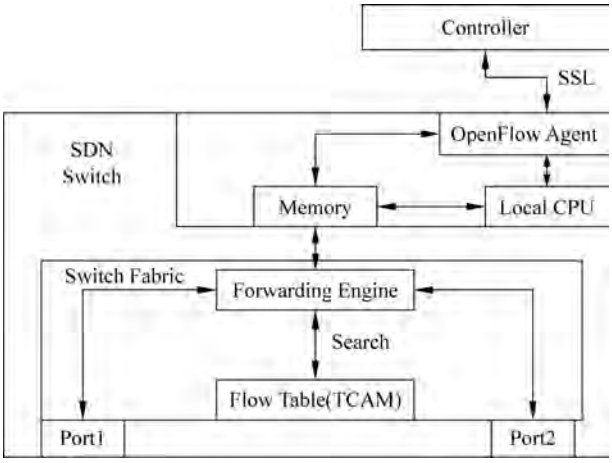


图 5.11 SDN 的数据包处理流程

### 5.3.2 ARTCS 的基本思想

ARTCS 的基本思想为：SDN 控制器对网络中所有数据流信息进行记录和统计(包括端口可用带宽以及主机间的 RTT 值)；根据当前的统计信息计算合适的 TCP 初始窗口值并将此值传送给发送方；通过监测交换机的队列长度进行拥塞检测；当网络发生拥塞时，控制器根据当前的拥塞程度计算接收窗口值，并下发到交换机；当网络拥塞缓解时，控制器删除流表修改项，恢复正常的窗口更新。ARTCS 的基本架构如图 5.12 所示。

ARTCS 的总体工作流程如图 5.13 所示，ARTCS 主要包括信息收集模块、参数计算模块、初始窗口计算模块、初始窗口更新模块、拥塞检测模块、发送窗口估算模块、发送窗口更新模块和拥塞恢复模块。控制器启动后通过信息收集模块获取网络拓扑信息、端口统计信

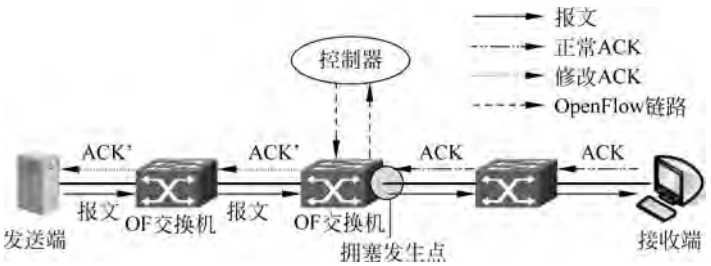


图 5.12　ARTCS 的基本架构

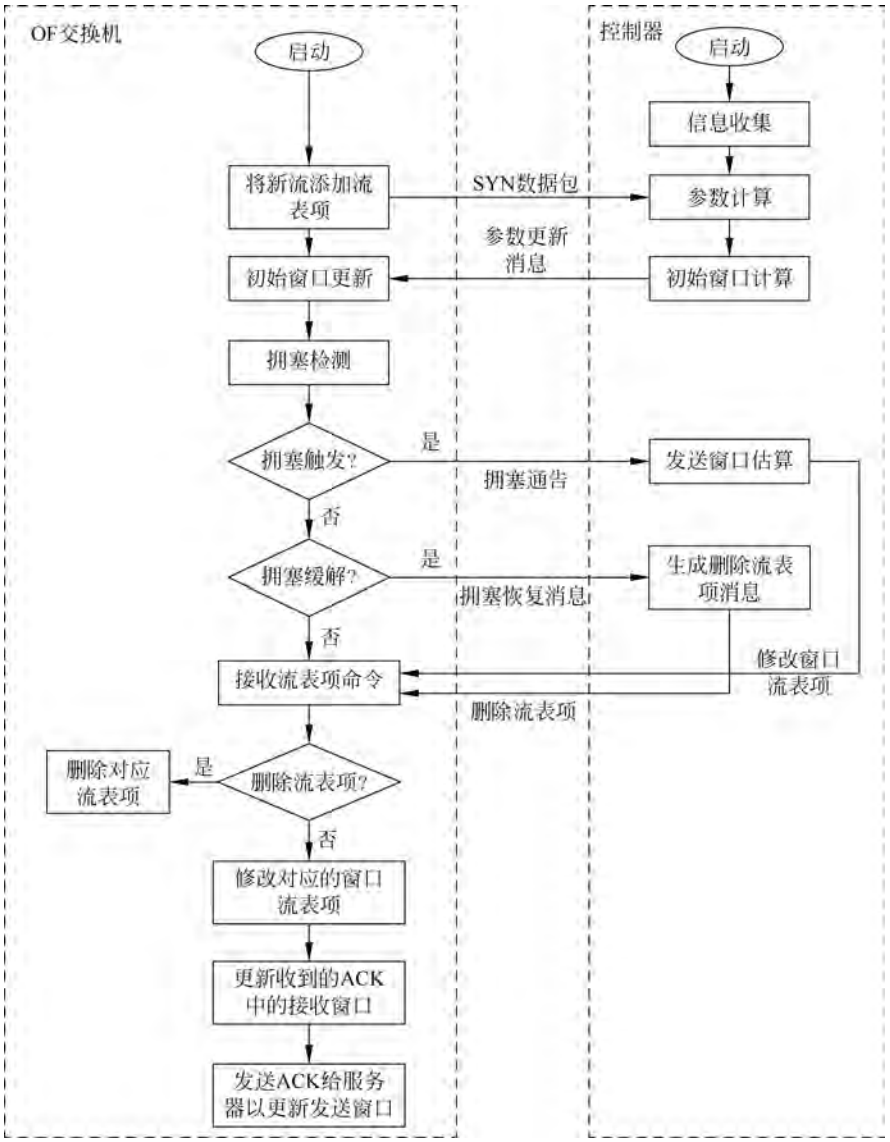


图 5.13　ARTCS 的总体工作流程

息；OF 交换机在检测到有新的 TCP 数据流到达时，将新数据流的信息添加到流表项，同时将收到的 SYN 数据包发给控制器；新的 TCP 连接建立后，控制器通过参数计算模块计算链路可用带宽以及端到端传输时延；控制器根据可用带宽等参数计算合理的 TCP 初始窗口，并将更新消息下发到 OF 交换机，OF 交换机将更新消息转发到发送方更新初始窗口值；OF 交换机通过监测端口队列长度变化进行拥塞检测，一旦发现网络拥塞则向控制器发送拥塞通告，控制器收到拥塞通告后，根据当前的拥塞程度进行发送窗口估算，并将修改窗口流表项命令下发 OF 交换机；OF 交换机收到 ACK 时便会进行窗口流表项匹配，若与流表项相符则修改 ACK 中的接收窗口值，否则保持 ACK 不变，进行正常的窗口更新；当 OF 交换机检测到拥塞缓解时，会向控制器发送拥塞恢复消息，控制器向 OF 交换机下发命令删除窗口流表项。

### 5.3.3 ARTCS 的具体方法

在 ARTCS 方法中，控制器周期性地获取 SDN 网络统计信息，并根据带宽利用率设置 TCP 连接的初始窗口值；交换机检测端口队列长度，当队列长度超过阈值时，将该交换机设置为拥塞状态，控制器计算经过拥塞交换机的所有 TCP 流的发送速率并下发流表至拥塞交换机，使交换机修改 TCP 流 ACK 报文中的接收窗口字段，降低 TCP 流的速率，达到拥塞控制的目的。控制器是数据控制逻辑的核心，它接收来自交换机的拥塞通告，并根据网络状态分配 TCP 流的速率，拥塞探测更加快速和准确，实现传输速率的自适应调整；发送端作为源主机接收控制器下发的初始窗口更新消息并更新窗口值，增强了带宽的利用率；接收端不需要进行任何修改，可以兼容现有的 TCP 协议栈，并且部署灵活方便。本节将给出每个模块的详细步骤，图 5.14 为 ARTCS 方法的详细流程图。

1. 信息收集模块

信息收集模块主要用于获取网络拓扑和端口统计信息。

(1) 控制器启动并基于链路层发现协议(link layer discovery protocol，LLDP)收集各 SDN 网络设备的连接信息(包括在每个交换机上的输入端口和输出端口)，根据 LLDP 数据包携带的交换机标识和端口号动态生成网络拓扑。

(2) 控制器周期性地下发端口统计消息到交换机，交换机将流表项计数器中记录的端口统计数据发回控制器，这些统计数据包括交换机端口收发的包数、字节数、丢包数以及统计持续的时间，用于计算带宽利用率。

(3) 控制器基于 TCP 连接的三次握手记录所有 TCP 数据流信息并生成全局数据流表 GVT。图 5.15 为基于 TCP 连接的 GVT 生成过程，建立 TCP 连接的第一次握手时，客户端发送 SYN 报文，由于交换机中没有该数据流的流表项，该交换机通过 OpenFlow 协议发送包含 SYN 报文的 OF_PacketIn 数据包到控制器，该控制器将该数据流信息添加到 GVT 中。GVT 包括数据流标识(Flow_ID)、源 IP 地址(SRC_IP)、目的 IP 地址(DES_IP)、发送数据报文的大小(size)、数据流路由信息(Flow_Path)和往返时延(RTT)。

控制器发送流表修改消息 FLOW_MOD 报文到交换机，该消息根据 OpenFlow 1.3 协议中 Nicira 私有扩展匹配域(Nicira extensible match，NXM)要求的特定匹配结构 OXM(OpenFlow extensible match)添加 TCP SYN 匹配域。OXM 的头部结构如图 5.16 所示，其中定义 OXM_CLASS 字段为 0x0001，表明这是自定义的 NXM；定义 OXM_FIELD 字段

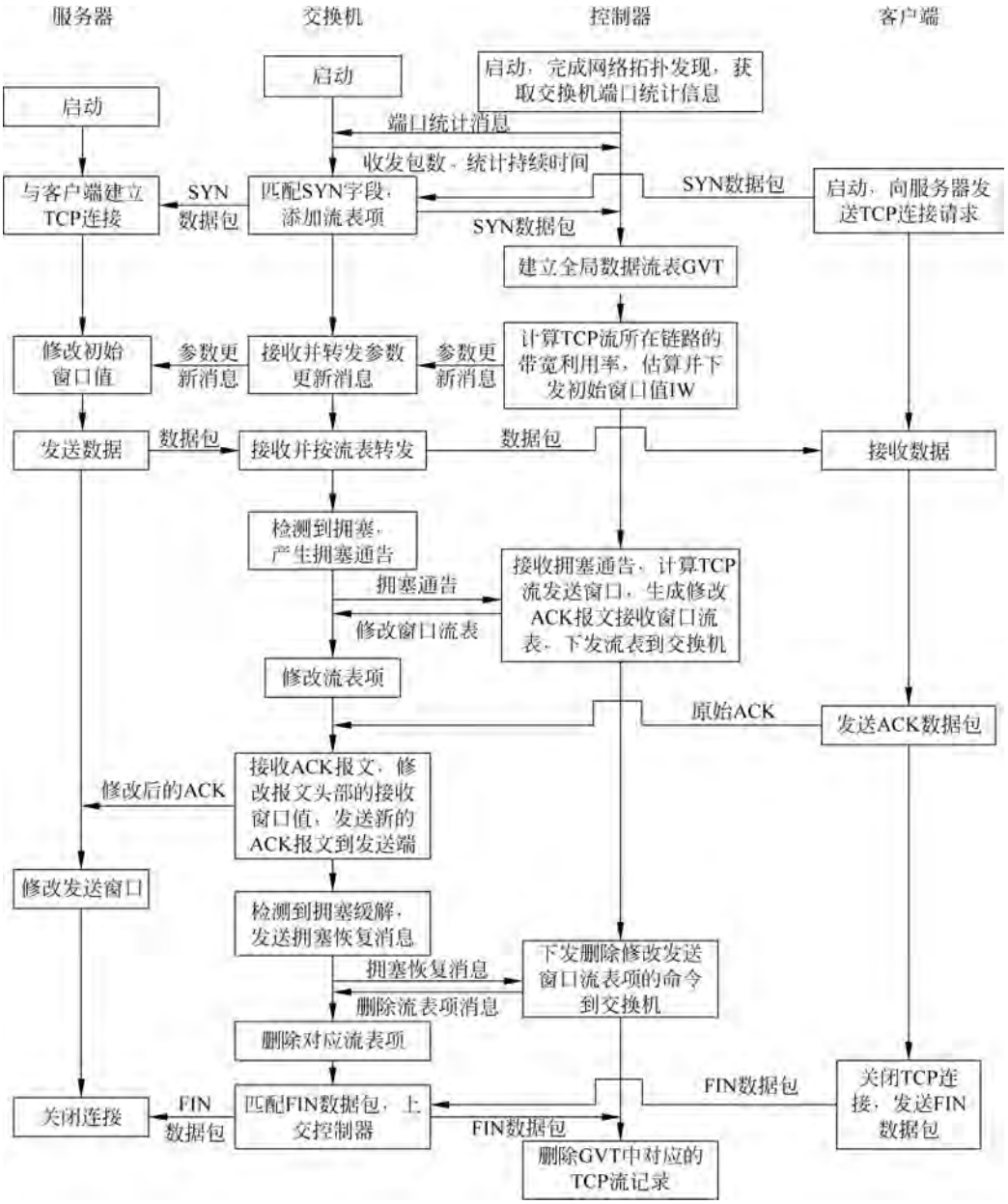


图 5.14　ARTCS 方法的详细流程图

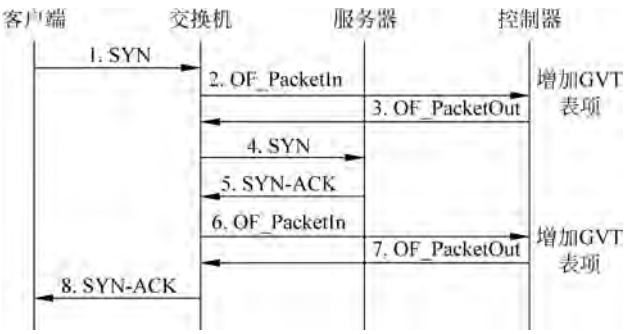


图 5.15　基于 TCP 连接的 GVT 生成过程

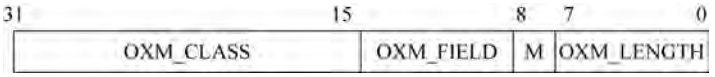


图 5.16 OXM 的头部结构

值为 TCP_SYN，表示匹配域为 TCP 头部的 SYN 选项；定义 M 字段值为 0，表示没有掩码；定义 OXM_LENGTH 字段值为 5 字节。OXM 负载 payload 中第 1 字节的值设置为 1，表示 TCP 头部的 SYN 位为 1。同时，消息下发动作指令为 Output，端口设为交换机与控制器通信的端口 $\mathrm{Port}_{\mathrm{StoC}}$。交换机收到流表修改消息后，将 TCP_SYN 字段作为 NXM 添加到该交换机的流表项中，同时添加动作指令 Output。之后交换机均会根据流表项匹配到 TCP SYN 并将数据包上交控制器，该控制器接收 SYN 数据包并记录所有 TCP 数据流信息，然后添加到 GVT 中，获得网络中所有 TCP 流的信息。

2. 参数计算模块

参数计算模块主要是根据信息收集模块中获得的数据计算一些基本参数值，包括链路可用带宽及端到端时延。

(1) 控制器根据相邻两次获取的端口发送字节数和统计持续的时间，计算每个端口的可用带宽，计算公式如下：

$$\mathrm{Port\_available\_bw}_{\mathrm{Port\_ID}}=B_{\mathrm{Port\_ID}}-\frac{\mathrm{tx\_bytes}_{i+1}-\mathrm{tx\_bytes}_{i}}{t_{i+1}-t_{i}},\quad i=1,2,\cdots$$

式中，$B_{\mathrm{Port\_ID}}$ 是端口的最大带宽，在网络部署时已知，$\mathrm{tx\_bytes}_i$ 和 $\mathrm{tx\_bytes}_{i+1}$ 分别为第 $i$ 次和第 $i+1$ 次统计时端口 Port_ID 传输的字节数，$t_i$ 和 $t_{i+1}$ 分别为从第一次统计到第 $i$ 次和第 $i+1$ 次获取端口统计信息时持续的时间。

将 TCP 流经过的所有端口可用带宽的最大值作为该 TCP 流所在链路的可用带宽，即 $\mathrm{Available\_bw}_{\mathrm{Flow\_ID}}=\max\limits_{\mathrm{Port\_ID}\in\mathrm{Flow\_ID}}\{\mathrm{Port\_available\_bw}_{\mathrm{Port\_ID}}\}$，其中 Flow_ID 表示 TCP 流的标识，Port_ID 表示端口号，Port_ID∈Flow_ID 表示流标识为 Flow_ID 的 TCP 流经过了端口号为 Port_ID 的交换机端口。

(2) 利用 OpenFlow 发现协议(OpenFlow discovery protocol，OFDP)，在类型长度值结构(type-length-value，TLV)中放入时间戳作为负载，由控制器产生带有时间戳的 OFDP 分组，并将 OFDP 分组放入 Packet_out 消息的数据部分，通过 Packet_out 命令向 TCP 流经过的所有交换机发送 OFDP 分组，命令中的操作设置为 Flood，即要求交换机向邻居交换机转发 OFDP 分组。

(3) 交换机收到转发的 OFDP 分组后，由于没有对应的流表项，因此通过 Packet_in 命令将其转发到控制器。控制器用当前系统时间减去 OFDP 分组中的时间戳可得到控制器到交换机 $S_i$、交换机 $S_i$ 到交换机 $S_j$ 以及交换机 $S_j$ 到控制器的时延，记为 $T_1$；同样，也可得到控制器到交换机 $S_j$、交换机 $S_j$ 到交换机 $S_i$ 以及交换机 $S_i$ 到控制器的时延，记为 $T_2$。

(4) 利用 ICMP 协议，通过控制器向交换机 $S_i$ 和 $S_j$ 分别发送带有时间戳的响应请求。交换机收到之后回复携带响应请求时间戳的响应回复消息。控制器将当前系统时间与响应回复分组中的时间戳相减得到对应交换机 $S_i$、$S_j$ 和控制器之间的往返时延，分别记为 $T_i$ 和 $T_j$，则交换机 $S_i$ 和 $S_j$ 之间链路的往返时延为 $L_{S_i\to S_j}=T_1+T_2-(T_i+T_j)$，路径 $R$ 上

两个交换机 $S_i$ 和 $S_k$ 之间的链路往返时延为 $L_{S_i \to S_k} = \sum_{R_{i \to k}} L_{S_i \to S_{i+1}}$，其中 $S_i$ 和 $S_{i+1}$ 表示路径 $R$ 上相邻的两个交换机。

(5) 控制器产生携带时间戳的地址解析协议(address resolution protocol，ARP)探测包，并发送到与 TCP 流两端主机 $A$ 和 $B$ 相连的交换机，由交换机转发到两个主机，主机产生携带 ARP 探测包时间戳的 ARP 回应包发回控制器，控制器用当前系统时间减去时间戳分别得到两个主机经所连交换机 $S_A$ 和 $S_B$ 到控制器的往返时延，该往返时延减去交换机 $S_A$ 和 $S_B$ 到控制器的往返时延即得到主机 $A$、$B$ 与相连的交换机 $S_A$、$S_B$ 之间的往返时延 $d_A$ 和 $d_B$，由此得到 TCP 流的往返时延 $\mathrm{RTT}_{A \to B} = L_{S_A \to S_B} + d_A + d_B$。控制器将 $\mathrm{RTT}_{A \to B}$ 计入全局数据流表 GVT 中。

3. 初始窗口计算模块

初始窗口计算模块主要根据当前的可用带宽和端到端的 RTT 值估算合理的 TCP 初始窗口值。控制器根据每条流的可用带宽和 RTT 值计算初始窗口值 IW，并通过 Packet_out 命令向连接发送端主机的交换机下发初始窗口更新消息，消息格式如图 5.17 所示，其中包含 OpenFlow 标准报文头(ofp_header)、对应 TCP 流的标识(Flow_ID)、与发送端主机相连的交换机标识(Switch_ID)、端口号(Port_ID)、初始窗口值(IW)、优先级和 cookie 字段。报文头 ofp_header 中的 type 字段设为 OFPT_IW_UDP，表示初始窗口更新消息。初始窗口值 IW 计算如下：

$$\mathrm{IW} = \max\left\{\frac{\alpha \times \mathrm{Available\_bw}_{\mathrm{Flow\_ID}} \times \mathrm{RTT}_{A \to B}}{\mathrm{MSS\_length}}, 1\mathrm{MSS}\right\}, \quad 0 < \alpha < 1$$

式中，MSS_length 表示一个数据分段的长度，1MSS 表示一个数据分段。

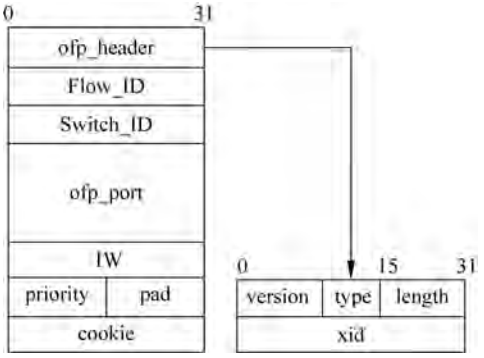


图 5.17　初始窗口更新消息格式

4. 初始窗口更新模块

交换机收到来自控制器的初始窗口更新消息后，读取消息中的端口号 Port_ID，从对应端口将该消息发送到主机。主机上的守护进程检测到初始窗口更新消息，该进程读取消息中的初始窗口值 IW，并调用 Linux 内核命令修改 TCP 初始窗口值。通过修改初始窗口值，使数据流的传输能够尽快适应当前的网络带宽，有效缩短老鼠流的传输时间。

5. 拥塞检测模块

交换机通过队尾丢弃队列管理方式实时监控 TCP 流经过的每个端口队列长度 $Q(t)$，

当交换机中端口的队列长度超过阈值 $L=Q/3$($Q$ 为队列缓存最大值)时,交换机产生拥塞通告消息(congestion notification message,CNM),并通过 Packet_in 命令将其发送到控制器。消息结构如图 5.18 所示,其中包含 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_buff,优先级 priority 和 cookie 字段。报文头 ofp_header 中的 type 字段设为 OFPT_BUF_CN,表示拥塞触发消息。交换机收到拥塞通告消息后则进入拥塞状态,并周期性地监视队列长度。

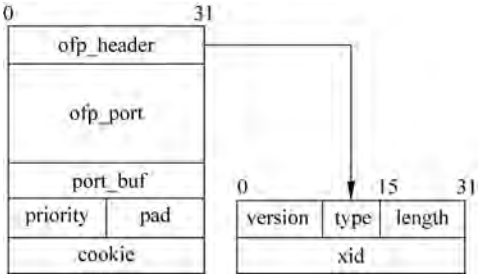


图 5.18 拥塞通告消息格式

6. 发送窗口估算模块

控制器接收来自交换机的拥塞通告,从通告中获取交换机的队列长度 $Q(t)$,估算经过交换机的 TCP 流的发送窗口大小,TCP 流 $i$ 发送窗口大小的估算公式如下:

$$\mathrm{SWND}_i=\frac{\mathrm{BW}\times\mathrm{RTT}_{\mathrm{avg}}}{n}$$

式中,$n$ 是经过拥塞端口的 TCP 流的总数,可由全局数据流表获得,BW 是交换机拥塞端口的最大带宽,在网络部署时就已知,$\mathrm{RTT}_{\mathrm{avg}}$ 是所有经过拥塞端口的 TCP 流的平均 RTT,即 $\mathrm{RTT}_{\mathrm{avg}}=\dfrac{\sum_{\text{拥塞端口在}A、B\text{路径上}}\mathrm{RTT}_{A\to B}}{n}$。

控制器估算出发送窗口大小后,根据队列长度 $Q(t)$来表征交换机的拥塞程度,计算经过交换机的所有 TCP 流的 ACK 报文中的接收窗口大小。ACK 报文中接收窗口的大小计算如下:

设置阈值 $Q_{\mathrm{L}}=Q/2$ 和 $Q_{\mathrm{H}}=(4/5)Q$,即满足 $L<Q_{\mathrm{L}}<Q_{\mathrm{H}}$,利用上述两个阈值检测交换机的拥塞程度,根据拥塞程度的不同,设置不同的接收窗口大小:

(1) $L\leqslant Q(t)<Q_{\mathrm{L}}$:轻度拥塞,设置 TCP 流 $i$ 的接收窗口

$$W_r(i)=\max\left\{\frac{4}{5}\mathrm{SWND}_i,1\mathrm{MSS}\right\}$$

(2) $Q_{\mathrm{L}}\leqslant Q(t)<Q_{\mathrm{H}}$:持续拥塞,设置 TCP 流 $i$ 的接收窗口

$$W_r(i)=\max\left\{\frac{1}{2}\mathrm{SWND}_i,1\mathrm{MSS}\right\}$$

(3) $Q(t)\geqslant Q_{\mathrm{H}}$:严重拥塞,设置 TCP 流 $i$ 中 ACK 报文的接收窗口

$$W_r(i)=1\mathrm{MSS}$$

式中,$\mathrm{SWND}_i$ 为 TCP 流 $i$ 当前发送窗口大小。

7. 发送窗口更新模块

控制器生成并发送流表修改消息 Flow_Mod 报文到交换机，修改 ACK 报文接收窗口的流表。在 OXM 结构中添加 TCP ACK 匹配域，OXM 的头部结构如图 5.16 所示，其中定义 OXM_CLASS 字段为 0x0001，表明这是自定义的 NXM；定义 OXM_FIELD 字段值为 TCP_ACK，表示匹配域为 TCP 头部的 ACK 选项；定义 M 字段值为 0，表示没有掩码；定义 OXM_LENGTH 字段值为 9 字节。OXM 负载 payload 中第 1 字节的值设置为 1，表示 TCP 头部的 ACK 位为 1。在 OXM 负载中第 2～5 字节设置接收窗口值 $W_r(i)$。另外，在流表修改消息中扩展动作集合，根据 OpenFlow 1.3 协议，在 Instruction 结构中，将 type 字段设为 OFPT_APPLY_ACTIONS，在动作集合 OFPT_SET_FIELD 中添加修改接收窗口(MOD_WINDOW)的新动作。

当交换机收到来自控制器的修改窗口流表消息，将 TCP_ACK 字段作为 NXM 添加到该交换机的流表项中，同时通过 Write_Action 指令添加动作 MOD_WINDOW。之后交换机均会根据流表项匹配到 TCP ACK 报文并根据动作指令 MOD_WINDOW 将 TCP ACK 报文中的接收窗口值字段修改为 $RRWND(i)' = \min(W_r(i), RWND(i))$，其中 $RWND(i)$ 为数据流 $i$ 的 ACK 报文中原有的接收窗口字段的大小。ACK 报文中接收窗口的调整促使发送方调整发送窗口 $W_s = \min\{W_c, W_r\}$，因此可有效缓解拥塞，提高数据流的传输效率。

8. 拥塞恢复模块

拥塞恢复模块主要用来处理网络拥塞缓解时的情况。交换机持续监视队列长度，只要每个周期(设为流经交换机的所有 TCP 流的 RTT 平均值 $RTT_{avg}$)监视队列的长度仍超过阈值 $L$，则持续向控制器发送拥塞通告。当连续三个周期监视的队列长度均小于阈值 $L$，交换机向控制器发送拥塞恢复消息，该消息的结构与图 5.18 所示的拥塞通告消息类似，只需将报文头 ofp_header 中的 type 字段设为 OFPT_BUF_CR，表示拥塞恢复消息。控制器收到拥塞恢复消息后，向交换机下发 Flow-Mod 消息，将该消息中的 command 字段设为 OFPC_DELETE_STRICT。交换机收到上述消息后，删除修改窗口流表项，并恢复正常状态。

当 TCP 连接关闭时，交换机第一次接收到客户端或服务器发送的 TCP FIN 数据包后，解析包头，将解析包头后的数据包上交控制器，该控制器从全局数据流表 GVT 中将对应的 TCP 流记录删除，并发送流表修改消息到交换机，该流表修改消息在 OXM 结构中添加 TCP FIN 匹配域。与之前的流表消息类似，OXM 的头部结构如图 5.16 所示，其中定义 OXM_CLASS 字段为 0x0001，表明此为自定义的 NXM；定义 OXM_FIELD 字段值为 TCP_FIN，表示匹配域为 TCP 头部的 FIN 选项；定义 M 字段值为 0，表示没有掩码；定义 OXM_LENGTH 字段值为 5 字节。OXM 负载 payload 中第 1 字节的值设置为 1，表示 TCP 头部的 FIN 位为 1。同时，消息下发动作指令为 Output，端口设为交换机与控制器通信的端口 $Port_{StoC}$。交换机收到流表修改消息后，将 TCP_FIN 字段作为 NXM 添加到该交换机的流表项中，同时添加动作指令 Output。之后交换机均会根据流表项匹配到 TCP FIN 并将数据包上交控制器，该控制器接收 FIN 数据包并从 GVT 表中将对应的 TCP 流记录删除。

## 5.4 本章小结

Incast 拥塞是数据中心网络拥塞的主要形式。现有的解决方法要么难以部署，要么难以及时检测和响应网络拥塞，因此，它们不能同时实现高带宽利用率和低排队时延。

本章从两方面解决上述问题，一方面，改进 DCTCP 协议并提出了 FDCTCP 协议，利用 ECN 来推断网络拥塞并计算拥塞因子。在接收到带有 ECN 标记的 ACK 时，FDCTCP 使用拥塞检测方法来估计网络拥塞程度和趋势。然后 FDCTCP 使用快速拥塞缓解机制，根据拥塞程度来减少拥塞窗口。若随后接收到的 ACK 没有 ECN 标记，FDCTCP 则使用快速恢复机制将拥塞窗口恢复到 Incast 发生前的数值。仿真结果表明，在服务器数量和缓冲区大小不同的情况下，FDCTCP 在数据流完成时间、吞吐率和队列长度方面都比 DCTCP 有显著的性能提升。

另一方面，提出一种基于 SDN 网络的自适应可靠数据传输机制 ARTCS，在连接建立时，根据 SDN 控制器获得的网络状态统计信息，设置 TCP 流的初始传输窗口，有效减少数据中心网络中老鼠流的传输时间。传输过程中自动检测拥塞，并根据网络拥塞程度调整 TCP 流传输速率，有效缓解拥塞，提高网络带宽利用率，实现数据中心网络中数据的高效传输。该机制针对当前网络状态和数据中心网络的数据特性，增强数据传输对网络状态的自适应性，在兼顾当前的网络状态和数据中心网络的数据特性的同时，优化传输过程中的传输速率。

## 参考文献

[1] A1izadeh M，Greenberg A，Maltz D，et a1. Data center TCP(DCTCP). ACM SIGCOMM ComDuter Communication Review，2010，40(4)：63-74.

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

[3] 王娟. 数据中心网络"多对一"流量模式的高吞吐率传输. 四川师范大学硕士学位论文，2020.

[4] 2020 年中国数据中心行业市场现状及发展趋势分析——工业计算将成为行业新发展动力. https://www. sohu. com/a/408012332_99922905，2021-7-28.

[5] Bari M F，Boutaba R，Esteves R，et al. Data center network virtualization：a survey. IEEE Commun Surv. Tutorials，2013，15(2)：909-928.

[6] Zhang Y，Ansari N. On architecture design，congestion notification，TCP incast and power consumption in data centers. IEEE Commun. Surv. Tutorials，2013，15(1)：39-64.

[7] Xie D，Ding N，Hu Y C，et al. The only constant is change：incorporating time-varying network reservations in data centers. In Proc. of the ACM SIGCOMM 2012 Conference on Applications，Technologies，Architectures，and Protocols for Computer Communication，SIGCOMM 12. Helsinki，Finland，2012：13-17.

[8] Guo C，Yuan L，Xiang D，et al. Pingmesh：a large-scale system for data center network latency measurement and analysis. In Proc. of SIGCOMM 15，London，United Kingdom，2015.

[9] Ren Y，Zhao Y，Liu P，et al. A survey on TCP Incast in data centernetworks. Int. J. Commun. Syst.，2014，27(8)：1160-1172.

[10] Benson T，Akella A ，Maltz D A. Network traffic characteristics of data centers in the wild. In Proc. of

the 10th ACM SIGCOMM Conference on Internet Measurement, Melbourne, Australia, 2010.

[11] Monsanto C, Reich J, Foster N, et al. Composing software defined networks. In Proc. of 10th USENIX Symposium on Networked Systems Design and Implementation, Lombard, IL, United States, 2013: 1-13.

[12] 毛健彪. Open vSwitch 流表查找分析. https://www.sdnlab.com/15713.html, 2021-8-6.

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

[14] He K, Khalid J, Gember-Jacobson A, et al. Measuring control plane latency in SDN-enabled switches. In Proc. of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, New York, 2015: 1-6.