西安电子科技大学

硕士学位论文

队列调度算法的研究与逻辑实现

姓名：赖金志

申请学位级别：硕士

专业：微电子学与固体电子学

指导教师：蔡觉平

20090101

摘 要

在 Internet 发展初期，网络中的业务主要是 FTP、WWW、E-mail 等基于数据传输的应用，它们并不关心服务质量。但随着 Internet 技术的发展，网络规模和用户数不断扩大，一些新业务如视频点播、IP 电话等也要通过 Internet 得以实现，它们需要相对较小的端到端时延、时延抖动和较大的带宽，传统的 Internet 已不能满足需求。网络通讯量的日益迅猛地增长，用户对网络带宽的要求不断上升，远远超出了现有的连接与转发容量。当前网络设备的发展方向是研究如何在高负载情况下实现高速转发。对现有的网络设备进行硬件升级是很好的选择，但是也带来了成本的增长。为了在现有条件下提高设备的性能，大量的工作集中在调度算法的研究上。

本文首先对队列调度技术的相关内容进行了较为详细的介绍，包括队列调度算法的技术指标和常见队列调度算法的分类、分析和比较等。然后着重说明了 WFQ 调度算法的逻辑实现，并搭建基于 Vera 的验证环境，对 WFQ 设计进行充分验证。最后，对本文所做的工作加以总结，并提出今后努力的方向。

关键词：WFQ　调度算法　VERA　验证

ABSTRACT

At the beginning of developing of Internet, because the main services such as the FTP, WWW, E-mail are based on the data transmission, it is not important for them to afford Quality of Service (QoS). Now with the developing of Internet technology, increasing of the scale of network and the amount of users, the traditional Internet can not satisfy the need of users. New services such as VOD, IP phone and so on, which need the relatively short end-to-end delay, delay jittering, and wider bandwidth, must be realized through Internet. Nowadays, the network equipments such as routers and switches are required to work faster to achieve a high throughput. Hardware upgrading sounds like a good idea, but it also makes these equipments more expensive. Enhancing the performance of current equipments is acceptable. In this situation, the studying of queue scheduling algorithms is becoming a hot topic at recent research in this field.

Correlation study contents of queue scheduling scheme are detailed in this paper firstly, including technical targets of queue scheduling algorithm, as well as classification, analysis and comparison of familiar queue scheduling algorithm. And then, the design of Weighted Fair Queuing (WFQ) scheduling algorithm is explained. The verification environment is built based on Vera and sufficient verification of WFQ algorithm is implemented after WFQ's design. At last, this paper summarizes the research work and put forwards of striving to the research in the future.

Keyword: WFQ Scheduling algorithm Vera Verification

西安电子科技大学

学位论文独创性（或创新性）声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切的法律责任。

本人签名：赖金志　　日期 2009.1.12

西安电子科技大学

关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律署名单位为西安电子科技大学。

（保密的论文在解密后遵守此规定）

本学位论文属于保密，在___年解密后适用本授权书。

本人签名：赖金志　　日期 2009.1.12

导师签名：蔡益朝　　日期 2009年1月12日

第一章 绪 论

1.1 网络 QOS 综述

随着 Internet 的日益全球化和信息技术的飞速发展，计算机网络的应用领域和网络规模都呈爆炸式增长。人们对信息的需求已不满足于传统的电话电报业务以及传统的文件传输、电子邮件等数据业务，而是追求更高品质的集视频、图象、声音、文字、甚至动画等于一体的多媒体宽带应用服务。这些应用不仅要有带宽保证，而且还需要时延、时延抖动、分组丢失率的限制，使得传统网络服务设备已成为网络发展的瓶颈。网络服务质量 (Quality of Service，简称 QoS)是互连网络所必须考虑的一个重要的问题，网络的流量控制和拥塞处理都是在网络服务设备中得以实现。因此，设计高性能的连接骨干网的网络设备也就显得尤其重要。

1.2 队列调度算法的重要性与研究意义

网络 QoS 控制的本质在于资源的管理，即控制缓冲队列、链路带宽等网络资源的分配和使用。队列管理在网络传输控制中发挥着相当大的作用，是网络 QoS 控制的核心技术之一，也是实现网络拥塞控制的重要手段。控制算法的设计需要满足一定的性能要求，而系统性能的好坏是针对一定的性能评价标准而言的，应用不同的评价标准往往会导致不同的评价结果和 QoS 控制方案设计。不论在综合服务[1]（Integrated Service，简称 IntServ）还是在区分服务[2]（Differentiated Service，简称 DiffServ）里，都涉及队列调度的问题。简而言之，队列调度的功能就是如何从一个或多个队列中选择下一个将要转发的分组，这与队列管理机制有着本质的区别。一个有效的队列调度算法应达到的性能指标有公平性、时延特性、对恶意业务流的隔离能力、链路带宽的利用率、复杂性等，前四个指标与 QoS 密切相关。在队列调度算法研究中，如何提供较好的公平性、时延特性，同时算法复杂度较低且易于实现的特性成为人们关心的焦点，所以开展对这一方面的研究具有较高的研究价值和实际意义。

众所周知，服务质量已成为当今网络发展的一个重要课题，网络中的服务是面向不同的用户的，提供包括及时性、准确性和高效率几个方面的服务质量，然而传统的尽力而为型的网络不能够很好提供这些方面的服务，所以对于如何有效的提高和改善网络中的服务质量这一问题，各种方面的解决方案层出不穷，如流

分类、链路技术、协议等等。在这其中，比较行之有效且被大家所公认的一种解决方式就是调度算法。

调度算法决定了被服务的分组接受服务的顺序，使其所在的数据转发设备能够智能地控制数据转发的优先级，每条链路的带宽以及平均时延等特性以满足一定程度上的服务质量。调度算法的研究由来已久，随着网络规模的膨胀以及人们对业务种类和质量要求的上升，调度算法已经成为众多网络工程师所面临的一个重要难题。当前调度算法的发展的趋势基本分为三种：分组公平排队类调度算法、轮循类调度算法和服务曲线类调度算法[3]。公平类调度算法最早来源于对 GPS[4]（Generalize Processor Sharing 通用处理机共享）流体模型的研究，其后衍生出多个不同的算法，其中以 WFQ（Weighted Fair Queuing 加权公平队列算法）最具有代表性，也包括了基于 WFQ 的若干种减少算法复杂度的方法，这些方法有的已经被成功得用于网络设备中。轮循类算法是一类比较简单的算法，虽然公平性不如 WFQ 算法，但是算法的简单性却是其在硬件实现上的最大优势所在，具有代表性的算法有 WRR（Weighted Round Robin 加权轮循算法）、DRR（Deficit Round Robin 差额轮循算法）等。服务曲线类算法又是另一种与前两者完全不同的算法，它从去耦时延与带宽的角度出发，解决了如何同时获得低时延和低带宽的问题。

通信网络正朝着宽带化、综合化和数字化的方向发展，链路的速率越来越快，网络中需要承载的业务也呈现多样化，包括语音、数据、图像，而且不同的业务有不同的要求，这就要求网络节点能对其输入业务进行快速、有效和合理的调度，以尽量满足不同业务的不同需求。

调度是系统资源管理的核心机制之一，是解决多个业务竞争共享资源问题的有效手段。管理系统资源大致包含三个部分：缓冲区、链路带宽、处理机资源。对队列调度的研究主要从以下两个方面来研究：

（1）缓冲队列管理

在缓冲管理中，如何设置合适的队列缓存空间容量，如何在网络运行期间合理地控制队列长度的动态变化，以平衡系统吞吐量和分组排队延迟之间的矛盾，是缓冲管理乃至整个网络控制需要解决的重要问题。缓冲管理的研究方向有：

- 基于流量预测提高系统资源利用率；
- 与分组调度相结合融入带宽分配；
- 队列长度的控制与维护；

（2）分组调度

分组调度是指按照一定的规则来决定从等待队列中选择哪个分组进行发送，使得所有输入业务流能够按照预定的方式共享输出链路带宽。分组调度的算法很多，但基本过程都是根据一定的系统信息做出判断，控制各个队列占用链路带宽的频率，进而影响时延、丢失等服务质量。一般来说，考虑的信息越多，算法越

有效，复杂性也就越高。先进的队列调度算法都是把分组存到不同的队列里，然后为其计算一个动态优先级作为控制参数，根据这些参数进行调整。未来网络有两个发展趋势带宽高速化和业务多样化。分组调度算法的研究要适合这些发展趋势，既要保证高的分组调度速率，又要提供时延等服务质量和公平性保证，即在追求简单性和易实现性的同时，考虑算法的综合性能。

1.3论文内容安排

第一章为绪论部分，主要介绍了现代网络对 QoS 的要求以及队列调度算法对 QoS 的重要性和研究队列调度算法的意义。

第二章说明调度结构的相关理论，通过对输入排队、输出排队和输入输出排队调度结构的比较来分析它们之间的优缺点。

第三章介绍了队列调度算法的相关概念及算法原理，对网络设备中常用的调度算法进行比较分析。

第四章为 WFQ 算法的设计与逻辑实现，介绍了 WFQ 算法的基本原理，以四输入 WFQ 电路为例详细介绍了整个模块的组成结构以及实现过程，并对各子模块的电路设计实现做了详尽的阐述。

第五章详细介绍了 WFQ 模块的验证过程，主要包括对 WFQ 模块基于 Vera 的验证环境的搭建、测试点分解、测试向量执行等的描述，对 WFQ 模块的功能和性能进行了充分的验证。

最后是全文的总结、致谢以及参考文献。

第二章 调度结构的基本理论

2.1 分组调度结构

随着网络应用的迅速发展，其规模越来越庞大，结构日趋复杂，仅仅依靠端到端的拥塞控制是不够的，网络本身也必须参与资源的控制和管理。当正在发生的通信量小于所提供的带宽时，不存在拥塞问题；反之，当前通信量大于所提供的带宽时，就会造成拥塞。如果网络的处理能力不足的话，则出现时延抖动、分组丢失等问题而达不到我们所预期的服务质量。排队和调度算法就是为了解决在网络拥塞的时候，如何更好的提高 QoS 的问题。

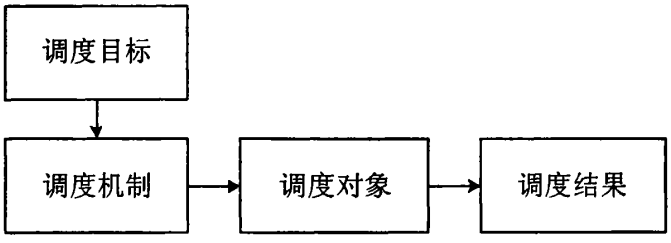

图2.1 分组调度结构

一个完整的调度系统一般包括四个部分：被调度对象、调度目标、调度机制和调度结果，如图 2.1 所示。其中调度目标是要求对调度对象实现怎样的预期目标；调度对象就是这里所说的分组队列；调度机制就是调度算法或是调度规则，实际上也是一个最优化的问题；调度结果就是调度机制施加于调度对象后得到的一定输出结果。本文中主要讨论定长分组调度，调度对象就是分组队列。在网络调度系统中，常用的与调度算法相关的概念如下：

（1）阻塞（Blocking）：

当两个以上报文在同一时隙指向同一输出端口时，就会出现输出阻塞，所以必须放置缓冲器以增加报文时延为代价来消除阻塞。在传统的输入缓冲结构中，在一个报文时隙内当 FIFO 队列的排头报文参与输出竞争失败后，即使排在它后面的报文指向的输出端口是空闲的，该报文也无法输出，这种现象称为队列头阻塞。队列头阻塞可以通过虚拟输出队列(Virtual Output Queue, 简称 VOQ)加以解决。

（2）加速比（Speedup）：

网络结构内部的速率和端口速率的比值称为加速比。如果每个输出端口在一

个时隙内可同时接收 S 个报文，就说这种结构具有加速功能，S 称为加速因子。可通过时分和空分两种方法实现加速，前者指结构的内部速率是外部端口的 S 倍，后者则是为每个输出端口提供 S 条并行路径。

（3）性能（Performance）：

通常使用三个参数描述报文转发结构的性能：吞吐率 TP（ThroughPut）、时延 D（Delay）和报文丢失率 CL（CellLoss）。

TP 定义为每输入端口、每时隙成功传送报文到目的地端口的平均数，最大吞吐率（TPm）则是最大负荷下的 TP 值。

D 指的是一个报文从进入输入端口到传送到输出端口所经历的平均时间，它由缓冲器平均排队时延和一个报文的传输时间组成。

CL 定义为报文丢失的数目与总到达数目之比，报文丢失是由于阻塞或缓冲器溢出造成的，显然降低 CL 的一个直接办法就是增加缓冲器的容量。

此外，延时抖动也是描述网络转发结构性能的一个重要指标，它指的是一个报文在网络转发结构中经历的最大时延和平均时延的差值，通常用差值出现的概率表示该指标对于实时业务有重要影响。不难看出，一个理想的调度结构应具有尽可能大的 TP、较小的时延、报文丢失率和延时抖动。

（4）优先级（Priority）：

不同的业务在传送时延和丢失率业务上有不同的要求，如像语音这样的实时业务要求有较小的传送延时，像数据这样的非实时业务则要求有较小的丢失率，而像高质量视频业务则对时延和丢失率均有较高的要求。多媒体业务的不断引入要求调度结构应能处理不同的优先级别以保证不同业务的 QoS。

2.2 分组调度的排队策略

目前的分组调度策略都是采用多队列排队机制，这就涉及到对多个业务流进行有区别的服务。在 IP 网络中，可以基于 IP 源/目标地址、传输层/源/目的端口和协议类型对输入业务进行分类，也可以根据 IP 报头中的 ToS（Type of Service）字段进行分类。分类的结果使不同队列能满足不同的 QoS 要求。

队列管理也是一个热门研究领域。目前计算机网络使用的队列管理策略是一种很简单的去尾 (Drop-tail)算法，但存在死锁、满队列和全局同步等问题。当前的队列管理算法可以分为两大类：被动式队列管理(Passive Queue Management，简称 PQM)和主动式队列管理(Active Queue Management，简称 AQM)[6]。其中 AQM 是 IETF 推荐的拥塞控制的关键技术，比较成熟的 AQM 有随机早期检测算法(Random Early Detection，简称 RED)、BLUE、GREEN 及其衍生算法[7]。比较简

单的队列管理是采用先进先出（First In First Out，简称 FIFO）进行排队，图 2.2 为原始 FIFO 排队队列结构图。

一般不把 FIFO 看成 QoS 队列，在一定程度上，FIFO 还是提供了某些拥塞管理特性。对于 FIFO 通俗的理解就是增加了一个缓存队列，当到达接口的报文因为拥塞无法在第一时间得到发送的时候，先存入 FIFO 队列中（前提是队列未满），然后在接口有发送能力的时候按照进入队列的先后顺序（也就是到达接口的顺序）进行发送。这样实际上为数据提供了一个缓存空间，在一定程度上实现了拥塞管理。

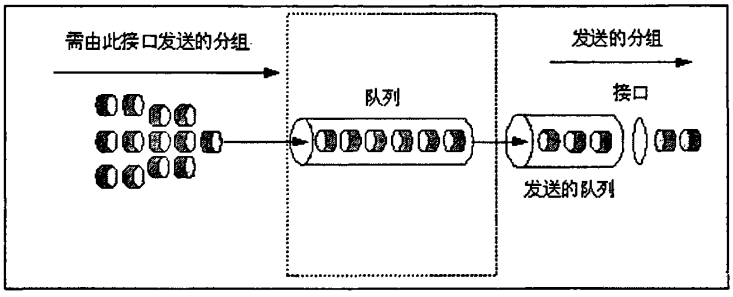

图2.2 原始 FIFO 排队队列结构图

FIFO 队列实现简单，因为没有考虑复杂的数据分类，也没有复杂的调度机制，所以在处理数据的速度比其他队列机制更快，这样就保证了对时延的要求。FIFO 队列的关键在于队列的长度：队列长度设置较长时，可以提供更大的缓存空间，有助于减少报文的丢弃，但同时也降低了时延的保证，进而降低了对抖动的保证；选择较短的队列时，提高了对时延的保证，但增加了报文丢弃的可能。

FIFO 的优点很明显：实现简单，处理速度很快，提高对时延的保证。FIFO 简单设计的代价就是作为 QoS 队列机制，它无法区分各种业务数据的优先程度，从而无法保证关键业务数据类型。

作为解决多个业务竞争共享资源问题的有效手段，分组调度的作用是决定下一时刻发送哪一个分组，在研究分组调度算法的过程中，我们将假设排队系统已经有了一个很理想的分类器，采用的队列管理算法可以是 Drop-tail 也可以是 PQM/AQM 的一种。对于流量到达也采用了漏桶或令牌漏桶等措施来进行整形以避免大量的分组重传。

分组调度主要是对链路带宽的管理，按照一定的规则来决定从哪个分组进行发送，使得所有输入业务流能够按照预定的方式共享输出链路带宽。分组调度的目的是使多条流以较高的效率使用资源，影响的主要性能参数包括带宽分配、时延和时延抖动等。

分组调度算法的目的在于提供不同等级的 QoS 满足不同业务流的带宽、延迟等要求。概括地说，算法性能的好坏主要涉及到时延性能、公平性和复杂性[8] [9]

这三个因素：

（1） 时延性能

分组调度算法应为不同的业务流提供端到端的时延保证，并且只与此业务流的某些参数(如带宽需求等)有关，而与其它业务流无关。

（2） 公平性

公平性是一个非常重要的因素。由于多条流共享资源，既要保证每条流的 QoS 需求又要尽量使各条流之间不受彼此影响，保证流的分割性，这样可以避免恶意攻击者占用大量的共用信道。关于公平性的定义主要有三种：①服务公平指数 SFI（Service Fairness Index）[10][11]、最坏公平指数 WFI(Worst case Fairness Index)[12]；②Raj Jain 公平指数[13]；③比例公平原则[14]。

（3） 复杂性和可扩展性

复杂性直接决定算法能否实现。速度的不断提高和规模的不断扩大是网络发展的一个趋势。在高速网络中，调度结构必须能够在很短的时间内完成调度转发，这就要求调度算法在调度计算过程中应该尽量简单。此外，当业务流数量增加和链路速率变化范围较大时，调度算法仍能有效的工作，即要求算法具有良好的可扩展性。设计一个有效的算法并降低实现的复杂性是调度研究领域追求的目标。

2.2.1 输入排队调度结构

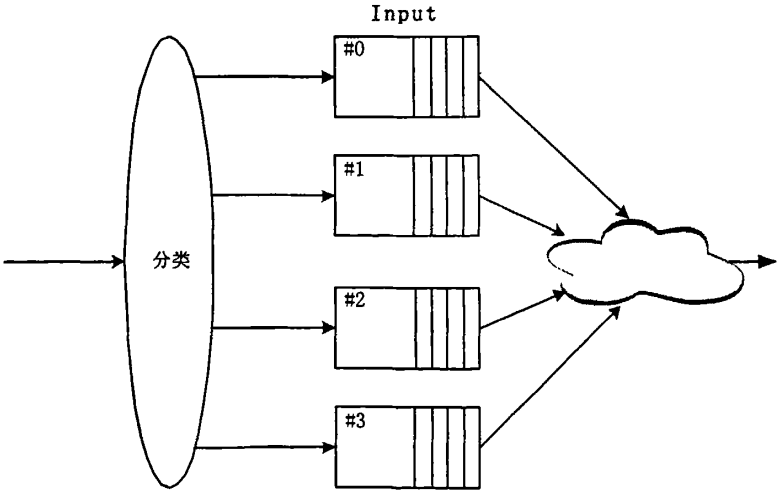

图2.3 输入排队结构

输入排队（Input Queuing，简称 IQ）如图 2.3 所示。IQ 排队调度结构将到达的报文首先保存在输入端口的缓冲区中，然后由调度算法决定报文何时传送到输出端口，如图 2.3 所示。在每个时隙一个输入端口最多只能有一个报文被发送，

而一个输出端口也最多只能接收一个报文。由于发送到输出端口的报文速率和到达输入端口的报文速率相同，因此这种结构的加速比为 1。IQ 对内部带宽和存储器访问速率的要求较为宽松，只需要达到其链路速率即可，因而可扩展性较好。

输入排队调度结构的主要问题就是存在队头阻塞（Head of Line Blocking，简称 HOL 阻塞）现象。由于每个时隙到达时，只有在缓存区队头位置的报文才能参与对输出端口的竞争，且每个输出端口最多只能接收一个报文，那么位于输入缓存队列 HOL 之后的报文由于 FIFO 缓冲区 HOL 报文的阻塞，即使当前时隙该报文对应的输出端口处于空闲状态，也无法在此时隙被发送。显而易见，在 FIFO 队列中，当队列的头报文受阻时，跟在其后的所有准备发送的报文也将受阻。这种现象就称为队头阻塞，此时系统吞度率只能达到 58.6%[15]。如果在每一个输入端口采用适当的缓冲策略和调度算法，HOL 阻塞是能够被完全消除的，也就是在输入端口采用虚拟输出队列[16]（Virtual Output Queue，简称 VOQ）技术对输入的报文进行排队。

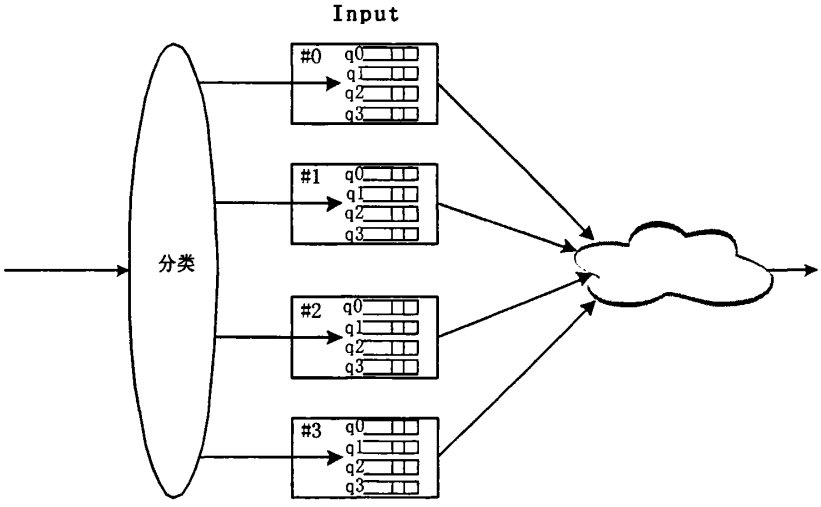

图2.4 VOQ 缓存排队结构

VOQ 缓存排队调度结构是在输入端按照报文的目的端口来组织队列。图 2.4 是基于 VOQ 缓存的排队结构图，VOQ 结构在每个输入端口都设置了 N 个缓存队列，分别对应于 N 个输出端口。如图 2.4 所示#0 号输入端口中分成 4 个 FIFO 队列分别对应 4 个输出端口，#0～3 号输入端口中每个输入端口都有 4 个 FIFO 队列对应 4 个输出端口，经过 VOQ 排队后，同一缓存队列中的报文具有相同的输出端口，不同缓存队列的报文对应不同的输出端口。对于 N 个端口的系统，需要 N ×N 个缓存队列。如果每个端口又有不同的优先级，那么每个输出端口队列还可以细化成由数个优先级队列组成。那么不同的优先级报文就根据目的端口进入不同的队列中排队。由于在每个时隙每个输入端口输出到任一输出端口的报文都可

以参与输出竞争，因此可以完全消除 HOL 阻塞，吞吐量从 58.6%提高到 100%[17]。同时，在同一个缓存队列中遵循 FIFO 原则，保证报文传输的顺序性，而在每个时隙哪个队列的报文被发送由调度算法决定。

2.2.2 输出排队调度结构

输出排队（Ouput Queuing，简称 OQ）如图 2.5 所示，在这种结构中，只要报文一到达输入端口立即就发送到输出端口。

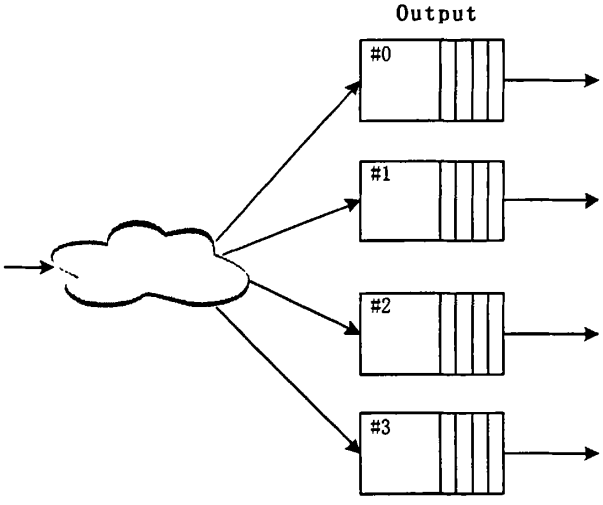

图2.5 输出排队结构

这种排队结构对吞吐量和时延特性有利，但是所有到达报文同时传送到输出端口需要太多的内部连接带宽和存储器带宽。由于输入端没有缓存，输入端口接收到报文以后，立即发送到相应的输出端口，因此输出排队排队结构具有最低的传送时延，这对设备提供了良好的 QoS 保证，仅需在线路输出调度时才对 IP 分组实施 QoS 调度。

输出排队调度结构没有队头阻塞，其吞吐率高于输入排队调度结构，理想的输出排队调度结构吞吐率可以达到 100%。但是由于在同一时隙可能有多个报文同时到达一个输出端口，当有 N 个输入端口时，就有可能 N 个报文同时到达任意一个输出端口。要在一个时隙里接收 N 个报文，输出缓存区队列的存储器必须能在一个时隙里完成 N 次写入，这使得输出排队队列更容易发生溢出而造成报文丢弃，导致高的报文丢失率，这在突发业务流下尤其严重。这就要求输出端口需要运行在更高的线路速率。如果有 N 个输出端口，那么输出排队调度结构要求调度结构和 OQ 必须以外部线路速率的 N 倍运行。

输出排队可以实现 100%的吞吐率，但需要 N 倍的加速比，当端口速率达到 G bit/s 时，对硬件实现要求较高，在现有的工艺水平下难以实现。从原理上讲，输出排队是一种理想的排队结构，它一般作为其他排队结构性能评估的基准，作为参照物存在。

2.2.3 组合输入输出排队结构

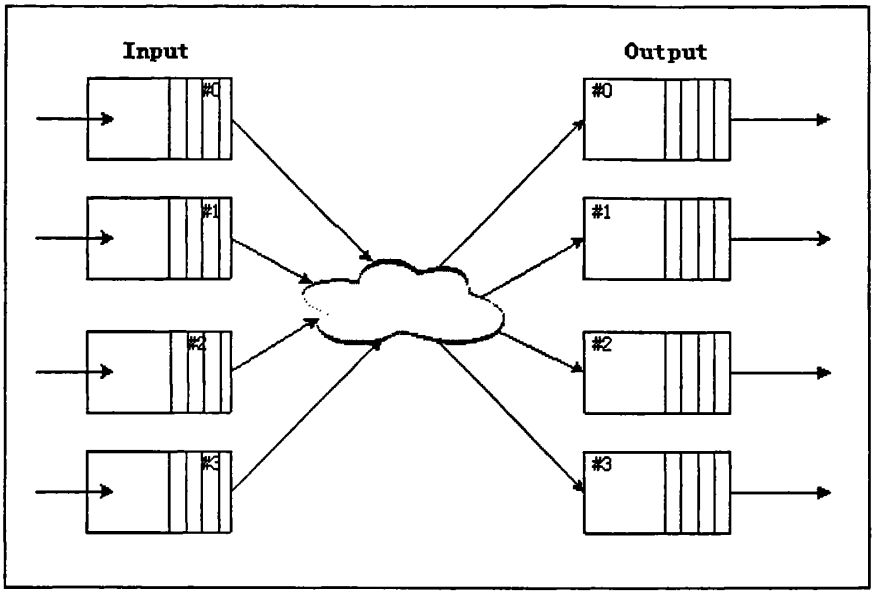

图2.6 组合输入输出排队结构

组合输入输出排队结构（Combined Input and Output Queuing，简称 CIOQ），是一种在输入和输出端口都设置了缓存的排队结构，如图 2.6 所示，CIOQ 型结构在每个输入输出端口都有 FIFO 队列。CIOQ 结构结合了输入排队和输出排队的特点。由于传统的输入排队存在 HOL 阻塞，该 CIOQ 型排队结构存在相同的问题。在 2.2.1 节中可知，VOQ 排队可以解决 HOL 阻塞，因此在输入排队结构上稍加改进便可以解决该问题。

一种改良的带 VOQ 的 CIOQ（Combined Input and Output Queuing structure with VOQ）结构如图 2.7 所示，在输入端使用了 VOQ 结构，输出端仍然使用 FIFO 队列。输入端的 VOQ 队列同样是为了消除 HOL 阻塞而设置小缓存排队结构，这样使得输入和输出调度分开。其中输入调度主要解决多个输入端口不同报文间对输出端口的竞争，而输出调度主要解决一个时隙内多个输入缓存非空时，输出链路竞争问题。各个输入输出端口可独立完成自己的调度，不需要端口间状态信息交互。因此大大降低了输入排队调度算法的实现复杂度，同时具有输出排队在网络性能上的诸多优点。所以该结构从理论上来说能够克服输入排队和输出排队的

不足从而达到非常好的性能，提供更好的延时与吞吐量[18]。带 VOQ 的 CIOQ

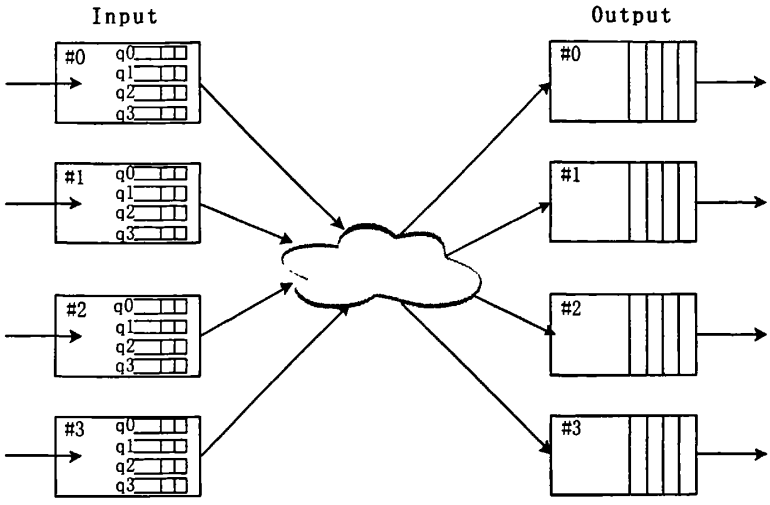

图2.7 改良型 CIOQ 排队结构

已被证实在均匀非相关流量下具有 99%的吞吐率[19][20]。

2.3 本章小结

本章主要分析各种排队结构的性能优缺点。首先介绍了简单 FIFO 排队结构，随后在输入排队结构的基础上分析了虚拟输出排队的优点以及输出排队的结构的优势和存在的问题，最后介绍了具备输入排队与输出排队结构优点的组合输入输出排队结构。

第三章 队列调度算法的基本原理

队列调度算法运行在网络节点中发生冲突需要排队等待调度之处，它按照一定的服务规则对网络节点的不同输入业务流分别进行调度和服务，使所有的输入业务流能按照预定的方式共享网络节点的输出链路带宽。如图 3.1 所示，输入业务到达网络节点后，分别暂存到相应的缓存队列中，然后需要在什么时间按照什么规则调度什么队列就由调度算法来处理。

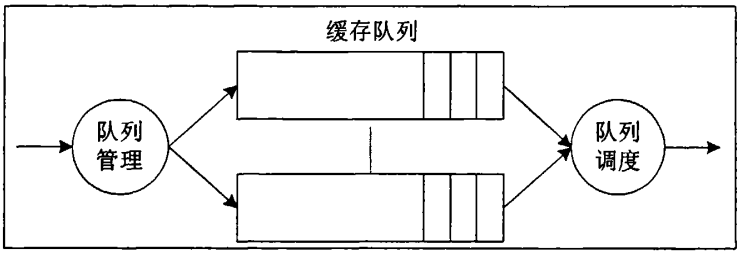

图3.1队列调度算法的基本原理

通常情况下，队列管理机制位于队列的输入端，负责管理系统中缓存资源的分配，根据系统策略和到达分组的信息来决定是否允许其进入队列，进入哪个队列，而队列仅仅作为一个容器，在没有调度指令的前提下不会完成出队列操作，在队列的输出端，则有队列调度机制负责带宽分配和延迟调整，就是按照业务需求对各队列进行调度，两者相互配合完成完整的队列调度操作。

3.1 队列调度算法的简介

3.1.1 队列管理的简介

为了给不同的业务提供不同的服务质量保证，通常在网络节点把业务放到不同的队列里，再利用不同的调度规则对业务队列进行调度。

在 Internet 中，可以基于 IP 源/目的地址、传输层/源/目的端口或协议类型对输入业务进行分类，也可以根据 IP 头中的服务类型（Type of Service）字段进行分类，每一类对应一个队列。

如图 3.2 所示，对于队列调度，我们需要得到每个队列的有效长度，队列长度决定该队列是否有效，调度算法就是根据该结果对所有队列进行预处理，并且从有效的队列中调度。当队列长度为 0 时，认为该队列无效，需要释放带宽将本

次调度机会给其他非空的队列，从而不会浪费有效带宽。

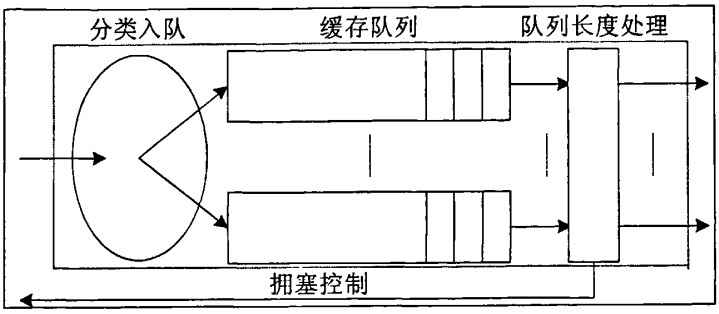

图3.2队列管理结构图

由于调度是按照队列长度来进行的，在每个时隙至多有一个报文入队，一个报文出队，且报文入的队列和出的队列都是随机的，对于某一条队列，很可能会出现入队次数大于出队次数的情况，如果情况持续下去缓存必然溢出，所以为每个业务流维护独立的队列缓冲区的同时，还需要使用拥塞控制，负责产生入队许可，根据一定时间间隔内各队列的丢包率自动调整队列的缓冲区长度，最大程度上优化系统缓冲区的利用率。

3.1.2 调度算法的简介

队列管理仅仅是缓冲资源的分配机制，然而从这一方面来考虑整个网络传输节点中的队列机制是不全面的。在网络传输节点中，最重要的资源是输出链路的带宽资源，配合计算资源和缓冲资源，实现完整的排队、转发等控制机制。因此，网络传输控制机制的首要目标就是要在到达的用户数据流之间合理地分配输出链路的带宽资源，而这个目标是通过队列调度算法来实现的。调度算法是一系列规则，它按照一定的服务规则对网络节点的不同输入业务流分别进行调度和服务，使所有输入业务流能够按照预定的方式传输到下一节点[21]，同时使所有的输入业务流能按预定的方式共享网络节点的输出链路带宽。假设总共有 N 个队列，队列调度算法的任务是如何从这 N 个队列中选择下一个要传输的分组，如何把输入业务流对应到不同的队列中，不同的调度算法在不同的网络环境里有不同的方法，如先入先出 FIFO 只根据分组的到达时间对之进行服务，这时队列数为 1，这种调度算法的粒度较大，因为它是把所有输入业务流无区别地放在一个队列里。而较复杂的调度算法则会根据一定的规则把输入业务流对应到不同的队列里，从而对输入业务进行有区别的服务。

现有的调度算法有很多种，可以根据不同的分类标准进行划分。按服务规则分类，队列调度算法可以分为：先到先服务、循环调度、处理机共享、优先级服

务、随机服务等。根据调度算法的调度目标分类，调度算法可以分为基于时延的和基于速率的。根据通信网络环境，又可分为无线环境下的队列调度有线环境下的队列调度，对于无线环境下的队列调度算法，基本上是在有线队列调度算法基础上考虑到无线网络环境下的一些特定情况，进行了相应的改进。根据调度算法的工作状态，又可分为工作保持和非工作保持，其中工作保持算法表示只要系统中有等待分组，调度算法就一定会工作。非工作保持算法则意味着即使系统中有等待分组，调度算法也可能暂时不对其进行调度，这类调度算法一般要求在输入业务流被调度之前需经过一个整形器进行整形处理。工作保持算法具有更高的链路利用率，而非工作保持算法能对端到端延迟及延迟抖动进行控制。我们说一个调度算法是稳定的，是指输入队列长度的期望值不能无限增长，如果在所有独立的和可允许的到达下都是稳定的，我们说这个在这个调度算法下调度结构能够达到 100%的吞吐量。在高速网络中传输一个报文的时间很短，所以调度算法必须在短时间内完成对报文的调度，这就要求调度算法要简单、易实现。另外，当业务流量增加和线路速率变化范围较大时，调度算法仍能够有效的工作，这要求调度算法应该具有良好的扩展性。实际上，对于某一特定的调度算法，根据不同的分类标准，又可属于多个不同的类，而且现代的网络业务同调度算法提出了各种各样的性能要求，促进了各种算法的相互融和，并形成不同的算法，所以并没有一种统一的分类标准。

3.1.3 队列调度算法的性能指标

为了平衡影响调度算法设计的各种因素，获得比较好的性价比，调度算法的设计必须根据每类业务的特点选择不同的算法，但是任何调度算法都必须具有以下基本特征：

a) 吞吐量

吞吐量是单位时间内传送的通过网络的给定点的平均比特数。调度算法必须能够隔离不同数据流之间的相互影响，即使是在有恶意数据流存在的情况下，调度算法也能够提供最基本的 QoS 保证；

b) 调度算法必须为单个数据流提供端到端时延保证

调度算法必须在不过分影响网络资源利用率的前提下，可以为特定要求的业务流提供端到端的时延保证。同时，调度算法还应该具有能够通过只控制保留带宽资源来控制数据流的时延上下限的能力；

c) 调度算法必须能够有效地利用共享链路带宽

调度算法必须充分利用数据流的统计复用特性，有效地处理突发信源；

d) 共享链路带宽必须在队列中加权公平分配

公平性是指链路带宽必须以公平合理的调度方式分配给共享链路带宽的业务流，这样即使发生高突发型业务也不至于影响其他正常业务流。而不公平的调度算法在短时间间隔内可能会为两个具有相同保留带宽的数据流提供差别很大的服务速率；

e) 可实现性

调度算法的可实现性指的是算法复杂度和工程中的易实现性与易于扩展性。高速网络中传输一个分组的时间很小，所以调度算法必须能在很短的时间完成对分组的调度，因此调度算法必须有很小的复杂度。而且好的调度算法应该易于硬件实现，且具有良好的可扩展性，可以适用于巨大的数据流和变化范围很大的链路速率；

f) 调度算法可以满足多样化的业务需求

网络的发展趋势是业务多样化，在实际网络中，彼此有分层关系的业务流常常共享链路，在业务流到来时，调度算法能够根据所面临的业务类型，尽可能满足某些特定要求，例如能够处理某些端口大幅度到达的突发业务流，能够及时响应具有更高优先级的报文等等。这时单一的调度算法无法满足链路带宽共享的需求，需要考虑使用综合结构的调度算法。也就是说，在追求简单性和易实现性的同时，考虑算法的综合性能。

3.2 网络设备中最常用的调度算法

3.2.1 Round Robin

轮循算法（Round Robin，简称 RR），各队列之间轮循调度。RR 的原理比较简单，即队列按顺序轮流调度，机会均等；其中无效队列（使能关闭、队列空、反压等）不参与轮循，其带宽被释放以供其他有效队列调度。如图 3.3 所示：在一个子端口有四个队列的情况下，采用 RR 对各队列进行调度，每个队列得到调度的机会都是均等的。假设以 16 次调度为统计范围，如果所有的队列队列都非空，那么它们在 16 次调度内得到的调度机会都是 4 次。只有在某些队列空的情况下才会使调度比例不均，例如队列 0 为空，那 queue0、1、2 和 3 最后的调度比例将是 0：6：5：5，就是说队列 0 不参与调度，调度机会给予非空的队列，并且在非空的队列中轮循。只要有队列非空 RR 算法都在工作，而且各非空队列机会均等，所以 RR 算法体现了公平与工作尽力的原则。

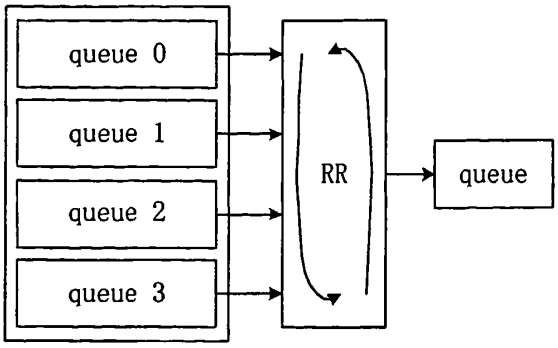

图3.3 RR 调度算法结构图

虽然队列 RR 调度简单易行，但是无法体现报文的不同业务类型，由于对所有业务类型一视同仁，也就没有了轻重缓急的区别，对一些优先级别很高的类型处理不及时。所以队列 RR 调度性能不高，在队列很多业务类型较多并且对一些关键业务类型需要及时得到响应的情况下不推荐使用。

3.2.2 Strict Priority

严格优先级调度（Strict Priority，简称 SP），该算法按照队列优先级的高低进行调度，高优先级优先调度，低优先级后调度。高优先级队列非空，就会一直调度优先级高的队列出队，只有在优先级高的队列无效后（队列空、被反压、使能关闭），才调度低优先级队列。如图 3.4 所示，从 queue0 到 queue3 也按照优先级的降序做绝对优先调度，在报文出队的时候，SP 首先让处于最高优先级的 queue0 队列中的报文出队，直到 queue0 队列中的报文发送完，然后发送中优先队列中的报文，同样直到发送完，依次类推，只有 queue0、queue1 和 queue2 队列的报文全部出队才去调度 queue3 队列。只要高优先级队列非空就一直调度高优先级队列，优先级别越低的队列得到调度机会就越少（queue0>queue1>queue2>queue3）。这样，分类时属于较高优先级队列的报文将会得到优先发送，而较低优先级的报文将会在发生拥塞时被较高优先级的报文抢先，使得关键业务的报文能够得到优先处理，非关键业务的报文在网络处理完关键业务后的空闲中得到处理，既保证了关键业务的优先，又充分利用了网络资源。应用上 SP 调度通常针对低延时业务，保证该业务报文快速出队。该算法简单，电路实现相对比较容易。同时它可以对优先级高的业务提供优先服务，保证关键的数据流得到优先对待，提高关键数据流的响应速度。

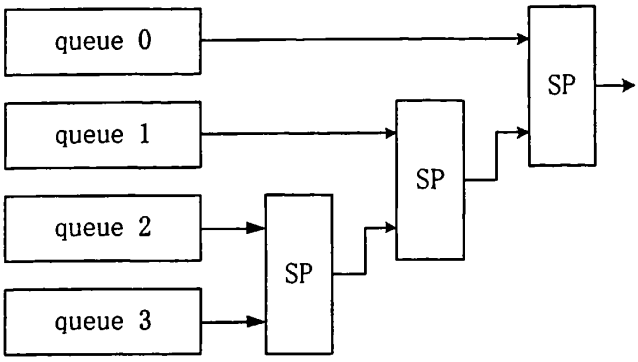

图3.4 SP 调度算法结构图

但 SP 算法通常导致的结果是优先级低的数据流常不能分配到足够的带宽，最坏的结果可能导致优先级较低的数据永远发不出去。例如当优先级 0 的队列一直有报文入队，队列一直不为空，这时就会导致 queue1、2 和 3 队列里面的报文长时间无法出去，甚至会堵死在队列里，这在实际应用中是有可能出现的，这样就不满足调度的基本要求，高优先级的数据流严重影响了其他数据流。

解决 SP 算法中低优先级“饿死”的对策是分隔队列，即设置调度阈值[22][23] (Queue Length Threshold，简称 QLT)。给各个队列设置调度阈值，需要进行调度时从最高优先级开始比较队列的长度和调度阈值。当最高优先级队列的长度大于等于调度阈值时，该队列头部的报文首先被选择调度。当最高优先级队列的长度小于其调度阈值时，不再服务调度该队列，而是检查具有次高优先级的队列，依次类推。通过调度阈值的合理配置，QLT 算法能够对队列进行简单的分隔，限制了它们之间的相互影响，提高了公平性。

3.2.3 Weighted Round Robin

加权轮循算法（Weighted Round Robin，简称 WRR），是一种加权轮循分组调度算法，其核心思想是通过周期性的访问各个队列达到为队列提供带宽保证的目的。WRR 算法主要适用于分组为定长的应用环境，它在 RR 基础上为每个队列赋予一个权值，同时为每个队列维护一个计数器。在每次轮循时，当该队列获得调度机会时，每调度一个报文，权重值减 1，直到权重减为 0，同时将该队列权重恢复为配置值。无效的队列不参与调度，带宽被释放以供其他有效队列使用。WRR 算法主要过程：调度之前，用对应的非空队列权重配置值 weight_config 更新对应的 weight_counter。以计数器 Rr_counter 表示的队列号 n 开始依次轮循，直到第一个非空的 weight_counter 非 0 队列 m，轮循停止，那么 m 就是当前调度出的队列号。当调度完成后，把对应的 weight_counter(m)减 1。然后 Rr_counter 加 1

指向下一次轮循的队列。

WRR 的调度流程如图 3.5 所示：

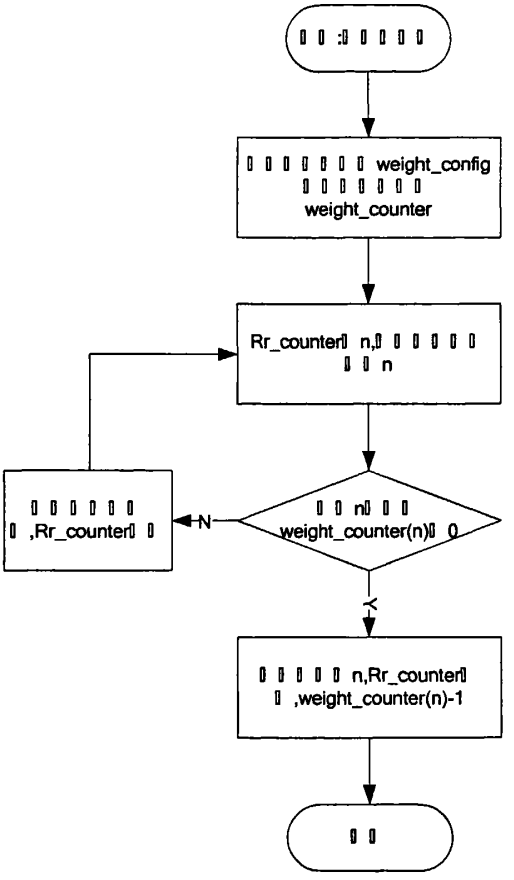

图3.5 WRR 调度算法流程图

WRR 算法能够比 RR 更灵活地控制带宽在不同队列的分配，而且同时以比较平滑的方式输出业务，但仍然存在由于分组变长带来的不公平性。

3.2.4 Deficit Round Robin

差额轮循算法（Deficit Round Robin，简称 DRR）的提出是为了解决 RR 和 WRR 算法中由于分组变长带来的不公平性，DRR 算法以字节为单位为每个队列分配一个带宽配额，该配额的比例对应于队列服务读率的比例。同时为每个队列维护一个计数器，初始化为其带宽配额。每次轮循时，如果待发送分组长度小于

或等于计数器值，则允许发送，并把计数器减去此分组长度值；如果待发送分组长度大于计数器值，则检查下一个队列，同时把该队列计数器值累加到下一次循环(即下次调度该队列之前把上次剩余值和配额之和赋予计数器)。DRR 由于考虑了分组变长这一信息，很好地解决了带宽分配的公平性问题，缺陷是不能很好地满足业务的时延特性，不能像 WRR 那样以较平滑的方式调度输出业务流，不能有效地支持实时业务。

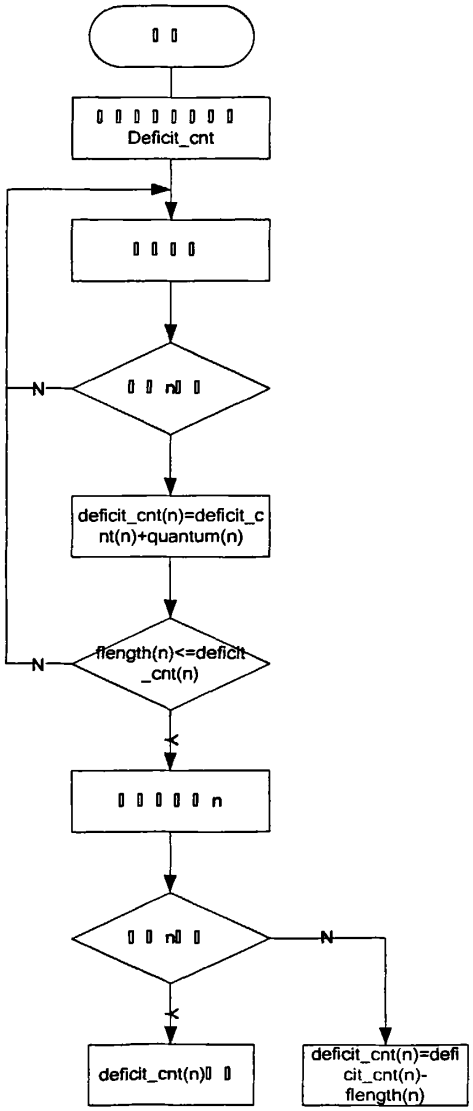

图3.6 DRR 调度算法流程图

DRR 给待调度的每个队列分配一个可配置的服务量额度 quantum，如果一个队列在当前轮循中的数据帧的帧长大于分配给该队列的服务额度 quantum，那么该数据帧将不被发送，帧长和当前服务额度 quantum 的差值将加到该队列下一次轮循的服务额度中，作为队列下一次轮循的可用带宽。

DRR 算法资源(假设有 N 个队列)一个轮循计数器 RR_CNT 每个队列各一个服务额度配置值 Quantum(N)，假设以字节为单位每个队列各一个服务量赤字计数器 Deficit_cnt(N)，即每个队列当前允许被服务的数量，假设以字节为单位各队列待服务的数据长度 Flength(N)。DRR 调度流程如图 3.6 所示，所有队列的 Deficit_cnt 都初始化为 0 轮循计数器 RR_CNT 轮循到非空队列 n。使 Deficit_cnt(n)=Quantum(n)+Deficit_cnt(n)。如果队列 n 待被服务的数据长度 Flength(n)小于等于 Deficit_cnt(n)，则调度出队列 n 如果调度后队列 n 变空，使 Deficit_cnt(n)清零，否则 Deficit_cnt(n) =Deficit_cnt(n)-Flength(n)。

3.2.5 Weighted Fair Queuing

在介绍加权公平队列前，先要理解公平队列（Fair Queuing，简称 FQ）[24]。FQ 是为了公平地分享网络资源，尽可能使所有流的延迟和抖动达到最优而推出的。

加权公平队列(Weighted Fair Queuing, 简称 WFQ)是最常见的 FQ 调度算法，是一种基于数据流的排队算法，它提供了动态的、公平的排队方式，基于权重来划分通信队列的带宽。WFQ 用于减少延迟变化，为数据流提供可预测的吞吐量和响应时间。目标是为轻载用户和重载用户提供公平一致的服务，并能自动适应不断变化的网络通信环境

其基本思想是：在 WFQ 算法中，每个队列都有一个离开时间 DT(Departure Time)，其实就是时间戳(timestamp)，每次调度都会从所有有效队列中选取一个 DT 值最小的队列；由于是 DT 最小队列被优先调度，所以各队列所获得的带宽与他们的 DT 初始值成反比；WFQ 算法的优点是能够同时保证流量和延时，特别是对于流量较小的流，延时能够得到公平的待遇。在资源允许情况下 WFQ 调度算法是首选，缺点是 DT 最小值的计算逻辑需要消耗较大资源，特别是在参与调度的队列数很多的情况下，要从 N 个 M 比特的 DT 值中选择一个最小值，最少需要 $\log_2 N$ 级 M 比特的比较运算，具体的逻辑复杂度和资源占用率由 N 和 M 决定。从原理上讲 WFQ 是理想的调度算法，要比 RR 类调度性能优越，但实现起来要比 RR 类调度复杂，运算量很大，消耗的资源也会非常的大，尤其在队列或流的数量很大的时候。

本论文主要实现了 WFQ 算法，在第四章中会对实现过程做具体介绍。

3.2.6 调度算法的比较

表 3.1 给出了本文讨论的各种调度算法的优缺点比较。

表3.1 调度算法比较表

调度算法	队列数	优点	缺点
FIFO	1	实现简单，器件天然支持	HOL：不能体现业务类型
RR	可配置	RR 的实现简单，工作尽力	具有时延特性
SP	可配置	算法简单,电路实现简单。同时它可以对优先级高的业务提供优先服务，保证关键的数据流得到优先对待	优先级低的数据流常不能分配到足够的带宽，最坏的结果可能导致优先级较低的数据永远发不出去
WRR	可配置	避免了 SP 低优先级队列的饿死问题。同时对优先级较高的队列分配较大的权重，保证关键的数据流得到优先对待	各队列调度不均，权重高的队列易产生流量 burst
DRR	可配置	主要解决由于不同数据流的不同包长引起对队列服务的不公平性	不能很好地满足业务的时延特性，不能象 WRR 那样以较平滑的方式调度输出业务流，不能有效地支持实时业务
WFQ	可配置	理想的调度算法，从原理上讲要比 RR 类调度性能优越	实现要比 RR 类调度复杂一点，运算量很大，在队列或流的数量很大时一般不采用

3.3 本章小结

随着网络流量的提高，网络设备的转发能力需要加强，虽然通过提高内部转发速率即从硬件上加以改善能较好的达到很高的吞吐量，但这显然不是研究调度算法的初衷。队列调度算法是网络调度系统中的一个重要组成部分，其性能优劣将会影响到整个网络系统的性能。本章主要介绍了 RR、SP、WRR、DRR 和 WFQ 等网络设备中常用的调度算法，分析各调度算法的基本原理、结构与优缺点。

第四章 WFQ 调度算法的设计与实现

WFQ 是一种经典的用于网络带宽分配的调度算法，相对于其他调度算法（比如 WRR），WFQ 在调度的公平性、时延、时延抖动等方面都具有更佳的性能。本章说明基于四个队列的 WFQ 算法实现，支持四个队列进行 WFQ 调度。WFQ 算法的实现方法有多种，本文中描述的电路并非是唯一的实现方式。

4.1 WFQ 调度算法的基本原理

WFQ 是一种基于时间戳（TimeStamp）的调度算法，基本思想是，每个报文在入队时都被打上一个时间标记，该时间标记表示报文的离开时间（Departure Time），出队时对 DT 进行比较，选择 DT 值最小的报文出队。算法已在上文具体说明，这里就不再赘述。

图 4.1 是一个 n 输入的 WFQ 算法流程图。可以按照如下步骤最终输出 DT 值最小的有效队列号：

（1）配置 W_1，……，W_n n 个队列的 DT 初始值；

（2）将 DT 值赋给一个临时变量 W_i'，留作下面阶段处理；

（3）当 W_i'（i=0~n）对应的队列状态有效时，从所有有效队列中找出最小的 DT 值 W_x，判断所有队列是否与 W_x 相等，非最小 DT 值减掉 W_x 赋值给对应的 DT 值 W_i' 作为 i 队列最新的 DT 值，如果无效队列对应的 DT 值小于 W_x，则 W_i'=0；留作下一次判断用；

（4）可能会找到 m 个相同的最小 DT 值，分别用 W_{x1}，……，W_{xm} 来表示；在这 m 个队列中找出队列号最小的队列 z；输出最小队列的队列号 qnum；

（5）输出队列（队列号等于 z）对应的 DT 值恢复到最初配置值，其他最小 DT 值对应的 W_i' 被减为 0；

（6）按照步骤（2）～（5）循环操作，选择输出每次 DT 值最小的有效队列号；

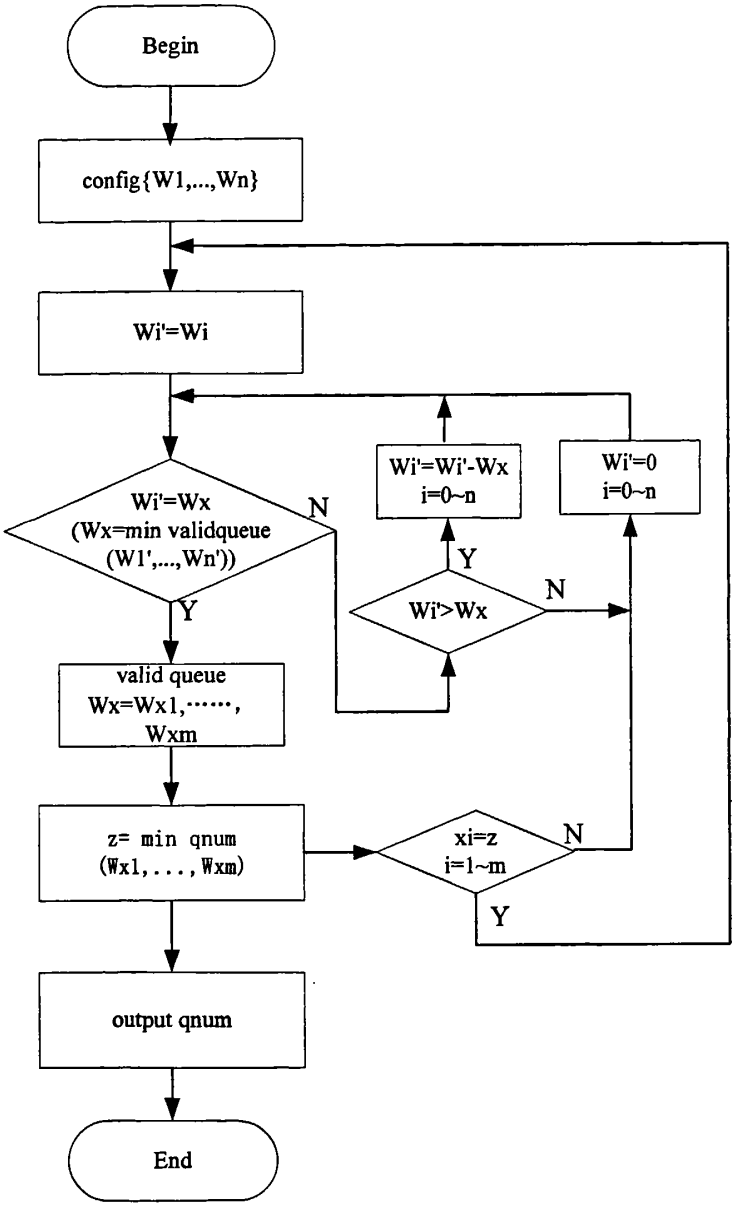

图4.1 WFQ 算法流程图

本文实现的是基于四个队列的 WFQ 电路，队列编号为 0～3。每个队列可单独配置 DT 初始值，根据初始值在这四个队列之间进行 WFQ 调度，从而使它们按照配置比例共享端口带宽。

表4.1 WFQ 调度过程表

	队列 0	队列 1	队列 2	队列 3	选中输出的队列号
配置 DT 值	4	5	1	6	
产生调度后更新的 DT 值	4	5	1	6	2
	3	4	1	5	2
	2	3	1	4	2

	1	2	1	3	**0**
	4	1	0	2	**2**
	4	1	1	2	**1**
	3	5	0	1	**2**
	3	5	1	1	**2**
	2	4	1	0	**3**
	2	4	1	6	**2**

表 4.1 是一个四队列 WFQ 调度过程的演示表，以表中第一次输出有效调度信号为例说明，假设队列 0～3 都为有效队列：

（1）首先给队列 0 配置 DT 值为 4，队列 1 配置为 5，队列 2 配置为 1，队列 3 配置为 6；

（2）选择所有队列中的最小 DT 值，选择结果为队列 2，最小 DT 值为 1；

（3）输出队列号为 2；队列 0、1、3 的 DT 值分别减去 1 后赋给对应的队列，队列 2 的 DT 值恢复到配置的初始值 1；

（4）重复以上步骤；

从表中可以看出，队列获得的调度机会与它们的 DT 值成反比，只要队列状态非空，DT 值越小的队列得到的调度机会越多。

4.2 WFQ 调度算法的电路实现

WFQ 调度模块的主要功能是根据队列状态信息与队列 DT 值，在所有有效队列中选择输出 DT 值最小的队列号，同时更新所有队列 DT 值以备下次调度使用。对于报文的分类与入队，还有队列的拥塞控制不作说明，本章重点说明 WFQ 算法的实现。如图 4.2 所示，WFQ 模块的实现过程主要分为三个部分，每个部分为一子模块。WFQ 的调度过程就是循环执行这三个子模块的功能。

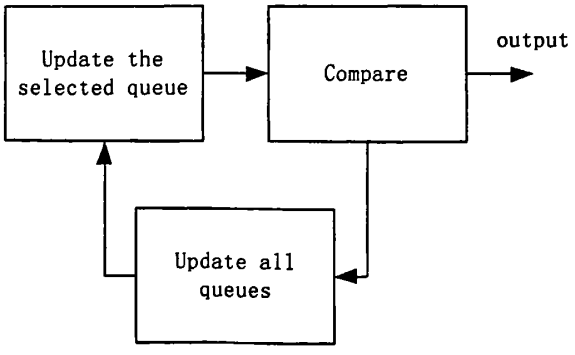

图4.2 WFQ 结构框图

各部分功能如表 4.2 所示，各子模块间按流水操作，每个模块在 1 个时钟周期内完成操作，子模块内所有操作使用组合逻辑。

表4.2 WFQ 子模块列表

子模块	功能简介
DT 值比较模块	队列 DT 值的比较，选出四个队列中 DT 值中最小的一个作为调度的结果，并将调度结果输出给 DT 值更新模块
更新 DT 值模块	更新所有队列 DT 值，所有队列 DT 值需减去被调度队列的 DT 值，包括被选中的队列
更新被调度队列 DT 值模块	恢复被调度队列的 DT 值，然后将所有更新后的队列 DT 值送给 DT 值比较模块

后文将分别介绍这三个子模块的具体实现。

4.2.1 DT 值比较

该模块主要负责在四个等候队列中通过比较选择 DT 值最小的队列作为调度输出。主要的模块间接口信号如表 4.3 所示：

表4.3 DT 值比较子模块接口信号列表

信号	方向（I/O）	描述
ptr_out	O	本次调度选择输出的队列号
ptr_out_vld	O	输出有效信号
dt_comp_out0~3	O	队列对应的 DT 值
q_sta0~3	I	队列 0～3 的队列状态
dt0~3	I	队列 0～3 的 DT 值

DT 值比较之前，首先要预处理输出的四个队列的队列长度，如果某队列长度为 0 或被反压，则该队列的队列状态会被置为 0，认为该队列状态无效，本次调度时不会调度该队列，相当于释放网络带宽给有报文排队的队列；如果某一队列里有报文在排队，则无论报文多少，该队列状态被置为 1，表示该队列非空。对于队列状态的处理是一个持续的过程：当有报文入队的时候队列的长度为增加，当该队列被调度的时候队列的长度会减少，有时会二者有之。

得到所有队列的队列状态之后，就可以进行四个队列的 DT 值比较。在比较过程中可以采用四个队列依次比较的方式进行：队列 0 与队列 1 进行比较，得到较小者与队列 2 再进行比较，依此类推，共进行三次比较后得到四个队列中 DT 值最小者，寄存输出该队列的队列号。用下面一段伪代码来表示大概的实现过程：

```
input : q_sta3, q_sta2, q_sta1, q_sta0; //预处理后的队列状态信号
input : dt3, dt2, dt1, dt0;   //输入的处理后的四个队列对应的 DT 值

reg dt_reg[0:3];   //dt_reg 存放四个队列的 DT 值
reg qstat[0:3]; //qstat 存放四个队列的状态
dt_reg [0]= dt0;
```

```
dt_reg [1]= dt1;
dt_reg [2]= dt2;
dt_reg [3]= dt3;

qstat[0] = q_sta0;
qstat[1] = q_sta1;
qstat[2] = q_sta2;
qstat[3] = q_sta3;
//通过 DT 值比较选择输出的队列号
for(index=0;index<3;index++)
{
  if((dt_reg[index] < dt_reg[index+1] ) && （qstat[index]!=0))
  {
      dt_reg[index+1] = dt_reg[index];
       ptr_out = index;
  }
}
//本次比较输出有效信号
if((q_sta0 != 0)&& (q_sta1 != 0) && (q_sta2 != 0) && (q_sta3 != 0))
    ptr_out_vld = 1;
else
    ptr_out_vld = 0;
```

DT 值的比较需要在一个时钟周期内完成。由于使用了组合逻辑实现，电路实现的关键在于如何减少逻辑级数，防止产生时序瓶颈。如果按照伪代码中的一个 for 循环的方式，实现比较的功能需要深度为 3 的组合逻辑如图 4.3 所示。虽然逻辑上在一个时钟周期内即可得到比较结果，但是逻辑比较深，综合之后会产生较大的物理电路，在时钟频率较高的情况下时序可能会过不去，尤其在队列较多的时候情况会更加恶劣。

为了减少组合逻辑的深度，可以将四个队列分成两组，两个队列一组，如图 4.4 所示。在进行比较时，先组内进行比较，然后再进行组间比较。由于两个组内的比较是并行执行的，相当于比上面的比较方式减少了一级组合逻辑。通过分组比较的方式，实现比较功能只需要深度为 2 的组合逻辑。综合出来的电路无论是面积还是时序都比上一种做法好，由于本模型中只有较少的四个队列，无法体现优势，在队列数目较多的时候，优势将更加明显。

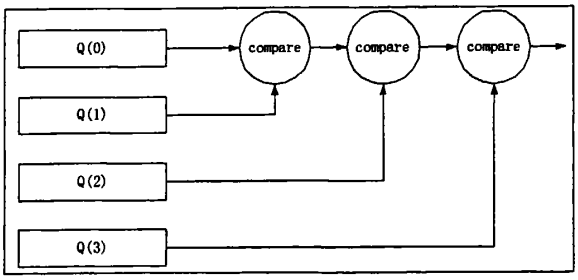

图4.3 WFQ 比较逻辑示意图

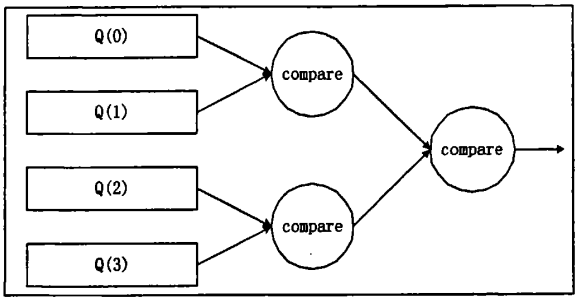

图4.4 WFQ 分组比较逻辑示意图

在比较过程中会遇到两个问题：其一，队列状态为空的队列是否参与比较；其二，遇到队列 DT 值相等的情况如何处理。对于第一个问题，当一个队列为空时代表队列内没有任何报文在排队，根据 WFQ 的基本原理，该队列是不应该得到调度的。根据这个原则，如果在组内比较时得出 DT 值较小的那个队列为空就选择剩下的那个队列，如果两个队列都为空，则该分组的比较输出无效，不参与下一级的比较，第二个分组执行相同的操作。如果所有的四个队列状态都为空，则此次产生无效的调度，将调度有效信号置为 0，代表本次调度输出无效。虽然所有队列都为空的情况在队列状态处理的时候就可以得到无效调度的判断，但由于每次调度都要执行比较的步骤，那在队列状态处理的时候做输出无效判断就显得浪费逻辑了。对于第二个问题，如果组内比较时发生两个队列的 DT 值相等，则优先选择队列号小的队列；组间比较时也是遵循这个原则。如果某一队列被配置成绝对优先队列即 DT 值配置成 0，这里不需做特殊的处理，因为在比较过程中该队列的 DT 值一直都是最小的。如果出现数个 DT 值为 0 的队列，就按照队列号小者优先的原则调度。解决以上所有问题后，DT 值比较模块将寄存输出 DT 值最小队列的队列号，并将输出有效信号置为 1，代表本次调度结果有效。

本模块产生比较结果后，调度结果将被用于下一个模块做所有队列 DT 的更新操作，此次产生的队列号会直接输出。虽然已经得到 WFQ 调度结果，但是从 WFQ 工作流程来看，一轮完整的调度行为并未结束，只有当所有队列的 DT 得到更新并保存后才算完全结束。

4.2.2 更新所有队列 DT 值

当产生一次有效得调度之后，需要根据所选择的队列 DT 值去更新所有的四个队列的 DT 值，并将所有队列更新后的 DT 值输出给下一模块。主要的模块间接口信号如表 4.4 所示：

表4.4 队列 DT 更新子模块接口信号列表

信号	方向（I/O）	描述
ptr_out	I	本次调度选择输出的队列号
ptr_out_vld	I	调度选择输出有效信号
dt_comp_out0~3	I	比较模块输出的四个队列的 DT 值
dt_up_out0~3	O	更新后所有四个队列的 DT 值

当本模块收到上一模块送来的调度队列号、调度输出有效信号与所有四个队列的 DT 值后，先要根据调度输出有效信号来判断此次送过来的队列号是否有效，若无效则不做任何操作，本模块的输出维持原来的值。如果本次送来的调度输出有效，则根据调度队列号从同时送过来的各队列 DT 值得到本次调度队列所对应的 DT 值，然后所有队列的 DT 值都减去这个最小的 DT 值，完成队列 DT 值的更新。所有信号处理完后都寄存输出。下面的伪代码表示本模块大概的实现过程：

```
input  : ptr_out;
input  : ptr_out_vld
input  : dt_comp_out3, dt_comp_out2, dt_comp_out1, dt_comp_out0;
output : dt_up_out3, dt_up_out2, dt_up_out1, dt_up_out0;

reg  dt_reg[0:3];  //存储比较模块输出的各队列的 DT 值
reg  dt_up_out[0:3];
dt_reg[0] = dt_comp_out0;
dt_reg[1] = dt_comp_out1;
dt_reg[2] = dt_comp_out2;
dt_reg[3] = dt_comp_out3;

for(i=0;i<4;i++)
{
  if(ptr_out_vld==1'b1)  // 如果本次调度有效
  {
      if(dt_reg[i] > dt_reg[ptr_out])
          dt_reg[i] = dt_reg[i] – dt_reg[ptr_out];
```

```
        else
            dt_reg[0] = 0;
        dt_up_out[i] = dt_reg[i];
    }
}
//更新后输出的各队列的 DT 值
dt_up_out0 = dt_up_out[0];
dt_up_out1 = dt_up_out[1];
dt_up_out2 = dt_up_out[2];
dt_up_out3 = dt_up_out[3];
```

本模块逻辑功能如图 4.5 所示：

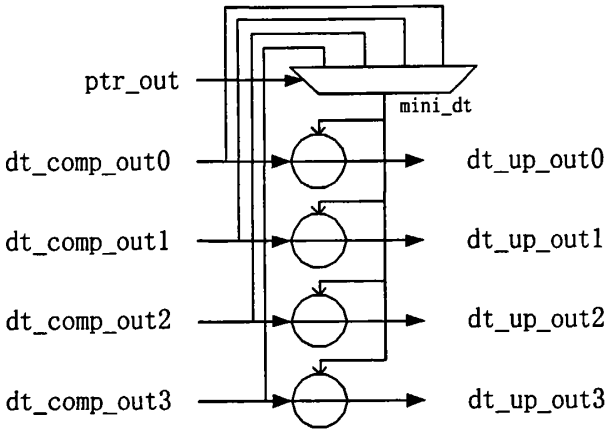

图4.5 WFQ 队列 DT 值更新逻辑电路示意图

如果某一队列的状态为空，且空状态持续时间较长，那么该队列将一直得不到调度，此时空队列的 DT 值持续被减，在得到调度机会并更新 DT 值前有可能被减少到小于 0。从代码可以看出，为了解决这个问题，在设计的时候就需要在 DT 值更新的时候要增加一个保护逻辑，当检查到某一队列的 DT 值小于选中队列的 DT 值（说明该队列由于状态为空而未被选中），就把该队列的 DT 值置为 0，直到该队列状态变为非空后而且得到调度后该 DT 值才会得到改变。在这个模块中所有的队列的 DT 值得到更新，而当前被调度队列的 DT 值更新为 0，然后将所有队列的新 DT 值输出给被调度队列 DT 值更新模块。

4.2.3 更新被调度队列 DT 值

经过上一个模块的处理，被调度队列的 DT 值被更新为 0，进行下一次比较前需要重新给该队列重新赋 DT 值。输出队列 DT 值更新模块就是完成这个操作。主要的模块间接口信号如表 4.5 所示：

表4.5 队列 DT 更新模块接口信号列表

信号	方向（I/O）	描述
ptr_out	I	本次调度选择输出的队列号
ptr_out_vld	I	调度选择输出有效信号
dt_up_out0~3	I	DT 值更新模块输出的四个队列更新后的 DT 值
dt_table0~3	I	DT 初始值表
dt0~3	O	用作比较的所有四个队列的 DT 值

本模块会根据本次调度选择的队列号，将被选择的队列的 DT 值更新到 DT 值表对应的值，其余队列的 DT 值不做处理。同样，DT 值的更新依然是在调度选择输出有效的情况下才执行，如调度选择输出无效，各信号维持不变。所有队列的 DT 值经过修正后寄存输出到 DT 比较模块，然后进入下一次 DT 值的比较阶段。下面的伪代码表示本模块大概的实现过程：

```
input  : ptr_out;
input  : ptr_out_vld
input  : dt_ up _out3, dt_ up _out2, dt_ up _out1, dt_ up _out0;
input  : dt_table3, dt_table2, dt_table1, dt_table0;
output : dt3, dt2, dt1, dt0;

reg  dt_table;    //各队列原始配置的 DT 值
reg  dt_up_in;    //各队列更新后的 DT 值
dt_table = {dt_table3, dt_table2, dt_table1, dt_table0};
dt_up_in = {dt_up_out3, dt_up_out2, dt_up_out1, dt_up_out0};
for(i=0;i<4;i++)
{
  if(i == ptr_out)
      dt[ptr_out] = dt_table[ptr_out];
  else
      dt[i] = dt_up_in[i];
}
```

本模块逻辑功能如图 4.6 所示

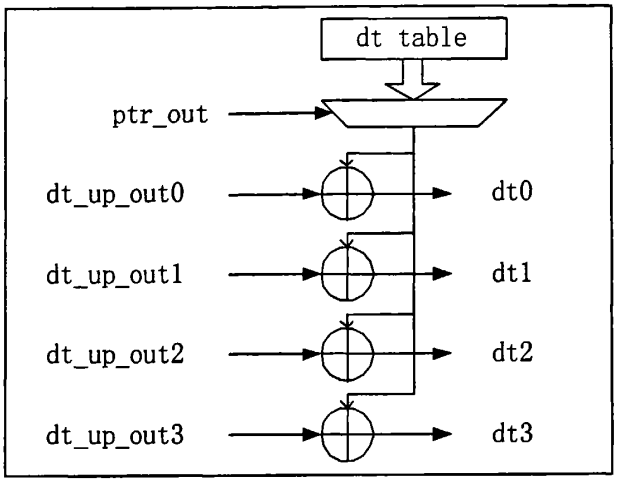

图4.6 WFQ 更新 DT 逻辑电路示意图

至此，完成一次完整的 WFQ 调度。

4.2.4 WFQ 总结

DT 值比较、各队列 DT 值更新和更新被调度队列 DT 值这三个阶段都完成后，一次调度过程才算完成。在报文调度过程中这三个阶段是不断重复的，WFQ 会基于队列状态严格按照所配置的 DT 值去调度队列。

WFQ 算法中实现关键与实现难点在于 DT 值的比较逻辑电路。由于 DT 值的比较用组合逻辑实现，参与比较的队列越多组合逻辑级数就越多，在只要四个队列的时候影响还不是很大，当队列数目很大的时候无论是时序还是电路面积都是无法接受的。此外，DT 值的位宽也限制了电路的实现。DT 值越大可以实现的调度比例精度就越大，但是位宽的增加将使组合逻辑急速增大，大大增加比较逻辑电路实现的难度。在使用 WFQ 算法时，队列数目与 DT 值大小是不得不考虑的两个因素。

4.3 WFQ 算法的改进

通信网的迅速发展使得拥塞控制算法不断推陈出新，WFQ 能提供较好的公平性、时延特性，在资源允许情况下 WFQ 调度算法是首选。在分析 WFQ 的基础上，参考文献[25]提出了 WFQ 的不足和不精确性，以及这种不精确性在链路共享和实时服务中可能带来的影响。对分组的时延保证随时间产生波动，这将导致一些通过发送探测数据包利用回馈信息来估计判断信道瓶颈速率的算法，如 Keshav 提出的

基于速率的可用带宽估计算法，由于这种不精确性而不可用。J.C.R. Bennett 和 H. Zhang 提出了一个最坏情况下的公平度量因子（Worst-case Fair Index，简称 WFI）参数来度量这种时延导致的不精确性。在分层的链路共享的分组公平调度系统中，服务器需要尽可能小的 WFI 值，而 WFQ 具有较大的 WFI 值[26]，所以 WFQ 在分层的调度系统中实时的时延性能不好。

WF^2Q（Worst-case Fair Weighted Fair Queueing，简称 WF^2Q）即最坏情况下的公平排队是在文献[27]中由 J.C.R. Bennett 和 H. Zhang 等提出的。文献[28]同时给出了 WFQ 和 WF^2Q 的仿真结果，并对算法进行了比较。

4.4 本章小结

本章首先大概介绍了 WFQ 算法的基本原理，然后根据基本原理划分了 WFQ 算法的逻辑框图和设计实现的步骤。接着详细说明了各个子模块的实现过程，包括接口信号、逻辑结构图和伪代码实现，对 DT 值比较模块给出了不同的实现结构，并对设计中需要注意的问题进行说明。详细说明 WFQ 逻辑电路实现之后总结说明 WFQ 算法实现的难点与关键点。最后介绍业界所提出的对 WFQ 的改进方法。

第五章 WFQ 调度算法的验证

随着 IC 产业的不断发展，集成电路设计向大规模和超大规模发展，这给设计效率提升带来了巨大挑战。一直以来，电子设备的质量很大程度都取决于功能验证过程的好坏以及所需要的时间。不完全的功能验证无论什么原因都可能导致差的产品质量，甚至产品的失败。芯片的有效验证逐渐成为了 ASIC 设计中的关键步骤。现在流行的验证方法主要有仿真验证和 FPGA 验证。仿真验证由于其施加的激励具有可控性，所以测试的覆盖率广、监测点多和便于观察内部节点的信号，其缺点就是仿真速度较慢。

5.1 Vera 与验证平台的简介

Verilog、VHDL 是硬件描述语言（Hardware Description Language，简称 HDL），而 Vera 是硬件验证语言（Hardware Verification Language，简称 HVL），它是 Synopsys 公司开发的一种直观的、面向对象的高级语言，专门为了满足功能测试的独特需求而开发，用于大规模逻辑验证过程中的 Test-Bench 建立与仿真激励产生的语言。Vera 目前专用于 UNIX 工作站仿真系统中，它支持业界主流仿真器：VCS、Verilog-XL、NC-Verilog 等，它既支持 Verilog 设计也支持 VHDL 设计。Vera 则结合了高级语言（C++）和电路描述语言各自的优点，它象 C++一样有更高的抽象层次，能描述复杂的数据结构、协议、算法，又有电路的概念，特别适合于数通、电信类芯片开发过程中的仿真激励产生。它目前已经被 Sun、NEC、Cisco 等广泛使用以验证其实际产品，从单片 ASIC 到多片 ASIC 组成的系统，从定制、半定制电路到高复杂的微处理器以及 SoC 上。此外，由于 Vera 是一种专门的验证语言，它又包含了验证工作所要求的特有特点，如：激励的自动产生、具有一定数据结构的随机数据流的产生、功能强大的并发控制机制、功能覆盖率分析、输出结果的 Self-check 等。

基于 Vera，Synopsys 公司还提出一种系统验证解决方案，该方案基于一种系统分层的设计思想：测试向量通过验证平台连接到待验模块，验证平台是不同测试向量和待验证模块的连接器件。Vera 验证平台与待测设计（Design Under Verification，简称 DUV）之间的简单关系如图 5.1 所示：

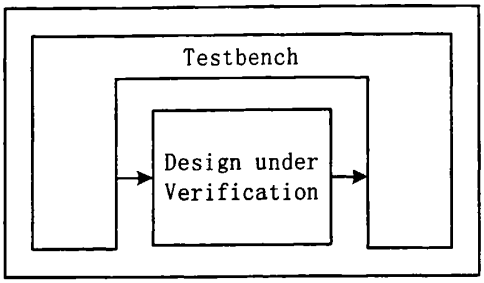

图5.1 Vera 验证平台与 DUV 之间的关系示意图

Test-Bench 通常包含以下几个部分：参考模型（Reference Module，简称 RM）、总线功能模型（Bus Function Module，简称 BFM）、数据流激励（Traffic）、配置激励（Command）、结果比较（Compare）和测试向量（Test Case），图 5.2 显示了 testbench 与 DUV 之间的结构关系。

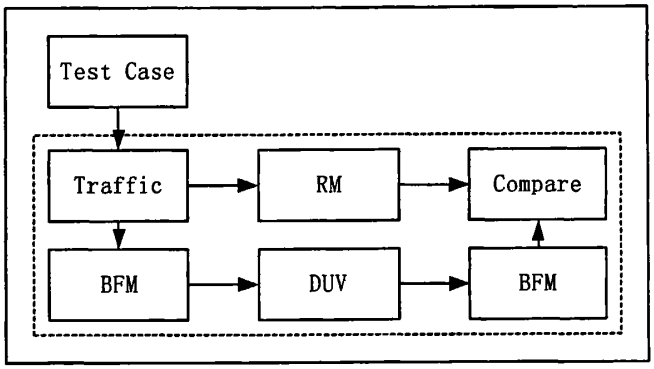

图5.2 Testbench 与 DUV 结构示意图

5.1.1 验证流程

仿真验证是指在软件的环境下模拟硬件电路工作的实际情况，在近似实际应用环境的情况下，完成对硬件电路设计的功能测试，从而提高芯片设计的成功率。也就是说用仿真验证来检测一个设计是否达到设计要求，并尽可能早的发现设计中的缺陷。图 5.3 说明了验证流程的主要步骤。

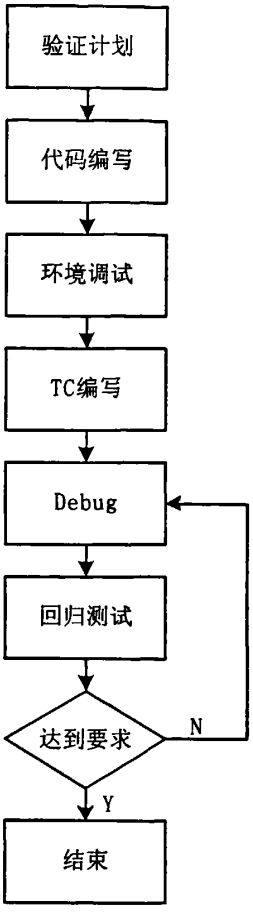

图5.3 验证流程图

5.1.2 验证环境的设计

验证环境其实就是要模拟所做的设计在实际应用环境下。但是也必须知道，在实际验证平台实现时，整个的仿真测试平台不必是一个集成的模块，可以是一个很大的框架，里面的每一个模块对应的可能就是设计或者验证环境的一个功能实体。验证环境包括以下四个要求：

（1） 完备性：

验证环境设计与验证目的直接相关，满足验证目的、能够执行相关的测试向量是设计验证环境的宗旨。验证环境设计不周全完备，测试自动化难于实现，测试向量也难于执行。

（2） 层次化：

设计越来越复杂，规模也越来越大，相应地验证环境也越来越复杂。通过分

层设计，可以在简化验证环境设计的同时，完成比较复杂的功能，也便于重用。

（3） 自动化：

验证环境要尽可能的自动化。我们要测试的系统是一个非常复杂的系统，包含成千上万种情况，没有自动化的工具是不可想象的！除非自动化的难度非常大，投入非常高。

（4） 重用：

系统原型建模环境、单元验证环境、集成验证环境、系统验证环境、FPGA 原型验证环境、样片测试环境和应用环境测试环境各不相同，其结构、实现方法都有很大差异。但其中的很多功能都比较一致，可以在不同验证环境中进行重用，减少重复开发的工作量，大大提高搭建验证环境的效率。在设计一个验证环境的某个模块时，要时时考虑是否有其他模块可以重用，并考虑现在的设计是否可以被其他验证环境或其他项目重用。

5.1.3 验证的目的

验证首先需要知道待验证设计所要实现的功能，然后通过仿真验证去测试待验证设计是否实现要求的功能规格。WFQ 实现的功能就是根据配置的权队列重，对四个队列进行调度。考察 WFQ 设计是否实现所要求的功能，就需要通过验证环境测试 WFQ 的调度结果是否符合设计要求。除了 WFQ 的基本功能验证外，还要对一些异常情况进行验证，覆盖常规与异常两种情况，尽可能与实际应用场景趋于一致。

5.2 WFQ 模块的验证

5.2.1 测试点分解

验证一个模块时，需要对整个模块的功能、性能点进行验证，对于 WFQ 模块的验证，主要考虑从以下几个方面进行：

（1） WFQ 的调度符合 DT 表的配置；

（2） 配置某一队列 DT 值为 0 时 WFQ 的调度情况是否为该队列绝对优先；

（3） 队列为空时 WFQ 的调度情况；

（4） 在运行过程中修改 DT 值表后 WFQ 是否正常工作；

5.2.2 WFQ 验证环境的搭建

对于验证环境来说，它只需要知道 DUV 的两个方面的信息：首先是 DUV 的功能，这是验证的对象；然后就是 DUV 的接口，这是环境与 DUV 通信的通道。表 5.1 说明了 WFQ 的接口信号。在这里用 10bit 来表示一个队列的状态，表示任一队列最多缓存 1k 个报文；用 5bit 来表示一个队列的调度 DT 值，表示正常情况下任一队列在 32 次调度中可以得到的调度机会。

表5.1 WFQ 的接口信号

<table>
<tr><th>信号名</th><th>方向（I/O）</th><th>位宽（bit）</th><th>信号说明</th></tr>
<tr><td>clk</td><td>I</td><td>1</td><td>时钟</td></tr>
<tr><td>reset</td><td>I</td><td>1</td><td>复位信号</td></tr>
<tr><td>q_sta0</td><td>I</td><td>10</td><td>队列 0 的状态</td></tr>
<tr><td>q_sta1</td><td>I</td><td>10</td><td>队列 1 的状态</td></tr>
<tr><td>q_sta2</td><td>I</td><td>10</td><td>队列 2 的状态</td></tr>
<tr><td>q_sta3</td><td>I</td><td>10</td><td>队列 3 的状态</td></tr>
<tr><td>dt_table</td><td>I</td><td>20</td><td>队列 0～3 的 DT 值表；
<table>
<tr><td>比特</td><td>19：15</td><td>14：10</td><td>9：5</td><td>4：0</td></tr>
<tr><td>代表队列</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table></td></tr>
<tr><td>ptr_out</td><td>O</td><td>2</td><td>队列号调度输出</td></tr>
<tr><td>ptr_out_vld</td><td>O</td><td>1</td><td>调度输出有效信号</td></tr>
</table>

（1） Vera 的顶层（Test Harness）

Test Harness（简称 TH）是 Vera 的顶层，也就是仿真验证平台的模拟硬件环境。这一部分是对整个验证平台的实例化，将所有的验证模块组合起来，组成一个真正的仿真测试环境。

首先要例化 DUV：

```
WFQ   U_WFQ(
    .clk (clk),
    .reset   (reset),
    .q_sta0 (q_sta0),
    .q_sta1 (q_sta1),
    .q_sta2 (q_sta2),
    .q_sta3 (q_sta3),
    .dt_table (dt_table),
    .ptr_out(ptr_out),
    .ptr_out_vld (ptr_out_vld),
                );
```

然后将 DUV 与环境的接口连接上：

```
main    main_shell(
    .\wfqbfm_if.clk  (clk),
    .\wfqbfm_if.reset   (reset),
    .\wfqbfm_if.q_sta0 (q_sta0),
    .\wfqbfm_if.q_sta1 (q_sta1),
    .\wfqbfm_if.q_sta2 (q_sta2),
    .\wfqbfm_if.q_sta3 (q_sta3),
    .\wfqbfm_if.dt_table (dt_table),
    .\wfqbfm_if.ptr_out(ptr_out),
    .\wfqbfm_if.ptr_out_vld (ptr_out_vld),
                );
```

完成 DUV 接口与验证环境接口间的连接后，才能将测试激励发进 DUV 进行检测。

（2） 界面（Interface）

Vera 是一种高级测试工具，它通过在 DUV 的输入施加激励，在输出端口采样信号，来达到测试 DUV 的目的。所以像其他的仿真工具一样，Vera 和 DUV 之间必定有一种传输机制来达到信号的转发。这里主要说明 interface 和信号的定义。

```
interface    wfqbfm_if
{
  input   clk          CLOCK,
  input   reset        INPUT_EDGE INPUT_SKEW;
  input   q_sta0       INPUT_EDGE INPUT_SKEW;
  input   q_sta1       INPUT_EDGE INPUT_SKEW;
  input   q_sta2       INPUT_EDGE INPUT_SKEW;
  input   q_sta3       INPUT_EDGE INPUT_SKEW;
  input   dt_table     INPUT_EDGE INPUT_SKEW;
  output  ptr_out      OUTPUT_EDGE OUTPUT_SKEW;
  output  ptr_out_vld  OUTPUT_EDGE OUTPUT_SKEW;
}
```

其中，INPUT_EDGE 和 OUTPUT_EDGE 两个宏分别定义了输入采样和输出驱动使用时钟沿的类型：上升沿驱动为 PHOLD，下降沿驱动为 NHOLD，上升沿采样为 PSAMPLE，下降沿采样为 NSAMPLE；INPUT_SKEW 和 OUTPUT_SKEW 两个宏则分别定义了采样和输出的 SKEW，时间缺省为 1 个时钟单位。

（3） 总线功能模型（BFM）

BFM 模拟接口模块的行为，对设计的接口进行驱动和采样，供上层的验证组

件调用。BFM 在实现的时候一般都提供 Monitor 的功能，即检测被测设计的接口时序或者数据格式是否满足接口协议。由于 BFM 直接与被测设计连接，因此 BFM 需要处理被测设计的各种输入、输出信号时序。一般来说，被测设计具有什么样的接口，就应该有相应的 BFM 与它对应，如图 5.4 所示：

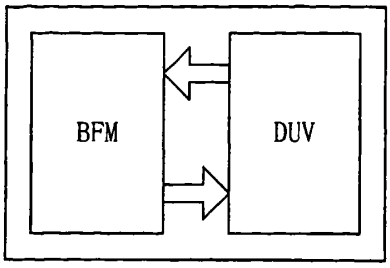

图5.4 BFM 与 DUV 结构示意图

使用 BFM 的一个很明显的优势就是使验证平台具有可重用性，它常常是和一些能够在不同的设计中重复使用的 IP 模块联系起来。Vera 中使用 BFM 相对于传统验证语言来说，优势在于它的灵活性和层次性。在 Verilog 的使用中，在主程序里调用 BFM 时必须是整体调用，而不能随意选取其中某一块来对其进行操作，重用性就很难发挥出来。而在 Vera 中，将 BFM 定义为一个类，在这个类中有完成各种功能的 task 或 function。在一个 BFM 的内部，各个 task 和 function 之间可以相互调用，非常灵活。在主程序中，调用 BFM 时可以根据自己的需要来调用有用的 task 或 function，这样就无需定义多个 BFM，减少了很多工作量，也增加了程序的可读性。

在实现 BFM 时，所有的功能都用 task 或者 function 来描述，具体的物理信号的时序（哪个是时钟信号、上升沿还是下降沿有效、相对于有效沿的什么时刻采样信号、在定义新的驱动值之前有多长时间的保持时间等）这些都可以在统一的一个接口文件中独立定义。所有的接口信号被定义在同一个独立的虚拟接口文件中，通过虚拟接口的绑定，功能描述可以直接驱动和采样接口信号，而不用去关心详细的时序。为了模拟 WFQ 的接口模块行为，设计一个 WFQBFM 主要提供 WFQ 的输入和接收 WFQ 的输出，还有对输出进行比较。在 WFQBFM 里定义 BFM 的端口和测试所需要的 task 或 function。 下面大概说明 WFQBFM 的实现步骤和主要部分。首先需要声明 WFQBFM 的输入输出端口：

```
port    P_WFQBFMPort
{
  clk      ;
  reset    ;
  //output
```

```
    q_sta0   ;
    q_sta1   ;
    q_sta2   ;
    q_sta3   ;
    dt_table;
    //input
    ptr_out ;
    ptr_out_vld;
}
```

一般来说设计的输出就是 BFM 的输入，设计的输入则应该是 BFM 的输出，对于时钟与复位信号，有可能存在环境与 DUV 同步使用的情况，所以由环境提供。

为了将 WFQBFM 捕捉到的 WFQ 的输出信号保存下来，以便作比较，在这里使用了 Vera 所提供的邮箱 mailbox。mailbox 可以理解为一个具有无限深度的 FIFO，无需担心溢出，并且可以得到所捕捉到的信息顺序，为测试带来了极大的便利。mailbox 的定义方法如下：

```
integer    sd_mb;
sd_mb = alloc(MAILBOX,0,1);
```

使用语法 alloc(MAILBOX,0,1)生成了一个名为 sd_mb 的 mailbox，用来存放 WFQ 的输出即调度信息。在使用的过程中不同的文件可以通过模块间的传递使用这个 mailbox。

为了实现 BFM 驱动和采样的功能，可以通过在 BFM 中设立的 task 或 function 往 DUV 发送测试激励、输出采样和对 DUV 进行配置。正常工作的时候，WFQ 需要得到四个队列的队列状态，还有各队列的 DT 值。可以在 WFQBFM 里封装这样一个任务：

```
task set_wfq(bit[9:0] qsta3, bit[9:0] qsta2, bit[9:0] qsta1, bit[9:0] qsta0, bit[19:0]
dt)
{
    //set the state of all the queue
    this.m_WFQBFMPort.$q_sta0 = qsta0;
    this.m_WFQBFMPort.$q_sta1 = qsta1;
    this.m_WFQBFMPort.$q_sta2 = qsta2;
    this.m_WFQBFMPort.$q_sta3 = qsta3;
    this.m_WFQBFMPort.$dt_table = dt;
}
```

通过“this.m_WFQBFMPort.”语法将任务的入口参数直接赋给 WFQ 的输入，

实现测试激励的发送，同时还配置了WFQ各个队列的DT值。通过修改任务set_wfq的入口参数达到设置不同队列状态和DT值的测试。

```
task cap_data(time)
{
  integer   sd_cnt;   // the time of schedule
  while(1)
  {
    @(posedge m_WFQBFMPort.$clk);

    //capture the output of WFQ module
    sd_vld   = this.m_WFQBFMPort.$ptr_out_vld;
    sd_qnum = this.m_WFQBFMPort.$ptr_out;

    if(sd_vld == 1'b1)
    {
      mailbox_put(sd_mb,sd_qum);  //save the queue number of WFQ
                                  //scheduling
    sd_cnt++;
    }
    if(sd_cnt == time)
    break;
  }
}
```

通过任务 cap_data，WFQBFM 实现另外一个功能，就是捕获 WFQ 的调度输出，为了模拟实际应用场景，在这里使用 while(1)来模拟 WFQ 一直工作的状态。从上面的代码可以看出，每个一个时钟周期只要 WFQ 的调度有效，就把 WFQ 的调度结果即调度的队列号用 mailbox_put 的命令放到之前定义好的邮箱 sd_mb 里面去，实现了输出信息的存放。使用入口参数 time 来限制测试的次数，当收集到的信息达到要求就结束调度信息的捕捉。

（4） 测试向量（Test Case）

测试向量（Test Case，简称 TC）是整个环境的最顶层。测试向量通过调用 BFM 中封装的任务或者函数，或者甚至调用更底层的任务、函数，在某种配置情况下对 WFQ 进行仿真测试。它包括：产生数据激励，完成配置，发送数据，比较捕捉到的数据，通过判断自动比较结果以及其它的一些测试结果得出仿真是否通过等几个步骤。TC 与 BFM 还有 DUV 的关系如图 5.5 所示：

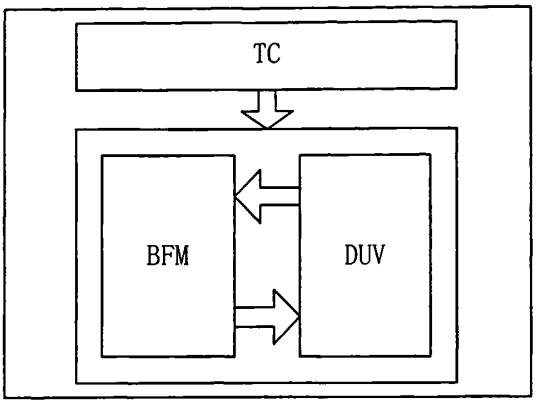

图5.5 TC、BFM 与 DUV 的连接关系图

每条测试向量都是以一个函数的形式存在的，它的原型为：

function integer testcase（）

当测试向量成功执行完之后，它返回一个整数型数据 0，否则认为是非正常退出。

下面介绍测试向量基本步骤

```
function   integer   TestCase()
{
  integer     sd_cnt0; //the times of scheduling queue 0
  integer     sd_cnt1; //the times of scheduling queue 1
  integer     sd_cnt2; //the times of scheduling queue 2
  integer     sd_cnt3; //the times of scheduling queue 3
  integer     mb_len; //the length of mailbox
  integer     i;
  integer     sd_qnum; //the queue num of WFQ scheduling

  //initialization
  sd_cnt0 = 0;
  sd_cnt1 = 0;
  sd_cnt2 = 0;
  sd_cnt3 = 0;
  mb_len   = 0;
  //send stimulation and set the DUV
  set_wfq (10'd500, 10'd500, 10'd500, 10'd500,{5'd6, 5'd10, 5'd4, 5'd12});
  // statistic and calculation
  cap_data(320);
```

```
    mb_len = mailbox_get(COPY_NO_WAIT,TH.m_WFQBFM.sd_mb);
    for(i=0;i<mb_len;i++)
    {
      sd_qnum = mailbox_get(NO_WAIT,TH.m_WFQBFM.sd_mb);
      case(sd_qnum)
      {
        2'd00:    sd_cnt0++;
        2'd00:    sd_cnt1++;
        2'd00:    sd_cnt2++;
        default:  sd_cnt3++;
      }
    }
  }
```

在这条测试向量中，首先定义了测试需要的变量；然后调用环境任务，赋值给 WFQ 的输入信号即各队列的队列状态，同时还配置了各队列的调度比例（此处只是给予了一个参考值）。只捕捉 WFQ 320 次调度。使用 Vera 提供的 mailbox_get(COPY_NO_WAIT, XXX_MB.XXX)可以得到 mailbox 的长度，也就是捕获调度的个数，然后将存放在 mailbox 里面的信息全部取出来分别统计每个队列的调度次数。至此，就完成了测试激励的发送、DUV 的配置、输出的捕捉以及输出的统计等步骤，剩下的就是对统计进行比较，检测每个队列的调度次数与配置的 DT 值之间的关系是否相一致。

上面仅仅是说明各 TC 都具备的基本步骤，然而要达到更好的测试效果，就需要覆盖尽可能多的测试点与测试方式。除了上文所提到的所有测试点都应该测试完全之外，还需要增加大流量测试。

一般来说设计中的功能都不会只有很少的几种，这样设计中的功能组合就会有很多种组合情况，就需要为环境添加更多的函数或任务。不同的灵活的测试向量可以检测到 DUV 的不同方面，大量的测试向量对 DUV 的冲击测试更是可以发掘出隐藏在 DUV 里面的 bug，对验证尤为重要。所以测试向量的设计要遵循以下四个原则：

1） 测试向量设计的针对性：

测试向量是针对测试点而设计的，为验证计划服务。

2） 设计自动化程度高的测试向量：

自动化的测试向量设计成本会高一些，但执行效率高、出错几率低，回归测试容易。减少人机交互、界面操作和人工干预可以提高测试向量的自动化程度。

3） 控制测试向量的复杂程度：

过于复杂的测试向量设计成本高，调试困难，出错几率高。但利用分层思想、结合高效的验证工具和方法，可以通过简单的结构实现复杂的测试向量，完成复杂的测试。

4）针对验证特性，设计相应的测试向量在不同的验证环境中执行：

功能特性、接口特性的测试向量尽量在仿真环境中执行，性能特性的测试向量尽量在 FPGA 验证环境中执行。

（5） 测试向量的执行

测试过程中首先执行基本的测试向量：在初始验证阶段，设计中往往存在很多缺陷，很可能导致一个测试向量都无法执行。执行简单的、基本的测试向量可以验证设计的基本功能是否已实现、接口连接是否基本正确。这类测试向量的执行结果，可以决定是否正式启动某个验证阶段。将环境或代码中的一些明显问题修改后执行自动激励的随机测试向量，可以最大程度地得到高的覆盖率，最快地暴露设计的缺陷。对于一些特殊特性在常规测试过程中没有被覆盖到，那就需要执行受约束的随机测试向量，这类用例是针对未覆盖的验证特性而设计的，约束越多，针对性越强，约束越少，针对性越弱。而执行直接测试向量，这类用例具有很强的针对性，直接针对具体的验证特性而设计。在测试过程中如果发现了设计中的缺陷并修改代码后要进行进行回归测试，回归测试时先执行覆盖率高的测试向量，再执行覆盖率低的测试向量。对覆盖率没有贡献的测试向量应该剔除而不再执行，减少仿真验证时间。

5.2.3 验证结果

（1） 测试向量执行结果

本文所采用的仿真软件是 Synopsys 公司开发的 VCS，版本为 2006.06SP2。根据上面说明的测试向量基本步骤，对于本章第 5.2.1 节中功能点的验证做了以下检测：

1) 通过 WFQBFM 的任务 set_wfq 配置队列 0～3DT 值为 4：5：1：6，保持四个队列的状态为非空，最后检测到在 WFQ 模块的调度中四个队列的调度顺序与表 4.1 一致；
2) 通过 WFQBFM 的任务 set_wfq 配置队列 0～3DT 值为 4：5：1：6，只有队列 0 为空，并在测试过程中一直保持这个状态，最后检测到 WFQ 一直没有队列 0 的调度；
3) 通过 WFQBFM 的任务 set_wfq 配置队列 0～3DT 值为 0：5：5：6，保持四个队列的状态为非空，最后检测到在 WFQ 模块的调度只有队列 0 的调

度；

4) 通过 WFQBFM 的任务 set_wfq 配置队列 0～3DT 值为 0：5：5：6，只有队列 0 为空，并在测试过程中一直保持这个状态，最后检测到 WFQ 一直没有队列 0 的调度；
5) 通过 WFQBFM 的任务 set_wfq 配置队列 0～3DT 值为 0：5：5：6，保持四个队列的状态为非空，检测到 WFQ 一直调度 0 队列；测试一段时间后修改 DT 值配置为 4：5：1：6，等一段时间后（需要等到 WFQ 输出稳定后才开始进行检测），最后检测到在 WFQ 模块的调度中四个队列的调度顺序与表 4.1 一致；
6) 所有的队列 DT 配置成 0，各个队列的长度都为 20，检测到 WFQ 的输出先将队列 0 调度完后再调度队列 1 然后才是队列 2 和 3；
7) 随机测试：将各队列的队列状态与 DT 值配置成随机值，测试结果由 TC 自动比较，连续测试 20 次；
8) 大流量测试：将队列设置为最大容量，在测试期间修改 DT 值，在 WFQ 在新的 DT 值下稳定工作的时候捕捉 WFQ 调度输出，TC 自动比较检测结果，最后检测到所有队列都为空，说明所有报文都调度完毕。

（2） 覆盖率分析结果

在执行测试向量对 WFQ 进行功能验证的同时，需要对 WFQ 进行功能覆盖率分析和代码覆盖率分析，其中代码包括行覆盖率和条件覆盖率。

功能覆盖率相当于从顶层的角度去分析测试向量的功能覆盖情况。实现的基本过程是：首先提取 WFQ 模块的功能点，保证该模块的每个功能点都被正确并且完备的提取；然后构造不同的激励产生多条测试向量以便覆盖到所有的功能点；从验证环境中搜集相关的数据，仿真完成后得到相应的统计结果。由于对 WFQ 每个功能点都设置了专门的测试向量，所有测试向量执行完后，统计得到所有的功能点都已覆盖。

行覆盖率表示代码行的执行情况，检查每一行代码是否都已经执行，没覆盖到的代码无法保证其正确性。图 5.6 为仿真平台实现的 WFQ 模块的代码行覆盖率分析报告，说明 WFQ 模块达到了 100%行覆盖率的要求。

	Instance	Module	Blk C(%)	Line C(%)	Stmnt C(%)
	U_WFQ	WFQ	100.00	100.00	100.00

图5.6 WFQ 模块代码行覆盖率

条件覆盖率表示代码中条件语句的执行情况，工具会自动统计每条向量对所有条件的覆盖情况，如果统计的条件覆盖率结果不能达到 100%，则表明输入的激励使得该条件无法触发，对于无法覆盖的条件需要进行分析并给出说明。在代码

中一般都存在一些对异常情况的判断和处理，这部分代码并不一定都能仿真测试到，对于这种情况就需要特别说明并与进行分析确认。图 5.7 为仿真平台实现的 WFQ 模块的代码条件覆盖率。

	Instance	Module	Total C(%)	Logical C(%)
	U_WFQ	WFQ	100.00	100.00

图5.7 WFQ 模块代码条件覆盖率

5.3 本章小结

本章对验证语言 Vera 与验证平台作了简单介绍，随后介绍了验证流程以及验证环境的结构与验证环境设计的一些要求。针对 WFQ 验证详细说明了验证环境搭建的具体步骤，包括验证环境的顶层、BFM 与测试向量的设计。同时指出了验证环境搭建需要注意的问题，测试向量设计以及测试向量执行的一些原则。

第六章 结束语

市场的巨大驱动力使 Internet 正面临着新的机遇与挑战。提供满意服务质量的应用已经成为 Internet 当前和未来发展的重要方向。高性能的网络服务设备也正经历这快速的发展，同时也面临着多项关键技术的挑战。关于队列调度算法的研究一直是一个热点问题，人们通过系统仿真的方式对各种队列调度算法进行分析研究，从而产生各种各样的调度算法。

本文对网络设备中常用的排队结构以及常用的调度算法进行了分析和研究，介绍了 WFQ 算法的基本原理，以四输入 WFQ 电路为例详细介绍了 WFQ 模块的组成结构以及实现过程，并对各子模块的电路设计实现做了详尽的阐述。然后介绍了基于 Vera 的验证环境的搭建以及 WFQ 模块的验证过程，主要介绍了 WFQ 模块验证环境主要步骤，以及验证中该遵循的一些原则。

由于时间和水平的限制，本论文还存在下面几个方面的不足之处：1、主要研究定长分组调度算法，但实际网络中变长分组仍然是通信的主要组成部分；2、本文使用的流量模型较小，队列数目比较少，仅仅说明算法的基本原理，但是与真实的网络环境还有差距。

在下一步的工作中，考虑增加 WFQ 处理的队列数，提高算法实现的复杂度与可应用性。分析多队列条件下进行使用 WFQ 的调度效率，以及实现 WFQ 的时序要求和综合面积，并进行实际应用。

随着人们对网络提供业务种类和服务质量要求的上升，调度起着越来越重要的作用，而调度算法的研究也就有着越来越重要的研究价值和现实意义。

致 谢

首先感谢我的导师蔡觉平副教授在整个研究生阶段对我的悉心指导和严格要求。正式在蔡老师的引导下，我才可以在芯片设计领域中进行探索。蔡老师严谨的治学作风、求实的科学态度和高昂的工作热情，所有的这些都将使我受益终身。论文期间，蔡老师在研究方向和研究方法上给予我极大的指导和帮助，为此，谨向导师致以诚挚的谢意！

非常感谢海思半导体有限公司的指导老师宋玲玲和吴振耀，他们无论是学习、工作还是生活都给予了我极大的指导和关怀！

感谢微电子学院的各位领导以及海思半导体有限公司的领导，感谢他们为我提供了良好的学习、实习环境。

感谢海思半导体有限公司 HiXXXX 项目组的全体成员，在实习期间他们对我的工作和学习给予了很大的帮助。尤其是张忠琴与陶利春，给予了我很多帮助，教予我很多东西。诚挚的感谢项目组的全体兄弟姐妹们，他们在我最需要帮助的时候给予了我大力的支持。

感谢我的同学们：姚正茂、施旺根、刘兴旺、刘俊、王宇楠等，感谢他们在研究生期间给予我生活、学习上的帮助，感谢他们给我带来的欢乐和真诚的友谊。

最后由衷的感谢我的父母，他们含辛茹苦地养育了我，培养了我，使我能幸福地成长。感谢他们多年来对我无私的奉献，感谢他们对我的支持和信任，并支撑我完成漫长而艰苦的学业。在此，向他们致以最最衷心的感谢和祝福。

参考文献

[1]. BREDENR, CLARKD, SHENKERS. Integrated Services in the internet Architecture and Overview, IETF RFC 2475[S].1994.

[2]. BLACKD, BLAKES, CARLSONM, et al An Architecture for Differentiated Services, IETF RFC 2475[S].1998.

[3]. Sariowan H, Cruz RL, Polyzos GC. Scheduling for quality of service quarantees via service curves. In: Proc Int't Conf on Computer Communication and Networks (ICCCN) 1995. Sept 1995. 512-520.

[4]. Abhay K. Parekh. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case. In: IEEE INFOCOMM'92. IEEE Computer Society Press, 1992. 915-924.

[5]. Fabio M. Chiussi, Andrea Francini. Scalable Electronic Packet Switches. IEEE Journal on Selected Areas in Communications, 2003, 21(4). 486-500.

[6]. Bradeb B, et al. Recommendations on queue management and congestion avoidance in the Internet. RFC 2309. 1998.

[7]. Floyd S, Jacobson V. Random early detection gateways for congestion avoidance. ACM/IEEE Transactions on Networking. 1(4),1993.297-413

[8]. 林闯，单志广，任丰原 . 计算机网络的服务质量（QoS）第一版. 北京：清华大学出版社.2004. 219-225.

[9]. A. Varma, D. Stiliadis. Hardware implementation of fair queuing algorithms for asynchorous transfer mode network [J], IEEE Communication Magazine, December 1997. 54-67.

[10]. D. Stiliadis, A. Varma. A general methodology for designing efficient traffic scheduling and shaping algorithm [A]. IEEE GLOBECOM'97[C], 1997. 519-526.

[11]. S.J.Golestani. A self-clocked fair queuing scheme for broadband applications[C], 1997. 326-335.

[12]. R.Bennett, H.Zhang. WF^2Q: Worst-case fair weighted fair queuing[C].IEEE INFOCOM'96, Mar. 1996. 120-128.

[13]. Peterson L L, Davie B S. Computer Networks: A System Approach, 2ed, Morgan Kaufman. 2000. 457-457.

[14]. 林闯，单志广，任丰原.计算机网络的服务质量（QoS）第一版.北京：清华大学出版社.2004. 214-215.

[15]. Karol M., Hluchyj M., Morgan S. Input Versus Output Queuing on a SpaceDivision Packet Switch Communications. IEEE Transactions on Communications, 1987, 35(12). 1347-1356.

[16]. McKeown N., Mekkittikul A., Anantharam V., et al. Achieving 100% Throughout in an Input-Queued Switch. IEEE Trans. Commu., 1999, 47(8). 1260-1267.

[17]. Mekkittikul A. McKeown N. A Fair Algorithm for Achieving 100% Throughput in an Input-Queued Switch. in Proceedings of IC3N 96, Oct. 1996. Maryland.

[18]. S. Iyer and N. McKeown, "Using constraint sets to achieve delay bounds in CIOQ switches,"IEEE Communications Letters.June,2003, 7(6). 275-277.

[19]. A. L. Gupta and N. D. Georganas, "Analysis of a packet switch with input and output buffers and speed constraints" in Proc. IEEE INFOCOM' 91, Bal Harbour, Florida. Apr, 1991, 2. 694-700.

[20]. J. S. C. Chen and T. E. Stern, "Throughput analysis, optimal buffer allocation, and traffic imbalance study of a generic nonblocking packet switch" IEEE Journal on Selected Areas in Communications. Apr, 1991, 9 (3). 439-449.

[21]. Chuang S.T., Goel A., McKeown N., Prabhakar B. Matching Output Queueing with a Combined Input Output Queued Switch. IEEE Journal on Selected Areas in Communications, 1999, 17(6). 1030-1033.

[22]. Kounavis M E, Campbell A T, etl. The genesis kernel: A programming system for spawning network architectures. IEEE Journal on Selected Areas in Communications, 2001, 19(3). 511-526.

[23]. 林闯，单志广，任丰原.计算机网络的服务质量（QoS）第一版.北京：清华大学出版社.2004. 219-225.

[24]. Keshav S. On the efficient implementation of fair queuing. Internetworking: Research and Experience, 1991, 2(9). 234-239.

[25]. R. Cruz. Service burstiness and dynamic burstiness measures: A framework. Journal of High Speed Networks, 1992, 1(2). 105-127.

[26]. Jon C.R. Bennett and H. Zhang, Hierarchical Packet Fair Queueing Algorithms. IEEE/ACM Transactions on Networking, Oct 1997, 5(5). 675-689.

[27]. Benny Amrami. WFQ and WF^2Q. Course: Topics in MultiProcessing. 2001.

[28]. C.R.Bennett, H.Zhang. WF^2Q:Worst-case Fair Weighted Fair Queueing. Proc of IEEE INFCOM'96. San Francisco. U.S.A. Mar 1996. 120-128.

队列调度算法的研究与逻辑实现

作者：赖金志
学位授予单位：西安电子科技大学
被引用次数：5次

本文读者也读过(10条)

1. 刘凤格 基于网络QoS的队列调度算法研究[学位论文]2007
2. 王沛 基于IP网络队列调度算法的研究[期刊论文]-大众科技2009(2)
3. 远震东 基于区分服务的队列调度算法研究与仿真[学位论文]2007
4. 尹德斌 IP拥塞控制中队列调度及优化问题的研究[学位论文]2007
5. 马骏.杨功流.MA Jun.YANG Gong-liu 基于服务时间的加权公平队列调度算法[期刊论文]-计算机工程 2009,35(16)
6. 李剑.涂晓东.LI Jian.TU Xiaodong 基于优先级的包插入队列调度算法仿真研究[期刊论文]-现代电子技术 2008,31(2)
7. 汪华斌 TCP拥塞控制中主动队列调度算法研究[学位论文]2008
8. 杨永斌.唐亮贵.YANG Yong-bin.TANG Liang-Gui 队列调度算法在网络中的应用研究[期刊论文]-计算机科学 2005,32(7)
9. 许莹 基于区分服务（DiffServ）的IP QoS控制策略研究[学位论文]2003
10. 刘宴兵.李秉智 具有QoS特征宽带IP网络队列调度算法的研究[期刊论文]-计算机科学2003,30(5)

引用本文格式：赖金志 队列调度算法的研究与逻辑实现[学位论文]硕士 2009