摘 要

作为网络性能测试工具之一，网络模拟器不仅要能够适应网络的快速发展，同时它还要具有易于使用和高效等特点。IPv6 被视为下一代互联网络的核心，Windows 操作系统也被越来越多的人所接受，但目前在相关研究领域中仍然没有一个广泛应用在 Windows 平台下并支持 IPv6 的网络模拟器。

针对这一问题，作者设计并实现了一个可以运行在 Windows 平台上并对 IPv4 和 IPv6 均支持的网络模拟器。本文以目前已经存在的优秀网络模拟器的核心算法为基础，以 Windows 下内核驱动编程为技术手段，通过 WDM 驱动与网络驱动程序接口规范（NDIS）相结合的方法来进行模拟器的构建工作。在设计方面，作者充分考虑到 Windows 操作系统本身的特点，将模拟器的核心模块与用户模块进行分离构建，从而使模拟器能够达到一定的高效性和精确度并仍能够保持良好的易用性；在实现方面，本文所构建的模拟器不仅利用 NDIS 来截获操作系统中较底层的封包，还采用了 WDM 驱动来实现应用层与驱动层的通信功能以及高精度的定时功能，从而能够较好地模拟出 IPv6 网络中的各种网络环境（延迟、丢包、及带宽限制）。

本文使用该网络模拟器在 IPv6 环境下搭建实验床，进行了关于 IPv6 网络性能参数（带宽、延迟和丢包率）的测试工作。通过测试本文发现模拟器的工作效果是令人满意的。实验结果表明，该网络模拟器能够有效地对各种网络参数进行改变和控制，可以较为方便地搭建出研究人员所需要的目标网络，能够成为 IPv6 环境下的测试工作中一个有用的工具。

关键词： 网络模拟 网络模拟器 Windows IPv6 NDIS

ABSTRACT

As one of the evaluation tools, network emulator should not only catch up with the development of the Internet, but also be useable and efficient. IPv6 is considered as the backbone and characteristic of the NGI. And Windows operating system is used by more and more people. But unfortunately, there has no general purpose and widely used network emulator running on Windows platform for IPv6 nowadays.

This paper proposes the design and implementation of a useable and accurate network emulator which supports both IPv4 and IPv6 protocols. It works on Windows platform. Following the base idea of emulation, the implementation of this emulator is on the basis of the reference of the network emulators that already exist. It is implemented with the NDIS and WDM driver. On the design part, considering the characteristic of Windows kernel, this paper guarantees the precision and amiability. On the implementation part, this emulator uses NDIS to capture packets and WDM for timing. By manipulating the packets in data link layer, it can generate various network characteristics and conditions including bandwidth, delay and packet loss.

In this paper the author also builds an IPv6 testbed with the emulator to generate various network characteristics and conditions including bandwidth, delay and packet loss. Through implementation and experimentation study, it has been shown that this network emulator does provide the real-time control and change on the parameters of IPv6 network conditions effectively and expediently on Windows. It also gives enough accuracy and more satisfactory convenience to the development and test work for the new protocols. The author believes that it will be a useful tool for protocols and services testing work under IPv6.

KEY WORDS: Network Emulation, Network Emulator, Windows, IPv6, NDIS

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作和取得的研究成果，除了文中特别加以标注和致谢之处外，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 天津大学 或其他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作了明确的说明并表示了谢意。

学位论文作者签名：赵冰　　签字日期：2007年 6月 14日

学位论文版权使用授权书

本学位论文作者完全了解 天津大学 有关保留、使用学位论文的规定。特授权 天津大学 可以将学位论文的全部或部分内容编入有关数据库进行检索，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复印件和磁盘。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：赵冰　　导师签名：金志刚

签字日期：2007年 6月 14日　　签字日期：07年 6月 14日

第一章 绪论

1.1 网络模拟（Network Emulation）概述

随着网络规模的不断扩大，现在有越来越多的应用程序及协议在网络上运行，而这些应用程序或者协议对网络环境的需求也是千差万别的。测试这些协议或者应用程序的性能本身就是一件非常困难的工作。在网络高速发展的今天，如何准确和全面的测试与评价一个网络的性能，是当前网络性能测试工作面临的一个日益严峻的问题。特别是随着网络技术的发展，新的网络协议大量涌现，网络所提供的服务也日趋多样，因此开发者要想分析和研究它们的性能，不仅需要构建庞大而复杂的拓扑结构，还需要能够可控并可再生网络条件，这要求都对网络性能评价工作提出了巨大的挑战。

在网络技术研究过程中，可供选择的测试、评估和验证的手段主要有三种：

1．构建数学分析模型。此种方法就是对所要研究的对象和所依存的网络系统进行初步分析，根据一定的限定条件和合理假设，对研究对象和系统进行描述，从而抽象出研究对象的数学分析模型[1]。这种方法主要是通过数学推理证明、与现实实例对照或与仿真的结果比较等方法来验证模型的有效性和精确性，最后利用求精后的数学分析模型对问题进行解答。这种方法的优点是具有一定的灵活性，不受软件或者硬件等物质资源的限制，但其模型的有效性和精确性受假设的限制很大[1]。当一个系统很复杂时，无法用一些限制性的假设来对系统进行详细性的描述，所以这种方法比较适用于网络节点协议实现的理论研究及简单的网络行为分析，而不大适用于当前的比较复杂的网络系统。

2．仿真测试（Simulation）。它是指开发或者使用网络仿真软件来搭建网络系统模型，并根据模型运行后输出的结果进行系统分析[1]。这种方法是一种在全人工合成的环境中用代码来描述网络模型的运行过程，被用于这种仿真的软件统称为 Simulator。Simulator 是在一个节点上建立一个虚拟的网络平台，以此来模拟整个网络的业务和行为。网络仿真器（Network Simulator）可以方便的产生一个可控，可再现的网络协议或是网络服务的分析环境，但是这个环境仅仅是在一个网络节点上模拟网络业务和网络行为。目前有许多仿真软件存在，例如 OPNET、NS 和 OMNeT++。

由于仿真是基于模型而非真实对象本身进行实验的，所以仿真的结果不可能完全精确的代表真实的对象[2]。也就是说在仿真测试中，在仿真器 Simulator 上

跟本没有真正的业务流穿过，也没有网络各部分真正的相互影响的逻辑网络。而且，在进行仿真之前要将被分析的网络服务的代码修改成适应仿真器之后才能在仿真器上运行，由此得出的运行结果很可能是会有别于其运行在真实网络的结果，所以软件仿真的最大缺点就是失真。

3．全真测试。此种方法是在真实的网络环境之中，测试人员运行被分析的网络协议或网络服务，以在现实的网络上实现对网络性能、网络协议及网络行为的研究。但这样做的难点在于，现在的网络协议越来越复杂，搭建所需拓扑结构的难度也不断加大，使得此种测试方法的成本不断增加；其次，即使建立起网络拓扑结构也很难产生出为测试所需要的网络环境。这种测试方法虽然能如实的反映网络协议或是网络服务的性能状态，但是因为搭建困难且不便于分析，往往只在实验后期才会采用。

为了更好的适应网络日益增加的多样性以及解决这种多样性所带来的网络性能测试方面的问题，一种被称为网络模拟（Emulation）的测试方法被提出来，它更像是仿真测试和全真测试两种方法的调和，也像是这两种测试方法的结合体。在网络模拟测试中，会在真实的物理网络中运行真正的数据业务流，所以它能提供给研究员一个更加接近真实网络的模拟环境；另外，它还提供给研究者以控制接口以便于动态地调控实验所处的网络条件。除此之外，它还提供给研究者一个可控并可再生的物理网络实验环境：在一个真实的物理网络中，网络模拟能在不需要针对仿真环境重新编码的情况下便可重复地运行真实的业务流，从而在此基础上运行有待分析的协议和服务。

网络模拟继承了仿真测试和全真测试的优点，又克服了它们的缺点，它的产生为网络性能评价技术注入了新的活力。它可以像仿真测试一样，提供可控制的，可重复实现的网络运行环境，但避免了为仿真而重新编写的代码的麻烦；同时它又同全真测试一样如实的反映网络协议运行在物理网络上的种种表现和特征，而且还消除了构建庞大而复杂的网络拓扑结构所带来的困难。因此，网络模拟技术已被广泛地应用到网络性能测试工作中去。

1.2 IPv6 的产生和意义

就如同计算机工业飞速发展一样，因特网在过去的 20 年内经历了巨大的发展。IPv4 作为网络的基础设施被广泛地应用因特网和难以计数的小型专用网络上，虽然它是一个比较成功的协议，但随着网络的快速发展，IPv4 的局限性及缺点被越来越多的暴露出来，这些局限性或者缺点成为促使其进行 IP 协议升级的主要原因[3]：

地址空间的局限性：目前占有因特网地址的主要设备早已由 20 年前的大型机变为 PC 机，而且越来越多的其它设备也连接到因特网上，包括 PDA、手机和传感器等。IPv4 的地址容量显然无法满足当前因特网的业务发展需求，这是促使其升级的主要动力。

性能：从网络的发展历史上看，很明显在性能方面 IP 还有可改进的余地，而且新的网络协议和产品的开发导致修改 IP 的呼声越来越高。所以 IPv4 中一些源自 20 年前或者是更早的设计还能够进一步得到改进。

安全性：在许多情况下，IPv4 被设计为只具备最少的安全性选项，而安全性也被认为由网络层以上的层进行负责。虽然有一些基于 IP 选项关于 IPv4 安全性的机制，但在实际应用中并不成功[3]。

自动配置：对于 IPv4 的节点的配置一直比较复杂，而用户则更喜欢类似于“即插即用”的功能，即将计算机插到网络上然后就可以开始使用。而 IP 主机移动性的增强也要求当主机在不同的网络间移动和使用不同的网络接入点时能提供更好的配置服务，而目前的 IPv4 显然无法满足这些功能。

早在 20 世纪 90 年代，因特网工程任务组（IETF）就开始着手下一代因特网协议 IPng（IP next Generation）的制定工作。1994 年 7 月，因特网工程任务组决定以 SIPP（Simple Internet Protocol Plus）作为 IPng 的基础，同时把地址长度从 32 比特增加到 128 比特，这种新的 IP 协议被称为 IPv6。制定 IPv6 的专家们充分总结了早期制定 IPv4 的经验以及因特网的发展和市场需求，将容量和性能作为下一代因特网协议的重点。IPv6 在地址容量、安全性、网络管移动性以及服务质量等方面有明显的改进，是下一代因特网可采用的比较合理的协议。可以说，IPv6 继承了 IPv4 的优点，摒弃了它的缺点。

1998 年 12 月，草案标准 RFC2460 发布后，IPv6 实际上已经相当成熟，但其在之后的一段时间内并未得到推广和应用。近几年，情况开始发生变化，商用 IPv6 网布设进入议事日程并开始实施。在移动数据通信市场需求拉动下，各国及各方面都加大了向 IPv6 过渡的投资力度，纷纷建立各种规模的 IPv6 实验网，不少厂商也推出了各种支持 IPv6 的网络设备，各种操作系统也都开始支持 IPv6。我国已经有多个正在进行实验的 IPv6 科研和实验网，一些电信运营商也在建立自己的 IPv6 网，而且国家的下一代因特网计划也已经开始实施。可以说，由庞大的 IPv4 网络转换到 IPv6 将是网络协议组历史上最重要的一次升级。

IPv6 的建设和开发向广大的研究和开发人员提出了更高的要求，尤其是 IPv6 协议的很多新特性需要在接近真实的环境下并且进行大量重复性实验的基础上才能进行更好的优化和改进，这些都给 IPv6 下的协议和产品的开发及测试工作带来了新的问题和负担。

1.3 Windows 下的 IPv6

微软已经在主机操作系统上领先多年，其开发出的 Windows 操作系统是一种操作方便、直观性强、功能强大的图形窗口式操作系统，尤其是 Windows XP 整合了 Windows 2000 的强大功能（基于标准的安全性、可管理性和可靠性）以及 Windows 98 和 Windows Me 的最好特性（即插即用、简化的用户界面和创新的支持服务），更加利于用户的使用。

特别是在 20 世纪 90 年代，微软公司伴随着因特网而成长。1998 年微软开始开发用于 Windows NT 和 Windows 2000 平台的 IPv6 协议栈，众所周知的 Windows NT 和 Windows 2000 平台就支持可用于研究、试验以及用于纯学习目的的 IPv6。2000 年微软发行了 Windows 2000 的 IPv6 技术预览（IPv6 Technology Preview）并向因特网团体分发。2001 年，微软做出了一个明确的支持 IPv6 的承诺，在 Windows XP 的主流代码中包括 IPv6 支持。同年，微软在 Windows XP 中打包了 IPv6 支持，IPv6 在 Windows XP Professional、Windows XP Home Edition、Windows XP Pro 是可用的。

不同版本的 Windows 对 IPv6 的支持是相似的，并支持 IPv6 协议的主要特性，如无状态自动配置和某些过渡机制。尤其是对 Windows XP 来说，IPv6 是内置的，用户只要使用命令启动它们就可以了。

1.4 网络模拟器的现状

IPv6 是下一代互联网络的核心，虽然现在已经存在了一些不同工作原理的网络模拟器，但是仍然没有广泛应用于 IPv6 环境下的网络模拟器。因此，大部分关于 IPv6 产品及协议的相关测试工作仍然没有能够有效的进行。

众所周知，Windows 操作系统的各个特点决定了它成为目前应用最广泛，用户人数最多的计算机操作系统。与以上问题相近似的是，目前也没有一个可以应用在 Windows 操作系统中的网络模拟器。相关的开发人员为了使用网络模拟器搭建目标网络，不得不放弃自己熟悉的 Windows 操作系统而改用其它。然而这种对其它操作系统的不熟悉性一定会影响到工作的效率。随着网络测试工作的进一步发展，这两种缺陷势必会成为阻碍网络模拟发展的一个瓶颈。

目前 Windows 下的技术不断成熟，再加之 Windows 本身具有的较好的易用性和兼容性，尤其是 Windows XP 的推出，使得操作系统与 IPv6 的整合性越来越好，这也为网络性能评价工作提供了一个得天独厚的优势。在此条件下，对于一个方便的、在 Windows 平台上并可支持 IPv6 的网络模拟器的需求也更加迫切。

1.5 本文的主要工作及意义

本文以已经存在的几种非 Windows 平台的网络模拟器为基础，在 Windows 平台上对支持 IPv6 协议的网络模拟器进行了构建，主要工作包括如下几个方面：

1. 分析了几种非 Windows 平台网络模拟器的工作原理，并结合 Windows 内核驱动的特点，从中找出一种适合在 Windows 内核中运行的模拟器的框架结构。
2. 使用 Windows 内核驱动来截获数据链路层的网络封包，并在内核中实现模拟器的核心功能算法，达到对 IPv6 网络封包进行操作的目的。
3. 在构建过程中还考虑了一些 Windows 操作系统的特点，使此 Windows 平台的网络模拟器具有同 Windows 操作系统一样的易用性及较好的兼容性等特点。

在完成对此 Windows 平台网络模拟器构建的基础上，本文在以太网中环境下使用此模拟器搭建基于 IPv6 协议的实验床，并用它模拟出各种网络环境如带宽限制、延迟和丢包等以进行模拟器功能的验证工作。

实验结果和测试数据表明，本文所构建的 Windows 平台上的 IPv6 网络模拟器能够达到既定的要求，可以较为精确并快速地产生出研究人员所需要的 IPv6 的目标网络环境，相信它可以为 IPv6 下产品的开发和测试工作带来极大的方便。

1.6 全文安排

本论文共分为六章，第一章为绪论，首先结合相关的文献，对一些概念及其术原理进行了综述，而后结合了当前的模拟器的应用现状分析了作者所要做的主要工作及意义；第二章则对网络模拟及已有的网络模拟器进行更深一步的分析和比较，并指出当前模拟器在应用方面上的问题，并给出了 Windows 平台网络模拟器的设计特点；第三章主要介绍了与模拟器构建相关的技术手段，主要包括 NDIS 驱动以及 WDM 驱动在模拟器构建中的应用；第四章是本论文的重点，这一章首先给出了作者所设计的整个模拟器的体系结构，接着对模拟器的核心定时器队列的构建做出了阐述，最后介绍了模拟器核心功能模块（延迟、丢包及带宽限制）和辅助功能模块（封包截获和封包匹配）的算法设计及其实现；第五章是对此模拟器核心功能的测试，本章在 IPv6 的环境下应用此模拟器搭建相应的实验床，进行了关于模拟器核心功能部分的实验，不仅给出了实验结果并对结果进行了一定的分析；第六章是对全文工作的总结，以及在此基础上对整个模拟器系统可深入改进的地方提出展望。

第二章 网络模拟技术

网络模拟概念的提出是为了解决目前网络上所运行的软硬件的日益多样化所带来的在网络性能评测工作方面的困难。它可以使研究人员轻而易举地将那些待测试的代码转换到真实的网络环境中去运行。

2.1 网络模拟的概念

网络模拟(Emulation)在这里可以理解为测试网络性能的两种实验性方法(仿真测试和全真测试)的集成，它是一种在半人工的环境中来运行真正代码的网络测试方法[4]。其中半人工环境是指运行了真正的网络执行，但其所提供的网络延迟和其环境是人工制造出来的。与仿真测试和全真测试这两方法相比较，网络模拟可以提供很多的优点。因此，应用它可以最大的减少网络测试的投入代价。另外，对于在实验室进行研究的人员，这个工具可以使他们在实验室的条件下就能模拟出各种网络环境，而且真正的业务流就在这些需要被测试的协议或者算法上传输，这对于了解网络的各种情况和研究网络的性能是很有帮助的。

目前，网络模拟技术已经被广泛地应用到网络协议及产品的开发、调试及测试的工作中去。这种方法可以用于发现和研究与网络相关的问题或者用于评估已经存在的网络协议或算法的性能[5]。

2.2 常见的网络模拟器（Network Emulator）

近年来，各研发部门和企业生产出多种用于测试网络性能的网络模拟器。其中较有影响的产品如下：

1. Ohio Network emulator (ONE)

ONE 是由俄亥俄大学网络研究小组开发的。它是基于 SUN 工作站的模拟软件，可工作在 Solaries 操作系统上[6]。ONE 有两个物理的网络接口，可以分别连接两个不同的网络。在这两个不同网络间的主机要经过 ONE 才能进行通信，ONE 跟据用户的配置来影响需要转发的网络业务流，从而模拟出需要的网络试验环境和网络条件。如图 2-1 所示，ONE 在这里相当于是一个路由器。

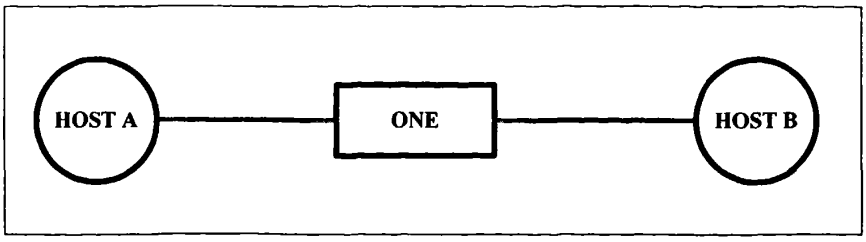


图 2-1 ONE 模拟出的网络拓扑结构

ONE 可以跟据用户配置模拟出网络延迟及基于网络拥塞的丢包机制。另外，它还能提供传输延迟及排队延迟的功能。

2. Dummynet

Dummynet 是一种运行在 FreeBSD 操作系统上的模拟器[4]。它通过截获协议栈中相关协议层之间的通讯来实现对网络条件的模拟。Dummynet 主要通过控制被截获的业务流来模拟出网络的带宽、队列的大小以及延迟和丢包率等环境。Dummynet 的原始版本工作在传输层和 IP 层的接口之间，它的构建是使用 FreeBSD 的核心代码[4]。它截获了由 TCP 模型发起的对函数 ip_output 的调用，也同样截获了在 IP 层由协议的多路输出选择对于函数 tcp_input 的调用，其原理如图 2-2 所示。

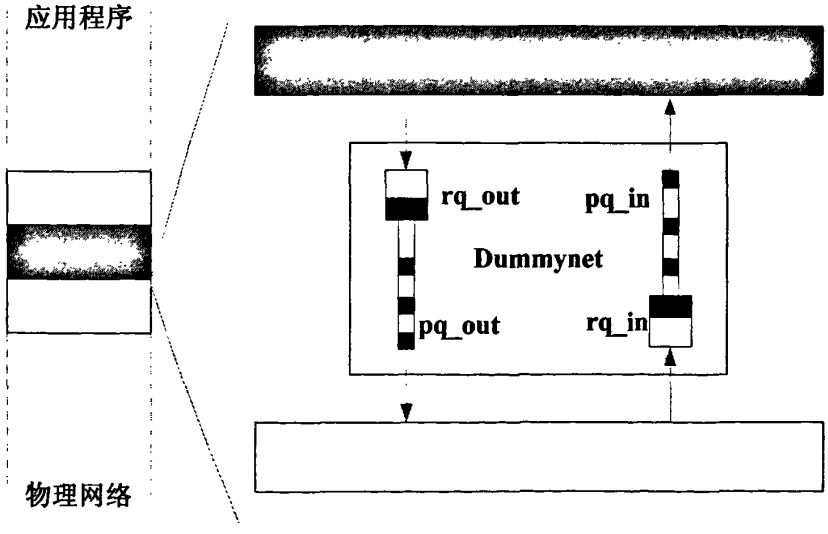


图 2-2 Dummynet 的工作原理

假如有业务流经过 Dummyet 时，比如当一个基于 TCP 的传输发生时，封包会进入事先准备好的队列。实现这个请求队列的管理是由每若干秒便执行的一个周期性工作来完成的。但是需要注意的是，周期性工作的进行是以系统的队列中

有数据为前提的。对于模拟器的目的来说，只有在时间粒度 T 足够小的情况下，Dummynet 才对系统要求具有可用性，而这个时间粒度可以由模拟器或者用户自己来决定，默认的粒度为 10 毫秒。可以调整的时钟粒度对于更高标准的实验或是带宽是非常有用的。

Dummynet 具有全真测试和仿真测试的优点：使用简单，对运行的参数的高度可控制性及高准确性，它不需要复杂的硬件设备，也没有昂贵的模拟成本，而且运行在真实的业务流之上。但是 Dummynet 仅能近似的模拟出一个给定特征的真实系统的行为。大多数的这种近似模拟是来源于用户的操作系统的时钟粒度和准确性，所以这对模拟器的精度会有一些影响和限制。Dummynet 的第二个问题是，需要周期进行的工作可能晚一些进行，也可能甚至错过一个或更多的时钟滴嗒，这主要是取决于系统的整体的负荷。

3. Packetstorm

Packetstorm 模拟器是由 Packetstorm 网络传输公司推出的[7]。它可以在实验室的条件下，产生关于 IP 网络和局域网的各种环境。它可以模拟出如延迟、抖动、丢包、失序（Out-of-Order）及比特误码（Bit Error）等网络环境。它所模拟出的环境是可控制的并可再生产的，而且它还可以重复产生因特网的动态行为。

PacketStorm 模拟器可提供最多六个网络接口插槽：包括以太网桥接、路由模式和端口映射模式。以太网桥接是提供连接至以太网设备的简便方式。路由模式用于在不同接口类型（例如T1和以太网）之间发送IP数据流量。在端口映射模式下，可以在任何不同的物理接口之间进行流量的映射。此外，PacketStorm还具有动态的模拟网络能力，它可以动态的创建网络传输模型。在这种模型中，网络环境随时间或者带宽等网络状况的不同而不断地变化，从而为研究人员创建真正有效的测试环境。

总的来说，Packetstorm所具有的优点包括：可以提供带有传输情况的分散的服务；备有ToS模拟器，IP监视器，包的计数器和计时器等；可以进行网络捕获和重放，具有多个网络接口。

4. NIST Net

NISTNet 是一种运行在 Linux 上的网络模拟器[8]，它工作在 IP 层，由于此模拟器采用了实时的硬件中断(MC146818)作为时钟粒度，因此它的精确度很高。NISTNet 能把一台运行 Linux 的 PC 机作为一个路由器，从而模拟出常见的网络环境如丢包、延迟、网络拥塞及带宽限制等。由此可见，NISTNet 网络模拟器是模拟 IP 网络动态行为的通用软件，它可以产生可控并可重现的网络环境。在简单的实验室环境下，使用 NISTNet 搭建实验床就可以分析网络行为并控制网络协议的运行。应用 NISTNet 模拟的网络拓扑如图 2-3 所示。

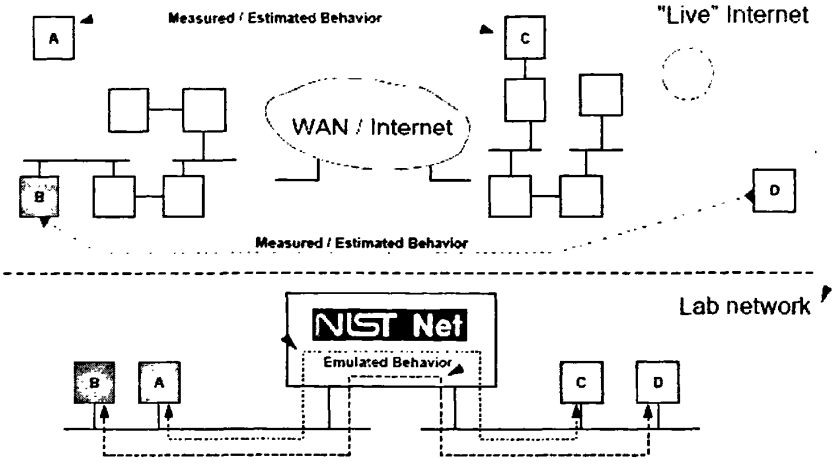


图 2-3 NISTNet 所模拟的拓扑结构

NISTNet 软件包作为内核可装载模块被添加到 Linux 操作系统中，它通过调节底层设备驱动来实现模拟目标网络的网络条件。它截获进入该系统的网络封包，并根据用户预定义的规则来决定对该数据包的处理方式。当数据包进入模拟器时，先要通过底层设备，然后被 NISTNet 截获。NISTNet 会根据预定义的条件对包进行重新调度，转发或是丢弃处理。之后，数据包向协议栈的上层传送。在进行重新调度的同时，时间加速器会被添加到操作系统中以提高系统的时间粒度，从而提高模拟精度。

在 NISTNet 进行模拟的过程中，NISTNet 分开地对待每一个通过它的数据流，而对每个数据流的操作是通过模拟条目（emulate entry）表来产生作用的。模拟条目可以被手动地添加和修改，或者在网络模拟器运行期间按预先写好的程序对条目进行改动。

NISTNet 主要有两部分组成[8]：（1） 可装载的内核模块。这部分被添加到正常 Linux 的网络和实时时钟代码里，以实现运行固有的网络模拟功能和输出一套控制模拟器的 API；（2）一套用户界面。它的作用是主要用来使用这些 API 来配置和控制内核模拟器的操作。代码提供的两个用户界面为：一个简单的命令行界面，适合于脚本编辑；还有一个交互式的图形界面，允许同时控制和监测大数量的模拟登录条目。

模拟器的这种组织结构提供了很多的优点：由于所有的内核功能都被集成到一个可装载的模块中，网络模拟器可能在运行时不用中断任何积极的联系就会开始、挂起、修补和重载，无论这些数据流是不是那些正在被此模拟器所影响的。另外，模块的分离也提供服务使 NISTNet 代码大程度地隔离于基于内核的改变。

NISTNet 可以产生的网络条件有：包延迟（即可确定不变也可以变动）包重组、包丢失、随机性的包依赖、包重复和带宽限制。

NISTNet 可以利用各范围的网络情形来仿真端到端的性能。它设计成在一般的实验室条件下，用网络性能试验程序和控制协议就能实现，而且能被控制和再重复。它可以简单地在真正的网络和模拟的网络之间实现代码的移植，由很小的实验室安装就可以模拟大范围的网络。

2.3 当前网络模拟器的应用问题

虽然目前已经有了各种功能并可以工作在不同操作系统下的网络模拟器，但有时使用这些网络模拟器并不是十分的方便。

例如，在使用 ONE 进行模拟测试时，根据其要求，一台 SUN 工作站上只能安装两个网卡，这样一台 SUN 工作站只能仿一个点，不利于搭建大型仿真环境。另外，启用模拟功能时要关闭 SUN 的其他业务，否则影响准确性，当有多台主机通过 SUN 相连时，会严重干扰其仿真精度。

对于Dummynet来说，虽然它运行在真实环境中，但它仅能近似的模拟出一个给定特征的真实网络的行为[4]。大多数的这种近似模拟是来源于操作系统的时钟粒度和准确性。虽然Dummynet使用了系统时钟的时间粒度T限制了对于所有和时间度量相关的解决方案，但是在模拟高速的网络和较短的管道时，造成了与T相关的网络封包的重叠，以上情况会对Dummynet的模拟精度产生极大的影响。

另外，在使用 NISTNet 和 Dummynet 进行模拟时，这些软件首先必须被编译成操作系统的内核扩展，在这之后它们才能被装载进操作系统进行使用。

在实际的模拟器的应用过程中，并不能保证研究或者开发人员对以上模拟器所工作的操作系统比较熟悉，所以很多研究人员在使用这些模拟器时遇到了本不该有的使用方面的困难，这无形中增加了研究人员的开发周期及测试难度。

另外，IPv6 协议的很多新特性需要在接近真实的环境下并且进行大量重复性实验的基础上才能进行更好的优化和改进。虽然 IPv6 协议及其产品的开发和实现早已在全世界范围内展开，但包括以上常见的各种模拟器在内，目前可以支持 IPv6 协议并广泛应用的网络模拟器似乎并不多见，这给在 IPv6 环境下的新协议及新服务的开发和测试工作带来了一定的困难。

Packetstorm 虽然功能强大并支持 IPv6，但其价格却稍有一些昂贵，并不适合在测试和研究工作中大规模的使用。

由本文的叙述可见，虽然目前因特网向 IPv6 过渡的步伐不断加快，而且 Windows 操作系统的功能也日益强大，但目前仍然没有广泛应用在 Windows 平

台上并支持 IPv6 的网络模拟器供研究人员使用。相对来说，网络模拟器在这一部分的领域还是一片空白。

2.4 Windows 平台网络模拟器的设计特点

跟据以上对各个模拟器的分析，可以发现，在 Windows XP 平台下，要想让所构建的模拟器工作在 IPv6 网络环境中，它要能够根据用户的设置来对 IPv6 网络中的特性做出模拟。在充分考虑了 Windows XP 系统特点的基础上，并不断地与其它已经存在的网络模拟器进行比较，作者对于此模拟器的设计主要遵循了以下几点原则，同时它们也是基于 Windows 平台网络模拟器的主要特点：

（1） 保证模拟器的高效性和准确性。此模拟器要能根据用户对模拟参数的设定，可以对网络环境的各种特点能够做出准确和快速的模拟。即此模拟器的工作不能像 Dummynet 那样在很大的程度上受操作系统时钟精度的干扰，至少应该保证模拟器的工作能在同等条件下优先进行。

（2） 保持模拟器的易用性和灵活性。Windows 操作系统的一个优点就是易用性。此模拟器本身作为 Windows XP 下的一个软件，一定要继承 Windows 易用的特点。即对于它的使用不能像 NISTNet 或者 Dummynet 那样先要导入系统中才能工作。除此之外，即使是对各种不同的网络拓扑结构或者网络环境进行模拟时，此模拟器也要能够保持在其自身的层次结构的不变性。

（3） 保证模拟器的相对独立性。网络模拟器虽然依附于 Windows 操作系统的内核及网络协议栈而存在，但此网络模拟器在工作的同时也要有一定的相对独立性，即其模拟的精度不能像 ONE 那样与操作系统的其它功能有较高的耦合性。

（4） 减少模拟器的副作用。此模拟器虽然要工作在 Windows XP 操作系统中，并对目标网络进行模拟，但在开启模拟器功能的同时，要保证对原 Windows 操作系统的影响尽可能的小。

2.5 Windows 平台网络模拟器工作的拓扑结构

通过对以上各模拟器的研究发现，几乎所有的网络模拟器都是将自身做成了一个路由器在使用，本文所构建的模拟器同样沿用了此想法，图 2-4 为 Windows 平台下网络模拟器工作的拓扑结构：

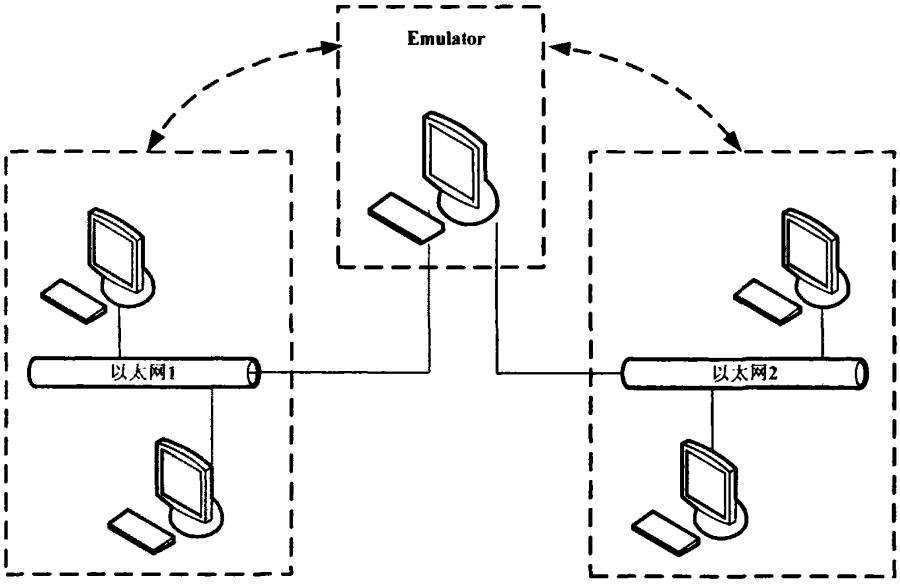


图 2-4 Windows 平台网络模拟器工作的拓扑结构

从图中可以看出，网络模拟器作为一个路由器在两个不同的以太网中进行网络封包的转发。将网络模拟器同路由功能绑定在一起有着如下的优点：

（1） 比较接近真实情况。互联网由许多网络链路及相互连接的网络节点组成，在每个节点和每条链路上，都存在着网络使用者与网络服务器之间传输的信息流丢失、延迟或错误的可能性。网络业务流的损伤通过增加接入（下载）信息的时间或反映为“应用不可用”而单方面的影响用户的使用。因此，在网络模拟器以路由器的角色出现在网络拓扑中是一个相对比较接近真实的选择。

（2） 可以比较容易地获得需要的业务流。路由器作为连接两条或更多链路并决定数据最有效路径的硬件设备，起着在网络间截获发送到远端网段的报文并转发的作用。如果网络模拟器能在实验的网络中同时发挥着路由功能，可以比较容易并准确地获得想要进行控制的业务流，从而免去了在实验过程中大量的人为的设置步骤。

（3） 可以模拟出更广泛的网络环境。将路由功能与网络模拟器绑定在一起，可以模拟出更加复杂的网络结构，并能够对广域网（WAN）的环境做出模拟，这是将模拟器作为网桥所不具备的功能。

本章前半部分主要介绍了网络模拟的概念及目前已有的网络模拟器的工作原理，随后本文结合各个模拟器的优缺点，给出了 Windows 平台网络模拟器的设计特点及其工作环境。构建模拟器所需要的技术手段将在下一章进行介绍。

第三章 构建 Windows 平台网络模拟器的关键技术

3.1 Windows 中的网络模型

OSI 模型将网络通信结构分为 7 层[9]，从下到上依次为物理层，数据链路层，网络层，传输层，会话层，表示层和应用层。UNIX，NetWare, Linux, Windows 这些支持网络连接的操作系统都用到了这 7 层协议。

但是，尽管 OSI 7 层协议在 Windows 中体现得非常明显，然而在 Windows 中却无法严格地将各个层次划分出来。这是因为在 Windows 操作系统本身就没有严格地划分这些层次，也就是说，这些层次会出现一些功能和行为上的交叉[10]，这是由 Windows 本身对这些各个分层的实现手段造成的。

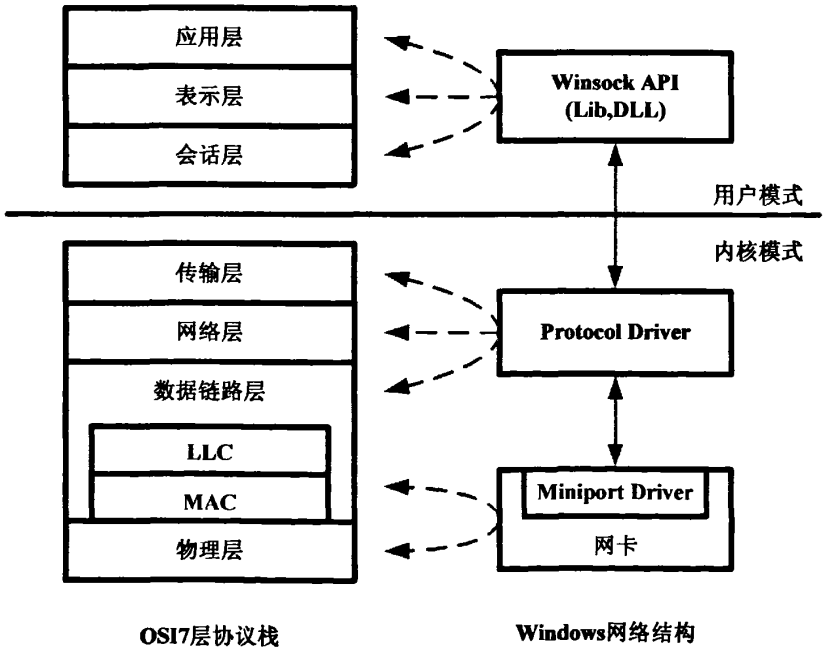


图 3-1 OSI 模型在 Windows 中的实现情况

如图 3-1 所示，Winsock 的 API 主要用于实现了 OSI 模型的应用层、表示层及会话层并为这些应用程序提供接口，而这些程序也都是在 Windows 的用户模式下运行的。OSI 模型的传输层、网络层及数据链路层的 LLC 子层是由协议驱

动（Protocol Driver）实现的[9]。而数据链路层的 MAC 子层及物理层则是由物理网卡实现的，而微端口驱动程序（Miniport Driver）则起着控制网卡的作用。另外，协议驱动程序和微端口驱动程序都是 NDIS 驱动的一部分，关于 NDIS 的介绍会在下一节中进行阐述。

模拟器要能够截获在网络中真实的封包，并且能根据用户设定的参数来模拟和改变网络环境。因此，模拟器要运行在操作系统中处于较低次层上，它或者是对操作系统中的内核扩展，或者是工作在操作系统中的驱动模型中[11]。通过以上描述可以知道，由于网络协议栈中的数据链路层对应着 Windows XP 操作系统的网络设备驱动层，作者选择了将此模拟器构建在 Windows XP 的驱动模型中，通过将它的核心功能模块做成一个设备驱动模块加载到 Windows XP 系统中去，从而可以较快速地对网络接口卡（NIC）进行操作，故模拟器对网络环境的模拟工作不仅是高效率的，而且还保证了其在功能上的准确性。

截包功能是模拟器最原始而基本的功能。由以上介绍可以看出，采用 NDIS 驱动程序来进行网络封包的截获是一种非常理想的方法，而此模拟器工作在操作系统核心态的功能模块正是由 NDIS 驱动和 WDM 驱动共同构建的，二者之间相互协调的工作保证了整个模拟器系统高速而精确的工作。模拟器中 NDIS 功能模块与 WDM 功能模块之间的关系如图 3-2 所示。

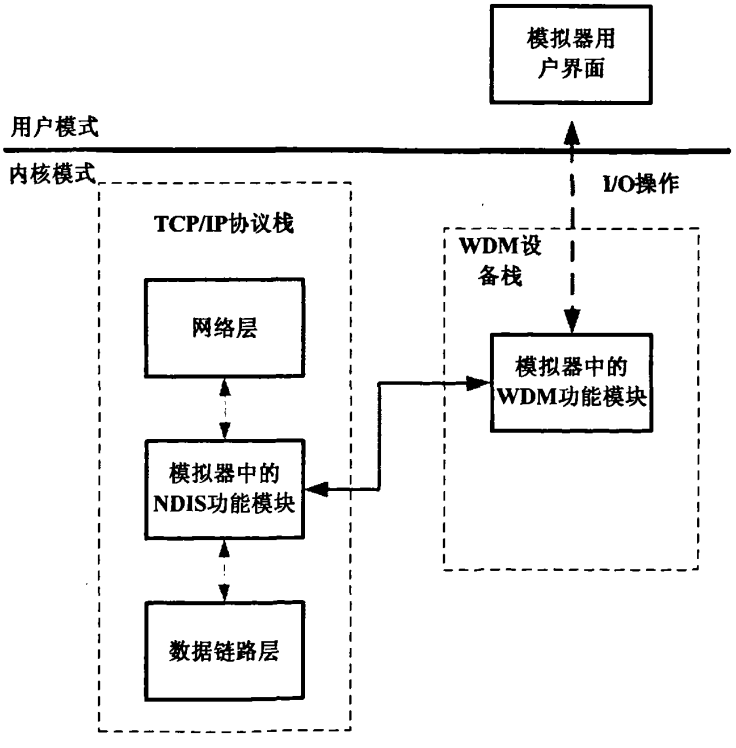


图 3-2 模拟器中 NDIS 功能模块与 WDM 功能模块之间的关系

3.2 NDIS 在模拟器中的应用

3.2.1 NDIS 介绍

NDIS 即网络驱动程序接口规范，是微软开发的网络驱动程序接口规范的简称。为了能够在操作系统较底层截获到网络封包，作者采用了 Windows 下的 NDIS 驱动程序来搭建此模拟器的截包功能模块。

NDIS 可以将网络硬件抽象为网络驱动程序并提供了一个规范的开发流程和函数框架[9]，使得开发者只要实现这些函数，而不用考虑操作系统内核以及与其它驱动程序的接口问题就可以开发出高效的网络驱动程序。NDIS 也说明了网络驱动程序间的标准接口，因此它将用来管理硬件的底层驱动程序抽象为上层驱动程序，为在 Windows 平台上开发网络驱动程序和网络协议驱动程序提供了框架指导。NDIS 同时也维护着状态信息和网络驱动程序的参数，包括指向函数的指针，句柄和链接时参数块的指针，以及其他系统参数。

NDIS 驱动横跨数据链路层、网络层和传输层这三个层次，从网卡驱动程序到协议驱动程序都要利用 NDIS 这个规范来进行操作。NDIS 提供标准接口，网卡驱动程序和协议驱动程序可以利用这个接口来完成必需的任务，NDIS 的拓扑结构如图 3-3 所示。

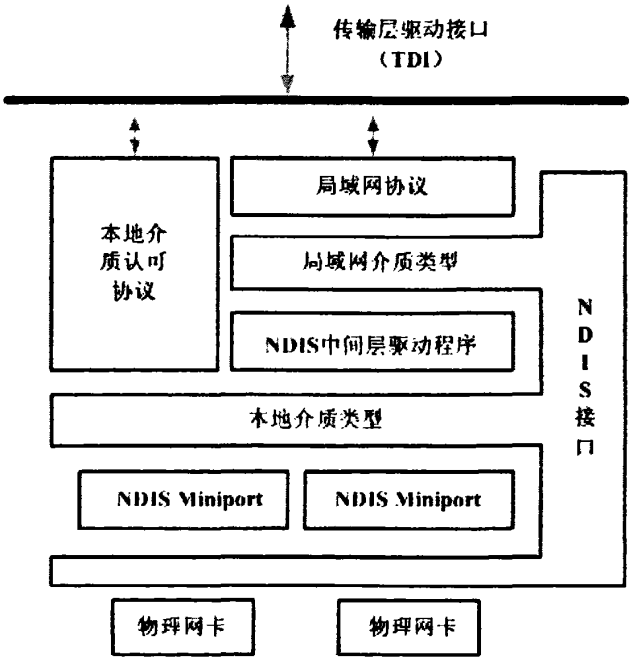


图 3-3 NDIS 拓扑结构

NDIS 支持以下几种类型的网络驱动程序[9]：

- 微端口驱动程序（Miniport driver）
- 中间层驱动程序（Intermediate driver）
- 协议驱动程序（Protocol driver）

微端口驱动程序：一个 NDIS 微端口驱动程序（微端口 NIC 驱动程序）有两个基本功能：1.管理一个网络接口卡（NIC），包括通过 NIC 发送和接收数据。2.与高层驱动程序相接，例如中间层驱动程序和传输协议驱动程序。利用微端口驱动程序可以实现对网卡的驱动。

协议驱动程序：协议驱动程序是在 NDIS 驱动程序层次结构中属于最高一层的驱动程序，它会用来分配包，从应用程序中将数据拷贝到包中，并且通过调用 NDIS 函数将这些包发送到下层驱动程序中；协议驱动程序也为从下层驱动程序中接收包提供了接口。一个协议驱动程序将接收到的数据转换成相应的客户应用数据。

中间层驱动程序：中间层驱动程序一般位于微端口驱动程序和传输协议驱动程序之间。因为它在驱动程序层结构的中间位置，所以既与上层协议驱动程序通信又要与下层的微端口驱动程序通信。利用 NDIS 中间驱动程序可以在网卡程序和传输驱动程序之间插入一层自己的处理，可以截获和操作较为底层的封包。本文所构建的模拟器就是用中间层的驱动程序来实现 Windows 下以太网封包的截获。在下一节中，本文会介绍中间层驱动程序在模拟器中的具体用法。

3.2.2 NDIS 中间层在模拟器中的应用

由上一节的介绍可知，NDIS 实际上就是一个处理数据流程的规范及一些抽象出的接口（interface）和 API。同样，模拟器中与 NDIS 相关的模块必须要遵循 NDIS 中间层的规范及接口要求才能工作。本文所构建的模拟器主要遵循了以下相关的规范：

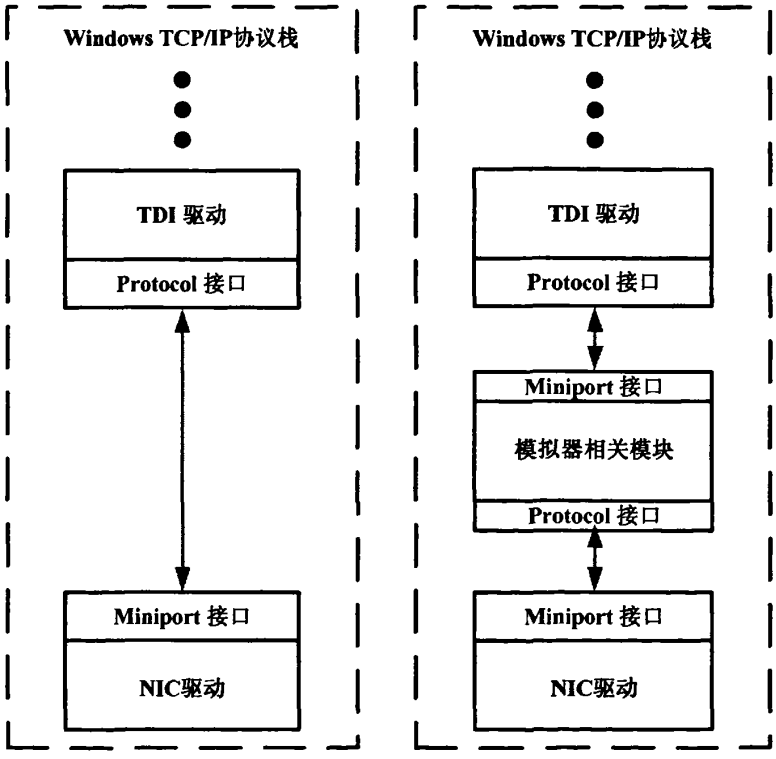
（1）模拟器的逻辑位置与 NDIS 中间层相要求的位置一致。

NDIS 中间层可以嵌在 NDIS 传输驱动程序 TDI（如，TCP/IP）和底层的 NDIS 网络接口驱动程序的中间，这种类型的驱动程序被称为 NDIS 中间层驱动。所以，采用 NDIS 中间层所构建的模拟器模块同样工作在这样的逻辑位置上。如图 3-4 所示，按照 NDIS 中间层的要求，模拟器中与 NDIS 相关的模块在其上边界导出 MiniportXxx 函数，它对于位于其上层的 Protocol 接口提供 Miniport 接口的功能；在其下边界导出 ProtocalXxx 函数，对于位于其下层的 Miniport 接口提供 Protocol 接口的功能。

需要注意的是，模拟器向上层导出的微端口部分（Miniport 接口）必须是非

串行的，因为系统将依赖这些非串行驱动程序并对内部生成的输出包进行排队操作以提供良好的接口以导出 TDI 驱动程序，所以 NDIS 并不是对 MiniportXxx 函数的操作进行串行化处理。这样，驱动程序只要保持很小的临界区（每次只能有一个线程执行该代码）就能提供性能良好的全双工操作。另外，采用 NDIS 构建的相关模块在其上边界仅提供面向无连接通信支持，而在其下边界，则即可支持面向无连接通信，也可支持面向连接通信。

从图 3-4 中可以看出，采用 NDIS 中间层驱动程序所构建的相关模块可以很容易的获得 Windows 中 TCP/IP 协议栈的访问权，而且相关的代码会在系统的内核中工作，可以获得足够高的效率。同时，采用 NDIS 中间层结构可以截获到数据链路层的封包，为模拟器的下一步的构建提供了强大的功能。



图（a）安装网络模拟器之前　　图（b）安装网络模拟器之后

图 3-4　Windows 安装网络模拟器前后的对比（NDIS 角度）

（2）模拟器遵循了 NDIS 中间层的收发封包的流程。

由于 NDIS 中间层处于微端口驱动程序和协议驱动程序之间，它在 NDIS 中起着向上层驱动程序传递已到达的数据包，或者向下层驱动程序发送封包的接口功能，所以它需要负责网络数据的双向传输。而在本文所构建的网络模拟器中同

样需要负责这部分的工作。

模拟器所遵循的接收数据包的流程：

一般情况下，模拟器接收网络数据封包的流程如图 3-5 所示：

1. 当有数据包到达时，下层驱动会通过中断来通知网络模拟器的相关模块；
2. 当网络模拟器的相关模块从下层驱动程序接收到数据包时，它首先会为其分配一个包描述符并依照得到的封包的原始数据来设置它，然后它会调用 NdisMXxxIndicateReceive()函数或者 NdisMindicateReceivePacket()函数向上层指示该数据包；
3. 当上层协议驱动得到了一个完整的数据包并且处理完毕以后，它会通知传递给它封包的网络模拟器的相关模块。
4. 网络模拟器的相关模块在释放完自己分配的资源（主要是其分配的包描述符）后，会调用 NdisSReturnPacket()函数来通知下层去释放它们自己的资源；

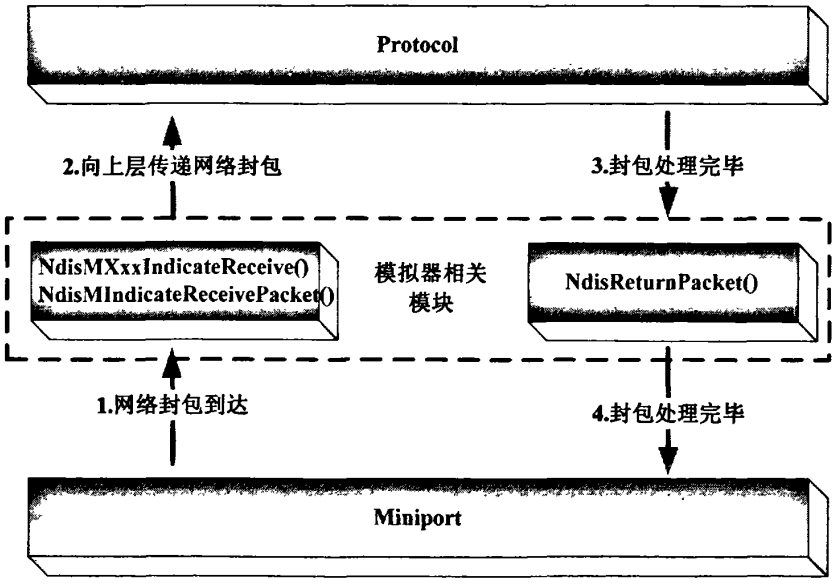


图 3-5 NDIS 中间层接收网络数据封包的一般流程

注意，在以上流程的第一步中，如果在网络模拟器的相关模块中没有接到一个完整的数据包时，需要调用 NdisMEthIndicateReceive()等函数通知 NDIS 的下层，在这之后会发生一个异步的过程来进行接收数据包。这种情况只有在当前的网卡资源比较紧张且传输的单个数据包很大时才会发生，由于整个过程比较复杂而且此种情况极少发生，所以这一过程不在本文中进行介绍。

模拟器所遵循的发送数据包的流程如图 3-6 所示：

1．当上层想要发送网络封包时会通知模拟器的相关模块；
2．模拟器在分配包的描述符后，会通过调用 NDIS 提供的函数 NdisSend（）或 NdisSendPackets（）来向网络低层 NDIS 驱动程序发送数据包或者包数组；
3．当底层驱动将此封包发送完毕后，会通知模拟器的相关模块；
4．在模拟器的相关模块释放本层所分配的资源（主要是包的描述符）后，它会通知上层此网络封包已发送完毕，并让上层去释放它们的资源。

需要指出的是，中间层驱动程序使用符合 NDIS 标准的函数，来提高其在支持 Win32 接口的微软操作系统上的可移植性。

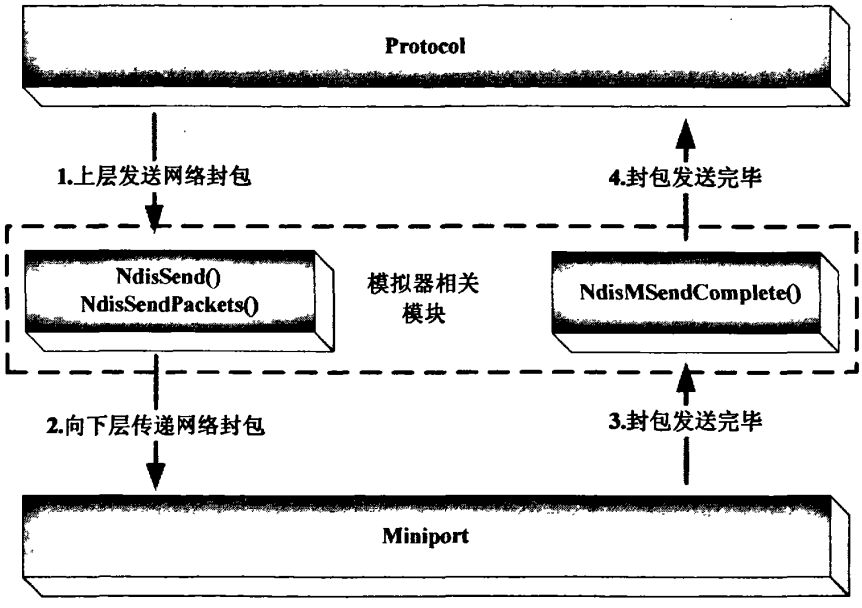


图 3-6 NDIS 中间层发送网络数据封包的流程

通过以上的介绍可以看出，无论是网络封包的接收还是发送过程都会经过模拟器相关模块的处理，所以采用 NDIS 可以较为轻松地获得 Windows 操作系统中 TCP/IP 协议栈的处理权，从而使模拟器更加容易地处理其想要进行操作的封包。

如前文所述，包括本文所构建的模拟器在内的所有 NDIS 中间层驱动程序必须按顺序来执行各项操作，所以 NDIS 中间层主要的局限性如下：

中间层驱动程序至少应该用其分配的新的包描述符代替内入的数据包描述符，不管该数据包是要向下传递给低层驱动程序进行发送，还是要向上传递给高层驱动程序进行接收。当完成操作后，如接收数据操作的第 4 步和发送数据操作的第 4 步，还必须用原始包描述符（最初与传递给中间层驱动程序的数据包相关

联）替换其自己的包描述符。另外，中间层驱动程序还必须通过返回包描述符及其所指定的资源，及时地返还其从高层或低层驱动程序借入的资源。

本文专门针对此局限性的影响做了一下测试，发现以上 NDIS 的局限性并不是影响其性能的瓶颈，在同样的硬件配置条件下，当系统中没有安装 NDIS 中间层驱动时，用 Iperf 的 TCP 流可以发到 92.4Mbps, 当安装上 NDIS 中间层（采用 Windows XP 下 DDK 的示例代码 passthru）时，仍然可以发到 92.3Mbps。所以，本文为认为 NDIS 此处的局限性只是造成资源上临时的一点浪费，而对性能的影响是微乎其微的。

总的来说，采用 NDIS 在 Windows 下实现模拟器的构建是相对标准和规范而有效的方式。它不仅功能强大效率高，而且相对下层的设备驱动要简单。更加重要的是，在满足了 NDIS 中间层驱动程序的规范之后，它在对收发网络封包两方面的强大支持以及可以对包属性的修改功能有利于今后对模拟器的功能进行扩展和完善，能够使其实现更多模拟功能并应用在更多类型的网络领域中。

此模拟器的构建虽然使用了 NDIS 驱动程序，但仅依靠 NDIS 驱动程序只能实现模拟器最原始的截包功能以及对网络封包的操作。为了给模拟器的功能进行更大一步的扩展，本文使用了 Windows 下的另一种驱动程序 WDM 驱动来实现模拟器中较为核心的功能模块，如定时器及延迟功能等。

3.3 WDM 在模拟器中的应用

3.3.1 WDM 驱动介绍

在 Windows XP 中，主要有如下种类的驱动：虚拟设备驱动 VDD（Virtual Device Driver）、文件系统驱动、显示驱动及用于即插即用的 PnP（Plus and Play）驱动。WDM 驱动程序是 PnP 驱动程序的一种，它的全称是 Windows Driver Model，这种模型可以使驱动开发者编写出的驱动程序具有与 Windows 操作系统在源代码级上的兼容性。一个 WDM 驱动程序必须要具备以下所有的性质：

- 在代码中使用了 DDK（Driver Development Kit）的 wdm.h 头文件的数据结构或者库函数；
- 必须是总线驱动程序（Bus drivers）、功能驱动程序（Function drivers）或者过滤驱动程序（Filter drivers）的一种；
- 能够在系统的设备栈中产生设备对象并支持即插即用功能；
- 支持电源管理及 WMI（ Windows Management Instrumentation)。

在本文中，这里所用的通用术语“总线（bus）”是来描述与设备进行电气连接的硬件，这是一个相对广义的定义，它不仅包括 PCI 总线，还包括 SCSI 卡、并行口、串行口、USB 集线器（hub）等等。

WDM 模型使用了如图 3-7 的层次结构。图中左边是一个设备对象堆栈。设备对象是系统为帮助软件管理硬件而创建的数据结构，而且一个物理硬件可以有多个这样的数据结构。处于堆栈最底层的设备对象称为物理设备对象 PDO（Physical Device Object）。在设备对象堆栈的中间某处有一个对象称为功能设备对象(Functional Device Object)，或简称 FDO。在 FDO 的上面和下面还会有一些过滤器设备对象(Filter Device Object)。位于 FDO 上面的过滤器设备对象称为上层过滤器（Upper-level filter drivers），位于 FDO 下面(但仍在 PDO 之上)的过滤器设备对象称为下层过滤器（Lower-Level Filter Drivers）。操作系统的 PnP 管理器按照设备驱动程序的要求构造了设备对象堆栈。

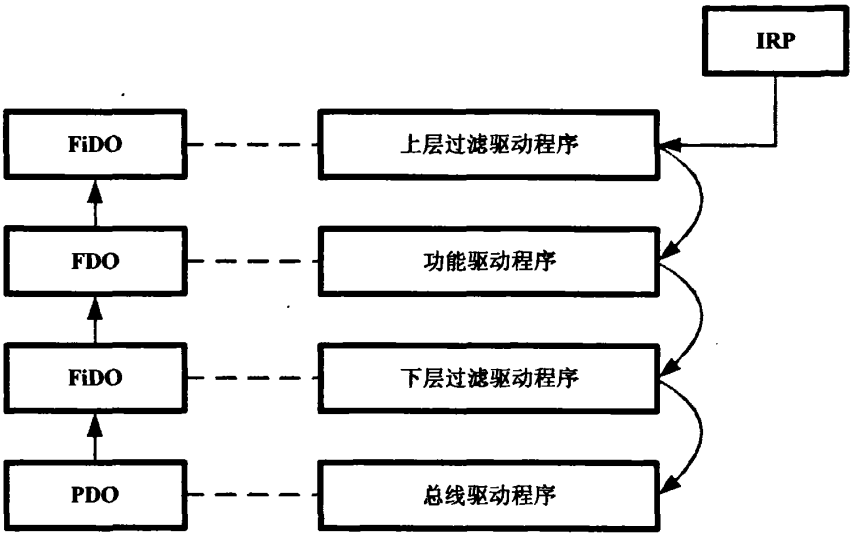


图 3-7 WDM 中设备对象和驱动程序的层次结构

在单个硬件的驱动程序堆栈中，不同位置的驱动程序扮演了不同的角色。功能驱动是用来改变物理设备的行为或者是对设备功能进行扩展的驱动程序，它的主要功能是驱动程序管理 FDO 所代表的设备。总线驱动程序管理计算机与 PDO 所代表设备的连接。过滤器驱动程序用于监视和修改 I/O 请求包即 IRP（I/O Request Package）流。IRP 是 WDM 预定义的重要数据结构，是驱动程序操作的中心。见图 3-7 的右侧，每个影响到设备的操作都使用 I/O 请求包。通常 IRP 先被送到设备堆栈的最上层驱动程序，然后逐渐过滤到下面的驱动程序。每一层驱动程序都可以决定如何处理 IRP。有时，驱动程序不做任何事，仅仅是向下层传

递该 IRP；有时，驱动程序直接处理完该 IRP，不再向下传递；还有可能驱动程序既处理了 IRP，又把 IRP 传递下去。但并非每个设备驱动程序都能够处理 IRP 的全部数据，驱动程序所进行的动作取决于设备以及 IRP 所携带的内容。事实上，一个驱动程序只能处理 IRP 中与之所处栈位置相对应的数据。

3.3.2 WDM 在构建模拟器中的应用

本文所构建的模拟器的内核模块是作为上层过滤驱动加载上去的。这样做的好处是因为上层过滤驱动可以典型地为一个设备提供具有一定特点的功能扩展，而且这样的驱动本身对于操作系统来说是可选的。与此同时，在一个操作系统中可以有多个这样的上层过滤驱动。另外，一个上层过滤驱动程序可以为这一类的所有设备提供这种功能上的扩展。由此，将模拟器的内核模块作为一个上层过滤驱动来构造，不仅可以在系统的核心层比较方便地构建出模拟器本身想要实现的功能，而且对操作系统及原有的设备功能的影响比较小，同时又加强了对底层不同物理设备的兼容性。

更为重要的是，WDM 本身具有处理 IRP 的功能。内核通常通过发送各种 IRP 来运行驱动程序中的代码。由上一节可以知道，由于 WDM 中的驱动程序是分层的，称之为驱动程序栈或设备栈，IRP 便是沿着设备栈向下传递（如图 3-7 右侧所示），这样的工作方式使得采用了 WDM 驱动程序的网络模拟器有机会得到由系统的上层传递下来的数据。为了能够使应用程序与驱动程序进行通信，本文所构建的模拟器正是采用 IRP 使内核驱动程序来得到用户在应用程序中所输入的用于网络模拟的参数；而且作为上层过滤驱动程序的模拟器会最先得到与自己相关的 IRP，在处理完这个 IRP 后便不会向下传递。关于这一部分功能的具体实现会在本文的第四章中用户配置模块的一节中进行详细的介绍。

WDM 驱动程序是 Windows XP 操作系统重要的组成部分，它的正常工作需要有 Windows XP 其它内核组件的支持，同时大部分的内核组件也必须同 WDM 驱动程序交互来完成它们的功能。WDM 所构建的驱动程序在运行时也具有较高的优先级和较好的效率，这也是作者选用 WDM 作为模拟器主要功能模块构建手段的另一个原因。

最后需要指出的是，从 Windows Vista 开始，WDM 的驱动模型不再被采用，取而代之的是 WDF（Windows Driver Foundation）驱动模型。

另外，虽然此模拟器是作为一个设备的驱动模块加载到 Windows XP 操作系统中，但它并没有对系统中原有的驱动模块做出改变，这样就保持了 Windows XP 系统的完整性和模拟器的相对独立性；另一方面，模拟器又可以通过驱动程序来调整和控制网络设备，从而可以模拟出目标网络环境。

3.4 Windows 平台下模拟器的开发及运行环境

3.4.1 模拟器的开发环境

设备驱动程序工作在Windows的核心层，它可以完成非常底层的操作并拥有对CPU的完全控制权，还可以突破Windows的保护机制。无论是网络驱动程序NDIS还是WDM，它们本质上同样是都驱动程序。对于驱动程序的开发，微软公司提供了设备驱动程序工具包Driver Development Kits（DDK），使用它可以编译生成Windows中各种驱动程序，而且开发出来的驱动程序效率也是最高的。

但DDK仅仅是一个开发包，它只是提供了编译和生成驱动程序的环境，即只负责与驱动相关部分的代码的编译和生成。对于其它的功能代码，同时它也需要Visual Stdio编译器的支持。

Visual Studio虽然包含了一系列高效的、智能的开发工具，但并没有提供一个可以和DDK相互关联的功能，所以直接使用Visual Studio编写驱动程序的代码只能发挥它的代码编辑器的功能。但是，Visual Studio提供了一种叫做makefile的功能[10]。这种工程在编译程序时可以使用用户自定义的命令，它直接调用命令行模式的命令来编译和创建程序，而且命令行的输出信息直接输出在Visual Studio的信息输出框里。利用这一特性，只需把编译驱动程序的一系列命令做成一个批处理文件，然后让makefile工程执行这个批处理来完成驱动程序的编译和创建工作。

本文在Windows XP（sp2）环境下，使用Visual Stdio.NET 2005 与Windows XP DDK相结合的方式来进行模拟器的构建工作。在对代码进行编译时采用DDK的Checked Build Environment 的环境进行，在编译后的目标目录下会生成以.sys为后缀名的驱动程序。

需要说明的是，在使用 Visual Studio 来编辑并编译驱动程序的代码时要先创建一个空的项目。

3.4.2 模拟器的运行环境

此模拟器工作在 Windows XP 操作系统中。目前来看，只要是 Microsoft Windows XP 可以识别的以太网卡或者是经过 Windows XP 抽象出来的以太网络设备，都可以运行此模拟器，但目前此模拟器只能工作在以太网中。

另外，模拟器的用户界面部分需要进行在已安装 Microsoft .NET Framework v2.0 的环境下才能运行。

第四章 Windows 平台下 IPv6 网络模拟器的构建与实现

虽然本文所构建的网络模拟器主要是利用网络层的封包信息进行识别和控制业务流的，但是跟据前文所述的原因以及为了提高模拟器的工作效率并减少构建模拟器的工作量，作者将其设计在了是数据链路层，即模拟器的工作大多数是在数据链路层完成的，它只有小部分工作需要在网络层和传输层进行。如图 4-1 所示，工作在三个网络层次之间，通过影响进入模拟器的业务流，此模拟器具有可以快速并准确地模拟出各种目标网络的特点。

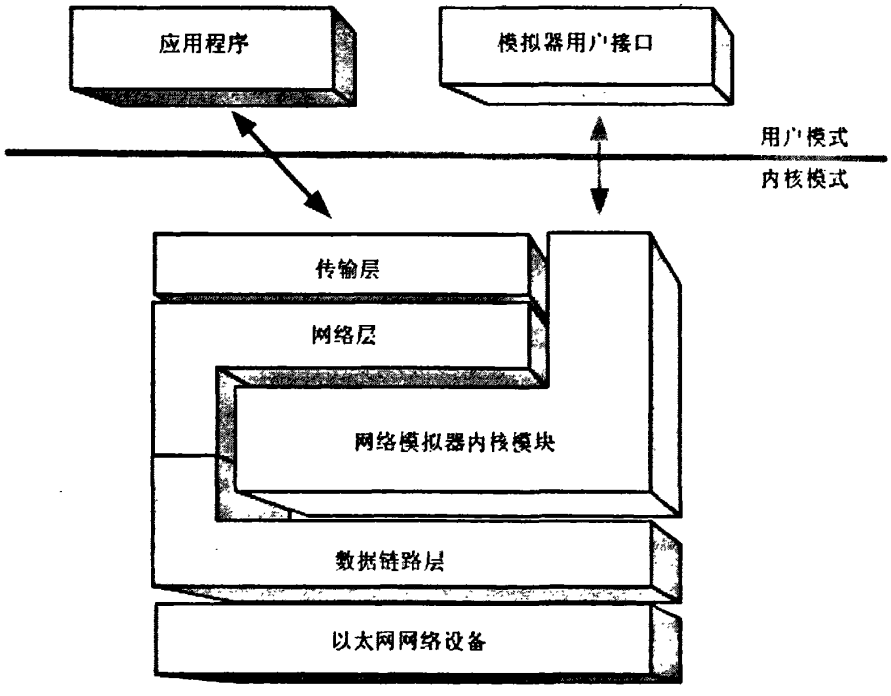


图 4-1 模拟器的层次结构

需要说明的是，在此模拟器的设计及构建方面，本文并没有严格的遵守 OSI 七层网络协议，即此模拟器的内核模块横跨数据链路层、网络层和传输层三个层次，采用这种做法可以在很大程度上降低构建模拟器的技术难度。如果想要在网络层完全的实现模拟器所有的核心功能，则模拟器不仅要有访问 Windows 操作系统的 TCP/IP 栈的权限，还要有能够阻塞及控制网络数据包的能力，这样的功

能可能需要通过修改 Windows 操作系统的 TCP/IP 栈来实现，而目前所公开的技术还没有能够满足这一要求。虽然 SPI[13]（Service Provider Interface）具有类似的功能，但其效率并不如 NDIS 高（NDIS 所构建的内核驱动模块比应用程序有更高的优先级），而且其功能也非常的有限。更为重要的是，更改 Windows 操作系统的 TCP/IP 协议栈加大了网络模拟器同操作系统的耦合性，而且还可能对 Windows 操作系统原有的网络功能的产生很大的影响，反而不利于模拟器的使用。相对来说，采用 NDIS 可以很方便地截获到数据链路层的封包，而且不用改变 Windows 操作系统原有的 TCP/IP 协议栈，从而保持了模拟器的相对独立性。

除此之外，模拟器需要知道网络封包 IP 地址的同时也需要知道传输层的端口号才能将其与用户设定的参数联系起来，如果为了遵守 OSI 协议，就必须要使模拟器的内核模块分为两部分，一部分工作在网络层，另一部分工作在传输层，这样做不仅使构建模拟器的复杂程度大大加大，而且在效率上可能也会有一些下降。如果模拟器构建在网络层，对于那些将要被丢弃的数据包（即模拟丢包操作时），在数据链路层就将此数据包丢弃会比在网络层将其丢弃的效率要高一些。

虽然模拟器的内核模块横跨了三个层次，但它并没有去进行网络层或者是传输层的任何其它工作，它所做的只是读取这两个层次的信息，所以说在功能上来讲内核模块并不是真正的跨层次的操作。

4.1 模拟器的体系结构

像许多其它操作系统一样，Windows XP 通过硬件机制实现了内核模式和用户模式两个特权级别。与此对应，如图 4-2 所示，此模拟器的体系结构由两个大部分组成：（1）模拟器的上层用户界面。此部分工作在用户模式下，用户可以使用这个界面来控制模拟器的行为和配置所要模拟的目标网络。（2）网络模拟器的内核模块。此模块工作在 Windows XP 的核心模式，主要用于实现网络模拟器的核心功能。与上层用户界面相比较，它主要负责网络封包的接收和控制等一系列操作以及对目标网络环境的模拟工作。它不仅要与底层的网卡驱动进行交互以实现模拟器的核心功能，同时此模块还要对工作在用户模式下的界面开放用于控制的接口以接收用户输入的延迟和丢包率等信息。

此模拟器是充当路由器的角色在网络中出现的，所以通过它的业务流可能不只一条。此模拟器通过一个三元组来标识和区分每个业务流，这个三元组的成员是：源端和目的端的 IP 地址、协议号及传输层的源端和目的端的端口号（对于 ICMP 及 ICMPv6 则没有端口号，取而代之的是使用它们的服务类型来进行区别）。模拟器的封包截获模块在数据链路层截获数据包，在将其交给操作系统的

路由功能模块之前会得到关于此数据包的三元组。模拟器的用户界面会要求用户在启动模拟器之前输入用户要要控制的业务流的三元组以组成与这个三元组相对应的模拟条目（Emulate Entry），每个模拟条目还包括延迟和丢包率及带宽限制等信息。

如图 4-2 所示，此模拟器只控制进入模拟器方向的业务流，而对从模拟器发送出的数据不做任何控制。对于需要经过模拟器但不需要模拟器控制的业务流本文将它们统称为背景业务流（background traffic）。从进入模拟器的业务流流向来看（图 4-2 中的实线箭头），考虑一般的情况（即丢包率不为 0，有带宽限制及延迟），封包匹配模块会通过匹配三元组来找出那些需要模拟器进行控制的业务流（即非背景业务流），这些数据包可以进到网络模拟器的内部，而背景业务流的数据包会被封包截获模块直接送给系统的上层去处理。

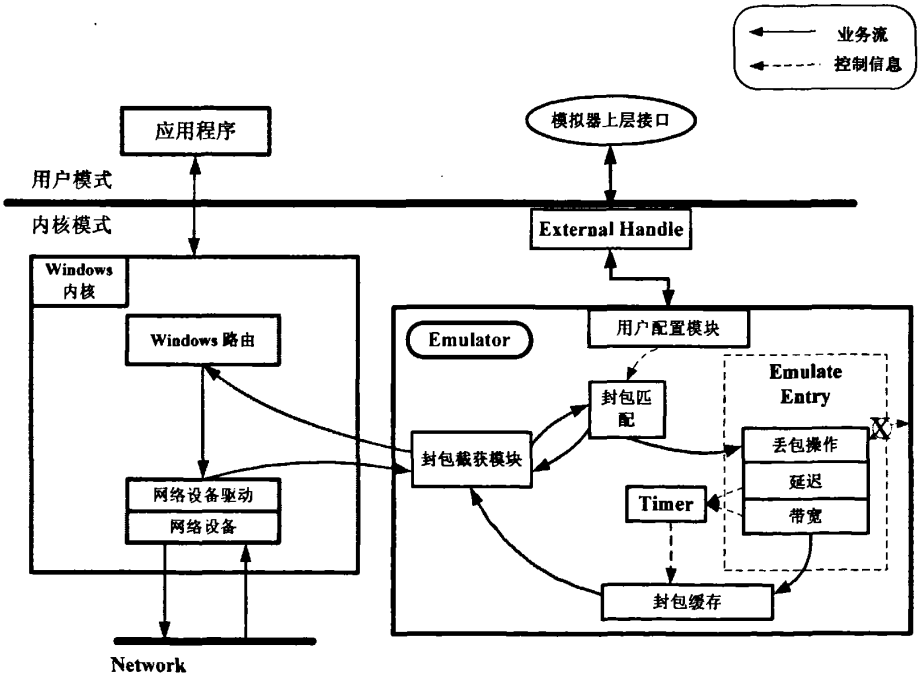


图 4-2 模拟器的体系结构

进入模拟器内部的数据包会按照用户输入的模拟条目进行操作。在这里，如果丢包率不为 0，那么一部分的数据包要在这里被丢弃掉，其余的数据包会被放到缓存中去。通过用户设置的延迟信息可以控制模拟器的定时器，使它产生出相对较精确的延迟时间。除此之外，用户设置的带宽也会对封包的延迟产生影响。那些延迟时间已经到达的数据包会被送到封包截获模块去交给系统的上层进行处理。

注意，此模拟器利用了 Windows XP 本身的路由功能，这样做可以减少构建模拟器的工作量，以便于将主要的精力放在模拟器核心模块的构建上。

此模拟器下层的网络设备可以有多个，这些设备全部是由 Windows 来进行管理而不必由模拟器负责。另外，模拟器的核心模块是工作在网络设备驱动程序的上层，这样做不仅使模拟器完全脱离了与硬件的关联，而且又继承 Windows XP 的兼容性，即使物理的网络设备不是以太网卡，只要它的驱动程序能将其模拟成以太网卡（如 802.11 网卡驱动），模拟器同样可以使用这个网络设备进行工作。

从图 4-2 中可以看出，在此模拟器中，用户配置模块向外部提供 hooks，用户接口通过对 Windows XP 中的 I/O 控制向下产生指向模拟器内核模块的 hook 的句柄，两个模块通过此种方法便可进行交互。其中模拟器的用户接口界面是用 C#语言实现的（如图 4-3 所示）。

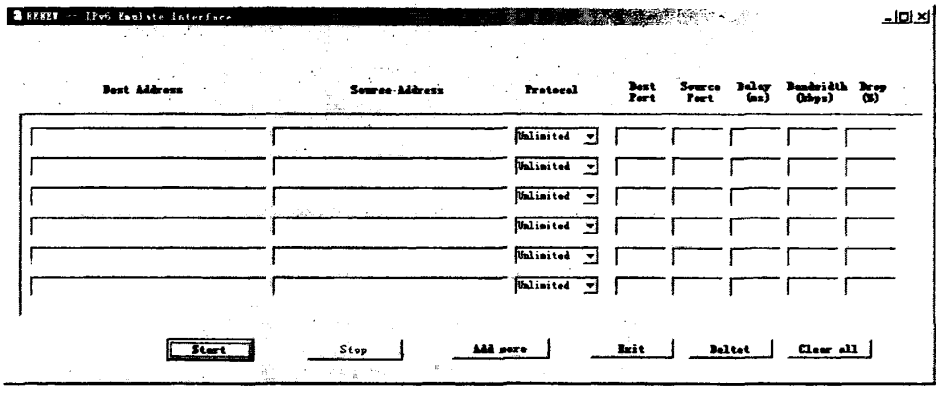


图 4-3 网络模拟器的用户界面

从图中可以看出，用户所需要做的工作只有将需要模拟器控制的业务流的信息相应的填写好，然后启动模拟器就可以了。而且通过“Stop”按钮用户可以随时的停止模拟器的工作并修改参数。

由于模拟器的核心模块是由 C 语言构建的，所以在用户界面与核心模块之间有一个用 C 语言构建的 DLL 作为二者的接口[14]。

4.2 定时队列的设计

定时功能是模拟器中最核心的功能，这是因为无论是实现网络封包的延迟还是带宽限制，都需要通过定时器来进行触发。可以说，定时器的性能决定了整个模拟器的性能。

在本文构建的模拟器中，有一个专门用于实现定时功能的定时队列，这个队列是一个带有表头和表尾指针的双向链表，这个双向链表的每个的成员的数据结

构组成如图 4-4 所示。这里所说的超时并不是指当前时刻已经超过了队列成员需要触发的时刻，而是指定时成员被触发的时刻刚好来到。每个成员都记录了其超时的时刻 T（64bits），这个时刻 T 是一个绝对时间，其度量的单位是距 1601 年 1 月 1 日起的 100 纳秒（ns）的个数。那么这个定时队列中所有的成员均按照各个的超时时刻 T 的大小进行升序排列，因此，越早超时的成员距离表头指针越近而最后一个超时的成员排在队列的尾部。而表头指针则指向目前情况下这些成员中最先超时的那个成员，而表尾指针向这些成员此情况下最后一个超时的成员，如图 4-4 所示。

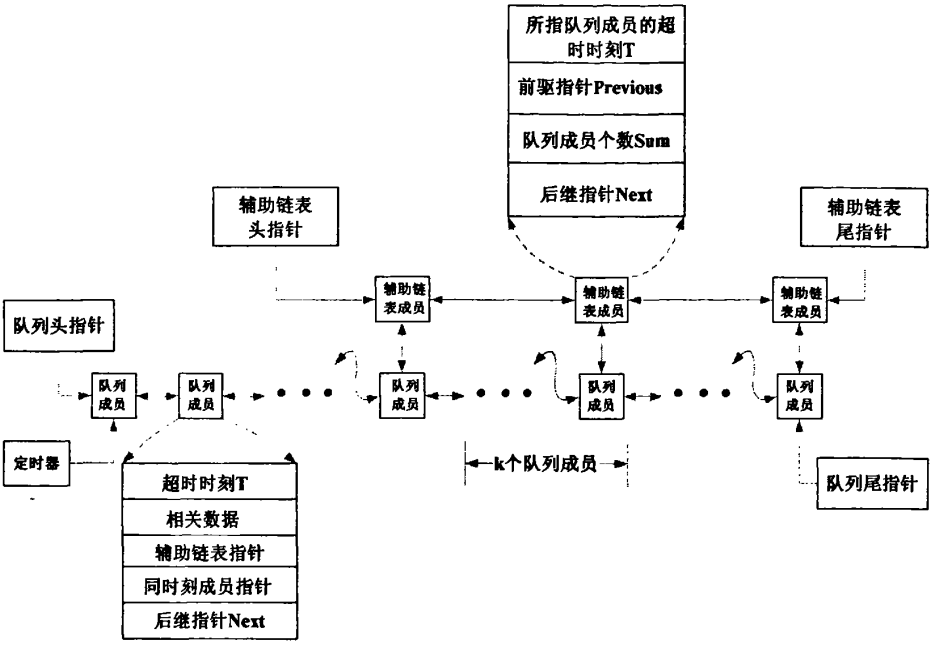


图 4-4 定时队列的数据结构

与 NISTNet 不同的是，模拟器并不为每一个队列成员都分配一个定时器，而是整个队列只使用一个定时器，这个定时器被用来监视链表的头指针所指向的成员。采用此方法的原因有三个：（1）在大多数情况下，模拟器在单个 CPU 上运行，这样任意时刻 CPU 只能处理一个超时的队列成员，在此情况下采用多个定时器与单个定时器是一样效果的；（2）即便此模拟器运行在多个 CPU 的机器上，由于超时时刻是以 100 纳秒为单位的，所以各个成员之间的相对时间差会比较明显，从而各个成员的超时时刻发生“聚集”的情况会比较少，所以一个定时器已经足够；（3）在 Windows 内核中，当队列成员的数量非常庞大时，为每一个成员都分配一个定时器需要占用系统中大量而有限的资源（主要为系统的非分页内存），而只使用一个定时器会比较节省资源。

但在定时队列的构建过程中，作者还是采用了一个用来记录超时时刻相同或极为相近（相差 1 毫秒以内）的数据成员，这就是图 4-4 中队列成员的同时刻成员指针。

这个定时队列并不是固定不变的，它会随着时间的变化而变化，在一段时间里，一些队列的成员会因为超时时刻已到而从队列中离开；同样，还可能会有一些新的成员被加到队列中去。对于新加入的成员来说，已存在的定时队列是一个按超时时刻升序排列的双向链表。为了保持此链表的这种排序，新加入的成员应该使用插入排序（比较它们的超时时刻 T）来找到它所应该插入的位置。但是后加入的成员不一定会排在链表的最后，所在当队列中的数据很多时，这种查找可能会花费相对多的时间，所以作者又引入了一个辅助链表来加速查找。

这个辅助链表同样是一个带表头和表尾的双向链表（如图 4-4 所示），它的每个成员分散的指向定时队列的某些成员，分散的间隔为 k-1。当有新的成员要加入队列时，会首先与队尾的成员进行比较（因为新进的成员插入在队尾的可能性相对比较大），如果其发送时刻比队尾的要提前，便会沿着辅助链表向前去比较，当找到第一个比它的超时时刻提前的成员时，它便会沿着辅助链表成员的指针在定时队列中找到它合适的位置，其流程图如图 4-5 所示。

模拟器专门启动了一个内核线程来维护这个定时队列相关的算法。所以此模拟器是多线程的。在 Windows XP 系统中，模拟器产生的是核心态的线程，这样的线程由于没有 TEB（Thread Environment Block）和用户态的环境[12]，故它能具有较高的优先级和较好的效率。模拟器运行了以下两种功能不同的线程：主线程和时钟线程。在初始化时，主线程主要负责各个句柄及适配器的绑定和初始值的设定。在初始化阶段的最后，它会开启一个新的线程，即时钟线程。在模拟器的工作阶段中，主线程主要负责网络封包的接收和发送，时钟线程则主要负责监视停留在定时队列中的每一个成员，当有成员的延迟时间到达时会把它从队列中移出交给主线程去处理。

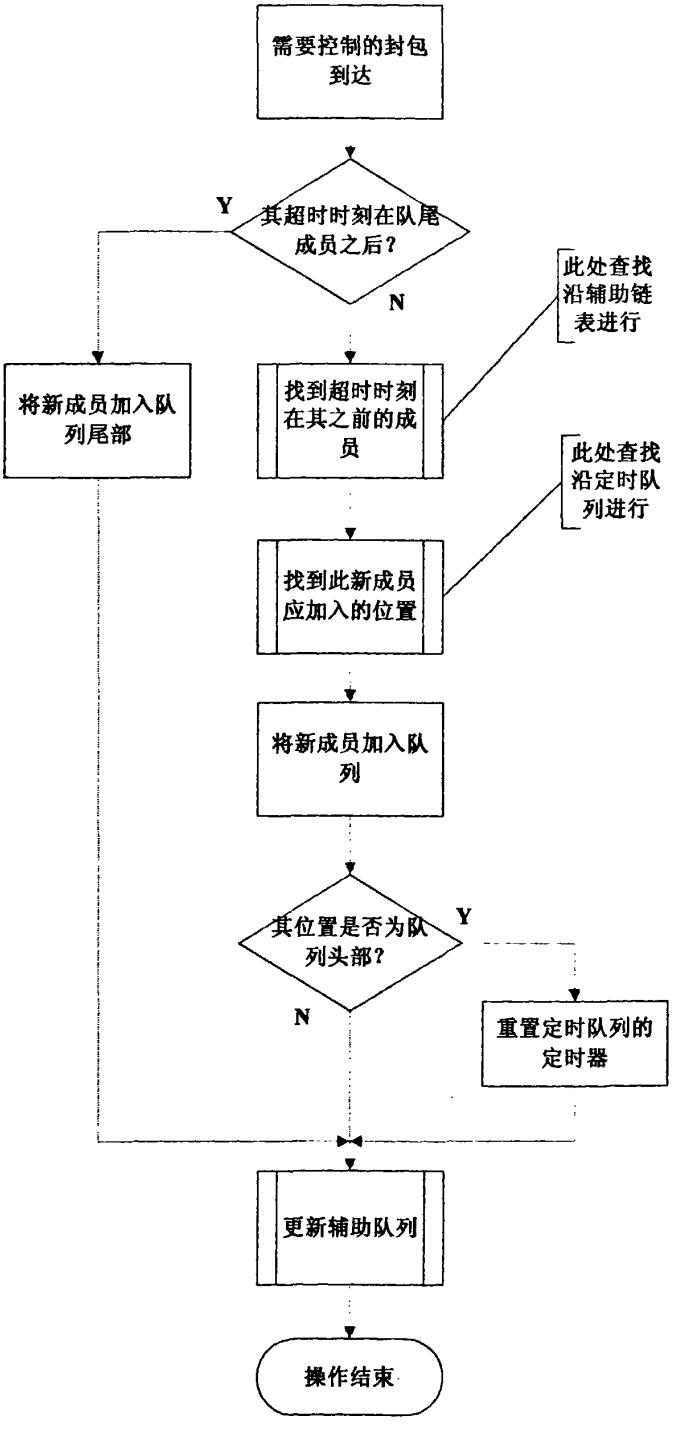


图 4-5 定时队列实现的流程图

4.3 模拟器核心功能模块的设计与实现

4.3.1 封包截获模块的设计与实现

当采用 NDIS 驱动程序截获到网络封包时，NDIS 交给中间层驱动程序实际上是一个包描述符（packet descriptors）。在 NDIS 4.x 系列和 NDIS 5.x 系列驱动程序中，每个封包都是由结构为 NDIS_PACKET 的包描述符来标识的（NDIS 6.0 采用结构 NET_BUFFER_LIST 来描述），本文采用 NDIS 4.x 系列来构建模拟器。如图 4-6 所示，在中间层所看到的网络封包结构是一种类似于单向链表的形式，而包描述符相当于整个链表的表头指针。

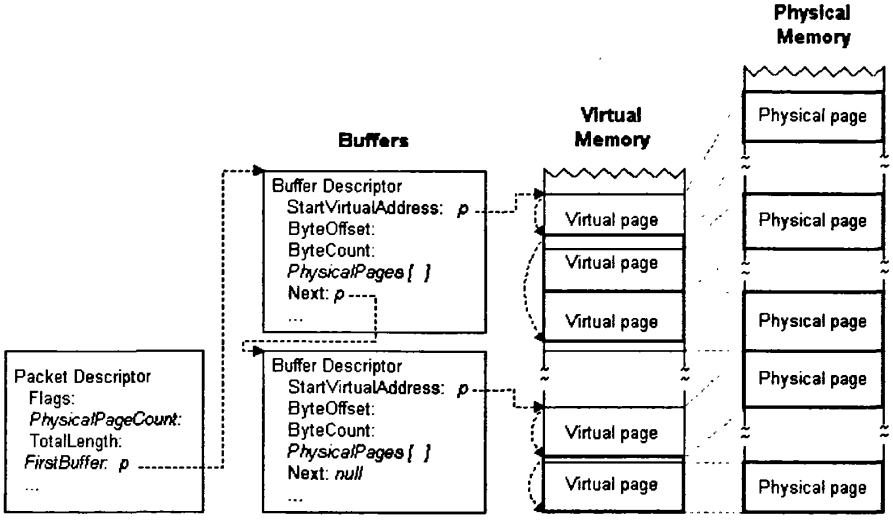


图 4-6 中间层网络封包的结构

如图 4-6 可以看出，一个完整的网络封包是由以下三部分组成的：

- 包描述符。它包含了此数据包的一些整体信息如数据包的大小、占用的物理内存页的页数及指向第一个缓存描述符（buffer descriptor）的指针。
- 一系列的缓存描述符（buffer descriptors）。缓存描述符主要用来记录每个缓存的虚拟地址及它们的偏移量，在缓存描述符中还有指向下一个缓存描述符的指针。
- 虚拟地址空间。这些虚拟地址的空间会被映射到数据包所占用的真正的物理内存中去。

由第三章第二节的叙述可知，模拟器是工作在 NDIS 中间层驱动程序的，所以它同样要为每个网络封包分配一个自己的包描述符。由于下层驱动程序传递上

来的网络封包在内存中的存储地址并不是连续的，所以模拟器在进行封包匹配之前需要先把数据包中的所有数据先拷贝到一个连续的内存中才能进行使用。而对于每个网络封包的包描述符，模拟器均开辟一块连续的内存来存储这些封包中的数据，当对其进行的操作完成后再将其交给上层进行处理。

当将一个数据包的所有数据拷贝到一个连续的内存中后，封包截获模块可以按照与数据进入协议栈进行封装时相反的顺序读入数据包的信息，即以太网首部->IP 首部（IPv4 或 IPv6）->传输层首部（TCP 或 UDP）的顺序。

当需要将控制过的数据包向上层传递时，模拟器会把它分配的这块连续的内存与自己分配的包描述符相关联起来，然后再传递给上层。尽管这时封包描述符中只有一个缓存描述符和一块虚拟内存，但在系统的上层看来这个数据包与未经过网络模拟器控制过的数据包是没有任何区别的。

4.3.2 延迟功能模块的设计与实现

由上一节的描述可知，如果要想实现对一个封包的延迟，只要使这个封包与一个定时队列的成员联系起来就可以了。当模拟器的封包截获模块获得到网络封包后，会根据用户设定的延迟来计算出这个封包的超时时刻，然后便会将其交给专门处理定时队列的时钟线程来处理。当这个包的超时时刻已到，它便会被时钟线程从队列中取出，去交给主线程进行与发送相关处理。

对于定时功能的实现，作者使用了 Windows XP 提供的内核同步对象（Kernel Synchronization Object）。在任何时刻，任何内核同步对象都处于两种状态中的一种：信号态或非信号态。通过对内核同步对象的控制可以阻塞线程的执行，并且可以通过调用系统函数 KeWaitForSingleObject 来使代码（以及特定线程）在一个或多个同步对象上等待，以等待它们进入信号态。与 Linux 等操作系统相比，在 Windows XP 中的核心态下，模拟器通过使用工作在非分页内存中的内核对象可以获得时钟粒度为 100 纳秒的定时器。

在延迟的模块中，不仅将内核同步对象与网络封包相互关联起来，而且还将内核对象的延时期限和封包的超时时刻 T 一一对应起来。通过此种方法，模拟器的时钟线程只需要检查有哪些内核同步对象延时期限已到就足够了。这是因为内核同步对象的延时期限到达时，它所对应的网络封包的延迟时间一定已经到达，模拟器就可以对相应的封包进行发送等操作了。

虽然在内核驱动中可以比较容易地使用粒度为 100 纳秒的内核对象，但对于 Windows 操作系统来说，其内核模式下默认的时钟粒度为 10 毫秒[9]，这是因为 Windows 本身并不是一个实时的操作系统[15]。在这种情况下，一个内核同步对象可能提前 10 毫秒或者是滞后 10 毫秒其状态就会发生变化，虽然这种精度对于

普通的应用程序来说已经足够，但对于模拟器来说并不能达到足够的精确度。虽然 Windows XP 的内核驱动程序提供了设置定时器粒度的 API 即 ExSetTimerResolution（），但利用此函数仍然没有得到本文相要的时钟精度，本文发现尤其是当系统的负载较重时，定时器的精度仍然在 10 毫秒左右。

由于 Windows XP 是一个支持多线程的操作系统，当一个内核对象的超时时刻到来时，它会被送到系统的事件队列中去（Event Queue）[13]，而这个事件队列是按照各个对象所处的线程的优先级来进行排列的。因为最高的优先级是被硬件所占据的，对于一个软件来说，要想提高定时器的精度，就一定要提高定时器所在线程的优先级。本文所构建的模拟器将时钟线程的优先级设置为软件所能支持的最高一级，为 LOW_REALTIME_PRIORITY，经测试发现这一做法可以使时钟的精度有大幅度的提高。

可见，在模拟器中，所有对封包的处理不是在主线程就是在时钟线程中进行的，这种做法一方面保证了模拟器的准确性，另一方面也减少了主线程的工作量，较好地减少了由于某个线程的繁忙而引起的对于封包处理时间上的延误。

4.3.3 丢包功能模块的设计与实现

模拟器会将丢包的事件均匀地分散到这个业务流中去，即如果用户将某个业务流的丢包率设为一个非 0 值，这个业务流中的每个数据包被模拟器丢掉的概率是相同的即为用户设定的丢包率。例如用户将一个业务流的丢包率设为 10%，那么这个业务流在经过模拟器时，它的每个数据包被模拟器丢掉的概率为 0.1。

为了能够实现以上均匀的分布，应该使用一个均匀分布的随机数生成器。如果假定需要生成介于 0 和 100 之间的一个随机数，每一个数出现的几率应该都是一样的。即在理想情况下，应生成 0 到 1 之间的一个值，这个范围中的每一个值出现的几率都相同，然后再将该值乘以 100。但需要注意的是，在 0 和 1 之间本身就有无穷多个值，而计算机不能提供这样的精度，所以在模拟器的构建过程中本文只能使用非连续且有限的伪随机数生成器。

常见情况下，伪随机数生成器生成 0 到 N 之间的一个整数，返回的整数再除以 N，从而得出的数字总是处于 0 和 1 之间，然后再将该值乘以 100 以得到 0~100 间的数值。这意味着，由任何伪随机数生成器返回的数目会受到 0 到 N 之间整数数目的限制。在本文所采用的随机数发生器中，N 等于 RAND_MAX 其值为 2^{15}-1（32767）[17]，即此伪随机数生成器能够至多生成 32767 个可能值。

当用户将一业务流的封包的丢包率设为一个非 0 值时，为了避免使用浮点数，模拟器首先会将这个丢包率转化为一个 0~100 之间的数值使其变成转化值，例如用户设定的丢包率为 10%，那么模拟器会将这个数值其扩大为原来的 100 倍即转

化为 10。当模拟器的内核模块每接到此业务流的一个数据包时，会先用伪随机数生成器算出一个 0~100 之间的随机数，然后再用这个随机数与从用户所设定的丢包率转化来的数值进行比较。如果生成的随机数大于或等于转化的数值，那么程序就会把这个封包做相应的处理（也许放入缓存中，也许向上传递）；反之，如果这个随机数小于这个转化值，那模拟器直接可以调用 NDIS 中的 NdisFreePacket（）函数将这个封包丢掉即可。整个过程的流程图如图 4-7 所示。

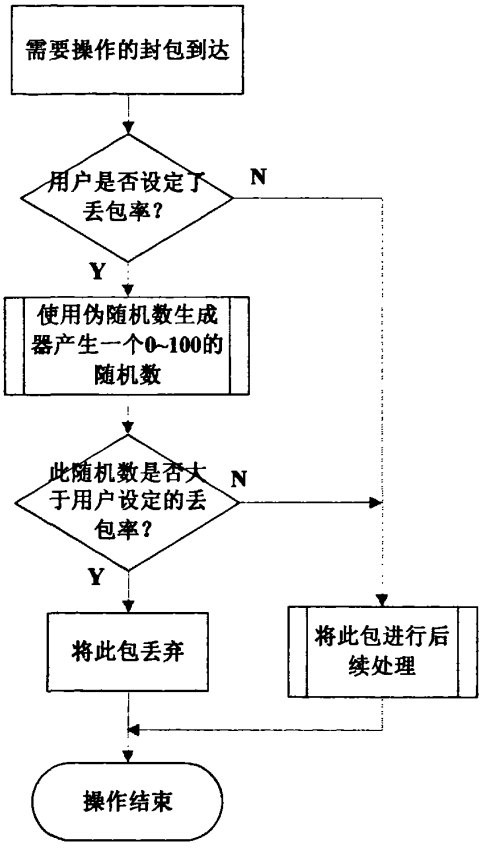


图 4-7 丢包模块的流程图

4.3.4 带宽限制功能模块的设计与实现

带宽限制功能是通过对网络封包添加传输延迟来实现的。

传输延迟主要是用来模拟出封包在物理链路上传输所用的时间，它主要是由传输链路的带宽和包的大小来决定：

设一个包的大小为 *packet_size*（bits），在传输的带宽的比特率为 *bit_rate*（bps）的情况下，本文的模拟器要求这个网络封包至少需要被延迟的时间 D_{trans_delay}（以

秒为单位）由以下公式决定：

$$D_{trans\_delay}= \frac{packet\_size(bits)}{bit\_rate(bps)} \qquad \text{公式（4-1）}$$

对于那些设置了带宽限制的业务流，模拟器都会记录上一次发送此业务流的网络封包的时刻，以确保两个网络数据包的时间间隔足够大以防止超过用户设定的带宽限制。

从理论方面，一个业务流的每个网络数据包可以是在 D_{trans_delay} 时间内的开始，中间或者结尾发送出去。本文所构建的模拟器均将封包的超时时刻定在 D_{trans_delay} 的结尾。即此业务流的第一个数据包到来时也要被延迟 D_{trans_delay} 的时间。

在模拟器的实际构建过程中，由于数据包的大小太小或者用户设定的带宽限制很大等原因，以秒作为传输延迟的最小单位仍然不够精确，所以在实际的实现代码中传输延迟同样是以 100 纳秒的个数来计算的。

4.4 模拟器辅助功能模块的设计和实现

4.4.1 用户配置模块的设计与实现

通过以上本章的描述，模拟器的核心功能模块已经构建完毕，只要有关于网络模拟的参数输入给这些核心功能模块，相应的网络环境便会被模拟出来。但由以上的描述可知，模拟器的核心功能模块都是工作在操作系统的内核模式下的，而用户只能从用户模式输入相关的模拟参数，所以内核模块要想从用户模式下的用户界面中得到用户输入的参数，就要使用本文前面介绍过的 IRP。此过程一共需要三个步骤：

1．产生句柄

如第本章的第一节所述，应用程序与驱动模块要想通信，首先要产生一个用于连接的句柄，方法如下：

（1）模拟器的内核模块首先使用 NdisMRegisterDevice（）函数来产生一个设备对象并将其命名。这个设备对象的名字就像一个链接一样将在用户模式下可识别的字符同这个设备对象连接起来[9]，同时此对象带有 IRP_MJ_DEVICE_CONTROL 派遣例程。在模拟器的构建过程中，作者将此对象命名为“Emulator”，所以其带有路径的全部字符串为“\DosDevices\Emulator”。而且此名称对于用户模式下的应用程序是可见的。

（2）在用户模式下，利用 CreateFile（）函数不仅可以产生或者打开一个文件，它还可以打开一个已命名的对象，并返回一个用于访问该对象的句柄。这个函数的第一个参数就是要访问的设备的名字，模拟器的用户界将模拟器内核模块所用的名称“Emulator”传递给了该参数，如图 4-8 中的第（1）步操作。采用此种方法，在用户模式下就可以获得到指向内核中的模拟器核心模块的句柄（如图 4-8 中的第（2）步操作）。

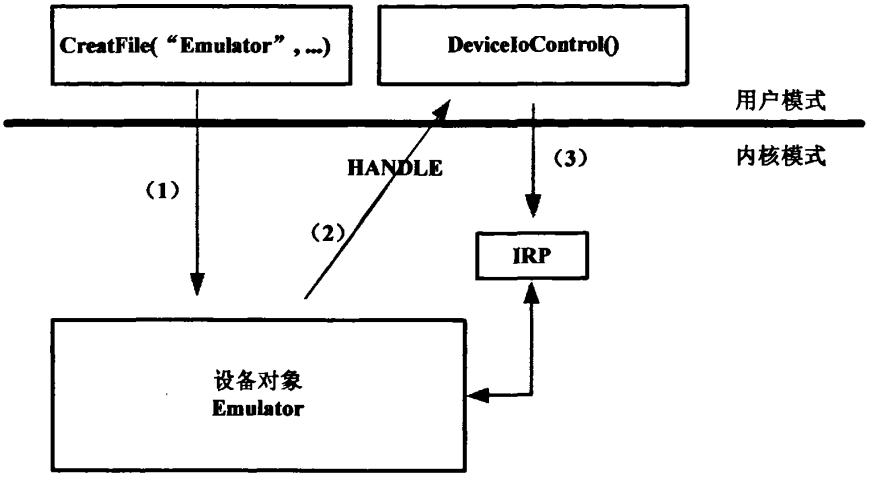


图 4-8 模拟器用户配置模块产生 IRP 的过程

2．产生 IRP

如第三章所述，既然 WDM 驱动本身具有处理 IRP 的能力，那么本文就可以使用模拟器的应用程序接口产生一个 IRP 然后通过产生的句柄把它传给下层的模拟器的核心模块，从而使相关的模块通过 IRP 可以得到它们所需要的参数。

因为产生 IRP 需要设备执行 IOCTL 操作，所以模拟器用户界面的应用程序使用标准 Win32 API 函数 DeviceIoControl（）来执行这样的操作。而对于模拟器的内核模块来说，这个 DeviceIoControl（）调用被 Windows XP 操作系统转化成一个带有 IRP_MJ_DEVICE_CONTROL 功能码的 IRP，如图 4-8 中的第（3）步操作。

DeviceIoControl（）的 Code(DWORD)参数是一个控制代码，它指出应用程序要执行何种控制操作，而这些控制代码的数值都是由模拟器自己定义并均予以实现的。控制代码的值与模拟器内核模块进行的操作是一一对应的，当上层需要内核模块进行某种的操作时，只要向下传递其对应的控制代码就可以了。函数 DeviceIoControl（）还具有描述为驱动程序提供数据缓冲区及其大小的两个参数 InputData（PVOID）和 InputLength（DWORD）。其中 InputData 和 InputLength

代表用于向驱动程序输入的缓冲区起始位置的指针以及数据缓冲区的大小。而其 OutputData（PVOID）和 OutputLength（DWORD）参数被视为接受驱动程序输出数据的缓冲区及其大小。驱动程序会应用程序以指出它返回了多少字节的输出数据。所以在用户模式下工作的用户界面只要将模拟条目的起始地址传入参数 InputData 即可。

3．传入数据

一个控制操作是否需要输入输出缓冲区要取决于其执行的功能。例如，一个接收驱动程序版本号的 IOCTL 可能仅需要输出缓冲区；一个仅用于通知驱动程序属于应用程序的某些事实的 IOCTL 可能仅需要输入缓冲区。有些操作可能同时需要这两种缓冲区，这完全取决于控制操作的具体内容。对于目前所构建的模拟器来说，用户需要将业务流的三元组及对应的控制信息传到内核模块中去，所以模拟器只需要输入缓冲区，但作者同时也保留了使用输出缓冲区的接口，为以后模拟器在功能上的拓展做准备。

在 Windows XP 中，用于缓冲用户模式数据到内核对象的方式一共有三种：BUFFERED 模式、Direct 模式和 Neither 模式。模拟器采用了第一种模式，这是因为使用 BUFFERED 模式时，I/O 管理器可以创建一个足够大的内核模式拷贝缓冲区(与用户模式输入和输出缓冲区中最大的容量相同)[12]。

用户配置模块的工作原理如图 4-9 所示。

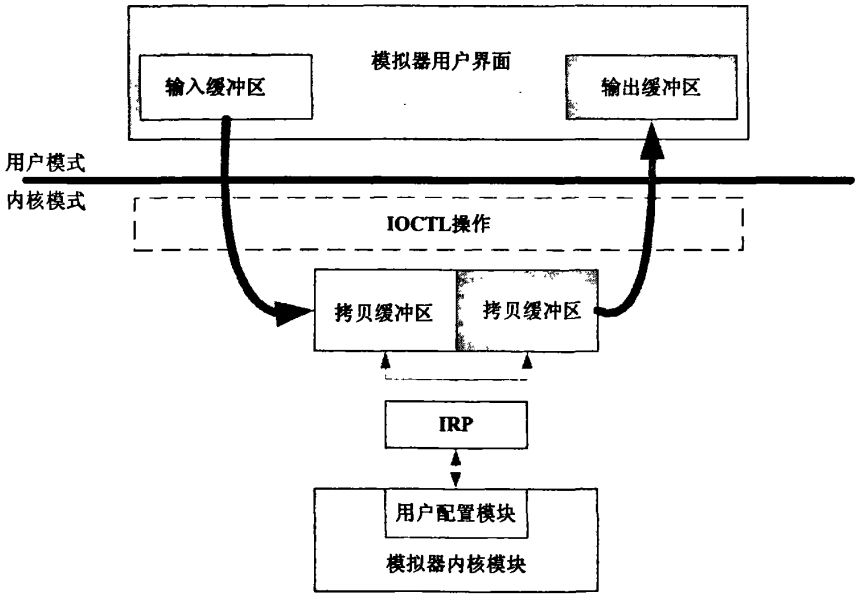


图 4-9 用户配置模块的工作原理

当内核对象的派遣例程获得控制时，用户在模拟器用户界面输入的数据被复制到一个拷贝缓冲区中以向模拟器的内核模块传送。在 IRP 完成之前，模拟器的内核模块首先会从拷贝缓冲区中复制自己需要的数据，然后向拷贝缓冲区填入需要发往应用程序的输出数据；当 IRP 完成时，模拟器内核模块设置 IPR 的 IoStatus.Information 域并放入拷贝缓冲区中的输出字节数，然后 I/O 管理器把数据复制到用户模式缓冲区并设置反馈变量。

4.4.2 封包匹配模块的设计和实现

通过模拟器可能有很多个业务流，模拟器通过 IP 地址、协议及端口号共同区分每个业务流。这些信息再加上对它们的控制信息（延迟，丢包率和带宽控制等）共同组成了一个模拟条目。

封包匹配模块会从封包截获模块中得到数据包的信息，从而找到与之对应的模拟条目，模拟器就是通过这样的手段找到对每一个数据包的控制信息实。

从模拟器的体系结构中（图 4-2）可以看出，每一个通过模拟器网络封包都要进行封包的匹配工作，正因为如此，此模块不仅很可能成为模拟器的性能瓶颈，还有可能影响那些背景业务流的性能。尤其是当用户载入的模拟条目数量过多时，接照一般的顺序查找可能会浪费过多的时间；如果当通过模拟器的业务流量很大时，数据包的到达会非常频繁，对于每一个数据包的查找操作耗费太多的查找时间可能成为影响模拟器性能的一个因素。为了能够使查找模拟条目的速度足够快以便跟上网络封包的到达速度，本文对于封包匹配模块的构建主要使用了文献[21]所提出的哈希查找算法。

在前一节介绍的用户配置模块中，模拟器将从用户界面处得到的模拟条目信息按照模拟条目的索引（Emulate Entry Index）存储到了哈希表中（如图 4-10 所示），其中模拟条目索引和封包信息之间的关系如下：

$$Emulate\_Entry\_Index = Hash(IP\ address, Protocol, Port\ Number)$$

此哈希算法中的 IP 地址、协议及端口号既可以是用户在模拟器的用户界面中输入的，也可以是模拟器从网络封包中读取的，同时它们还对应了一些控制信息（延迟、丢包率和带宽的数值）。其中由用户输入的这些信息会在模拟器启动时按照哈希索引存储到哈希表中。

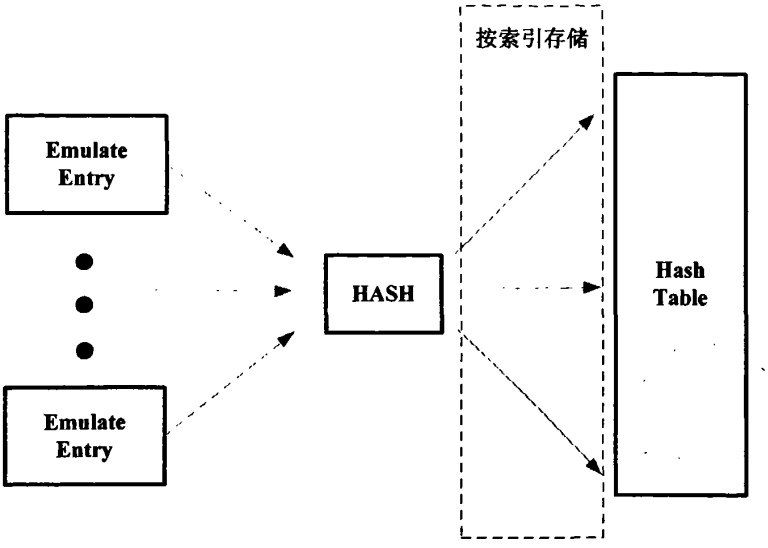


图 4-10 模拟条目的存储过程

当有网络封包到达时，模拟器会用此封包的网络信息重新进行一次哈希，以找到对其的控制信息，如图 4-11 所示。

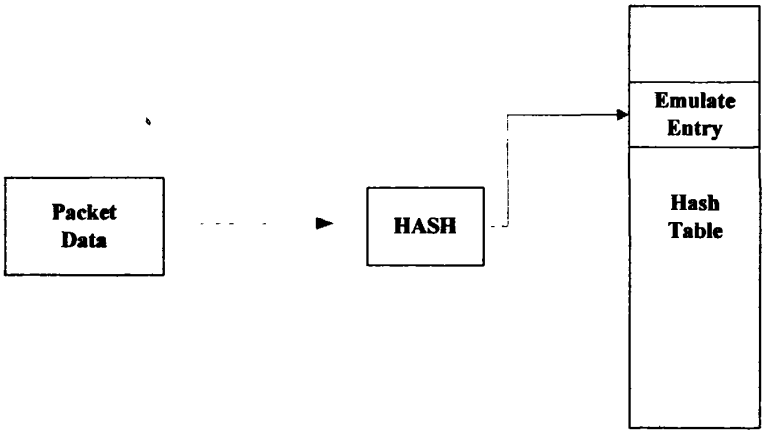


图 4-11 封包匹配过程

此模拟器所采用的这种哈希算法有以下几个特点：

（1）此种哈希算法没有取模运算，没有乘除法运算，只有大量的移位操作；

（2）这种哈希算法以 12 个字节为一个单位进行处理，而不像其它大多数哈希算法一样以 1 个字节为单位。而且它的指令是经过精心设计的，在理想情况下，在超流水线的 CPU 上性能可以提高一倍[21]；

（3）此种哈希算法对于文本、数字、压缩数据以及稀疏比特数组（sparse bits arrays）具有同样的性能。

此哈希算法产生 32bit 的返回值，即如果全部使用话一共需要 2^{32} 即 4G 个哈希条目。但是封包匹配模块始终是工作在内存中，不可能为其提供如此大的空间，所以在模拟器在一般情况下只使用了其返回值的后 8 位即 256 个条目。如果用户设置的模拟条目比 256 个多，模拟器便会使用后 16 位即 64K 个条目，这些条目对于模拟器来说已经足够多。之所以选择这两个数值（8 位和 16 位）是因为此哈希算法在哈希条目的个数为 2 的幂数时性能最好[21]。

为了能够加快查找的速度，模拟器还开辟了一个较小的缓存，它的大小为两个模拟条目的大小。上两次被匹配成功的模拟条目会被存储在缓存中优先进行与封包的匹配。使缓存仅有两个模拟条目的大小是因为在大多数情况下仅有一到两个业务流需要模拟器去进行控制。

经过测试发现，如果没有与缓存中的模拟条目相匹配，那么这种方法平均需要 62 微秒；如果与缓存中的模拟条目匹配，那么相应的平均处理时间仅需要 11 微秒。

另外，如果此种哈希算法产生冲突时，模拟器采用链表法来解决冲突问题。

4.5 关于模拟器代码的特点

与普通的应用程序不同，此模拟器将要成为一个内核模式的驱动模块加载到系统的内核中去，一旦运行时稍有差错，整个操作系统会面临溃的危险。所以在进行模拟器内核功能模块的构建过程中，作者在代码的级别上已经充分考虑了以下几点，当然这些特点本身也是 Windows 内核驱动所提倡的：

● 多线程及多处理器安全：

Windows XP 系统支持多线程程序，同时它又可以运行在多处理器的计算机上。Windows XP 使用对称多处理器模型，即所有的处理器都是相同的，系统任务和用户模式程序可以执行在任何一个处理器上，并且所有处理器都平等地访问内存。多线程以及多处理器的存在给设备驱动程序带来了一个困难的同步问题，因为执行在多个 CPU 上的代码或者多个线程可能同时访问共享数据或共享硬件资源。所以为了解决这样的问题，模拟器使用了大量的内核同步对象及自旋锁（spin lock），来解决多线程以及多处理器的同步问题。

● 可移植性：

Windows 要求内核模式驱动程序的源代码可以移植于所有 Window NT 平台。为了实现这种可移植性，模拟器的有关内核驱动的模块全部用 C 编写，并且只使用 ANSI C 标准规定的语言元素。在编写代码的同时避免了使用编译器厂商专有的语言特征，并避免使用没有被操作系统内核输出的运行时间库函数。这

样构建出的模拟器仅需要重新编译连接源代码，生成的程序就可以运行在任何新的 Windows NT 平台上。所以虽然此模拟器是工作在 Windows XP 操作系统中，但如果想让它工作在 Windows 2000 中，只需要将内核部分的源代码在 Windows 2000 的 DDK 中重新编译即可

● 可配置性：

作者在对模拟器的构建过程中没有对设备特征或某些系统设置作绝对假设，因为这些系统设置会随着平台的改变而改变。为了实现可配置性，模拟器避免了直接引用硬件，即使是在平台相关的条件编译块中也是这样。

本章详细地介绍了模拟器的各个核心模块及构建模拟器所采用的核心算法和实现细节，从中可以看出整个模拟器的设计及构建始终以高率为中心。在下一章中，本文将对模拟器的核心功能模块（延迟，带宽限制及丢包功能）进行较为深入的测试。

第五章 模拟器性能测试

本章针对模拟器的最重要的特性即支持 IPv6 的功能进行了测试，并对模拟器的延迟功能、带宽限制及丢包功能进行了实验测试。

5.1 IPv6 实验床的搭建

如图 5-1 所示，在一台装有 Windows XP 的 PC 机上（即中间的结点），先对其启用 IPv6 的支持，然后将本文所构建的模拟器安装在这台 PC 机上，此 PC 机为 DELL Dimension 3000 系列，其主要的硬件配置为 Intel Pentium 4 CPU 2.80GHz、1GMB 内存。作者在其上安装了两个以太网卡，图 5-1 中的网卡 1 为 Realtek RTL8139 Family Fast Ethernet 系列（100Mbps），网卡 2 为 Intel PRO/100 VE Network 系列（100Mbps）两个以太网卡的 IPv6 地址的配置如图 5-1 所示。

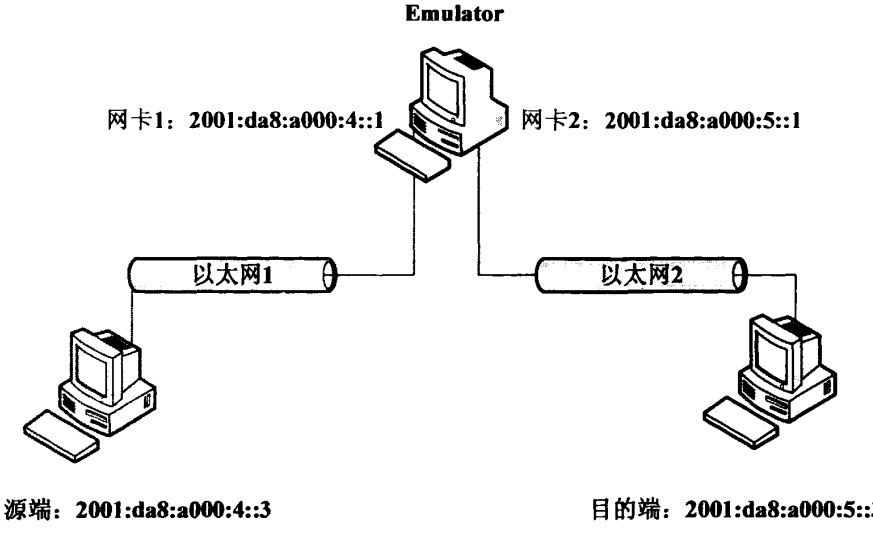


图 5-1 采用模拟器所搭建的 IPv6 实验床

此实验环境需要构建两个以太网，本文使用了两个交换机组成了以太网 1 和以太网 2。工作在以太网 1 中的交换机为 Cisco Catalyst 2900 XL（10 Base/100BaseTx）系列，工作在以太网 2 中的交换机为 EDIMAX ES-3116RL（10/100Mbps）系列。

源端是一台运行在以太网 1 中的 PC 机（图 5-1 的左边结点），其主要硬件配置为 Intel Celeron CPU 1.4GHz、512MB 内存，Intel PRO/100 VE Network 系列（100Mbps）以太网卡，其 IP 配置如图 5-1 所示。

目的端是一台运行在以太网 2 中的 PC 机（图 5-1 的右边结点），其主要硬件配置为 Genuine Intel CPU T2250 1.73GHz、512MB 内存，Realtek RTL8168/8111 Gigabit 以太网卡，其 IP 配置如图 5-1 所示。

在装有网络模拟器的中间结点上只要打开 Windows XP 中对 IPv6 的路由功能，两边的结点就可以进行通信了。

5.2 本文所选择的测试工具

对于测试工具本身来说，其自身不仅要满足一定的准确性之外，还要具有一定的易用性和可视性。但对于本文的工作来说，只采用单一的测试工具也许不能满足以上的要求。为了能获得较为准确的实验结果并方便对实验结果的收集，本文在对模拟器不同的功能进行测试时采用了不同的测试工具。

由于 Windows XP 操作系统本身已经提供了通过用户键入的命令（即 ping6 命令）来产生 ICMPv6 Echo Request 网络封包的功能[24]，而且此功能还可以将所测试的结果进行输出以便观察，所以在 IPv6 环境下，为了能使模拟器所产生延迟的数值具有一定的可测量性和可视性，本文使用 ICMPv6 Echo Request 来测试网络模拟器的封包延迟功能。

另外，本文还使用了 Windows 平台下一个网络性能测试工具 Iperf[30]，它是一个 TCP 和 UDP 的性能测量工具，能够提供网络吞吐率信息，以及丢包率、最大段和最大传输单元大小等统计信息。使用它发送的 TCP 流可以测量出链路的最大带宽；而使用 UDP 流可以测量出丢包率等信息。Iperf 已经有多个版本，本文所使用版本的是 1.7.0，这是一个已经开始支持 IPv6 的版本。

5.3 模拟器性能开销（overhead）测试

如前文所述，本文所构建的网络模拟器在整个测试环境中还充当着路由器的角色，所以此模拟器自身的性能开销对整个网络的性能有着极大的影响，有必要对其自身所带来的性能开销做一下测试。

由上文可以知道，本文所构建的网络模拟器是作为一个驱动模块加载到操作系统中的内核中去，它其实上是工作在 Windows XP 操作系统中的 TCP/IP 协议栈中。所以，所有经过网络模拟器的封包都会经过模拟器，而对于那些属于背景

业务流的网络封包同样需要经过模拟器的处理。而模拟器对它们的最终操作就是将它们交给模拟器的上层，但是在这之前需要经过资源的分配（为封包匹配做准备）和封包的匹配（主要为哈希操作）操作，所以模拟器的操作对于这些数据包可能会有一定的影响。

采用本文 5.1 节的介绍的硬件配置，经测试发现，模拟器在分配内存及进行其它的相关工作总共需要花费平均为 54 微秒的时间；而进行哈希操作如第四章第四节所述进行一次哈希操作需要花费 62 微秒。另外，此模拟器调用 NDIS 中的 NdisMIndicateReceivePacket（）函数来向上层进行封包的传递，整个相关的工作平均需要 25 微秒。所以当一个背景业务流的封包到来时，这个封包因为模拟器的存在而引入的平均额外操作时间 $t_{overhead}$ 为：

$$t_{overhead} = t_{pre} + t_{match} + t_{Hash} + t_{Indication}$$
$$= 54 + 11 + 62 + 25 = 152 \quad (us)$$

而对于那些真正需要模拟器进行操作的封包，除了需要经过与背景业务流的封包相同的操作以外，由于 NDIS 中间层的限制，还要必须对其进行缓存的分配及封包描述符的封装，这些都是模拟器的操作所带来的额外负担。经测试发现，在 NDIS 中间层中进行单个网络封包的封装工作需要花费 11 微秒，所以当一个需要模拟器进行操作的业务流的封包到来时，没有与缓存中的模拟条目匹配所引入的额外操作时间为：

$$t_{overhead} = t_{pre} + t_{match} + t_{Hash} + t_{encap} + t_{Indication}$$
$$= 54 + 11 + 62 + 11 + 25 = 163 \quad (us)$$

如果当网络封包的信息与缓存中的模拟条目相匹配时，模拟器引入的额外操作时间为：

$$t_{overhead} = t_{pre} + t_{match} + t_{encap} + t_{Indication}$$
$$= 54 + 11 + 11 + 25 = 101 \quad (us)$$

因为用户所设定的延迟的最小单位为毫秒级，由此可以看出模拟器的性能开销还是相对比较小的。

需要说明的是，网络模拟器所带来的性能损耗可能会因 CPU 性能的不同而不同。

当启动模拟器但设定它不对任何业务流进行操作时（即此时模拟器仅相当于路由器），经测试发现，从源端到目的端的平均延迟几乎为 0 毫秒（如图 5-2 所示）；采用 Iperf 产生的 TCP 业务流测量出最大的链路带宽约为 88.9Mbps；通过采用 Iperf 产生的 UDP 业务流经测量发现在源端和目的端之间没有发生丢包现象。

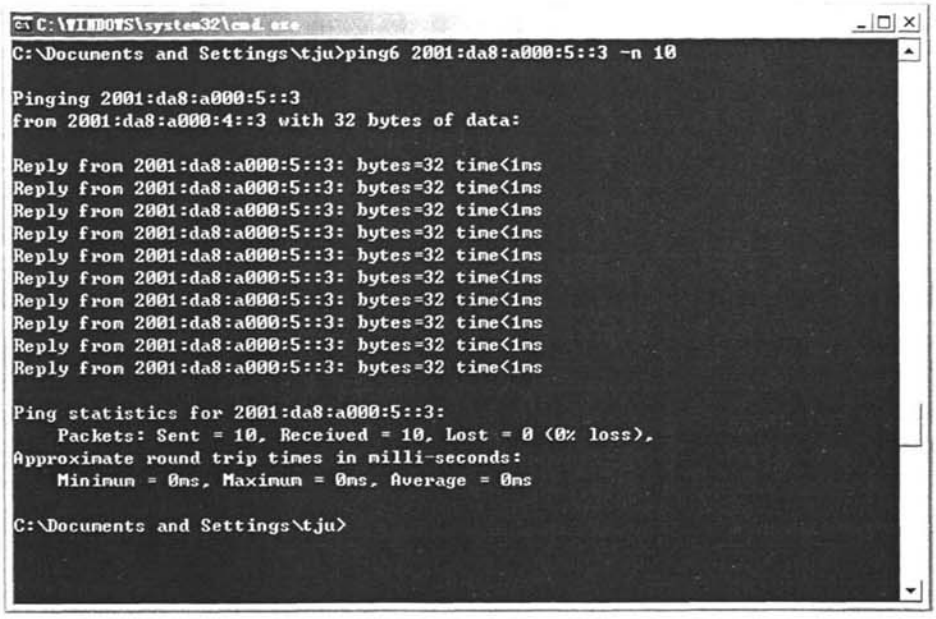


图 5-2 模拟器没有进行操作时的延迟测试

5.4 模拟器核心功能测试

此小结主要对网络模拟器的核心功能如延迟，带宽限制和丢包功能进行测试，此部分的测试仍然采用如图 5-1 的实验床。

5.4.1 延迟功能测试

在模拟器中将源端到目的端的延迟设为 50 毫秒，图 5-3 为采用 ping6 命令所测量到的网络延迟结果的截图。

为了能更好地测试网络模拟器的延迟功能，本文共测试了模拟器先后所产生的 11 组不同数值的延迟，其变化范围从 0 毫秒至 100 毫秒，平均每 10 毫秒取一个测试点；而且每个测试点使用 ping6 命令一共发送 100 个 ICMPv6 Echo Request 的网络封包，所采用的命令为：ping6 2001:da8:a000:5::3 –n 100。测试结果与期望结果相互间的关系如图 5-4 所示，图中每个测试点的结果为该次所产生的 100 个网络封包延迟数值的平均值。

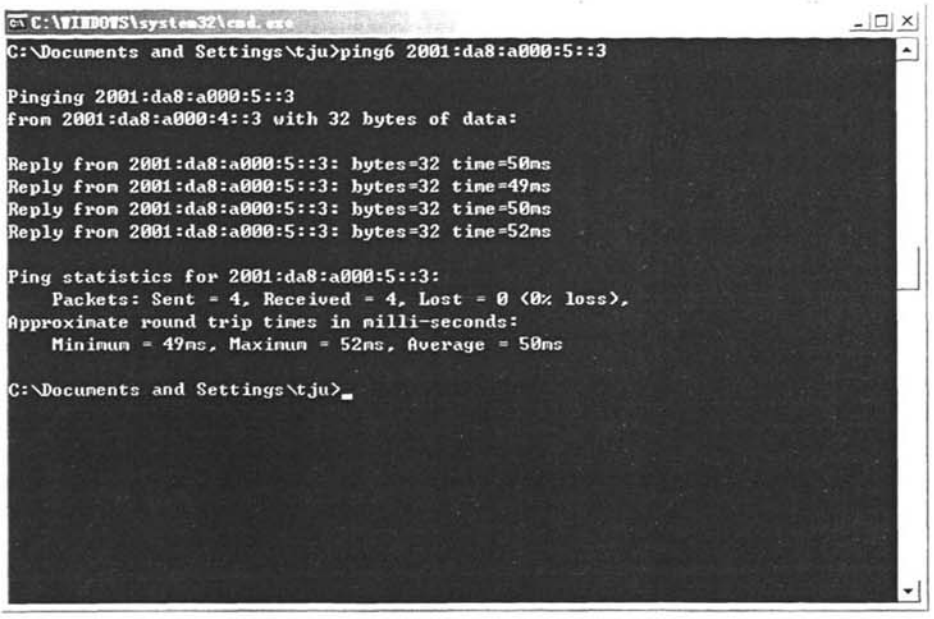


图 5-3 Windows XP 下 ping6 命令所测试出的网络延迟

在本文的第四章已经介绍过，此模拟器只是按用户所输入的信息控制进入模拟器的封包，尽管 ICMPv6 Echo Request 协议会产生一个双向的业务流（请求包和应答包），但模拟器只是控制请求包而没有控制返回的应答包，所以图 5-3 中所看到的延迟是 50 毫秒而并非 100 毫秒。

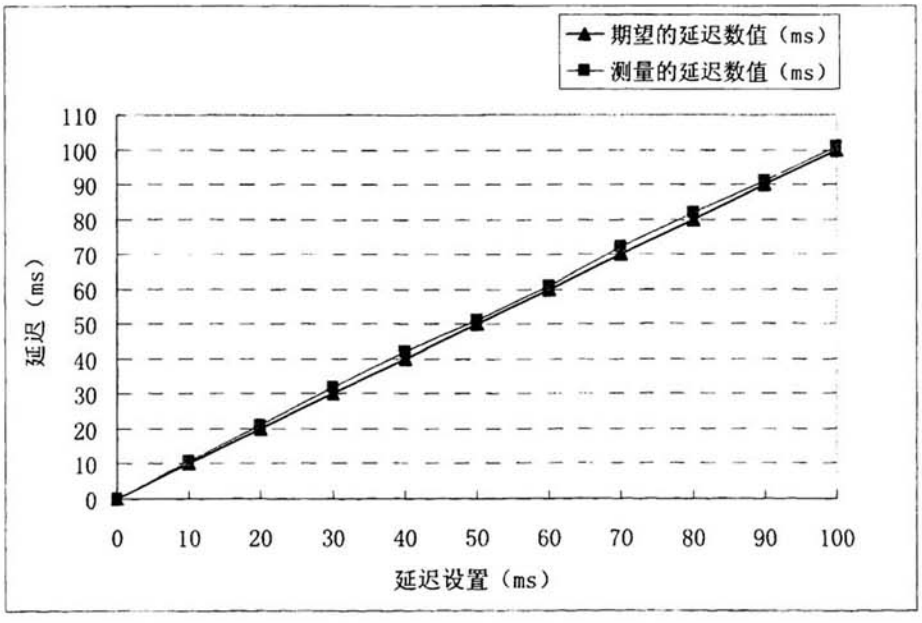


图 5-4 网络模拟器延迟功能测试的实验结果

5.4.2 带宽限制功能测试

在图 5-1 的实验环境中，本文对模拟器的带宽限制功能进行了测试。

在带宽限制的测试中，本文使用 Iperf 作为测试工具。在模拟器中将带宽限制设为 100kbps，源和目的端使用 Iperf 来建立一个 TCP 的业务流，其实验结果的截图如图 5-5 所示。

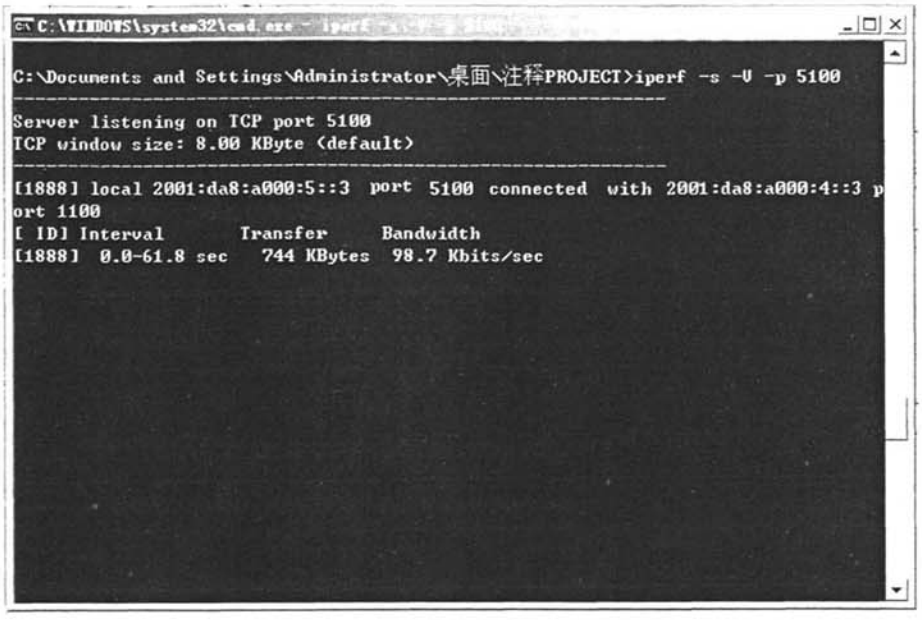


图 5-5 带宽实验的测试结果截图

在带宽限制功能的测试实验中，本文一共使用 Iperf 测试了 10 组不同的带宽值，其范围从 100kbps 至 1000kbps 不等，所使用的传输协议为 TCP，使用的工具为 Iperf 1.7.0，端口号为 5100。此次实验的测试结果如图 5-6 所示，图中每个测试点的结果为该次所产生的 60 秒内 TCP 业务流的平均带宽。

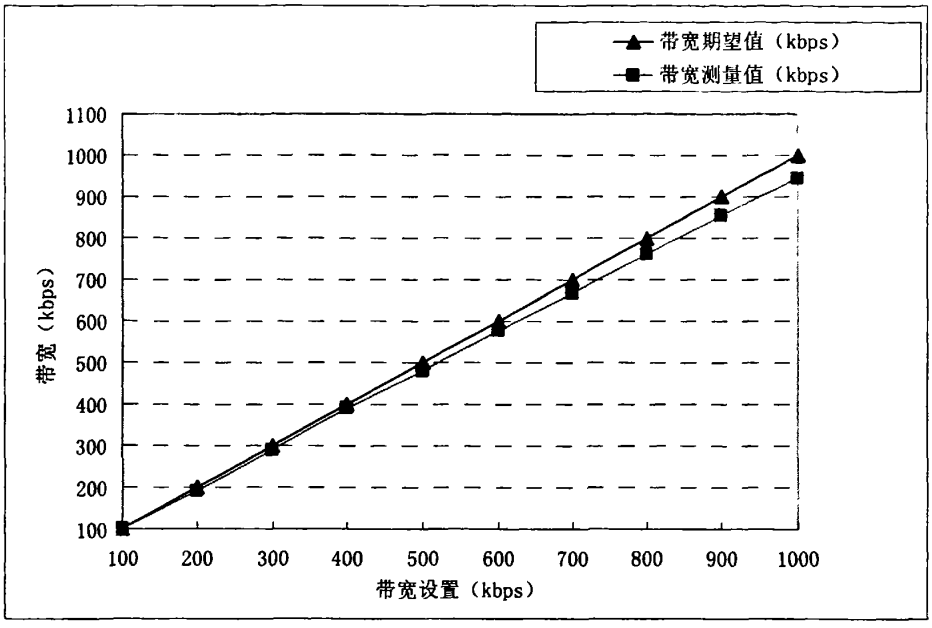


图 5-6 带宽实验的测试结果

需要额外说明的是，在本文所构建的模拟器中，如果用户将最大的带宽数值设为 0，那么模拟器将会忽略此值从而不会去限制对应业务流的带宽。如果用户想要模拟带宽为 0 的目标网络，那么用户可以将对应的丢包率设定为 100%而不用在带宽上做任何操作。

5.4.3 丢包功能测试

在如图 5-1 的实验环境中，本文对模拟器所产生的丢包效果进行了测试。对于丢包功能的验证，本文同样使用了网络性能测试工具 Iperf，当使用它产生 UDP 的业务流时，其运行接收端的一方可以测量出业务流中的丢包率。

使用 Iperf 在源到目的端产生一个 UDP 的业务流，使用的目的端口号为 5100，在网络模拟器上，将对应此业务流的丢包率设为一个非零值，所以当此业务流经过网络模拟器时，会有一部分网络封包要被丢掉，其丢到的比例与用户所设定的丢包率的大小有关。在网络模拟器中将丢包率为 10%，图 5-7 为在使用 Iperf 接收 UDP 业务流时所测量到的丢包率的截图。

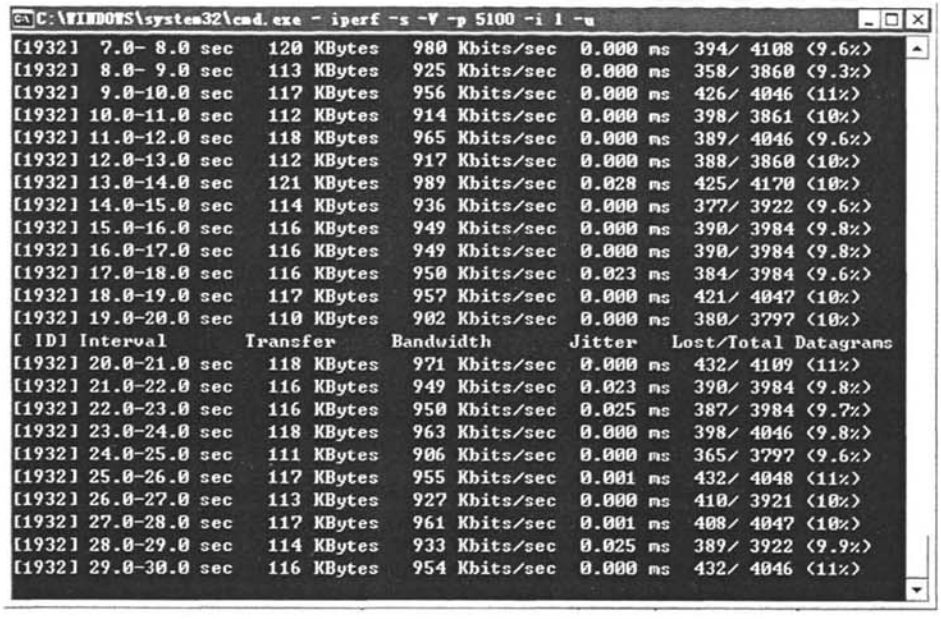


图 5-7 使用 Iperf 接收 UDP 业务流时所测量到的丢包率

如图 5-7 所示，在图的最右边的一列，即括号中的百分数即为使用 Iperf 接收端所测量到的实际丢包率。从图中可以看出由 Iperf 测量到的实际丢包率与模拟器所产生的丢包率是非常接近的。

为了进一步测试模拟器的丢包功能，本文在网络模拟器中一共设定了 11 组不同的丢包率，其范围为 0 至 100%，每 10 个百分点取一个测试点，每个测试点均为使用 Iperf 来产生持续 60 秒的 UDP 业务流，当这个业务流经过模拟器时，会有一部分的数据包被模拟器丢掉。整个实验结果如图 5-8 所示。每个测试点的数值均为 Iperf 接收端测量到的 60 秒内丢包率的平均值。

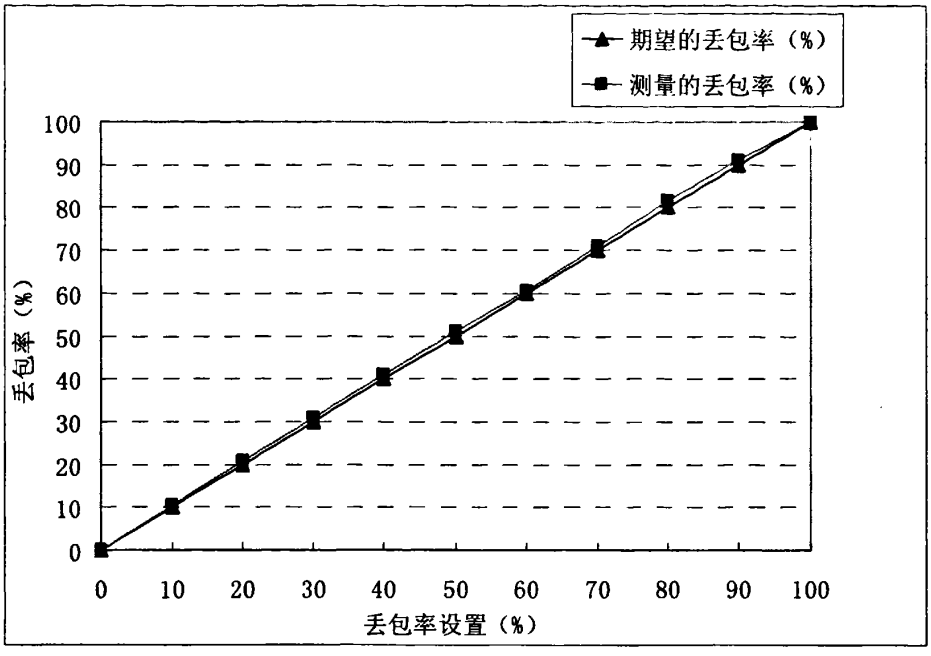


图 5-8 网络模拟器丢包功能测试的实验结果

需要说明的是，当在网络模拟器中将丢包率设为 100%时，所有符合用户所设定的业务流的网络封包都会被模拟器所丢弃，这种结果同源端和目的端之间的网络被切断的效果是一样的。所以在这样的实验中，虽然通过 Iperf 的接收端不会接收到任何一个由源端所发送出来的网络封包，但这也同时证明了此时模拟器所产生的丢包率是 100%。

5.4.4 综合功能测试

用户在使用模拟器时可能需要的功能不只一种，同样，本文所构建的模拟器可以对同一业务流进行综合的操作。

由于带宽限制同延迟在实现的本质上是一样的，而且为了使实验结果更加具有可视性，在这里本文给出了测试了丢包与延迟综合的操作。图 5-9 为丢包率为 40%，延迟为 100 毫秒的实验结果的截图，图中“请求超时”的字样表明该封包已经被网络模拟器丢弃。

从图中可以看出，延迟的功能与丢包的功能均工作得很好，没有发生相互影响的现象。

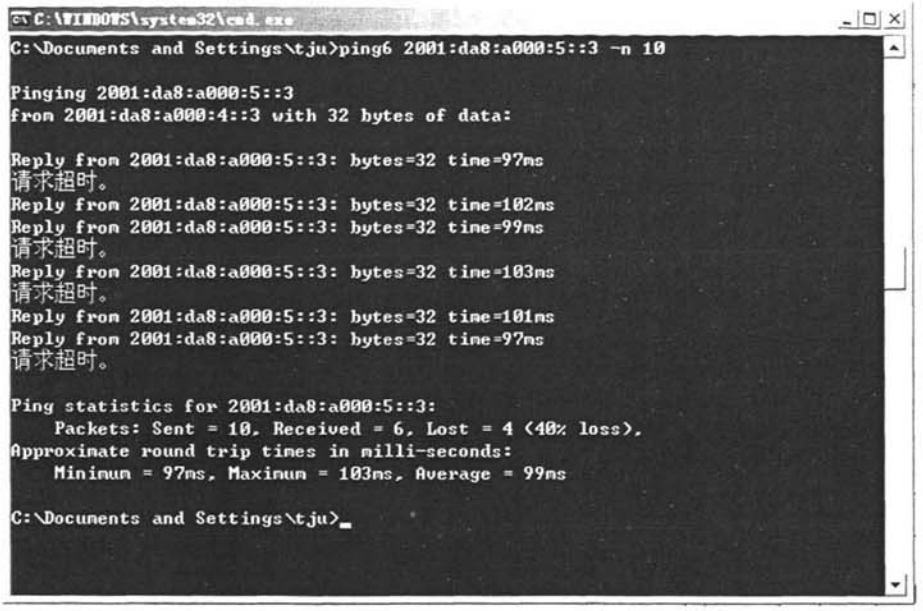


图 5-9 延迟与丢包功能的综合实验截图

5.5 实验结果分析

从上一节的各个实验中可以看到，无论是延迟，丢包操作还是带宽限制，模拟器都可以很好地完成用户所设定的模拟条件，但通过实验发现实际测量到的数值还是和理论值有一定的偏差。

对于延迟测试和带宽限制测试结果来说，本文发现对于延迟其测量出的平均值总是大于理论值，而带宽的测试值正好与其相反，即测量的平均值小于理论值。本文认为产生这种现象主要有两方面的原因：一方面，Windows XP 并不是实时系统，而 Windows XP 操作系统的定时器并没有十分严格地遵循定时时间，理论上只有真正的实时系统才能达到定时非常精准的要求。在 Windows XP 中，当采用 KeWaitForSingleObject（）函数进行等待时，它的定时器有时候会滞后一小部分时间才会被触发，这就会增加一部分模拟器的响应时间。另一方面，由于此网络模拟器在实验床中充当路由器的角色，所以它在源和目的端之间转发网络封包时也需要一部分操作时间，虽然相关的操作占用的时间极少，但这部分时间也同样被计算在延迟测试及带宽限制的测试结果之内。

所以，当定时器的真正触发时间大于其理论时间时，实验所测量出的延迟会

大于理论值，而实际的带宽便会变得比理论值小一些。

对于丢包率测试的结果，实际测量出的丢包率会略大于理论值。这种现象主要是因为模拟器所产生的伪随机数的数量毕竟有限，在这种情况下丢包的操作可能会发生“聚集”现象，所以模拟器在使用这个伪随机数发生器时会与真正的均匀丢包有误差。

另外，从综合的测试结果中可以看出，模拟器各功能之间并没有很强的耦合性，无论是单个功能启动还是所有功能全部启动，模拟器都能完成好各个部分的要求，模拟出用户所需要的目标网络。

经过计算，延迟的平均误差为 3.8%，带宽限制的平均误差为 6.0%，而丢包率的平均误差为 3.4%。本文认为，网络模拟器的误差相对来说可以接受，基本上已经达到了预期的工作目的。

第六章 结论与展望

前面各章节已经对网络模拟器的构建工作及网络模拟器的性能做了一些测试，本章对全文工作所得出的一些结论进行了归纳，并对以后工作的进一步开展方进行了展望。

6.1 本文结论

网络模拟（Network Emulation）被视为仿真测试和全真测试两种测试方法的集成，这是一种在半人工的环境中来运行真正代码的网络测试方法。与仿真测试和全真测试这两方法来比较，网络模拟可以最大的减少网络测试的投入代价。对于在实验室进行研究的人员，网络模拟器（Network Emulator）可以使他们在实验室的条件下就能模拟网络的各种条件。可以说，网络模拟技术为网络测试工作提供很多的方便之处。

IPv6 被认为是下一代互联网络的核心，而 Windows 仍是目前世界上最流行，用户人数最多的操作系统。不过令人遗憾的是，目前还没有广泛应用在 Windows 平台上并工作在 IPv6 环境下的网络模拟器，这势必会对 IPv6 网络性能的价评工作、尤其是网络模拟器的发展造成一定的不利影响。

本文便是围绕着如何构建 Windows 平台上 IPv6 的网络模拟器这一问题来展开讨论。首先，本文充分研究了几种较为经典的非 Windows 平台上网络模拟器的工作原理，在此基础之上，提出了所要构建的 Windows 平台下 IPv6 网络模拟器需具有的特点。接下来对如何使用 NDIS 驱动及 WDM 驱动来构建 Windows 平台下的网络模拟器进行了相应的介绍和讨论。

随后，本文跟据 Windows 操作系统自身的特点及已存在的非 Windows 平台上的网络模拟器为参考，给出了此模拟器体系结构，并对模拟器中核心模块的关键算法进行了较为详细的介绍，具体包括模拟器中定时队列、丢包功能、带宽限制及封包匹配等算法。另外，本文还对这些模块的具体构建过程和技术手段如网络封包的截获及 IRP 的产生与使用进行了阐述。

最后本文使用此模拟器在 IPv6 的环境下对其性能及模拟效果进行了深入的测试， 具体包括性能损耗（overhead）、延迟、带宽限制及丢包率的测试。实验结果表明，该网络模拟器能够比较有效地对各种网络参数（带宽、延迟和丢包率）进行改变和控制，基本上达到既定的工作和研究的目的。

6.2 工作展望

虽然本文所实现的 Windows 下的网络模拟器能够模拟出 IPv6 网络中的各种网络条件，但是它的功能还很单薄，还有许多工作需要进行。在当前已有工作的基础上，对模拟器的构建工作还可以在以下几个方面进行更进一步的完善：

（1）进一步减少网络模拟器所占用的系统资源。

由本文前面的描述可知，网络模拟器在工作中所使用的内存资源均为 Windows XP 中内核的非分页内存，这种内存属于操作系统中比较稀有的资源。在以后的模拟器的构建过程中，可以将一些一次性及不重要的工作放在分页的内存中进行，以节省操作系统本身的资源。

（2）引入更多的功能。

除了带宽，延迟和丢包率以外，在IPv6网络中仍然有许多其它的网络特性如延迟抖动（Jitter）、封包失序（Out-of-Order）及比特误码率(Bit Error Rate)等同样值得关注，而对这些功能的模拟也同样需要加入到本文所构建的模拟器中，以不断增加网络模拟器的功能，以满足不同用户的测试需求。

（3）提供对多个结点的模拟。

随着因特网的发展，网络的拓扑结构也越来越复杂，而采用多个模拟器来共同组成一个庞大的网络结构的做法有时可能会花费很高的成本。因此，网络模拟器也应该具有可以模拟多个网络结点的功能，从而为大规模的网络测试工作带来方便。

（4）向无线网络扩展。

随着网络技术的日益成熟，无线传输网络和无线局域网也被人们提出并实现。和传统有线网络相比，无线网络具有组网方便、节点的可移动性高的特点，这使得这种类型的网络具有强大的生命力和广泛的应用前景。但与此同时，无线网络的高丢包率与高差错率更加需要采用网络模拟这一手段进行测试。因此，针对这种发展趋势，本模拟器应该提供有关无线网络的功能模块，使模拟器的应用领域更加广泛。

参考文献

[1] 王行刚，雷擎，计算机网络模拟方法与工具[J]，通信学报，2001，22（9）：84 - 90.

[2] 王国平，芮筱亭，仿真系统可信度分析[J]，现代防御技术，2006，34（2）：68 - 72.

[3] Ldshin Pete，IPv6 详解（沙斐译）[M]，北京：机械工业出版社，2000. 1-7.

[4] Rizzo Luigi. Dummynet:：a simple approach to the evaluation of network protocols[J]. ACM Computer Communication Review，1997. 31 - 41.

[5] Pei Zheng，Lionel M.Ni. EMPOWER：A Network Emulator for Wireline and Wireless Networks[C]. INFOCOM 2003 Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies，2003. 1933 - 1942.

[6] Allman M.，Caldwell A.，Ostermann S.. One: The Ohio network emulator[EB/OL]. Ohio University Tech Report，1997. 1 - 8.

[7] PacketStorm Communications Inc. White Papers of PacketStorm IP Network Emulator[EB/OL]. http://www.packetstorm.com/IP_Network_Emulation.pdf

[8] NIST Internetworking Technology Group. NIST network emulation package[EB/OL]. http://snad.ncsl.nist.gov/itg/nistnet/.

[9] Windows XP DDK Document[R]. Network Drivers：2.2 NDIS Drivers.

[10] 朱雁辉，Windows 防火墙与网络封包截获技术[M]，北京：电子工业出版社，2002. 39 - 76.

[11] Kayssi Ayman，El-Haj-Mahmoud Ali. EmuNET: a real-time network emulator[J]. Proceedings of the 2004 ACM symposium on Applied computing，2004. 357 - 362.

[12] Walter Oney. Programming the Windows Driver Model[M]，美国：Microsoft Press，2000. 23 - 358.

[13] Jones Anthony，Ohlund Jim. Network Programming for Windows[M]，美国：Microsoft Press，2000. 102 - 675.

[14] 张威，C#语言基础教程[M]，北京：人民邮电出版社，2001. 274 - 282.

[15] Jeffrey Richter，Windows 核心编程（王建华译）[M]，北京：机械工业出版社，2000. 463 - 477.

[16] Stevens W.Richard，TCP/IP 详解（范建华，张涛等译）[M]，北京：机械工业出版社，2000. 15 - 37.

[17] Microsoft Developer Network[EB/OL]. http://msdn2.microsoft.com

[18] Vahdat Amin，KenYocum，Kevin Walsh. Scalability and Accuracy in a Large-Scale Network Emulator[C]. ACM SIGOPS Operating Systems Review，2002. 271 - 284.

[19] Chen H.C.，Asau Y.. On generating random variates form an empirical distribution[J]. AIEE Transactions，1974. 163 - 166.

[20] Wang S.Y.，Kung H.T.. A simple methodology for constructing extensible and high-fidelity tcp/ip network simulators[C]. INFOCOM 1999 Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies，1999. 1134 - 1143.

[21] Robert J.Jenkins，Jr.. Hash functions for hash table lookup[EB/OL]. http://www.burtleburtle.net/bob/hash/evahash.html，1997.

[22] Avvenuti M.，Vecchio A.. Application-level network emulation: the EmuSocket toolkit[J]. Journal of Network and Computer Applications，2006，29（4）：343 - 360.

[23] Noble Brian D.，Satyanarayanan M.，Giao T. Nguyen. Trace-based mobile network emulation[J]. ACM SIGCOMM Computer Communication Review，1997，27（4）：51 - 61.

[24] Regis Desmeules，Cisco IPv6 网络实现技术（王玲芳，张宇，李颖华等译）[M]，北京：人民邮电出版社，2004. 215 - 225.

[25] Zhigang Jin，Xiuli Wu，Yantai Shu. Designing and implementing of a wireless network emulator[J]. Electrical and Computer Engineering，2004. 341-344.

[26] Fall K.. Network Emulation in the VINT/NS Simulator[EB/OL]. Fourth IEEE Symposium on Computers and Communications，1999. 59 - 67.

[27] 驱动开发网 NDIS 专区[EB/OL]，http://bbs.driverdevelop.com

[28] Ahn J.S.，Danzig P.B.，Liu Z.. Evaluation of TCP vegas: Emulation and experiment[C]. In Proceedings of SIGCOMM，1995.

[29] Brian White，Jay Lepreau，Leigh Stoller. An integrated experimental environment for distributed systems and networks[J]. ACM SIGOPS Operating Systems Review，2002. 255 - 270.

[30] Iperf Document[EB/OL]. http://dast.nlanr.net/projects/Iperf/

[31] Pei Zheng，Lionel M.Ni. EMWIN: emulating a mobile wireless network using a wired network[J]. In Proceedings of the 5th ACM international workshop Wireless mobile multimedia，2002. 64 - 71.

[32] Jay Chen，Diwaker Gupta，Vishwanath K.V.. Routing in an Internet-scale network emulator[C]. The IEEE Computer Society's 12th Annual International Symposium on Modeling，Analysis，and Simulation of Computer and Telecommunication Systems (MASCOTS 2004)，2004. 275 - 283

发表论文和参加科研情况说明

发表的论文：

[1] Bing Zhao，Zhigang Jin，Yantai Shu，Yu Li，“RENEW: A Real-time and Effective Network Emulator of Windows for IPv6”，Next-Generation Communication and Sensor Networks 2007，In Proceedings of SPIE，2007，EI Indexed

参与的科研项目：

致　谢

本论文的工作是在我的导师金志刚教授的悉心指导下完成的，金志刚教授严谨的治学态度和科学的工作方法给了我极大的帮助和影响。从导师身上，我学到的不仅仅是其求实严谨、勤勤恳恳的治学作风，更有其作为一名学者所具有的严于律己、宽厚待人等高尚品格。在此衷心感谢两年来金志刚老师对我的关心和指导。

张连芳教授、赵政教授、许林英老师及葛卫民老师对于我的科研工作和论文都提出了许多的宝贵意见，在此向各位老师表示衷心的感谢。

在实验室工作及撰写论文期间，师兄陈奇延、师妹岑丹丹等同学对我论文中的算法等研究工作给予了热情帮助，在此向他们表达我的感激之情。

另外也感谢我的家人尤其是我的父母，他们的理解和支持使我能够在学校专心完成我的学业。