南开大学

硕士研究生毕业（学位）论文

姓　　名：吴筱桉

年　　级：2005 级

专　　业：计算机应用技术

研究方向：网络与信息系统

论文题目：VNC 系统中 RFB 协议分析及视频播放性能改进

完成日期：2008 年 5 月

导　　师：张建忠 教授

南开大学信息技术科学学院

二零零八年五月一日

南开大学学位论文电子版授权使用协议

（请将此协议书装订于论文首页）

论文《VNC系统中RFB协议分析及视频播放性能改进》系本人在南开大学工作和学习期间创作完成的作品，并已通过论文答辩。

本人系本作品的唯一作者（第一作者），即著作权人。现本人同意将本作品收录于“南开大学博硕士学位论文全文数据库”。本人承诺：已提交的学位论文电子版与印刷版论文的内容一致，如因不同而引起学术声誉上的损失由本人自负。

本人完全了解《南开大学图书馆关于保存、使用学位论文的管理办法》。同意南开大学图书馆在下述范围内免费使用本人作品的电子版：

本作品呈交当年，在校园网上提供论文目录检索、文摘浏览以及论文全文部分浏览服务（论文前16页）。公开级学位论文全文电子版于提交1年后，在校园网上允许读者浏览并下载全文。

注：本协议书对于“非公开学位论文”在保密期限过后同样适用。

院系所名称：信息技术与科学学院

作者签名：吴筱梭

学号：2120050406

日期：2008年 5月 22日

南开大学学位论文版权使用授权书

本人完全了解南开大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：吴筱桉

2008 年 5 月 22 日

经指导教师同意，本学位论文属于保密，在　　年解密后适用本授权书。

指导教师签名：		**学位论文作者签名：**	
解　密　时　间：	年　　月　　日		

各密级的最长保密年限及书写格式规定如下：

内部　5 年（最长 5 年，可少于 5 年）
秘密★10 年（最长 10 年，可少于 10 年）
机密★20 年（最长 20 年，可少于 20 年）

南开大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：吴筱枝

2008 年 5 月 22日

摘要

随着网络技术的成熟，基于网络的计算模式占据了越来越重要的地位。由于瘦客户端系统为实现基于网络的计算模式提供了良好的支持，因而越来越受到人们的重视。

瘦客户端系统中所有计算都在服务器中完成，用户数据也存储在服务器上。客户端负责同用户进行交互、完成输入和输出工作，通常不具备或仅具备有限的处理能力、不存放任何与用户相关的数据、不存在状态变化。客户端的无状态化意味着用户从任意客户端连接到同一服务器，得到的工作环境是相同的，这很好的满足了基于网络的计算模式的要求。

本文首先对瘦客户端系统设计中的一些基本问题进行了介绍，分析了不同设计策略的选择对系统整体性能可能产生的影响，并结合几种常见的瘦客户端系统，对其整体性能进行了对比。同时，针对 VNC 系统中视频播放性能不佳的问题，在对 RFB 协议进行详细分析的基础上，指出 VNC 系统对视频播放应用缺乏良好支持的原因是采用了 Client-pull 模式的显示更新策略。为改善其对视频播放应用的支持，本文将 VNC 系统的显示更新策略由 Client-pull 模式改为 Server-push 模式，并在开源项目 Vino 的基础上实现了设计改进。

为了验证该设计改进产生的性能提升，本文对常见的性能测量方案进行了分析和比较，选择 slow-motion 基准测量作为改进的性能测量方案，对 VNC 系统中的视频播放进行了性能测量。测量结果表明，本文提出的 VNC 系统的设计修改方案提高了视频播放应用的性能，能够提供更好的用户体验。

关键词：瘦客户端，虚拟网络计算，视频播放，性能测量方案

Abstract

With the maturity of network technology, the network computing model is becoming more and more important. Thin-client systems provide a good support for network computing model, thus its importance and practical applications are continually growing.

In thin-client systems all the computations are executed in the server, and user's data are all stored in the server. The client is responsible for interacting with the user and perform I/O operations and has no or limited computation capacity and does not store any user related data and has no change in status. The stateless client means that the user can connect to the server from any client, and get the same working environment. Thus the thin-client systems provide good support for network computing model.

This dissertation first introduces some basic designing issues in thin-client systems, then analyzes the impact of the choice of different design strategies on overall system performance and then, taking several popular thin-client systems for example ,compares the overall system performances,.

VNC system lacks good support for video applications. Based on analyzing the RFB protocol in detail, this dissertation figure out that the reason why the VNC system lack good support for video application is that VNC system adopts the Client-pull mode as its display-update strategy. In order to improve its support for video application, this dissertation changes the VNC system's display-update strategy from Client-pull mode to Server-push mode, and implements that designing changes based on open source projects Vino.

This dissertation analyzes and compares three common measurement methods, and take the slow-motion benchmarking as the final choice of credible measurement method, and use this methods to measure and compare the performance of video application in the original and modified VNC system .Comparative analysis of the measured performance show that the changing of the VNC system's display-update

strategy does improves the performance of video application, and provide better user experience.

Key words: Thin-Client, VNC, RFB protocol, multimedia application, slow-motion benchmarking

目录

第一章 绪论

第一节 研究背景

随着网络技术的成熟和 Internet 的飞速发展，人们对于信息和计算资源共享的需求在日益增强。基于网络的计算模式能够为用户提供一致的工作环境，而无论用户处于什么位置，因而其重要性在不断增加，实际应用也在不断增加。在这样的背景下，使用 PC 作为基于网络的计算模式中的客户端已显露出越来越多的弊端，主要体现在以下几方面：

1）安装成本。PC 要求丰富的计算和存储资源，其安装和升级会导致可观的经济成本。同时，PC 自身资源的闲置又是普遍存在的现状。无论对于个人用户还是对于企业用户，这样的成本和浪费都希望能够避免或降低。

2）管理成本。PC 在本地存储用户数据，因而自身具备状态。它们的大量存在意味着难以进行有效的集中式管理，从而导致用户在系统配置和管理上花费大量的时间。

3）安全成本。PC 客户端上病毒、木马、蠕虫的存在，对于网络资源和用户的数字财产安全都是严重的威胁。

出于消除这些弊端的考虑，人们开始渴望一种既能节约成本又便于管理的 PC 的替代品。瘦客户端（Thin-client）技术很好的符合了这样的要求，因此其所受的重视日益增加，也越来越多的投入实际使用。

第二节 瘦客户端系统的基本原理

在瘦客户端系统中所有应用程序的计算和数据存储都在服务器端进行的，客户端负责处理系统与用户之间的输入和输出。

瘦客户端系统的应用示例如图 1.1 所示。服务器会为每个用户生成相应的配置文件并分配计算和存储资源。Client 端负责将用户输入发送给 Server 端，Server 端根据用户输入执行计算并返回显示更新。两者之间的数据交互通过设计好的底层传输协议来进行。

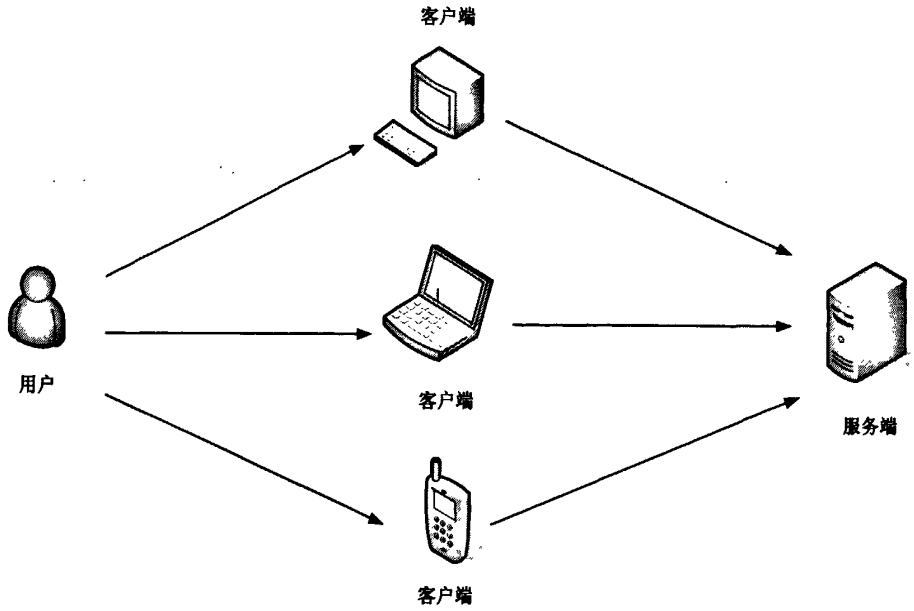

图 1.1 瘦客户端系统应用示例

瘦客户端系统中使用的网络传输协议的核心内容是远程显示机制。具有良好远程显示机制的传输协议能够高效利用网络带宽、减轻客户端运行负载、降低对客户端性能的要求、提供良好的平台独立性、协调机器处理速度和网络带宽之间的平衡。此外，良好的显示机制还能够改善用户界面在客户端的更新速度和效果，提供高质量的实时视频播放等等。

第三节 瘦客户端系统的优势与应用

瘦客户端系统的原理决定了客户端的基本构成就是带有键盘及鼠标的显示终端。与传统 PC 相比，高处理能力的 CPU、大容量的内存和硬盘等部件都可以被去掉。

1.3.1 瘦客户端系统的优势

与传统 PC 相比，瘦客户端具有如下几点优势：

1）管理方便。很多行业业务系统中，用户需要完成的工作内容相对固定，防止用户执行工作外的操作占用了网络管理员很大的精力。在瘦客户端系统中，客户端可以执行的操作受限于服务器端的设置，大大提高了管理力度和效率。

2）成本低廉。由于几乎所有的计算都是在服务器端运行，管理员对整个系

统的安装、调试、管理、维护、升级几乎也都在服务器端进行。这样节省了系统的部署时间，降低了人力资源成，提高了管理效率。同时，由于瘦客户端的硬件组成很简单，提高了客户端的可靠性与稳定性，同时也有效的降低了噪声污染和能源消耗，这些都使得系统的整体成本降低。

3）安全性高。瘦客户端仅仅具有输入输出功能，不具备或仅具备有限的处理能力，因而降低了病毒、蠕虫、木马等造成的危害。另一方面，由于所有数据都集中存放在服务器中，管理员只需要考虑如何提高服务器的安全性，降低了安全成本。

4）易于升级。瘦客户端系统的一个优势是可以方便的对整个系统的硬件和软件进行升级。因为所有的计算都是在服务器中进行，所以当系统中的应用程序对系统硬件提出更高的需求、或者用户对性能提出了更高的要求时，只需要对服务器的硬件设备进行更换，即可实现整个系统的性能提升；当系统中的应用软件需要升级时，管理员也仅需升级服务器端的软件即可，无需在每台机器上做重复操作，节省了时间和人力成本。

1.3.2 瘦客户端系统的应用领域

瘦客户端系统目前主要用于实现基于网络的计算模式。使用瘦客户端系统，用户可以在任何地点使用任何合适的设备连接到服务器，得到一致的工作环境。因而瘦客户端系统在全球性大型企业的内部工作网络中正得到愈来愈多的应用。

瘦客户端系统也可以用于在线教学和培训。教师和学生都使用客户端连接到服务器。这种应用需要瘦客户端系统具有权限控制机制，为教师和学生分配不同的权限，保证只有教师的操作会产生显示更新，并且该显示更新被同步到每个学生的屏幕上。

除此之外，瘦客户端系统还可用于实现远程控制机制。此时客户端充当控方，服务器端充当受控端。瘦客户端在远程控制方面的常见应用包括以下方面：

1）远程技术支持。利用远程控制技术，技术人员可以远程控制用户的电脑，就像直接操作本地电脑一样，只需要用户的简单帮助就可以很快就可以找到问题的所在，并加以解决。

2）远程维护和管理。网络管理员或者普通用户可以通过远程控制技术，在远程主机上完成安装和配置软件、安装补丁程序、调整系统性能、获取进程列

表、关闭或重启计算机等管理操作。

3）高危环境的远程作业和远程监控。

第四节 本文的主要工作

目前瘦客户端系统对于视频播放应用缺乏良好的支持。本文以 VNC 系统为例，分析了其对视频播放缺乏良好支持的原因，并给出了改善视频性能的一种设计修改方案，并对该方案进行了实现。选择 VNC 系统主要是考虑到 VNC 系统是最常见的瘦客户端系统之一，具有良好的可移植性，整体设计简单明了，并且存在开源实现，对现有系统进行修改和优化较为方便。

本文首先介绍了各种瘦客户端系统设计中的一些共有基本问题以及这些基本问题中可选的设计策略，并对包括 VNC 在内的几种常见瘦客户端系统进行了简要介绍和性能对比。

本文继而对 VNC 系统中的基础协议 RFB 协议进行了详细的分析，为后文对 VNC 系统的性能分析和改进提供了必要的知识准备。对 VNC 系统设计的分析表明其对视频播放应用缺乏良好支持的原因在于采用了 Client-pull 模式的显示更新策略。对原有 VNC 系统进行改进的基本思路是将 Client-pull 模式改为 Server-push 模式。本文基于开源项目 Vino 对这一设计更改进行了实现。

本文使用了 slow-motion 基准测量方案对原有和改进后 VNC 系统的视频播放性能进行了测量，并对测量结果进行了比较和分析。

第五节 论文结构

本文共分为六个部分，具体结构如下。

第一章介绍本文的研究背景，介绍了瘦客户端系统的基本概念以及其具备的优势和常见应用领域。

第二章分析了各种瘦客户端系统设计中共有的基本性问题，并对不同设计策略会如何影响系统整体性能做了简要的对比分析。

第三章详细介绍了 VNC 的底层协议 RFB 协议的设计原则和具体规格。

第四章对 VNC 系统在视频播放应用上存在的不足进行了分析，提出了对其进行性能改进的方案，并基于开源项目 Vino 实现了这个改进方案。

第五章对瘦客户端系统性能测量中存在的问题进行了分析，从几种测量方

案中选取了较为理想和可信的 slow-motion 基准测量，并使用该方案对原有和改进后的 VNC 系统中的视频播放进行了性能测量，并对测量结果进行了对比和分析。

第六章对本文的研究进行了简要总结，并对未来的工作进行了展望。

第二章 瘦客户端系统

第一节 基本设计问题

在目前的实际应用中存在着不止一种瘦客户端系统。这些系统在复杂度、性能以及最适用的领域等方面都存在着差异。然而这些系统中存在一些具有共性的基本设计问题，在这些问题上都面临着设计策略的选择。本节就这些基本设计问题进行了分析。

2.1.1 延迟与带宽

尽管网络带宽是影响瘦客户端系统性能的重要因素，然而实验数据[1]表明，除非是在网络带宽较为有限的情况，与网络带宽相比，Server 端和 Client 端之间的传输延迟（Propagation delay）对性能的影响更为显著。这样的实验结果意味着，瘦客户端系统的设计应优先考虑如何减轻网络延迟对系统性能的影响，而不是降低对带宽的需求。随着网络技术的发展，更高带宽的网络将日渐普及化。而物理规律决定了传输延迟存在着下限。因此从长远看，瘦客户端系统的抗延迟特性会成为决定系统整体性能的主导因素。

2.1.2 功能划分

瘦客户端系统中用户的计算环境由两部分组成，一部分是应用程序的运行逻辑或者说应用逻辑，另一部分是负责生成用户界面的窗口系统。

多数瘦客户端系统的首要共同特征是所有应用程序的运行逻辑都在 Server 端执行，而与 Client 端无关。Client 端只负责完成与用户之间的输入和输出，例如捕获用户的键盘和鼠标动作并发送至 Server 端、从 Server 端接收屏幕更新并显示给用户等。

这里一个重要的设计问题就是用户所需要的窗口系统的功能在 Client 端和 Server 端之间如何划分。

多数现代瘦客户端系统如 VNC、RDP 中，除了应用逻辑之外，窗口系统也在 Server 端运行，而 Client 端不具备任何窗口系统的功能。而在早期的瘦客户端系统如 X Window System 中，Client 端也具备窗口系统功能，某些任务是由 Client 端在本地完成的，例如如窗口定位、字体管理等。这意味着在 X Window

System 这样的瘦客户端系统中，Client 端并不是完全的无状态化，需要存储和了解与用户相关的信息，因而 Client 端和 Server 端之间需要更多的状态同步操作。

将窗口系统的功能划分到 Server 端具备三个优点：

1）对 Client 端计算资源的需求最小化。Client 端只需具备捕获键盘鼠标等设备上的用户动作以及显示图像的能力，从而降低了 Client 端在安装、配置、管理和升级等方面的成本。

2）为原有应用程序提供了无缝兼容性。传统桌面系统中的应用程序不需要进行任何修改，就可以在瘦客户端系统的 Server 端正常运行。

3）减少了 Client 端和 Server 端之间的同步操作。实验数据[2]表明，令 Client 端不具备窗口系统功能从而将 Client 端和 Server 端之间的同步操作最小化，将产生更好的整体性能，尤其是在 WAN 这样的传输延迟较高的环境中。

2.1.3 显示原语

不同的瘦客户端系统使用不同的显示原语来对显示更新进行编码。按照抽象层次的高低，常见的显示原语[3]可分为高阶绘图（high-level graphics）、低阶绘图（low-level graphics）、2D 绘图原语（2D draw primitives）和 Raw 象素这四种。

高层次的显式原语通常在减少数据传输量和有效利用带宽上更具有效率，但是另一方面它要求 Client 端能够执行更复杂的计算。例如，如果使用高层次的显示原语对字体信息进行编码，则要求 Client 端对文本和图像区分处理，而且还需要在 Client 端本地存储字体相关数据以正确解释收到的字体信息。

然而实验数据[4]表明，与低层次的显示原语相比，高层次的显示原语并不一定表现出更高效的带宽利用。实际上，高层次显示原语最适用于文本密集型应用，而不适用于图像和视频等类型的应用。随着应用程序多媒体化趋势的加强和网络带宽的增加，显示原语能否高效的支持图像和视频应用将变得更为重要，高层次的显示原语具备的带宽优势和其导致 Client 复杂度的增加相比，将变得弊大于利，因此优先使用低层次的显示原语将会是未来的趋势。

2.1.4 数据压缩

多数的瘦客户端系统都会采用低层次的压缩技术来降低 Server 端向 Client 端传输的数据量，以提高系统的整体性能。选择压缩算法的基本问题是在算法复杂度和数据压缩率之间进行平衡。过于复杂的高效压缩算法意味着 Client 端需

要执行对计算能力要求很高的解压缩操作，增大了对 Client 端性能的要求，不利于实现瘦客户端在广泛类型设备上的可用性。

2.1.5 更新策略

更新策略是指 Server 端在显示更新的生成和何时发送显示更新上采取的策略。何时发送显示更新具有和发送哪些显示更新同等重要的意义，这一点往往容易被忽视。

更新策略的选择涉及到两个方面的基本问题：

1．显示更新的生成

这个问题的实质是显示更新与窗口系统的图形命令之间的对应关系。Server 端可以采取积极更新策略（Eager display update），也可以采取懒惰更新策略（Lazy display update）。在积极更新的策略中，一旦窗口系统发出新的图形命令，Server 端都立刻生成该命令对应的显示更新。在懒惰更新的情况下，窗口系统产生的图形命令会被首先放入一个中间队列，以检测各条命令所更改的区域是否存在重合，在存在重合的情况下将对相关命令进行合并后为其生成显示更新。

懒惰更新由于可以对多个显示更新进行合并，在带宽利用上更有优势。然而懒惰更新并不适用于对交互性要求较高的应用。

2．显示更新的驱动模式

存在两种可选的显示更新的驱动模式，一种是 Server-push，另一种是 Client-pull。在 Server-push 模式中，由 Server 端决定何时向 Client 端发送显示更新，显示更新是由运行在 Server 端的应用逻辑驱动的。而在 Client-pull 模式中，Client 端在需要显示更新时向 Server 端发送请求，驱动着 Server 端返回显示更新。

Client-pull 模式的优势在于它是一种简单且易于实现的模式，并且整个系统具备一定的自适应能力，Client 端能够根据自身处理能力、网络状况对发送请求的频率进行调整。

Client-pull 模式的问题在于相邻的显示更新之间存在不可消除的时间间隔。Server 端在发出显示更新后，需要等到再次收到 Client 端的请求，才能发出下一次的显示更新，因此两条显示更新之间的时间间隔至少是 Client 端与 Server 端之间的 RTT（Round-Trip Time）。这意味着 Client-pull 模式在传输延迟较大的情况下，对于视频播放这样要求显示更新频繁的应用，将导致用户获得较差的使用体验。例如，当 Client 端与 Server 端之间的 RTT 超过 66ms 时，无论 Client

端的处理能力如何，都无法达到24fps的视频播放效果。

随着应用程序多媒体化趋势的加强和网络带宽的增加，采用积极更新和Server-push模式所具备的用户体验优势，将压倒采用懒惰更新和Client-pull模式所带来的带宽利用优势。

第二节 常见瘦客户端系统

2.2.1 X Window System

X Window System[5]是由麻省理工学院（MIT）开发的一套适用于广泛类型显示器、具有良好移植性的视窗系统。

X Window System并不是一个单一的软件，而是由一组软件协同工作的一个系统。X Server和X Client基于X协议和标准套接字通过网络通信协同完成应用程序的图形显示任务。其中，X Client发出图形显示命令，X Server则负责生成该命令对应的像素数据的并进行显示。

X Window System的并不是专门针对瘦客户端应用而设计的，但由于X协议的设计中充分考虑了应用程序的运行逻辑和界面显示实现之间的可分离性，使得基于X Window System的应用程序自然的具有网络透明性，即本地应用和远程在用户看来不存在区别，因此X Window System成为了最早的图形终端应用协议之一。与更晚出现的其它瘦客户端系统相比，X Window System的显著特点是在负责图像生成和显示的X Server是运行在其Client端的，对Client端的资源有较高的要求。

2.2.2 VNC

VNC[6]（Virtual Network Computing）是由AT&T开发的虚拟网络协议。由VNC Server和VNC Client组成，其中VNC Server产生图形输出，VNC Client将这个图形输出显示到本地的屏幕上。VNC系统的示意图如图2.1所示。

VNC系统中Server端只是简单地将屏幕中特定矩形区域内的像素数据提取出来，放入帧缓冲区后发送给Client端，然后再Client端负责在正确位置进行显示。通常在VNC系统中只传输屏幕显示中发生了变化的那部分信息。

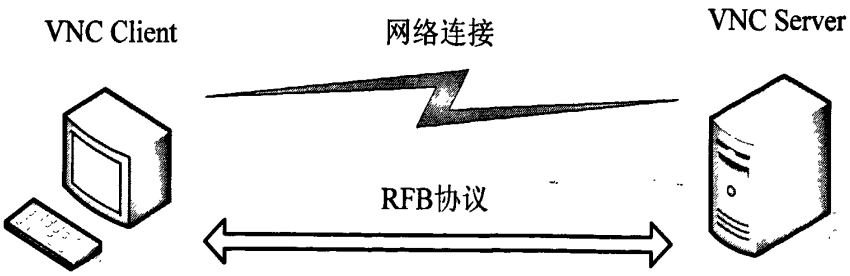

图 2.1 VNC 系统的示意图

VNC 系统的设计以 RFB[7]（Remote Frame Buffering）协议为基础，即通过 Server 端来更新 Client 端的帧缓冲区，来实现远程访问图形用户界面的目的。由于该协议仅仅考虑如何将图像传送到远程进行显示，把其中大量工作留给服务器端，而只要负责显示的客户端支持很少几条命令，因此有效地减轻了客户端的负担。与 X System 和 RDP 系统相比，VNC 要简单得多，但是由于传送的是简单的像素数据信息，传送的数据量增大，加大了网络传输的负担。

在 VNC 系统中，客户端为无状态化的极瘦客户端，对计算资源的需求很低。服务端通过 RFB 协议接收客户端发来的键盘、鼠标等输入动作，转发给相应的应用程序，并将产生的帧更新发送给 Client 端。

Unix/Linux 环境下的 VNC 系统通常以 X 协议作为图形显示协议，即在 VNC Server 端运行 X Server。X Server 作为 Server 端中应用程序和 RFB 协议之间的连接点而存在。应用程序作为 X Client，其发出的绘图命令被 X Server 转换为对应显示的像素数据后，由 RFB 协议负责将像素数据发送至 VNC Client。另一方面，在收到 VNC Client 发来的用户输入时， VNC Server 将用户输入转发给 X Server，由 X Server 负责将其转发给合适的应用程序。

VNC 系统具有一定的自适应能力，Server 端发往 Client 端的显示更新是由 Client 端的需求所驱动的，Client 的处理能力和网络环境越差，更新的频率就越低。

2.2.3 RDP

RDP[8]（Remote Desktop Protocol）是 Microsoft 公司在国际电信联盟（ITU）的 T.120 协议族[9][10][11]基础上的一个扩充，目的在于提供终端客户与服务器之间的连接。RDP 的主要特性在于运行 Microsoft 系列办公软件和 IE 网页浏览器时效率很高。实际上 RDP 的设计目的就是为了提供更好的 Windows 终端服务。

RDP 主要是通过 RDP 提供的虚拟通道实现的，其下层使用的是 TCP/IP 协

议和网络的物理连接（LAN/WAN）。RDP 协议只在服务器和客户端之间传送键盘，鼠标和屏幕的更新数据，以此来实现 Windows 下的多用户模式和终端访问功能。RDP 协议是支持 Unicode，支持网络定位，自动断开连接和远程配置等功能[12]。RDP 协议对于不同类型的信息的处理机制是不同的[13]。视频描述信息在 Client 端和 Server 端的显示驱动中进行转换，而来自客户端的键盘鼠标信息则是通过驱动 RDP 协议本身的虚拟鼠标和键盘驱动器进行接受的。

第三节 设计策略对系统性能的影响

下面将以 VNC、RDP 和 X Window 三个瘦客户端系统为例，从显示原语、更新策略、压缩算法和客户端缓存这四个方面，分析和比较它们设计中采取的不同设计策略对系统整体性能的影响。

2.3.1 显示原语对系统性能的影响

在 3 种协议中出现了 4 种不同类型的显示编码原语，按照抽象层次从高到低的顺序分别为高级绘图（high-level graphics）、低级绘图（low-level graphics）、2D 绘图原语（2D draw primitives）和 Raw 象素。

X 协议中采用高级绘图原语，并支持大量的绘图原语。RDP 则建立在低级绘图原语上，通过发送上层的绘图命令来实现远程显示的目的，这些命令中包含了对字体、图标、绘图命令等的支持。VNC 采用 2D 绘图原语，例如用像素填充矩形区域；也可以采用 Raw 象素编码，但在默认情况下不会采用这种编码方式。

有数据表明[14]，在 100Mbps 网络环境下关闭各个协议中的缓存和压缩功能（VNC 无法关闭压缩功能），单独测量协议的编码原语，发现无论是单纯传输文本内容的网页，还是传输图像和文本混合的网页，X 均具有最低平均延迟，其次是采用 2D 绘图原语的 VNC，而 Raw 编码方式的延迟时间最长。如果传输相同内容的图像文本混合网页，VNC 传输的数据量最少。如果是传输相同内容的纯文本网页，则 RDP 传输的数据量最少，VNC 则稍高些。无论是传输混合类的网页，还是纯文本的网页，Raw 编码方式传输的数据量都是最多的。

从上面的结果中可以看出：减少传输内容的数据量方面，高层次的显示原语具有的优势。

2.3.2 更新策略对系统性能的影响

X Window System 在 Server 端采用了积极更新策略。当应用程序产生图形显示命令时，瘦客户系统立即为命令生成对应的显示更新，并将显示更新放入帧缓冲区准备传输。在应用程序需要高频率的传输大量显示更新的情况下（例如视频播放），积极更新策略能保证 Server 端满足应用程序的需要。

RDP 和 VNC 这两个系统在 Server 端均采用了懒惰更新策略。在这种情况下，多条画图命令首先被缓存，然后合并，最后再将合并后的显示更新发送给客户端。在 RDP 系统中，显示更新是由 Server 端以确定时间间隔发送的；而在 VNC 系统中，显示更新则是在 Client 端显式的发出请求的情况下发送的。

数据[3]表明，在传输大量连续图像数据的情况下，X 具有最好的视频传输质量。RDP 由于采用懒惰更新策略，它的更新速度难以满足实时视频显示的要求，导致无法将显示更新及时发送给 Client 端进行，因而传输视频的质量不高。一方面，在 VNC 系统中因为显示更新请求的发送是由 Client 端驱动，Client 端必须频繁的发送更新请求，这增大了 Client 端的负担，可能会成为系统的性能瓶颈；另一方面，在客户端发出下一条更新请求之前，Server 端最初的显示更新可能已被更新的显示更新所覆盖了，因此 VNC 在视频传输方面的效率和质量是最差的。

2.3.3 压缩算法对系统性能的影响

X 没有采用压缩算法。RDP 采用了的 RLE（run-length encoding）压缩。RLE 是一种不需要大量处理器时间进行解码的简单压缩算法，因此可以有效减轻给客户端带来的负担。RLE 的基本思想可以简述为：设法将原图像区域分成多个子矩形区域，而每个子区域由一个基本象素点和该子区域的区域描述所组成。

VNC 采用了 Hextile（2D RLE）压缩和 Zlib 压缩。Hextile 算法是一种二维的 RLE 编码，它将整个显示窗口划分为 16×16 个子块，再针对每个子块划分子矩形区域，但每一子块划分的大小和位置并不需要明确定义；Zlib 压缩是一种压缩率较高、并且可以调整图像压缩级别和压缩质量的压缩算法。

有实验数据表明[15]，RDP 在压缩纯文本数据和图像数据时，可以将数据量压缩到原来的 40%；在压缩视频数据时，压缩比可以达到 58%。VNC 由于默认采用了压缩算法，所以在压缩纯文本数据时，压缩后的数据量可以压缩到原数据量的 3%；而在压缩图像数据和压缩视频数据时，这个比例分别为 6%和 30%。可以看到如果和原始数据量比较，VNC 的压缩算法具有较高的压缩比。

2.3.4 客户端缓存对系统性能的影响

X 中由应用程序决定是否在 Client 使用缓存，一般情况下是不使用的。RDP 将字体和小位图的数据保存在 Client 端的本地缓存中。VNC 仅使用帧缓冲区机制，即总是简单的从 Server 端获取屏幕区域对应的像素数据，存放在本地帧缓冲区中。如果 Client 端所需要的像素数据已经在其本地帧缓冲区中存在，则无需向服务器请求显示更新，只需对本地帧缓冲区的数据执行复制操作。

对 VNC 来说，由于采用的帧缓冲区机制仅保留当前的显示数据，对于整个传输过程中的历史信息不敏感，所以无法充分利用历史信息优化都显示更新的要求。但是如果是在屏幕中移动窗口或滚动窗口中的内容时，VNC 具有一定的优势。RDP 由于本身设计上存在的漏洞，其缓存机制并没有发挥作用[16]。

第四节 本章小结

本章分析了各种瘦客户端系统共有的基本设计问题，并介绍了各种问题中的不同设计选择。在此基础上对不同瘦客户端系统的整体性能进行了对比。从对比结果中，可以得出以下结论：

1）在网络带宽良好的情况下，编码方式是决定整个系统性能的主要因素。采用高层次绘图编码方式具有较好的网络带宽利用率，但是对 Client 端的复杂度要求更高，平台独立性差。比较而言，采用低层次的绘图编码方式能够保持较低 Client 端的复杂性，具备较好的平台独立性。

2）显示更新策略是对要求连续显示更新的应用，特别是视频播放，具有非常重要的因素。采用积极显示更新策略具有非常好的显示效果，但是懒惰显示更新策略可以节省网络带宽。

3）在网络带宽低的情况下，采用好的压缩算法可以极大提高系统性能，减少数据量的传输。

4）采用客户端缓存机制是提高远程显示性能重要原因之一。客户端缓存可以用来保存经常使用的显示元素，如字体和图标位图，这样客户端就可以从本地缓存中获得需要的显示元素，而不是重复从服务器端请求获得。

5）使用合并机制可以有效地降低传输的数据量，减轻对网络带宽和 Client 端处理能力的压力。

第三章 RFB 协议分析

RFB 协议是 VNC 系统的基石。不论是对 VNC 系统进行分析还是进行修改，都需要首先了解 RFB 协议的原理和特性。本章介绍了 RFB 协议的基本性质，并对其协议规格进行了详细分析。

第一节 RFB 协议简介

VNC 系统使用 RFB 协议作为底层的数据传输和交互协议。在 RFB 协议中，远程用户所使用的端点（即显示终端加上键盘鼠标）被称为 RFB Client，而生成显示更新的端点则被称为 RFB Server，如图 3.1 所示。

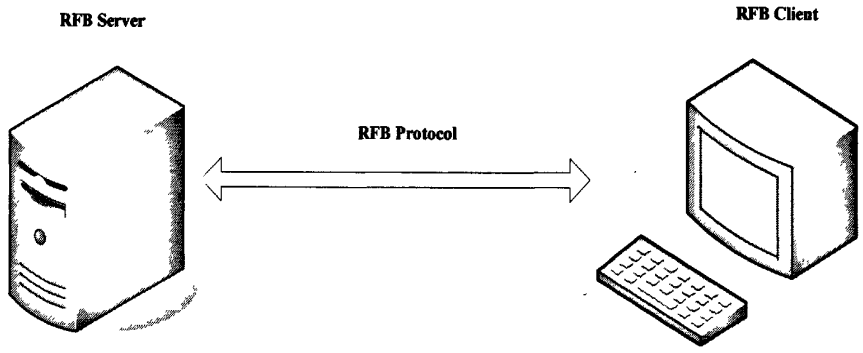

图 3.1 RFB 协议示意图

RFB 协议从功能上看不能算是一个完整的终端图形显示协议。RFB 协议并不涉及到应用程序的运行逻辑与图形显示的分离过程，而只负责直接从帧缓冲区中获得应用程序的显示更新，以及 Client 端和 Server 端之间的数据交互。至于如何将图形显示命令转换成帧缓冲区中的像素数据，则由类似于 X Window 这样的视窗系统负责完成。与 X 协议这样的复杂协议相比，RFB 协议要简单得多。VNC 系统中 Client 端、Server 端、RFB 协议以及视窗系统的层次结构关系如图 3.2 所示。

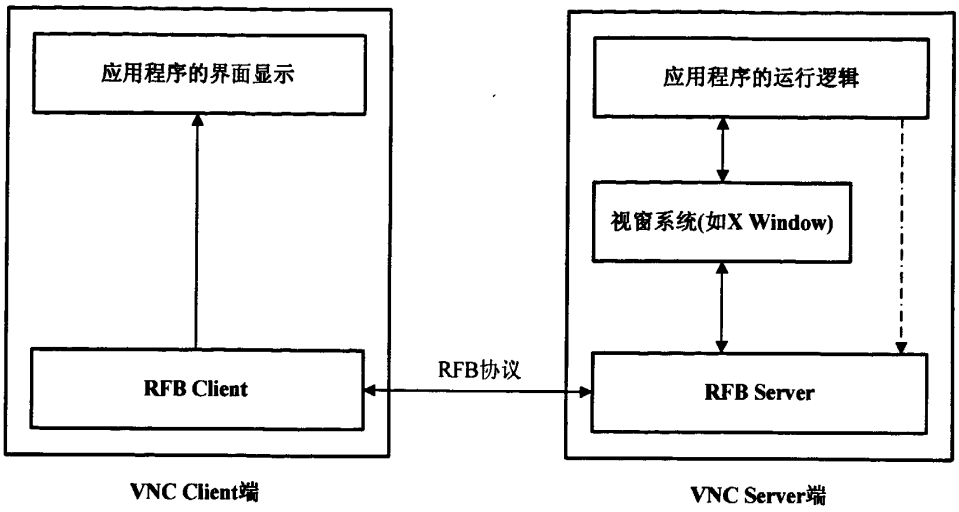

图 3.2 VNC 系统的整体层次结构

RFB 协议是名副其实的瘦客户端协议。RFB 协议的设计原则和目标是保持 RFB Client 尽可能简单，这样 RFB Client 就可以在类型更广泛的、具备不同处理能力的硬件设备上运行，并且要实现 RFB Client 也变得更加容易。RFB 协议工作在帧缓冲区这一抽象层，在所有主流系统平台上都适用，包括 Linux、Windows 和 Macintosh。

RFB 协议中的 Client 端是无状态化的。如果用户从 Server 端断开连接后重新连接到同一 Server 端，那么用户界面仍保持在原来的状态。而且，用户可以通过不同的 Client 端连接到同一 Server 端；在每个连接中，用户都会看到相同的图形用户界面。这样用户的工作环境就变得可移动化，无论用户处于什么位置，只要存在合适的网络连接，就能够访问 Server 端上的个人应用程序，并且应用程序的状态在不同的访问中被保持，不会因为用户断开连接应用程序的工作就丢失了。

第二节 图像显示

RFB 协议中的图形显示基于一个简单的图形原语："以平面坐标（x，y）指定的位置为基点，画一个由指定像素数据填充的矩形"。在绘制大量用户界面组件时，这似乎是一种很没有效率的方法。然而由于允许对像素数据采取各种不同编码方式带来了的灵活性，RFB 协议能够针对各种参数如网络带宽、Client 端的显示性能和 Server 端的计算能力，进行均衡和调整[17]。

帧缓冲区的一次更新由若干这样的矩形区域构成。帧缓冲区的更新意味着帧缓冲区由一个合法状态转换到另一个合法状态，某种程度上类似于视频的一帧。通常构成帧缓冲区一次更新中的矩形区域都是互相邻接的，但这并不总是成立的。

RFB 协议采用了由 Client 端的需求驱动的更新策略，Server 端将帧更新作为对 Client 端发出的请求的回应。这样 RFB 协议就具备了一定程度的自调节性。Client 端的处理能力和网络速度越慢，那么帧更新的频率也就越低。

针对相同区域的帧缓存区更新的连续生成，这是相当典型的情况[18]。在 Client 端性能和网络带宽有限的情况下，帧缓冲区的中间状态可能会被丢弃，这样能减少数据传输量，减轻 Client 端的压力。

第三节 用户输入

RFB 协议中的用户输入基于标准的工作站模型，即键盘加鼠标。用户在 Client 端触发的键盘或鼠标动作被简单的转发到 Server 端，Server 端在将这些操作转发给对应的应用逻辑，由应用逻辑完成用户操作引发的命令。其他非标准的输入设备如手写板上的用户输入，可以被转换为等效的标准键盘、鼠标动作后发送到 Server 端。

第四节 像素数据的表示方式

RFB 的 Client 端和 Server 端在建立连接的初始化过程中，双方对将要传送的像素数据的格式和编码方式进行协商。这个协商过程存在的目的为了使得 Client 的工作尽可能简单。协商过程的底线是 Server 端必须总能按 Client 要求的格式和编码提供像素数据。然而，如果 Client 端支持多种格式或编码，那么 Server 端有权选择其中最合适的一种。

像素格式指的是像素值中各种色度的表示方法。最常见的像素格式是 24-bit 或 16-bit 的“真彩色”，像素值中的位域是和红、绿、蓝三种颜色的亮度直接对应的。

编码指的是矩形区域中的像素数据以何种形式被传输。每个矩形区域对应的像素数据前被加上一个头部，记录着这个矩形区域的基准位置在屏幕上的平面坐标、矩形区域的宽度和高度、以及矩形区域内像素数据所使用的编码种类。

这个头部后面的数据就是使用前面指明编码的像素数据。当前定义的编码种类包括 Raw，CopyRect，RRE，Hextile 和 ZRLE。实际应用中通常只使用 ZRLE，Hextile 和 CopyRcet 这三种编码方式，因为它们在常见桌面系统中提供最好的压缩性能。

第五节 协议扩展性

RFB 协议在设计上为以下方面的提供了扩展空间：

1）增加新的编码方式。在保证对已有 Client 和 Server 提供兼容性的前提下，为 RFB 协议增加一种新的编码方式是相当容易的。已有的 Server 对于 Client 提出的对新增编码的要求，只需要简单的忽略即可。已有的 Client 则永远不发出对新增编码的请求，这样也就永远不会从 Server 端收到使用新增编码方式的像素数据。

2）使用伪编码。Client 端还可向 Server 端发出对伪编码的请求，以表明该 Client 端支持 RFB 协议的某种扩展。如果 Server 端不支持此种扩展，将简单的忽略该请求。这意味着 Client 端在从 Server 端得到支持某个扩展相关的确定信息前，必须假设 Server 端不支持该扩展。

3）增加新的安全类型。通过增加新的安全类型，可在同既有 Client 端和 Server 端保持兼容性的前提下，灵活的调整协议的安全性。

第六节 协议消息

RFB 协议通常是基于 TCP/IP 实现的。RFB 协议中 Client 端和 Server 端的交互可分为三个阶段。第一阶段是握手阶段，在这个阶段双方对使用的协议版本和安全类型达成一致。第二阶段是初始化阶段，在这个阶段 Client 端和 Server 端交换 ClientInit 和 ServerInit 消息，来完成双方的初始化工作。第三阶段的是 Client 端和 Server 端之间正常的数据交互。

Client 端可以发送任何想要发送的消息，并且会从 Server 端收到相应的回应消息。所有消息的第一个 byte 中都记录着该消息的类型，该字节后面跟随的则是同特定消息类型相关的数据。

本章下文中在协议消息的描述中使用了基本类型 U8，U16，U32，S8，S16，S32 这些符号，它们分别表示 8 位、16 位和 32 位的无符号数和 8 位、16 位和

32 的有符号数。所有多字节整数使用的字节序都是大端。下文中的 PIXEL 结构表示一个像素值，其大小 bytesPerPixel，单位为字节；这里 bytesPerPixel 等于 Client 和 Server 端协商一致的单个像素值对应的字节数。

3.6.1 握手阶段

1．协议版本

握手过程是由 Server 端向 Client 端发送 ProtocolVersion 消息开始的。Client 端通过该消息获知 Server 端所支持的最高 RFB 版本号。Client 端随后向 Server 端回应一条类似的消息，通知其在后续过程中将要使用的 RFB 版本号。客户端应该从不发出对高于 Server 端所支持的最高 RFB 版本的要求。该机制的目的是使得 Client 端和 Server 端都可以提供某种程度的向后兼容性。当前公开的 RFB 协议版本包括 3.3，3.7 和 3.8。增添新的编码和伪编码并不需要对版本号进行修改，因为 Server 端可以简单的忽略其所不支持的编码。

ProtocolVersion 消息的格式如表 3.1 所示：消息的长度为 12 byte 组成，其内容为“RFB xxx.yyy\n”这样的 ACSII 字符串，其中“xxx”和“yyy”是分别主版本号和次版本号，并都以‘0’作为补齐。

表 3.1 ProtocolVersion 消息

字节数	值
12	"RFB 003.003\n" (hex 52 46 42 20 30 30 33 2e 30 30 33 0a)

2．安全协商

在就版本号达成一致后，Server 端和 Client 端还必须对连接所使用的安全类型达成一致。安全类型的协商过程由 Server 端发起。Server 端向 Client 端发送格式如表 3.2 所示的消息，在该消息中列出其所支持的所有安全类型。

表 3.2 安全类型消息#1

字节数	类型	描述
1	U8	number-of-security-types
number-of-security-types	U8 数组	security-types

如果 Client 端支持其中所列的任何一种安全类型，就向 Server 端回应一个

消息，表明后续连接将要使用的安全类型。消息的格式如表 3.3 所示。

表 3.3 安全类型消息#2

字节数	类型	描述
1	U8	security-type

如果表 3.2 中 number-of-security-types 该项的值为 0，这表示 Server 端由于某种原因导致无法建立连接 （例如，Server 端无法支持 Client 所要求的 RFB 版本)；这种情况下 Server 端会在这条消息后附加一条包含失败原因字符串的消息，其格式如表 3.4 所示。之后 Server 端将关闭此次连接。

表 3.4 安全类型消息#3

字节数	类型	描述
4	U32	reason-length
reason-length	U8 数组	reason-string

3．安全协商结果

Server 端会向 Client 端发送一条消息，通知其安全类型的协商过程是否成功。消息格式如表 3.5 所示。如果协商成功的话，RFB 协议就进入到第二阶段即初始化阶段。

表 3.5 安全类型消息#4

字节数	类型	值	描述
4	U32		结果
		1	成功
		0	失败

3.6.2 安全类型

RFB 协议中使用的基本安全类型包括如下两种：

1）不使用任何安全机制。连接的建立过程中不使用认证机制，并且在连接建立后传输的数据是未加密的。

2）启用 VNC 认证机制。连接的建立过程中使用 VNC 认证机制，连接建立

后传输的协议数据是未加密的。

VNC 认证机制的工作流程分为以下三步：

1）Server 端发送一个 16-byte 的随机数。

2）Client 端收到该随机数后，将用户提供的密码作为 key 值，使用 DES 加密算法对收到的随机数进行加密。

3）将 16-byte 的加密结果返回给 Server 端。

3.6.3 初始化阶段

在 Client 端和 Server 端对使用的安全类型达成一致后，RFB 协议进入第二阶段即初始化阶段。初始化过程中首先 Client 端发送 ClientInit 消息，Server 端在收到该消息后发送 ServerInit 消息作为回应。

1．ClientInit 消息

ClientInit 消息的格式如表 3.6 所示。

表 3.6 ClientInit 消息

字节数	类型	描述
1	U8	shared-flag

表 3.6 中的 shared-flag 为非 0 时，表示当前 Client 端允许其它 Client 端共享该 Client 端使用的用户界面；shared-flag 为 0 时，则表明该 Client 端所使用的用户界面是独占、排它性的，不允许其它 Client 端与其共享。

2．ServerInit 消息

在收到 ClientInit 消息后，Server 端回应以 ServerInit 消息。该消息中包含 Server 端帧缓冲区对应的宽度和高度、像素格式以及和用户界面相关联的名字。ServerInit 消息的格式如表 3.7 所示。

表 3.7 ServerInit 消息

字节数	类型	描述
2	U16	framebuffer-width
2	U16	framebuffer-height
16	PIXEL_FORMAT	server-pixel-format
4	U32	name-length
name-length	U8 数组	array name-string

ServerInit 消息中的中 PIXEL_FORMAT 结构的格式如表 3.8 所示。

表 3.8 PIXEL_FORMAT 结构

字节数	类型	描述
1	U8	bits-per-pixel
1	U8	depth
1	U8	big-endian-flag
1	U8	true-colour-flag
2	U16	red-max
2	U16	green-max
2	U16	blue-max
1	U8	red-shift
1	U8	green-shift
1	U8	blue-shift
3		padding

Server-pixel-format 所表示的是 Server 端的正常像素格式。该格式将作为默认值被使用，除非 Client 发送 SetPixelFormat 消息显式的要求 Server 端使用另外的像素格式。

bits-per-pixel 表示单个像素值所对应的 bit 数。该值必须不小于 depth 的值，即单个像素值中有效 bit 的个数。目前所支持的 bits-per-pixel 的值必须是 8，16 或 32。big-endian-flag 的值若为非 0，则意味着多字节的像素将按大端字节序进行解释。

如果 true-colour-flag 的值为非 0，则表 3.8 中最后 6 项是用来告知 Client 端如何从单个像素值中提取出红、绿、蓝三种色度。例如，red-max 代表红色色度的最大值，而 red-shift 则是从单个像素值中提取红色色度需要执行向右位移的 bit 数。

如果 true-colour-flag 的值为 0，则意味着 Server 端发送像素值并非是红、绿、蓝三种色度的组合，而应该被解释为色彩映射表的索引值。色彩映射表的条目是由 Server 端通过 SetColourMapEntries 消息来通知 Client 端的。

3.6.4 Client 发往 Server 的消息

Client 端发往 Server 端的主要消息如表 3.9 所示。

表 3.9 Client 至 Server 的主要消息类型

编号	消息名字
0	SetPixelFormat
2	SetEncodings
3	FramebufferUpdateRequest
4	KeyEvent
5	PointerEvent
6	ClientCutText

1．SetPixelFormat 消息

SetPixelFormat 消息的格式如表 3.10 所示。

表 3.10 SetPixelFormat 消息

字节数	类型	值	描述
1	U8	0	message-type
3			padding
16	PIXEL_FORMAT		pixel-format

该消息的作用是设定由 Server 端发往 Client 端的 FramebufferUpdate 消息中包含的像素值所使用的格式。如果 Client 端没有显式的发送 SetPixelFormat 消息，那么 Server 端将默认使用之间握手阶段发送的 ServerInit 消息中声明的正常像素格式。

如果 PIXEL_FORMAT（详见表 3.8）中的 true-colour-flag 为 0，则意味着将要使用色彩映射表。Server 端可以通过发送 SetColourMapEntries 消息，设定该色彩映射表中的任意条目。

2．SetEncodings 消息

SetEncodings 消息的格式如表 3.11 所示。

表 3.11 SetEncodings 消息

字节数	类型	值	描述
1	U8	2	message-type
1			padding
2	U16		number-of-encodings
4 × number-of-encodings	S32 数组		encoding-types

该消息的作用是设定 Server 端在传输像素数据时使用的编码类型。消息中列出的编码类型的顺序应被被 Server 端理解为优先级顺序。然而这种优先级并不具有强制性。即使 Client 端显式的在该消息中指定了其它编码类型，Server 端总是可以选择采用 Raw 编码来传输数据。

3．FramebufferUpdateRequest 消息

FramebufferUpdateRequest 消息的格式如表 3.12 所示。

表 3.12 FramebufferUpdateRequest 消息

字节数	类型	值	描述
1	U8	3	message-type
1	U8		incremental
2	U16		x-position
2	U16		y-position
2	U16		width
2	U16		height

该消息的作用是告知 Server 端：Client 端对帧缓冲区中由 x 坐标、y 坐标、宽度和高度所制定的区域感兴趣。通常情况下，Server 端会以发送 FramebufferUpdate 消息作为回应。然而需要注意的是，一条 FramebufferUpdate 消息可能对应若干条 FramebufferUpdateRequest 消息。

Server 端通常假设 Client 端在其本地帧缓冲区中拥有其感兴趣的矩形区域的完整副本，这意味着 Server 端只需要发送增量更新部分。

然而，如果由于某种原因 Client 端丢失了其所需要的某个区域的原有数据，那么在发送 FramebufferUpdateRequest 消息时，需要将消息中的 incremental 项设

为 0 值。这样 Server 端就会尽可能快的发送帧缓冲区中指定区域的完整内容。

如果 Client 端没有丢失其所需要的某个区域的原有数据，在发送 FramebufferUpdateRequest 消息时，会将 incremental 项设定为非 0 值。这样，在帧缓冲区中该区域的内容发生变化时，Server 端会发送 FramebufferUpdate 消息。这里需要注意的是，Server 端在收到 FramebufferUpdateRequest 消息与发送 FramebufferUpdate 消息之间的时间间隔是不确定的。

4．键盘动作消息

该消息的格式如表 3.13 所示。

表 3.13 键盘动作消息

字节数	类型	值	描述
1	U8	4	message-type
1	U8		down-flag
2			padding
4	U32		key

该消息用于表示 Client 端的键盘上键的按下或释放。消息中 down-flag 的值为非 0，则表示键的按下；值为 0，则表示键的释放。该消息中 key 的值对应 X Window System 中定义的“keysym” 值。大多数常用键在“keysym”和 ASCII 中的编码是一样的。

5．鼠标动作消息

该消息的格式如表 3.14 所示。

表 3.14 鼠标动作消息

字节数	类型	值	描述
1	U8	5	message-type
1	U8		button-mask
2	U16		x-position
2	U16		y-position

该消息用于表示光标的移动以及鼠标按钮的按下、释放。光标的位置由（x-position, y-position）指明，而 button-mask 项则表示鼠标按钮的当前状态。

6．Client 端的文本剪切消息

当用户在 Client 端执行文本剪切操作时，Client 端向 Server 端发送该消息，消息中包含剪切操作所涉及的字符串。该消息的格式如表 3.15 所示。

表 3.15 Client 端的文本剪切消息

字节数	类型	值	描述
1	U8	6	message-type
3			padding
4	U32		length
length	U8 数组		text

Client 端会在其本地剪切缓存区中存放 ISO 8859-1（Latin-1）文本。目前 RFB 协议还不支持传输 Latin-1 字符集之外的文本信息。

3.6.5 Server 发往 Client 的消息

Server 端发往 Client 端的主要消息类型如表 3.16 所示。

表 3.16 Server 至 Client 的主要消息类型

编号	消息名字
0	FramebufferUpdate
1	SetColourMapEntries
3	ServerCutText

1．FramebufferUpdate 消息

FramebufferUpdate 消息的格式如表 3.17 所示。

表 3.17 FramebufferUpdate 消息

字节数	类型	值	描述
1	U8		message-type
1			padding
2	U16		number-of-rectangles

该消息后跟数量为 number-of-rectangles 的矩形区域像素数据，其格式如表

3.18 所示。

表 3.18 矩形区域像素数据的结构

字节数	类型	值	描述
2	U16		x-position
2	U16		y-position
2	U16		width
2	U16		height
4	S32		encoding-type

Server 端通过向 Client 端发送该消息对其端帧缓冲区进行更新。帧缓冲区的更新会涉及包含若干个矩形区域的像素数据。Client 端在接受到该更新消息后会将这些像素数据复制到本地的帧缓冲区中。该消息是 Server 端对收到的 Client 端发送的 FramebufferUpdateRequest 消息的回应。

2. SetColourMapEntries 消息

如果双方所协商的像素格式使用色彩映射表，那么该消息用于告知 Client 端如何正确将收到的单个像素值作为数组索引，得到用于显示的 RGB 色度。

该消息由两部分组成：首先是如表 3.19 所示的消息头部，其次是由数量为 number-of-colours 条表 3.20 所示的映射表项所构成的色彩映射表。

表 3.19 SetColourMapEntries 消息头部

字节数	类型	值	描述
1	U8	1	message-type
1			padding
2	U16		first-colour
2	U16		number-of-colours

表 3.20 色彩映射表表项

字节数	类型	值	描述
2	U16		red
2	U16		green
2	U16		blue

3．Server 端的文本剪切消息

当用户在 Server 端执行文本剪切操作时，Server 端会向 Client 端发送该消息，消息中包含剪切操作涉及的字符串。该消息的格式如表 3.21 所示。

表 3.21 Server 端的文本剪切消息

字节数	类型	值	描述
1	U8	3	message-type
3			padding
4	U32		first-colour
length	U8 数组		number-of-colours

Server 端会在其本地剪切缓存区中存放 ISO 8859-1（Latin-1）文本。目前 RFB 协议还不支持传输 Latin-1 字符集之外的文本信息。

3.6.6 编码类型

RFB 协议的编码部分是整个协议的核心,它直接决定了在网络环境下图像更新和应用程序的响应速度。

RFB 协议目前普遍使用的编码类型如表 3.22 所示。

表 3.22 常用编码类型

编号	名称
0	Raw
1	CopyRect
2	RRE
5	Hextile
16	ZRLE

以上几种编码方法中，ZRLE 的压缩率最高,但是其解码所需的时间要高于其它所有的编码方法,而 Hextile 编码方法所需的解码时间与 RRE 差不多，压缩率虽高于这两者，但是大大低于 ZRLE。在实际应用中,一般只采用 Raw、Hextile 和 ZRLE 这三种编码方法[19]。

1．Raw 编码

Raw 编码的格式如表 3.23 所示。

表 3.23 Raw 编码

字节数	类型	描述
Width× height×bytesPerPixel	PIXEL 数组	Pixels

Raw 是最简单的编码类型。在这种情况下，每个宽度为 width、高度为 height 的矩形区域所对应的数据由 width ×height 个像素值组成。这些像素值按照从左至右的扫描线顺序与矩形区域中的每个像素一一对应。所有 RFB Client 端都必须具备处理 Raw 编码的能力，并且除非 Client 端显式要求使用其它编码，Server 端将默认使用 Raw 编码来传输数据。

2．CopyRect 编码

在 Client 端的帧缓冲区中已经包含所需的像素数据的情况下，CopyRect（Copy rectangle）是一种非常简单且高效的编码类型。这种类型的编码简单的由一对 X 和 Y 坐标组成。这对坐标在帧缓冲区中指定了一个位置，以这个位置为基点 Client 端可以得到其需要的像素数据。

CopyRect 编码适用于很多情形，最明显和常见的就是当用户在屏幕中移动窗口或是对窗口中的内容进行滚动的时候。此外，它还可以应用于优化文本或其它重复出现图元的绘制。一个优秀的 Server 端能做到对某个重复出现的图元模式只进行一次完整的像素数据传输，在了解该图元在帧缓冲区中最近一次出现位置的前提下，对相同图元的后续出现有效的使用 CopyRect 编码。

表 3.24 CopyRect 编码

字节数	类型	值	描述
2	U16		src-x-position
2	U16		src-y-position

3．RRE 编码

RRE 代表 rise-and-run-length encoding。如其名字所示，RRE 本质上是一种 run-length 编码的二维模拟。以 RRE 编码方式传输给 Client 端的矩形区域，能够被最简单的图形引擎快速而有效的生成和显示。RRE 不适用于复杂的桌面界面，但在某些情况下很有用。

RRE 的基本思想是将原本的矩形区域分割成若干子矩形区域，这些子矩形

区域都是由同一像素点填充而成。计算将给定矩形分割成若干这样的子矩形这的近优解，并不是个困难的问题。

该编码由一个背景像素点 Vb（通常是该矩形区域中出现最频繁的像素点）、数值 N 还有一个包含 N 个子矩形区域的链表组成；这里的每个子矩形区域表示为一个 5 元组<v,x,y,w,h>，其中 v（不等于 Vb）为像素点（值），（x,y）表示子矩形以原有矩形左上角为基点的相对坐标，而（w,h）表示该子矩形的宽度和高度。Client 端在收到 RRE 编码的数据后，可以按如下方式生成原有矩形区域：首先绘制由背景像素点 Vb 填充的原有矩形，然后再按照类似的方式为每个子矩形生成对应区域，即用每个子矩形对应 5 元组中的像素点 v 进行填充。

RRE 编码的头部以如表 3.25 所示，后跟如表 3.26 所示的子矩形像素数据的若干实例，实例的个数由编码头部中的 number-of-subrectangles 的值决定。

表 3.25 RRE 编码的头部

字节数	类型	值	描述
4	U32		number-of-subrectangles
bytesPerPixel	PIXEL		background-pixel-value

表 3.26 子矩形像素数据

字节数	类型	值	描述
bytesPerPixel	PIXEL		subrect-pixel-value
2	U16		x-position
2	U16		y-position
2	U16		width
2	U16		height

4. Hextile 编码

Hextile 编码是实际上 RRE 编码的一种变体。在 Hextile 编码中，原有矩形区域以 16×16 像素的块为单位，从矩形区域的左上角按照从左至右、从上至下的顺序进行分割。如果整个矩形区域的宽度不是 16 的整数倍，那么每行最后一块的宽度就适当缩小；类似的，如果整个矩形区域的高度不是 16 的整数倍，那么每列最后一块的高度也适当缩下。

分割得到的每个块或者使用 Raw 编码，或者使用 RRE 编码的变体。在使用 RRE 编码的变体时，每个块仍然有一个背景像素点，不过对一个给定块来说，如果它和排在其前面的块具有相同的背景像素点，那么就不需要显式指明其背景像素点。如果一个块的所有子矩阵都具有相同的像素点，那么这个像素点被定义为整个块的前景像素点。类似于背景像素点，前景像素点也可以不必显式指明，这种情况下该块同前一个块具备相同的前景像素点。

整个原有矩形的编码就是对每个块的依次编码。每个块的编码以一个“子编码类型”byte 开始，其值所表示的含义如表 3.27 所示。

表 3.27 Hextile 编码的头部

字节数	类型	值	描述
1	U8		subencoding-mask
		1	Raw
		2	BackgroundSpecied
		4	ForegroundSpecied
		8	AnySubrects
		16	SubrectsColoured

如果该 byte 中的 Raw 位被置位，那么其它位的值是无关紧要的；这种情况下该 byte 后面跟随的是 width x height 个像素值（这里 width 和 height 对应该块的宽度和高度）。

如果该 byte 中的 BackGroundSpecified 位被置位，则该 byte 后面跟随的是该块的背景像素点。原有矩形区域第一个非 Raw 编码的块的编码中必须将该位置位。

如果该 byte 中的 ForegroundSpecified 位被置位，该 byte 后跟该块的前景像素点。该位被置位时，则表 3.27 中的 SubrectsColoured 位必须为 0。

如果该 byte 中的 AnySubrects 位被置位，则该 byte 后跟一个 byte，表示后面所跟随的该块的子矩阵数量；如果 AnySubrects 位未被置位，意味着该块没有子矩阵，即整个块由背景像素点构成。

如果该 byte 中的 SubrectsColoured 位被置位，那么后面跟随的块的每个子矩阵的编码都要 其首部要加上其对应像素值；如果 SubrectsColoured 位被未置位，

该块的所有子矩阵具有相同的像素点，即块的前景像素点。

每个子矩阵中的位置和尺寸用 2 bytes 表示，即 x-and-y-position byte 和 width-and-height byte。由于每个块的最大尺寸为 16×16 像素，因此这 2 bytes 足够对块的子矩阵进行完整表示。x-and-y-position byte 的高 4 位对应子矩阵的 X 坐标，而低四位对应子矩阵的 Y 坐标。width-and-height byte 的高 4 位对应子矩阵的宽度减 1,而低四位则对应子矩阵的高度减 1。

5. ZRLE 编码

ZRLE[19]的全称是 Zlib Run-Length Enconding。ZRLE 编码综合使用了 zlib 压缩算法、分块和 run-length 编码。网络中传输的 ZRLE 数据流由两部分组成：4 字节的长度域，以及与该长度对应的使用 zlib 算法压缩后的数据（单位为 byte），如表 3.28 所示。

表 3.28 ZRLE 编码

字节数	类型	值	描述
4	U32		length
length	U8 数组		zlibData

Client 端将将 zlibData 对应的数据解压缩后，得到的是将原矩形区域以 64×64 像素的块为单位，按照从左至右、从上至下的顺序分割的结果，这和 Hextile 编码的情形很类似。如果矩形区域的宽度不是 64 的整数倍，那么每行最后一个块的宽度应适当缩小。类似的，如果矩形区域的高度不是 64 的整数倍，则每列最后一个块的高度也应适当缩小。

第四章 VNC 系统中视频播放性能的改进

第一节 VNC 对视频播放支持不足的原因

第二章常见瘦客户端系统之间的性能比较中，VNC 在视频方面的表现较差，对视频播放应用缺乏良好支持。

这个问题的主要原因是 VNC 系统的设计在低延迟和带宽利用之间如何权衡的问题上，优先考虑对带宽的有效利用，而把低延迟放在了次要位置。因此，VNC 系统采用的更新策略是 Client-pull 模式和懒惰更新，尽可能减少网络中传输的数据量。

视频播放应用的特点在于要求高带宽以及高频率的显示更新，并且这些显示更新的生成主要是由 Server 端的运行逻辑所驱动的，而不是由用户在 Client 端的操作所驱动的。现有 VNC 系统所采用的 Client-pull 模式中，显示更新则是由 Client 端的用户操作所驱动的，这导致了相邻的显示更新之间存在最小值为 RTT 的时间间隔，为显示更新的频率施加了上限，因而难以对视频播放应用提供良好的支持。

要改善 VNC 系统在视频播放方面的性能，需要改变 VNC 的更新策略，将现有的 Client-pull 模式改为 Server-push 模式，由 Client 端驱动改为由 Server 端驱动。

第二节 性能改进方案

如图 3.2 所示，在 VNC 系统中 RFB 协议位于较低的层次，它负责完成的工作包括对数据进行编码和执行数据传输；而 VNC 系统的上层逻辑，如传输哪些数据以及何时传输数据，则不属于 RFB 协议的工作，由 VNC Server 端的其它模块完成。这样的功能划分为 VNC 系统提供了良好的可修改性，在不对 RFB 协议进行修改的前提下，也可以改变 VNC 系统的整体特性。本文利用了这一优点来实现对视频播放应用的性能改进。

本章对现有 VNC 系统的性能改进，主要考虑或涉及以下三个问题：

1）将帧更新的驱动模式由 Client-pull 改为 Server-push。在 Server-push 模式中，无论 Client 端是否显式请求帧更新，视频播放中生成的每一帧后都放入帧缓

冲区中等待传输。这样就消除了原有系统中存在的相邻帧之间在传输中至少为 RTT 的时间间隔，为高频率的帧更新提供了实现的可能性。

2）原有系统中采用的懒惰更新与帧合并机制保持不变。单纯从降低延迟的角度考虑，积极更新是比懒惰更新更好的选择。然而要实现 VNC 系统对视频播放的良好支持，只是降低延迟是不够的，仍然需要考虑带宽利用的问题。视频播放是对带宽要求很高的应用，如果 Server 端改用积极更新策略的话，视频播放过程中的帧传输量会明显增大，而这些帧之间存在相当数量的冗余信息。数据传输量的增大意味着在带宽有限的情况下，帧在传输过程中的丢失率会变大，这可能导致系统整体能的下降而不是提高；即便是在带宽能够满足视频播放的需要情况下，数据传输量的增大也意味着对 Client 端处理能力要求的提高，这不符合 VNC 系统保持 Client 端尽可能简单的设计理念。

3）超时检查。帧在进行传输之前先检查其时效性，超时的帧将被丢弃。在视频播放这样的应用中，数据传输的及时性比可靠性更重要。对于 Client 端而言，一系列相邻帧中少数几帧的丢失对于视频显示效果的影响是有限的；然而对并不在预期时间之内到达的帧，其进行显示是无意义的。因此，在 Server 端增加超时机制、对无法及时传输的帧进行丢弃是有必要的，它减少了网络中无意义的数据传输；不过，在带宽有限的情况下，超时机制可能导致较高的帧丢弃率，对视频效果产生负面影响。

本节提出的设计修改方案的不利影响在于它会导致数据传输量的增大，增大了网络负担，在带宽条件有限的情况下系统性能不佳。

第三节 性能改进方案的实现

本文以 Linux 环境下的一个 VNC 系统的开源实现 Vino[20]为基础，对其更新策略进行修改，以改进现有 VNC 系统中视频播放应用的性能。

4.3.1 核心功能模块

Vino 的 Server 端部分包括如下几个核心功能模块：

1．vino-fb 模块

该模块负责实现 RFB 协议，完成低层次的编码和数据传输工作。

2．vino-input 模块

该模块负责处理从 Client 端收到的输入消息，并转发给对应的应用逻辑。

3．vino-server 模块

该模块负责 VNC 系统高层次的功能，如更新策略、合并机制等，并且包含了负责消息处理和发送帧更新的主循环。该模块是本章所进行的修改涉及的主要模块。

4．vino-main 模块

该模块负责实现 Server 端的初始化，同 Client 端建立连接，并启动 Server 端的主循环。

4.3.2 原有主循环

原有 vino-server 模块中的主循环的流程如图 4.1 所示。

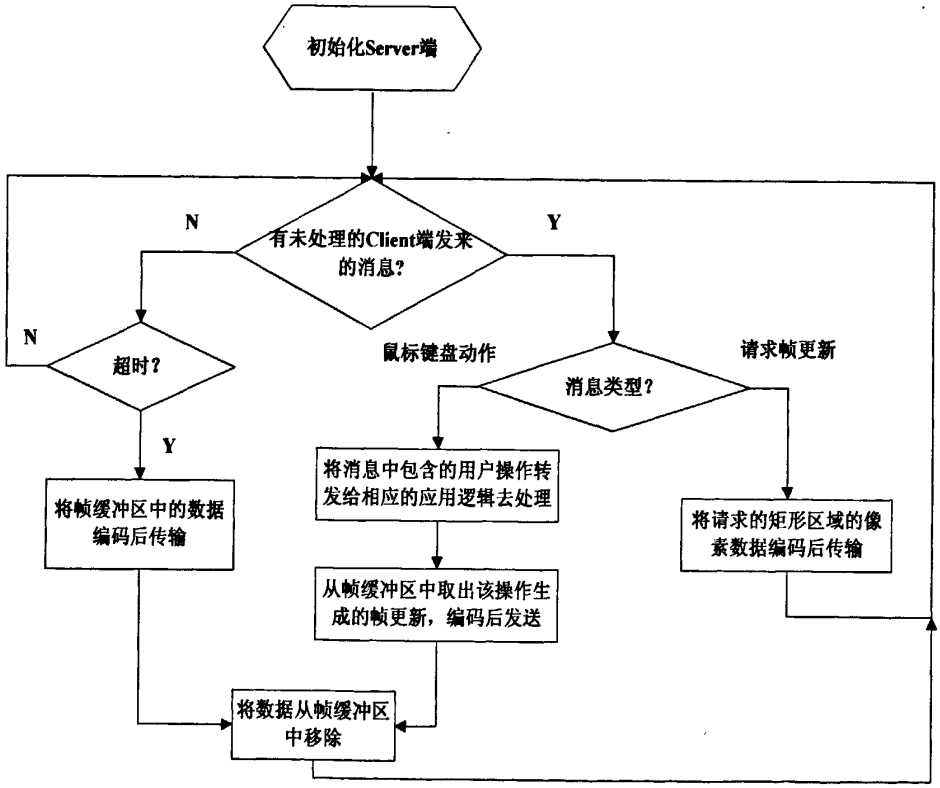

图 4.1 原有 vino-server 中的主循环

在原有的主循环中，Server 端在依照如下逻辑来发送帧更新：

1）首先检查是否有未处理的 Client 端发来的消息。如果没有，则跳到第 2 布。如果收到的是包含鼠标键盘操作的消息，Server 端将消息中的操作转发给相应的应用程序，等待应用程序完成操作对应的任务后，将产生的帧更新使用 RFB 协议编码后进行传输。如果如果收到的是显式请求帧更新的

FramebufferUpdateRequest 消息（参见 3.6.4），Server 端直接将该消息所请求的矩形区域的像素数据使用 RFB 协议编码后进行传输。

2）进行超时检查。Server 端将计算帧缓冲区中数据的滞留时间，如果超过了预设的上限，则将帧缓冲区中的数据编码后发送。这里超时检查的目的是为了降低如下问题的严重性：Server 端不断产生帧更新，却由于一直未收到 Client 端的请求信息而发送帧更新，结果 Client 端的显示一直未得到更新，Client 端和 Server 端之间出现了严重的显示脱节。如果 Vino 不包含该超时机制，那么其对视频播放的支持比现在更加糟糕，因为视频播放是典型的非用户驱动型的应用，视频播放过程中用户很少会执行操作。

4.3.3 修改后的主循环

修改后的 vino-server 模块中的主循环流程如图 4.2 所示。

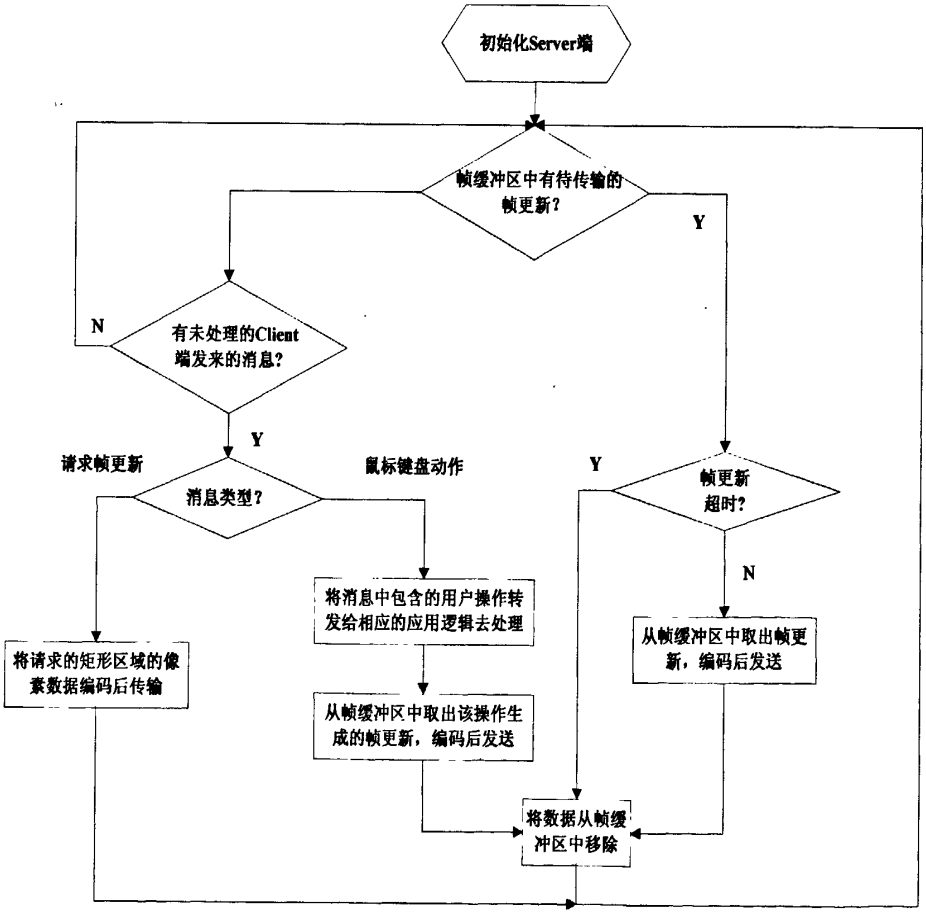

图 4.2 修改后 vino-server 中的主循环

修改后的主循环中，Server 端依照如下逻辑来发送帧更新：

1）首先检查帧缓冲区中是否有等待传输的帧更新。如果帧缓冲区中数据为空，则转到第 2 步。如果存在等待传输的帧更新，则计算从该帧更新被放入帧缓冲区中到此刻为止的时间间隔；如果该帧更新对应的时间间隔超出了超时机制设定的上限值，则意味着该帧更新无法以低延迟到达 Client 端，将会被丢弃，并从帧缓冲区中移除；若未超时，则将该帧更新使用 RFB 协议编码后进行传输，从帧缓冲区中移除该帧，并进入主循环的新一轮。

2）检查是否有未处理的 Client 端发来的消息。如果没有，则为主循环的下一轮作准备。如果收到的是显式请求帧更新的 FramebufferUpdateRequest 消息（参见 3.6.4），Server 端直接将该消息所请求的矩形区域的像素数据使用 RFB 协议编码后进行传输；如果收到的是包含鼠标键盘操作的消息，Server 端将消息中的操作转发给相应的应用程序，等待应用程序完成操作对应的任务后，将产生的帧更新使用 RFB 协议编码后进行传输。

4.3.4 修改前后的对比

修改前后主循环中完成的操作存在一定的相似之处，例如都要检查有未处理的 Client 端发来的消息，都要检查帧缓冲区中的数据是否超时。然而两者之间存在两点关键性的区别：

1）Clinet 端发送的请求消息同 Server 端发送帧更新之间的关系不同。在原有主循环中，不考虑超时机制的话，Client 端发送请求消息是 Server 端发送帧更新的必要条件。在修改后的主循环中，Client 端发送请求消息并不是 Server 端发送帧更新的必要条件。

2）超时检查机制的目的不同。在原有主循环中，超时检查是作为辅助性的被动式预防机制而存在，目的是限制 Client 端和 Server 端之间显示界面偏离化的严重程度。在修改的主循环中，超时检查则是作为不可缺少的主动优化机制而存在，目的是尽可能优化 Client 端和 Server 端之间显示界面的同步性。

第四节 本章小结

本章分析了现有 VNC 系统缺乏对视频播放应用的良好支持的原因，在此基础上为改善其对视频播放的支持对现有设计进行了部分修改，并基于 VNC 系统的开源实现 Vino 对这些设计修改进行了实现。下一章进行的性能测量将证实该

性能优化方案有效改善了视频播放应用的性能。

第五章 性能测量方案与测量结果

第一节 性能测量所面临的问题

由于瘦客户端系统和传统的桌面系统在设计和使用上存在很大差异，对其系统性能进行有效的测量存在一定的困难。在传统桌面系统中常用的基准测量（Benchmark）不能直接用于对瘦客户端系统进行性能测量。这是因为在瘦客户端系统中，基准测量程序的运行逻辑是在 Server 端运行的，因此这样的基准测量只是对 Server 端的处理能力进行测量，而无法对用户在 Client 端得到的实际效果进行准确测量。

在大多数瘦客户端系统中，为了提高 Client 端的显示性能，特别是在网络带宽有限的情况下，通常会采用三种优化方法：压缩，缓存和合并[21]。这三种优化方法的存在，特别是合并机制，都为瘦客户端系统的性能分析制造了困难。

压缩是指应用各种压缩算法（如 zlib）对要传输的图像数据进行压缩，在只增加有限计算开销的前提下，降低对网络带宽的要求。

缓存是指 Client 端可以在其本地存储某些图元(例如图标)对应的像素数据，这样对于某些频繁使用的图元，Client 就可以从本地获取像素数据而不要从 Server 端获取。

合并是指 Server 端在将图像更新（消息）发送给 Client 端之前在其本地进行排队和合并处理：如果两条图像更新针对的是屏幕的同一区域，那么较早的更新将被忽略，只有最近一次的更新将会被发送。合并机制的应用意味着应用逻辑的执行和图像更新的生成这两者不再是一一对应关系。这也意味着从用户在 Client 端发出操作到 Client 端得到该操作导致的帧更形并进行显示之间，除了网络传输延迟外还存在合并机制引入的附加延迟，附加延迟的长度同具体使用的合并机制相关。

第二节 性能测量方案的选择

要对瘦客户端系统正确的进行性能测量，应该基于用户在 Client 端得到的实际体验进行。因此，对瘦客户端系统的性能测量解决的主要问题是：如何正确、客观的测量用户在 Client 端体验到的实际性能，而不是错误的测量 Server 端的

性能。

5.2.1 待选的性能测量方案

待选的性能测量方案主要有以下三种：

1．直接使用现有的基准测试程序。

该方案将在 Server 端运行现有的基准测量程序，如同在普通桌面系统中一样。例如运行一个视频播放基准测量程序，以视频播放的帧率来作为性能测量的指标。然而，在瘦客户端系统中这种方法无法提供准确的测量结果。基准测量的所有运行逻辑都是在 Server 端执行的，由于前面提到的优化机制的存在，Server 端上的运行逻辑是和 Client 端的图像显示比不存在时间上严格的对应关系，这样基准测量程序只能测量出 Server 端对视频进行解码、生成图像帧的能力；然而，由于合并机制以及网络传输中丢包的存在，Server 端生成的帧并不会全部到达 Client 端，这样与比用户体验的实际性能相比，基准测量程序会得到一个被夸大的测量结果。

2．在 Client 端直接进行测量。

如果在瘦客户端系统中加入有效的日志和统计机制，从而在 Client 端记录所有的输入输出事件，那么就能够进行详细可靠的性能测量。与前一种方案相比，这种方案更加准确和可靠。然而，这种测量方法在实际应用中存在一定困难。首先，多数瘦客户端系统都是很复杂的，要加入有效的日志和统计功能并不容易。其次，这种对原有系统侵入式的功能扩展可能会导致系统性能的变化，从而影响测量结果的可靠性。最后也是更现实的问题是，多数瘦客户端系统都是闭源的私有软件，有些甚至连详细的系统设计规格都不公开提供。

3．基于网络传输监测进行测量。

如前所述，只是在 Server 端运行基准测量程序可能会得到不可靠的结果，而在 Client 端进行侵入式的直接测量又存在实现上的困难；另一方面，通过监测程序运行过程中 Server 端和 Client 端之间的网络传输，可以得到更为接近用户实际体验的测量结果。例如，要测量 Web 页面下载操作在 Server 端和 Client 端之间的延迟，可以监测 Client 端与 Server 端的网络传输情况，得到数据传输的起始和结束时间，进而计算出对应的延迟。

然而，尽管该方法能够更精确的测量 Server 端和 Client 端之间在显示更新上的延迟，但仍存在一个问题：难以全面的衡量用户体验。延迟是影响用户体验

的重要因素，但用户体验还受其它因素影响。例如显示更新帧的丢失率，特别是对于视频播放，同样是影响用户体验重要的因素；即使保证了低延迟，如果大量的帧被丢弃，也是难以在 Client 端实现良好的用户体验的。

对于普遍在 Server 端采用合并机制的瘦客户端系统，这种测量方法存在的问题尤为明显：Server 端在网络带宽有限的情况下常会选择将部分显示更新简单的丢弃，避免造成高延迟，即通过牺牲帧丢失率来控制延迟。这样在网络带宽很低的情况下，该方法仍然会得到良好的性能测量结果，因为它只测量延迟，而 Server 端只会选择能够立刻传输的包，而任何需要等待的更新帧都会被直接丢弃。很显然，这个良好的性能结果是具有欺骗性的。

5.2.2 Slow-motion 基准测量

上一节分析表明三种方案中任何一种在单独使用时都不够理想。要进行有效、可靠、准确的测量，较好的解决方案是将以上三种方案综合起来，使用基于网络监测的 slow-motion 基准测量[22]。在该种方案中，一方面通过抓取网络数据包来监测 Server 端和 Client 端之间的数据传输和延迟，另一方面对原有的基准测量程序进行修改，在不同帧的生成之间增加足够的延迟，以保证在 Server 端生成下一个帧之前，当前帧已经发送到 Client 端并被显示。

在使用这种测量方案时，由于 Server 端保证在每帧的传输之间存在一定的时间间隔，相应的 Client 端在帧的接受上也存在一定的时间间隔，因而在监控到的传输流量中会出现明显的空白。以这个空白为特征，就可以将捕获的数据包以帧为单位进行分组。这样就能够得到基准测量程序中每帧相对应的数据传输量和延迟。这个时间间隔的大小对于测量结果的准确性具有重要影响，应选取一个足够大的值，以保证不同帧在网络传输流量中呈现非连续化。

slow-motion 基准测量方案是以如下三点假设作为前提的。

1）帧之间加入的时间间隔不会对瘦客户端系统中在 Client 端和 Server 端之间原本应该传输的数据量产生影响

2）引入的时间间隔，将在 Client 端捕获的数据流中变现为可被观察到的空白。这意味着假设瘦客户端系统在其空闲时不会进行数据传输或者传输的数据量与非空闲时刻的数据量相比可以忽略不计。

3）Client 端从收到帧至帧被显示之间的间隔可被忽略不计。

slow-motion 基准测量方案具备如下优点：

1）首先也是最重要的一点，它能保证每帧都在 Client 端可靠被显示，这样通过监测网络传输就能提供一种具备准确性和可重复性的测量方法。

2）slow-motion 基准测量不要求对已有的瘦客户端系统进行侵入式的更改[23]。这一优点具备重要的现实意义，因为多数瘦客户端系统都是闭源的私有系统；此外由于避免了对被测系统进行侵入式修改，不会导致被测系统性能发生变化。

3）slow-motion 基准测量提供了一种有效的方法，用于检测测量过程中的帧丢失率。因为 slow-motion 基准测量在机制上能将帧丢失率尽可能降低，因此 slow-motion 基准测量得到的数据传输量可以作为基准，与常规基准测量中的数据传输量进行比较，来计算常规基准测量中的帧丢失率。

4）与常规基准测量相比，slow-motion 基准测量在某些情况下与典型的用户行为更相符[24]。例如，用户在浏览 Web 页面时，通常不会连续不断的请求新的数据，而是一种更离散化的行为模式，即用户在浏览完当前页面之前，不会发出对新页面的请求。

第三节 VNC 系统的视频播放性能的测量与比较

本节使用 slow-motion 基准测量来对 VNC 系统中视频播放进行性能测量，并对原有 VNC 和针对视频播放进行优化后的 VNC 的性能进行比较。

5.3.1 面临的问题

在瘦客户端系统中进行视频播放时，Server 端首先对视频进行解码，然后再将解码生成的帧发送给 Client 端。在对瘦客户端系统中视频播放进行性能测量时，存在以下两个问题需要解决：

1）大多数视频播放软件会在系统的计算能力无法满足对视频解码并生成帧的时候，对部分帧进行丢弃。

2）瘦客户端系统自身也可能存在帧丢弃的可能。在网络阻塞的情况下，Server 端会对部分帧进行丢弃[25]。

通过使用 slow-motion 基准测量，可以解决这两个问题。多数视频播放软件允许调整视频的播放帧率，因此可以通过将帧率降到足够低，以保证播放软件和瘦客户端系统这两个层次上的帧丢失。虽然用户通常并不会以这种低帧率播放视频，但是在该低帧率下得到的测量结果可以用来作为比照基准。

5.3.2 视频质量 VQ

本节将视频播放的数据传输率定义为通过网络监控得到数据传输总量与视频的播放时间之间的比值。将低帧率条件下的数据传输率作为基准值，用正常帧率条件下的数据传输率与基准值进行比较，得出的比值可以作为 Client 端视频质量（Video Quality）的衡量尺度。

正常播放帧率为 P 的视频，其在 Client 端的视频质量 VQ 的规范定义如下：

$$VQ(P)=\frac{\left[\frac{DataTransferred(P)/PlaybackTime(P)}{FPS(P)}\right]}{\left[\frac{DataTransferred(slow-motion)/PlaybackTime(slow-motion)}{FPS(slow-motion)}\right]} \tag{5.1}$$

例如，假设测量中使用的视频以正常帧率 24 fps 播放时，播放时间为 30s，通过网络监控得到的数据传输总量为 10M bytes；而在以 1fps 的低帧率播放时，播放时间为 360s，通过网络检控得到的数据传输总量为 20 Mbytes。此时根据公式(5.1)可计算出 Client 端的视频质量 VQ 为 50%，这意味着在以 24fps 的帧率播放时，Server 端产生的帧有 50%到达了 Client 端，而另外的 50%的帧则被 Server 端丢弃。在大多数情况下，1 fps 的低帧率就足以保证 Server 端生成的每一帧都被传输并在 Client 端生成图像。对于少数特殊情况，则需要将帧率降的更低[26]。

视频质量 VQ 提供了一种非侵入式的性能测量尺度。它实际上反映了帧的丢弃率。VQ 作为性能尺度的有效性和可信性依赖于以下假设：每一帧对于 Client 端显示效果的影响是相同的，并不存在具备特殊重要性的帧。

一方面，该假设在实际应用中并不总是成立的[27]。例如视频的画面变化较小的情况下，相邻帧之间的差异较小，丢弃一帧对 Client 端的显示效果影响不大；而视频的画面变化较大时，相邻帧之间的差异较大，丢弃一帧对显示效果则有明显影响。

另一方面，多数瘦客户端系统（包括 VNC）都在 Server 端采用了帧合并和压缩机制，以消除数据中的冗余信息[28]。如果将每一帧的重要性以其所包含的非重复信息来衡量，那么帧的重要性就与帧的数据大小基本呈正比关系。因而基于帧丢弃率的视频质量 VQ，即使在帧之间的重要性存在差异的情况下，也可以作为有效、可信的性能测量尺度。

5.3.3 实验环境

实验环境由三台机器组成：VNC Client，VNC Serer 和一台用于模拟 Client 与 Server 之间网络环境的网络模拟器，结构如图 5.1 所示。网络模拟器是通过软件模拟的方式实现的，所使用的模拟软件为 NIST NET[29]，它是 Linux 环境下基于命令行的网络模拟器软件，可以对网络带宽、传输延迟、数据包大小、网络负载等参数进行模拟。实验环境中的网络传输监控由通过 Tcpdump[30] 完成。

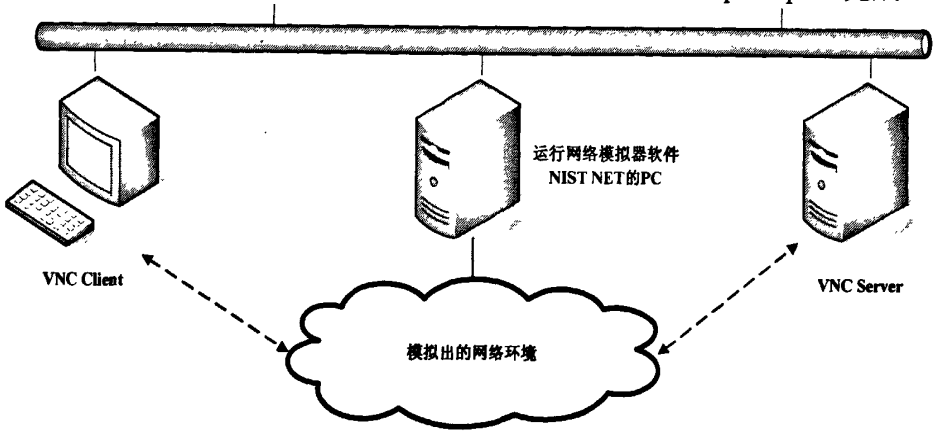

图 5.1 实验环境示意图

5.3.4 性能测量结果与分析

通过网络模拟器对四种不同带宽的网络环境进行模拟，并对原有和改进后 VNC 系统的视频播放进行性能测量，测量结果如图 5.2 所示。

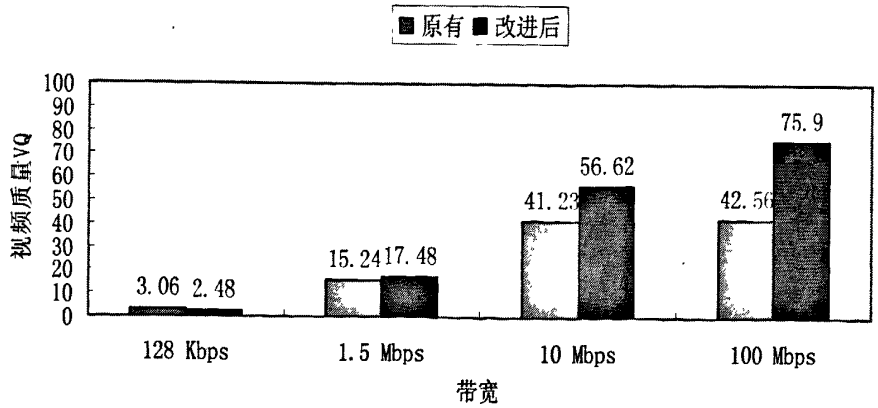

图 5.2 原有和优化后 VNC 系统的视频播放性能对比

对图 5.2 中 VNC 系统的视频播放性能在改进前后的对比进行观察，可以得

到如下结论。

1）在原有 VNC 系统中，视频质量 VQ 随着网络带宽的改善而提高；但当网络带宽增大到高于 10 Mbps 之后，VQ 的提高变得很小。这表示对于原有 VNC 系统，在网络带宽增大的一定程度后，继续增大带宽并不能进一步的提高视频质量 VQ。此时限制 VQ 提高的主要因素是 Client 端和 Server 端在显示更新上的延迟。

2）在改进后的 VNC 系统中，视频质量 VQ 同样随着网络带宽的改善而提高。与原有 VNC 系统不同的是，从 10 Mbps 到 100 Mbps，VQ 的值有着明显的提升。这表明与原有 VNC 系统相比，新系统能够通过网络带宽的改善获得更大的性能提升。

3）注意到在网络带宽为 128 Kbps 这样较为有限的情况下，改进后的系统与原系统相比并不存在性能优势，反而存在性能劣势。这是因为改进后的系统在设计中优先考虑降低延迟而不是节约带宽，因而导致数据传输量的增大，也意味着延迟的增大和网络传输中数据丢失率的增大。因此，在网络带宽有限的情况下，数据传输量增大所导致的延迟增大将超出采用积极更新和 Server-push 模式所带来的延迟降低，因而整体性能上表现为下降。这表明新系统以提高对带宽需求为代价而换取更佳性能，它并不适用于网络带宽有限的环境。

第四节 本章小结

本章分析了对瘦客户端系统进行性能测量时面临的问题，对几种潜在的测量方法进行了分析和比较，并选择了较为理想、易于实现的 slow-motion 基准测量方案，对现有 VNC 系统以及第四章中针对视频播放进行改进后的 VNC 系统在视频播放方面进行了性能测量，并对测量结果进行对比分析。分析表明，第四章中提出的设计修改方案确提高了视频播放的性能。虽然这样的设计修改方案会导致网络中数据传输量的增大，但是考虑到视频播放应用本质上是要求高带宽的应用，在多数情况下与原有系统相比，该方案对 VNC 系统的整体性能是利大于弊的。

第六章 总结与展望

第一节 论文总结

论文的主要工作和创新点包括以下几个方面：

1）介绍了各种瘦客户端系统的设计中共有的基本问题，并对每个问题中不同设计策略对系统性能可能产生的影响进行了分析。

2）指出现有 VNC 系统对视频播放应用缺乏良好支持的原因，并为改善其对视频播放应用的支持提出了一种设计修改方案，并基于开源项目 Vino 实现这一设计修改方案。

3）使用 slow-motion 基准测量方案对原有和改进后 VNC 系统中的视频播放性能进行了测量。测量结果和对比分析表明该修改方案是有效的。

第二节 进一步的工作

对 VNC 系统中视频播放应用的性能改善，还可以从以下几方面进一步进行：

1）为视频播放应用提供良好的支持，不仅需要保证相邻帧到达 Client 端的时间间隔尽可能的小，还需要将这个时间间隔保持在稳定的水平上，尽量减少一系列连续帧的时间间隔曲线中毛刺（spike）出现的频率。这可能需要在 Server 端采用比当前系统更为复杂的缓冲机制。

2）在本文提出的设计修改方案基础上，在降低延迟和增大数据传输量之间取得更好的平衡，尽可能消除低带宽环境下该设计修改方案产生的负面影响。

参考文献

[1] Albert Lai and Jason Nieh .Limits of Wide Area Thin Client Computing, Proceedings of the ACM SIGMETRICS 2002, Marina del Rey, CA, June15-19, 2002

[2] J. Nieh and S. J. Yan .Measuring the Multimedia Performance of Server-Based Computing. In Proceedings of the 10th International Workshop on Network and Operating System Support for Digital Audio and Video, pages 55–64, Chapel Hill, NC, June 2000

[3] 梁飞蝶，李锦涛.瘦客户端应用协议中远程显示机制的比较.计算机工程与应用，2004，21：135~138

[4] S. J. Yang, J. Nieh, M. Selsky, and N. Tiwari .The Performance of Remote Display Mechanisms for Thin-Client Computing .In Proceedings of the 2002 USENIX Annual Technical Conference, Monterey, CA, USA, June 2002

[5] R. W. Scheifler and J. Gettys . The X Window System, ACM Transactions on Graphics, 5(2), Apr. 1986

[6] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper .Virtual Network Computing. IEEE Internet Computing, 2(1), Jan./Feb. 1998

[7] Tristan Richardson. The RFB Protocol, AT&T Labs Cambridge Whitepaper, March, 2005.

[8] Microsoft, Remote Desktop Protocol Features and Performance White Paper, 2000.6.

[9] ITU-T T.122: "Multipoint communication service –Service definition", http://www.itu.int/, Feb. 1998.

[10] ITU-T T. 128 Draft - Application Sharing Protocol .

[11] ITU-T T. 125 Draft – Multipoint Communication Service.

[12] Microsoft Corporation, Windows 2000 Terminal Services Capacity Planning, Technical White Paper, Redmond, WA, 2000

[13] A. Y. Wong and M. Seltzer .Evaluating Windows NT Terminal Server Performance. In Proceedings of the 3rd USENIX Windows NT Symposium, pages 145–154, Seattle, WA, July 1999

[14] 梁飞蝶，李锦涛，史红周.虚拟网络计算(VNC)协议中的编码方法.计算机应用,2004,6,93~95

[15] S.Jac Yang, Jason Nich, Matt sclsky et al .The Performance of Remote Display Mechanisms for Thin-Client Computing[C]. In: Proceeding of the 2002 USENIX Annual Technical Conference, 2002-06

[16] J.Nich , S.J.Yang ,Novik.A . Comparison of Thin-Client Computing Architectures[R]. Technical Report CUCS-022-00 , Department of Computer Science , Columbia University , 2000-11

[17] B. Vandalore, W. Feng, R. Jain, and S. Fahny .A survey of Application Layer Techniques for Adaptive Streaming of Multimedia.Journal of Real-time Systems (Special Issue on Adaptive Multimedia) , January 2000

[18] Surya Nepal, Uma Shrinivasan .Adaptive Video Highlights for Wired and Wireless Platforms, Proceedings IEEE International Conference on Multimedia and Expo, Baltimore, USA, July 2003.

[19] Alex Buell . Framebuffer HOWTO, http://www.faqs.org/docs/, Feb. 2000.

[20] Vino，www.gnome.org/~markmc/remote-desktop-2.html

[21] The Three Components of Optimizing WAN Bandwidth, Ashton, Metzler& Associates, Technical Whitepaper, March 2003

[22] S. J. Yang, J. Nieh, and N. Novik .Measuring Thin-Client Performance Using Slow-Motion Benchmarking .In Proceedings of the 2001 USENIX Annual Technical Conference, pages 35–49, Boston, MA, June 2001

[23] D. Wu, Y. T. Hou, W. Zhu, Y-Q Zhang, and J. M. Peha . Streaming Video over the Internet: Approaches and Directions. IEEE Transactionson Circuirs and Systems for Video Technology, VOL. 11, NO.3, March 2001

[24] Surya Nepal and Uma Srinivasan. Quality Adaptation of Video for Delivery in Both Wired and Wireless Platform, CSIRO MIS Report, 2002.

[25] Sun Yang，Tay TengTiow . Long Distance Redundancy Reduction in Thin Client Computing,6th IEEE/ACIS International Conference on Computer and Information Sciecnce (ICIS 2007)

[26] Streaming video over the Internet: Approaches and Directions, IEEE Transactions on Circuit and Systems for Video Technology, Vol. 11, No. 1, February 2001

[27] P.L Tsai and C.L. Lei . Towards ubiquitous computing via secure desktop service, Proceedings of IEEE Region 10 International Conference on Electrical and Electronic Technology, 2001, Vol. 1, pp. 187-190,August 2001]

[28] Sun Yang ，Tay TengTiow . Improving Interactive Experience of Thin Client Computing by Reducing Data Spikes ，6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)

[29] NIST NET，http://snad.ncsl.nist.gov/itg/nistnet/index.html

[30] Tcpdump，http://www.tcpdump.org/

致谢

首先衷心地感谢我的导师张建忠教授。在我三年的研究生学习以及生活中，张老师给予了我热忱的关心和悉心的指导，正是他严格的要求和经常的督促，使我能够不断激励自己、提高自己。张老师渊博的知识、严谨的治学态度和一丝不苟的科研精神更是使我受益匪浅，为我今后的学习、工作和生活树立了榜样。

衷心地感谢吴功宜教授，吴老师敏锐的思维和严谨的治学态度，给我留下了十分深刻的印象。衷心地感谢徐敬东教授，徐老师的谆谆教诲令人如沐春风。

感谢赵奇、黄国伟、陈志、申庆勇、付治均、赖海明、刘晓欣、楼霓祥、杨洁、尹乐等师兄师姐，感谢他们在我进入实验室之后的科研和学习生活中给予我的无私帮助与指导。

感谢高立金、卢红波、兰小丰、王珺、董大凡、刘乾、阮星华、杨一林、杨林，与你们相处的三年里，我学会了很多也得到了很多，这段快乐时光点点滴滴铭记在心，将成为我一生中最美好的回忆。

感谢杨阳、沈鑫、殷宇辉、朱剑彬、于博洋、薛保华、张晨生、李潇、秦婧、彭玉娣、刘皎瑶、等师弟师妹，与你们相识，我的生活更加多姿多彩。

感谢我的父亲母亲，你们给我的细心照顾和默默的支持永远是我前进的动力。

最后，再一次向所有关心、支持和帮助过我的老师、亲人和朋友表示深深的谢意！

附录

附录 A：图索引

附录 B：表索引

个人简历

个人简历：

吴筱桉，男，1983 年 4 月 27 日生。2005 年 5 月毕业于天津南开大学信息技术与科学学院计算机系，获工学学士学位。2005 年 9 月至 2008 年 7 月就读于天津南开大学信息技术与科学学院计算机应用与技术系，攻读工学硕士学位。