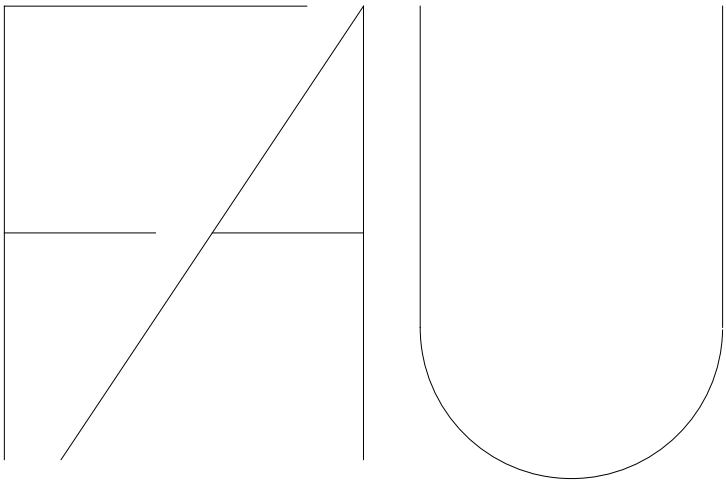
Entwurf und Implementierung eines RDP-Clients für das Java-Betriebssystem JX

Christian Flügel

Juli 2002 SA-I4-2002-11

Studienarbeit

Institut für Mathematische Maschinen und Datenverarbeitung der Friedrich-Alexander-Universität Erlangen-Nürnberg



Entwurf und Implementierung eines RDP-Clients für das Java-Betriebssystem JX

Studienarbeit im Fach Informatik

vorgelegt von

Christian Flügel

geb. am 24.11.1976 in Marktredwitz

Angefertigt am

Institut für Mathematische Maschinen und Datenverarbeitung (IV)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer:	***Prof. Dr. F. Hofmann***
	Dipl.-Inf. Michael Golm
	Dipl-Inf. Meik Felser
	Dipl-Inf. Christian Wawersich

Beginn der Arbeit:	1. Oktober 2001
Abgabe der Arbeit:	1. Juli 2002

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 08.08.2002 _____

Kurzfassung

Es existiert kaum noch ein Bereich des täglichen Lebens, der nicht von Rechnern dominiert wird. Ein zentrales Problem, das besonders im professionellen Umfeld besteht, ist der zunehmende administrative Aufwand, den die Rechenanlagen erfordern. Ein mögliches Konzept diesen Aufwand zu begrenzen besteht darin, alle Rechenkapazitäten auf einem zentralen Server zu bündeln. Die Arbeitsplätze bestehen nur noch aus Terminals, sogenannten “Thin-Clients”, die über ein Netzwerk an den Server angebunden sind und oft nur aus Monitor, Tastatur und einem minimalistischen Rechner bestehen. Die einzige Aufgabe eines solchen Thin-Clients besteht darin, Benutzereingaben entgegenzunehmen und die Ausgabe des Servers darzustellen.

Ein weitverbreitetes Produkt, das diese besondere Form der Client-Server Architektur umsetzt, ist der Terminal Server der Firma Microsoft [CCM99] und das dazugehörige “Remote Desktop Protocol”. In Thin-Clients kommen oft spezielle Betriebssysteme zum Einsatz. Das Java-Betriebssystem JX [GFW02] ist für diese spezielle Anwendungsklasse besonders gut geeignet, da sich dessen Betriebssystemkern gut an das jeweilige Anforderungsprofil anpassen lässt. Ziel dieser Studienarbeit ist es, eine unter JX lauffähige Implementierung des Remote Desktop Protokolls zu schaffen, welches Terminal Server und Client zur Kommunikation verwenden. Außerdem soll ein Client Programm entwickelt werden, mit dem man auch unter JX auf einen Terminal Server zugreifen kann.

Der erste Teil dieser Arbeit beschäftigt sich ausführlich mit dem grundlegenden Konzepten sowie den einzelnen Schichten des RDP-Protokolls. Im zweiten Teil wird die Implementierung des RDP-Clients, die in der Programmiersprache Java erfolgt ist, näher erläutert.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Zielsetzung und Rahmenbedingungen

In den letzten Jahren ist Computertechnologie immer preiswerter geworden. Kostete vor zwanzig Jahren eine leistungsfähige Rechenanlage noch ein vielfaches eines Einfamilienhauses, so ist es heute selbst einer Privatperson möglich sich einen leistungsfähigen Rechner zu leisten.

Die Forschritte in der Mikrotechnologie und in der Netzwerktechnik haben dazu geführt, dass der PC aus Industrie und Forschung heute nicht mehr wegzudenken ist. Doch mit der explosionsartigen Verbreitung der Rechner sind auch völlig neue Problembereiche entstanden. Waren es früher vor allem die hohen Anschaffungskosten für Hard- und Software, die den Rechenbetrieb kostenintensiv erscheinen ließen, so entfällt heute ein Großteil der Kosten auf die Administration und Absicherung der Systeme. Gerade in großen Firmen, wo viele hundert Rechner über mehrere Standorte verteilt sind, spielen die Anschaffungskosten nur noch eine untergeordnete Rolle. Dort ist vor allem wichtig den Aufwand für Wartung und Administration der Infrastruktur in den Griff zu bekommen.

Eine Möglichkeit dies zu erreichen ist die Verwendung sogenannter *Thin-Clients*. Es handelt sich dabei um eine Kombination aus Monitor, Tastatur und einem Gerät, welches nur noch die Verbindung zu einem Server über ein lokales Netzwerk (Ethernet, Token-Ring etc.) herstellt. Dieser Thin-Client stellt für den Benutzer eine Schnittstelle zum entfernten Hauptrechner dar. Der Vorteil liegt darin, dass nur noch ein zentraler Rechner beaufsichtigt und gewartet werden muss, trotzdem aber jedem Mitarbeiter ein eigener Rechnerarbeitsplatz zur Verfügung steht.

Fast alle größeren Computerhersteller bieten Produkte auf Basis von Thin Clients an. So zum Beispiel Sun Microsystems mit seiner SunRay Reihe [SUN], IBM mit der NetVista Serie [IBM] oder Compaq mit der Evo Produktlinie [COM]. Serverseitig kommen oft proprietäre Eigenentwicklungen zum Einsatz. Es existieren jedoch auch Alternativen dazu: Man kann entweder ein Unix-Derivat in Verbindung mit dem X-Window System [TOG] einsetzen oder man verwendet Microsofts Windows NT Server (entweder in der Version 4.0 oder 2000), der in einer speziellen Terminal Server Edition erhältlich ist. Auf Seiten des Clients kommen fast immer spezielle Betriebssysteme für Thin-Clients zum Einsatz. Es handelt sich dabei meist um Eigenentwicklungen

Diese Studienarbeit wurde im Rahmen des JX-Projektes erstellt. JX ist ein Java-basiertes Betriebssystem, bei dessen Entwicklung vor allem eine gute Modularisierung, ein leistungsfähiges softwarebasiertes Speicherschutzkonzept sowie die durchgehenden Anwendung von OO-Konzepten, bei der Entwicklung im Vordergrund standen. JX enthält eine Java-Laufzeitumgebung, einen vollständigen Netzwerkstack, Windowmanager und vieles mehr. Das herausragende an JX ist jedoch das Domainkonzept, welches eine sehr feine Aufteilung des Systems in einzelne Komponenten erlaubt. Das System lässt sich so sehr genau auf die jeweiligen Anforderungen anpassen. Diese Flexibilität macht JX auch als Betriebssystem für Thin-Clients interessant.

Ziel dieser Studienarbeit war es, einen lauffähigen RDP-Client für das JX-System zu entwikkeln. Bei RDP handelt es sich um das Protokoll, welches Microsofts Terminal Server zur Kommunikation verwendet. Der erste Teil der Arbeit beschäftigt sich mit dem eigentlichen RDP-Protokoll, der zweite Teil beschreibt die Implementierung des Clients in der Programmiersprache Java.

1.2 Kapitelübersicht

Im nächsten Kapitel werden zunächst einige Grundlagen näher erläutert, welche für die Studienarbeit von Bedeutung sind. Es geht dabei vor allem um eine Übersicht über die gängigsten Netzwerkprotokolle und die Einordnung von RDP in die entsprechenden Referenzmodelle. Außerdem wird eine kurze Übersicht über den Aufbau von JX gegeben sowie über die grundlegenden Konzepte auf denen JX aufbaut. Kapitel Drei beschäftigt sich näher mit dem RDP-Protokoll.Es werden die einzelnen Schichten des Protokolls detailliert beschrieben. Dabei wird auch auf die Unterschiede zu den zugrundeliegenden Standards näher eingegangen. Kapitel Vier behandelt die Implementierung des Clients und beinhaltet außerdem eine Leistungsbewertung des Clients. Die Arbeit endet mit einer kurzen Zusammenfassung der Arbeit sowie einigen abschließenden Bemerkungen.

2 Grundlagen

2.1 Einleitung

In diesem Kapitel werden einige für die Studienarbeit wichtigen Konzepte und Prinzipien näher erläutert. Der erste Abschnitt beschäftigt sich näher mit dem Client-Server Modell und dem Konzept der Thin-Clients. Danach werden die beiden wichtigsten Netzwerkarchitekturen, das ISO/OSI- und das TCP/IP-Referenzmodell sowie der Aufbau des RDP-Protokolls näher erläutert. Abschließend folgt eine Einführung in das Java-basierte Betriebssystem JX, das Ausgangsbasis für diese Studienarbeit war.

2.2 Thin Clients

In der Regel besteht ein Arbeitsplatzrechner aus mehreren Teilen: Monitor, Tastatur und Maus sowie ein Gehäuse, welches die Zentraleinheit sowie benötigte Zusatzkomponenten enthält. Für eine ganze Reihe von Anwendungsgebieten ist so ein Rechner jedoch entweder ungeeignet oder schlicht überdimensioniert. Die alltäglichen Büroarbeit kann selbst mit einem Rechner der vorletzten Generation mühelos bewältigt werden.

Abb. 2.1 Ein SunRay System

Leistungsfähige Computer sind heute so billig wie nie. Für weniger als 500 Euro bekommt man heutzutage Rechenleistung, die vor zehn Jahren noch das zehnfache gekostet hätte, und dabei selbst gehobenen Ansprüchen genügt.

Die Fortschritte in der Netzwerktechnologie haben auf der anderen Seite dazu geführt, dass man heute von jedem beliebigen Punkt der Erde aus auf seine Daten zugreifen kann, ganz so als sitze man real vor dem Rechner. Man benötigt also nicht immer und überall einen leistungsfähigen Computer, sondern nur noch den Zugang zu einem solchen über das Internet. Durch Techniken wie Wireless LAN, GSM oder UMTS ist ein Netzzugang heute praktisch von überall und mit jedem Gerät möglich. Wenn der Server über genügend Leistung verfügt, und eine ordentliche Anbindung an das Internet hat, benötigt ein solches "Netzzugangsgerät" eigentlich nicht mehr als einen Bildschirm, eine Tastatur sowie Zugang zum Internet. Der Server übernimmt sämtliche Funktionen eines klassischen Rechners, lediglich Ein- und Ausgaben werden zwischen Server und Client übermittelt.

Grundlagen

Solche Netzzugangsgeräte werden aufgrund ihrer minimalen Funktionalität oder der Tatsache, dass sie ohne ein Netzwerk nicht funktionieren, oft Thin-Clients oder Netzwerk-Computer (NC) genannt. Ein NC muss jedoch kein Computer im herkömmlichen Sinne sein. Jedes Gerät, welches über einen Netzanschluss sowie Möglichkeiten zur Ein- und Ausgabe verfügt, ist laut Definition bereits ein Thin-Client. Es könnte sich im Prinzip auch um eine Waschmaschine oder eine Mikrowelle handeln. Deshalb sind die Anwendungsgebiete für NCs auch nicht auf die klassische Datenverarbeitung begrenzt, auch Meß- oder Steueraufgaben sind denkbar. Ein Kühlschrank könnte zum Beispiel Nahrung nachbestellen, die zur Neige geht oder Haushaltsgeräte melden sich automatisch, wenn Wartungsintervalle anstehen. Da im Prinzip jedes Gerät immer und von überall aus über das Internet steuerbar ist und heutzutage selbst in Waschmaschinen leistungsfähige Prozessoren stecken, spricht man auch gerne von ubiquitous computing, dem allgegenwärtigen Rechner.

Heutzutage werden Thin Clients aber hauptsächlich in mittleren und großen Firmen eingesetzt. Die Hauptgründe für den Einsatz solcher Systeme liegen vor allem in der hohen Kostenersparnis. Die meisten Aufgaben im beruflichen Unfeld lassen sich genausogut mit Thin-Clients lösen. Man erspart sich aber vor allem finanziellen Aufwand im Bereich der Sicherheit und der Pflege der Rechner. Man muss nur noch die Server sichern und betreuen. Mit den Clients ist nahezu kein Aufwand verbunden. Die Clients lassen sich auch wesentlich schwerer kompromittieren als normale Workstation Rechner. Dies ist vor allem in großen Firmen, wo die vollständige Überwachung nur schwer möglich ist, ein Vorteil.

Historisches

Die Firma Oracle [ORA] hat das Konzept der Netzwerk Computer bereits 1995 vorgestellt. Im Gegensatz zu Heute war damals der Preisunterschied zwischen normalen Rechnern und den Thin Clients noch recht hoch. (ca. 500 - 1000$). Oracle ging es in erster Linie darum eine bes-

sere Marktposition und höhere Verkaufszahlen für seine Datenbankprodukte zu erreichen. Oracle erkannte, dass man bei den meisten Datenbankanwendungen eigentlich auf einen vollwertigen Rechner verzichten konnte, da der Großteil der Arbeit ohnehin auf dem Server stattfand. Oracle lieferte aber lediglich die Spezifikation, wie ein NC zu funktionieren hatte, und überließ anderen Herstellern die Ausarbeitung entsprechender Lösungen. Jeder konnte die Spezifikation lizenzieren.

Ein NC besteht laut Oracle aus einem modemgroßen Gehäuse, VGA-Graphik, Unix-artigem Betriebssystem, leistungsfähiger CPU und Netzwerkanschluss. Außerdem sollte ein NC mindestens einen Java-fähigen Browser sowie einige grundlegenden Internetprogramme enthalten. Oracle hat die aktive Vermarktung von NCs jedoch im Jahr 2000 aufgegeben.

Für die Firma Microsoft stellte das Konzept des NCs jedoch ein Problem dar, war diese Idee doch eine Bedrohung für ihre marktbeherrschende Stellung im Bereich der Desktop-Betriebssysteme. Es dauerte auch nicht lange, bis es ein ähnliches System auch für Microsoft Betriebssysteme gab. Microsoft lizenzierte zunächst ein Produkt der Firma Citrix und verkauften es in Verbindung mit Windows NT 3.51 als Windows Terminal Server Edition. Das Produkt, welches von Citrix MetaFrame genannt wurde, basiert auf dem X11-Protokoll. Es hat jedoch einige Erweiterungen erfahren, wie zum Beispiel die Möglichkeit Audioinformationen zwischen Client und Server zu übertragen.Citrix hat jedoch nicht nur Microsoft als Lizenznehmer gewonnen, sondern auch andere namhafte Firmen, wie Sun oder IBM.

Erst mit der Einführung von Windows NT 4.0 im Jahre 1996 stellte Microsoft eine Eigenentwicklung vor. Der Windows NT 4.0 Server, in der Terminal Server Edition, setzt auf das Remote Desktop Protokoll (RDP) RDP bietet fast die gleiche Funktionalität wie das ICA Protokoll der Firma Citrix. Es basiert auf einer Gruppe von Standards, welche von der International Telecommunications Union, in Zusammenarbeit mit Microsoft, entwickelt worden waren. RDP ist mittlerweile in die komplette Produktpalette von Microsoft integriert.

2.2.1 Ausblick

Nahezu jeder Hardwarehersteller bietet heutzutage eine Lösung auf Basis von Thin-Clients an. Die prominentesten Vertreter sind Sun mit ihrer SunRay Reihe und IBM mit der NetVista Serie. Die Betriebssysteme, die auf den Clients eingesetzt werden, sind fast immer Eigenentwicklungen. Dies ist nicht besonders problematisch, da fast alle Systeme im Bundle mit entsprechenden Software- und Serverlösungen verkauft werden. Im Hinblick auf die zunehmende Vernetzung der Welt und die Möglichkeiten die sich durch Thin-Clients eröffnen, ist das allerdings nicht die optimale Lösung. Hier muss an einheitlichen Standards gearbeitet werden, die es auch anderen Anbietern ermöglichen Anwendungen für solche Plattformen zu entwickeln. Ein weiteres Problem stellt die Skalierbarkeit der eingesetzten Betriebssysteme dar. Da es eine Vielfalt an Rechnern mit völlig unterschiedlicher Architektur und Leistungsfähigkeit gibt und man nicht für jede Plattform ein eigenes Betriebssystem entwickeln kann, muss man ein System finden, das sich leicht anpassen lässt und selbst auf Rechnern mit geringer Systemleistung noch zufriedenstellend funktioniert. Das JX Betriebssystem zeigt eine Möglichkeit diese Konzepte zu verwirklichen.

2.3 Netzwerkmodelle

Um die Komplexität von Netzwerkprotokollen zu vermindern, werden diese oft in Schichten oder Ebenen aufgeteilt. Jede Ebene hat dabei eine ganz spezielle Aufgabe. Schicht n eines Rechners kommuniziert dabei immer mit Schicht n eines anderen Rechners.die Anzahl oder die Aufgaben der Schichten können sich aber von Modell zu Modellunterscheiden.

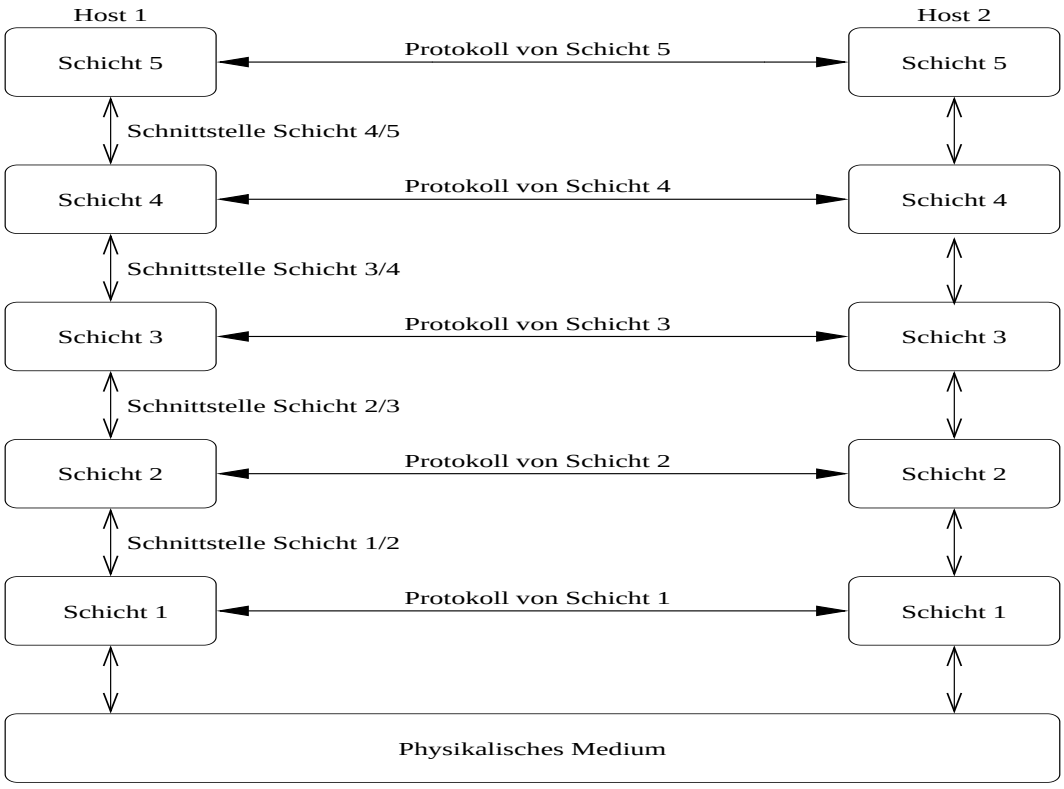


Abb. 2.2 Schichten, Protokolle und Schnittstellen (aus [TAN00])

Die Regeln, die den Ablauf einer Kommunikation bestimmen, nennt man Protokoll. Die Kommunikation geschieht aber nicht direkt, sondern jede Schicht stellt der nächsthöheren Schicht bestimmte Dienste zur Verfügung und nimmt wiederum Dienste der darunterliegenden Schicht in Anspruch. So wird eine Anfrage durch alle Ebenen nach unten gereicht bevor sie über das Netz gesendet wird. Am Zielort wird sie dann wieder durch alle Schichten nach oben gereicht, bis sie ihr Ziel erreicht hat.

Die konkrete Implementierung einer Schicht kann sich ändern ohne das die davon abhängigen Schichten etwas merken, da eine Ebene mit der darunterliegenden Ebene nur über spezielle Schnittstellen interagieren kann. Die Kommunikation geschieht dabei nachrichtenorientiert, auch wenn es sich für den Anwender so darstellt als gäbe es einen kontinuierlichen Strom an Informationen. Die Nachrichten werden in einzelne Datagramme oder Pakete aufgeteilt und getrennt übertragen. Jede Schicht hängt an die eigentlichen Nutzdaten die für die Verarbeitung nötigen Verwaltungsinformationen an. Ein Paket einer Schicht besteht also aus dem Datenbereich und einem Kopfteil. In der Regel enthalten diese Absender- und Zieladresse, sowie Informatio-

nen zur Interpretation der Nutzdaten. Eine bestimmte Sammlung von Protokollen, Diensten und Schnittstellen nennt man auch Referenzmodell. Die beiden bekanntesten sind das Open Systems Interconnection (OSI) Referenzmodell der International Standards Organization (ISO) sowie das Transmission-Control-Protocol/Internet-Protocol (TCP/IP)

2.3.1 Das ISO/OSI-Referenzmodell

Das OSI-Referenzmodell basiert auf einem Entwurf der International Standards Organization. Das System definiert einen Standard zur Verbindung offener Systeme über ein gemeinsames Netzwerk. Offen bedeutet dabei das die Systeme offen für die Kommunikation mit anderen Systemen sind.

Das OSI-Modell besteht aus sieben Schichten die hierarchisch angeordnet sind. Von unten nach oben sind dies:

(1) Die Bitübertragungsschicht (Physical Layer)

(2) Die Sicherungsschicht (Data Link Layer)

(3) Die Vermittlungsschicht (Network Layer)

(4) Die Transportschicht (Transport Layer)

(5) Die Sitzungsschicht

(6) Die Darstellungsschicht

(7) Die Verarbeitungsschicht

Folgende Prinzipien wurden beim Entwurf der OSI-Architektur angewendet ([TAN00]):

- Eine neue Schicht entsteht dort, wo ein neuer Abstraktionsgrad benötigt wird.
- Jede Schicht erfüllt eine genau definierte Funktion.
- Bei der Wahl der Funktion sollte die Definition international genormter Protokolle berücksichtigt werden
- Die Grenzen zwischen den einzelnen Schichten wurden so gewählt, dass der Informationsfluss zwischen den Schichten möglichst gering ist.

- Die Anzahl der Schichten sollte so groß sein, dass nicht mehrere Funktionen in eine Schicht gepackt werden müssen. Allerdings nicht so groß, dass die Architektur unhandlich wird.

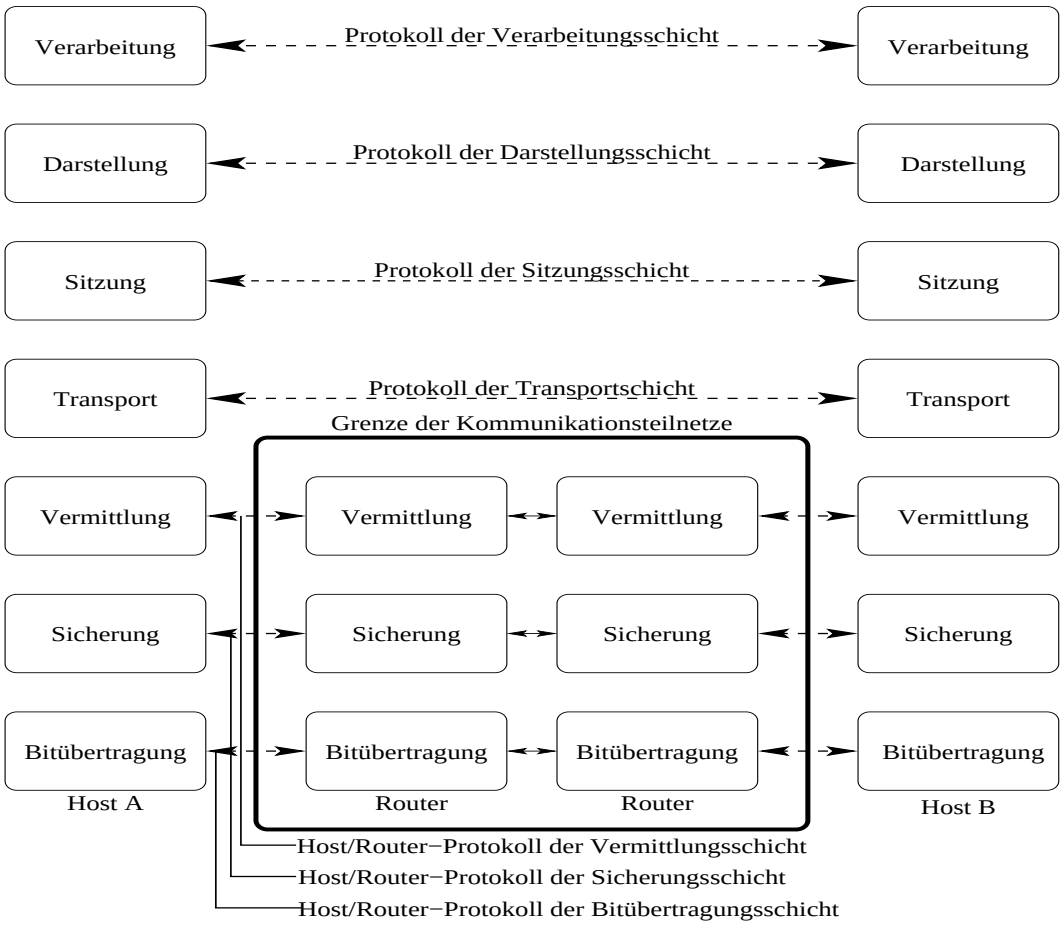


Abb. 2.3 Das ISO/OSI-Referenzmodell (aus [TAN00])

Die Bitübertragungsschicht

Die Bitübertragungsschicht ist für die physikalische Übermittlung der einzelnen Bits über einen Kanal zuständig. Also Dinge wie: Welche Stecker werden verwendet? Welche Arten von Kabel sind zulässig? Wie sieht die elektrische Repräsentation der einzelnen Bits aus? Die Bitübertragungsschicht kümmert sich hauptsächlich um die elektrischen und mechanischen Aspekte der Verbindung.

Die Sicherungsschicht

Die Sicherungsschicht verwandelt den rohen Übertragungskanal erst in eine richtige Leitung. Sie erkennt, eventuell bei der Übertragung aufgetretene, Fehler und korrigiert sie. Außerdem sorgt sie dafür, dass der Kommunikationskanal nicht mit zu vielen Nachrichten überflutet wird, so dass es dem Zielrechner unmöglich ist sie alle zu verarbeiten. Die Sicherungsschicht steuert zusätzlich noch den Zugriff auf das Medium.

Die Vermittlungsschicht

Die Vermittlungsschicht ist für alle Aspekte des Routings zuständig. Dazu gehört die Suche nach dem richtigen Weg zum Zielrechner, oder die Anpassung des Paketformates beim Übergang in ein anderes Netz. Ihre Aufgaben umfassen aber auch Kostenabrechnung, oder die Vermeidung von Engpässen auf der Leitung

Die Transportschicht

Die Transportschicht stellt eine Ende-zu-Ende Verbindung zwischen zwei oder mehreren Kommunikationspartnern bereit. Sie kümmert sich dabei um Dinge wie den Verbindungsauf- oder Abbau und die Aufteilung und das Zusammensetzen von Nachrichten, falls eine Nachricht länger sein sollte, als es der Kommunikationskanal zulässt. Die Transportschicht stellt dabei auch sicher, dass Nachrichten in der selben Reihenfolge ankommen, in der sie auch gesendet wurden.

Die Sitzungsschicht

Die Sitzungsschicht stellt prinzipiell die selben Dienste zur Verfügung, wie die Transportschicht, erweitert diese jedoch um einige Aspekte, die für ganz bestimmte Anwendungen nützlich sind. Die Sitzungsschicht fasst einen oder mehrere zusammengehörende Kanäle zu einer Verbindung, einer sogenannten Sitzung, zusammen und stellt dem Benutzer einfache Möglichkeiten zur Verwaltung dieser Sitzungen zur Verfügung.

Die Darstellungsschicht

Aufgabe der Darstellungsschicht ist es, die zu übertragenden Daten in einer eindeutigen und standardisierten Art und Weise zu kodieren. Im Gegensatz zu anderen Ebenen, die sich nur um die Übertragung einzelner Bits kümmern, sorgt sich die Darstellunsschicht auch um die Syntax und Semantik der Nachrichten. Dabei steht vor allem die Umsetzung in ein Standardformat für die Übertragung über ein Netzwerk im Vordergrund. Die Repräsentation von Daten kann von Plattform zu Plattform variieren und muss deshalb vor der Übermittlung, in ein allen Verständliches Format, umgewandelt werden.

Die Verarbeitungsschicht

Die Verarbeitungsschicht dient vor allem der Anwendungsunterstützung. Sie umfasst eine Vielzahl von Protokollen für die unterschiedlichsten Anwendungsgebiete. Zur Verarbeitungsschicht gehören Protokolle wie FTP, POP3, NFS und noch viele mehr.

Obwohl der ISO/OSI Standard bereits seit fast zwanzig Jahren existiert, konnte er sich doch nie wirklich durchsetzen. In der Praxis hat sich gezeigt das eine Implementierung auf Basis des OSI-Modells zu aufwendig ist. Die hohe Komplexität geht außerdem zu Lasten der Geschwindigkeit und Leistungsfähigkeit.

2.3.2 Das TCP/IP Referenzmodell

Der Name TCP/IP stammt von den beiden Hauptprotokollen, Transmission Control Protocol und Internet Protocol, ab. Das TCP/IP-Referenzmodell entstand im Rahmen eines Projektes des amerikanischen Verteidigungsministeriums (Department of Defense, DOD). Ziel dieses Projektes war der Aufbau eines landesweiten Kommunikationsnetzes, mit dessen Hilfe militärische und zivile Einrichtungen, auch im Falle eines Krieges, Daten austauschen können. Hauptaugenmerk wurde dabei vor allem auf die Fehlertoleranz gelegt. Kommunikation sollte auch dann noch möglich sein, wenn Teile der Kommunikationsinfrastruktur ausgefallen oder beschädigt worden sind. Diese Ziele wurden im sogenannten ARPANET verwirklich aus dem das heutige Internet hervorgegangen ist.

Das TCP/IP-Referenzmodell ist der OSI-Architektur sehr ähnlich. Beiden Architekturen liegt ein Schichtenmodell zugrunde und selbst die Funktionaliät der Schichten ähnelt sich stark.

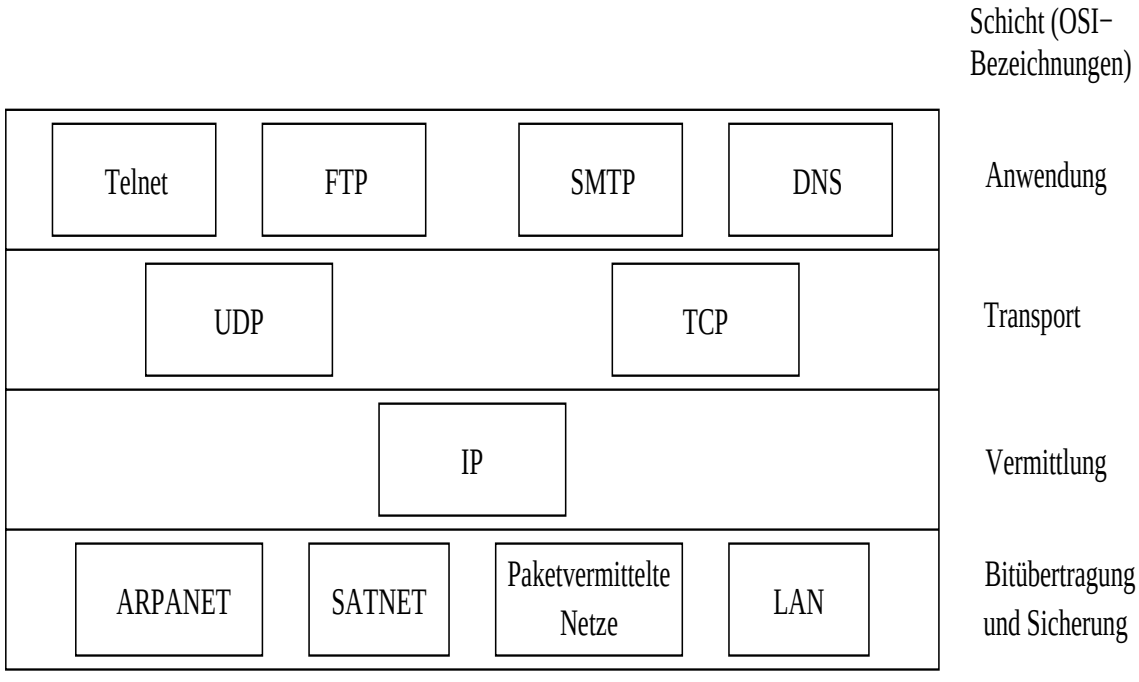


Abb. 2.4 Das TCP/IP-Referenzmodell (aus [TAN00])

Im TCP/IP-Modell fehlt allerdings sowohl die Sitzungs- als auch die Darstellungsschicht. Das TCP/IP-Modell unterschied ursprünglich nicht genau zwischen Protokollen, Diensten und Schichten, im Gegensatz zum OSI-Modell Das hat damit zu tun, dass bei der Entwicklung von TCP/IP die Entwurf die Anwendungen an erster Stelle standen, eine klare Trennung erst an Zweiter. Die in Abbildung 2.4 gezeigte Aufteilung wurde erst im Laufe der Zeit entwickelt. Im Gegensatz dazu wurde beim OSI-Modell zuerst die logische Aufteilung in Dienste, Schnittstellen und Protokolle entworfen. Diese sollte möglichst allgemein, und unabhängig von irgendwelchen konkreten Implementierungen, sein.

Im TCP/IP Referenzmodell gibt es also nur noch vier Schichten. Diese sollen hier kurz vorgestellt werden.

Die Host-an Netz Schicht

Das TCP/IP Referenzmodell sagt im Grunde nichts über die physikalischen Schichten aus. Dies hat dazu geführt, dass sich eine Reihe unterschiedlicher Medienzugriffsverfahren entwickelt haben. Die bekanntesten sind das Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Verfahren und das Token-Ring/Token-Bus Verfahren. Erst später wurde eine Aufteilung in eine hardwareabhängige Teilschicht, die sogenannte MAC-Schicht und eine hardwareunabhängige Teilschicht, der Logical Link Control Schicht getroffen. Diese Aufteilung ist auch im ISO/OSI-Modell zu finden.

Die Internet Schicht

Die Internet-Schicht erbringt einen paketorientierten verbindungslosen Dienst mit dem es möglich ist Daten in jedes beliebige andere Netz zu schicken. Da der Dienst verbindungslos ist, kann jedes Paket theoretisch einen anderen Weg zum Ziel nehmen. Es ist allerdings nicht festgelegt in welcher Reihenfolge die Pakete das Ziel erreichen. Außerdem stellt die Internet-Schicht nicht sicher das auch wirklich alle Pakete ihr Ziel erreichen. Diese Aufgaben fallen der Transportschicht zu. Die Internet Schicht definiert ein spezielles Paketformat und ein Protokoll namens IP. (Internet Protocol)

Die Transportschicht

Die Transportschicht im TCP/IP-Modell hat mehrere Aufgaben. Zum einen stellt sie einen zuverlässigen verbindungsorientierten Dienst namens TCP bereit, der mit Flusskontrolle und Fehlerkorrektur arbeitet. Zum anderen gibt es aber mit UDP auch einen verbindungslosen Datagrammdienst für Anwendungen, welche den zusätzliche Overhead des verbindungsorientierten Protokolls lieber vermeiden wollen. Bei TCP stellt sich dem Benutzer die Kommunikation so dar als ob es sich um einen kontinuierlichen Datenstrom handelte. Die Transportschicht teilt den Datenstrom in einzelne Nachrichten und setzt diese am Zielort wieder zusammen, ohne das es

die darüberliegende Schicht merkt. Das UDP Protokoll kennt keine Verbindungen, jedes Datagramm wird einzeln und unbestätigt übertragen. Dieser Dienst bietet sich vor allem in Situationen an, wo es mehr auf schnelle Übertragung, als auf eine zuverlässige Verbindung, ankommt.

Die Verarbeitungsschicht

Im TCP/IP Modell gibt es keine Sitzungs- oder Darstellungsschicht. Über der Transportschicht befindet sich die Verarbeitungsschicht. Sie umfasst viele unterschiedliche Protokolle. Die bekanntesten sind wohl Telnet, das File Transfer Protocol (FTP) sowie das Simple Mail Transport Protocol (SMTP)

2.4 Das Betriebssystem JX

Im Gegensatz zu anderen Gebieten der Informatik haben sich Betriebssysteme in den letzten Jahren kaum verändert. In der Regel kommt ein mehr oder weniger monolithischer Kern zum Einsatz, der den Zugriff auf die Hardware und die Verwaltungsstrukturen des Betriebssystems regelt. Er übernimmt außerdem die Ressourcenzuteilung (CPU-Zeit, Plattenzugriffe etc.). Der Kern sowie alle Anwendungen laufen in ihrem eigenen virtuellen Adreßraum ab. Die Trennung der Speicherbereiche wird durch spezielle Hardware, genannt Memory Management Unit, gewährleistet.

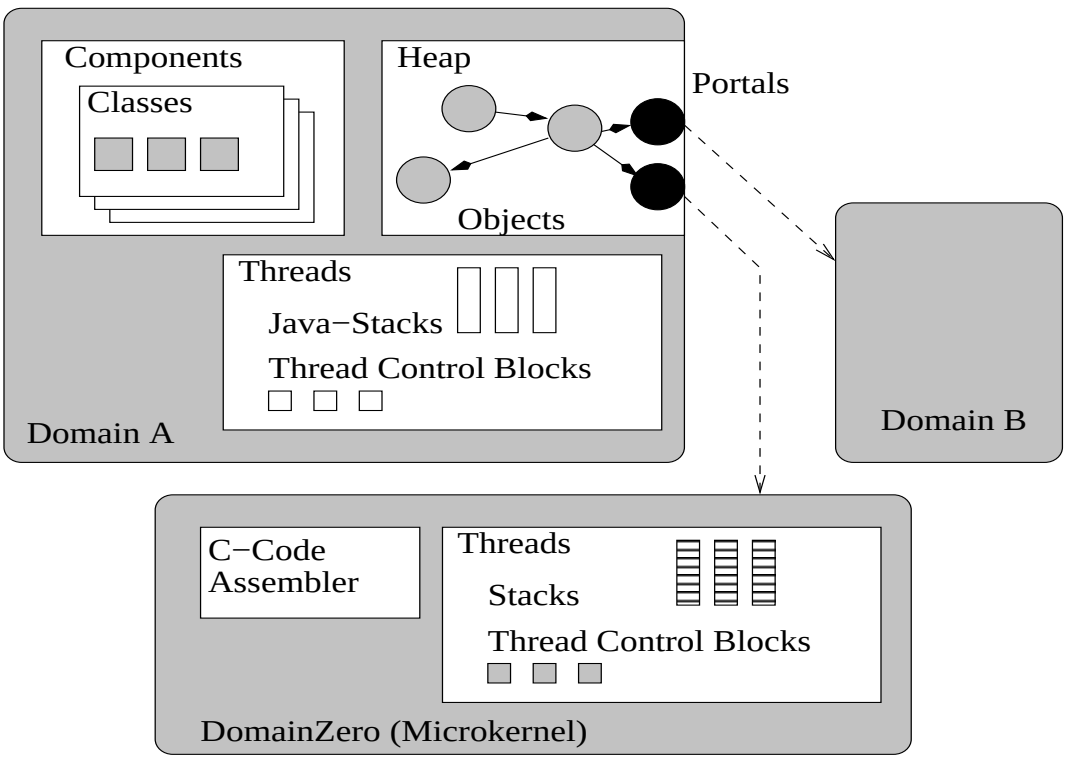


Abb. 2.5 Aufbau des JX-Betriebssystems (aus [GFW02])

Diese Trennung ist notwendig, weil die zur Programmierung des Systems verwendeten Sprachen den unkontrollierten Zugriff auf die Hardware sowie auf dem Hauptspeicher des Rechners zulassen. Der entscheidende Nachteil der hardwarebasierten Lösung besteht jedoch darin, dass man die Schutzbereiche nicht fein genug oder flexibel genug auswählen kann. Dies muss bei der Implementierung berücksichtigt werden und führt zu einer Erhöhung der Komplexität des Kerns. Außerdem entstehen so starke Abhängigkeiten zwischen verschiedenen Teilen des Betriebssystemkerns, welche die Erweiterung erschweren. Zusätzlich wird es dadurch schwieriger Funktionen aus dem Kern in eigene Module auszulagern.

Die starken Abhängigkeiten, welche aufgrund des hardwarebasierten Schutzes in heutigen Betriebssystemkernen existieren, erschweren es außerdem den Kern optimal auf spezielle Anwendungsgebiete anzupassen. Nicht jeder Rechner ist mit Hauptspeicher und Rechenkapazität im Überfluss ausgestattet. Außerdem wäre es sehr kostenintensiv für jeden speziellen Anwendungsfall ein eigenes Betriebssystem zu entwickeln. Ein ideales Betriebssystem muss also flexibel genug sein um sowohl auf einem Kleinstcomputer als auch auf einem großen Rechner laufen zu können. Außerdem stellt es eine einheitliche Programmierschnittstelle zur Verfügung, so dass Anwendungsprogramme ohne Anpassungen auf allen Architekturen lauffähig sind. Da heutige Computersysteme fast ausschließlich in Netzwerken eingesetzt werden, ist außerdem ein gutes Sicherheitskonzept notwendig. Das Betriebssystem JX wurde entwickelt um einige dieser Konzepte zu verwirklichen.

Das Betriebssystem JX setzt ein softwarebasiertes Schutzkonzept ein. Die einzelnen Adreßräume werden nicht mehr hardwareseitig voneinander getrennt, sondern dies wird durch die Verwendung einer inhärent sicheren Sprache gewährleistet. JX verwendet Java als Basis. Außerdem ist so höchstmögliche Portabilität der Anwendungsprogramme sicher gestellt.

Ein großer Teil von JX ist in Java geschrieben. Es gibt jedoch einige Dinge die sich nicht auf der Java-Ebene lösen lassen. Dazu gehört der Hardwarezugriff, Routinen zur Systeminitialisierung sowie das Low-Level Domain-Management. Abbildung 2.5 zeigt den prinzipiellen Aufbau des JX-Systems. Der Java-Programmcode ist in einzelne Komponenten aufgeteilt, welche verifiziert und in einzelne Domains geladen werden. Danach werden sie in nativen Code übersetzt. Die Kommunikation zwischen einzelnen Domains findet nur über Portale statt.

2.4.1 Domains

JX ist in sogenannte Domains aufgeteilt. Eine Domain fasst bestimmte zusammengehörende Teile des Systems zu einer Einheit zusammen. Sie stellt außerdem auch die Möglichkeit des Resourcenmanagements bereit und implementiert außerdem Schutzkonzepte. Dies steht im Gegensatz zu klassischen Betriebssystemen für Monoprozessoren, wo sich alle Teile des Betriebssystems einen Adreßraum teilen. Die Teile des JX Systems, die nicht in Java geschrieben sind oder welche besonders essentielle Systemdienste bereitstellen, müssen vertrauenswürdig sein. Dazu gehört der Kern, da dieser ohne hardwarebasierten Schutz läuft. Allerdings gehören auch einige Java-Komponenten z.B. der Code-Verifizierer, der Übersetzer sowie einige hardwareabhängige Komponenten, dazu. Diese Elemente bilden die sogenannte Domain Zero und stellen die minimale vertrauenswürdige Basis dar, die Trusted Computing Base.

Jede Domain hat ihren eigenen Heap und ihren eigenen Garbage Collector. Jeder GC kann außerdem seinen eigenen Sammelalgorithmus implementieren. Jede Domain kann so auf ihren speziellen Anwendungszweck hin optimiert werden. Jede Domain hat ihre eigenen Threads, die nicht zwischen einzelnen Domains migrieren können. Domains können sich Programmcode mit anderen Domains teilen.

2.4.2 Portale

Die Kommunikation zwischen Domains findet ausschließlich über spezielle Schnittstellen, sogenannte Portale, statt. Ein Portalaufruf funktioniert ähnlich wie eine remote method invocation (RMI) unter Java und stellt eine Art RPC-Mechanismus bereit, mit der einzelne Portale miteinander kommunizieren können. Die Ähnlichkeit mit dem RMI Mechanismus erlaubt es Java-Programmierern außerdem ohne große Anpassungen Java Code auf JX zu portieren.

Ein Objekt, welches über ein Portal angesprochen werden kann, heißt Service. Ein Service besteht aus einem ganz normalen Objekt, welches das Portal Interface implementieren muss und einem dazugehörenden Service-Thread. Ein Service wird über ein Portal aufgerufen, der aufrufende Thread blockiert sich und der Service Thread wird ausgeführt. Ein Portal stellt also ein Proxy-Objekt dar. Eine Domain, die einen bestimmten Service anbieten will, kann diesen bei einem Name-Server registrieren.

2.4.3 Speicherzugriff

Die Datenstrukturen der Java-Programmiersprache, wie zum Beispiel Arrays, sind für die Darstellung eines physikalischen Arbeitsspeichers ungeeignet. Arrays werden nicht über Methoden angesprochen, erlauben nicht den Nur-Lesen Zugriff und ermöglichen außerdem auch nicht die Bildung von Unterbereichen. Dies sind alles Dinge, die für das Speichermanagement zwingend notwendig sind.

In JX existieren deshalb sogenannte Memory-Objekte. Memory-Objekte werden vom Programmierer genauso angesprochen wie normale Objekte, aber sie werden vom Übersetzer anders behandelt. Der Übersetzer ersetzt die Methodenaufrufe durch Assemblerbefehle für den Speicherzugriff. Das macht den Speicherzugriff in JX so effizient wie den Zugriff auf Arrays in Java.

Memory-Objekte können als Portale an andere Domains übergeben werden und stellen so gewissermaßen Shared Memory bereit. Es ist außerdem möglich den Zugriff auf den Speicher zu begrenzen. Zu diesem Zweck gibt es Read-Only Memory, der nur das Lesen des Speichers erlaubt. Ein Objekt vom Typ Read-Only Memory kann nicht in normalen Speicher umgewandelt werden. Um den Zugriff auf Hardwarekomponenten zu regeln, existiert noch sogenanntes Device Memory. Da es sich bei Device Memory in der Regel um spezielle Hardwareregister handelt wird bei der Freigabe des Memory-Objektes der dazugehörige Speicherbereich nicht freigegeben.

JX verzichtet auf hardwarebasierten Schutz, wie ihn eine MMU bietet. Das macht JX ganz besonders geeignet um auf kleinen leistungsschwachen Prozessoren zu laufen, die in der Regel nicht über eine MMU verfügen. Dazu gehören speziell: Thin-Clients, Mobiltelefone oder Persönliche digitale Assistenten. Allerdings muss JX einige Mechanismen, welche normalerweise auch von der MMU bereitgestellt werden, selbst in Software implementieren.

2.4.4 Verifizierer und Übersetzer

Der Programm-Verifizierer ist ein besonders wichtiger Teil des JX Systems. Der gesamte Bytecode wird vor der Übersetzung überprüft. Zuallererst wird eine normale Bytecode-Überprüfung vorgenommen. Der JX-Verifier bestimmt danach allerdings noch Obergrenzen für die Ausführungszeiten von Interrupthandlern und Scheduler-Methoden. Der Translator ist für die Übersetzung des Bytecodes in die Maschinensprache zuständig. Im Falle von JX handelt es sich dabei um X86-Code. Maschinencode kann in jeder Domain erzeugt werden, aber nur wenn er in Domain Zero erzeugt wurde, kann er von anderen Domains genutzt werden.

2.4.5 Das Komponentensystem

Der gesamte Java-Code, der in eine Domain geladen wird, ist in Komponenten organisiert. Eine Komponente besteht aus den dazugehörigen Klassen und Interfaces und enthält außerdem noch alle notwendigen Zusatzinformationen, wie zum Beispiel Abhängigkeiten von anderen Komponenten. In JX sind alle Teile des Systems als Komponenten realisiert, sogar das Java Laufzeitsystem. Dies erlaubt die äußerst flexible Anpassung des Gesamtsystems oder einzelner Domains an unterschiedliche Systemanforderungen. So wäre zum Beispiel denkbar, den Scheduler an die Anwendung anzupassen (Echtzeitanforderungen oder hohe Prozessorauslastung) oder unterschiedliche GC Algorithmen zu implementieren. JX ist dadurch optimal für eine große Anzahl an Anwendungsgebieten geeignet.

2.4.6 Fazit

Die zugrundeliegenden Konzepte machen JX zu einem interessanten Betriebssystem für die unterschiedlichsten Anwendungsgebiete. Die erhöhte Flexibilität im Vergleich zu konventionellen Betriebssystemen, macht JX ganz besonders für Thin-Clients interessant. Die Einbindung von Java als Programmiersprache macht die Anwendungsentwicklung für JX sehr einfach und sorgt außerdem dafür, dass die Programme auf allen Systemen laufähig sind. Performancemessungen haben gezeigt, dass das JX System bereits jetzt, je nach Anwendungsgebiet, 50 - 100% der Performance eines X86 -Linux Systems erreicht (gemessen auf der gleichen Hardware). Dies lässt sich durch weitere Optimierung, vor allem des Übersetzers, noch weiter verbessern.

2.5 Zusammenfassung

Dieses Kapitel gab einen umfassenden Überblick über die grundlegenden Konzepte und Prinzipien, die für diese Studienarbeit von Bedeutung waren. Nach einigen einleitenden Bemerkungen wurde zunächst das Konzept des Thin-Clients näher erläutert und ein Überblick über die heute verfügbaren Technologien gegeben. Danach folgte eine Zusammenfassung der beiden gängigen Netzwerkmodelle, dem TCP/IP- sowie dem ISO/OSI-Referenzmodell. Abgeschlossen wurde dieses Kapitel mit einer kurzen Beschreibung des Java-basierten Betriebssystems JX, das die Grundlage für diese Studienarbeit bildet.

3 Das Remote Desktop Protokoll

3.1 Einleitung

Dieses Kapitel beschäftigt sich mit dem Aufbau und der prinzipiellen Funktionsweise des Remote Desktop Protokolls (RDP). Zuerst wird der grundlegende Aufbau des Remote Desktop Protokolls näher beschrieben. Danach werden die einzelnen Schichten der Protokollhierarchie detailliert erläutert, angefangen bei der ISO-Adaptionsschicht über die MCS- und die Sicherheitsschicht bis hin zur eigentlichen RDP-Schicht.

3.2 Der Aufbau des Remote-Desktop Protokolls

Eine Aufgabe bei der Entwicklung des Windows NT Terminal Servers war es, das von Citrix lizenzierte und damit kostenpflichtige, ICA-Protokoll durch eine Eigenentwicklung zu ersetzen. Diese Eigenentwicklung, die den Namen Remote Desktop Protokoll erhielt, ist aus einer Zusammenarbeit von Microsoft mit der International Telecommunication Union (ITU) entstanden. Es basiert auf den ITU Standards T.125 Multipoint Communication Service - Protocol Specification (MCS [ITU98b]) und T.128 Application Sharing [ITU98c]. Diese beiden Standards bauen ursprünglich auf dem ISO/OSI-Referenzmodell auf und nutzen zur Übertragung die ISO-Transportschicht, die in [ISO84] spezifiziert wurde

Fig. 3.1 Aufbau des RDP-Protokolls

Mittlerweile hat sich jedoch TCP/IP als Standard zur Datenübertragung durchgesetzt. Deshalb musste das MCS-Protokoll so angepasst werden, dass es auch TCP/IP als Transportschicht verwenden konnte. Dies wurde erreicht, indem man eine Adaptionsschicht eingefügt hat, welche die Schnittstellen der ISO-Transportschicht auf Schnittstellen der TCP-Schicht abbildet. Das

genaue Verfahren ist im Request for Comments 2126 ISO Transport on Top of TCP/IP [PY97] näher beschrieben. Microsoft selbst hat einige Modifikationen an den Protokollen vorgenommen, es existiert darüber aber so gut wie keine Dokumentation. Einige Freiwillige haben sich jedoch die Mühe gemacht, das RDP-Protokoll zu analysieren. Viele in dieser Studienarbeit gewonnenen Erkenntnisse basieren auf Rdesktop [RDE], einem RDP-Client Programm für GNU/ Linux, welches von Matt Chapman entwickelt worden ist.

Das RDP-Protokoll besteht aus vier Schichten, der ISO-Adaptionsschicht, der MCS-Schicht, der Sicherheitsschicht und der RDP-Schicht. (siehe Abbildung 3.1)

3.2.1 Die ISO-Adaptionsschicht

Die einzige Aufgabe der ISO-Adaptionsschicht besteht darin, die Dienstprimitive der ISO-Schicht auf die Dienstprimitive der TCP-Schicht abzubilden. Damit können auch Dienste, welche die ISO-Transportschicht nutzen, leicht auf TCP/IP portiert werden. Leider lassen sich nicht alle Dienste der ISO-Transportschicht auf solche der TCP-Schicht abbilden. Das ISO Protokoll hat viel weitreichendere Spezifikationen getroffen, als es im TCP-Modell der Fall war. Insbesondere handelt es sich dabei um sogenannte Quality-of-Service Dienste. Diese lassen sich nicht auf das TCP-Protokoll übertragen, da es keine Mechanismen für Quality of Service Dienste kennt, sondern lediglich Best-Effort bietet.

3.2.2 Die MCS-Schicht

Die MCS-Schicht stellt sogenannte Multicast Dienste zur Verfügung. Sie unterstützt nicht nur Punkt-zu-Punkt Verbindungen, sondern auch Punkt-zu-Multipunkt Verbindungen. Bei dieser Art der Kommunikation können einem Sender mehrere Empfänger zugeordnet sein. So wird der Kommunikationsaufwand minimiert, falls mehrere Clients die gleichen Daten erhalten sollen. Dies kann zum Beispiel bei Multimediaübertragungen der Fall sein. Da es sich aber nicht um einen Broadcast handelt sind Rechner, die nicht an der Datenübertragung teilnehmen, auch nicht Ziel einer Übertragung. Dies kann das Datenaufkommen das über das Netzwerk versendet wird verringern, wenn das Netz Multicast-fähig ist. Dies ist jedoch bei LANs in der Regel nicht der Fall.

3.2.3 Die Sicherheitsschicht

Die Sicherheitsschicht umfasst alle Dienste, die die Verschlüsselung und Sicherung der Verbindung betreffen. Diese Schicht ist nicht Teil des ITU Standards, sondern eine Eigenentwicklung der Firma Microsoft. Sie dient dazu die Verbindung vor unrechtmäßigen “Mithörern” zu schützen. Dies wird durch die Verwendung des RC4 Algorithmus der RSA Inc. erreicht. Außerdem werden die Nachrichten noch mit einer Kombination aus MD5 und SHA-1 signiert um die Manipulation von Nachrichten auszuschließen. Zuguterletzt ist die Sicherheitschicht auch noch für die Authentifizierung der Teilnehmer zuständig. Im Falle des MS Terminal Servers umfasst das auch das Lizenzmanagement

3.2.4 Die RDP-Schicht

Die RDP-Schicht ist für die eigentliche Kommunikation zwischen Server und Client zuständig. Das verwendete Protokoll ist relativ aufwändig gestaltet. Es erlaubt die Übertragung von Tastatur- oder Mauseingaben sowie die Übermittlung von Bitmaps oder einfachen Zeichenoperationen. Welche Operationen bei einer Sitzung verwendet werden sollen, kann beim Verbindungsaufbau ausgehandelt werden. Außerdem gibt es noch Befehle für die Zwischenspeicherung bestimmter Daten, um die Netzlast so gering wie möglich zu halten. In der Regel sind dies Bitmapdaten, da diese am meisten Speicherplatz verbrauchen. Die Kommunikation verläuft asymmetrisch. Der weitaus größere Teil der Daten wird vom Server zum Client übertragen.

Das RDP-Protokoll hat im Laufe seiner Existenz einige Verbesserungen erfahren. So unterstützt die neueste Version 5.1 neben der reinen Bildübertragung auch noch die Möglichkeit der Audioübermittlung sowie Mechanismen für die Ansteuerung von lokalen Druckern. Außerdem gibt es noch die Möglichkeit, das Windows-eigene Clipboard zu nutzen.

3.3 Die Transportschicht

Die Transportschicht ist die unterste Schicht in der RDP-Protokoll Hierarchie. Sie stellt die grundlegendsten Operationen bereit, die zur Datenübertragung sowie zum Auf- und Abbau einer Verbindung nötig sind. Das Protokoll, welches die Transportschicht zur Übertragung verwendet, basiert auf einer Spezifikation der International Standards Organisation (ISO) und ist in [ISO84] niedergelegt. Es ist eigentlich ein Teil des ISO/OSI-Referenzmodells. Heutzutage wird aber in fast allen größeren Netzen TCP/IP zur Datenübertragung verwendet. Um dennoch ISO-Protokolle weiterhin verwenden zu können wurde eine Möglichkeit entwickelt, wie sich die ISO-Transportschicht auf TCP/IP abbilden lässt. Das Verfahren ist in [PY97] genauer beschrieben.

Die ISO Spezifikation kennt insgesamt fünf verschiedene Klassen von Diensten, mit unterschiedlichen Fähigkeiten:

- Klasse 0: Simple Class
- Klasse 1: Basic error recovery class
- Klasse 2: Multiplexing class
- Klasse 3: Error recovery and multiplexing class
- Klasse 4: Error detection and recovery class

Klasse 0 besitzt nur minimale Funktionalität. Es werden lediglich diejenigen Funktionen bereitgestellt, die für den Verbindungsaufbau und den Datentransport absolut notwendig sind. Eventuell benötigte zusätzliche Funktionalität, wie Flusskontrolle, muss von den darunterliegenden Schichten bereitgestellt werden.

Klasse 1 bietet darüber hinaus noch Fehlertoleranz. Dienste der Klasse 1 können sich von Fehlern erholen, die in der darunterliegenden Schicht auftreten, ohne dass darüberliegende Schichten benachrichtigt werden müssen.

Klasse 2 bietet die Möglichkeit einer expliziten Flußkontrolle unabhängig von anderen Schichten. Außerdem erlaubt die Klasse 2 das Multiplexen von mehreren Transportverbindungen über eine Verbindung der darunterliegenden Netzwerkschicht. Allerdings muss man auf Fehlertoleranzmechanismen verzichten

Klasse 3 bietet alle Dienste der Klasse sowie darüber hinaus noch die Möglichkeit Fehler zu erkennen und eventuell zu beheben.

Klasse 4 schließlich bietet zusätzlich noch Funktionen um verlorengegangene oder duplizierte Pakete sowie Pakete welche außer der Reihe eintreffen zu erkennen und kann geeignete Korrekturmaßnahmen ergreifen.

Das RDP Protokoll nutzt lediglich die Dienste der Klasse 0. Mehr ist auch nicht notwendig, da die zusätzliche Funktionaltität, welche die anderen Klassen bieten würden, bereits durch die darunterliegende TCP-Schicht bereitgestellt wird. Die Klasse 0 stellt jedoch bereits alle nötigen Operationen bereit, die die darüberliegenden Schichten zur korrekten Funktion benötigen.

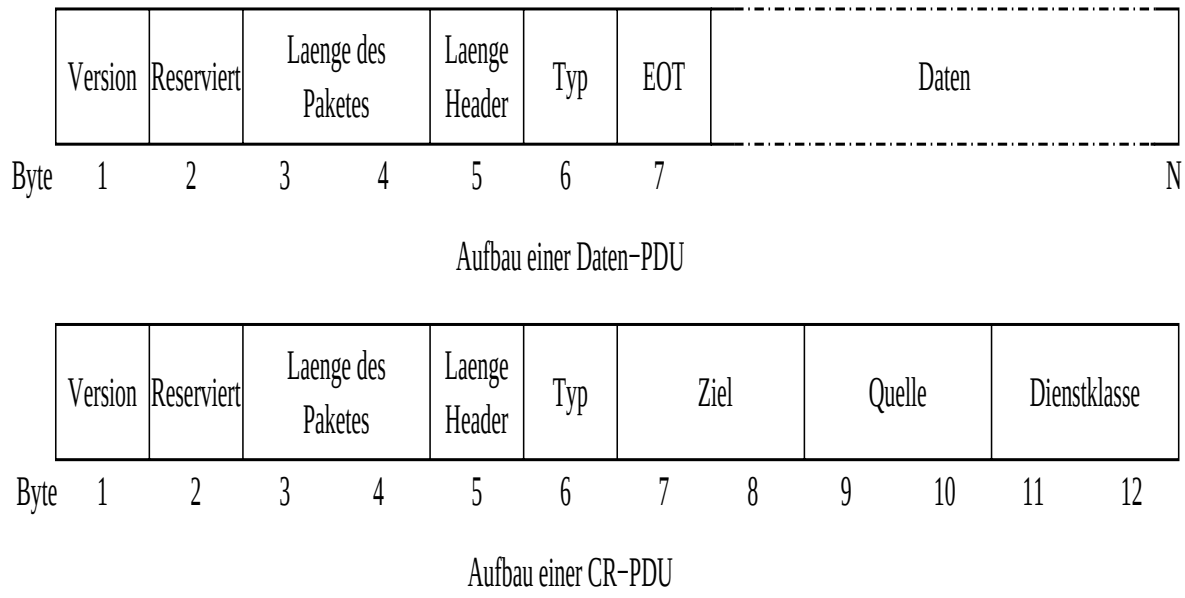


Fig. 3.2 Aufbau einer ISO-PDU

Die Transportschicht kennt nur vier Dienstprimitive. Drei davon dienen dem Verbindungsmanagement, nämlich CONNECTION-REQUEST (CR), CONNECTION-CONFIRM (CC) und DISCONNECT-REQUEST (DR). DATA-TRANSFER (DT) dient dem Austausch von Nutzdaten.

Die Datenpakete, welche beim Auf- und Abbau von Verbindungen ausgetauscht werden, haben alle den gleichen Aufbau. Dieser ist in Abbildung 3.2 beispielhaft anhand des CONNECTION-REQUEST Paketes dargestellt. Jedes Paket ist in einzelne Oktette aufgeteilt, sollte ein Wert

mehr als ein Oktett belegen, so ist das letzte Oktett das “least signifikant byte”. Das Feld Version enthält die Version des Transportprotokolls, die zur Übertragung der Daten verwendet wird. Zum Zeitpunkt als diese Arbeit entstand war das die Version Drei. Das zweite Byte ist als reserviert markiert und wird für zukünftige Erweiterungen des Protokolls freigehalten. Byte drei und vier enthalten die Länge des gesamten Paketes, inklusive des Headers. Die maximale Gesamtlänge eines Datenpaketes ist damit auf 65635 Bytes begrenzt. Byte Fünf enthält die Länge des Headers ohne die ersten vier Byte. Die Länge des Headers kann maximal 255 Bytes betragen. Nur die Klassen zwei bis vier verwenden mehr als zwölf Byte. Byte sechs enthält den Typ des Paketes. Mögliche Typen sind: CR wenn es sich um ein CONNECTION-REQUEST Paket handelt, CC wenn es sich um ein CONNECTION-CONFIRM Paket handelt oder DR wenn es ein DISCONNECT-REQUEST Paket ist. Die Oktette Sieben bis Zehn enthalten die Quell- und Zieladresse, diese wird jedoch im RDP-Protokoll nicht verwendet und ist auf Null voreingestellt. Diese Felder sind offenbar nur aus historischen Gründen erhalten geblieben. Das letzte Oktett ist für die verwendete Dienstklasse bestimmt, in unserem Falle Null.

Ein DATA-TRANSFER Paket ist ähnlich aufgebaut wie die Pakete, die den Verbindungsaufbau regeln. Allerdings verzichtet man auf die Felder für die Quell- und die Zieladresse sowie auf das Byte das die Dienstklasse beschreibt. Außerdem kommt ein Oktett hinzu das die Sequenznummer enthält für den Fall, dass Daten auf mehrere Pakete aufgeteilt worden sind. Da allerdings nur die Dienstklassen zwei bis vier Segmentierung und Reassemblierung unterstützen enthält dieses Feld immer den Code für das Ende einer Übertragung. (End of Transfer EOT).

Sowohl Verbindungsauf- als auch Abbau gehen vom Client aus. Um eine Verbindung aufzubauen sendet der Client ein CONNECTION-REQUEST Paket an den Server, der dieses mit einem CONNECTION-CONFIRM Paket bestätigt. Danach können beliebig viele Daten übertragen werden. Die Verbindung endet wenn der Client ein DISCONNECT-REQUEST Paket an den Server sendet. Das DISCONNECT-REQUEST Paket wird vom Server nicht bestätigt. Das Ende der Verbindung an sich ist die Bestätigung für den Client, dass das Paket angekommen ist. Sollte der Server die Verbindung beenden wollen, etwa weil ein Fehler in der Übertragung aufgetreten ist, wird dies dem Client nicht extra angezeigt. Die Clientseite muss für diesen Fall die entsprechenden Vorkehrungen treffen.

3.4 Die MCS-Schicht

Verbindungen zwischen Rechnern haben in der Regel einen Punkt-zu-Punkt Charakter. Das bedeutet, dass für jede gewünschte Kommunikation zwischen mehreren Rechnern immer eine eigene Verbindung benötigt wird. Dies ist bei vielen Anwendungsgebieten kein Problem, da meist sowieso nur zwei Rechner miteinander kommunizieren wollen. Sollen aber mehrere Empfänger die gleichen Daten erhalten, wie es bei Multimedia-Übertragungen häufig vorkommt, dann ist dieses Verhalten nicht optimal. Bei zehn Empfängern muss ein Sender zehn Verbindungen offenhalten und außerdem zehnmal die gleichen Daten versenden. Das treibt die Netzlast sehr schnell in die Höhe.

Zu diesen Zweck wurden sogenannte Multicast oder Multipoint Protokolle entwickelt. Der Vorteil liegt darin, dass Daten nicht mehr unnötig über das Netzwerk versendet werden müssen. Der Sender verschickt die Daten nur ein einziges mal, statt für jede Verbindung getrennt. Erst auf dem Weg zum Empfänger werden die Daten vervielfältigt. An jedem Netzübergang muss entschieden werden an welche Empfänger die Daten jeweils weitergeleitet werden müssen. Der Vorteil liegt vor allem in einer dramatischen Verringerung der Netzlast. Ohne Multicast-Unterstützung müssen bei mehreren Verbindungen mehrfach die gleichen Daten bereitgestellt werden. Dabei stößt selbst bei einer reinen HIFI-Audioübertragung mit 128 KB/s ein Rechner schnell an Kapazitätsgrenzen.

Der Multipoint Communication Service wurde genau zu den oben genannten Zwecken entwickelt. Er stellt alle notwendigen Mechanismen bereit um viele unterschiedliche Arten von Daten an alle Teilnehmer, oder nur eine begrenzte Anzahl von Teilnehmern, zu senden. Die Anzahl der möglichen Empfänger kann dabei theoretisch beliebig groß sein und muss auch nicht auf ein Netzwerk beschränkt bleiben.

Die MCS-Schicht unterstützt unter anderem

(1) Flexible Möglichkeiten des Datentransfers:
   - Broadcast: Alle Teilnehmer erhalten die gleichen Daten
   - Request/Response: Jeder Client bekommt nur die Daten die er selbst angefordert hat.

(2) Multipunkt Adressierung:
   - Ein Sender an alle
   - Ein Sender an eine Gruppe von Empfängern
   - Ein Sender an einen Empfänger

(3) Multipunkt Routing
   - Kürzester Weg zu jedem Empfänger
   - Gleichbleibende Sequenzierung der Daten, jeder Empfänger erhält die gleichen Daten in der gleichen Anordnung.

Das MCS-Protokoll ist außerdem unabhängig von darunterliegenden Netzwerk. Es wird lediglich davon ausgegangen, dass eine Transportschicht existiert, die fehlerfrei ist und Flußkontrolle anbietet.

3.4.1 Aufbau einer MCS-PDU

Eine MCS-PDU besteht aus zwei Teilen, einem Kopfteil sowie einem Datenteil. Der Header ist relativ einfach strukturiert. MCS-PDUs werden erst versendet, nachdem die MCS-Verbindung aufgebaut worden ist. Der Verbindungsaufbau läuft noch auf der ISO-Ebene ab.

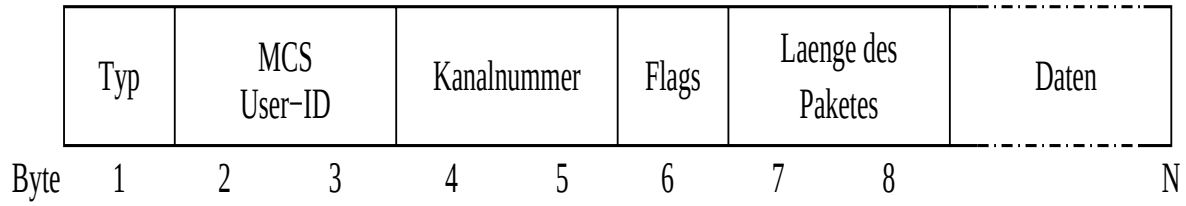


Fig. 3.3 Aufbau einer MCS-PDU

Die Länge des Headers beträgt immer 8 Byte. Das erste Byte des Headers bleibt dem Typ der Nachricht vorbehalten. Da MCS-Datenpakete erst nach erfolgreicher Verbindung versendet werden sind die erlaubten Typen lediglich SEND-DATA-REQUEST bzw. SEND-DATA-INDICATION um das Versenden bzw. den Erhalt von Daten anzuzeigen. DISCONNECT-PROVIDER-ULTIMATUM zeigt das Ende einer Verbindung an. Die nächsten zwei Byte sind für die MCS-USER-ID des Senders bzw. des Empfängers reserviert. Danach folgt die Nummer des Kanals über den die Daten versendet werden sollen bzw. empfangen worden sind. Die letzten drei Byte des Headers enthalten ein Feld für Flags und die Länge des Datenteils der MCS-PDU.

3.4.2 Aufbau eines MCS-Systems

MCS Systeme sind hierarchisch organisiert. In einem MCS Subsystem existieren zwei Arten von Teilnehmern:

(1) MCS Provider:

Ein MCS Provider stellt alle für das MCS Subsystem notwendigen Dienste bereit. Er kommuniziert mit den Anwendungsinstanzen und anderen MCS Providern und ist außerdem für das Domain Management verantwortlich.

(2) MCS User:

Ein MCS User nutzt die Dienste, die ihm vom MCS Provider bereitgestellt werden. Verschiedene MCS User können Daten untereinander entweder über Punkt-zu-Punkt oder über Multicast Verbindungen austauschen, vorrausgesetzt sie befinden sich in der gleichen Domain. Ein MCS User besteht aus einem Controller, der für den Verbindungsauf- und Abbau zuständig ist, sowie einem Client, der diese Verbindung lediglich nutzt.

Ein MCS System ist in Domains aufgeteilt. Eine Domain enthält alle MCS Provider und User die an einer bestimmten Konferenz beteiligt sind. Die Aufteilung einer Domain ist dabei hierarchisch organisiert. Es existiert immer ein oberster Provider, der für das Management der Domain, sowie der Kommunikationsverbindungen zuständig ist. Alle anderen Provider einer Domain sind diesem untergeordnet.

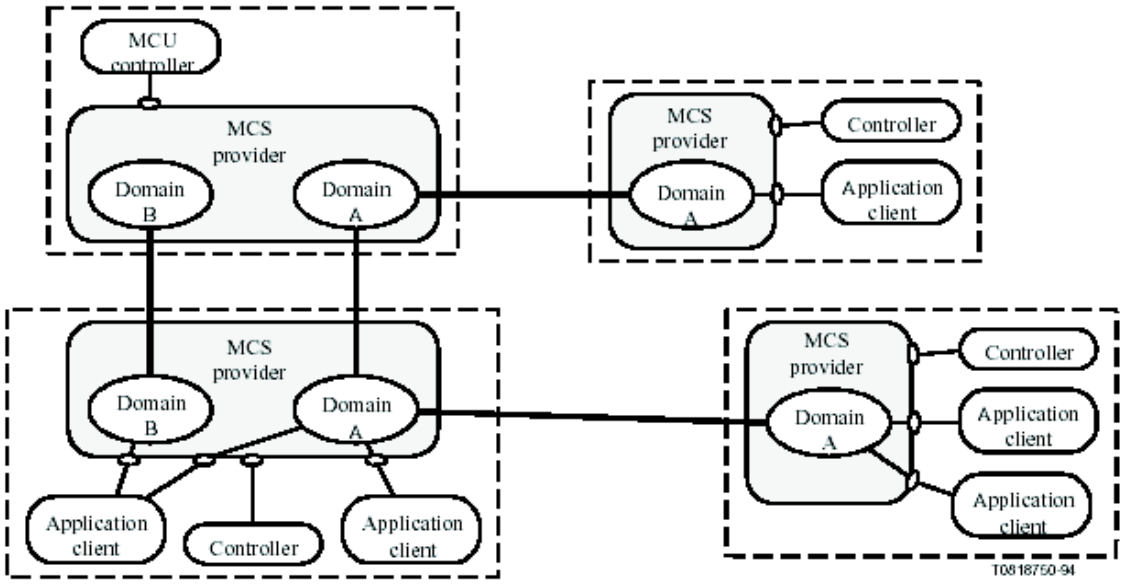


Fig. 3.4 MCS Kommunikationsmodell (aus [ITU98a])

Die Struktur der Domain wird beim Verbindungsaufbau ausgehandelt. Da jedoch während des Betriebes jederzeit weitere Provider hinzustossen oder die Domain verlassen können, muss der ausgewählte oberste Provider nicht während der gesamten Dauer der Verbindung der gleiche bleiben.

MCS Provider und User müssen keine unterschiedlichen Anwendungen sein, es handelt sich dabei lediglich um eine konzeptionelle Trennung. In der Implementierung des RDP-Clients sind beide Teil der selben Klasse.

3.4.3 Verbindungsaufbau

Will ein MCS User Verbindung zu einem anderen MCS User aufnehmen, so teilt er dies seinem MCS Controller mit. Ist der andere Teilnehmer nicht bereits Teil der selben Domain, muss der MCS Controller zu dem entfernten MCS Provider Verbindung aufnehmen, der für diesen Teilnehmer zuständig ist. Für diese MCS Verbindung wird eine neue Domain erzeugt, die nun beide Provider und alle an der Konferenz teilnehmenden Anwendungsinstanzen enthält. Sollten wei-

tere Nutzer an der Konferenz teilnehmen wollen, so können sie jederzeit Teil der aktuellen Domain werden. Sobald eine Anwendungsinstanz Mitglied der gewünschten Domain ist, kann sie den gewünschten Kommunikationskanälen beitreten.

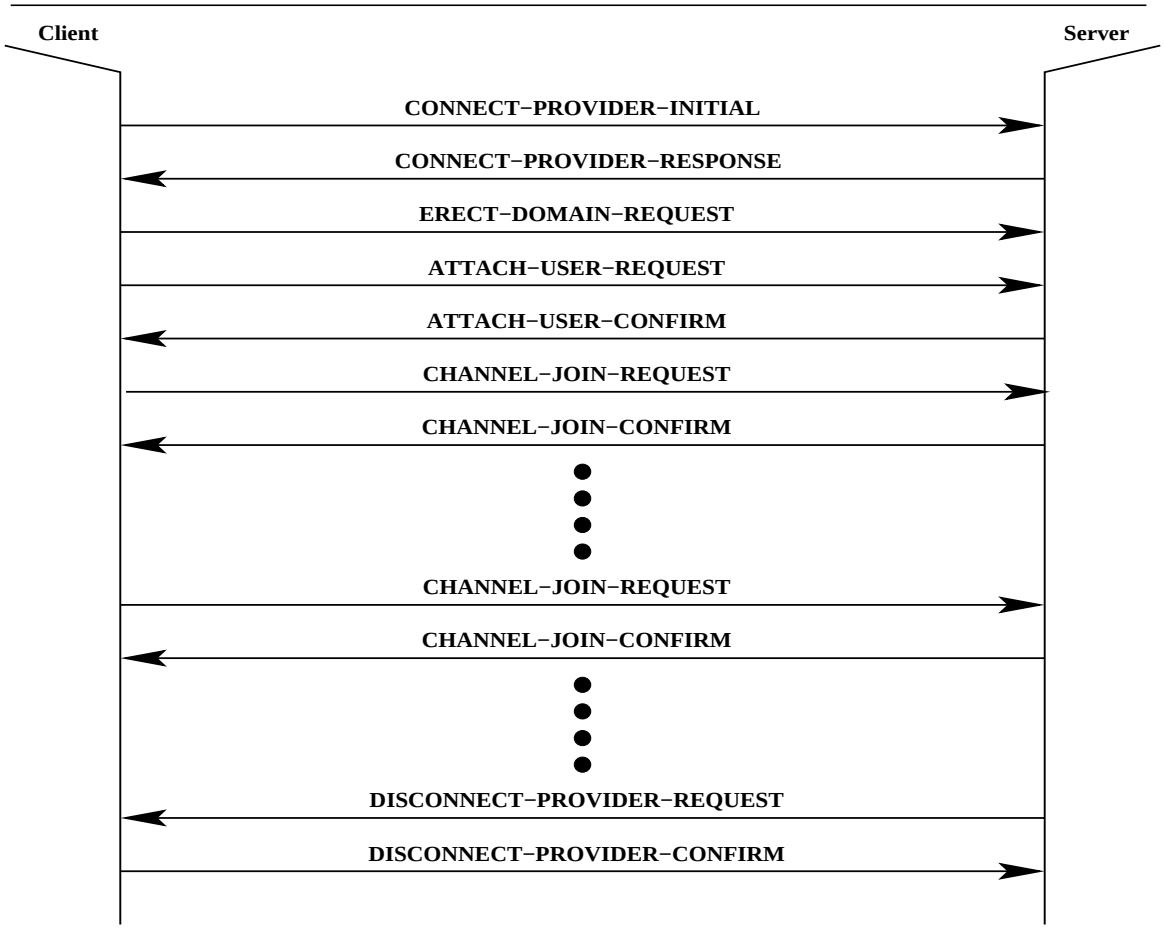


Fig. 3.5 Typische Kommunikation während einer MCS Verbindung

Ein MCS Provider baut eine Verbindung zu einem anderen MCS Provider auf in dem er eine CONNECT-PROVIDER-INITIAL Nachricht sendet. Diese kann von der gegenüberliegenden Seite angenommen oder abgelehnt werden. Mit Ausnahme der Protokollversion gelten alle während einer Verbindung ausgehandelten Parameter für alle Teilnehmer in einer Domain. Das bedeutet, dass Providern, die erst später Mitglied in einer Domain werden, keinerlei Mitspracherecht zukommt. Ein Teilnehmer kann die Parameter anhand der Antwort auf die CONNECT-PROVIDER-INITIAL Nachricht feststellen. Folgende Parameter werden beim Aufbau einer Domain ausgehandelt:

– Maximale Anzahl an MCS Kanälen, die simultan verwendet werden können: Dies umfasst alle Kanäle, sowohl solche die von allen Teilnehmern verwendet werden können, als auch private Kanäle

– Die maximale Anzahl von User IDs die gleichzeitig in Gebrauch sein dürfen. Dies begrenzt effektiv die Anzahl der teilnehmenden Parteien

- Die maximale Anzahl an Tokens, die ein Anwender halten darf. Token dienen der Zuteilung von knappen Ressourcen. Nur wer sich um ein Token bewirbt und es auch bekommt, darf auf diese spezielle Ressource zugreifen.
- Die Anzahl an Prioritätsstufen, die der Provider unterstützt. Im MCS Protokoll existieren mehrere Prioritätsstufen für die Übermittlung von Daten, Top, High, Medium und Low, durchnummeriert von 0 -3. Unterstützt ein Provider weniger als vier Prioritätsstufen, so werden Daten mit nicht unterstützter Priorität mit der niedrigstmöglichen Prioritätsstufe gesendet.
- minimaler Datendurchsatz. Zwei Provider können aushandeln, ob die MCS Verbindungen einen minimalen Datendurchsatz leisten müssen. Provider, die sich das nicht garantieren können, laufen Gefahr von der Teilnahme ausgeschlossen zu werden.
- Maximale Höhe der Domain. Domains sind baumartig strukturiert. Mit diesem Parameter kann die Höhe dieses Baumes begrenzt werden.
- maximale Größe einer MCS Dateneinheit
- Protokollversion
- Upward/Downward Flag. Hiermit erklärt der Sender ob der gerufene Provider in der Hierarchie über oder unter ihm stehen soll
- Anwendungsspezifische Daten. Diese Daten können beliebig lang sein. Sie sind gedacht um speziell auf die Anwendung zugeschnittene Daten auszutauschen, die nicht Teil des MCS Protokolls sind. Im Falle des RDP-Protokolls handelt es sich hierbei um Informationen, die Terminal Server und Client beim Verbindungsaufbau aushandeln. Darunter fällt unter anderem der verwendete Zeichensatz, die Größe des Anwendungsfensters, die Stärke der verwendeten Verschlüsselung und noch einiges mehr.

Ein Provider kann eine Domain jederzeit durch Senden einer DISCONNECT-PROVIDER Nachricht verlassen. Die Domain wird dabei in zwei Teile aufgespalten.

Wenn der Verbindungsaufbau erfolgreich abgeschlossen worden ist, sendet der in der neuen Domain-Hierarchie weiter unten stehende Provider eine ERECT-DOMAIN-REQUEST Nachricht an den über ihn stehenden Provider. In der ERECT-DOMAIN-REQUEST Nachricht wird allen übergeordneten Providern mitgeteilt, dass sich die Höhe der Domain geändert hat. Die Höhe einer Domain kann sich ändern, wenn eine MCS Verbindung aufgebaut oder beendet wurde oder ein untergeordneter Provider eine Veränderung meldet. Ein Provider erhält auf eine ERECT-DOMAIN-REQUEST Nachricht keine Antwort.

Sobald die Domain errichtet worden ist, können sich Anwendungen an der Domäne anmelden. Dies geschieht durch Senden einer ATTACH-USER-REQUEST Nachricht. Wenn sich ein Client einer Domäne anschließt erhält er eine für diese Domäne eindeutige Benutzeridentifikationsnummer, die von obersten Provider der Domain ausgegeben wird. Ein Anwendungsprozess kann eine Domain jederzeit durch Senden einer DETACH-USER-REQUEST Nachricht wieder verlassen.

Hat sich ein Anwender erfolgreich an einer Domäne angemeldet muss er lediglich noch Zugang zu den Kommunikationskanälen beantragen, die für ihn von Interesse sind. Dies geschieht durch das Senden eines CHANNEL-JOIN-REQUEST. Ein CHANNEL-JOIN-REQUEST beinhaltet die MCS-USER-ID des betreffenden Prozesses sowie die Nummer des Kommunikationskanales dem der Prozess beitreten will.

Ein MCS System kennt drei Arten von Kanälen:

- Multicast Kanäle (Multicast Channels): Ein Multicast Kanal dient dazu Daten mit allen, oder einer Untermenge aller Clients, auszutauschen. Es kann mehrerer solcher Kanäle geben. Ein Kanal repräsentiert dabei möglicherweise alle Teilnehmer einer Konferenz, während ein anderer Kanal lediglich einigen wenigen Teilnehmern vorbehalten ist.
- Kanäle für einzelne Mitglieder (Single Member Channels): Single Member Channels werden für die Signalisierung verwendet. Jedem Teilnehmer ist ein solcher Kanal zugeordnet, um an ihn persönlich adressierte Nachrichten zustellen zu können. Bei der Kanalnummer handelt es sich um die User-ID des entsprechenden Teilnehmers.
- Private Kanäle (Private Channels): Multicast Kanäle und Single Member Kanäle dürfen nur durch MCS-Provider verwaltet werden. Mit privaten Kanälen hat jedoch jeder Teilnehmer an einer MCS Konferenz die Möglichkeit selbst Multicast Kanäle aufzubauen. Andere User können diese Kanäle durch die üblichen CHANNEL-JOIN und CHANNEL-LEAVE Nachrichten betreten oder verlassen. Der Teilnehmer, der die Kanäle aufgebaut hat, hat jedoch die Kontrolle darüber wer diese Kanäle betreten darf oder nicht. So wird ein privater Datenaustausch zwischen mehreren Teilnehmern gewährleistet.

Ein MCS-User kann einen Kanal jederzeit durch das Senden einer CHANNEL-LEAVE-REQUEST Nachricht wieder verlassen.

3.4.4 Besonderheiten der MICROSOFT Implementierung

Microsofts Variante des MCS-Protokoll enthält einige Besonderheiten. Jeder RDP-Client ist nicht nur MCS-USER sondern auch gleichzeitig ein MCS-Provider. Die Höhe einer MCS-Domain ist jedoch maximal zwei, da der Terminal Server als oberster Provider ganz oben in der Hierarchie steht und alle am Server angeschlossenen Teilnehmer in der selben Stufe der Hierarchie stehen. Theoretisch wären mehr als zwei Hierarchiestufen möglich. Jedoch ist es erst mit Version fünf des RDP Protokolls möglich auch von einem anderen Server aus Verbindungen zu einem Client aufzubauen. So ist es zum Beispiel möglich sich als Administrator den Desktop eines Benutzers anzeigen zu lassen um ihm bei der Lösung spezieller Probleme zu helfen.

Es gibt außerdem nur zwei MCS-Kanäle. Einen globalen Kanal an dem alle Klienten des Terminal Servers angeschlossen sind sowie einen privaten Kanal, den der Server verwendet um mit einem Client Daten auszutauschen. Erst mit Version fünf des RDP-Protokolls ist es möglich mehr als einen Kanal aufzubauen. Außerdem gibt es erst dort die Möglichkeit, dass Clients untereinander Verbindungen aufbauen können.

Durch die oben genannten Einschränkungen ist klar, dass die MCS-Implementierung lediglich die grundlegendsten Funktionen zum Verbindungsmanagement bereithält.

3.5 Die Sicherheitsschicht

Die Kommunikation über ein TCP/IP Netz ist prinzipbedingt äußerst unsicher. Daten durchlaufen auf dem Weg zum Ziel fast immer mehrere Vermittlungsrechner, über die man selbst keine Kontrolle ausübt. Ein Angreifer, der administrativen Zugriff auf solch einen Rechner hat und über Kenntnisse des RDP-Protokolls verfügt, kann die Kommunikation abhören. Er könnte auf diese Weise so eventuell an wichtige Daten, wie Passwörter, Benutzernamen oder Schlüsselzertifikate, gelangen. Um dies zu verhindern wurde die Sicherheitsschicht als dritte Schicht in die RDP-Protokollhierarchie aufgenommen.

Bei der Sicherungsschicht handelt es sich um eine Eigenentwicklung der Firma Microsoft, die nicht Teil der T.120 Standards ist. Die zwei Hauptaufgaben der Sicherheitschicht sind die Verschlüsselung der Kommunikation sowie das Lizenzmanagement. Microsoft verlangt nicht nur für den Server an sich Lizenzgebühren, sondern es existieren auch sogenannte Clientlizenzen. Microsoft verlangt also auch für jeden angemeldeten Benutzer eine Gebühr. Die Sicherungsschicht stellt die nötigen Mechanismen bereit um festzustellen ob ein Client sich auch rechtmäßig am Server anmeldet. Um Manipulationen an den Daten auszuschließen werden diese außerdem noch digital signiert.

Der verwendete Verschlüsselungsalgorithmus basiert auf dem RC4 Algorithmus der RSA Data Security Inc.. Die digitale Signatur wird durch eine Kombination aus dem SHA-1 Algorithmus und dem MD5 Algorithmus gebildet.

3.5.1 Der Secure Hash-Algorithmus 1

SHA1 steht für Secure Hash Algorithm 1. Der Secure-Hash-Algorithmus wurde am amerikanischen National Institute for Standards and Technology (NIST) im Rahmen des Digital Signature Standards (DSS) entwickelt. Ziel war es ein einheitliches und sicheres Verfahren für die digitale Signatur von Nachrichten zu entwickeln.

SHA-1 nimmt als Eingabe beliebige Daten entgegen, deren Gesamtlänge 2^64 Bit (das entspricht 16384 Petabyte) nicht überschreiten sollte und produziert daraus einen sogenannten Message Digest von insgesamt 160 Bit Länge. Zwei unterschiedliche Nachrichten ergeben mit einer sehr hohen Wahrscheinlichkeit auch zwei unterschiedliche Message Digests. Der Empfänger einer Nachricht kann deshalb durch die erneute Anwendung des Algorithmus auf die gleiche Nachricht feststellen, ob die Nachricht während der Übermittlung auf irgendeine Weise manipuliert worden ist.

SHA-1 gehört zur Familie der Block-Message Digests, das bedeutet das SHA-1 immer einen Block fester Größe (512 Bits) verarbeitet. Nachrichten deren Länge kein Vielfaches von 512 Bytes sind müssen deswegen eventuell verlängert werden. SHA-1 gilt als sicher, weil es mathe-

matisch äußerst aufwendig ist die Nachricht zu finden, die zu einem Message Digest gehört, beziehungsweise zwei Nachrichten zu finden, denen der gleiche Message Digest zugeordnet ist. Detaillierte Informationen zum Verfahren welches SHA-1 verwendet, findet man in [EJ01]

3.5.2 Der MD5 Algorithmus

Der MD5 Algorithmus wurde von Professor Ronald Rivest vom Massachusetts Institute of Technology in Zusammenarbeit mit der RSA Data Security Inc. entwickelt. Der MD5 Algorithmus ist eine Weiterentwicklung des MD4 Algorithmus und entstand nach dem im MD4 Algorithmus eine schwerwiegende Sicherheitslücke festgestellt wurde.

Der MD5 Algorithmus nimmt als Eingabe Nachrichten beliebiger Länge entgegen und generiert daraus einen Message Digest von 128 Bit Länge. Genau wie bei SHA-1 ergeben auch beim MD5 Algorithmus zwei unterschiedliche Nachrichten mit sehr hoher Wahrscheinlichkeit unterschiedliche Message Digests. Der Empfänger einer Nachricht kann auch hier durch erneutes Anwenden des Algorithmus auf diese Nachricht feststellen, ob die Nachricht manipuliert worden ist.

MD5 gehört, genau wie SHA-1, zur Familie der Block-Message-Digests. Ein Block enthält bei MD5 ebenfalls 512 Bits. Beim MD5 Algorithmus wird jedoch zusätzlich noch die Länge einer Nachricht mitberücksichtigt und als 64 Bit Wert ans Ende der Nachricht angehängt. Dies begrenzt die effektive Länge einer Nachricht auf 16384 Petabytes. Detaillierte Informationen zur Funktionsweise des MD5 Algorithmus lassen sich in [RIV92] finden.

3.5.3 Der RC4 Algorithmus

Der RC4 Algorithmus wurde von Professor Ronald Rivest vom Massachusetts Institute of Technology 1987 in Zusammenarbeit mit der RSA Data Security Inc. entwickelt. Im Gegensatz zum MD5 Message Digest Algorithmus ist RC4 ursprünglich nicht zur freien Verwendung bestimmt gewesen. Im Jahre 1992 tauchte der Quellcode zu RC4 jedoch auf einer Mailing Liste zum Thema Datensicherheit auf. Seitdem existieren auch Implementierungen des Algorithmus, die ohne Lizenz der RSA Data Security Inc. verwendet werden dürfen.

Bei RC4 handelt es sich um ein symmetrisches Verfahren. Man verwendet im Gegensatz zu Public Key Verfahren den selben Schlüssel sowohl zur Ver- als auch zur Entschlüsselung. Die Länge eines Schlüssels kann zwischen 1 und 2048 Bits liegen. Aus dem Schlüssel wird durch ein spezielles Verfahren eine Zufallszahl generiert. Der verschlüsselte Text wird durch die Anwendung der X-OR Operation auf Zufallszahl und Text erzeugt. Der Empfänger einer Nachricht erhält den Klartext durch erneute Anwendung der X-OR Operation auf Schlüssel und der verschlüsselten Nachricht. Es ist deshalb äußerst wichtig, das der Schlüssel nicht im fremde Hände gerät.

Detaillierte Informationen zum RC4 Verfahren findet man in Kapitel 17 von [SCH95].

3.5.4 Aufbau eines Datenpaketes der Sicherheitsschicht

Eine PDU der Sicherheitsschicht ist relativ einfach strukturiert. Sie besteht aus einem Kopfteil sowie einem Datenteil.

Fig. 3.6 Aufbau einer PDU der Sicherheitsschicht

Die Länge des Kopfteils ist variabel. Für den Fall das die Verbindung verschlüsselt abläuft hat der Header eine Länge von zwölf Byte. Davon sind vier Byte für Flags reserviert und acht für die Signatur der Daten. Im Flagfeld wird angezeigt ob ein Paket verschlüsselt ist oder im Klartext vorliegt. Außerdem ob bereits eine Lizenz ausgegeben wurde oder noch nicht. Die Signatur wird aus einer Mischung von SHA-1 sowie MD5 gebildet.

Ist keine Verschlüsselung aktiv dann ist die Länge des Headers davon abhängig, ob eine Lizenz erforderlich ist oder nicht. Für den Fall das eine Lizenz erforderlich ist, bleibt nur noch das Flag-Feld übrig. Ist keine Lizenz erforderlich schrumpft die Länge des Headers auf null.

3.5.5 Aushandeln der Schlüssel

Ein Windows NT/2000 Terminal Server kennt drei verschiedene Verschlüsselungsstufen: High, Medium und Low:

- High: Sowohl die Kommunikation in Richtung Server, als auch die Kommunikation in Richtung des Clients sind verschlüsselt. Die Schlüsselstärke beträgt 128 Bit. Ursprünglich gab es die Stufe High nur in der US-amerikanischen Version des Servers, doch seitdem die USA die Exportbeschränkungen für Cryptographieprodukte gelockert hat, findet sich diese Einstellung auch in den internationalen Versionen wieder.
- Medium: Sowohl die Kommunikation in Richtung Server, als auch die Kommunikation in Richtung des Clients sind verschlüsselt. Die Schlüsselstärke beträgt 40 Bit.

– Low: Nur die Kommunikation in Richtung des Servers läuft verschlüsselt ab. In Gegenrichtung findet die Übertragung hingegen im Klartext statt. Die Schlüsselstärke beträgt 40 Bit.

Die momentan gültige Stufe teilt der Server im Rahmen des MCS-CONNECT-RESPONSE Paketes mit. Der Server sendet eine sogenannte RSA-Info Struktur. Diese enthält die eingestellte Verschlüsselungsstärke, die Schlüssellänge sowie eine vom Server erzeugte Zufallszahl. Kann ein Client die geforderte Verschlüsselungsstufe nicht einhalten, so wird die Verbindung an diesem Punkt der Übertragung beendet.

Es werden drei verschiedene Schlüssel verwendet. Einer zum Verschlüsseln der Daten (Encrypt-Key), einer zum Entschlüsseln der Daten (Decrypt-Key), sowie ein Schlüssel für die Signatur. Der Client generiert aus der vom Server erzeugten Zufallszahl und einer selbst erzeugten Zahl alle drei Schlüssel selbst.

Es wird so sichergestellt, dass bei jedem Verbindungsaufbau andere Schlüssel verwendet werden. Hat ein potentieller Angreifer eine Verbindung abhören können bleiben ihm die anderen trotzdem verborgen Wenn ein Schlüssel 4096 mal verwendet worden ist, dann wird ein neuer Schlüssel erzeugt. Die ersten acht Byte des Encrypt- sowie des Decrypt-Keys werden für diesen Zweck verwendet.

Wurden die Schlüssel erfolgreich erzeugt, dann werden sie an den Server übertragen. Ab diesem Zeitpunkt läuft die Übertragung nur noch verschlüsselt ab.

Der JX RDP-Client unterstützt zur Zeit nur 40 Bit Schlüsselstärke. Bei der 128 Bit Verschlüsselung hat Microsoft das Verfahren geändert mit dem die Schlüssel ausgehandelt werden. Leider ist das neue Verfahren bisher noch nicht bekannt geworden

3.5.6 Lizenzierung

Ein Terminal Server kann sich in zwei verschiedenen Betriebsmodi befinden: Remote-Administration Mode oder Client-Mode.

– Remote-Administration Mode: In diesem Modus akzeptiert der Server lediglich zwei Verbindungen gleichzeitig. Der Remote-Administration Modus ist zur Fernwartung von Windows NT Servern gedacht und man benötigt hierfür keine Lizenz.

– Client Modus: Im Client Modus nimmt ein Server maximal so viele Verbindungen an, wie Clientlizenzen existieren. Es gibt hierbei zwei Möglichkeiten. Entweder werden die Lizenzen pro Verbindung vergeben oder pro Client. Werden die Lizenzen pro Verbindung vergeben, dann können beliebige Rechner eine Verbindung zum Server aufbauen, jedoch maximal so viele wie Client-Lizenzen existieren. Werden Lizenzen pro Client vergeben, dann gibt bleibt die Lizenz beim Client. Das bedeutet das bei beispielweise 25 Lizenzen immer nur die selben 25 Rechner den Server nutzen dürfen. Die zweite Möglichkeit bietet sich an, wenn man bestimmen will welche Rechner den Server überhaupt nutzen dürfen. Eine einmal vergebene Lizenz muss dann beim Client zwischengespeichert werden.

Die Lizenzierung selbst läuft in drei Stufen ab. Zuerst sendet der Server ein DEMAND-LICENCING Paket. Das Paket enthält eine neue Zufallszahl mit deren Hilfe ein neuer RC4-Schlüssel generiert wird. Im Gegensatz zur Aushandlung der Verschlüsselung antwortet der Client hier nicht mit dem generierten Schlüssel sondern lediglich mit der vom Client verwendeten Zufallszahl. Der Server generiert sich den entsprechenden Schlüssel selbst.

In der zweiten Phase bekommt der Client ein REQUEST-AUTHENTICATION Paket. Das Paket enthält ein verschlüsseltes Token das mit Hilfe des vorher generierten Schlüssels entschlüsselt werden kann. Wenn der richtige Schlüssel verwendet wurde, sollte das Token den String “TEST” in Unicode codiert enthalten. Aus dem Token und dem Rechnernamen wird nun eine sogenannte Hardware-ID generiert. Die Hardware-ID wird verschlüsselt, signiert und am Ende an den Server zurückübertragen.

Ist bei der bisherigen Prozedur kein Fehler aufgetreten, antwortet der Server in der dritten Phase der Lizenzierung schließlich mit der erteilten Lizenz. Dies geschieht durch ein LICENCE-ISSUE Paket.

3.6 Die RDP-Schicht

Die RDP-Schicht ist die oberste Schicht in der RDP-Protokollhierarchie. RDP steht für Remote Desktop Protocol und ist gleichzeitig namensgebend für die gesamte Protokollfamilie. Das RDP Protokoll entstand aus einer Zusammenarbeit von Microsoft mit der International Standards Organisation und ist niedergelegt in der Recommendation T.128: Multipoint Application Sharing (nachzulesen in [ITU98c]).

Mit den Diensten des RDP-Protokolls kann man eine Anwendung, welche auf einem Rechner ausgeführt wird (dem sogenannten Host), auf einem oder mehreren anderen Rechnern darstellen. Dabei kann, unter bestimmten Vorraussetzungen, ein Rechner die Kontrolle über die Anwendung übernehmen, indem sie Maus- oder Tastaturbefehlen an die Anwendung sendet. Das RDP-Protokoll erzeugt dadurch die Illusion, dass die Anwendung lokal abläuft, obwohl sie in Wahrheit auf einem anderen Rechner ausgeführt wird und lediglich Aus- und Eingabe umgeleitet werden.

Das Protokoll, so wie es in [ITU98c] beschrieben ist, erlaubt die Darstellung einzelner Anwendungen. Im Falle des Windows NT Terminal Servers ist die Anwendung eine komplette Windows Session. Es werden also nicht einzelne Windows Anwendungen (wie z.B. Notepad, Word oder Excel) dargestellt, sondern ein kompletter Windows Desktop inklusive Startmenü. Ein gewöhnlicher Windows Desktop in der Auflösung 800 x 600 erzeugt in der Standard Farbtiefe von 256 Farben schon fast 500 Kilobyte an Daten. Da das RDP-Protokoll auch bei Verbindungen mit einer geringen Datenübertragungsrate (z.B. Modems) noch akzeptabel funktionieren soll, enthält das RDP-Protokoll Mechanismen um das Datenaufkommen zu reduzieren.

Das RDP-Protokoll kennt zwei unterschiedliche Betriebsmodi, den Legacy Mode und den Base Mode. Der Hauptunterschied zwischen beiden Betriebsarten ist, dass der Base Modus zukünftige Erweiterungen berücksichtigt. Eine Anwendung die sich an den T.128 Standard hält soll

beide Betriebsmodi unterstützen. Der Base Mode verwendet jedoch Dienste, welche im ITU Standard T.124 - Generic Conference Control festgelegt sind und vom RDP-Protokoll wie es Microsoft umgesetzt hat, nicht unterstützt werden. Das RDP-Protokoll implementiert deshalb nur den Legacy-Mode.

3.6.1 Aufbau einer RDP-PDU

Eine RDP-PDU ist relativ einfach strukturiert, sie besteht aus einem Kopfteil sowie einem Datenteil. Der Kopfteil enthält drei Felder. Ein Feld enthält die Länge des Datenteils, das Zweite den Typ des Paketes. Erlaubte Typen sind DEMAND-ACTIVE, CONFIRM-ACTIVE, DATA, sowie DEACTIVATE.Das letzte Feld enthält die MCS-User-ID, die dem Client zugeordnet ist, und auch auf der RDP-Ebene zur Identifikation verwendet wird.

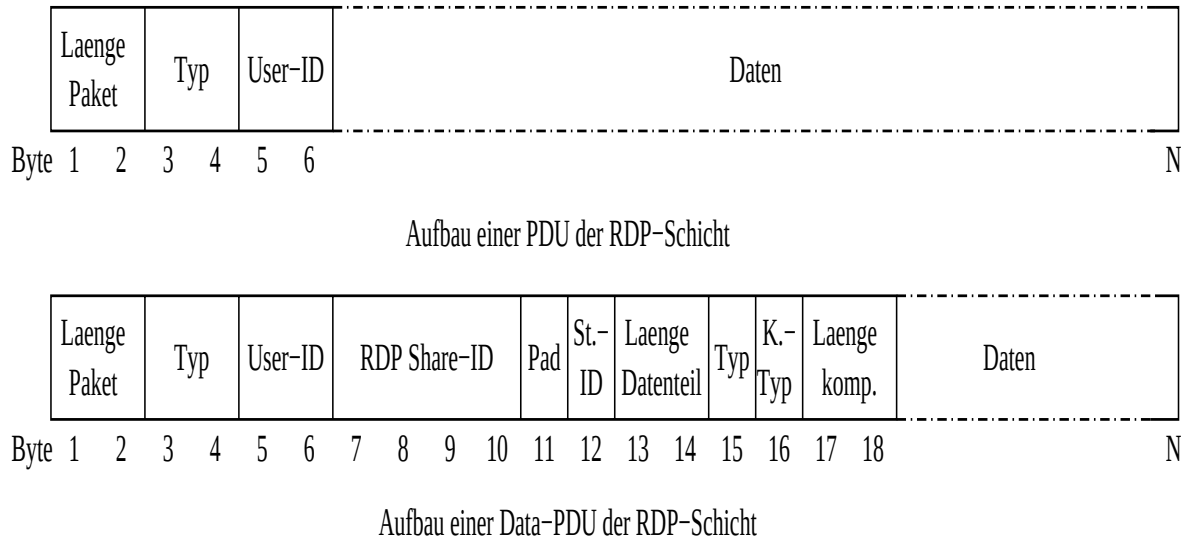


Fig. 3.7 Aufbau einer RDP-PDU

Im Kopfteil des Paketes sind zwei Byte für die Länge des Paketes reserviert. Dies begrenzt die effektive Länge des Datenteils auf 65535 Byte (64 Kilobyte). Der Datenteil selbst kann unterschiedlichste Informationen beinhalten. Die Art der Informationen wird im Feld Typ angegeben. Eine DATA-PDU kann beliebig viele Datentypen aufnehmen. Zulässige Typen sind:

- POINTER: Eine POINTER-PDU enthält Daten zur Verwaltung der Mauszeiger. Entweder wird ein neuer Mauszeiger übermittelt oder ein existierender Mauszeiger (der im Zwischenspeicher liegt) soll angezeigt werden. Eine POINTER-PDU geht immer vom Server aus
- INPUT: Die einzige Daten-PDU die vom Client selbst gesendet wird. Eine INPUT-PDU enthält Tastatur- oder Mausbefehle, die der Client an den Server zur Verarbeitung sendet.
- BELL: Weist den Client an, einen Systemton auszugeben.

- LOGON: Eine LOGON-PDU wird vom Server gesendet, wenn die Anmeldung des Clients erfolgreich war.
- UPDATE: Eine UPDATE-PDU weist den Client an einen Bereich des Bildschirms zu aktualisieren.

Die Stream-ID kennzeichnet den RDP-Stream über den das Datenpaket versendet werden soll. Das RDP-Protokoll sieht verschiedene Datenkanäle für bestimmte RDP-Daten vor, unter anderem Audio. Laut dem ITU T.128 Standard dürfen Daten auch in komprimierter Form vorliegen. Das Feld Kompressionstyp kennzeichnet die Art der verwendeten Datenkompression, das letzte Feld des Headers enthält die komprimierte Länge der Daten.

Eine UPDATE-PDU kann wiederum in einzelne Unterkategorien unterteilt werden:

- UPDATE-ORDERS: Weist den Client an eine bestimmte Zeichenoperation durchzuführen
- UPDATE-BITMAP: Weist den Client an einen Teil des Bildschirms mit einer neuen Bitmap zu aktualisieren.
- UPDATE-PALETTE: Alle Bitmapdaten liegen in indizierter Form vor. Das bedeutet dass zur Interpretation der Daten eine Palette nötig ist. Diese Palette wird in einem UPDATE-PALETTE Paket übermittelt.
- SYNCHRONIZE: Fordert den Client auf sich mit dem Server zu synchronisieren. Diese PDU tritt lediglich zu Beginn einer Sitzung auf.

Eine RDP-Sitzung besteht aus beliebigen Abfolgen dieser Daten-PDUs. Sie endet wenn der Server eine DEACTIVATE-PDU sendet.

3.6.2 Verbindungsaufbau

Ein Windows Terminal Server kennt zwei verschiedene Arten der Benutzeranmeldung: den Autologin und den “normalen” Login. Beim normalen Login zeigt der Client zu Beginn der Session das normale Anmeldefenster eines Windows NT/2000 PCs und der Benutzer kann sich wie gewohnt anmelden. Für den Fall dass das nicht gewünscht oder möglich ist (z.B. bei Kiosk Systemen), kann man sich auch automatisch anmelden.Welche Art der Anmeldung vom Client gewünscht wird, wird im LOGON-INFO Paket mitgeteilt.

Der Verbindungsaufbau beginnt mit dem Senden eines LOGON-INFO Paketes vom Client zum Server. Im LOGON-INFO Paket sind alle Daten enthalten, die der Server für den Anmeldeprozess benötigt. Dazu gehören die Art der Anmeldung (automatisch oder manuell), Benutzername, Passwort, die Domäne an der man sich anmelden will sowie etwaige Kommandos die der Server nach dem Login ausführen soll. Bis auf die Art der Anmeldung sind jedoch alle Daten optional.

Der Server antwortet auf ein gültiges LOGON-INFO Paket mit einer DEMAND-ACTIVE Nachricht. Damit wird nun der eigentliche Verbindungsaufbau eingeleitet. Der Client antwortet nun mit einem CONFIRM-ACTIVE Paket. Das CONFIRM-ACTIVE Paket enthält alle Informationen, die für den Server wichtig sind. Dies sind unter anderem die Anzahl der auf dem Client installierten Fonts, die Anzahl der unterstützten Zeichenoperationen, die Größe des Anwendungsfensters und vieles mehr.

Handelt es sich um gültige Daten, dann ist der eigentliche Verbindungsaufbau an dieser Stelle abgeschlossen. Sollte beim Verbindungsaufbau ein Fehler aufgetreten sein, beendet der Server die verbindung mit einem DEACTIVATE Paket. Ein Deactivate Paket wird auch gesendet, wenn die Anwendungssitzung beendet ist.

3.6.3 Capabilities

RDP-Client kann prinzipiell jedes an den Server angeschlossene Gerät sein. Es muss sich dabei nicht um einen PC handeln. Deshalb ist es nötig dem Server beim Verbindungsaufbau mitzuteilen, welche Funktionalität der Client beherrscht. Dies geschieht über sogenannte Capabilities.

Capabilities beschreiben die Eigenschaften des angeschlossenen Clients und erlauben es dem Server die Datenübertragung relativ flexibel auf die Bedürftnisse des Clients anzupassen. Capabilities sind in mehrerer Gruppen unterteilt:

- General Capabilities
- Bitmap Capabilities
- Order Capabilities
- Bitmap Cache Capabilities
- Colortable Cache Capabilities
- Window Activation Capabilities
- Control Capabilities
- Pointer Capabilities
- Share Capabilities
- Non-Standard Capabilities

General Capabilities

Die General Capabilities beschreiben die grundlegenden Eigenschaften des Clients. Dies sind unter anderem:

- verwendete Plattform: Windows, Unix, OS/2 oder ähnliches
- verwendete Betriebssystemversion: Windows 95, OS/2 Warp 3.0 etc.
- Protokollversion

– unterstützte Datenkompression

Die verwendete Plattform sowie die Betriebssystemversion dienen lediglich der Information.

Bitmap Capabilities

Die Bitmap Capabilities beschreiben die Fähigkeiten des Clients im Bezug auf Bitmaps.

Zu den Bitmap Capabilities gehören die Höhe und Breite des virtuellen Desktops, die bevorzugte Farbtiefe sowie die unterstützten Farbtiefen. Außerdem teilt der Client in den Bitmap Capabilities mit, ob er Bitmap Kompression unterstützt. Microsoft verwendet eine einfache Lauflängenkodierung. Treten mehrere Pixel der gleichen Farbe hintereinander auf, so wird nur ein Farbwert übertragen und der Client fügt die fehlenden Pixel selbst wieder hinzu.

Order Capabilities

Um die Menge an zu übertragenden Daten zu begrenzen kann ein RDP-Client gewisse, besonders häufig vorkommende, Zeichenoperationen selbst implementieren. Der Server braucht in diesem Fall lediglich die Zeichenoperation zu übertragen, anstatt komplette Bitmaps. Besonders häufige vorkommende Operationen sind, das Zeichnen von Text sowie das Verstecken/Anzeigen von Fenstern. In den Order Capabilities legt der Client fest, welche Zeichenbefehle er unterstützt.

Bitmap Cache Capabilities

Um weitere Daten bei der Übertragung einzusparen kann ein Client häufig benutzte Bitmaps zwischenspeichern. Welche Fähigkeiten ein Client auf diesem Gebiet mitbringt, wird in den Bitmap Cache Capabilities festgelegt

Colortable Cache Capabilities

Es gibt zwei Formate für Bitmaps. Entweder wird für jeden Pixel der Farbwert einzeln gespeichert oder es werden alle verwendetet Farbwerte in einem Index eingetragen. Bei der letzten Möglichkeit müssen anstelle der Pixeldaten lediglich Verweise in die Tabelle gespeichert werden. Eine solche Tabelle nennt man auch Palette. Ein RDP-Client kann besonders häufig verwendete Paletten lokal zwischenspeichern, so dass diese nicht jedesmal wieder neu übermittelt werden müssen. Ob ein Client Paletten zwischenspeichern kann, wird in den Colortable Cache Capabilities festgelegt.

Window Activation Capabilities

Legt fest ob der Client spezielle Kommandos zur Steuerung von Anwendungsfenstern unterstützt. Diese Capabilities sind nur für den Fall von Interesse, dass einzelne Anwendungsfenster dargestellt werden. Diese Capabilities werden deshalb vom Terminal Server ignoriert.

Control Capabilities

Legt fest welche Arten der Fernsteuerung vom Client unterstützt werden. Erst die Version Fünf des RDP-Protokolls erlaubt Fernsteuerung eines Clients. In Version Vier werden Control Capabilities deshalb vom Server ignoriert.

Pointer capabilities

Üblicherweise sind Mauszeiger nur zweifarbig. Moderne Windows Versionen unterstützen jedoch auch Mauszeiger mit beliebig vielen Farben. Pointer Capabilities legen fest, ob ein Client diese farbigen Mauszeiger darstellen kann

Share Capabilities

Share Capabilities spielen im RDP-Protokoll keine Rolle. Sie sind lediglich für Anwendungen interessant, die die Generic Conference Control verwenden, also im Base Mode operieren. Diese Capabilities werden vom Server ignoriert. Sie sind lediglich aus Kompatibilitätsgründen enthalten

Non Standard Capabilities

Damit sind alle Capabilities gemeint, die nicht Teil des T.128 Standards sind. Non-Standard Capabilities werden verwendet um anwendungsspezifische Eigenschaften auszutauschen. Microsoft verwendet diese Capabilities intensiv, sie sind jedoch leider nicht dokumentiert.

3.6.4 Zeichenoperationen

Der T.128 Standard schreibt sogenannte Drawing Orders fest. Dabei handelt es sich um besonders häufig vorkommende Zeichenoperationen. Diese Zeichenoperationen sollen die Menge der übertragenen Daten reduzieren, so dass eine Serververbindung auch über Verbindungen mit geringer Bandbreite möglich ist.

Ein Client kann alle Zeichenoperationen unterstützen, oder nur eine Teilmenge davon. Er sollte jedoch zumindest die Textoperationen beherrschen, da diese von allen Zeichenoperationen am häufigsten verwendet werden.

RDP unterscheidet zwischen zwei Klassen von Befehlen: Primärbefehle (Primary Orders) und Sekundärbefehle (Secondary Orders). Bei Primärbefehlen handelt es sich um Befehle, die direkt im Fenster des RDP-Clients umgesetzt werden. Sekundärbefehle sind Befehle deren Ergebnis nicht sofort sichtbar sein muss. Ein Beispiel für einen Primärbefehl wäre das Zeichnen eines Bitmaps, ein Sekundärbefehl ist z.B. die Anweisung ein Bitmap in den Cache zu schreiben.

Die meisten Befehle verwenden eine sogenannte Drei-Wege-Rasteroperation (ROP3). Eine ROP3 beschreibt wie die Bits in einer Quellbitmap und einem Muster (einem sogenannten Brush), auf eine Zielbitmap angewendet werden. Die Operation interpretiert nur die Bits der Daten. Zugelassen sind alle bitweisen Operatoren. (und, oder nicht, xor etc.)

Jeder Zeichenbefehl kann zusätzlich noch Clipping-Informationen enthalten.

Das Remote Desktop Protocol unterstützt folgende Primärbefehle:

- Destination Blocktransfer: Wendet eine ROP3 auf die Zielbitmap an. Die Zielbitmap liegt auf dem Bildschirm. (Quellbitmap und Brush werden nicht verwendet).
- Pattern Blocktransfer: Wendet eine ROP3 mit einem bestimmten Muster auf die Zielbitmap an. Die Zielbitmap liegt auf dem Bildschirm (Es wird keine Quellbitmap verwendet).
- Screen Blocktransfer: Wendet eine ROP3 auf die Quell- und die Zielbitmap an. Beide Bitmaps liegen auf dem Bildschirm (Ein Brush wird nicht verwendet).
- Memory Blocktransfer: Gleiche Funktion wie Screen Blocktransfer. Das Quellbitmap liegt jedoch im Cache und nicht auf dem Bildschirm.
- Memory Dreiwege Blocktransfer: Wendet eine ROP3 auf eine Zielbitmap auf dem Bildschirm an unter der Verwendung eines Musters. Die Quelle liegt im Cache.
- Erweiterter Text: Ein Befehl für erweiterten Text enthält einen String mit Formatinformationen. Dieser String enthält Positionierungsinformationen für Buchstabenbitmaps (sogenannte Glyphs). Ein Formatstring bestimmt, wo und wie die Buchstaben auf dem Bildschirm dargestellt werden sollen sowie welche Buchstaben dargestellt werden sollen. Häufig vorkommende Formatstrings werden lokal zwischengespeichert. Diese Glyphs wurden vorher mit Hilfe einer Cache Font Order im Schriftsatzcache abgelegt.
- Line: Zeichnet eine Linie von Punkt A nach Punkt B.
- Rect: Zeichnet ein Rechteck auf dem Bildschirm.
- Polyline: Zeichnet eine Linie mit beliebig vielen Eckpunkten auf den Bildschirm.
- Desksave: Ein Desksave nutzt die Eigenschaften einer Fensterumgebung aus. In der Regel werden Fenster verschoben, verkleinert, vergrößert oder geschlossen. Bei einem Desksave wird der momentane Zustand des Desktops aufgezeichnet (z.B. Desktop leer, kein Fenster geöffnet) und kann bei Bedarf wiederhergestellt werden. (z.B. wenn das letzte Fenster geschlossen wurde). Für Desksave Befehle existiert ein eigener Zwischenspeicher.

Das RDP-Protokoll unterstützt außerdem noch folgende Sekundärbefehle:

- Cache Bitmap: Sichert ein bereits verarbeitetes Bitmap im Cache oder holt es wieder hervor.
- Cache Font: Sichert ein Glyph im Cache oder holt es aus dem Cache wieder hervor.
- Cache Colortable: Sichert eine Palette im Cache oder holt sie wieder hervor.

3.6.5 Caches im RDP-Protokoll

Das RDP-Protokoll sieht Zwischenspeicher für besonders häufig verwendete Daten vor. Auf der einen Seite wird so das Datenaufkommen reduziert, auf der anderen Seite erfordern einige Zeichenoperationen zwingend Cachespeicher. (Zum Beispiel Memory- und Memory- Drei-Wege Blocktransfer). Die Größe dieser Zwischenspeicher wird beim Verbindungsaufbau ausgehandelt.

Es existieren fünf verschiedene Zwischenspeicher:

- Bitmap Cache
- Font Cache
- Desktop Cache
- Cursor Cache
- Text Cache

Bitmap Cache

Im Bitmap Cache werden verschiedenste Arten von Bitmap Daten zwischengespeichert. Der Bitmap Cache wird hauptsächlich von den Memory- und Memory Dreiwege Blocktransfer Operationen verwendet. Die Anzahl sowie die Größe der Caches wird beim Verbindungsaufbau festgelegt. In unserem Fall sind es drei Cachespeicher mit jeweils 600 Elemente.

Font Cache

Im Font Cache werden Buchstabenbitmaps, sogenannte Glyphs, zwischengespeichert.Ein Cache zum Speichern von Zeichen muss groß genug sein um alle Zeichen eines Zeichensatzes aufzunehmen. Im Falle des Windows NT Terminal Servers sind das 256 Zeichen. Die Anzahl der zwischengespeicherten Fonts wird beim Verbindungsaufbau festgelegt. Der JX-Client kann 12 Fonts mit 256 Zeichen pro Font zwischenspeichern.

Desktop Cache

Der Desktop Cache enthält die Momentaufnahmen des Desktops, die eine Desksave Order erzeugt. Die Größe des Desksave Caches wird beim Verbindungsaufbau festgelegt. Der JX Client kann 4 komplette Images zwischenspeichern.

Cursor Cache

Im Cursor Cache werden Bitmaps der Windows-eigenen Mauszeiger zwischengespeichert. Wenn ein Windows Cursor (z.B die Sanduhr) zum ersten Mal benötigt wird dann bekommt der Client ein Bitmap des Cursors. Ein lokaler Mauscursor kann mit diesem Bild versehen werden. Jeder Cursor wird im Cache zwischengespeichert. Sollte der entsprechende Cursor später erneut gebraucht werden, muss er nicht erneut übertragen werden. Die Größe des Cursor Caches wird beim Verbindungsaufbau festgelegt. Der JX Client kann 32 Mauszeigerzwischenspeichern.

Text Cache

Im Text Cache werden häufig verwendete Formatierungsstrings zwischengespeichert. Ein Formatierungsstring legt die Position von Zeichen auf dem Bildschirm fest. Außerdem legt er fest welche Buchstaben verwendet werden. Im Formatstring sind lediglich die Indizes im Fontcache angegeben, wo der entsprechende Buchstabe zwischengespeichert wurde. Die Größe des Text Caches wird beim Verbindungsaufbau festgelegt. Der JX Client kann 256 Formatierungsstrings zwischenspeichern.

3.7 Zusammenfassung

Dieses Kapitel gab einen umfassenden Überblick über das Remote Desktop Protokoll. Zuerst wurden die Grundlagen des Protokolls näher erläutert. Danach wurden die einzelnen Schichten der Protokollhierarchie detailliert beschrieben. Angefangen mit der Transportschicht über die MCS-Schicht bis hin zur Sicherheitsschicht. Das Kapitel schloss mit der Beschreibung der RDP-Schicht.

4 Implementierung

4.1 Einleitung

Dieses Kapitel gibt einen detaillierten Überblick über die Implementierung des RDP-Clients. Zunächst wird der Aufbau und die Struktur des Programmes beschrieben. Danach wird näher auf die einzelnen Klassen der Implementierung eingegangen. Zuerst werden die Kommunikationsklassen erläutert, danach werden die Klassen beschrieben, die für die grafische Ausgabe zuständig sind. Das Kapitel endet mit einer Leistungsbewertung des JX RDP-Clients anhand eines speziellen Benchmarks.

Bei der Beschreibung der Funktion wird, aus Gründen der Übersichtlichkeit, nur auf die wichtigen Methoden näher eingegangen.

4.2 Aufbau des RDP Clients

Der RDP-Client besteht aus insgesamt 40 Java Klassen. Diese lassen sich grob in drei Bereiche unterteilen: Klassen, die die Kommunikation zwischen Client und Server regeln; Klassen, die für die Darstellung der Daten auf dem Bildschirm zuständig sind sowie Unterstützungsklassen.

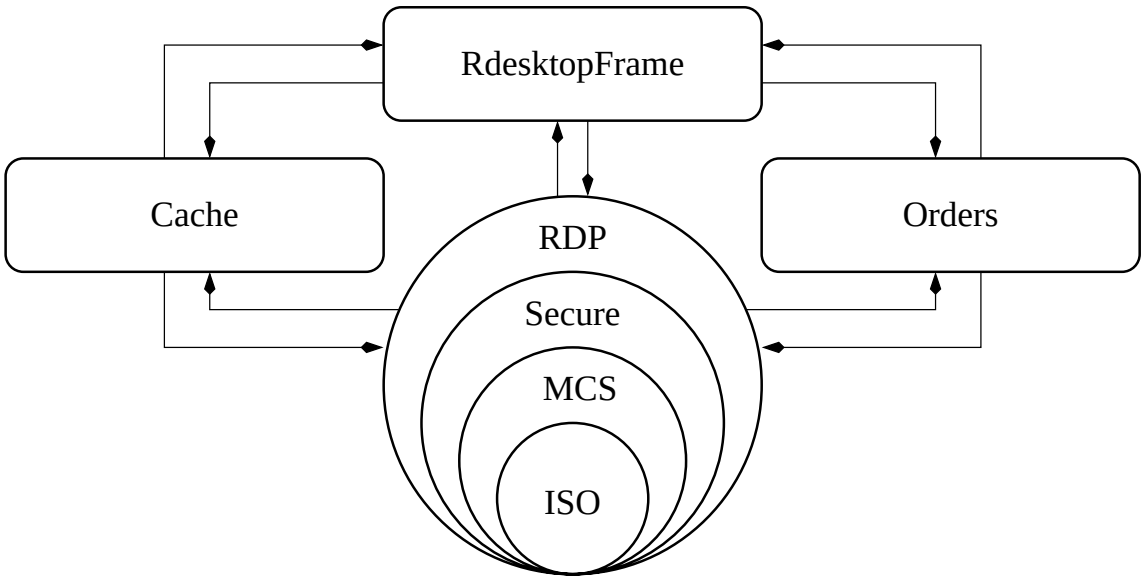


Fig. 4.1 Aufbau des RDP-Clients

Die Kommunikationsklassen halten sich an das RDP-Referenzmodell. Jede Schicht des Modells wird durch eine einzelne Klasse repräsentiert. Das Abstraktionsprinzip bleibt hierbei gewahrt, da jede Klasse nur auf Methoden der in der Hierarchie direkt darunterliegenden Klasse zugreifen darf. Die Kommunikationsschicht besteht aus vier Klassen:

- `ISO`: Die Klasse `ISO` implementiert alle Dienste und Schnittstellen der ISO-Transportschicht. Sie ist außerdem die einzige Klasse, die direkten Zugriff auf das Netzwerk hat.
- `MCS:` Die Klasse `MCS` implementiert alle Dienste und Schnittstellen der MCS-Schicht. Sie nutzt die von der Klasse `ISO` angebotenen Dienste.
- `Secure:` Die Klasse `Secure` ist für die Verschlüsselung der Daten sowie das Lizenzmanagement zuständig. Sie nutzt zur Übertragung die Methoden der `MCS` Klasse. Die verwendeten Cryptoalgorithmen sind in den Unterstützungsklassen `RC4, SHA1` und `MD5` implementiert.
- `RDP:` Die Klasse `RDP` implementiert alle Dienste und Schnittstellen der RDP-Schicht. Die Klasse `RDP` ist außerdem die einzige Klasse, mit der die Klassen zur Grafikdarstellung direkt kommunizieren.
- `Packet:` Die Klasse `Packet` stellt eine Abstraktion eines RDP-Paketes dar. Sie enthält Zugriffsmethoden um Daten aus einem Paket zu extrahieren oder in ein Paket zu schreiben. Außerdem können einzelne Bereiche oder ganze Pakete kopiert werden.

Das Grafik-Subsystem ist für die Verarbeitung und die Darstellung der RDP-Grafikdaten zuständig. Außerdem kümmert es sich noch um die Verarbeitung von Tastatur- und Mausbefehlen. Zu den Klassen des Grafiksubsystems gehören:

- `Cache:` Die Klasse `Cache` ist für alle im RDP-Client verwendeten Cache-Speicher zuständig. Sie implementiert die Font- und Textcaches, die Zwischenspeicher für Mauszeiger und den Desksave Befehl sowie Zwischenspeicher für Bitmapdaten.
- `Orders:` Die Klasse `Orders` ist für die Verarbeitung der Zeichenbefehle (Drawing Orders) zuständig. Dazu gehören die Mechanismen zur Interpretation der Zeichenbefehle sowie Methoden für die Bildschirmdarstellung. Die Optionen der Zeichenbefehle werden in der Klasse OrderState zwischengespeichert.
- `RdesktopFrame:` Die Klasse `RdesktopFrame` enthält das Fenster das zur Darstellung des virtuellen Desktops auf dem Bildschirm verwendet wird.

Diese acht Hauptklassen machen von einigen Hilfsklassen Gebrauch. Die wichtigsten sind:

- `OrderState:` Die Klasse `OrderState` nimmt alle Informationen auf, die zur Verarbeitung der Zeichenoperationen nötig sind. Zeichenoperationen können von vorhergehenden Orders abhängig sein, deshalb bietet die Klasse `OrderState` auch die Möglichkeit sich den momentanen Zustand der Verarbeitung zu merken.
- `RC4, SHA1` und `MD5:` Diese Klassen implementieren die von der Sicherheitsschicht verwendeten Crypto- und Message Digest Algorithmen. Diese Klassen wurden nicht selbst implementiert, sondern stammen aus einer freien Cryptobibliothek namens Cryptix (www.cryptix.org).

- `KeyLayout` und `KeyCode:` Diese Klassen enthalten den Code, der zur Unterstützung von internationalen Keyboards notwendig ist.
- `Rdesktop: Rdesktop` ist die Klasse welche die main Methode enthält. Sie dient lediglich zum Starten der Anwendung.

Diese Klassen werden in den nachfolgenden Kapiteln näher erläutert.

4.3 Die Kommunikationsschicht

Die Kommunikationsschicht ist für alle Aspekte einer RDP-Verbindung zuständig. Darunter fallen der Verbindungsauf- und Abbau sowie die korrekte Übertragung und Interpretation der RDP-Daten. Die Kommunikationsschicht besteht aus vier Klassen: `ISO, MCS, Secure und RDP.` Jede dieser vier Klassen kapselt eine Schicht aus dem RDP-Referenzmodell. Die RDP-Pakete werden in einer eigenen Klasse namens `Packet` verwaltet.

4.3.1 Abstraktion eines RDP-Paketes - die Klasse `Packet`

Die Klasse `Packet` stellt eine Abstraktion eines RDP-Paketes dar.

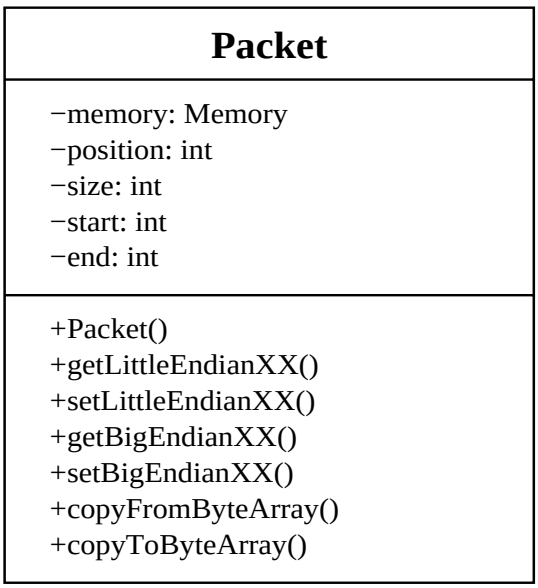


Fig. 4.2 Aufbau der Klasse `Packet`

Die RDP-Protokollfamilie kennt drei Arten von Datentypen: Unsigned Byte, Unsigned Short und Unsigned Integer. Die Datentypen die mehrere Bytes enthalten (short und integer) können sowohl im Big-Endian als auch im Little-Endian Format vorliegen. Die Klasse `Packet` stellt Methoden bereit um diese Datentypen zu schreiben und zu lesen. Da Java keine vorzeichenlosen Datentypen kennt, muß ein Objekt der Klasse `Packet` außerdem noch die Umsetzung von vorzeichenbehafteten zu vorzeichenlosen Datentypen gewährleisten. Die Klasse `Packet` enthält darüberhinaus noch Methoden zum Kopieren von Daten oder ganzen Paketen.

Herzstück eines Paketes ist ein Memory-Objekt. Alle Methoden eines Paketes arbeiten intern auf diesem Memory-Objekt. Ein Memory-Objekt erlaubt jedoch nur den wahlfreien Zugriff auf den verwalteten Speicherbereich. Ein Paket wird jedoch in der Regel sequentiell gelesen oder geschrieben. Die Klasse `Packet` enthält deshalb eine Variable `Position` in der die aktuelle Position im Paket gespeichert wird sowie `End` in der die letzte Position vermerkt wird an der gültige RDP-Daten vorhanden sind. (Für den Fall das mehr Speicher für ein Paket reserviert worden ist, als gebraucht wird).

Ein Paket enthält alle Kopfteile der einzelnen Schichten. So wird unnötiges Kopieren, bei der Übergabe in eine andere RDP-Schicht, vermieden. Beim Schichtwechsel wird lediglich die Position im Paket angepasst. Die Klasse `Packet` enthält deshalb Variablen um die Position der einzelnen Kopfteile zu speichern.

Die Methoden der Klasse `Packet` sind im einzelnen:

- public Packet(Memory data):

  Der einzige Konstruktor der Klasse `Packet` erzeugt ein neues Objekt und verwendet zur Speicherung der Daten das übergebene Memory-Objekt. Die Größe des Paketes ist die Größe des in ihm enthaltenen Memory-Objektes.

- void set8(int value), void setLittleEndian16(int value), void setBigEndian16(int value), void setLittleEndian32(int value), void setBigEndian32(int value):

  Mit diesen Methoden kann ein Byte, Short oder Integer an die aktuelle Position des Paketes gesetzt werden. Der Positionszeiger wird automatisch auf die nächste freie Stelle im Paket gesetzt. Die Methode `set8(int value)` verwendet lediglich das unterste Byte, `getLittleEndian16(int value)` und `getBigEndian16(int value)` verwenden lediglich die unteren zwei Byte des übergebenen Wertes. Diese Methoden erledigen die Konvertierung in die entsprechende Byte-Order automatisch falls nötig.

- void set8(int position, int value), void setLittleEndian16(int position, int value), void setBigEndian16(int position, int value) void setLittleEndian32(int position, int value), void setBigEndian32(int position, int value):

  Mit diesen Methoden kann ein Byte, Short oder Integer an eine beliebige Position des Paketes geschrieben werden. Der Positionszeiger wird nicht beeinflusst. Die Methode `set8(int value, int position)` verwendet lediglich das unterste Byte, `getLittleEndian16(int value, int position)` und `getBigEndian16(int value, int position)` verwenden lediglich die unteren zwei Byte des übergebenen Wertes. Diese Methoden erledigen die Konvertierung in die entsprechende Byte-Order automatisch falls notwendig

- int get8(), int getLittleEndian16(), int getBigEndian16(), int getLittleEndian32(), int getBigEndian32():

Liest ein Byte, Short oder Integer an der aktuellen Position im Paket. Der Positionzeiger wird automatisch um die notwendige Anzahl von Bytes erhöht. Die Byte-Order wird, falls erforderlich, automatisch in das richtige Format konvertiert.

- int get8(int position), int getLittleEndian16(int position), int getBigEndian16(int position), int getLittleEndian32(int position), int getBigEndian32(int position):

  Liest ein Byte, Short oder Integer an einer beliebigen Position im Paket. Der Positionzeiger wird automatisch um die notwendige Anzahl von Bytes erhöht. Die Byte-Order wird, falls erforderlich, automatisch in das richtige Format konvertiert.

- `void copyFromByteArray(byte[] source, int src_offset, int dst_offset, int length)`:

  Kopiert ein Byte Array an eine beliebige Position im Paket. Mit `dst_offset` und `length` kann die Position im Byte Array sowie die Länge des zu kopierenden Teils beeinflusst werden. `src_offset` bestimmt die Position innerhalb des Paketes

- void copyToByteArray(byte[] destination, int src_offset, int dst_offset, int length):

  Kopiert den Inhalt dieses Paketes in ein Byte Array. Mit `src_offset` und `length` kann die Position im Paket sowie die Länge des zu kopierenden Teils beeinflusst werden. `dst_offset` bestimmt die Position innerhalb des Byte Arrays

- void copyFromPacket(Packet source, int src_offset, int dst_offset, int length):

  Kopiert ein anderes Paket an eine beliebige Position in diesem Paket. Mit `dst_offset` und `length` kann die Position im Quellpaket sowie die Länge des zu kopierenden Teils beeinflusst werden. `src_offset` bestimmt die Position innerhalb des Zielpaketes.

- void copyToPacket(Packet destination, int src_offset, int dst_offset, int length):

  Kopiert dieses Paket an eine beliebige Position in einem anderen Paket. Mit `src_offset` und `length` kann die Position im Quellpaket sowie die Länge des zu kopierenden Teils beeinflusst werden. `dst_offset` bestimmt die Position innerhalb des Zielpaketes.

- int size():

  Mit Hilfe dieser Methode kann die Größe eines Paketes bestimmt werden. Ein Paket kann jedoch weniger Nutzdaten enthalten, als seine Größe vermuten lässt. Will man das Ende der Nutzdaten kennen sollte man die Methode `getEnd()` verwenden.

- `int getPosition()`:

  Liefert die aktuelle Position im Paket.

- int getEnd():

  Liefert die Position innerhalb des Paketes, an der die Nutzdaten enden.

- void incrementPosition():

Erhöht den Positionszeiger um einen beliebigen Wert.

- void setPosition(int position):

  Setzt den Positionszeiger auf eine beliebige Position innerhalb des Paketes.

- void markEnd(), void markEnd(int position):

  Markiert das Ende der Nutzdaten. Der Zeiger `End` wird entweder auf die aktuelle Position, oder auf den übergebenen Wert gesetzt

- int getHeader(int header):

  Liefert die Position innerhalb des Paketes, an der sich der entsprechende Header befindet. Gültige Werte für den Übergabeparameter sind: `MCS_HEADER`, `SECURE_HEADER`, bzw. `RDP_HEADER`.

- void setHeader(int header):

  Legt die aktuelle Position als Beginn des entsprechenden Headers fest. Gültige Werte für den Übergabeparameter sind: `MCS_HEADER`, `SECURE_HEADER`, bzw. `RDP_HEADER`.

- Memory getMemoryObject():

  Liefert eine Referenz auf das interne Memory-Objekt.

4.3.2 Implementierung der ISO-Transportschicht - Die Klasse `ISO`

Die Klasse `ISO` repräsentiert die ISO-Transportschicht in der RDP-Protokollhierarchie. Sie stellt Methoden bereit für den Verbindungsaufbau auf der Transportebene und das Senden und Empfangen von ISO-Daten. Die Klasse `ISO` ist außerdem für die Initialisierung des TCP/IP Netzwerkes zuständig. Zum Datenaustausch mit höheren Schichten werden Objekte der Klasse `Packet` verwendet.

Die Methoden der Klasse `ISO` sind im einzelnen:

- public ISO():

  Der einzige Konstruktor der Klasse `ISO`. Im Konstruktor wird, über den Namensdienst des JX Betriebssystems, eine Referenz auf den Memorymanager eingeholt.

- void connect(IPAddress host, int port), void connect(IPAddress host):

  Mit diesen Methoden wird eine Verbindung auf der ISO-Ebene hergestellt. Die Adresse des Servers wird dabei als Objekt vom Typ IPAddress übergeben. Zusätzlich kann noch der Server Port angegeben werden ansonsten wird der Standard Port 3389 verwendet. In der `connect` Methode wird der Socket sowie alle notwendigen Input- und OutputStreams erzeugt. Details zum Verbindungsaufbau werden in Kapitel 3.2.1 behandelt.

- void disconnect():

Beendet eine vorher aufgebaute ISO-Transportverbindung wieder. Der Server wird über das Ende der Verbindung in Kenntnis gesetzt und alle Input- und Output-Streams sowie der Socket werden geschlossen.

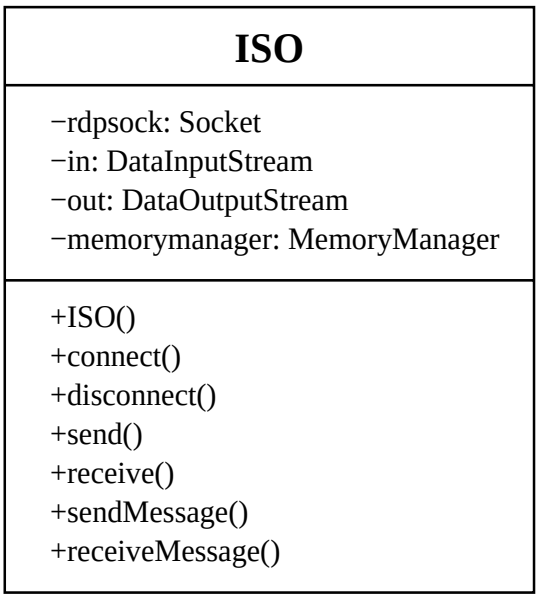


Fig. 4.3 Aufbau der Klasse `ISO`

- void sendMessage(int type):

  Versendet eine einfache Nachricht vom Typ `type` über den Socket. Zulässige Typen sind CONNECT-REQUEST und DISCONNECT-REQUEST. Diese Methode wird nur zum Verbindungsauf- bzw Abbau verwendet.

- int receiveMessage():

  Dient zum Empfang einer einfachen Nachricht auf der Transportebene. Der Typ der Nachricht (CONNECT-CONFIRM, DATA) wird zurückgeliefert. Diese Methode wird nur beim Verbindungsaufbau verwendet.

- Packet init(int length):

  Erzeugt ein Paket der Transportschicht vom Typ DATA und liefert das fertig initialisierte Paket zurück. Der Positionszeiger zeigt auf den Beginn des Datenteils.

- void send(Packet data):

  Erlaubt das Versenden einer ISO-PDU vom Typ DATA. Die `send` Methode füllt alle Felder des ISO-Headers mit den korrekten Daten und schickt die ISO-PDU schließlich über die TCP-Verbindung.

- Packet receive():

Diese Methode liefert die nächste über die TCP-Verbindung hereinkommende ISO-PDU zurück. Die `receive` Methode wertet das Feld Länge im Header aus und reserviert nicht mehr Platz als tatsächlich verwendet wird.

4.3.3 Die Implementierung der MCS-Schicht - Die Klasse `MCS`

Die Klasse `MCS` repräsentiert die MCS-Schicht des RDP-Protokolls. Sie stellt alle Methoden bereit, die zum Verbindungsauf- und Abbau auf der MCS-Ebene sowie zur Übermittlung von MCS-Daten notwendig sind. Außerdem kümmert sich die Klasse `MCS` um das Domain-Management, die An-und Abmeldung von Benutzern und den Auf- und Abbau von MCS-Kanälen. Zum Datenaustausch mit höheren Schichten oder der darunterliegenden ISO-Schicht werden Objekte der Klasse `Packet` verwendet.

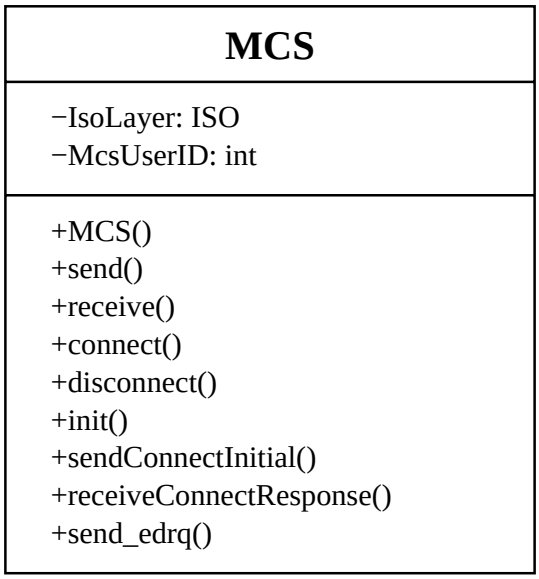


Fig. 4.4 Aufbau der Klasse `MCS`

MCS-Daten liegen im ASN.1 Format vor. ASN.1 steht für Abstrakt Syntax Notation 1. Es handelt sich hierbei um einen ISO-Standard, der eine einheitliche Repräsentation vom primitiven und zusammengesetzten Datentypen festlegt. Die Regeln, nach denen ASN.1 Datentypen kodiert werden, nennt man Basic Encoding Rules (BER). Ein nach dem Basic Encoding Rules kodierter Datentyp besteht immer aus zwei Teilen: einem Tag der die Art der Daten beaschreibt (Zusammengesetzt, Integer, String etc.) gefolgt von den eigentlichen Nutzdaten. Die `MCS` Klasse enthält auch Methoden um Daten BER-konform zu schreiben oder BER-kodierte daten zu lesen.

Die Klasse `MCS` enthält folgende Methoden:

- public MCS():

Der Konstruktor der Klasse `MCS` erzeugt ein privates Objekt vom Typ ISO. So ist sichergestellt das lediglich die MCS-Schicht direkt auf die Dienste der ISO-Schicht zugreifen kann.

- void connect(IPAddress host, int port, Packet data):

  Erzeugt eine Verbindung auf der MCS-Ebene. Dazu wird zunächst eine Verbindung auf der ISO-Ebene aufgebaut. Erst wenn diese Verbindung erfolgreich verlaufen ist, wird eine MCS-Dom5ane aufgebaut. Der Ablauf einer MCS-Verbindung ist in Kapitel 3.4.3 detailliert beschrieben. Beim Paket `data` handelt es sich nicht um ein vollständiges Datenpaket. Das MCS Protokoll erlaubt die Übermittlung von Benutzerdarten. Es handelt sich dabei um Daten, die nicht Teil des MCS-Standards sind, von den angeschlossenen Anwendern aber bereits während der Errichtung der Domain gebraucht werden. Das Paket `data` enthält solche speziellen Daten. Es handelt sich hierbei zum Beispiel um den Hostnamen des Clients, das Layout des angeschlossenen Keyboards oder der Größe des virtuellen Desktops.

- void disconnect():

  Beendet die MCS-Verbindung.

- `Packet init(int length)`:

  Erzeugt ein leeres MCS-Datenpaket und liefert es zurück. Der Positionszeiger zeigt auf das erste Element des Datenteils der MCS-PDU.

- void send(Packet data):

  Sendet ein MCS-Datenpaket. Die `send` Methode füllt zunächst alle Felder im MCS-Header mit den korrekten Werten und übergibt das Paket danach an die ISO-Schicht zur Weiterverarbeitung.

- Packet receive():

  Holt die nächste MCS-PDU von der ISO-Transportschicht ab. Handelt es sich um eine PDU des Typs SPLIT-DOMAIN-INDICATION (SDIN), bedeutet das, dass die Verbindung vom Server beendet wurde. Das Ende der Verbindung wird den höheren Schichten durch eine EOFException angezeigt.

- void sendConnectInitial(Packet data):

  Sendet eine MCS-PDU vom Typ CONNECT-INITIAL. Dieses Paket enthält die Benutzerdaten, die der `connect` Methode übergeben worden sind.

- void send_edrq():

  Sendet eine MCS-PDU vom Typ ERECT-DOMAIN-REQUEST.

- void send_aurq():

  Sendet eine MCS-PDU vom Typ ATTACH-USER-REQUEST.

- void send_cjrq(int channel_id):

Sendet eine MCS-PDU vom Typ CHANNEL-JOIN-REQUEST. Außerdem wird die Nummer des Kanals angegeben dem man beitreten will.

- void receiveConnectResponse(Packet data):

  Holt das nächste Datenpaket von der ISO-Transportschicht und interpretiert es als CONNECT-RESPONSE-PDU.

- void receive_aucf():

  Holt das nächste Datenpaket von der ISO-Transportschicht und interpretiert es als ATTACH-USER_CONFIRM.

- void receive_cjcf():

  Holt das nächste Datenpaket von der ISO-Transportschicht und interpretiert es als CHANNEL-JOIN-CONFIRMATION.

- void sendBerHeader(Packet data, int tagval, int length):

  Fügt den Header eines nach den Basic Encoding Rules kodierten Datentyps an der aktuellen Position in das Paket ein. Der zweite Parameter bestimmt dabei die Art der Daten, der dritte Parameter die Länge.

- void sendBerInteger(packet data, int value):

  Fügt ein nach den Basic Encoding Rules kodiertes Integer an die aktuelle Position in das Paket ein.

- void sendDomainParams(Packet data, int maxchannels, int maxusers, int maxtokens, int maxpdusize):

  Sendet eine MCS-PDU vom Typ DOMAIN-PARAMS.

- int berParseHeader(Packet data, int tagval):

  Prüft ob der nächste nach den Basic Encoding Rules kodierte Datentyp vom Typ `tagval` ist und liefert die Länge des Datentyps zurück.

- void parseDomainParams(Packet data):

  Interpretiert eine vom Server erhaltene DOMAIN-PARAMS Datenstruktur. Ein Terminal Server beendet sofort die Verbindung falls die vom Client angebotenen Domänenparameter umgültig sind. Die Domänenparameter, die hier interpretiert werden, sind demnach lediglich die Bestätigung, dass der Verbindungsaufbau ordnungsgemäß abgelaufen ist. Deshalb werden diese nicht weiter beachtet.

- int getUserID():

  Liefert die User ID des RDP-Clients zurück.

4.3.4 Verschlüsselung und Lizenzmanagement - Die Klasse `Secure`

Die Klasse `Secure` ist für die Verschlüsselung der RDP-Kommunikation und das Lizenzmanagement zuständig. Sie baut auf den Diensten der MCS-Schicht auf und implementiert alle Funktionen, die von der Sicherheitsschicht im RDP-Referenzmodell verlangt werden.

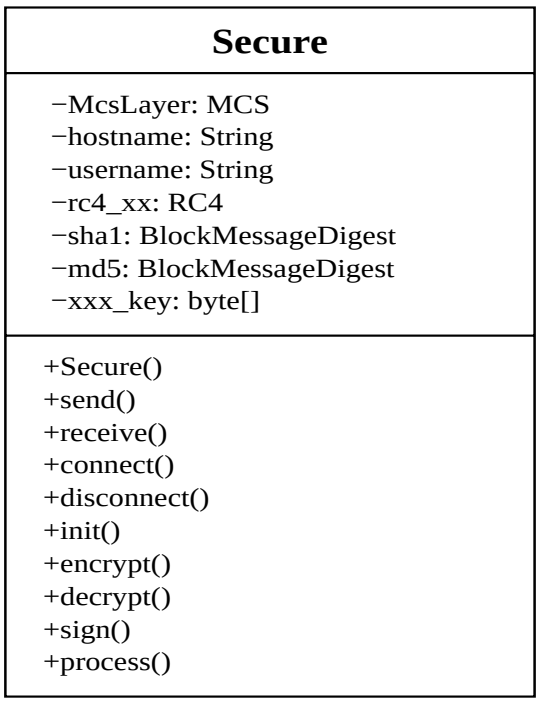


Fig. 4.5 Aufbau der Klasse `Secure`

Fast alle wichtigen Daten der Sicherheitsschicht sind im ASN.1 Format. Alle verwendeten Tags liegen als Klassenvariablen vor.

Die wichtigste Methode der Klasse `Secure` ist sicherlich `connect`. Sie ist für den Verbindungsaufbau auf der Sicherheitsschicht zuständig.

Verbindungsaufbau

Wie bereits früher erwähnt wurde, werden beim Verbindungsaufbau auf der MCS-Schicht bereits erste RDP-Daten übertragen. Diese Daten werden in der Methode `sendMcsData` erzeugt und der `connect` Methode der MCS-Schicht im Paket `mcs_data` übergeben. War die Erzeugung der Verbindung erfolgreich, dann enthält das Paket `mcs_data` die Antwort des Servers auf die Benutzerdaten des Clients. Sie werden in der Methode `processMcsData` verarbeitet.

Am wichtigsten sind hierbei die Zufallszahlen, die zum Erzeugen der Schlüssel verwendet werden. Die Methode `processCryptInfo` extrahiert diese Zufallszahlen aus den MCS-Daten und übernimmt die weitere Verarbeitung, an deren Ende die neu erzeugten Schlüssel stehen. Diese Schlüssel müssen jetzt nur noch dem Server bekanntgemacht werden. Dafür ist die Methode `establishKeys` zuständig.

Lizenzierung

Der Header eines Paketes der Sicherheitsschicht enthält ein Feld für ein Flagbyte. Ist dieses Flag auf den Wert SEC_LICENCE_NEG eingestellt beginnt die Lizenzierungsphase. Die Methode `process` ist dafür zuständig. Die drei Phasen aus denen die Lizenzierung besteht wurden in Kapitel 3.5.6 genauer erläutert. Für jede dieser Phasen ist eine eigene Methode zuständig. `processDemand` für die DEMAND-LICENCING Phase, `processAuthreq` für die REQUEST-AUTHENTICATION Phase und `processIssue` wenn die Lizenz ausgegeben wird.

Senden und Empfangen

`init` erzeugt ein leeres Paket der Sicherheitsschicht. Mit der Methode `send` wird ein Paket auf der Sicherheitsschicht versendet und mit `receive` empfangen. Sowohl `send` als auch `receive` verwenden `encrypt` bzw. `decrypt` zum Ver- und Entschlüsseln und `sign` zum signieren der Daten.

4.3.5 Die RDP-Schicht - Die Klasse `RDP`

Die Klasse `RDP` implementiert die oberste Klasse der RDP-Protokollhierarchie. Sie ist für die Aufbereitung der RDP-Daten, den Verbindungsaufbau sowie Senden und Empfangen von Daten zuständig

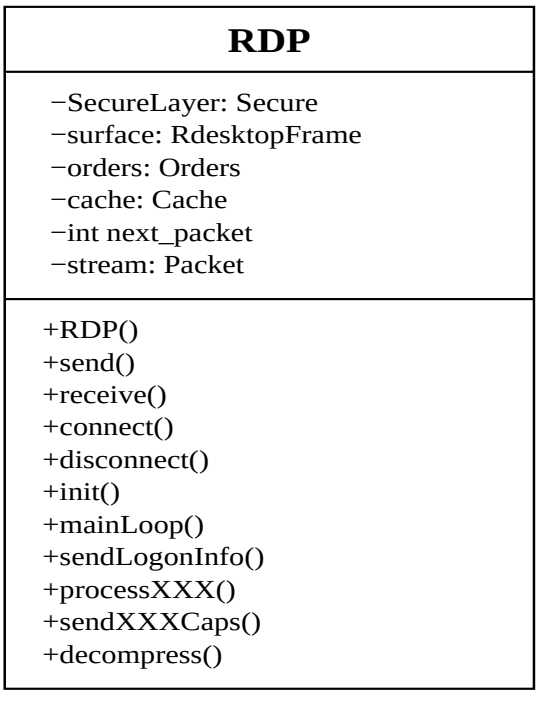


Fig. 4.6 Aufbau der Klasse `RDP`

Für die Zwischenspeicherung und die Darstellung der Daten sind andere Klassen zuständig. Insbesondere die Klassen `Cache`, `Orders` und `RdesktopFrame`. Im Konstruktor der Klasse `RDP` werden Instanzen der entsprechenden Klassen erzeugt.

Verbindungsaufbau

Der Verbindungsaufbau wird, wie in den anderen Klassen der Kommunikationsschicht auch, mit Hilfe der `connect` Methode bewerkstelligt. Zuallererst wird eine Verbindung auf der Sicherheitsschicht aufgebaut. Danach wird ein Logon-Info Paket gesendet (Details zum Verbindungsaufbau findet man in `Kapitel 3.6.2`). Dafür ist die Methode `sendLogonInfo` zuständig. Ein Logon-Info Paket enthält einige spezielle Parameter, die für die Verbindung mit dem Terminal Server gebraucht werden.

War der Verbindungsaufbau erfolgreich muß die Methode `rdpMainLoop` aufgerufen werden. `rdpMainLoop` ist eine Endlosschleife. Sie wird erst beendet wenn ein Fehler aufgetreten ist (also eine Exception geworfen wird) oder die Kommunikation mit dem Server beendet wurde.

Datenaustausch.

`rdpMainLoop` kümmert sich um den Empfang und die Verarbeitung der RDP-Daten. Zunächst wird `receive` aufgerufen um an das nächste RDP-Paket zu gelangen. Das RDP-Protokoll kennt drei Arten von Daten: DEMAND-ACTIVE, DEACTIVATE und DATA. Eine RDP-PDU kann mehrere dieser Datentypen enthalten. In der `receive` Methode wird deshalb mit `next_paket` ein Zeiger verwaltet, der anzeigt, ob in der aktuellen PDU noch Daten enthalten sind.

Als nächstes wird die Art der Daten festgestellt und das Paket an die entsprechenden Hilfsfunktionen weitergegeben. Für Datenpakete ist das `processData`, für Pakete von Typ DEMAND-ACTIVE `processDemandActive`. DEACTIVATE zeigt das Ende einer RDP-Verbindung an und muß deshalb nicht weiterverarbeitet werden.

Ein DEMAND-ACTIVE Paket markiert den Beginn einer Verbindung und wird vom Client mit einer CONFIRM-ACTIVE PDU beantwortet. Die CONFIRM-ACTIVE PDU enthält auch die Capabilities des Clients. `sendConfirmActive` greift dafür auf die `sendXXXCaps` Methoden zurück.

Das RDP Protokoll kennt unterschiedliche Arten von Daten. Die Methode `processData` leitet deshalb das Datenpaket je nach Typ an unterschiedliche Hilfsfunktionen weiter. Um eine UPDATE-PDU kümmert sich `processUpdate`, für eine POINTER-PDU ist `processPointer` verantwortlich. Die BELL-PDU wird nicht verarbeitet weil man unter JX leider nicht auf den Lautsprecher des PCs zugreifen kann. LOGON kann einfach ignoriert werden.

Da es auch wieder unterschiedliche Sorten von UPDATE-PDUs gibt, leitet `processUpdate` die Pakete auch an Hilfsfunktionen weiter. `processOrders` verarbeitet ORDER-Pakete, `processPalette` kümmert sich um Paletteninformationen und `processBitmapUpdates` ist für reine Bitmapdaten zuständig. Handelt es sich um komprimierte Bitmaps werden diese an die Methode `decompress` weitergeleitet und dort dekomprimiert.

`init` erzeugt eine leere RDP-PDU beliebigen Typs, `initData` erzeugt eine leeres Datenpaket. Für das Senden von Daten gibt es ebenfalls zwei Methoden: `send` kann beliebige RDP-PDUs senden, `sendData` ledigl;ich Daten-PDUs.

4.4 Das Grafiksubsystem

Die im vorherigen Kapitel vorgestellten Klassen der Kommunikationsschicht stellen lediglich die Mechanismen zum Senden und Empfangen korrekter RDP-Daten zur Verfügung und kümmern sich um den Verbindungsaufbau. Sie sind jedoch nicht für die Darstellung dieser Daten auf dem Bildschirm verantwortlich. Dies erledigt das Gtafiksubsystem.

Das Grafiksystem besteht aus drei Hauptklassen. Die Klasse `RdesktopFrame` ist für die Darstellung der RDP-Daten auf dem Bildschirm zuständig. Außerdem verarbeitet sie die vom Benutzer getätigten Tastatur- und Mauseingaben. Die Klasse `Cache` ist für die Verwaltung aller Zwischenspeicher zuständig, die zur korrekten Darstellung der RDP-Daten benötigt werden. Die Klasse `Orders` dient der Verarbeitung aller primitiven Zeichenoperationen.

4.4.1 Die Klasse `Orders`

Ein Ziel bei der Entwicklung des RDP-Protokolls lag in der Reduzierung der zu übertragenden Datemmenge. RDP-Clients sollten nicht auf LAN-Verbindungen angewiesen sein, sondern auch über ISDN- oder Modemverbindungen funktionieren. Dies führte zur Entwicklung von Zeichenbefehlen. Anstatt das Ergebnis einer Operation als Bitmap über die Kommunikationsverbindung zu übertragen, werden die Zeichenbefehle selbst übermittelt. Für die Interpretation dieser Zeichenbefehle ist die Klasse `Orders` zuständig.

Eine Orders-PDU enthält immer die Art des Zeichenbefehls sowie alle zur Durchführung dieses Befehls notwendigen Parameter. Diese Parameter unterscheiden sich von Befehl zu Befehl deutlich. Zeichenbefehle können in beliebiger Reihenfolge auftreten. Außerdem kann ein Befehl vom Ergebnis eines vorhergehenden Befehls abhängig sein. Zum Beispiel können die Koordinaten einer Order relativ zu den Koordinaten eines vorherigen Befehls angegeben sein (Del-

ta). Für einige Zeichenbefehle besteht außerdem noch die Möglichkeit Clipping-Regionen zu definieren. Die Parameter aller Zeichenbefehle sowie der aktuelle Zustand bei der Verarbeitung von Befehlen wird in einem Objekt der Klasse OrderState zwischengespeichert.

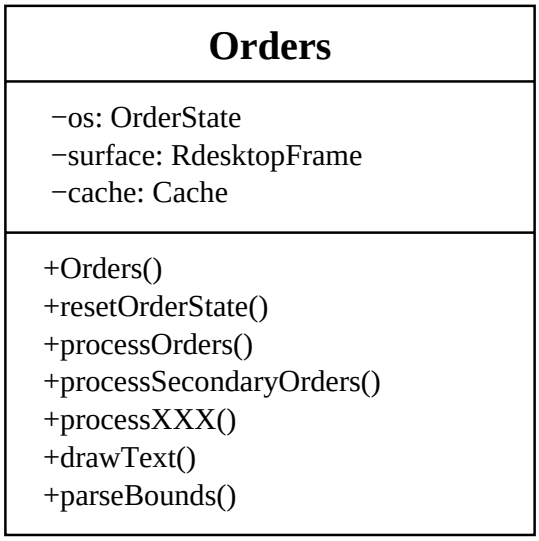


Fig. 4.7 Aufbau der Klasse `Orders`

Die Klasse `OrderState`

Bei der Klasse `OrderState` handelt es sich lediglich um einen Container für die Daten der einzelnen Zeichenoperationen. Ein Objekt der Klasse `OrderState` enthält für jeden Zeichenbefehl ein Datenobjekt, das dessen Parameter beschreibt. Durch die entsprechenden `getXXX()` Methoden kann man eine Referenz auf eines dieser Objekte erhalten um die darin enthaltenen Daten weiterzuverarbeiten oder einzelne Parameter des Objektes zu verändern. Durch den Befehl `reset()` werden alle Datenobjekte in den Urzustand zurückversetzt.

Jedes einzelne Datenobjekt enthält Methoden zum Zugriff auf die in ihm enthaltenen Parameter. Um welche Parameter es sich dabei handelt ist jedoch von Befehl zu Befehl unterschiedlich. Gemeinsam haben alle Klassen jedoch eine `reset()` Methode mit der sie wieder in den Urzustand zurückversetzt werden können.

Befehlsverarbeitung

Die Zeichenbefehle lassen sich in zwei Kategorien einteilen. Primärbefehle (primary orders) und Sekundärbefehle (secondary orders). Primärbefehle sind alle die, die unmittelbar auf dem Bildschirm sichtbar sind. Sekundärbefehle müssen keine unmittelbare Auswirkungen auf die grafische Ausgabe haben. Die Methode `processOrders(Packet data, int next_packet)` ist für die Verarbeitung der Zeichenbefehle zuständig. Sie ermittelt die An-

zahl sowie die Art der Zeichenbefehle und delegiert die Verarbeitung dieser Befehle an entsprechende Hilfsfunktionen. Handelt es sich um einen Sekundärbefehl, so wird die Kontrolle an die Funktion `processSecondaryOrders(Packet data)` weitergeleitet.

Für jeden Zeichenbefehl existiert eine eigene Verarbeitungsfunktion. Diese Verarbeitungsfunktionen sind für die Aufbereitung der Parameter und die Darstellung auf dem Bildschirm zuständig. Sekundärbefehle arbeiten hauptsächlich auf Daten die sich in einem der zahlreichen Cachespeicher befinden. Ein Objekt der Klasse `OrderState` benötigt deshalb die Möglichkeit auf den Bildschirm zu zeichnen oder auf den Cache zuzugreifen.Die entsprechenden Objekte können mit Hilfe der Methoden `registerCache(Cache cache)` beziehungsweise `registerDrawingSurface(RdesktopFrame surface)` registriert werden.

Einige der im Rdesktop Protokoll festgelegten Zeichenoperationen sind sehr aufwändig und erfordern spezielle Funktionen, die die grafische Oberfläche des Betriebssystems bereitstellen muß. JX implementiert allerdings nur die grundlegenden Funktionen, die zur Darstellung von Grafiken notwendig sind. Der Client unterstützt deshalb nur eine Teilmenge der Zeichenoperationen. Es wurden lediglich die besonders häufig auftretenden Befehle implementiert.

4.4.2 Verwaltung der Zwischenspeicher

Das RDP-Protokoll verwendet Cachespeicher zur Speicherung häufig verwendeter Daten. Die Zwischenspeicherung geschieht einerseits um die Menge der zu übermittelnden Daten zu begrenzen, ermöglicht auf der anderen Seite aber auch erst bestimmte Funktionalität. Die Klasse `Cache` kapselt den Zugriff auf alle Zwischenspeicher.

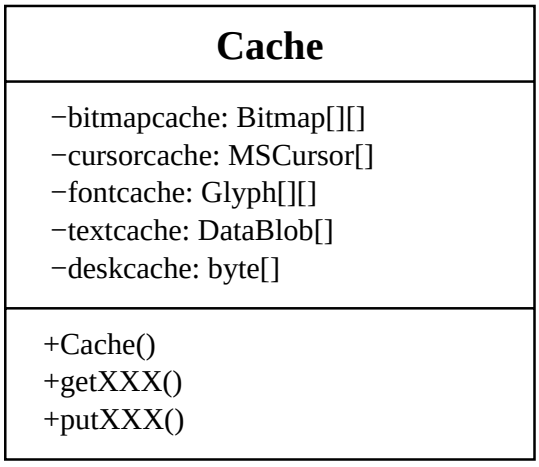


Fig. 4.8 Aufbau der Klasse `Cache`

Das RDP-Protokoll kennt fünf Zwischenspeicher:

- Fontcache: Nimmt eine bestimmte Anzahl von Zeichensätzen auf. Ein Zeichensatz besteht aus einem oder mehreren Buchstaben. Die Buchstaben werden durch spezielle Bitmaps repräsentiert

- Textcache: Der Textcache speichert besonders häufig verwendete Textteile. Ein Textteil besteht aus einem String der die Indizes einzelner Buchstaben aus dem Fontcache sowie Formatierungsoptionen enthält.
- Cursorcache: Der Cursorcache speichert alle Windows-spezifischen Mauszeiger
- Deskcache: Der Deskcache wird bei der Desksave Operation verwendet. Er enthält besonders häufig verwendete Darstellungen des aktuellen Desktops. So kann zu einer früheren "Aufnahme" des Desktops geschaltet werden.
- Bitmapcache: Einige Zeichenbefehle verwenden zusätzliche Bitmaps. Diese können im Bitmapcache gespeichert werden.

Zu jedem Zwischenspeicher gehört ein eigenes Objekt, das die entsprechenden Daten aufnimmt. Die Caches selbst werden durch Arrays dieser Objekte repräsentiert und besitzen eine feste Größe. Da der Client die Größe der Zwischenspeicher beim Verbindungsaufbau selbst bestimmt ist das völlig ausreichend. Mit `putXXX` können die Datenobjekte im Cache platziert werden. Mit `getXXX` werden sie aus dem Cache gelesen.

Will man an einer Position des Caches Daten lesen, an der keine vorhanden sind, wird eine Exception generiert.

4.4.3 Bildschirmdarstellung und Verarbeitung von Eingaben

Ein RDP-Client stellt für den Benutzer einen virtuellen Desktop dar. Der Anwender soll das Gefühl haben, dass alle Anwendungen lokal auf seinem Rechner laufen. Ein RDP-Client muß deshalb die grafischen Ausgaben des Servers auf dem Bildschirm des Anwenders darstellen und die Eingaben des Benutzers verarbeiten. Für diese Aufgaben ist die Klasse `RdesktopFrame` zuständig.

Die Klasse `RdesktopFrame` stellt die von der RDP-Schicht bereitgestellten Daten auf dem Bildschirm dar und leitet die Tastatureingaben und Mausbewegungen als RDP-Daten an den Terminal Server weiter. Da sich zum Zeitpunkt dieser Arbeit eine native AWT Implementierung noch in der Entwicklung befand, basiert die grafische Darstellung nur auf Klassen und Methoden, die der JX-eigene Windowmanager bereitstellt (Details zum Windowmanager kann man in [OBE02] nachlesen).

Grafische Darstellung

Die Klasse `RdesktopFrame` leitet sich von `WWindow` ab. `WWindow` stellt ein einfaches Fenster mit den Standard-Fensterdekorationen auf dem Bildschirm dar. Die Größe des Fensterinnenbereichs entspricht dabei genau der Größe des virtuellen Desktops. Die Größe des Fensters wird beim Erzeugen des Frames festgelegt. Da sich diese Größe während der gesamten Laufzeit des Programmes nicht verändert wird es auch als nicht veränderbar gekennzeichnet.

Für die eigentliche Darstellung des Fensterinhaltes ist jedes Fenster selbst verantwortlich. Der Frame muß deshalb sämtliche Daten, die er zur Darstellung benötigt, selbst speichern. Dies geschieht mit Hilfe eines Memory-Objektes. Das Memory-Objekt `backstore` enthält den gesamten Fensterinhalt, sämtliche Zeichenoperationen verändern lediglich die Daten dieses Objektes. Wird vom Windowmanager die `paint` Methode des Fensters aufgerufen, so wird der Inhalt des Memory-Objektes im Innenbereich des Fensters dargestellt. Dazu wird ein Objekt vom Typ `WBitmap` erzeugt, dem das Memory-Objekt übergeben wird. Diese Technik nennt man auch Double Buffering.

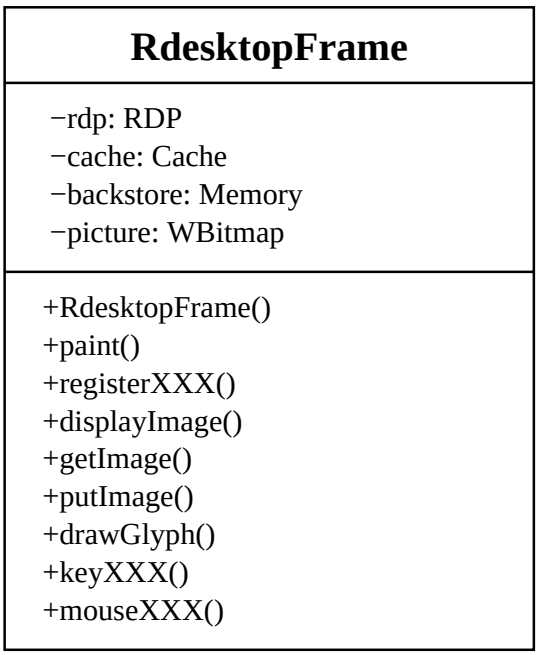


Fig. 4.9 Aufbau der Klasse `RdesktopFrame`

Die Klasse `WBitmap` kann mit den unterschiedlichsten Farbräumen umgehen. Leider kann sie jedoch nicht mit selbstdefinierten Paletten ausgestattet werden. Die Daten im Memory-Objekt liegen deshalb im 32-Bit True Color Modus vor. Jeder Pixel belegt so vier Byte, eines für jeden Farbanteil sowie eines für den Transparenzwert. Der Nachteil liegt im relativ grossen Speicherverbrauch. Ein Desktop mit 1024 x 768 Pixeln belegt circa 3 Megabyte im Hauptspeicher.

Die Zeichenfunktionen, wie zum Beispiel `displayImage` oder `fillRectangle` rechnen die Daten selbst vom 8-Bit ins 32-Bit Farbmodell um. Dazu benötigen sie die aktuelle Palette, die als Integer-Array `colormap` vorliegt und mit der Funktion `registerPalette` festgelegt werden kann.

Tastatur und Maus

Die Klasse `WWindow` stellt Methoden bereit, mit deren Hilfe sich jede Anwendung selbst um die Verarbeitung von Tastatur- oder Mauseingaben kümmern kann. `mouseDown, mouseUp, mouseMoved, keyDown` und `keyUp` sind dafür zuständig. Die Eingaben werden mit Hilfe der `sendInput` Methode der Klasse `RDP` an den Server übermittelt.

Bei den Mauseingaben ist die Vorgehensweise relativ einfach. Die Methoden `mouseDown, mouseUp` und `mouseMoved` liefern die Position an der die Eingabe stattfand sowie welcher Mausknopf gerade gedrückt wurde. Das sind alle Angaben die notwendig sind um ein Mausereignis zu beschreiben. Bei der Verarbeitung von Tastaturereignissen ist etwas mehr Aufwand erforderlich.

Der Windowmanager von JX beschreibt, ähnlich wie das AWT bei Java, alle Zeichen durch spezielle Codes, die für jedes mögliche Zeichen (dazu gehören auch Modifikatortasten wie ALT oder CONTROL) eindeutig sind. Das RDP-Protokoll auf der anderen Seite, übertragt lediglich den Code der Taste, die gerade gedrückt wurde. Der Buchstabe, der dieser Taste zugeordnet ist, kann jedoch je nach verwendetem Keyboard unterschiedlich sein. Auf einer deutschen Tastatur beispielsweise liegt dort, wo bei einer US-Amerikanischen Tastatur der Doppelpunkt sitzt, das “ä”. Die Art des angeschlossenen Keyboards wird dem Server beim Verbindungsaufbau mitgeteilt. Es muß demzufolge eine Umsetzung erfolgen, die ein Zeichen auf die jeweilige Taste im RDP-Protokoll abbildet. Dies ist Aufgabe der Klassen `KeyCode` und `Keylayout.`

Die Klasse `Keylayout` enthält nur einzige Methode nämlich `getLayout.` Diese Methode kennt als einzigen Parameter die Identifikationsnummer der entsprechenden Tastatur und liefert ein Objekt vom Typ `KeyCode` zurück. Momentan werden circa dreissig verschiedene Tastaturen unterstützt, unter anderem kyrillisch und türkisch. Die Methode `getScancode` in der Klasse `KeyCode` übernimmt schließlich die Umsetzung der Zeichen in die entsprechenden Scancodes einer MFII-kompatiblen Tastatur mit 104/105 Tasten.

Das RDP Protokoll sieht Zeitstempel vor um die einzelnen Events zeitlich zu ordnen. Im Gegensatz zur AWT verzichtet der JX Windowmanager jedoch auf die Generierung von Zeitstempeln für seine Events. Die Klasse `RdesktopFrame` enthält deshalb auch eine Funktion um eindeutige Zeitstempel zu erzeugen. `getTime` liefert die laufende Nummer als Integer zurück.

4.5 Leistungsmessungen

Performancemessungen an Thin-Client Systemen sind erfahrungsgemäß schwieriger zu realisieren als Messungen an normalen Rechnern. In Thin-Client Systemen geschieht die Ausführung einer Anwendung auf dem Server, die grafische Ausgabe der Ergebnisse findet aber auf einem Client statt, der über ein Netzwerk mit dem Server verbunden ist. Die Darstellung auf dem Client ist also von der Ausführung der Anwendung auf dem Server entkoppelt.

Um die Performanz des Bildschirmaufbaus auf den Clients zu verbessern werden beim RDP-Protokoll drei Arten von Techniken verwendet, nämlich Datenkompression, Caching und Merging. Durch die Anwendung gängiger Kompressionstechniken, wie Lauflängenkodierung oder

LZW, auf die grafischen Daten, kann der Bandbreitenverbrauch sinnvoll reduziert werden. Zwischenspeicher ermöglichen dem Client den lokalen Zugriff auf häufig wiederkehrende Daten, so dass er sie nicht ständig wieder neu anfordern muß. Merging schließlich ist eine Technik, die es dem Server erlaubt mehrere Display-Updates zu einem einzigen zusammenzufassen. Durch das Zusammenfassen von Bildschirmupdates wird die Darstellung auf dem Client von der eigentlichen Ausführung der Anwendung auf den Server entkoppelt. Kann ein Client die Updates nicht schnell genug verarbeiten, dann werden einzelne Updates zusammengefasst oder gehen ganz verloren.

Durch Einsatz dieser Techniken ist die Ausführungsgeschwindigkeit einer Anwendung auf dem Server nicht durch den langsamen Bildschirmaufbau auf dem Client begrenzt. Allerdings bedeutet dies auch das gängige Messtechniken bei der Bewertung der Leistung eines Thin-Client Systems nicht greifen.

4.5.1 Versuchsaufbau

Die Leistungsfähigkeit eines RDP-Clients hängt besonders davon ab, wie gut die grafische Darstellung funktioniert. Kann ein Client zu wenige Updates in einer Sekunde verarbeiten, dann wird der Benutzer in seiner Arbeit beeinträchtigt.

Eine Möglichkeit die Leistungsfähigkeit eines Thin-Client Systems zu messen ist, einen Standard Benchmark ablaufen zu lassen. Da die Darstellung der Anwendung auf dem Client aber von der Ausführung auf dem Server entkoppelt ist, gibt ein solcher Benchmark keinen Aufschluss darüber, wie gut die grafische Darstellung und damit die “gefühlte” Leistung auf der Seite des Clients ist.

Eine zweite Möglichkeit besteht darin, den Client selbst zu modifizieren und geeignete Messfunktionen zu integrieren. Man kann damit zum Beispiel alle Ein- und Ausgaben protokollieren und damit sehr detaillierte Informationen über die Kommunikation zwischen Client und Server gewinnen. Allerdings fehlt die Möglichkeit um aus diesen rohen Daten auf die übertragenen Informationen zu schließen. Hat man zum Beispiel ein Video abgespielt, so lässt sich aus den protokollierten Daten nur sehr schwer auf die einzelnen Frames des Videos schließen, da die Daten die oben genannten Optimierungsmethoden unterlaufen haben.

Eine dritte Möglichkeit besteht darin, die Kommunikation zwischen Client und Server zu messen, in dem man die über das Netzwerk übermittelten Pakete protokolliert. Diese Methode lässt Rückschlüsse auf die übertragene Datenmenge und die Latenz zwischen Display Updates zu.Allerdings besteht auch hier das Problem, dass die übertragene Datenmenge unter umständen geringer wird, wenn der Client mit der Darstellung der Daten nicht hinterherkommt.

Am Department of Computer Science der Columbia University wurde deshalb eine Technik namens “Slow Motion Benchmarking” entwickelt. Eine genaue Beschreibung des Messverfahrens lässt sich in [YNN01] oder [YNS02] finden. Bei Slow-Motion-Benchmarking werden auch Packet Traces verwendet um die Menge an übertragenen Daten sowie die Latenz zwischen Display Updates zu messen. Die verwendeten Benchmarks werden jedoch so abgeändert, dass sicherge-

stellt ist das jedes visuelle Ereignis auf dem Client auch dargestellt wurde, bevor ein neues auf dem Server verarbeitet wird. Dies geschieht durch das Einfügen von Pausen zwischen die einzelnen visuellen Ereignisse. Der Benchmark läuft sozusagen in Zeitlupe ab.

Da die Benchmarks in Zeitlupe ablaufen ist die Wahrscheinlichkeit geringer, dass mehrere Bildschirmupdates zu einem zusammengefasst werden. Ist der Intervall zwischen zwei Aktualisierungen groß genug gewählt, so kann davon ausgegangen werden das Merging nicht vorkommt. Die Ergebnisse aus dem Lauf in Zeitlupe werden nun mit den Ergebnissen verglichen, die man bei einem normalen Durchlauf des Benchmarks gewonnen hat. So lässt sich feststellen, wie viele Updates bei einem normalen Durchlauf verloren gehen.Man erhält somit ein Maß für die Darstellungsqualität auf dem Client.

Der im Rahmen dieser Studienarbeit verwendete Benchmark besteht aus dem Abspielen eines AVI-Videoclips im Windows Media Player. Der Videoclip hat eine Länge von 30 Sekunden und enthält insgesamt 897 Einzelbilder im Format 352x240 Pixel. Die Größe der Videodatei liegt bei 216 MB.Die ideale Bildwiederholrate des Clips liegt bei 29,97 Bildern pro Sekunde, da es sich um NTSC-Bildmaterial handelt. Um Referenzwerte zu erhalten, wurde die Bildwiederholrate soweit reduziert, dass sichergestellt ist das jedes Einzelbild auch vollständig auf dem Client dargestellt werden konnte bevor der Server ein Neues sendet. Die Messungen haben gezeigt, dass dies der Fall ist wenn die Bildwiederholrate bei einem Einzelbild pro Sekunde liegt.

Lässt man nun die gleiche Videodatei mit der normalen Bildrate von 29,97 Bildern pro Sekunde laufen, so kann man die übertragenen Datenmengen vergleichen und Rückschlüsse auf die Bildqualität des Clients ziehen. Die Formel hierzu lautet:

$$BQ(P) = \frac{\left\lceil \frac{(DataTransferred(P))/(PlaybackTime(P))}{IdealFPS(P)} \right\rceil}{\left\lceil \frac{(DataTransferred(slow-mo))/(PlaybackTime(slow-mo))}{IdealFPS(slow-mo)} \right\rceil}$$

Hierbei bezeichnet P die Bildwiederholrate mit der die Videodatei abgespielt wird und slow-mo die Bildwiederholrate der Slow-Motion Version.

Messungen der Bildqualität wurden bei zwei, vier, acht, zwölf, sechzehn, zwanzig, vierundzwanzig und 29,97 Bildern pro Sekunde durchgeführt. Es wurden insgesamt vier verschiedene RDP-Clients miteinander verglichen. Es handelt sich hierbei um Rdesktop für Linux, die Java-Implementierung des JX-Clients, die native Implementierung für das JX-Betriebssystem sowie zum Vergleich Microsofts eigenes RDP-Tool. Als Client Rechner kam ein Pentium III mit 500 MHz und 256 MB Speicher zum Einsatz. Sowohl Rdesktop, als auch die Java-Implementierung des JX RDP-Clients wurden unter SuSE Linux 7.3 getestet. Microsofts RDP-Client lief auf einem Windows 2000 mit Service Pack 2.

Beim Windows 2000 Terminal Server handelt es sich um einen Dual Pentium III Rechner mit 500 MHz Taktfrequenz. Der Server besitzt 256 MB Speicher. Client und Server sind über ein 100 MBit-Hub miteinander verbunden. Als Paketmonitor kam ein Pentium III 500 zum Einsatz auf dem Caldera Open Linux lief. Als Software kam TCPdump zum Einsatz. Bei allen Programmen wurde das Caching von Daten abgestellt, um die Ergebnisse nicht zu verfälschen.

4.5.2 Ergebnisse

Lässt man den Benchmark mit einer Bildwiederholrate von einem Bild pro Sekunde laufen, so liegen alle vier Client-Systeme gleich auf. Daran zeigt sich das die Aktualisierungsrate langsam genug gewählt wurde, so dass auf allen vier Plattformen die Display Updates rechtzeitig abgeschlossen werden konnten. (siehe Abbildung 4.10) Auffällig ist lediglich, dass der JX-Client mehr als doppelt so viele Pakete empfängt, als alle anderen Systeme.

Vergleicht man die durchschnittliche Größe eines Datenpaketes auf der IP-Schicht miteinander so zeigt sich, dass unter JX die Pakete weniger als halb so groß sind, als auf den anderen Plattformen. Dies führt dazu, dass bei gleicher Datenmenge nahezu die doppelte Anzahl an Paketen übertragen werden muß. (siehe Abbildung 4.12)

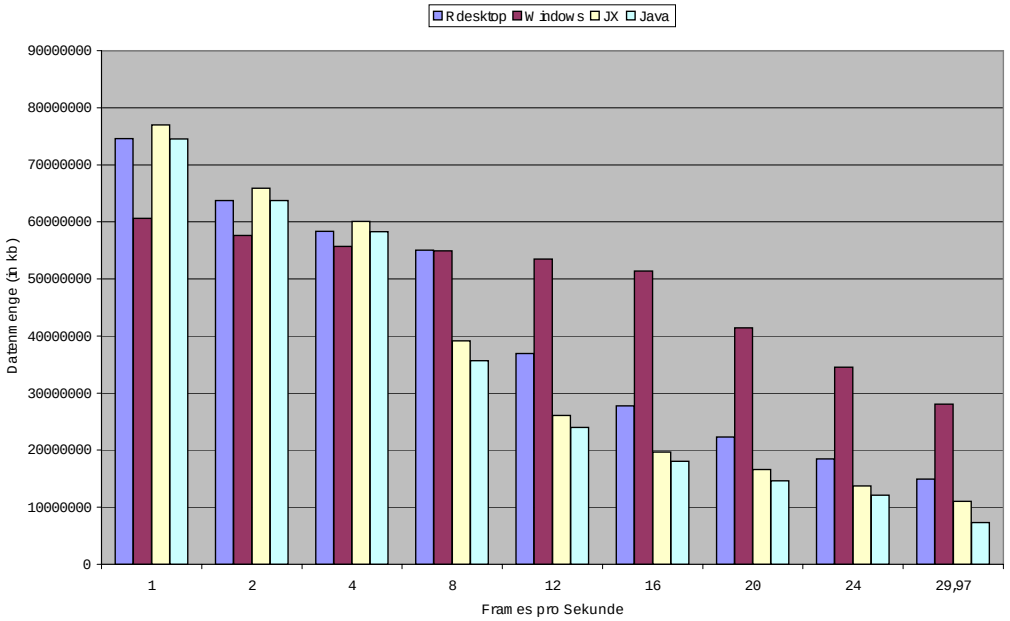


Fig. 4.10 Übertragene Datenmenge

Erhöht man die Bildwiederholrate auf zwei beziehungsweise vier Bilder die Sekunde, so sind die Unterschiede zwischen den Rdesktop basierenden Systemen und der Referenz von Microsoft immer noch relativ gering. Der Unterschied in der Bildqualität beträgt nur ca. zehn Prozent, was im Betrieb eigentlich nicht weiter auffallen sollte.(siehe Abbildung 4.11)

Die drei Rdesktop Systeme liegen jedoch immer noch gleichauf. Erst bei einer weiteren Erhöhung der Bildrate zeigen sich die ersten Unterschiede. Während die Microsoft Implementierung selbst bei sechzehn Bildern pro Sekunde noch eine Bildqualität von über 80 Prozent halten kann brechen sowohl der Linux Client als auch die beiden Java-basierten Systeme sehr stark ein und erreichen noch nicht einmal die halbe Leistung des Microsoft-Clients.

Rdesktop unter Linux kann sich bei acht Bildern pro Sekunde noch etwas von dem JX-Client und der Java-Implementierung absetzen und erreicht um etwa 40 Prozent bessere Werte. Er fällt aber bei höheren Bildraten auch auf das Niveau der anderen beiden Implementierungen zuück.An die Leistungsfähigkeit des Microsoft Clients reicht jedoch keine der anderen Client Implementierungen heran

Bei hohen Bildwiederholraten zwischen 16 und 20Bildern pro Sekunde leidet die Bildqualität aller Clients sehr stark. Selbst die Microsoft Implementierung erreicht bei 29,97 Bildern pro Sekunde (was der Bildwiederholrate von NTSC-Video entspricht) nur noch 40% der Bildqualität die theoretisch möglich wäre. Auffällig ist, dass Rdesktop unter Linux etwa zwanzig Prozent besser abschneidet als die JX-Version.

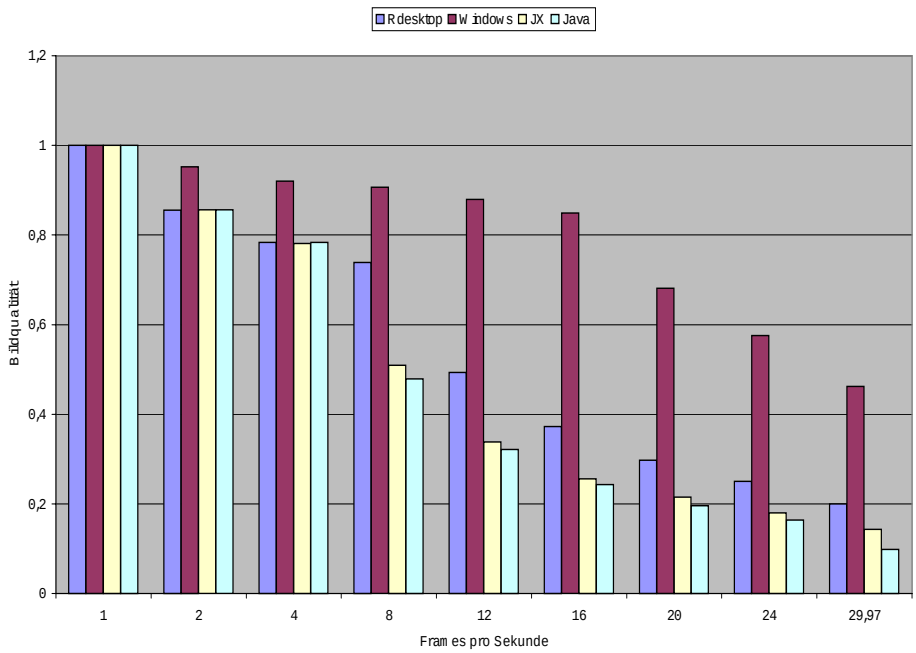


Fig. 4.11 Bildqualität

Ein großer Teil der Anwendungen unter Windows verlangen allerdings nicht nach einer sehr hohen Bildaktualisierungsrate. Bei Standardanwendungen, wie Web Browsing oder Textverarbeitung, gibt es keine spürbaren Unterschiede zwischen der JX-Variante und der Linux Implementierung. Das RDP nicht auf die Übermittlung von Multimedia-Daten hin ausgerichtet worden ist, zeigt sich auch darin, dass die Bildqualität auch bei der Windows Variante des RDP-Clients bei hohen Bildraten sehr stark nachlässt.

Die JX-Portierung von Rdesktop erreicht also, je nach Belastung, zwischen 60 und 100 Prozent der Leistung, die das Linux Pedant vorweisen kann. Dies deckt sich mit Ergebnissen anderer Arbeiten. (zum Beispiel [ID02])

Es handelt sich bei JX allerdings um ein Betriebssystem das sich noch sehr stark in der Entwicklung befindet. Der TCP-Stack der Grundlage für die Netzwerkmessungen war, wurde im Rahmen eines zweiwöchigen Seminars entwickelt und hat das Beta-Stadium zum Zeitpunkt als diese Arbeit abgeschlossen war noch nicht verlassen. Der verwendete Windowmanager muß auf

hardwarebeschleunigte grafische Routinen verzichten und deshalb alle Operationen in einer Software-Emulation ablaufen lassen. Doch auch am JX-Grundsystem sind sicher noch einige Optimierungen möglich. Zuguterletzt gibt es sicherlich auch in der JX-Implementierung des Clients noch einige Verbesserungsmöglichkeiten.

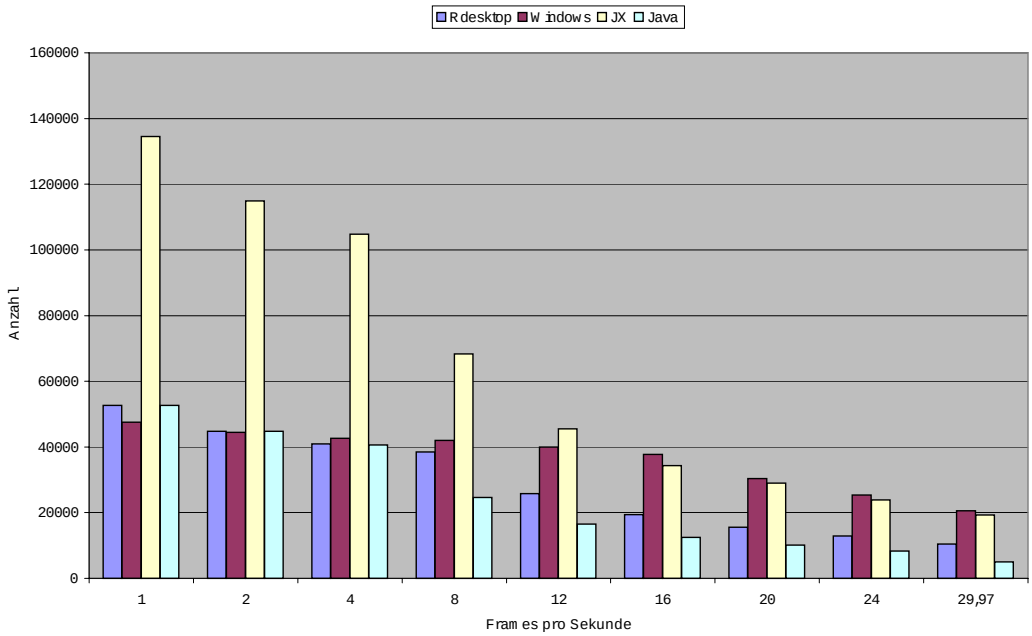


Fig. 4.12 Anzahl der übermittelten Datenpakete

4.6 Zusammenfassung

Dieses Kapitel gab eine umfassende Übersicht über die Java Implementierung des RDP-Clients. Zuerst wurde der allgemeine Aufbau des Clients erläutert danach wurden die wichtigsten Klassen des Programms, ausführlicher behandelt.das Der letzte Abschnitt befasste sich mit der Leistungsbewertung des JX-Clients und den Vergleich mit anderen RDP-Clientprogrammen.

Dabei zeigt sich, dass die Implementierung des RDP-Protokolls für das JX-Betriebssystems je nach Anwendungsfall zwischen 60 und 100 Prozent der Leistung der Linux-Version erreicht.

5 Zusammenfassung und Ausblick

5.1 Einleitung

Dieses Kapitel fasst noch einmal kurz die Ergebnisse der Arbeit zusammen. Außerdem werden einige Probleme, die bei der Ausfertigung der Arbeit aufgetreten sind, aufgeführt sowie ein Überblick über mögliche Verbesserungen gegeben.

5.2 Ergebnisse

Der JX RDP-Client ist nahezu vollständig, bietet jedoch noch einigen Raum für Erweiterungen oder Verbesserungen. Die Protokollversion Vier wird bis auf 128-Bit Verschlüsselung vollständig unterstützt. Der Client erlaubt somit Verbindungen zu Windows NT 4.0 Systemen sowie Windows 2000 oder Windows XP-Systemen, sofern sie nicht auf hohe Verschlüsselungsstärke eingestellt sind.

Bereits bei der Implementierung wurde auf eine klare Aufteilung der Funktionalität geachtet. Dies spiegelt sich vor allem in der Trennung der Übermittlung von der Darstellung der Daten wieder. Außerdem wurden auch für noch nicht unterstütze Funktionen bereits Schnittstellen im System integriert, was die Implementierungsarbeit erleichtern sollte.

Verbesserungswürdig ist vor allem die grafische Darstellung der RDP-Daten. Besonders die Funktionen zur Darstellung von Text bremsen das System unnötig aus. Aber auch im Protokollstack sind sicherlich noch einige Verbesserungen möglich.

Alles in allem zeigt sich allerdings, das unter JX auch komplexe Anwendungen möglich sind. Die Leistungsfähigkeit hängt zwar noch hinter einer Implementierung in C oder C++ zurück, allerdings kann man mit der momentanen Implementierung in Java bereits relativ gut arbeiten..Wie die Messungen gezeigt haben liegt die Leistung des JX-Clients nur etwa 20 bis 40 Prozent unter der Performanz des Linux Programms auf dem er basiert. Diese Leistungsunterschiede sollten sich durch weitere Optimierungen des Clients sowie des TCP-Stacks eigentlich weiter reduzieren lassen. Der JX TCP-Stack befand sich zum Zeitpunkt als diese Arbeit abgeschlossen war noch in einem recht frühen Stadium der Entwicklung.

5.3 Probleme bei der Arbeit

Eine erste Implementierung des RDP-Clients entstand auf einer Sun Ultra 1 unter einem Sun Java-Development-Kit 1.1.8. Diese Implementierung enthielt bereits einige grundlegende JX-Klassen, wie zum Beispiel die Memory-Objekte, die auch in einer “Java-Emulation” vorliegen. So konnte auf ein JX-Entwicklungssystem verzichtet werden.

Die größte Schwierigkeit bei dieser Implementierung bestand darin, dass Tests immer nur an bestimmten Stellen der Protokollhierarchie möglich waren. Circa achtzig Prozent der Programmlogik sind für den Verbindungsaufbau zuständig. Erst wenn wieder ein Abschnitt des Verbindungsaufbaus einer Schicht implementiert war, konnte wieder getestet werden. Ein zweites Problem bestand darin, dass am Terminal Server selbst keine Tests durchgeführt werden konnten, da sich der Server nicht in einen Debug-Modus versetzen lässt. Da die RDP-Daten verschlüsselt sind, waren auch Werkzeuge zum Abhören einer Verbindung (z.B. TCPdump) relativ nutzlos. Die Kommunikation des Java-Clients musste also ständig mit der des Linux-Clients verglichen werden, was mit einem großen Zeitaufwand verbunden war. Zu diesem Zweck musste außerdem die Verschlüsselungsroutine des Linux-Clients angepasst werden, so dass beide Clients auch nach erfolgter Verschlüsselung noch die gleichen daten liefern.

Der zweite Teil der Arbeit bestand in der Portierung des Java-Clients auf das JX-Betriebssystem. Ein Problem bei der Portierung lag darin, das JX noch keine hundertprozentige JDK1.1.8 Kompatibilität bietet. Zum einen sind einige Klassen noch nicht implementiert oder nur ungenügend getestet, zum anderen ist an einigen Stellen die API unterschiedlich zu der im Java Development Kit.

Eine weitere Schwierigkeit liegt darin begründet, das an JX noch entwickelt wird. So können sich Teile des Betriebssystem und der zugrundeliegenden API durchaus im Laufe eines Projektes noch ändern, was auch umfangreiche Anpassungen am eigenen Projekt nach sich ziehen kann. Dies erschwert auch das Testen, da einige Teile des Systems noch mit gewissen Instabilitäten zu kämpfen haben.

5.4 Weiterführende Arbeiten

Die Implementierung des RDP-Clients war relativ aufwändig, deshalb musste schon allein aus Zeitgründen auf einige Funktionalität verzichtet werden. So wurden nur die wichtigsten Zeichenoperationen implementiert. Die restlichen Befehle könnten zukünftig auch noch in den Client integriert werden, die Schnittstellen dafür sind bereits vorhanden. Allerdings sind einige dieser Operationen recht rechenzeitintensiv und wohl nur im Zusammenspiel mit der Grafikhardware effizient zu realisieren.

6 Literaturverzeichnis

CCM99 Brian Craig Cumberland, Gavin Carius, Andrew Muir: *Microsoft Windows NT Server 4.0 Terminal Server - Technical Reference.* Microsoft Press, München, 1999

COM Compaq: *Evo Thin Client.* http://www.compaq.com/products/thinclients/index.html

EJ01 D. Eastlake 3rd, P. Jones: *Request for Comments 3174 - US Secure Hash Algorithm 1.* National Institute for Standards and Technology, 2001

GFW02 Michael Golm, Meik Felser, Christian Wawersich: *The JX Operating System.* Proceedings of the USENIX Annual Technical Conference, Monterey, 2002

GKB01 Michael Golm, Jürgen Kleinöder, Frank Bellosa: *Beyond Address Spaces - Flexibility, Performance, Protection, and Resource Management in the Type-Safe JX Operating System.* Technical Report TR-14-01-04, Universität Erlangen, 2001

IBM International Business Machines: *NetVista Thin Clients.* http://www.pc.ibm.com/ww/netvista/thinclient

ID02 Ivan Dedinski: *Entwurf und Implementierung eines hochperformantenen Datenbanksystems für das Java-Betriebssystem JX.* Studienarbeit SA-I4-2002-01, Universität Erlangen, 2002

ISO84 International Standards Organisation: *Request for Comments 905 - ISO Transport protocol Specification.* International Standards Organisation, 1984

ITU98a ITU-T Study Group 16: *ITU-T Recommendation T.122 - Multipoint Communication Service - Service Definition.* International Telecommunication Union, Genf, 1998

ITU98b ITU-T Study Group 16: *ITU-T Recommendation T.125 - Multipoint Communication Service - Protocol Specification.* International Telecommunication Union, Genf, 1998

ITU98c ITU-T Study Group 16: *ITU-T Recommendation T.128 - Multipoint Application Sharing.* International Telecommunication Union, Genf, 1998

KNU99 Jonathan Knudsen: *Java 2D Graphics.* O'Reilly and Associates, Sebastopol, 1999

OBE02 Jürgen Obernolte: *Entwurf und Implementierung eines Windowmanagers für das Java-Betriebssystem JX.* Studienarbeit SA-I4-2002-04, Universität Erlangen, 2002

ORA Oracle Systems:http://www.oracle.com.

PY97 Y. Pouffary, A. Young: *Request for Comments 2126 - ISO Transport Service on Top of TCP.* Digital Equipment Corporation, 1997

RDE Matt Chapman: *Rdesktop - A Remote Desktop Protocol Client.* http://www.rdesktop.org

RIV92 Ronald L. Rivest: *Request for Comments 1321 - The MD5 Message Digest Algorithm,* RSA Data Security Inc., 1992

SCH95 Bruce Schneier: *Applied Cryptography.*John Wiley and Sons Inc., 1995

SM00 Joseph T. Sinclair, Mark Merkow: *Thin Clients clearly explained.* Morgan Kaufmann, San Francisco, 2000

SUN Sun Microsystems: *Sun Ray Integrated Solution.* http://www.sun.com/products-n-solutions/hardware/infoappliances.html

TAN00 Andrew S. Tanenbaum: *Computernetzwerke, Dritte revidierte Auflage.* Pearson Studium, München, 2000

TOG The Open Group: *The X-Window System.* http://www.x.org

YNN01 S. Jae Yang, Jason Nieh, Naomi Novik: *Measuring Thin-Client Performance Using Slow-Motion Benchmarking.* Proceedings of the USENIX Annual Technical Conference, Boston, 2001

YNS02 S. Jae Yang, Jason Nieh, Matt Selsky: *The Performance of Remote Display Mechanisms for Thin Client Computing.* Proceedings of the USENIX Annual Technical Conference, Monterey, 2002