Inhaltsverzeichnis

Zusammenfassung

Ziel der vorliegenden Arbeit war die Entwicklung einer plattformunabhängigen Netzwerkmanagement-Lösung mit Fokus auf Integrierbarkeit, Transparenz und Dynamik mit Rücksicht auf Handhabung und Wartbarkeit. Besondere Aufmerksamkeit wurde dem Einsatz in WLAN-Umgebungen gewidmet, wodurch die entwickelte Lösung Funktionen für Handover und Roaming beinhaltet. Die Arbeit gliedert sich in eine Einleitung, der Technik, der Implementierung, dem Testen und dem praktischen Einsatz der IronGate genannten Software. Im Abschnitt Einleitung wird der grundsätzliche Aufbau von IronGate nähergebracht, die Technik erklärt die Funktionsweise des Systems, die Implementierung befasst sich mit der programmtechnischen Umsetzung der beschriebenen Algorithmen und das Testen und der praktische Einsatz zeigen das Verhalten des Systems in Extremsituationen und im Echtbetrieb.

Abstract

This work describes the developement of a platform-independent network-management solution focussing on integrableness, transparency and dynamic behavior considering handling and maintenance. A primary goal of the work was the employment in WLAN-environments resulting in routines for handover and roaming. The document consists of an introduction, the technology, the implementation, the testing and the practical usability of the solution called IronGate. The introduction describes the basic concept of IronGate, the chapter technology describes it's algorithms and functionality, the chapter implementation focusses on the software-design of IronGate where the chapters testing and practical usability show the system's behavior in extreme situations and production environments.

Danksagungen

Besonderer Dank gebührt an dieser Stelle Dipl.-Ing. Reinhard Fleck vom Institut für Breitbandkommunikation für seine stetige Unterstützung während der Durchführung der Arbeit, wodurch dieses Projekt überhaupt erst ermöglicht wurde.

Großer Dank gebührt Dr. Dipl.-Ing. Norbert Jordan vom Institut für Breitbandkommunikation für seine Hilfsbereitschaft und für die Mitbetreuung der Arbeit sowie auch Hr. Michael Stahl für seine Unterstützung bei der Implementierung von IronGate.

1 Einleitung

Dieses Kapitel behandelt den Grundgedanken hinter IronGate als AAA-Managementsoftware näher und stellt Irongate anderen Produkten der Netzwerkverwaltungssparte gegenüber.

1.1 Zusammenfassung

IronGate ist eine plattformunabhängige AAA-Netzwerkmanagementlösung, deren Grundidee es ist, die übliche Idee der Netzwerküberwachung und -absicherung umzudrehen, und den Gefahren und Missbräuchen von interner Seite entgegenzuwirken. Die Hauptaugenmerke liegen dabei auf Offenheit gegenüber anderen Systemen, Transparenz und Dynamik in Kombination mit einer einfachen Handhabung und Wartbarkeit. Dabei wurde besondere Aufmerksamkeit dem Einsatz in WLAN-Umgebungen gewidmet, wodurch IronGate Funktionen für Handover und Roaming beinhaltet. Neben der Kontrolle der einzelnen Pakete wurden auch Funktionen für Traffic-Shaping, Traffic-Priorizing, Zeit- und Volumenaccounting, Zeitbeschränkungen, gruppenweise Benutzerverwaltung und eine Heuristic Usage Detection implementiert, was IronGate zu einer umfassenden Netzwerküberwachungssoftware macht und ein vollwertiges Accounting nach Zeit oder Volumen ermöglicht. Das Kapitel „Einleitung“ befasst sich eingehend mit den Grundkonzepten und Besonderheiten sowie den möglichen Szenarien für IronGate.

IronGate ist eine Client-Server-Lösung, deren Kern ein auf Linux arbeitender Serverprozess samt SQL-Datenbank bildet, welcher auf dem Gateway eines Subnetzes installiert werden muss. Dieser blockiert den Datenfluss zwischen dem eigenen und anderen Subnetzen (bspw. dem Internet) solange, bis ein Benutzer sich erfolgreich mithilfe des Clients authentifiziert. Der Benutzer meldet sich asymmetrisch verschlüsselt am Server an, woraufhin seine Identität und seine Historie überprüft und vorgegebene Regeln bzw. Einschränkungen (wie z.B. Bandbreite, Maximalvolumen, freie Ports etc.) gesetzt werden. Während ein Benutzer am Server eingeloggt ist, kann er je nach den für ihn vorliegenden Einstellungen völlig normal über das Subnetz hinaus arbeiten (kein Proxy etc.). Während eine Sitzung läuft, werden sämtliche Pakete aufgezeichnet und ausgewertet, wodurch ein Administrator stets auch laufende Sitzungen und deren Aktivitäten kontrollieren kann. Nach Ablauf bzw. Beendigung einer Sitzung werden die gesammelten Werte aufsummiert dem Accounting zur Verfügung gestellt. Die Funktionen, Algorithmen und Abläufe sowie deren Implementierung sind in den Kapiteln „Die Technik“ sowie „Die Implementierung“ nachzulesen.

IronGate ist Multi-Server-fähig (bis ca. 100 Server mit einer zentralen Datenbank, je nach Leistung), wobei ein Server mindestens ein Klasse-C Netz mit voller Auslastung verwalten kann. Das Kapitel „Testergebnisse“ schildert die Ermittlung dieser Werte. Neben der internen Benutzerdaten-

bank hat IronGate eine offene Backend-Schnittstellendefinition, welche es ermöglicht, beliebige weitere Benutzerdatenbanken anderer Systeme (bspw. LDAP, ADS, SQL etc.) in die Authentifizierung einzuschließen. Dadurch kann ein IronGate-System sich schnell und einfach in ein vorhandenes Netzwerk mit gewachsener Benutzerverwaltung integrieren, ohne ein Neuanlegen von Benutzern oder Gruppen vorauszusetzen oder Redundanz zu erzeugen.

Die Datenspeicherung erfolgt in einer relationalen Datenbank, über welche auch die Interprozesskommunikation erfolgt. Die Wartung der Datenbestände, Benutzer und Sitzungen erfolgt über ein Web-Interface, welches mit der zentralen Datenbank interagiert. Somit können auch mehrere Server im Verbund über ein einziges Interface gesteuert und kontrolliert werden. Die Clientsoftware besteht wahlweise auf einem Java-Applet oder einer Java-Applikation, welche lokal installiert wird. Somit ist IronGate Client-seitig mit allen Java-fähigen Plattformen kompatibel und auf diesen läuffähig.

Die Entwicklung am IronGate bzw. dessen Vorgänger begann Anfang 2001, wobei IronGate sich seither mit stetig neuen Versionen im Echtbetrieb mit etwa 100 Maschinen befand und dort getestet wurde. Das Kapitel „Praxiseinsatz“ widmet sich den Ergebnissen des Echtbetriebs.

1.2 Die Idee hinter IronGate

Vor den ersten Gedanken, welche erst später zu Machbarkeitsstudien und in weiterer Folge zu MOSIAP (bzw. IronGate) geführt haben, war eine wirklich dynamische und transparente Benutzerkontrolle in einem TCP/IP Netzwerk, bezogen auf den Layer-3 Verkehr, unbekannt. IronGate wurde entwickelt, um dieses Manko zu beheben.

Transparenz sowie Dynamik sind zwei äußerst wichtige Schlagworte im Bereich des Netzwerkmanagements, sprechen jedoch dasselbe Kernthema bzw. Ziel an: Den Benutzer soweit wie möglich managen, ihn jedoch nicht zu sehr einschränken. Das Optimum dieser beiden Kurven (Kontrolle über das Netzwerk/ den Verkehr und Grad der Einschränkung, Penetration, Umständlichkeit) ist also genau im Schnittpunkt derselben. Die meisten zum damaligen Zeitpunkt (im Detail war dies im Jahr 2000) bekannten Softwarelösungen waren Bordmittel der Mainstream-Betriebssysteme Windows NT / 2000, Novell Netware und Linux. Alle hatten gewisse Dinge für sich, und alle beherrschten zu einem marginalen Anteil auch eine Art Dynamik, jedoch nie in Verbindung mit Transparenz. Man konnte einem Benutzer zwar unabhängig von seiner IP-Adresse Einschränkungen geben, jedoch nicht transparent (Bsp. MS Proxy, SQUID). Andersherum konnte ich einem Benutzer transparente Einschränkungen geben (Bsp. iptables, ipchains), jedoch musste seine IP-Adresse zu diesem Zweck statisch vergeben sein (oder dieser seine MAC-Adresse angeben, sei es nun, um statische DHCP-Einträge und Paketfilter auf IP-Basis oder direkte Paketfilter auf MAC-Basis, was zum damaligen Zeitpunkt noch in den Kinderschuhen steckte, anzulegen). So

oder so war von der Kombination dieser beiden so wichtigen Punkte keine Spur. Novells Bordermanager hatte einige interessante Ansätze, krankte jedoch an der Konfiguration der Clients auf diese hin, soweit das beurteilen werden konnte. Weiter unten wird im Detail auf die alternativen anhand der Bordmittel anderer Betriebssysteme eingegangen, dieser Absatz sollte nur dazu dienen, die relativ komplizierte Wartung größerer Netzwerke mit Benutzerkomfort aufzuzeigen.

Nun spricht für diesen Mangel an Bordmitteln, dass in einem Trusted Environment keine solche Einschränkung notwendig sei. Dieser Aussage ist bis zu einer Größe des Netzes von ca. fünf Benutzern Recht zu geben - als Beispiel sei eine mittelgroße Familie genannt. Schon bei einer Wohngemeinschaft gilt dieser Einwurf nicht mehr. Eine gemeinsame Leitung wird nie fair geteilt, es wird immer jemanden geben, der damit Unfug treibt, einen anderen, der 24h am Tag die Leitung und damit die Bandbreite belastet, und einen dritten, der vehement bestreitet, im letzten Monat das Volumspensum überzogen zu haben. Bereits in diesem kleinen Rahmen macht eine Kontrolle des Datenflusses Sinn - wendet man die vollständige Induktion auf diese Behauptung an, wird man rasch feststellen, dass sie auch für alle Netze mit einer Benutzeranzahl größer fünf gilt. Sie gilt für Internate, Schulen, Provider, Campusse, öffentliche zugängliche Einrichtungen, Firmen - schlicht für sämtliche Einrichtungen, in welchen Benutzern bzw. Benutzergruppen, zu welchen man weniger als einen sehr persönlichen Kontakt hat, der Zugang zu anderen Netzen gewährt wird und man um eine Kontrolle dieses Zuganges bemüht ist. Dabei ist die oberste Direktive ganz klar das oben genannte Optimum zu erreichen - weder ist es die absolute Zufriedenheit des Benutzers noch die absolute Macht des Verantwortlichen.

Mitunter ein wichtiger Punkt des Netzwerkmanagements ist aber nicht nur die momentane Kontrolle des Verkehrs, sondern auch die nachhaltige - der Administrator muss also auch morgen noch in der Lage sein, nachzuvollziehen, was heute an Verkehr angefallen ist und welche Anomalien dieser Verkehr aufwies. Die Einsatzgebiete der so gesammelten Informationen sind genauso vielfältig, wie die Probleme, welche beim Data Collecting auftreten können. Auch hier gilt eine ähnliche Formel wie bei der momentanen Kontrolle des Netzes: man möchte soviel wie möglich kontrollieren, jedoch so wenig wie notwendig. Im Klartext sei damit gesagen, dass es fixe Grenzen der (vor allem nachhaltigen) Kontrolle eines Netzes aus vielerlei Gründen geben muss und immer geben wird. Ein ganz wesentlicher Grund ist der Datenschutz: Es darf nur soviel aufgezeichnet werden, wie zum Nachvollziehen rechtlich und kommunal bedenklicher anomalischer Vorgänge notwendig ist. Gleichzeitig kommt diese moralisch gesetzte Einschränkung dem Wunsch nach vertilitärer Beschränkung der Datenmenge zugute, da es schlichtweg unmöglich ist, sämtliche Vorgänge unendliche lange aufzuzeichnen. Es gilt also, eine Ausgewogenheit zwischen dem Umfang der Daten und der Vorhaltezeit dieser Daten zu finden, immer mit den Gedanken an die eigentliche

Zielsetzung (Accounting, Abuse Control) und der Moral. Es gilt also auch, einen latenten Teil der Kontrolle zu entwickeltn, welche den eben genannten Zielen dient.

Um den Netzwerkverkehr kontrollieren zu können, muss am Gateway des Netzes eine Kontrollsoftware arbeiten, welche die Aktivitäten protokolliert, nur authorisierte Pakete passieren lässt (die angemeldeter Benutzer) und ein Accounting durchführt. IronGate arbeitet daher wie in Abbildung 1.1 dargestellt.

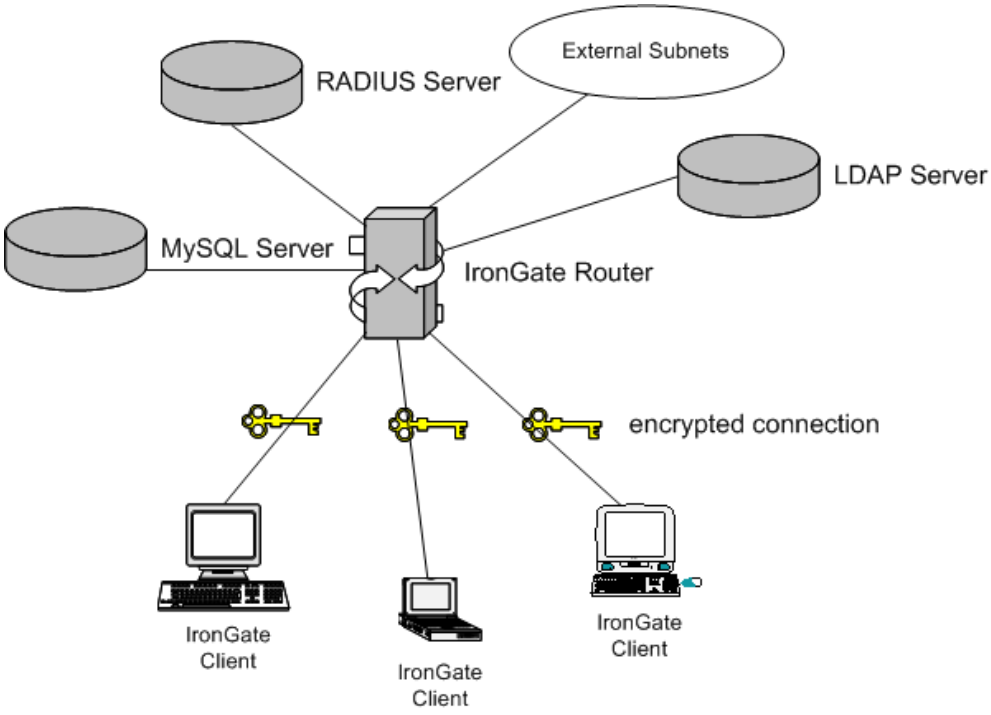

Abbildung 1.1: *Einfaches IronGate-Szenario*

1.3 Anforderungen

1.3.1 Benutzerkontrolle

Der primäre Sinn hinter der zu entwickelnden Software ist die Kontrolle der Benutzer eines (Teil-)netzes. Um dem Wunsch nach Uneingeschränktheit nachzukommen, bedarf diese einer gewissen Transparenz. Um der Verfügbarkeit nachzukommen, sollte diese Lösung transparent, wartbar und flexibel sein. Damit den Wünschen des Benutzers genüge getan wird, ist eine gewisse Skalierbarkeit und vor allem die einfache Handhabung der Lösung unumgänglich.

Sei nun soweit die Notwendigkeit der Kontrolle der User beschrieben, sei

nun kurz die Thematik „Schutz vor außen“ im Gegensatz zu „Schutz von innen“ aufgegriffen. Wird heute von Firewall-Lösungen gesprochen, so ist in den meisten Fällen eine Lösung zum Schutz vor Eindringlingen von außen gesucht. Was dabei nur spärlich Beachtung findet, ist der immer wieder gepredigte Schutz vor der „internen Kompromittierung“. Oftmals aus Unwissenheit und auch aus reiner Neugier oder Experimentierfreudigkeit heraus tendieren Benutzer dazu, sich selbst in eine, wenn auch gerne ignoriert, gesetzliche Grauzone hineinzubewegen. Jede Benutzung von Servicse über den ursprünglichen Sinn hinaus ist in den meisten Fällen zumindest anstößig. In der „Community“ mögen viele Dinge lockerer gehandhabt werden, mit zunehmendem Wachstum der Netze kommt jedoch auch der Gesetzgeber zum Zuge. Und so sind Vergehen, welche vor fünf Jahren noch als „Kavaliersdelikte“ geduldet wurden, inzwischen strafbar. Die meisten heutigen Netze haben eine Größe erreicht, welche es unmöglich macht, jeden Benutzer nach außen hin klar und eindeutig zu identifizieren. Das bedeutet, dass oftmals eine IP-Adresse ganze Subnetze und sogar Tausende Benutzer abdeckt. Kommt es nun zu dem gar nicht so seltenen Fall, dass eine Attacke oder sonstige Anomalie von dieser IP ausgeht, so obliegt es dem Administrator, bei einer eingehenden Anzeige oder Beschwerde den jeweiligen Benutzer zu belangen. Dies fällt beim übermäßigen Großteil (ca. 95 Prozent) heutiger Firewall-Lösung schwer bzw. gestaltet sich sogar als unmöglich. Teilweise fällt der Verwalter selbst dabei sogar in die gesetzliche Grauzone, da ihm das Aufzeichnen solch relevanter Daten zur Nachverfolgung je nach Gesetzgebung durchaus schwer fallen kann. Auch ist das Volumen der anfallenden Daten keinesfalls zu unterschätzen. Um diese Nachverfolgung im Ernstfall so einfach wie möglich zu machen, muss die Software auf so einen Fall ausgelegt werden. Das bedeutet, dass komplex zu analysierende Logdateien genauso fehl am Platze sind wie das Aufzeichnen sämtlicher Netzaktivitäten.

Auch fällt natürlich das Auswerten der Übertragungsvolumen und Online-Zeiten in die Kategorie der Benutzerkontrolle, welche jedoch einige Kapitel weiter unten näher beschrieben werden.

Um es gar nicht erst zu einer Eskalation der Situation kommen zu lassen, ist es mitunter notwendig, die Benutzer sowenig wie möglich und soviel wie nötig (siehe oben) bereits im Vorfeld einzuschränken. Dies geschieht durch das gezielte Setzen von Restriktionen bezogen auf jede in einem Netzwerk relevante Variable: Verbindungsart, Volumen, Bandbreite, Zeit und damit in Summe Geld. Auch damit muss sich die zu entwickelnde Software also auseinandersetzen.

1.3.2 Transparenz

Man steht vor dem Problem, eine vollständige Netzwerkbenutzerermanagementlösung zu entwickeln, jedoch auf solche Vereinfachungen wie die Verwendung eines Proxies oder SOCKS-Servers nicht zurückgegriffen werden

darf, da dies eine enorme Einschränkung der Möglichkeiten eines Benutzers darstellen würde. Dies bedeutet, dass viele Bordmittel von Betriebssystemen oder auch Lösungen von Drittanbietern nicht in Frage kommen.

Diese Einschränkung erfolgt durch die Anforderungen des Benutzers: Würde sich die zu entwickelnde Software Nutzen aus dem vorhandenen Pool von Relayern wie Proxies und anderen Circuit-Relay-Firewalls machen, so würde sie gleichzeitig vom Benutzer verlangen, jede von ihm verwendete Software (ob diese nun dazu in der Lage ist oder nicht) für die jeweilige Lösung zu konfigurieren. Das bedeutet, dass sich der Benutzer wiederum in jede Software weiter einarbeiten muss als es nicht schon notwendig ist und zusätzlich auch noch jede Software den von uns verwendeten Typus von Relayer auch unterstützen muss. Für die Verwaltung mag dies ein Optimum sein, denn schwach entwickelte Software würde außen vor bleiben und das Logging sowie die Authentifizierung wären um ein Vielfaches einfacher, dies kann dem Benutzer jedoch nicht zugemutet werden. Denn dieser bezahlt den vollen Einsatz JEDER beliebigen, selbstverständlich schadensfunktionfreien, Software, ohne dabei Rücksicht auf die Verwaltbarkeit legen. Und diese Software soll so verwendet werden können, wie sie vor der Installation einer neuen Sicherheitslösung verwendet werden konnte, oder wie es auch andere Netze ermöglichen. Daher ist man dazu gezwungen, mit all den Funktionen eine neuen Sicherheitslösung ein paar Layer weiter unten anzusetzen - so, dass es der Benutzer im besten Falle gar nicht bemerkt. Es wird also dort angesetzt, wo jedes Paket durchlaufen muss - im TCP-IP Stack des Gateways (bzw. dessen Routing-Prozessor). Sämtliche Pakete müssen ihn durchwandern, haben sie denn als Ziel ein fremdes Netzwerk.

Wie sich diese Funktionen im Detail zu verhalten haben, sei weiter unten beschrieben.

1.3.3 Wartbarkeit

Es muss eine Applikations-unabhängige, transparente und unauffällige Lösung geschaffen werden. Damit diese zwar für den Benutzer so einfach wie möglich bleibt, für den Administrator aber nicht nach hinten losgeht, gilt es, diese Lösung auch leicht wartbar zu halten. Somit ist es nicht Ziel dieser Arbeit, den Netzwerkmanager im Quellcode von Datensätzen nach Logeinträgen suchen zu lassen oder gar Benutzer über Kommandozeilenparameter dem Netz hinzuzufügen. Erreicht werden kann dies nicht durch den Einsatz proprietärer Software oder Protokolle, sondern durch Zurückgreifen auf vorhandene Standards und Softwarepakete.

1.3.4 Flexibilität

Wie weiter unten beschrieben und oben bereits angesprochen, gibt es zahlreiche mögliche Szenarien für die zu entwickelnde Software IronGate. Genannt

werden können nur einige mögliche Einsatzgebiete und, um ein Vielfaches an Arbeit vor allem in der Wartung der Versionen zu vermeiden, die Software soll ganz klar in einem Paket sämtliche dieser Einsatzgebiete abdecken.

Die Software muss somit also bezüglich des Umfeldes, der gewünschten Funktionen und des anfallenden Verkehrs sehr flexibel sein. Ein ISP setzt z.B. die Bandbreitenbegrenzungen zumeist nicht über zentrale Knoten, sondern auf jeder Bridge (HF-Modem) direkt. Diese loggen jedoch weder für ihn noch setzen sie auf einfache Art Zeitrestriktionen und erlauben Volumsabrechnungen. Diese Funktionen soll unserer Software überlassen sein, nicht jedoch das Bandbreitenmanagement. Eine WG wiederum, eine sehr persönliche Umgebung, hat sehr wohl Interesse an einer fairen Verteilung der Bandbreite, nicht jedoch am Loggen der Zugriffe. Schulen hingegen haben Interesse daran, sämtliche Funktionen auszuschöpfen, benötigen jedoch kein Accounting.

Es kristallisiert sich heraus, dass die Anforderungen denkbar unterschiedlich sind und dennoch von nur einem Softwarepaket übernommen werden sollen. Dazu kommt, dass in beinahe jedem Szenario unterschiedliche Stammdaten eine Rolle spielen - ein ISP hat ganz klar eine andere Benutzerverwaltung als z.B. ein Internat.

Dabei ist es natürlich auch interessant, ob die Software die brachliegenden Funktionen beim Abarbeiten der Connection ausklammert oder leer durchlaufen lässt (vergeudete Ressourcen) und wieviel Aufwand die Konfiguration der einzelnen möglichen Funktionen bedeutet. Auch ist zu entscheiden, was als SStandardkonfigurationängesehen wird, zumal im Sinne der einfachen Handhabung ein öne click setupnicht gänzlich uninteressant wäre, wenn auch angesichts der Komplexität der Software relativ unrealistisch.

Zusammengefasst muss sich IronGate also einer Vielzahl an verschiedenen Aufgabengebieten stellen - und hat dabei in jeder Hinsicht funktionell, stabil, sicher, schnell, unproblematisch und überschaubar zu bleiben.

1.3.5 einfache Handhabung

Es steht außer Frage, dass mit genügend Hartnäckigkeit praktisch jede denkbare Netzwerkfunktion implementiert werden kann - ob es dann auch überschaubar bleibt, ist jedoch nicht gesagt. Irongate hat als Ziel ganz klar das Schaffen einer einfach zu handhabenden Netzwerk- und Benutzerkontrolle, einfacher als bisherige Produkte, aber mindestens genauso mächtig. Dies wird in erster Linie durch zwei Prinzipien erreicht: Das Verhindern von Redundanz in der Entwicklung und das Steuern über Frontends.

Unter diesen beiden Prinzipien ist zu verstehen, dass das Schaffen von Redundanz ganz klar das Neuimplementieren bereits vorhandener Steuerprogramme für die Kernelfunktionen, mit denen Irongate in erster Linie arbeitet, wäre. Das schließt das Ansteuern der Firewall genauso ein wie das Konfigurieren der Bandbreitenregeln. Dafür existieren bereits System-

Programme, auf welche zugegriffen wird. Da diese in den meisten Fällen sehr ressourcenschonend implementiert sind, geht die Kosten-/Nutzenrechnung bezogen auf Wartbarkeit/Implementierung und Geschwindigkeitseinbußen dennoch auf. Die wirklich rechenintensiven Arbeiten wie das Anwenden der konfigurierten Regeln auf die passierenden Pakete und das manipulieren der Pakete bei Bedarf findet nach wie vor direkt im Kernel statt.

Das Steuern über Frontends ist eine der ganz klaren Besonderheiten von IronGate. Die meiste Software, welche entwickelt wird, reagiert auf „direkte“ Kommandos. Die Idee, im Intervall abgefragte Werte als Kontrollwerte zu verwenden, ist verhältnismäßig neu. Wird auch im weiteren Verlauf der Dokumentation dieses Feature desöfteren angesprochen und auch noch näher beschrieben, so soll dies doch an dieser Stelle etwas genauer erläutert werden. Die meiste Software wird insofern aktiv angesteuert, indem ihr Befehle gepiped werden und diese wiederum Functions und Prozesse anstoßen. Man „schiebt“ ihr also praktisch Kommandos zu. Irongate hingegen wird nur beim Starten des Mutterprozesses, dem Child-Control Prozess, mit Werten gefüttert (insofern auch passiv, da IronGate die Werte einer Konfigurationsdatei entnimmt). Möchte man dann aber einen Kind-Prozess manipulieren (also z.B. einen eingeloggten Benutzer), so gibt man kein z.B. „logout“-Kommando an die Mutter und diese ans Kind weiter (kein POSIX o.ä.), sondern man manipuliert Werte, auf welche das Kind regelmäßig zugreift. Im einfachsten Fall greift ein IronGate-Child während einer Session eines Clients alle paar Sekunden auf die zentrale Benutzerdatenbank zu, und überprüft, ob der Benutzer noch eingeloggt ist. Das ist eine an und für sich unübliche Idee, da es normalerweise in die Gegenrichtung zu verlaufen hat. Dem Kind wird aktiv mitgeteilt, dass es den Benutzer auszuloggen hat, und aktualisiert daraufhin die Datenbank („not logged in“-Status für den entsprechenden Benutzer). IronGate aber überprüft änderbare Werte einer Datenbank, und richtet sich nach diesen, ist aber auch in der Lage, sie selbst zu manipulieren. Auf diese recht eigenwillige Art und Weise des Prozessmanagements findet eine gepipete bidirektionale Kommunikation statt, welche über eine zentrale Datenbank ausgetragen wird. Damit hat man in Folge die Möglichkeit, die Manipulation der Prozesse von überall aus zentral und absolut I/O-unabhängig auszutragen. Denn für die meisten Datenbanken (sei es nun LDAP, NIS, mySQL, etc.) existieren zahlreiche Frontends und auch Schnittstellen. So kann man die IronGate-Prozesse direkt manipulieren, indem man Scripts eines Webservers und damit in Folge Browser verwendet.

Diese Methode des Prozessmanagements ermöglicht es also, das Management des kompletten Systems im besten Falle über einen plattformunabhängigen Browser abzuwickeln, welcher sich eine aktive Web-Plattform zunutze macht, die wiederum auf jedem denkbaren System implementiert sein kann. Diese Vorteile stehen natürlich auch dem Benutzer zur Verfügung, der dadurch deine kompletten privaten Netzwerkdaten per Browser verwalten kann (Kennwörter etc.).

Zusätzlich zu diesem Meilenstein in der Kontrolle eines Verwaltungssystems haben die Clients gewissen Vorgaben zu entsprechen. Ein System setzt sich genau dann durch, wenn es vereinfacht, und nicht verkomplext entwickelt wurde. Da IronGate eine dynamisch gesteuerte und arbeitende Software ist, muss ein Mindestmaß an Ein- und Auslogprozessen stattfinden. Dies wird simpel und einfach durch einen Client erzielt, welcher aber im Gegensatz zu vielen Layer-5 Protokollen nicht als Treiber oder eben Protokoll in das System eingebunden wird, sondern als simple Anwendung im Stile eines PIMs. Bei einer „privaten“ Maschine (kein permanenter Benutzerwechsel) kann der Client praktisch unsichtbar im Hintergrund die Kommunikation mit dem IronGate-Router aufrecht erhalten und die Verbindung authentifizieren, bei wechselnden Benutzern ist nicht mehr als der Benutzername und das Kennwort erforderlich, um „seine“ Internetverbindung zur Verfügung zu haben. Die Installation desselbigen schafft es dann tatsächlich, eine „one-click-installation“ vorzuführen, angeleitet durch eine einfach gehaltene Website.

Somit ist es tatsächlich möglich, nicht nur eine komplexe und hochfunktionelle, sondern auch eine praktische und einfach überschaubare Netzwerkmanagementlösung zu schaffen.

1.4 Einsatz-Szenarien

1.4.1 Gemeinde-WLAN

Durch die Unabhängigkeit vom Layer 1, dem physischen OSI-Layer, ist IronGate nicht davon abhängig, welches Trägermedium für den Einsatz verwendet wird. So ist auch ein Szenario auf WLAN-Basis problemlos denkbar.

Ein solches wäre zum Beispiel der Einsatz von IronGate im Bereich einer Gemeinde oder auch Stadt (MAN). Ein IronGate-Router pro Stadtteil (z.B. pro 20 Access Points, je nach geplanter Auslastung) würde genügen, gekoppelt mit einer zentralen Benutzerverwaltung, um ein komplettes Gemeine-/Stadt-WLAN unter Kontrolle zu haben. Detailliertes Accouting, Traffic Management und eine erweiterte Protokollierung ermöglichen eine präzise Abrechnung sowie Determinierbarkeit.

Das offene Backend von IronGate sowie dessen offengelegte Protokolle ermöglichen auch den direkten Import z.B. einer Einwohnerdatenbank, um sämtlichen Einwohnern mit geringem Migrationsaufwand den Zugang zum WLAN zu ermöglichen. Gäste und Touristen können völlig problemlos über Einweg-Konten versorgt werden, deren Übertragungsmenge und Onlinezeit von vorne herein entsprechend beschränkt ist (je nach Bezahlung).

Völlig einfach wäre es sogar möglich, Einwegkarten mit Freischaltcodes um fixe Preise in jeder Trafik zu verkaufen, um ähnlich wie bei z.B. Handies das Aufladen des Kontos zu ermöglichen (per Browser z.B.). Der Implementationsaufwand einer solchen Lösung ist sehr gering und ermöglicht dadurch

ein absolut inidividuell gestaltbares Abrechnungssystem.

1.4.2 Universitäts-Campus

Auch in einer instutionellen Umgebung wie der eines Unicampus kann IronGate äußerst vielseitig eingesetzt werden. Da Studenten in den meisten Fällen ein Großteil der Services kostenlos zuteil wird, herrscht ein reger Missbrauch der Dienste und damit der Bandbreiten. Direkt gekoppelt mit dem zentralen Verwaltungssystem der Uni (z.B. Whitepages von LDAP) kann IronGate jeden Studenten bzw. jeden auf der Uni verzeichneten User vollwertig kontrollieren, indem es seine Daten aus dem zentralen System übernimmt und anwendet. Schnell und einfach kann ein Schlüssel für die max. Bandbreite einer ganzen Gruppe von Benutzern (z.B. allen Studenten) zugewiesen werden, was dafür sorgt, dass sich der Missbrauch bald von selbst erledigt.

Die Anzahl der notwendigen IronGate Router ist gänzlich abhängig von der Netztopologie, der Anzahl der Subnetze und der Institute. Eine Einrichtung wie die Technische Universität Wien würde vermutlich mit ca. 20-30 IronGate Routern problemlos auskommen. Da IronGate Linux als Basis hat, würde es in den meisten Fällen gar keine eigene Hardware benötigen, sondern könnte einfach auf die bereits vorhandenen Linux-Router aufsetzen.

IronGate spielt in diesen Bereichen also ganz klar seine Fähigkeiten der übergeordneten Kontrolle des momentanen Traffics aus. Accounting ist hier eher zweitrangig, wichtig ist wohl aber die Proktollierung der Pakete, zumal Universtitäten als Wissenspool auch in Sachen Netzwerksicherheit bekannt sind - und daher auch ein Großteil der Übergriffe von solchen Subnetzen ausgeht.

1.5 Konkurrenzprodukte

Als kommerzielle und damit näher ins Auge gefasste Produkte seien hier folgende kurz angesprochen:

- CISCO PIX
- Microsoft's ISA
- Novell Bordermanager

Allen diesen drei Lösungen (und auch den mir bisher bekannt gewordenen) ist es gemein, nicht gleichzeitig über ein transparentes und dennoch auch dynamisches Firewalling zu verfügen. Dynamische Regelzuweisungen sind lediglich dann möglich, wenn ein Benutzer einen (nicht mehr transparenten) Proxy mit Authentifikation verwendet, was ihn auf einige wenige Protokolle einschränkt. Da das Relationenproblem sich im Prinzip wie ein

roter Faden durch das Design der genannten Produkte zieht (ein Benutzer ist entweder eine MAC-Adresse oder eine IP, nie jedoch der Benutzer selbst, wenn es darum geht, Regeln zu setzen oder Limits festzulegen), verfügen die Produkte auch über kein wirklich relationales DNS-Updating (nur in Verbindung mit DHCP, was also wiederum auf der MAC-Adresse basiert) geschweige denn über ein relationales Logging (es wird zwar ausführlich mitgeloggt, jedoch nur in Bezug auf die IPs der einzelnen Benutzer, nicht auf die Benutzer selbst). Die an und für sich möglichen Vorzüge, welche die Integration einer relationen und dynamischen Firewall-/Traffickontrolle in ein Netzwerkbetriebssystem als Bordmittel ergeben würde, wurden von keinem großen Hersteller bisher ausgeschöpft. Ansonsten wäre es noch naheliegender und reizvoller (wobei dabei die absolute plattformunabhängigkeit verloren ginge), die bei IronGate durch einen autonomen Client initiierten Abläufe am Server in einem Durchgang mit der Anmeldung am Verzeichnis oder dem Domänencontroller zu vollziehen. Dies wurde jedoch niemals verwirklicht und scheint auch in der mittelfristigen Zukunft nicht auf den Plänen der Hersteller verzeichnet zu sein (bsp. hat das geplante Windows Longhorn keinerlei solche Funktionen implementiert). Was die Backend-Kapazität der einzelnen Lösungen anbelangt, wird grundsätzlich den hauseigenen Entwicklungen (Microsoft: ADS, Novell: NDS, CISCO: TACACS+) der Vorzug gegeben. Alle Lösungen bringen zwar mit dem einen oder anderen Umstand einige Backends bspw. für LDAP oder auch Radius mit, welche jedoch in keinem Zusammenhang mit der eigentlichen Firewall- bzw. Zugangskontrolllösung stehen. Da IronGate auf Linux aufsetzt (auf Serverseite), hat es damit auch die Möglichkeit, auf sämtliche jemals als PAM-Modul implementierten Backends zuzugreifen, sofern man ein PAM-Backend-Modul für IronGate nachträglich implementiert (momentan nur analysiert). Diesen Vorzug kann aufgrund des hohen Entwicklungsaufwandes derartiger Module (bei Linux durch die freie Entwicklergemeinde entstanden) in absehbarer Zeit kein Hersteller wettmachen, und unter der Voraussetzung dass IronGate mit einem PAM-Modul ausgestattet wird, werden auch fortlaufende Backend-Supports für die Zeit des PAM-Supports folgen.

Um die einzelnen Vorzüge bzw. Features der einzelnen Lösungen noch einmal deutlich zu machen, sei auf Abbildung 1.2 verwiesen.

1.6 Vorzüge

Neben den bereits erwähnten Unterschieden zu anderen momentan verfügbaren Lösungen hat IronGate eine hohe Anzahl an z.T. innovativen Lösungsansätzen, welche ganz klar als Vorzüge dieser Implementierung zu nennen sind. In den späteren Kapiteln wird noch im Detail auf diese eingangen, hier seien sie stichwortartig aufgeführt:

- *Transparente und dynamische Firewall (benutzerrelational)* - vereint

Feature	IronGate	Microsoft ISA	Novell Bordermanager	CISCO PIX
Dynamisches Firewalling	X	X	X	
Transparentes Firewalling	X	X	X	X
Relationales Logging	X			
Plattformunabhängigkeit	X			X
Relationales dynamisches DNS-Update	X			
Relationale Zeitbegrenzung	X	X	X	
Relationale Volumensbegrenzung	X		X	
Dynamisches & relationales Firewalling	X			
Heuristische IDS	X			X
ADS-Backend	X	X		
LDAP-Backend	X	X	X	
SQL-Backend	X			
Handover-Fähigkeiten	X			

Abbildung 1.2: *Gegenüberstellung einzelner Lösungen*

die Stärken von Paketfiltern, die rein statisch sind aber transparent, mit denen der Proxies, welche durch eine mögliche Benutzerauthentifikation auch dynamisch sein können, jedoch nicht transparent. So kann ein Benutzer erst das Subnet verlassen, wenn er sich am IronGate-Server angemeldet hat, verfügt dann über keinerlei Einschränkungen bzgl. der möglichen Protokolle

- *Benutzerrelationales QoS in Form von Traffic Priorizing und Traffic Shaping* - egal von welcher IP/Mac ein Benutzer sich einloggt, es bekommt stets die für ihn vorgegebenen Regeln bzgl. der Priorität seines Traffics am Router und der zur Verfügung gestellten maximalen Bandbreite auferlegt

- *Benutzerrelationales Dynamic DNS Updating* - nicht mehr der DHCP-Server, sondern IronGate übernimmt das Setzen dynamischer DNS-Einträge - dadurch ist nicht mehr der Name der Maschine für das DNS ausschlaggebend, sondern der Name des eingeloggten Benutzers, egal, an welcher Maschine er sich einloggt

- *Wireless/Wired Handover-Funktionen* - wechselt ein Benutzer beim Roaming das Subnet (ändert sich seine IP-Adresse) oder verliert der Benutzer aus anderen Gründen die Verbindung zum Server, so wird er bei Wiederherstellung der Netzwerkverbindung sofort und ohne Konfigurationsänderung am Standard-Gateway des momentanen Subnetzes angemeldet

- *Benutzerrelationales Packet-Logging* - Ein detaillierter Log (prekonfiguriert wird jedes SYN-Paket aufgezeichnet) zeichnet jedes Paket, welches den Router passiert, auf, und stellt in Echtzeit eine fixe Verbindung zwischen dem entsprechenden Paket und dem Benutzer, von dem es stammt, her - so ist es gleichgültig, auf welcher Maschine ein Benutzer arbeitet, sofern er seinen Benutzer verwendet, wird sein Traffic immer nachvollziehbar sein (Abuse Control)

- *Heuristische Trafficanalyse* - durch die Verwendung eines erfahrungsbasierenden, heuristischen Algorithmus wird durch die Signaturen, die der erzeugte Traffic eines jeden Benutzer verursacht, analysiert, und einer gewissen Charakteristik zugeordnet. So kann man auch ohne die Analyse des Packetlogs jederzeit feststellen, was ein Benutzer momentan für Aktivitäten hat

- *Detailliertes Accounting* - der erzeugte Traffic und die Zeit, die ein Benutzer online war, kann über die ganze Lebenszeit einer IronGate-Installation hinweg nachvollzogen und abgerechnet werden. Dabei wird auch die verwendete Benutzerklasse beachtet.

- *Verschiedene Benutzerklassen* - abgesehen von den Benutzer- und Gruppenspezifischen Eigenschaften kann sich ein Benutzer auch dynamisch (sofern der Administrator die Möglichkeit lässt) zwischen 3 vordefinierten Benutzerklassen wählen. Diese können sich in den voreingestellten QoS-, Paketfilter- und Zeitmanagement-Einstellungen unterscheiden und schlagen sich beim Accounting im Endeffekt nieder (können also verschiedenen vergütet werden)

- *Offene Backend-Anbindung* - IronGate ist lediglich auf das interne Backend in Form einer mySQL-Datenbank angewiesen, welche lokal oder auch auf anderen Servern liegen kann. Darüber hinaus hat es eine offene Schnittstelle, welche momentan ADS, LDAP und mySQL als alternatives Backend unterstützt. Durch die Implementierung eines PAM-Backends auf Basis der offenen Schnittstelle kann jeder momentan verfügbare Verzeichnisdienst und jede Benutzerdatenbank angesprochen werden. Damit ließe sich IronGate schlichtweg überall integrieren.

- *Plattforumunabhängige Verwendung und Verwaltung* - außer dem Serverprozess, welcher auf Linux als Basis angewiesen ist, kann die Verwendung von IronGate durch die Benutzer in Form des Clients auf absolut jeder Plattform erfolgen (sofern diese Java unterstützt) und die Verwaltung auf jedem Betriebssystem bzw. in jeder Umgebung, in der ein HTML4-fähiger Browser installiert ist. Dies macht die Integration Irongate's in quasi jede Umgebung möglich.

- *Detaillierte Zeit- und Volumenslimitierungen* - neben dynamischen Paketfiltern und dynamischen QoS-Fähigkeiten bietet IronGate auch die Möglichkeit, jedem Benutzer einzeln oder ganzen Gruppen fixe Zeit- und Volumenslimitationen (auch begrenzt auf Tageszeit und Datum) zu erteilen, welche sich auf Tage, Wochen oder Monate beschränken können. Dabei wird auch Rücksicht auf den gesamt-Traffic, den Up- und den Download genommen.

- *Live-Informationen über die Auslastung einzelner Benutzer* - neben der Usage-Heuristik hält IronGate auch permanent einen Table aktuell, welcher Auskunft über den Online-Status der eingeloggten Benutzer gibt. Man damit jederzeit einen Überblick über die Auslastung der Leitungen aufgegliedert auf die einzelnen Benutzer. Parallel dazu erhält der Benutzer permanent über den Client eine Information darüber, wieviel Auslastung er gerade erzeugt und, falls er limitiert ist, wieviel Pensum in Form von Zeit oder Volumen er noch zur Verfügung hat

- *Verwendung von Strong Asymetric Encryption* - der Java-Client bietet die Option, über die Verwendung von asymetrischer Verschlüsselung (RSA) die Einlogprozedur vorzunehmen, wobei ein Salt dazu beiträgt, dass weder Repeat-Attacken noch Abhörangriffe zum Erfolg führen

- *Offene Schnittstellen für Clients und Web-Frontend* - die Protokolle der Kommunikation zwischen dem Server und den Clients sowie dem Web-Frontend liegen offen, was das Nachimplementieren anderer Clients bswp. in anderen Sprachen ebenso ermöglicht wie das Auswerten bzw. Anzeigen der gesammelten Informationen bspw. grafisch

- *Anti-Spoof Mechanismus* - spezielle Paketfilter-Regeln, welche nur korrekte Pakete in Verbund mit der beim Login verwendeten MAC-Adresse passieren lassen, unterbinden quasi jeden Versuch, die IP-Adresse oder sogar die MAC-Adresse zu spoofen

Nach einer späteren Reimplementierung des Projektes in Java wird jedes dieser und auch der unten näher beschriebenen Features als eigene Klasse existieren, was später die Möglichkeit bietet, diese auch für andere Projekte direkt und nahtlos weiterzuverwenden, sowie auch weitere Funktionen hinzuzufügen (einige Möglichkeiten siehe unten).

2 Die Technik

Dieses Kapitel durchleuchtete die Abläufe und die einzelnen Module in IronGate im Detail (abstrakt und analytisch).

2.1 Die Struktur / der Ablauf

IronGate läuft einerseits als Prozess(e) auf einem Server bzw. dem Router des lokalen Netzwerks (über welchen sämtlicher Datenverkehr zu anderen Subnetzen oder dem Internet läuft). IronGate greift zur Authentifikation auf Backends zu, welche entweder lokal installiert oder auf anderen Maschinen laufen können. Der Server sperrt solange sämtliches Routing zu anderen Netzen für alle Adressen des lokalen Subnets, bis sich ein Benutzer erfolgreich authentifiziert - erst dann wird das Routing aktiviert bzw. die Paketfilter für diesen einen Benutzer angepasst. Der Benutzer benötigt hierzu einen Client auf seiner Maschine, welcher die Verbindung mit dem Server aufnimmt und aufrecht erhält.

In den folgenden Kapiteln wird dieses Schema noch sehr viel näher behandelt, zur allgemeinen Verständlichkeit und den Positionen der einzelnen Elemente von IronGate sei auf die Struktur in Abbildung 2.1 hingedeutet.

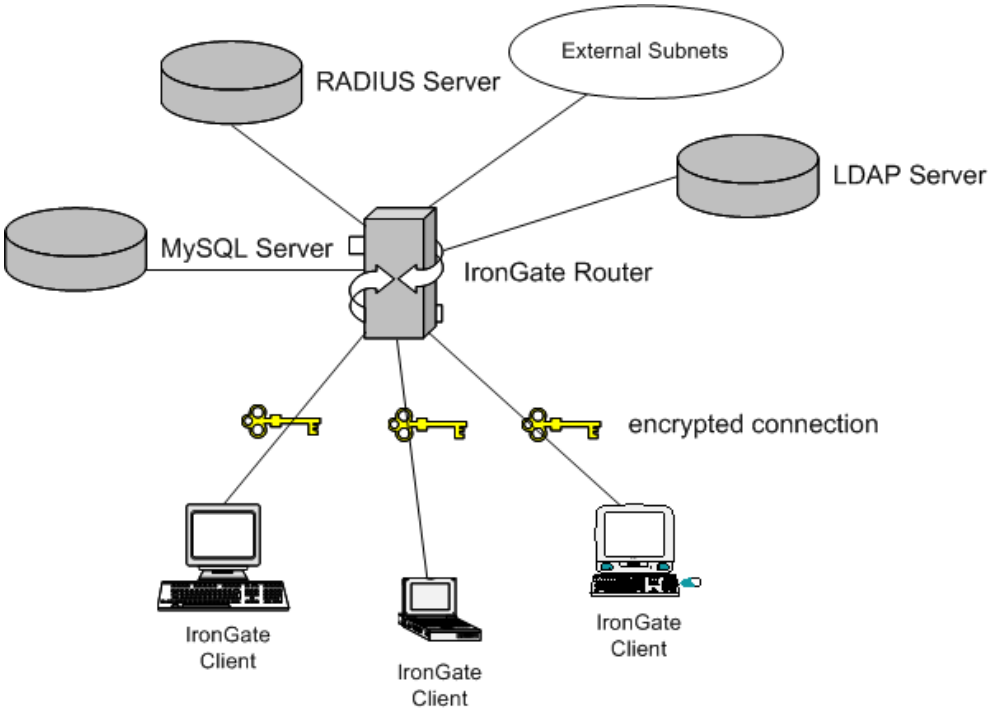

Abbildung 2.1: *Die Struktur von IronGate*

2.1.1 Der schematisch Ablauf Client/Server-Verbindung

Der in Abbildung 2.2 dargestellte Ablauf sieht kurz ausgeführt folgendermaßen aus:

1. Der Server wartet auf eine Verbindungseröffnung durch den Client
2. Falls keine Überlast herrscht, wird ein Child Process geforked, der den Client „uebernimmt“
3. Es erfolgt eine Handshaking-Sequenz, in welcher sich der Client und der Server über Protokollverschlüsselung- und Version einig werden
4. Sofern der Handshake erfolgreich ist, wird Benutzername und Kennwort sowie die Userclass übergeben (falls nicht, erfolgt ein Logout mit Aufzeichnung)
5. Es werden anhand dieser Werte daraufhin die Backends und die internen Benutzertabellen abgefragt und die gewonnen Werte „gemerged“ (Reihenfolge s.u.)
6. Es folgt die Zeit- und Volumensüberprüfung, die, sofern sie klappt, von dem Setzen der einzelnen abgefragten Regeln begleitet wird (falls nicht, erfolgt ein Logout mit Aufzeichnung)
7. Der Benutzer wird online gesetzt und die keep-alive-Sequenz eingeleitet
8. Parallel dazu erfolgt ein(e) permanente(s) Update/Abfrage des Online-Tables, um die Informationen über den Benutzer aktuell zu halten, und eine stetige Kalkulation seiner Restzeit und seines Restvolumens
9. Ebenfalls parallel dazu wird die Momentaufnahme des Clients an diesen weitergeleitet und steht dem Benutzer als Information zur Verfügung
10. Entweder durch Eigeninitiierung, durch Überschreitung der Zeit- und/oder Volumenlimits oder durch einen Verbindungsabbruch wird die Keep-Alive Sequenz und damit die Sitzung abgebrochen bzw. beendet
11. Der Benutzer wird offline gesetzt
12. Die individuell für den Benutzer gesetzten Regeln (Firewall, QoS, DynDNS etc.) werden wieder entfernt
13. Die gesammelten Informationen über den Benutzer während seiner Onlinezeit (verbrauchtes Volumen, Traffic, Zeit) werden an den Authlog zur späteren Weiterverwendung (Accounting) übergeben
14. Der Kindprozess wird beendet

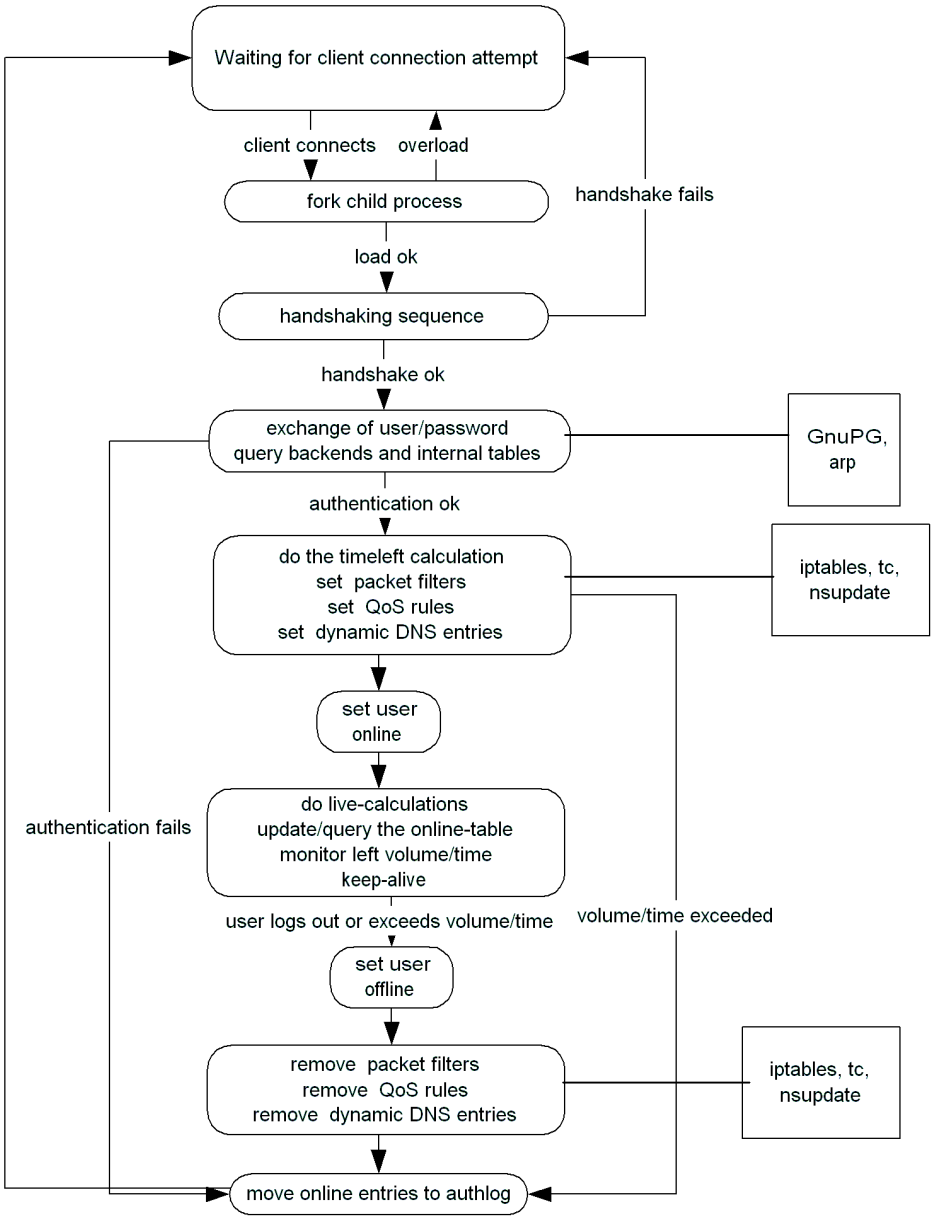

Abbildung 2.2: *Der schematische Ablauf*

2.1.2 Der schematische Ablauf am Server

Da der Client im großen und ganzen nur ein „dumb Terminal“ darstellt, ähnelt der Ablauf am Server der der Client-/Serververbindung. Die Hauptarbeit hat der Server zu verrichten, hier wird also nur detaillierter auf den Ablauf der Vorgänge am Server ohne einer Client-Verbindung (ergänzend s.o.) eingegangen.

1. Der Mutterprozess wird gestartet
2. Sämtliche Variablen werden eingelesen, Datenbankverbinbdungen aufgebaut, Basisregeln werden gesetzt
3. Das Forking wird initialisiert und ein Socket auf einem vordefinierten Port geöffnet (meist 1140)
4. Eine Session wird eingeleitet, wenn ein Client sich anmelden möchte (Socket wird geöffnet)
5. Der Prozess läuft im Child weiter
6. Username und Passwort werden abgefragt und verifiziert, im Falle korrekter Angaben werden sämtliche Benutzerinformationen abgefragt sowie die Gruppenrechte vererbt
7. Eine Erfolgs- oder Misserfolgsmeldung, je nachdem, wird dann den Client abgesetzt
8. Der Client sendet regelmäßige Keep-Alive Meldungen an den Server, woraufhin dieser mit dem Listening fortfährt
9. Wenn eine der o.h. Abbruchbedingungen eintritt, wird die Session abgebrochen
10. Alle Regeln werden zurückgesetzt und die Benutzerinformationen zurückgesetzt
11. Der Socket wird geschlossen und der Prozess wird beendet (terminiert)
12. Beim Herunterfahren des Mutterprozesses werden sämtliche Kindprozesse sauber durch ein table-seitiges Offline-Setzen aller Benutzer heruntergefahren

2.1.3 Der schematische Ablauf am Client

Der Client bietet ein ansehnliches Front-End, welches die Eingaben von Benutzername und Kennwort speichert bzw. übernimmt und dem Server mitteilt.

1. *Der Client wird entweder über den Autostart-Prozess oder manuell gestartet (nachdem die Installation erfolgte)*
2. *Der Client minimiert sich in den Tray je nach Konfiguration*
3. *Benutzername und Kennwort werden entweder aus einem File gelesen, vom Automatic-Configuration-File des Servers gelesen oder manuell eingegeben*
4. *Ein Socket wird am Server geöffnet (IP und Port vorkonfiguriert bei der Installation)*
5. *Die Werte werden übertragen und vom Server verifiziert*
6. *Je nach Ergebnis liefert der Server einige Informationen und ACCESS DENIED bzw. ACCESS GRANTED zurück*
7. *Je nach Antwort beginnt die Keep-Alive Sequenz, während welcher der Server permanent einen aktuellen „Snapshot“ vom Benutzer macht (Restzeit, Restvolumen, Traffic, etc.) und an den Client weiterleitet*
8. *Durch das Logout-Command des Benutzers wird diese Sequenz unterbrochen, wodurch der Server reagiert und den Benutzer ausloggt*
9. *Der Client kann jederzeit problemlos beendet werden*

2.1.4 Der genaue Ablauf der Client-/Server-Kommunikation

Im folgenden wird in Bezug auf Abbildung 2.3 auf den Handshake, die Authentifizierung und auf die Keep-Alive-Sequenz in Bezug auf die Client-/Serverkommunikation detailliert eingegangen:

1. Der Client initiiert durch Öffnung eines Sockets die Verbindung
2. Der Server fragt über „Client:“ und RETURN die Version des Clients ab
3. Der Client antwortet bspw. mit „java“ und RETURN
4. Der Server fragt über „Encryption:“ und RETURN die gewünschte Form der Verschlüsselung ab
5. Der Client antwortet bspw. mit „gpg“ und RETURN (ab dieser Stelle wird davon ausgegangen, dass GPG als Verschlüsselung verwendet wird - wenn keine Verschlüsselung verwendet wird, wird der Austausch der Schlüssel und des Salts übersprungen und bei „Username:“ fortgesetzt - Client müsste dafür noneünd RETURN übermitteln)

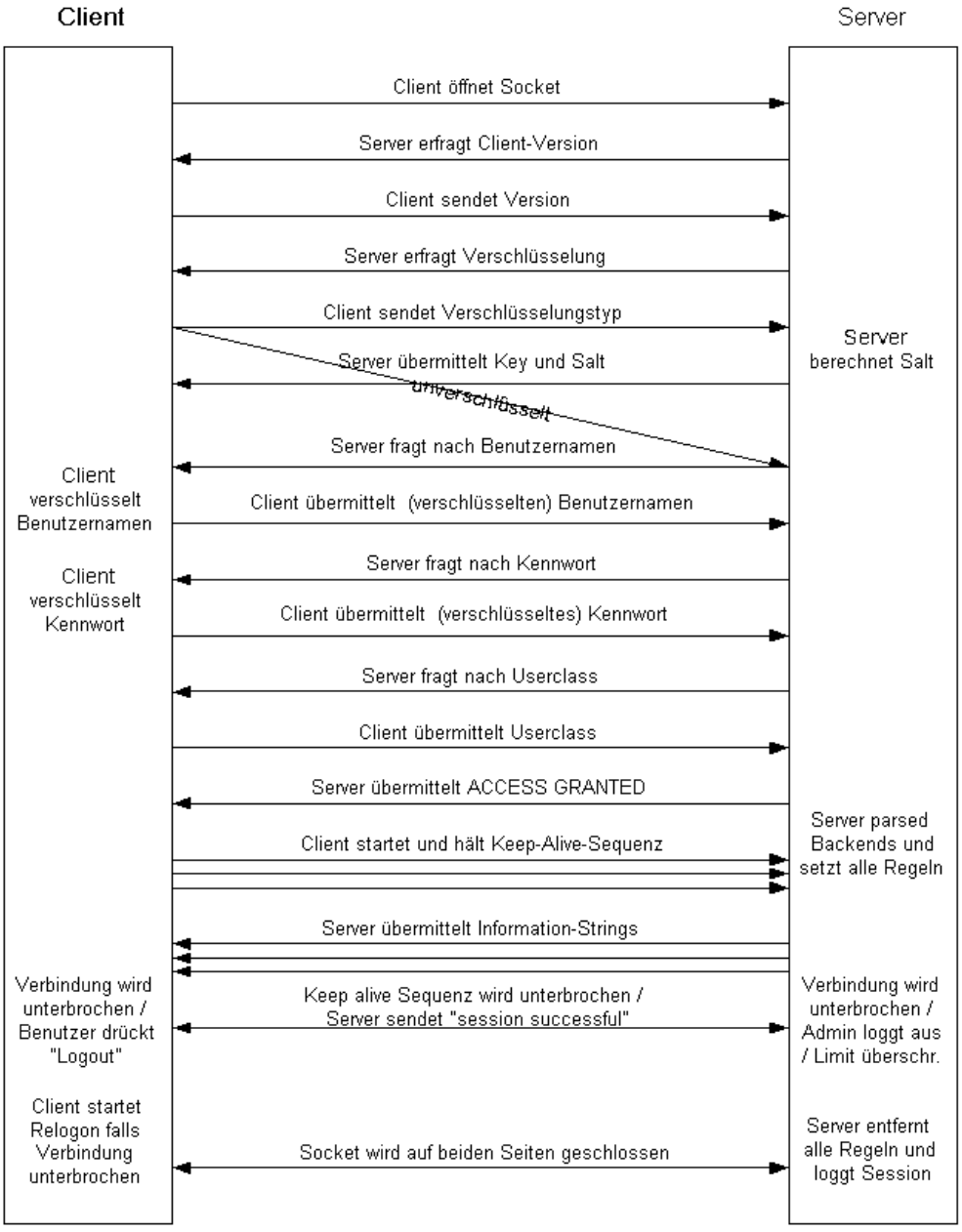

Abbildung 2.3: *Genauer Ablauf Client-/Serverkommunikation*

6. Der Server reagiert darauf mit dem Senden des Schlüssels und RETURN gefolgt vom Salt und RETURN - daraufhin fragt er mit „Username:“ und RETURN nach dem verschlüsselten Benutzernamen

7. Der Client verschlüsselt den Benutzernamen, nachdem der Salt angehängt wurde, und übermittelt ihn als verschlüsselten String gefolgt von RETURN

8. Der Server entschlüsselt den String über den Private-Key, entfernt den Salt und fragt über „Password:“ und RETURN nach dem verschlüsselten Kennwort

9. Der Client verschlüsselt das Kennwort, nachdem der Salt angehängt wurde, und übermittelt ihn als verschlüsselten String gefolgt von RETURN

10. Der Server entschlüsselt den String über den Private-Key, entfernt den Salt und fragt über „Class:“ und RETURN nach der unverschlüsselten Klasse

11. Der Client sendet die selektierte Klasse (default,bronce,silver,gold) gefolgt von RETURN an den Server zurück

12. Der Server verifiziert die übergebenen Daten und übergibt entsprechend dem Ergebnis (wir gehen an dieser Stelle von einer akzeptierten Verbindung aus) ein „ACCESS DENIED“ oder „ACCESS GRANTED“ (zweiteres) gefolgt von einem RETURN

13. Der Client beginnt daraufhin die Keep-Alive Sequenz, indem er im (voreingestellten) Abstand von 5 Sekunden einen „.“ überträgt

14. Der Server arbeitet die oben genannten Prozeduren ab und wartet 10 Sekunden auf den ersten Dot des Clients (diese Sequenz setzt sich bis zum Logout-Vorgang fort) - nach jedem Dot übergibt der Server die momenante Benutzerstatistik als String-Sequenz an den Client, welcher diese über einen Puffer an den Benutzer weitergibt. Durch die doppelte Wartezeit am Server ist dem Client ein „Timeout“ von fünf Sekunden gewährt, evtl. durch eine Systemauslastung

15. Durch einen Logout am Client bricht dieser die Keep-Alive Sequenz ab (alternativ bricht der Server die Verbindung nach einer Limitüberschreitung oder der Anweisung vom Table ab), nach max. 10 Sekunden bricht der Server damit das Warten auf den „.“ ab und leitet die Logout-Sequenz ein - ein ungewollter Abbruch der Verbindung (Roaming, Verbindungsunterbrechung jeglicher Art) wartet der Server auch nur die 10 Sekunden ab, und wertet den Client als ausgeloggt

16. Der Server übermittelt als letzten String ein „session successful“ gefolgt von RETURN an den Client, im Falle eines gewollten Verbindungsabbruches von Serverseite her

17. Der Socket wird geschlossen, der Client gilt als ausgeloggt

18. Im Falle eines unvorhergesehenen Verbindungsabbruchs (ohne Wissen des Clients, sei es durch einen gewollten Logout von Seiten des Clients oder des Servers) wird entweder Roaming oder ein Verbindungsabbruch angenommen und dreimal ein erneuter Login-Versuch mit vorhergehendem Auto-Configuration-Script-Query vom Server versucht. Bei Misserfolg wird die Sequenz beendet und die Verbindung gilt als unterbrochen, bei Erfolg beginnt der Handshake von neuem (s.o.)

Da dieses Protokoll absichtlich einfach gehalten wurde, in Bezug auf Belastbarkeit (Netzauslastung, Systemauslastung) und die Übermittlung der Werte auch sehr tolerant ist, ist es in kürzester Zeit mögliche, eigene, noch maßgeschneiderte Clients zu entwickeln, wie es bereits während der Feldtestphase (welche inzwischen 3 Jahre andauert) bspw. durch einen Schüler binnen kürzester Zeit in C++ durchgeführt wurde.

2.2 Verwendete Infrastrukturen und Protokolle

IronGate hat den Vorteil, auf eine Vielzahl standardisierter Protokolle zurückzugreifen. Diese seien im folgenden kurz angesprochen.

2.2.1 GPG

GPG ist die GNU-Version des PGP (Pretty Good Privacy) Verschlüsselungsmechanismus. PGP und damit auch GnuPG verwenden RSA für den Austausch symmetrischer Schlüssel (DES) und verschlüsseln die eigentliche Nachricht dann symmetrisch. Das macht (nach einmaliger RSA-Schlüsselgenerierung) die Kommunikation sehr schnell, ressourcenschonend und effizient.

Da der Server in Perl programmiert wurde und der Client in Java, waren keine Bibliotheken verfügbar, um ohne externe Tools und Zertifikateserver (welcher in dieser Umgebung schlichtweg überflüssig wäre) eine asymmetrisch verschlüsselte Übermittlung zu ermöglichen. GPG war deshalb das Mittel der Wahl, weil es auf beinahe allen Plattformen verfügbar ist, als sicher gilt und über Kommandozeilen ansteuerbar ist.

GPG wird daher einmalig verwendet, um das Schlüsselpaar des IronGate-Servers zu berechnen. Da ein Client ohne Konfiguration auskommen soll und muss, wird der öffentliche Schlüssel immer beim Login-Prozess an den Client übermittelt - so kann sich ein Client mit gleicher Konfiguration an zwei unterschiedlichen IronGate-Servern einloggen. Da der Private-Key stets im

Besitz des Server bleibt, ist das Kennwort zur Entschlüsselung stets dasselbe („irongate“). Um eine Replay-Attacke auf diese Art und Weise zu vermeiden, wird vom Server zufallsgeneriert ein Salt an den Client übermittelt, mit welchem der die zu übertragenden Strings vor der eigentlichen Verschlüsselung mischt. So werden abgefangene Crypts wertlos, da jeder verschnüsselte Wert trotz gleicher Aussage und gleichen Schlüsseln stets ein Unikat bleibt.

Mehr Informationen zu GPG finden sich unter [1] und die für die Verwendung unter IronGate wichtige Syntax im Kapitel der Implementierung.

2.2.2 CBQ

CBQ (Class Based Queueing) ist ein Mechanismus von vielen (jedoch der beinahe komplexeste), um ein Traffic-Shaping durchzuführen. Es erlaubt das Verschachteln von verschiedenen Traffic-Klassen nebst Priorisierung.

Unter Linux ist der CBQ-Shaping-Algorithmus bereits beim Kernel-Quellcode dabei und wird über das Tool „tc“ aus dem „iproute2“ Paket gesteuert und konfiguriert. Im Falle von IronGate wird für jeden Benutzer eine eigene Klasse angelegt, welche seiner Priorität entspricht und auf die ihm maximal zustehende Bandbreite konfiguriert. Jeweils für das ein- und ausgehende Interface müssen eigene Klassen angelegt werden. Auf die Klassen verweisen dann auf die Paketmarkierung des einzelnen Benutzers zugeschnittene Filter, welche ebenfalls über beide Interaces gelegt werden müssen.

CBQ mag zwar relativ kompliziert klingen im Verhältnis zu externen Möglichkeiten wie “shaper“, die rein auf das Traffic-Shaping zugeschnitten sind, verfügt jedoch über einen sehr effizienten und v.a. flexiblen Algorithmus, der überdies das Priorisieren im gleichen Algorithmus wie das Shapen unterbringt (zumeist sind dazu zwei Algorithmen notwendig).

Die Funktionsweise des CBQ-Algorithmus ist in Abbildung 2.4 schematisch dargestellt.

Mehr Informationen zu CBQ finden sich unter [2] und die für die Verwendung unter IronGate wichtige Syntax im Kapitel der Implementierung.

2.2.3 IPv4/v6

IronGate setzt in seiner bisherigen Version auf die Kernelunterstützung von IPv4 bzw. IPv6. Dies setzt korrekt konfigurierte Interfaces („ifconfig“, „ip“) sowie Routen voraus (gesetzt durch dieselben Tools). Bei der Implementierung von IronGate wurde darauf geachtet, alle Möglichkeiten der Kernelunterstützung von IPv4 auszunutzen, wodurch auch Paketlängen, Paketmarker, usw. ihren Einsatz bzw. ihre Auswertung finden. Zur Steuerung der Paketfilter-Tabellen wird iptables in der Version für IPv4 verwendet.

Mehr Informationen zu IPv4 finden sich unter [3] und die für die Verwendung unter IronGate wichtige Syntax von iptables im Kapitel der Im-

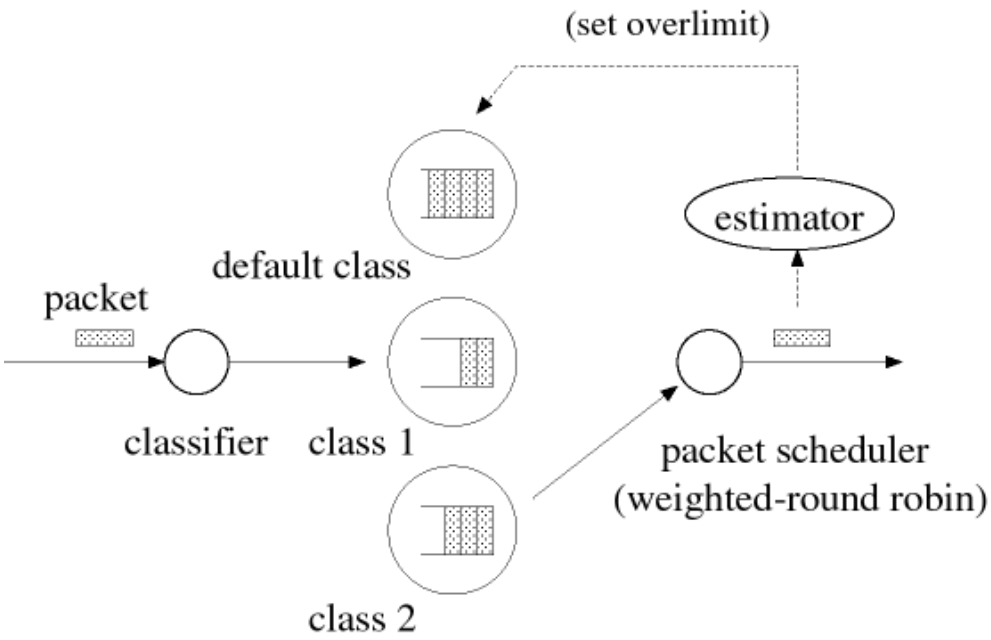

Abbildung 2.4: *Der CBQ-Algorithmus*

plementierung.

IPv6 als zukünftiger Ersatz für IPv4 wird bspw. unter [4] ausführlich behandelt.

2.2.4 ADS

Die ADS (Active Directory Structure/System) löst seit Windows 2000 die Domänencontroller ab (welche in ihren Funktionen dem Novell-Bindery nicht unähnlich war, mit dem Unterschied, dass Novell schon seit 1996 auf ein Directory in Form der NDS setzt). Die ADS ähnelt sehr bzw. baut z.T. auf LDAP.

Da zahlreiche Umgebungen ihre Benutzerdatenbank auf die ADS aufgebaut haben, diese jedoch v.a. in Bezug auf eine externe Benutzerauthentifikation nicht über LDAP-Schnittstellen ansteuerbar ist, verfügt IronGate über ein eigene ADS Backend. Dieses wird über ein Perl-Modul einerseits unter Linux und andererseits unter Windows angesprochen, setzt also die Installation dessen im momentanen Status noch voraus.

Mehr Informationen zu ADS finden sich unter [5] und die für die Verwendung unter IronGate wichtigen Programmbibliotheken und Tools im Kapitel der Implementierung.

2.2.5 mySQL

mySQL ist eine für die private und nicht-kommerzielle Nutzung kostenlos verfügbare Datenbank, welche vor allem durch ihre Stabilität und ihre extrem hohe Geschwindigkeit als ausgezeichnet für Anwendungen mit sehr vielen Einträgen und wenig Relationen (wie IronGate) gilt. mySQL ist an sich eine relationale Datenbank, getrimmt jedoch eben auf Geschwindigkeit und nicht auf Funktionsumfang, welche SQL als Abfragesprache erwartet. mySQL ist für zahlreiche Betriebssysteme verfügbar, was es IronGate auch ermöglicht, die Datenbank nicht nur auf andere Server, sondern ggf. auch auf andere Betriebssysteme auszulagern.

IronGate benötigt mySQL zwingend, um die internen Datenbanken zur Verfügung zu haben (konkret sind dies die tables online, authlog, packetlog, users, groups, freeids und classes). IronGate unterstützt mySQL auch als Backend, um bspw. vorhandene Benutzerdatenbanken integrieren zu können.

Mehr Informationen zu mySQL finden sich unter [6] und die für die Verwendung unter IronGate wichtigen Programmbibliotheken im Kapitel der Implementierung.

2.2.6 LDAP

LDAP (Lightweigth Directory Access Protocol) ist ein vereinheitlichter Standard, wie man Directories abfragen und manipulieren kann. Directories, hier nur ganz kurz angesprochen, unterscheiden sich bspw. von plaintext-Files, relationalen Datenbanken oder Benutzermanagement-Strukturen wie die Bindery durch ihre Baumstruktur, d.h. jeder Benutzer ist im Endeffekt ein Blatt in einem Baum, dessen Stamm eine Organisation und die Äste Unterorganisationen sind. Durch diese Art der Verwaltung lassen sich extrem große Mengen an Informationen gerade in Bezug auf Benutzer, Benutzergruppen usw. sehr übersichtlich speichern und verwalten. Als Nachteil gilt die relativ geringe Zugriffsgeschwindigkeit, weshalb das andauernde Ändern von Einträgen (in einer Geschwindigkeit und mit einem Datenaufkommen, mit dem bspw. IronGate die mySQL-Tabellen manipuliert) möglichst zu vermeiden ist. Aus diesem Grund wurde für die interne Datenspeicherung von IronGate auch auf eine relationale Datenbank zurückgegriffen.

Berühmte Vertreter der Directory Struktur waren in den letzten Jahren die Microsoft ADS und Novells NDS, wobei oftmals nur partiell der LDAP-Kommmunikationsstandard mitimplementiert wurde. Gerade unter freien Betriebssystemen jedoch wird der LDAP-Standard streng eingehalten, was es durch IronGate's LDAP-Backend ermöglicht, einfach in eine vorhandene LDAP-Struktur zu migrieren.

Mehr Informationen zu LDAP finden sich unter [7] und die für die Verwendung unter IronGate wichtigen Programmbibliotheken im Kapitel der

Implementierung.

2.2.7 Dynamic DNS

Dynamic DNS ist eine recht unbekannte und in vielen Fällen auch gefährliche Art und Weise der Manipulation an sich statischer DNS-Zonen. Es wurde populär durch das Zuweisen fixer DNS-Namen an dynamisch vergebene IPs (dyndns.org bspw.) und beim Einsatz in DHCP-Servern, welche dadurch einer Maschine immer ihren Namen in einer DNS-Zone eintrug, selbst wenn sich die IP änderte. Dynamic DNS ist inzwischen bei fast allen DHCP- und DNS-Servern möglich, wie auch Windows-Servern ab Version 2000 (und bei Linux bspw. ISC BIND und ISC DHCPd).

IronGate nutzt Dynamic DNS jedoch auf eine ganz andere Art und Weise - unabhängig von der MAC, der IP oder dem Namen einer Maschine selbst (der im Großen und Ganzen auch nur in einem NetBIOS Netz mehr relevant ist) vergibt IronGate A NAME und IN PTR-DNS Records an jeden Benutzer, welcher sich über IronGate erfolgreich bei einem Server anmeldet. Auf diese Weise ist nicht jede Maschine, sondern jeder Benutzer direkt über seinen Benutzernamen anstatt einer IP, die sich ja ändern kann, erreichbar. So kann man bspw. jedem Benutzer eine Nachricht über den NetBIOS Nachrichtendienst senden, egal, wo er sich befindet. Weiter gesponnen kann man auf diese Art und Weise Zugriffe auf jeden Server, welcher beim Aufbau einer Verbindung einen Reverse-DNS-Lookup initiiert, sofort relationell zu dem einloggenden Benutzer speichern, da auf den Reverse-Lookup bzgl. seiner IP-Adresse ja seinen Benutzernamen zurückliefert. Durch die volle Kontrolle über die Dynamic DNS Updates alleine durch IronGate (nicht in Kombination mit einem DHCP-Server, welcher selbige Eintragungen vornimmt), verlieren die Zonendaten auch nicht an Konsistenz und halten stets nur diejenigen Namen und Adressen, welche entweder statisch eingetragen wurden oder momentan eingeloggt sind.

Mehr Informationen zu Dynamic DNS finden sich unter [8] und die für die Verwendung unter IronGate wichtigen tools im Kapitel der Implementierung.

2.3 Die einzelnen IronGate-Module auf Systemebene

Da beim Ablauf und im Kapitel der Implementierung detailliert auf die einzelnen Funktionen und den Programmcode eingegangen wird, soll dieser Abschnitt in erster Linie dazu dienen, die Technik auf systemnaher Ebene ein wenig näher zu betrachten.

2.3.1 Der Server-Daemon

Der Server-Daemon bzw. Serverprozess von IronGate wurde durchgehend in Perl entwickelt. Perl kann bis zum jetzigen Zeitpunkt kein (den Entwicklern

zufolge) stabiles Multi-Threading, weshalb für das Handeln der einzelnen Client-Verbindungen Forking verwendet wird. Forking basiert darauf, den eigenen Prozess, versehen mit einer anderen PID, zu kopieren und parallel zum „Mutterprozess“ weiterlaufen zu lassen. Da hierbei, wenn immer es möglich ist, eine Ressourcen-Teilung erfolgt, leidet die Performance darunter nur kaum und auch die Auslastung des Memories etc. ist unwesentlich höher als bei der Verwendung mehrerer Threads.

IronGate übergibt im Detail an den geforkten Prozess die offenen Datenbankverbindungen, die Presettings und die gerade geöffnete Verbindung. Bei jedem neuen Verbindungsaufbau wird ein neuer Prozess geforked, welcher die Verbindung übernimmt und abhandelt. Nach Abhandlung beendet sich der Prozess wieder. Somit existieren im System immer soviele IronGate-Prozesse wie Benutzer eingeloggt sind plus eins. Um die Implementierung einer IPC (Interprozesskommunikation) nach dem POSIX-Standard zu umgehen, erfolgt die IPC quasi „passiv“ über die mySQL-internen Datenbanken. So stimmt sich IronGate immer mit der Datenbank ab, und wenn die Datenbank bspw. in Bezug auf die LogOff-Variable eines Benutzers von außerhalb modifiziert wurde, reagiert der Prozess, welcher den entsprechenden Benutzer abhandelt, mit einem LogOff darauf. So kann über Verändungen von Werten innerhalb der mySQL-Tabellen die Prozesssteuerung durchgeführt werden. Beim Shutdown der Maschine bzw. von IronGate werden alle Benutzerausgeloggt, womit sich alle Prozesse minus einem, welcher dann über ein TERM-Signal beendet wird, sauber abgeschlossen. Auf diese, wenn auch eher unorthodoxe Art und Weise der IPC, erspart man sich sehr viel systemnahe Programmierarbeit und erntet im Prinzip die gleichen Ergebnisse, wie problemlose Langzeittests ergaben.

Wann immer der Server externe Tools (also keine Programmbibliotheken) in Anspruch nehmen muss, öffnet Perl eine Pipe zum gewünschten Prozess, welche eine I/O-Kommunikation mit diesem zulässt. So werden die entsprechenden Parameter beim Öffnen der Pipe übergeben, und die Ergebnisse des Befehls, sofern diese auswertbar sind, über die Pipe an den IronGate-Prozess zurückgegeben und von diesem ausgewertet.

Die Basiskonfiguration des Mutterprozesses an sich erfolgt über eine Konfigurationsdatei bspw. im „/etc“ Verzeichis, welche als Inline-Code über die „require“-Funktion von Perl integriert wird. Der Aufbau dieser Konfigurationsdatei wird noch im Kapitel der Implementierung besprochen.

Der Serverprozess kann aufgrund folgender Gründe ausschließlich als „root“ bzw. mit der relativ aufwendigen Konfiguration unter der Verwendung von „su“ laufen:

- Das Öffnen von Sockets ist ausschließlich root gestattet
- Das Ändern von Netfilter-Regeln (iptables) ist ausschließlich root gestattet

- Das Ändern von Bandbreitenregeln (CBQ über iproute2 bzw. tc) ist ausschließlich root gestattet

- Die desweiteren notwendigen Tools (arp, nsupdate etc.) sind bei den meisten Linux-Installationen in Verzeichnissen untergebracht, auf die wiederum nur root Zugriff hat

Das Ausführen von Prozessen als root, v.a. wenn sie einen Netzwerksocket öffnen, gilt als riskant - daher wurde bei der Implementierung von IronGate erstens auf eine Sprache gesetzt, welche für C-typische Probleme wie Buffer Overflows nicht anfällig ist, und zweitens bei der Implementierung sehr darauf geachtet, einen Ausbruch aus der Umgebung unmöglich zu machen sowie bpw. SQL-Injections zu verhindern (Überprüfung der Übergabewerte durch Regular Expressions).

2.3.2 Die Clients

Die erste Version des Clients (Abbildung 2.5) wurde in Visual Basic 6 entwickelt und verfügte nur über marginale Kontrollfunktionen, keine Verschlüsselung, keine Klassen und kein Informationsinterface seitens des Servers. Darüberhinaus war diese Verion ob ihrer Natur nur unter Windows lauffähig, und wurde daher im Laufe der Weiterentwicklung von IronGate von einer Java-Version, welche auch als Applet implementiert wurde, ersetzt. Zwischenzeitig gab es eine experimentelle Version in C++ für Performance-Tests, diese ist jedoch mit der aktuellsten Version von IronGate nicht mehr kompatibel.

Der Java-Client (Abbildung 2.6) arbeitet auf Multithreading-Basis und verwendet für den Zugriff auf externe Programme (momentan nur GPG) die Kommunikation über Textfiles. Sobald IronGate gänzlich nach Java übersetzt wird, fällt die Verwendung externer Programm flach, und die Verschlüsselung erfolgt ausschließlich über mitgelieferte Java-Packages. Momentan verhindert diese eine Inkompatibilität zwischen den Bibliotheken von Perl und Java (siehe hierzu [9]).

Die Connection mit dem Server wird als eigener Thread ausgeführt, damit auch das Keep-Alive-Signal und auch das Information-Fetching vom Server. Die Windows-Steuerung erfolgt über einen weiteren Thread. Der Client beherrscht verschlüsselte und unverschlüsselte Verbindungen sowie drei Methoden der Konfiguration:

- Manuelle Konfiguration über die Eingabe der Parameter des Hosts, der IP, des Ports und der Verschlüsselung über den Configuration-Dialog (Abbilung 2.7)

- Automatische Konfiguration via Konfigurationsdatei im Verzeichnis des Clients (Konfigurationsdatei-Format siehe [9])

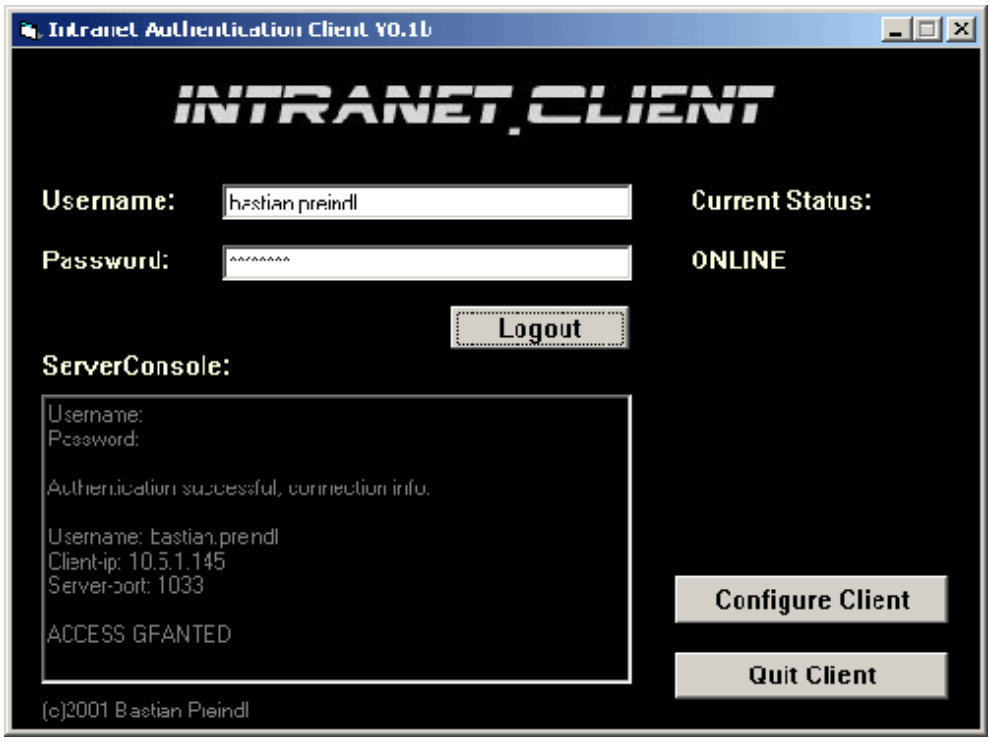

Abbildung 2.5: *Der VisualBasic-Client*

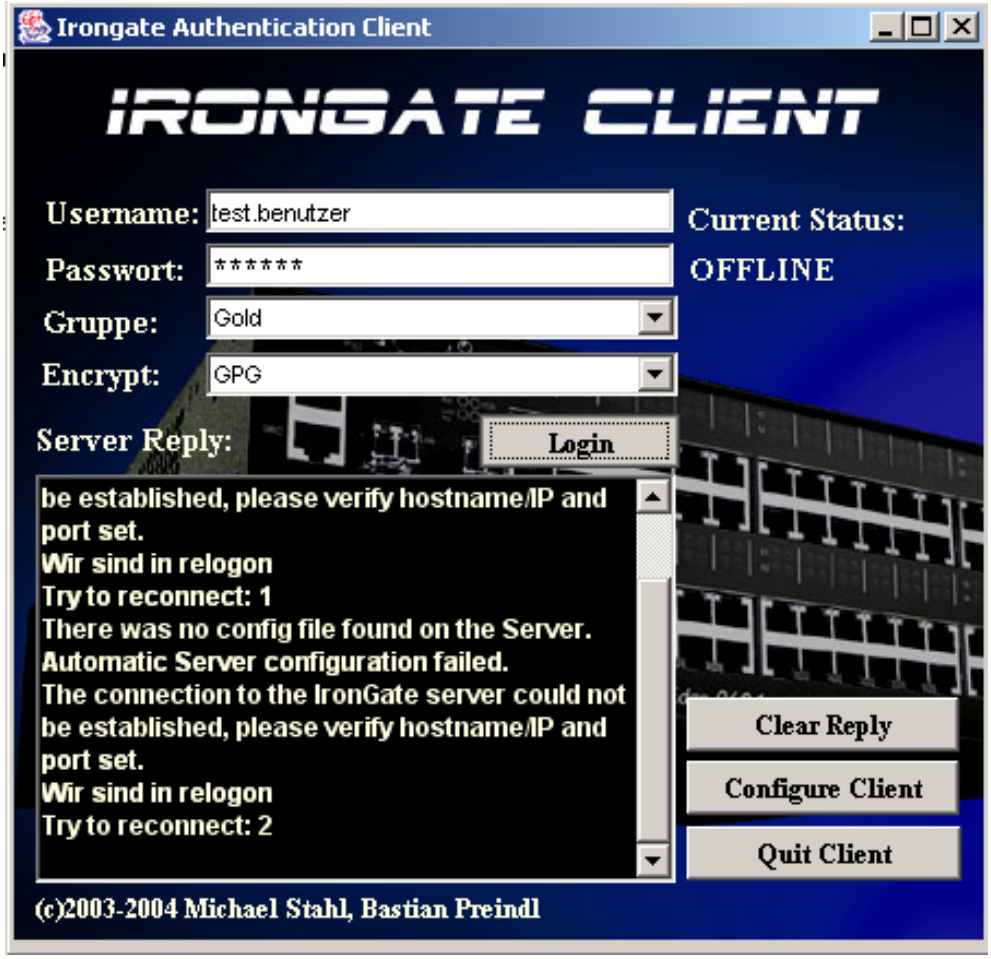

Abbildung 2.6: *Der Java-Client Login*

- Automatische Konfiguration via Auto-Configuration-Funktion ähnlich der eines Proxies mit automatischer Einstellungssuche. Hierbei liegt die Konfigurationsdatei am IronGate-Server und wird beim Start des Clients vom Server geladen (über die HTTP-Umleitung, wie später noch besprochen, muss die IP des Servers hierbei nicht bekannt sein)

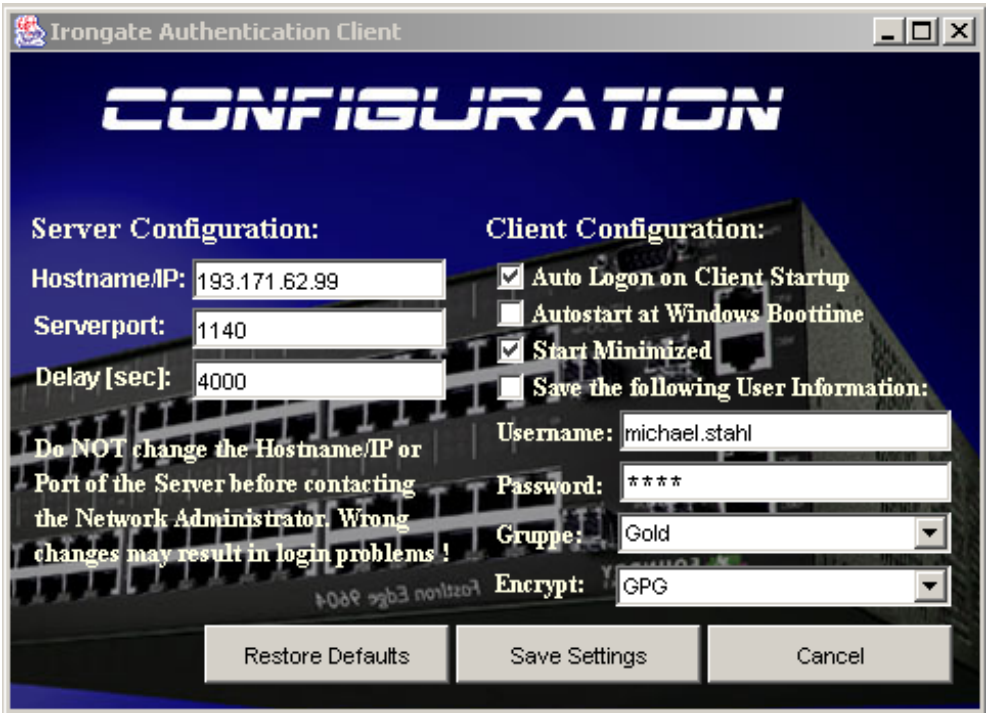

Abbildung 2.7: *Der Java-Client Konfigurationsdialog*

Je nach Anwendungsbereich von IronGate macht die eine oder andere Konfiguration mehr Sinn. In den allermeisten Fällen dürfte die automatische Konfiguration via File am Server die schnellste und benutzerfreundlichste Methode sein. Die Reihenfolge der Abarbeitung ist:

- Automatische Konfiguration via Auto-Configuration-Funktion
- Automatische Konfiguration
- Manuelle Konfiguration

Über den Java-Client als Class hinaus gibt es auch noch eine Applet-Version (Abbildung 2.8, welche nur Auto-Configuration kennt und nicht mit Verschlüsselung umgehen kann (da das Applet nicht GPG zugreifen kann). Diese Version ist in erster Linie für alle die Fälle gedacht, in denen eine lokale Installation des Clients nicht möglich ist oder sinnvoll erscheint (z.B.

Gastbenutzer, strenge Sicherheitseinstellungen etc.). Die einzige Voraussetzung hierfür ist ein Java-fähiger Browser, wie für den Java-Class Client ein installierter Java-Interpreter im Pfad der Maschine.

Abbildung 2.8: *Das Java-Applet*

Da der Java-Client selbst nicht Teil dieser Arbeit war, sei für alle weiterführenden Informationen auf [9] verwiesen.

2.3.3 Das Web-Interface

Das Web-Interface (Abbildung 2.9) setzt auf dem freien Hypertext Precompiler PHP auf und arbeitet ausschließlich mit HTML4 und DHTML auf Clientseite. Es ist eine rein serverbasierte Oberfläche zur Wartung und Manipulation der IronGate-Tables. Das Interface wurde in Hinsicht auf Benutzerfreundlichkeit entworfen und verfügt über ein einheitliches Design. Der Zugriff darauf erfolgt (kann man IronGate als Blackbox vertreiben, womit dies vorkonfiguriert wäre) über HTTPS, also SSL. Dies hält den hohen Sicherheitsstandard aufrecht, welcher durch die Strong Encryption zwischen Client und Server geschaffen wurde.

Wie schon beschrieben, verhält sich IronGate sehr unorthodox, was die Eingriffe in laufende Kindprozesse anbelangt. Da diese sich ausschließlich über zentral gespeicherte und geänderte Werte, vornehmlich in der Benut-

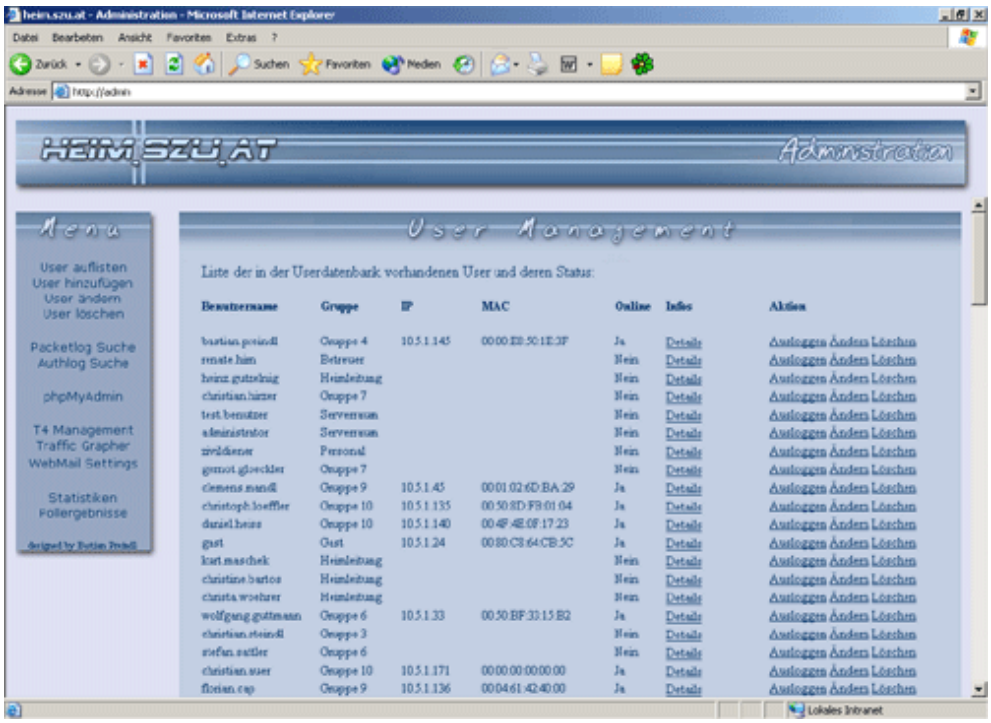

Abbildung 2.9: *Das Web-Frontend - Screenshot*

zerdatenbank, steuern lassen, gilt es, diese Werte so einfach wie möglich zu manipulieren. Und dabei wird auf einen Webserver mit serverseitiger Skriptverarbeitung gesetzt, was zumindest von der Wartungsseite her vollkommene Plattformunabhängigkeit garantiert. Jeder noch so einfache Browser, selbst ein Terminalbrowser wie Lynx, genügt, um eine vollständige Wartung der dynamischen Werte zu garantieren. Für alle oben genannten Funktionen gibt es HTTP-Frontends, welche problemlos über HTTPS angesprochen werden können, also sicher verschlüsselt über SSL. Dem Auswerten der geloggten Informationen bei Bedarf wurde eine ebensohoher Stellenwert zutage wie der Benutzerverwaltung selbst.

Sehr relevant für eine schnelle und unkomplizierte Verwaltung selbst großer Mengen an Benutzern sind Masken für das einfache und schnelle Serienanlegen von Benutzern. Während beim Anlegen nur die allerwichtigsten Felder konfiguriert werden müssen, können weitere Konfigurationen und Werte für jeden einzelnen Benutzer zu jedem beliebigen Zeitpunkt hinzugefügt werden. Diese Werte überschreiben dann auch die Gruppenkonfigurationen, an welchen Benutzer beteiligt sein können, sofern das Backend so etwas vorsieht. Die Gruppenkonfiguration wird als erstes für den eingeloggten Benutzer gesetzt, dessen etwaige individuelle Werte überschreiben diese dann im Bedarfsfall (temporär natürlich nur, pro Sitzung aufs neue).

Neben diesen grundlegenden Wartungsmöglichkeiten per Browser gibt es, dank der verwendeten relationalen Datenbank oder dem Verzeichnisdienst mit offenen Schnittstellen, die Möglichkeit, durch einfach gestrickte eigene Skripts tausende Benutzer innerhalb von Minuten anzulegen und zu konfigurieren. Außerdem ist man nie darauf beschränkt, die eigens für IronGate entwickelten Wartungsskripts zu verwenden, sondern kann sich dank völlig offener Standards eigene Skripts stricken und vorgefertige Wartungssoftware verwenden. Dies geht angesichts der mehreren Backends inkl. LDAP sogar soweit, Benutzer z.B. innerhalb des Windows ADS Baums anzulegen, und IronGate diese Daten direkt von dort ohne Redundanz übernehmen zu lassen, indem die LDAP-Schnittstelle von Windows genutzt wird.

Die Grundkonfiguration von IronGate kann verständlicherweise nicht innerhalb von Datenbanken gespeichert werden, da diese immer im Zugriff sein müssen, und darüber hinaus zu einem rekursiven Problem führen würden, wären die Daten über die Datenbank in z.B. eben dieser Datenbank gespeichert. So wird die einmalige Grundkonfiguration mit einigen Basiswerten bei der Installation von IronGate (oder im Bedarfsfall später) in einem durchdokumentierten und leicht verständlichen Konfigurationsfile gespeichert.

Da das Web-Interface zwar im Zuge dieser Arbeit entwickelt wurde, nicht jedoch den Kern dieser bildet, seien noch einige Details im Kapitel der Implementierung erwähnt, darüber hinaus jedoch nicht weiter behandelt.

2.3.4 Die Backends

Die Ansteuerung der implementierten Backends geschieht einmal mehr auf eine nicht ganz übliche Weise. Da im Vorhinein, sprich während der Implementierung, nicht klar sein kann, wieviele Backends von welcher Art letztendlich vom Server angesprochen werden sollen bzw. abgefragt, hat sich bei der Entwicklung von IronGate folgender Ansatz durchgesetzt: Wie das Konfigurationsskript über „require“ in das Programm eingewoben wird, werden auch die einzelnen Backend-Codezeilen ggf. auch mehrfach eingebunden, und zwar dynamisch.

Jedes Backend-Modul enthält, vom Administrator eingetragen, die Variablen, welche den Backend-Host, Port, usw. definieren. Dann wird das Modul als .pl-Datei in einem im Konfigurationsfile definierten Unterverzeichnis abgespeichert. Wenn man dreimal bspw. eine ADS-Backend-Verbindung benötigt, so speichert man dasselbe Modul mit unterschiedlichen Konfigurationen unter unterschiedlichen Namen in diesem Verzeichnis ab.

IronGate arbeitet das Verzeichnis als Liste ab und fügt jedes Modul als Code über „require“ ein. Jedes Modul überschreibt ggf. die Ergebnisse des vorhergehenden (bezogen auf Rechte, nicht auf eine allgemeine Gültigkeit des Benutzernamens), womit im Endeffekt, nach Abarbeitung aller Backendmodule, die Summe aller Rechte und Regeln aus den einzelnen Backends zusammengezogen wurde. Zuletzt wird die interne Benutzerdank und deren

Gruppentabellen abgefragt, wobei auch die Regeln dieser bereits vorhandene überschreiben können (so kann ein Benutzer zwar seine Authentifikation bei einem Backend einholen, welches auch seine Gruppenzugehörigkeit zurückliefert, seine Einschränkungen dann aber von der internen Gruppendatenbank, sofern eine Gruppe mit gleichem Namen existiert).

2.4 Die einzelnen Funktionen im Detail

In den folgenden Abschnitten wird jede einzelne (besondere) Funktion von IronGate im Detail erklärt.

2.4.1 Handshake

Wie schon im Abschnitt „Client-Server Kommunikation“ detailliert aufgezeigt, werden während des Handshakes folgende Punkte geklärt (vgl. Abbildung 2.10):

1. Der Client initiiert durch Öffnung eines Sockets die Verbindung
2. Der Server fragt über „Client:“ und RETURN die Version des Clients ab
3. Der Client antwortet bspw. mit „java“ und RETURN
4. Der Server fragt über „Encryption:“ und RETURN die gewünschte Form der Verschlüsselung ab
5. Der Client antwortet bspw. mit „gpg“ und RETURN (ab dieser Stelle wird davon ausgegangen, dass GPG als Verschlüsselung verwendet wird - wenn keine Verschlüsselung verwendet wird, wird der Austausch der Schlüssel und des Salts übersprungen und bei „Username:“ fortgesetzt - Client müsste dafür noneünd RETURN übermitteln)

Die Version des Clients spielt deshalb eine Rolle, weil in der zukünftigen Entwicklung bzw. in der vorhergehenden verschiedene Handshakes verwendet wurden oder verwendet werden könnten, bzw. verschiedene Protokolle der Client-Server-Kommunikation ausschlaggebend waren oder werden könnten, und der Server daher entsprechend auf die unterschiedlichen Clients reagieren muss.

Die Wahl einer reinen String-Kommunikation (wo auch eine binäre in Form von Objektübergaben denkbar gewesen wäre) dient der Offenheit des Protokolls gegenüber anderen Systemen und Programmiersprachen.

RETURN ist im Falle der C/S-Kommunikation ein CRLF (Carriage Return/Line Feed) im UNIX-Sinne, also Character 12 gefolgt von Character 15. Windows-Applikationen kennen ein anderes CRLF, weshalb bei der Übergabe des RETURN auf der Einhaltung der UNIX-Vorgaben zu achten ist.

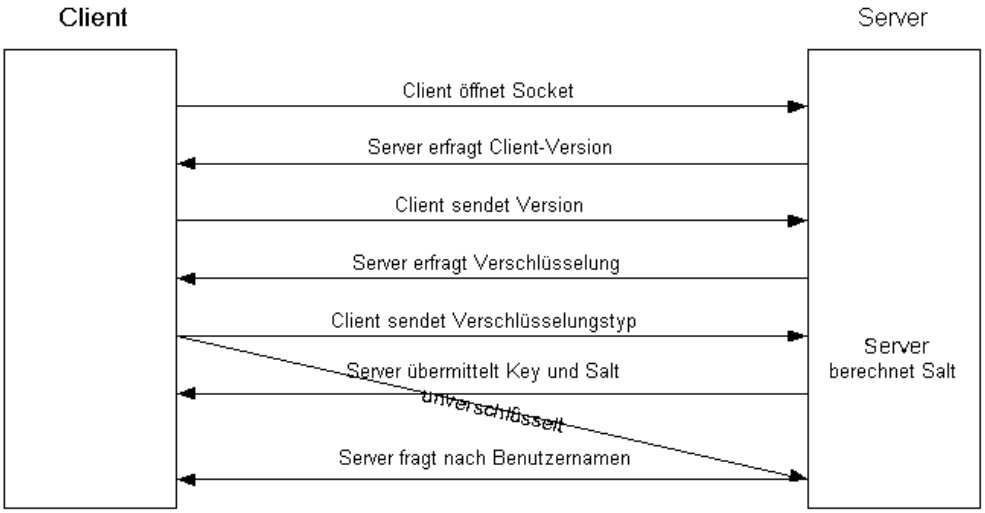

Abbildung 2.10: *Der Handshake zwischen Client und Server*

2.4.2 Verschlüsselung

Zur Verschlüsselung der Kommunikation während des Logins wird, wie bereits erwähnt, GPG verwendet. Von Serverseite her muss dazu vorher ein Schlüsselpaar erstellt werden, welches auf das Kennwort „irongate“ hin ausgerichtet ist (siehe Abschnitt „verwendete externe Tools“). Der öffentliche Schlüssel (public key) wird bei der Authentifikation bzw. dem Handshake an den Client übermittelt, wohingegen der private Schlüssel stets auf dem Server verweilt (und durch den Administrator nur „root“ zugänglich sein darf).

Nachdem der Client eine Verschlüsselung wünscht (Übergabe von „gpg“ beim Handshake als Verschlüsselungstypus) übermittelt der Server dem Client den öffentlichen Schlüssel. Vorher wird eine Integer-Random-Zahl generiert, welche dem Client mit dem Schlüssel mitübermittelt wird. Der Client fügt dem jeweiligen String (Benutzername und Kennwort) die Zahl hinzu und verschlüsselt diesen daraufhin. Damit ist eine einmalige verschlüsselte Sequenz kreiert worden. Der verschlüsselte Wert wird dem Server übermittelt, welcher daraufhin den String mit dem Private Key entschlüsselt und die angehängte Zahl mit der von ihm gesendeten vergleicht. Stimmt sie überein, wird der Benutzername und im weiteren Falle das Kennwort akzeptiert (welches auf die gleiche Art und Weise überprüft wird). Auf diese Art und Weise wird eine Replay-Attacke unmöglich, da die Chancen, dass ein Benutzer zweimal die idente Random-Zahl als additiven Wert vom Server vorgeschrieben bekommt, 1:(2 hoch 16) stehen. Es würde also 65536 Relogons benötigen, um denselben Zahlenwert noch einmal zu erwischen. Und selbst dann weiß der Angreifer noch immer nicht den Benutzernamen und

das Kennwort, er hat lediglich eine authorisierte Verbindung bekommen. Eine Erhöhung des Random-Wertes auf bigint oder noch höher lässt die Chancen exponentiell weiter sinken.

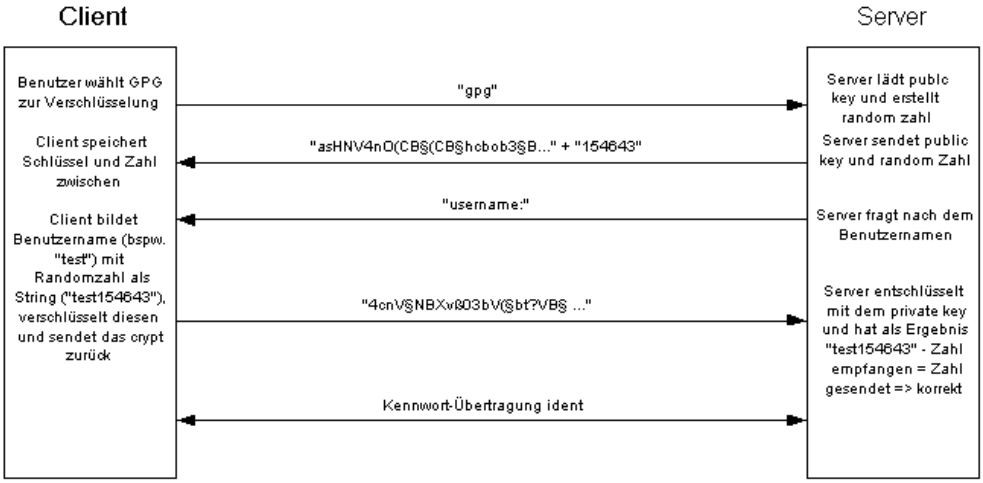

Abbildung 2.11: *Verschlüsselte Kennwortübertragung*

In Abbildung 2.11 ist dieser Ablauf im Falle einer einwandfreien Kommunikation noch einmal schematisch dargestellt. Verglichen hierzu zeigt Abbildung 2.12 eine Kommunikation zwischen dem Server und einem Man in the Middle (bzw. einem sniffenden Hacker).

Im zweiten Falle hat der Angreifer zwar die Übermittlungssequenz mitaufgezeichnet, diese ist jedoch für ihn wertlos, da sich die verschlüsselte Zahl geändert hat. Gegen eine „wirkliche“ Man-in-the-Middle Attacke, in der der Angreifer den Server mitsimuliert und seinen eigenen public und privatekey verwendet, kann dieses Verfahren freilich nicht bestehen - um diese zu entschärfen, müssten Zertifikate eingesetzt werden, welche es unmöglich machen würden, die Clients von Router zu Router springen zu lassen oder gar eine One-Click-Installation zu ermöglichen. In Netzwerken mit sehr hohen Sicherheitsvoraussetzungen ist es problemlos möglich, GPG so zu konfigurieren, dass ein zertifizierter Schlüssel Voraussetzung ist - dies obliegt dann jedoch dem Systemadministrator.

2.4.3 Authentifizierung

Wie bereits im vorigen Kapitel beschrieben werden Benutzername und Kennwort an den Server übertragen. Dieser führt, bevor die eingelesenen Variablen auf irgendeine andere Art und Weise weiterverarbeitet werden, eine Überprüfung ihrer Syntax mit Hilfe von Regular Expressions durch, um späteren Angriffen wie z.B. einer mySQL-Injection entgegenwirken zu können.

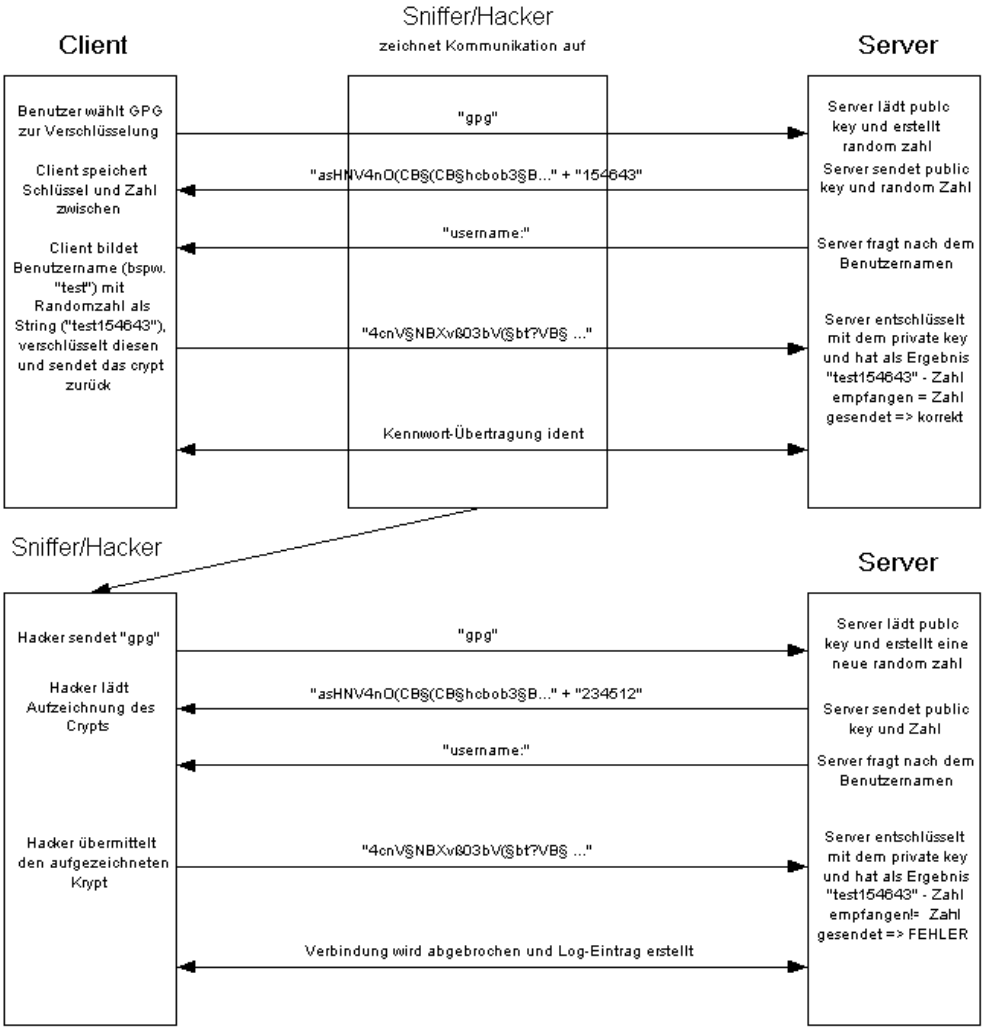

Abbildung 2.12: *Gesniffte Attacke*

Nach der Übermittlung von Benutzername und Kennwort erfragt der Server noch die Benutzerklasse. Diese sei weiter unten beschrieben, unterscheidet sich jedoch grundsätzlich in Default, Bronce, Silver und Gold.

Die Authentifizierung in Summe setzt sich also aus folgenden Punkten zusammen:

1. Handshake
2. Verschlüsselungswahl
3. Austausch des Schlüssels und der Randomzahl
4. Austausch von Benutzername und Kennwort sowie der Benutzerklasse (diese unverschlüsselt)
5. Überprüfung der übergebenen Werte gegen die Backends und die interne Datenbank (Backend-Ansteuerung)
6. Setzen aller Regeln
7. Starten der Keep-Alive-Sequenz

2.4.4 Keep-Alive Sequenz

Die Keep-Alive Sequenz dient mehreren Aspekten:

- Sie versichert dem Server, dass der Client nach wie vor aktiv ist, und sich weder das Programm, noch die Verbindung noch die Maschine des Clients einem Ausfall zum Opfer gefallen sind
- Sie „schiebt“ quasi den Prozess am Server an - der Server wartet immer zehn Sekunden auf ein Signal vom Client, und arbeitet, sobald dieses eintrifft (eben das Keep-Alive-Signal in Form eines „.“-es), sämtliche Prozeduren ab, welche durchgeführt werden sollen, während der Client online ist (siehe weiter unten)
- Der Serverprozess muss in Abständen von maximal 20 Sekunden der internen Datenbankmelden, dass er selbst noch aktiv ist - wenn nicht, beendet ein Prozess-Watchdog den Prozess selbst bzw. kennzeichnet den Client als ausgeloggt. Dies dient als Schutz davor, dass eine Systemüberlastung einen Prozess, ohne die Accounting-Daten zu speichern, über den Kernel aufgibt. Das würde im schlimmsten Falle dazu führen, dass Onlinezeit und übertragenes Volumen mehrerer Tage nicht aufgezeichnet werden.

Nachdem eine Keep-Alive-Sequenz zu Ende ist, bzw. der Server kein Signal mehr innerhalb der vorgegebenen Zeit vom Client erhält, beendet dieser

die Verbindung von sich aus und deklariert den Benutzer als „unsauber ausgeloggt“, dennoch wird er accounted.

Die eigentliche Keep-Alive-Sequenz (Kommunikation vor und nach einem Zyklus) ist in Abbildung 2.13 schematisch dargestellt.

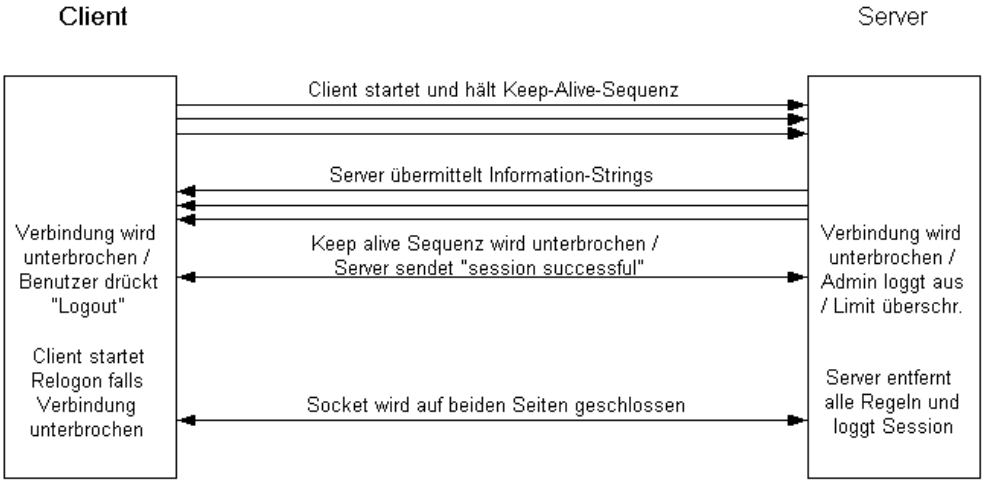

Abbildung 2.13: *Keep-Alive Sequenz*

Während eines Keep-Alive-Zyklus führt der Serverprozess folgende Prozeduren durch:

- Abfragen der iptables-Packet- und Bytecounter für den entsprechenden Benutzer (seine Chain)
- Abfragen der Momentanzeit in Unixtime [s]
- Berechnung der Differenzzeit seit dem letzten Zyklus [s]
- Berechnung der bereits verweilten Online-Zeit [s]
- Berechnung der noch übrigen Zeit anhand der während der Backend-Abfrage errechneten spätesten Logout-Zeit [s]
- Berechnung der Differenzmenge an Paketen up- und download seit des letzten Zyklus jeweils in udp und tcp [n]
- Berechnung der Differenzmenge an Bytes up- und download seit des letzten Zyklus [byte]
- Berechnung der bereits geladenen und gesandten Menge an Paketen und Bytes sowie gesamt [byte]
- Berechnung der noch übrigen Menge an Bytes up- und download sowie gesamt [byte] bis zum maximalen Volumen

- Abfrage des Logoff-Tables nach dem Benutzerstatus - wurde der Benutzer während des letzten Zyklus als offline deklariert, wird der Benutzer als ausgeloggt gekennzeichnet und die Logoff-Sequenz wird eingeleitet [i]

- Initialisierung des Logoffs falls in irgendeiner Form eine Überschreitung vorliegt mit Setzen des Fehlerstatus auf die entsprechende Überschreitung

- Auswertung der gesammelten Werte (Byte- und Packetcounter sowie Syncounter) durch die Heuristic-Usage-Engine, welche daraufhin die Usage-Variable entsprechend setzt [string]

- Errechnung der idle-time des Benutzers: Erhöhen der idle-time-Variable, falls der Benutzer auch in diesem Zyklus keinen Traffic hatte, ansonsten setzen der Variable auf 0 [n]

- Auswertung der gesammelten Werte durch die Load-Engine, welche den Load des Benutzers klassifiziert (idle, low, medium, high, extreme) [string]

- Auswertung des Traffics hinsichtlich der Ergebnisse der Heuristic-Usage-Engine hinsichtlich der Abnormität (bspw. Portscans) als 0 oder 1 [i]

- Berechnung des momentanen Datendurchsatzes Up- und Download anhand der errechneten Volumensdifferenzen und der vergangenen Zeit seit dem letzten Zyklus [byte/sec]

- Speichern aller errechneten Werte im Online-Table (siehe unten)

- Übergabe einiger Werte an den Client zur Selbstkontrolle und Benutzerinformation (siehe unten)

Folgende Werte werden im Online-table stets aktualisiert:

- *onlinetime* - Wie lange der Benutzer bereits online ist

- *timeleft* - Wie lange der Benutzer noch online sein darf

- *volupleft* - Wieviel der Benutzer noch Uploadvolumen übrig hat

- *voldownleft* - Wieviel der Benutzer noch Downloadvolumen übrig hat

- *volleft* - Wieviel der Benutzer noch an insgesamtem Volumen übrig hat

- *volupnow* - Wieviel der Benutzer bereits hochgeladen hat

- *voldownnow* - Wieviel der Benutzer bereits heruntergeladen hat
- *volnow* - Wieviel der Benutzer insgesamt schon transferiert hat
- *typeofuse* - Art der Verbindungsnutzung (scanning, leeching, gaming, streaming, surfing, idle)
- *alive* - Der Prozess hat noch eine aktive Verbindung
- *idletime* - Zeit, die der Benutzer nichts getan hat seit der letzten Aktivität
- *abnormal* - Ob der Traffic abnormal ist (=scanning, leeching)
- *throughputup* - Der momentane Datendurchsatz im Upload
- *throughputdown* - Der momentane Datendurchsatz im Download
- *producedload* - Der erzeugte Traffic als Ausdruck (low, medium, high, extreme)
- *syncount* - Wieviel Verbindungen der Benutzer bereits initialisiert hat
- *packetcountup* - Wieviele Pakete der Benutzer bisher versandt hat
- *packetcountdown* - Wieviele Pakete der Benutzer bisher empfangen hat

Folgende Werte werden dem Client regulär zur Übergabe an den Benutzer übermittelt:

- *Authentication successful, connection info:*
- *Username - Benutzername*
- *Client-IP - Die IP des Clients*
- *Client-MAC - Die MAC-Adresse des Clients*
- *Guessed Client-OS - Das erratene Client-Operating-System, welches beim Login mitübergeben wird (nach der Benutzerklasse)*
- *Encryption - Die verwendete Verschlüsselung*

Falls erweiterte Informationsübermittlung aktiviert wurde (vom Administrator), werden außerdem folgende Werte mitübergeben:

- *Status - Die momentane Benutzerklasse*
- *Guessed Usage - Was der Benutzer vermutlich für eine Aktivität hat*
- *Traffic up this session - Das gesamte Upload-Volumen bisher*

- *Traffic down this session - Das gesamte Download-Volumen bisher*
- *Upload Throughput - Der momentane Upload-Durchsatz*
- *Download Throughput - Der momentane Download-Durchsatz*
- *Online since - Seit wann der Benutzer online ist*
- *Estimated Logoff Date/Time - Berechnete späteste Logout-Zeit*
- *Online-time counter - Wie lange der Benutzer schon online ist in Sekunden*
- *Traffic Prio Level - Welchen Prioritätslevel sein Traffic besitzt*

Die Keep-Alive-Sequenz wird als Schleife mit dem Logout des Clients als Abbruchbedingung durchlaufen und verbraucht aufgrund der zahlreichen DB-updates und Berechnungen die meiste Rechenzeit neben der Authlog-Auswertung zu Beginn der Session.

Um keine DoS (Denial of Service)-Attacke gegen den Server starten zu können, indem man die Keep-Alive-Sequenz Client-seitig auf ein Minimum (1ms) stellt, verzögert der Serverprozess den Zyklus um 3 Sekunden, falls seit dem letzten Zyklus nicht mehr als eine Sekunde vergangen ist.

2.4.5 Backend-Ansteuerung

Die Backend-Ansteuerung ist eines der komplexesten Abschnitte von IronGate. Unter Backends sind die verschiedenen Arten der Benutzerverwaltung bestehender und zukünftiger Netze zu verstehen - es gilt hier, mehrere Aspekte nicht außer Acht zu lassen:

- Die Backends dürfen nicht manipuliert oder angepasst werden (müssen)
- Es müssen soviel wie möglich Backends unterschiedlichster Arten unterstützt werden
- Die Ansteuerung der Backends sollte so einfach und einheitlich wie möglich geschehen
- Es sollte mit Backends alleine, ohne Unterstützung der IronGate-internen Benutzerdatenbank, ein Betrieb möglich sein
- Die Abfragezeit muss so niedrig wie möglich gehalten werden
- Es sollte jede Kombination verschiedener Backends in Bezug auf Anzahl und Art möglich sein

- Es sollten sich die Rückgabewerte der Backends mit denen anderer Backends und der IronGate-internen Benutzer- und Gruppentabellen vereinen lassen

- Es soll eine Überschreibungsmöglichkeit dieser Werte durch die Selektion der Klassen durch den Benutzer geben können

- Mehrere IronGate-Router sollen auf die gleichen und auch unterschiedliche Backends zugreifen können

Ein etwas komplexerer Zusammenhang von IronGate-Routern in einem Netzwerk könnte bspw. wie in Abbildung 2.14 aussehen.

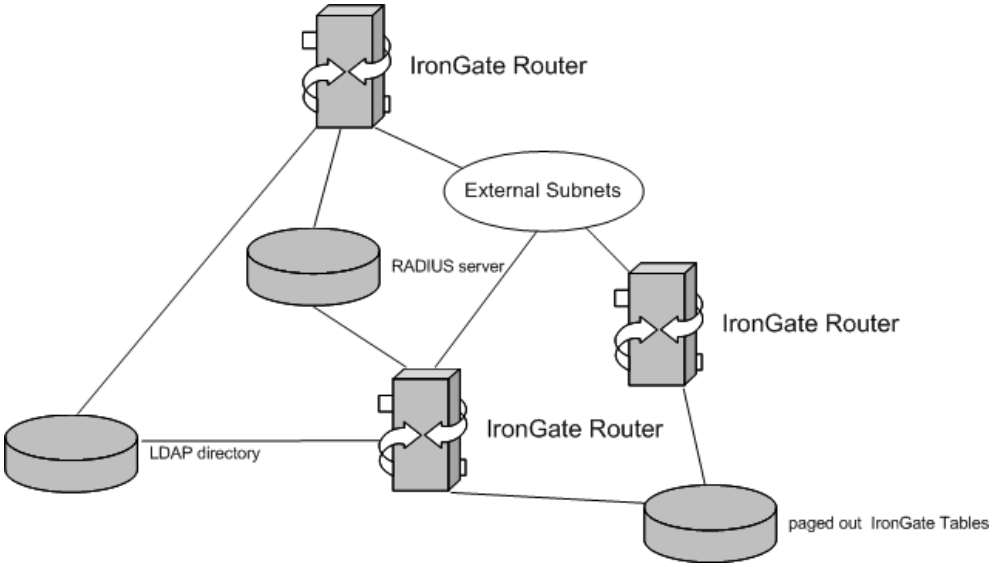

Abbildung 2.14: *IronGate Multi-Server Szenario*

Es handelt sich hierbei um ein Szenario mit einer Internetverbindung, zahlreichen Subnetzen (die an den verschiedenen Routern hängen), zwei Backends und einer IronGate-Database (ausgelagert, sie könnte aber ebensogut auf einem IronGate-Router liegen).

Um all diese Bedingungen in Form solcher Szenarien implementieren zu können, wurde ein Konzept entwickelt, welches die sequentielle Abfrage der Backends als erstes vornimmt, die dabei gesammelten Werte übernimmt (allen voran eine Gruppenzugehörigkeit), danach bei Existenz einer gleichnamigen Gruppe im IronGate-Backend deren Rechte übernimmt, bei Existenz eines Benutzers dessen Rechte übernimmt (wobei alle nachfolgend eingesammelten Rechte die vorhergehenden überschreiben), und zuletzt, sofern gewünscht, diese Rechte nochmals mit denen der vom Benutzer ausgewählten Klasse überschreibt.

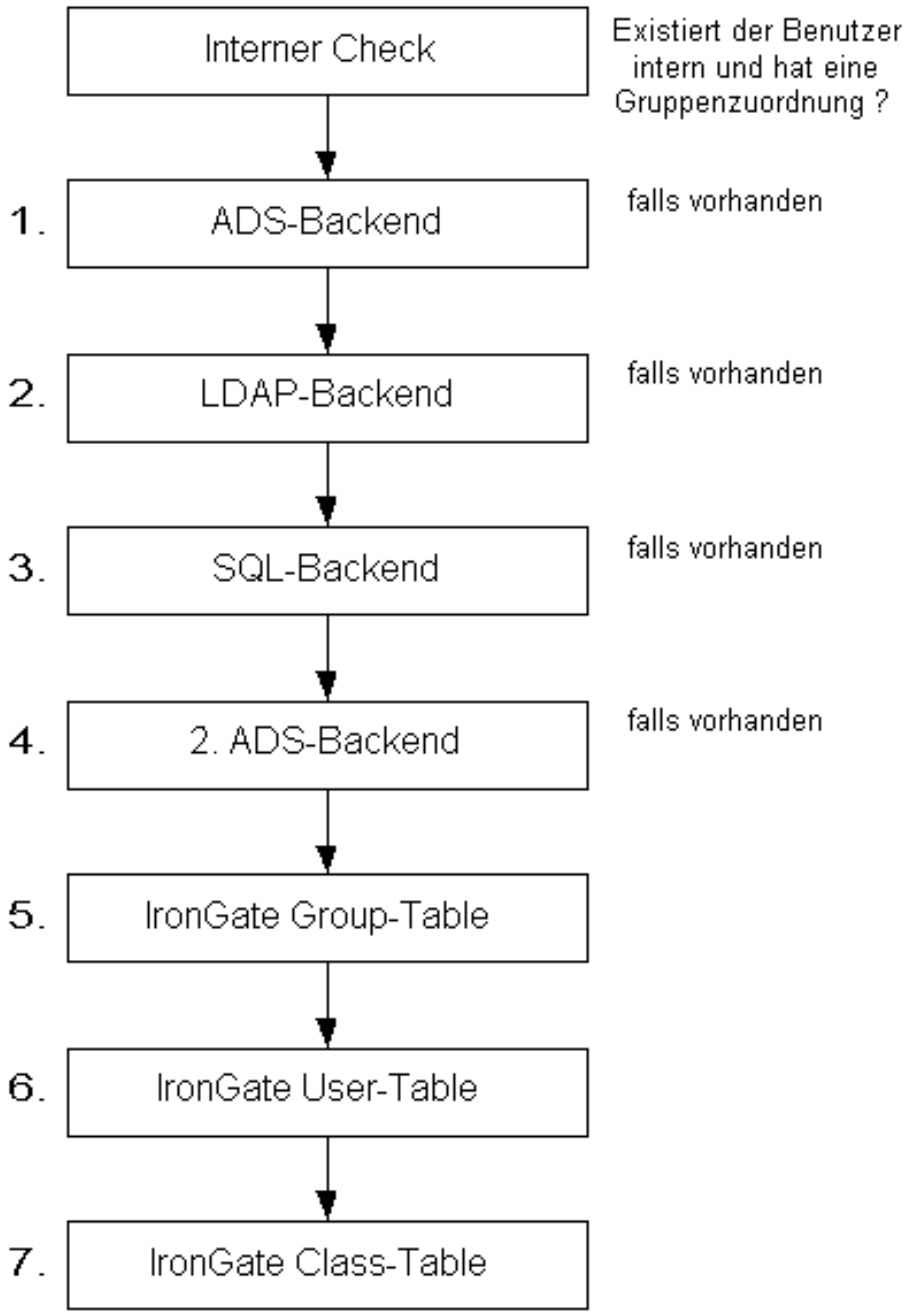

Abbildung 2.15: *IronGate Multi-Backend/-Server Szenario*

Stufenweise sei dieser Ablauf in Abbildung 2.15 beispielhaft dargestellt, wohingegen dieser sich als Flussdiagramm wie in Abbildung **??** realisieren ließe.

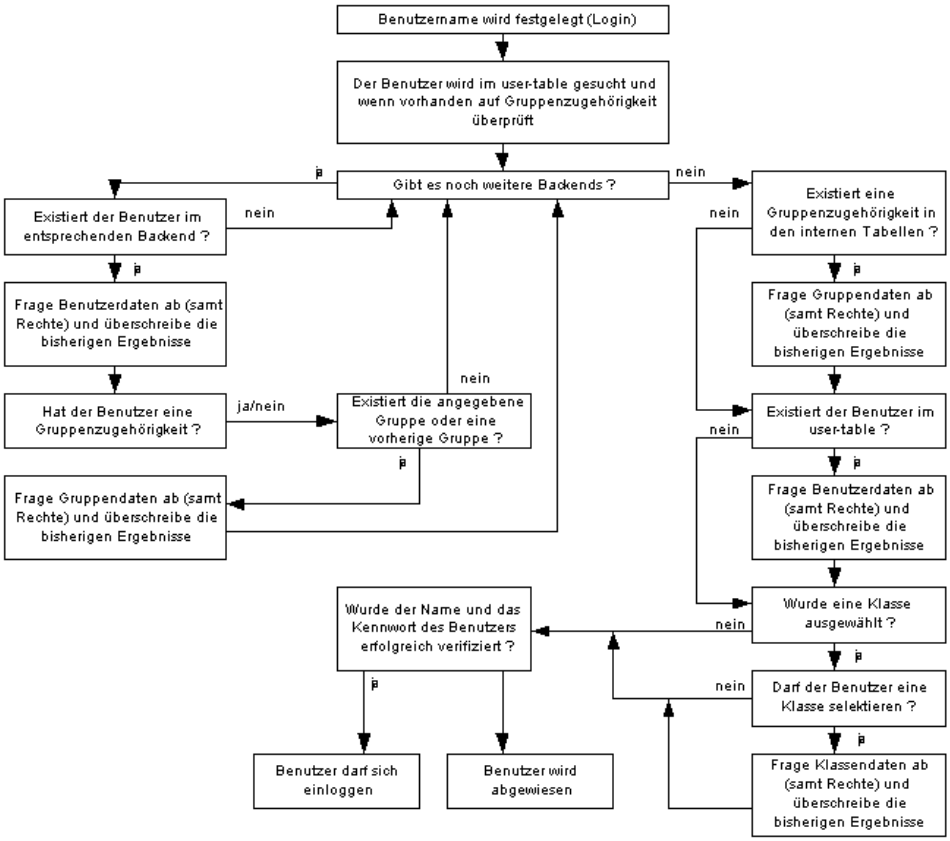

Abbildung 2.16: *IronGate Multi-Backend/-Server Szenario Flussdiagramm*

Man sieht, dass die Nachfolgeergebnisse stets die vorherigen überschreiben. Das mindeste, was ein Benutzer benötigt, um sich einloggen zu können, ist ein gültiger Benutzername und ein gültiges Kennwort. Im komplexesten Falle liegt z.B. auf einem ADS-Server nur eine Gruppenzugehörigkeit, im darauffolgend abgefragten LDAP-Server das Kennwort (welches korrekt ist), im user-table keine Gruppenzugehörigkeit, es gibt aber wohl eine Gruppe, welche mit der Gruppe des ADS-Servers übereinstimmt, weshalb deren Rechte übernommen werden, doch einen Benutzer im user-table (ohne Gruppenzugehörigkeit), dessen Rechte auch übernommen werden (mit Kennwort, jedoch falsch, doch das zuerst verifizierte Kennwort hat Vorrang - einmal verifiziert, immer verifiziert in dem Fall) und die der Gruppe zum Teil überschreiben und ergänzen, lt. Gruppenzugehörigkeit hat der Benutzer das Recht, eine Klasse zu selektieren, was dieser auch getan

hat, also werden die Werte aus dem class-table evenfalls übernommen und überschreiben z.T. die Werte, die im user-table über den Benutzer standen (in Bezug auf die Bandbreite bspw.).

So komplex wird es in einer wirklichen Umgebung kaum zugehen, IronGate kann mit einer solch heterogenen Umgebung jedoch durchaus sehr gut umgehen.

Der Ablauf der Backend-Ansteuerung wird im Kapitel über die Backends weiter oben ausführlich behandelt - beim Abfragen der Backends und den internen Tabellen werden letztendlich folgende Werte gesammelt:

- *nai* - Der Network Access Identifier des Benutzers (test@foo.com zum Beispiel)
- *groupname* - Die Gruppenzugehörigkeit des Benutzers
- *blockedports* - Die gesperrten Ports des Benutzers (für SYNs)
- *openports* - Die offenen Ports des Benutzers (für SYNs)
- *maxtimeoverall* - Die maximale Zeit, die ein Benutzer während seiner gesamten Existenz in den Datenbanken online sein darf
- *maxtimeaday* - Die maximale Zeit pro Tag, die ein Benutzer online sein darf
- *maxtimeaweek* - Die maximale Zeit pro Woche, die ein Benutzer online sein darf
- *maxtimeamonth* - Die maximale Zeit pro Monat, die ein Benutzer online sein darf
- *maxvoloverallup* - Das maximale Volumen im Upload, das ein Benutzer während seiner gesamten Existenz in den Datenbanken erzeugen darf
- *maxvoloveralldown* - Das maximale Volumen im Download, das ein Benutzer während seiner gesamten Existenz in den Datenbanken erzeugen darf
- *maxvoloverall* - Das maximale Volumen insgesamt, das ein Benutzer während seiner gesamten Existenz in den Datenbanken erzeugen darf
- *maxvoladayup* - Das maximale Volumen pro Tag im Upload, das ein Benutzer haben darf
- *maxvoladaydown* - Das maximale Volumen pro Tag im Download, das ein Benutzer haben darf
- *maxvoladay* - Das maximale Volumen pro Tag insgesamt, das ein Benutzer haben darf

- *maxvolaweekup* - Das maximale Volumen pro Woche im Upload, das ein Benutzer haben darf
- *maxvolaweekdown* - Das maximale Volumen pro Woche im Download, das ein Benutzer haben darf
- *maxvolaweek* - Das maximale Volumen pro Woche insgesamt, das ein Benutzer haben darf
- *maxvolamonthup* - Das maximale Volumen pro Monat im Upload, das ein Benutzer haben darf
- *maxvolamonthdown* - Das maximale Volumen pro Monat im Download, das ein Benutzer haben darf
- *maxvolamonth* - Das maximale Volumen pro Monat insgesamt, das ein Benutzer haben darf
- *bwup* - die maximale Bandbreite im Upload, die ein Benutzer verbrauchen darf
- *bwdown* - die maximale Bandbreite im Download, die ein Benutzer verbrauchen darf
- *status* - der Status eines Benutzers, falls er vordefiniert ist (default, bronce, silver, gold)
- *provider* - der Provider, dem der Benutzer angehört (für das Accounting relevant)
- *typeofuse* - worauf der Traffic hinoptimiert werden soll (große oder kleine Pakete, bulk oder viele Connections) - Platzhalter
- *priority* - die Priorität, die der Traffic des Benutzers in der CBQ einnimmt
- *domain* - die Domäne, in der der Benutzer eingetragen werden soll (relevant für das Dynamic DNS)
- *udp* - darf der Benutzer das UDP-Protokoll nutzen ?
- *ping* - darf der Benutzer das ICMP-Protokoll nutzen ?
- *datelogin* - Datum, ab welchem der Benutzer online sein darf (relevant für zeitweilige Accounts)
- *datelogout* - Datum, bis zu welchem der Benutzer online sein darf (relevant für zeitweilige Accounts)
- *timelogin* - Tageszeit, aber der der Benutzer online sein darf

- *timelogout* - Tageszeit, bis zu welcher der Benutzer online sein darf
- *username* - der Benutzername, welcher natürlich übereinstimmen muss
- *password* - das Kennwort, welches im Endeffekt auch eine Übereinstimmung finden muss
- *clientshowinfo* - soll der Client erweiterte Informationen über seine Online-Aktivitäten erhalten ?
- *allowchange* - ist es dem Benutzer erlaubt, seine Klasse zu ändern ?
- *disabled* - ist der Benutzer deaktiviert ?

Nachdem diese Werte aus den Backends entnommen wurden, werden die Verbindungen zu den Backends wieder abgebrochen, um Resourcen einsparen zu können - dies ermöglicht es auch, die Backends quasi „Hot-Plugged“ abzuändern, selbst während der Server noch läuft (da die Module jedesmal aufs Neue geladen werden), ohne diesen beenden zu müssen.

Details zur Weiterverarbeitung der entsprechenden Werte kommen in den weiteren Kapiteln zur Sprache.

2.4.6 Die interne Datenbank

Die interne Datenbank von IronGate dient dem Standalone-Betrieb von IronGate als Basis und bildet den Kern des Serverbetriebs. In einem Multiserver-Szenario muss und darf genau eine zentrale Datenbank existieren, auf welche die einzelnen Server paralllel zugreifen. Diese Datenbank kann entweder ausgelagert sein oder auch auf einem der IronGate-Server bzw. -router liegen. Die Datenbank dient dem Accounting, der Benutzerverwaltung, dem Verwalten der Benutzer-IDs während der Online-Phase und der Interprozesskommunikation über den Online-Table. Auch der Packetlog wird in dieser Datenbank gespeichert (kann jedoch am problemlosesten ausgelagert werden).

Die interne Datenbank läuft auf Basis einer mySQL-Engine und speichert ihre Daten relationell. Als Abfragesprache (auch für Dritthersteller-Tools) ist SQL. Dies ermöglicht es auch, auf Basis eines Hypertextprecompilers wie PHP mit einer mySQL-Datenbankanbindung auf einfache Weise ein sehr effizientes und mächtiges Web-Managementtool zu entwickeln.

Die interne Datenbank besteht aus den in Abbildung 2.17 aufgelisteten Tabellen.

- Der Authlog dient dem Accounting und der Verknüpfung zwischen dem Packetlog und den Benutzernamen. Durch ihn lässt sich während der Zeit seiner Existenz die genaue Online-Zeit, das verbrauchte Volumen, sein Backend, sein NAI usw.usf. herausfinden

	Tabelle	Aktion						Einträge	Typ	Größe
☐	authlog							90	MyISAM	19,9 KB
☐	freeids							1	MyISAM	2,0 KB
☐	groups							1	MyISAM	2,0 KB
☐	online							0	MyISAM	2,8 KB
☐	packetlog							0	MyISAM	1,0 KB
☐	status							0	MyISAM	1,0 KB
☐	users							2	MyISAM	2,2 KB
	7 Tabellen	Gesamt						94	--	31,0 KB

Abbildung 2.17: *Die IronGate-internen Tabellen*

- Die FreeIDs dienen der User-ID-Verwaltung während der Online-Phase von Benutzern - hier werden freie IDs zwischengespeichert, um in einem Art Lease-System vom nächsten Prozess wiederverwendet zu werden (dazu später noch mehr)

- Der groups-Table beinhaltet die Gruppeninformationen und -regeln

- Der Online-Table ist der Kern der IPC und zeigt immer live alle aktuellen Informationen zu den eingeloggten Benutzern an - er ist daher auch sehr dynamisch und wird permanent von den arbeitenden Prozessen abgefragt und aktualisiert. Im Falle eines Neustarts des IronGate-Systems ist diese Tabelle leer (da kein Benutzer eingeloggt ist)

- Der status-Table beinhaltet in etwa die gleichen Felder wie der groups-Table und dient als Basis für die einzelnen Klassen (default, bronce, silver, gold)

- Der users-Table beinhaltet sämtliche Informationen zu über IronGate angelegten Benutzern - sie beinhaltet einige Werte mehr als die groups- und status-tables

- Der packetlog-Table beinhaltet alle aufgezeichneten Pakete jedes Benutzers, welche einen IronGate-Router passieren und relevant sind (in erster Linie SYN-Pakete). Der Table wächst sehr schnell an, je nach Datenaufkommen, und ist daher einer sequentiellen Wartung unterworfen

In den Abbildungen 2.18 - 2.24 sind die einzelnen Tabellen im Detail aufgelistet - da alle Feldnamen selbsterklärend sind, wird auf eine nähere Beschreibung verzichtet.

Feld	Typ	Feld	Typ
userid	bigint(20)	dnsname	text
nai	text	username	text
groupname	text	domain	text
provider	text	os	text
backend	text	encrypted	text
status	text	roaming	text
blockedports	text	udp	text
openports	text	ping	text
loginserver	text	logout	text
onlinetime	bigint(20)	logintime	time
timeleft	bigint(20)	logouttime	time
volupleft	bigint(20)	logindate	date
voldownleft	bigint(20)	logoutdate	date
volleft	bigint(20)	clientshowinfo	text
volupnow	bigint(20)	typeofuse	text
voldownnow	bigint(20)	idletime	bigint(20)
volnow	bigint(20)	abnormal	text
priority	int(11)	throughputup	bigint(20)
bwup	bigint(20)	throughputdown	bigint(20)
bwdown	bigint(20)	producedload	text
pid	int(11)	syncount	bigint(20)
alive	int(11)	packetcountup	bigint(20)
mac	text	packetcountdown	bigint(20)
ip	text		

Abbildung 2.18: *Der IronGate-Online-Table*

Feld	Typ	Feld	Typ
id	bigint(20)	logouttime	time
userid	bigint(20)	onlinetime	bigint(20)
nai	text	server	text
username	text	pid	int(11)
domain	text	loginproblems	text
provider	text	mac	text
status	text	ip	text
priority	int(11)	os	text
volup	bigint(20)	encrypted	text
voldown	bigint(20)	packetcountup	bigint(20)
logindate	date	packetcountdown	bigint(20)
logoutdate	date	accounted	text
logintime	time	backend	text

Abbildung 2.19: *Der IronGate-Authlog-Table*

2.4.7 Die Benutzerklassen

Um Benutzern die Möglichkeit zu geben, bei Bedarf ihre Verbindung individuell (bis zu einem gewissen Grad) zu konfigurieren, verfügt IronGate über sog. Benutzerklassen. Dies sind Klassen, welche fixe Einstellungen für QoS (Bandbreite, Prioritäten, ToS etc.), Zeit- und Volumsbeschränkungen sowie Paketfilterregeln besitzen. Von Haus aus ist ein Benutzer der Klasse „default“ zugeordnet, welche von allen anderen Einstellungen überschrieben werden kann - hat ein Benutzer die Erlaubnis, seine Klasse individuell über den Client zu verändern, so kann er zwischen den „besseren“ Klassen Bronce, Silver und Gold wählen. Schematisch dargestellt (vom „Level“ her gesehen, bzw. dem Vorzug des Benutzers in der jeweiligen Klasse) zeigt dies Abbildung 2.25.

Eine mögliche Konfiguration der einzelnen Klassen (mitunter auch die Basiskonfiguration von IronGate) könnte folgendermaßen aussehen:

- *Klasse Default*: Kein ICMP, kein UDP, 64Kbit up / 128Kbit down, 20 MByte/Tag Limit, 100 MByte/Monat Limit, max. 2h/Tag online in der Zeit von 16h bis 22h, max. 50 Stunden pro Monat online, Traffic-Prio 0, keine Extraverrechnung zum Basisentgelt (20.- EU)

- *Klasse Bronce*: ICMP erlaubt, kein UDP, 64KBit up / 256Kbit down, 50 MByte/Tag Limit, 500 MByte/Monat Limit, max. 4h/Tagonline in der Zeit von 12h bis 24h, max. 200 Stunden pro Monat online, Traffic-Prio 1, +1.- EU / Stunde, oder +30.- EU zum Basisentgelt

Feld	Typ	Feld	Typ
id	bigint(20)	bwup	bigint(20)
nai	text	bwdown	bigint(20)
groupname	text	status	text
blockedports	text	provider	text
maxtimeoverall	bigint(20)	typeofuse	text
maxtimeaday	bigint(20)	priority	int(11)
maxtimeaweek	bigint(20)	domain	text
maxtimeamonth	bigint(20)	udp	text
maxvoloverallup	bigint(20)	ping	text
maxvoloveralldown	bigint(20)	datelogin	date
maxvoloverall	bigint(20)	datelogout	date
maxvoladayup	bigint(20)	timelogin	time
maxvoladaydown	bigint(20)	timelogout	time
maxvoladay	bigint(20)	username	text
maxvolaweekup	bigint(20)	password	text
maxvolaweekdown	bigint(20)	clientshowinfo	text
maxvolaweek	bigint(20)	allowchange	text
maxvolamonthup	bigint(20)	disabled	text
maxvolamonthdown	bigint(20)	openports	text
maxvolamonth	bigint(20)		

Abbildung 2.20: *Der IronGate-Users-Table*

Feld	Typ	Feld	Typ
id	bigint(20)	bwup	bigint(20)
blockedports	text	bwdown	bigint(20)
maxtimeoverall	bigint(20)	status	text
maxtimeaday	bigint(20)	provider	text
maxtimeaweek	bigint(20)	typeofuse	text
maxtimeamonth	bigint(20)	priority	int(11)
maxvoloverallup	bigint(20)	domain	text
maxvoloveralldown	bigint(20)	udp	text
maxvoloverall	bigint(20)	ping	text
maxvoladayup	bigint(20)	datelogin	date
maxvoladaydown	bigint(20)	datelogout	date
maxvoladay	bigint(20)	timelogin	time
maxvolaweekup	bigint(20)	timelogout	time
maxvolaweekdown	bigint(20)	clientshowinfo	text
maxvolaweek	bigint(20)	allowchange	text
maxvolamonthup	bigint(20)	disabled	text
maxvolamonthdown	bigint(20)	openports	text
maxvolamonth	bigint(20)		

Abbildung 2.21: *Der IronGate-Status(Klassen)-Table*

Feld	Typ	Feld	Typ
id	bigint(20)	bwup	bigint(20)
groupname	text	bwdown	bigint(20)
blockedports	text	status	text
maxtimeoverall	bigint(20)	provider	text
maxtimeaday	bigint(20)	typeofuse	text
maxtimeaweek	bigint(20)	priority	int(11)
maxtimeamonth	bigint(20)	domain	text
maxvoloverallup	bigint(20)	udp	text
maxvoloveralldown	bigint(20)	ping	text
maxvoloverall	bigint(20)	datelogin	date
maxvoladayup	bigint(20)	datelogout	date
maxvoladaydown	bigint(20)	timelogin	time
maxvoladay	bigint(20)	timelogout	time
maxvolaweekup	bigint(20)	password	text
maxvolaweekdown	bigint(20)	clientshowinfo	text
maxvolaweek	bigint(20)	allowchange	text
maxvolamonthup	bigint(20)	disabled	text
maxvolamonthdown	bigint(20)	openports	text
maxvolamonth	bigint(20)		

Abbildung 2.22: *Der IronGate-Groups-Table*

Feld	Typ
id	bigint(20)
userid	bigint(20)
protocol	text
destport	bigint(20)
destip	text
time	time
date	date
servername	text

Abbildung 2.23: *Der IronGate-Packetlog-Table*

Feld	Typ
id	bigint(20)
userid	bigint(20)

Abbildung 2.24: *Der IronGate-FreeIDs-Table*

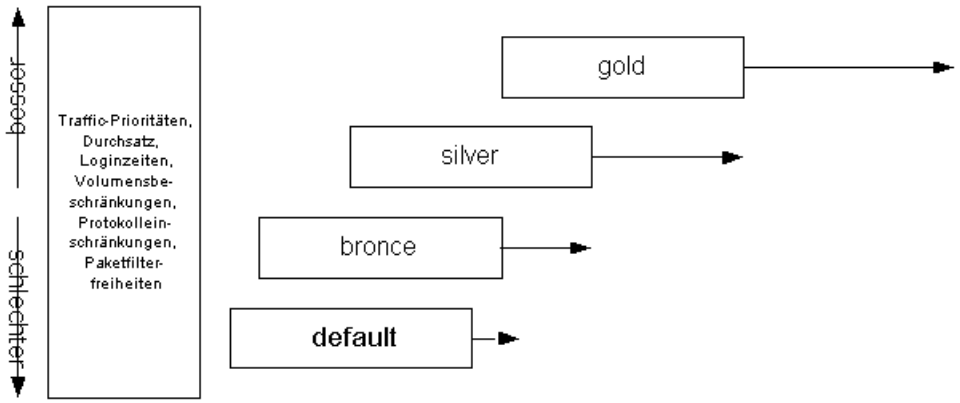

Abbildung 2.25: *Die IronGate-Klassen*

- *Klasse Silver*: ICMP und UDP erlaubt, 128Kbit up / 512Kbit down, 300 MByte/Tag Limit, 5 GByte/Monat Limit, keine Zeitbeschränkung, Traffic-Prio 2, +2.- / Stunde oder +50.- EU zum Basisentgelt

- *Klasse Gold*: ICMP und UDP erlaubt, 512Kbit up / 1Mbit down, kein Tagesvolumslimit, 50 GByte/Monat Limit, keine Zeitbeschränkung, Traffic-Prio 5, + 5.- / Stunde oder +150.- EU zum Basisentgelt

Es ist einfach zu erklären, was durch diese Klassifizierung erreicht werden kann: Ein Benutzer nimmt sich sich einen Default-User, der, seiner Ansicht nach, für seine Bedürfnisse ausreicht. Tatsächlich trifft er auf Anwendungen, die mindestens einen Bronce-User, eher noch einen Silver-User verlangen (Spiele z.B. oder größere Downloads). U.u. übersteigt er sogar die 5GByte/-Monat Grenze schnell und benötigt dadurch den Gold-User, welcher ihm ein Mehrvolumen gegen ein höheres Entgelt problemlos ermöglicht. Vielleicht ist der Benutzer aber auch auf ein Internettelefonat angewiesen, welches nach UDP verlangt und noch einer hohen Traffic-Priorität mit hohem Durchsatz. Auf diese Art und Weise kann der Benutzer die Art und Weise seiner Bezahlung angemssen auswählen und bei Bedarf, gegen höheres Entgelt, auch zeitweise in den Genuss einer besseren Anbindung kommen, obwohl er eigentlich nur für eine geringere bezahlt hat.

Entgegengesetzt kann man die Volums- und Zeitverfügbarkeiten der

höheren Klassen herabsetzen und die Default-Klasse zwar langsam und zurückgestuft, jedoch unlimitiert freistellen. Dadurch kann auch Benutzern in Non-Profit-Umgebungen die Möglichkeit gewährt werden, bspw. für Spiele auf eine sehr schnelle Leitung zurückgreifen zu können, dies jedoch nur im engen Rahmen zu ermöglichen, um den Durchschnittstraffic (Bulk, Leeching etc.) nicht die Qualität der Leitung einbüßen zu lassen.

Die Benutzerklassen wurden erst ab dem Java-Client implementiert und werden vom Backend-Abfrage-Algorithmus zuletzt ausgewertet.

2.4.8 User-ID Zuweisung

Ein Benutzer muss während seiner Login-Phase eine eindeutige Benutzer-ID haben. Diese zieht sich durch den ganzen Verlauf der Session und wird für folgende Funktionen benötigt:

- Die Erstellung der Regelketten einzelner Benutzer (s.u.) im Stile von IRONGATE-USERID
- Das Markieren der Pakete zur Weiterverwerwendung im relationalen Logging und dem CBQ
- Die Auswertung der geloggten Pakete durch den relationalen Packetlogger
- Das Shaping und Priorisieren der Pakete / des Traffics im CBQ
- Das Identifizieren des Benutzers über den Authlog bei Paketen im Packetlog
- Das Verhindern von zwei gleichnamigen Ketten für zwei verschiedene Benutzer

Für das Zuweisen der Unique-ID während einer Session wurde ein einfacher Algorithmus entwickelt, welcher sich auf einen internen SQL-Table (FreeIDs) stützt, und dessen Funktion in Abbildung 2.26 dargestellt ist.

Dadurch wird erreicht, dass niemals eine höhere ID in einem IronGate-System vorhanden ist, als Benutzer eingeloggt sind. Würde sich die ID stetig erhöhen, würde bei Überschreitung der Grenzen wie z.B. den maximalen iptables-Marker oder die höchste CBQ-Class-ID das System nicht mehr funktionieren. Auch wird somit garantiert, dass beim Einsatz mehrerer IronGate-Router im Verbund niemals zwei User, welche gleichzeitig online sind, die gleiche ID zugewiesen bekommen.

2.4.9 Packet Filtering

IronGate verwendet zur Filter-Steuerung grundsätzlich das NetFilter-iptables-System des Linux-Kernels. NetFilter bzw. die iptables bauen dazu

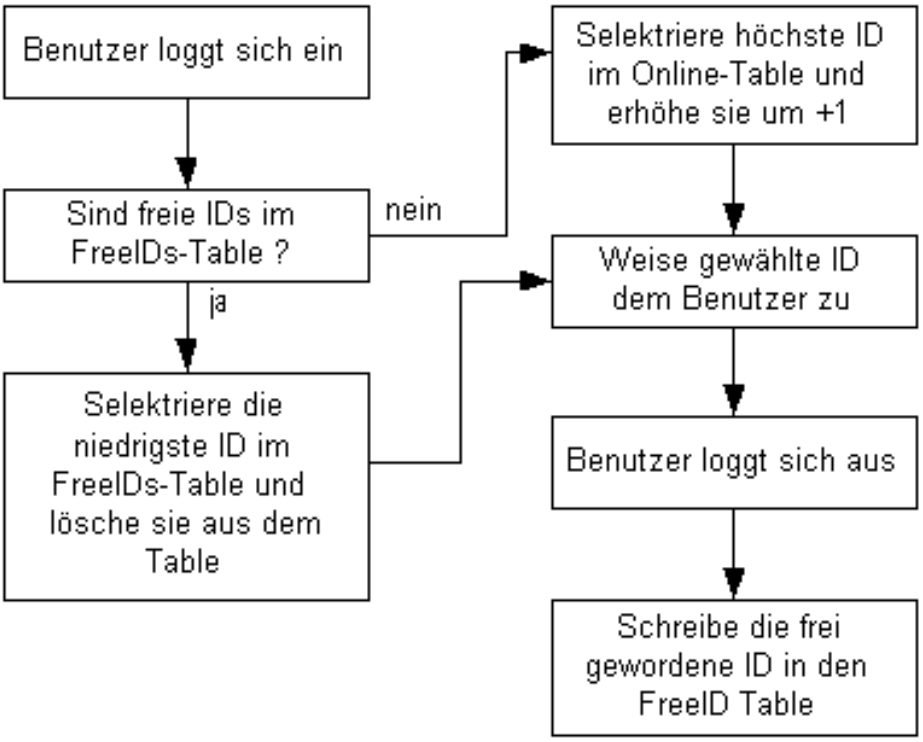

Abbildung 2.26: *ID Allocation Sequence*

auf drei Regeltabellen auf, welche wiederum sog. Chains, also Regelketten beinhalten können. Dieses Regelwerk und vor allem die Regeln und dessen Abarbeitung werden in Abbildung 2.27 nähergebracht.

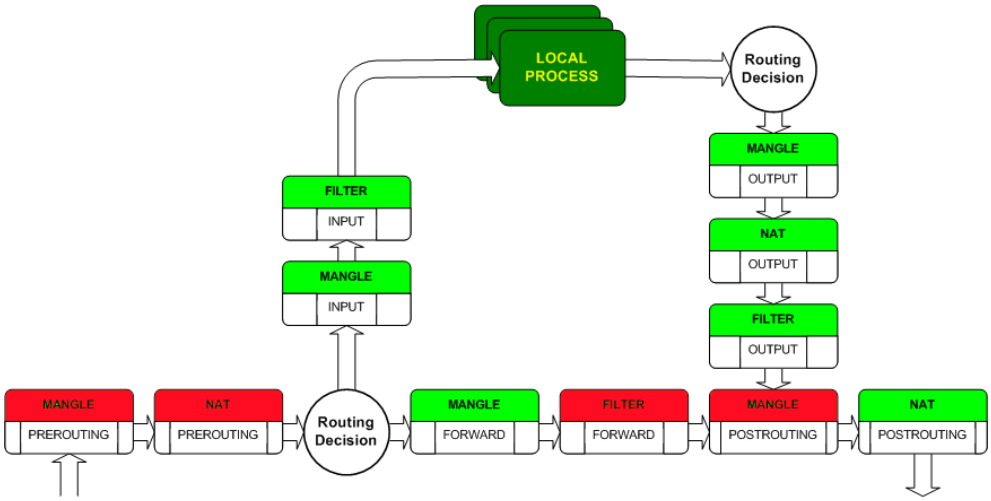

Abbildung 2.27: *IP-Tables Packet Traversal*

IronGate legt in jedem Table eine eigene Chain IRONGATE an, in welche die Pakete prinzipiell umgeleitet werden (dennoch wird im folgenden auf die grundsätzlichen Chains eingegangen). Die von IronGate verwendeten Tables pzw. Chains sind rot (dunkelgrau) unterlegt und sind im Detail folgende:

- *-table NAT -chain PREROUTING*: Diese Chain wird verwendet, um die DNAT-Regeln (Port-Redirecting) einzurichten
- *-table MANGLE -chain PREROUTING*: Diese Chain wird verwendet, um die ausgehenden Pakete mit der MARK-Regel zu markieren
- *-table MANGLE -chain POSTROUTING*: Diese Chain wird verwendet, um die eingehenden Pakete mit der MARK-Regel zu markieren
- *t-table FILTER -chain FORWARD*: Diese Chain wird verwendet, um die Paket-Filter-Regeln zu definieren (ACCEPT und DENY) und die Pakete/Bytes zu zählen

Bei der Initialisierung von IronGate passiert folgendes:

1. Es werden IRONGATE-Chains in allen drei Tables angelegt und der gesamte Traffic von und zum Subnet in diese umgeleitet
2. Alle Pakete von und zum Subnet, in welchem IronGate operiert, werden gedropped

3. Die DNAT-Regeln werden gesetzt (siehe unten)
4. Im NAT-Table wird im PREROUTING eine Regel gesetzt (-j LOG), welche sämtliche Paketheader von SYN-Paketen an den Syslogweitergibt

Wenn sich ein Client erfolgreich einloggt, werden folgende Regeln gesetzt:

1. Es werden in jeder IronGate-Chain eine neue Chain in Form von „IRONGATE-USERID“ angelegt und der Traffic die IP und MAC des Clients betreffend in diese umgeleitet
2. In jeder Chain, welche im Table NAT oder Table FILTER liegt, werden alle Pakete, welche die Userkette durchlaufen, auf ACCEPT gesetzt (damit diese geroutet werden und nicht mehr gefiltert)
3. Es werden im Table FILTER in der Userkette (IRONGATE-USERID) die Zählfilter für TCP SYN, UDP und den Gesamttraffic up/down angelegt
4. Je nach Benutzereinstellung werden in einer Schleife die Paketfilter (DROP und daraufhin ACCEPT) als Regeln der Reihe nach eingetragen (in den SQL-Tabellen als „blockedports“ und „openports“ angeführt)
5. Im Table MANGLE werden in der Userkette die Pakete mit der Unique-ID des Benutzers markiert (später wichtig für das relationelle Logging und die CBQ-Regeln), jeweils die ausgehenden und eingehenden

Während der Keep-Alive Sequenz werden die Packet- und Bytecounter in der Userkette von FILTER zyklisch abgefragt.

Damit hat der Benutzer nun alle ihm zugewiesenen Rechte, je nach Konfiguration, und kann absolut transparent und „frei“ mit seiner Verbindung arbeiten, als wäre weder eine Firewall noch sonst etwas zwischen ihm und seinem Ziel.

Nach Beendigung der Session (Log-Out Sequenz wird initialisiert) geschieht folgendes:

1. Die Benutzerketten werden geleert (-FLUSH)
2. Die Verweise aus den jeweiligen IronGate-Chains auf die jeweiligen Benutzerketten werden gelöscht (-DELETE)
3. Die Benutzerketten werden gelöscht (-X)

Damit bleiben nach dem LogOut eines Benutzers keinerlei Rückstände in den Filtertabellen, und es bilden sich nicht nach zahlreichen Loginversuchen etwaige Artefakte, welche das System unnötig belasten (oder dem Benutzer gar ohne eingeloggt zu sein zu Rechten verhelfen).

2.4.10 Port Redirecting

IronGate initialisiert beim Start über das NetFilter-Regelwerk von Linux sog. DNAT-Regeln (Destination Network Address Translation), auch genannt Port-Redirecting. Da IronGate einmal grundsätzlich sämtliche Ports im PREROUTING und im FORWARD sperrt, also alle Pakete dropped, hat der Benutzer auch keine Möglichkeit, zum DNS, zum ICMP-Stack (Ping), zu IronGate selbst oder zum HTTP-Server, welcher das Auto-Config-Script lagert und die Client-Installation anleitet, zu verbinden.

Daher leitet IronGate Pakete an die entsprechenden Ports vor den DROP-Filtern der IRONGATE-Chains im Table NAT und FILTER an diese um. Dieses Verfahren nennt sich DNAT, und lässt sich wie in Abbildung 2.28 darstellen.

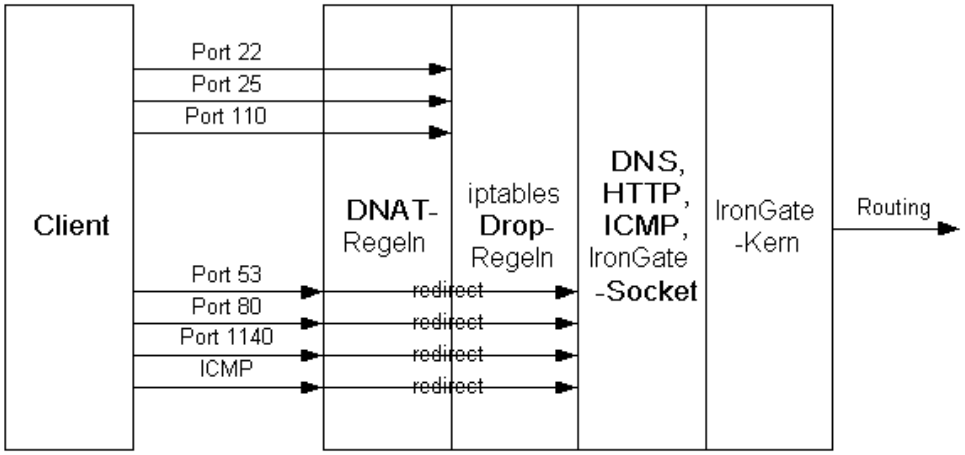

Abbildung 2.28: *DNAT-Vorgang*

Wenn ein Benutzer sich einloggt, werden seine Regeln vor den DNAT-Regeln abgearbeitet (sobald ein ACCEPT in einer Regel auftritt, wird das Paket in der CHAIN-Abfolge in die nächste Regel weitergeleitet, bzw. in den nächsten Table), um eine dauernde Umleitung zu unterbinden (nur wenn der Benutzer nicht eingeloggt ist, macht sie auch Sinn).

2.4.11 Relationales Packet-Logging

Das relationale Packet-Logging stützt sich auf die Paket-Markierungen mit der Unique-ID eines Benutzers, welche durch den MANGLE-Table vorgenommen werden. Die Pakete nehmen dabei den in Abbildung 2.29 dargestellten Weg.

Dadurch hat jedes Paket im Packetlog automatisch bereits die ID als Zuweisung (ohne weitere Datenbankabfragen), welche der Benutzer zugewiesen

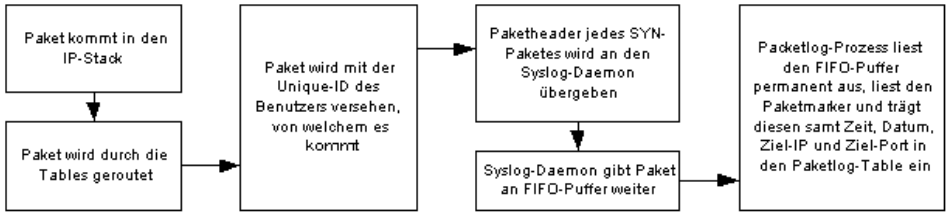

Abbildung 2.29: *Ablauf relationelles Logging*

hatte, während sein Paket den IronGate-Server erreichte. In Kombination mit dem Authlog kann so jeder Benutzer anhand der ID, welche sein Paket trug, auch nach Monaten noch identifiziert werden.

Zur Verwirklichung dieses Verfahrungs musste ein Linux-Kernelpatch geschrieben werden, welcher den Paketmarker mit an den Syslog-Daemon übergibt, wenn dieser die Paketheader über iptables gepiped bekommt. Der Patch wird im Kapitel der Implementierung genau beschrieben.

Der Packetlogger ist ein eigener, unabhängiger Prozess, welcher lediglich das IronGate-Konfigurationsskript zur eigenen Konfiguration verwendet. Der Prozess wird mit IronGate mitgestartet, jedoch nicht geforked.

2.4.12 Traffic Shaping und Priorizing

Das Traffic-Shaping wird in IronGate über den CBQ-Algorithmus abgewickelt. Bei der Initialisierung von IronGate werden zwei Klassen (up and down) mit den IDs „1:“ jeweils auf dem internen und auf dem externen Interface angelegt.

Wenn sich ein Benutzer einloggt, wird eine Klasse mit der ID „1:USERID“ am jeweiligen Interface angelegt, welche der ihm zugewiesenen Up- bzw. Downloadbandbreite (maximal) entspricht. Außerdem wird die Priorität der Klassen mit der in den Regeln des Benutzers eingestellten konfiguriert. Daraufhin wird ein CBQ-Filter auf diese Klasse gelegt, welcher nach der Paketmarkierung, die vorher über den MANGLE-Table angelegt wurde, die Pakete der für den Benutzer bestimmten Klasse in diese „Queued“.

Somit durchlaufen alle Pakete eines Benutzers in die jeweilige Richtung die entsprechenden CBQ-Klassen und werden durch diese wir vorgegeben priorisiert und geshaped. Während der Logoff-Sequenz eines Benutzers werden seine CBQ-Klassen und Filter wieder gelöscht, um keinerlei Artefakte zu hinterlassen.

2.4.13 Dynamic DNS Updates

Das Verfahren, wie es weiter oben schon erläutert wurde, wird in IronGate dazu eingesetzt, Benutzern temporär A NAME und IN PTR-Records zur

Verfügung zu stellen. Hierzu muss der DNS-Server, mit welchem IronGate kommmuniziert (lokal oder auf einem anderen Server) das dynamische Update für die Adresse des IronGate-Server erlauben - dazu bedarf es einer eigenen Konfiguration. Getestet wurde IronGate lediglich mit dem sehr weit verbreiteten BIND-DNS-Server. Der Nachteil des dynamischen DNS ist, dassder Server hierbei selbst seine Zonendateien umschreibt, was eine übersichtlich gegliederte Zonendatei zwar inhaltlich nicht verändert, die Gliederung jedoch „maschiniert“. Daher ist es zu empfehlen, sofern die Zonendateien lokal liegen, diese beim Start von IronGate gescriptet mit statischen Original-Dateien zu überschreiben.

IronGate setzt den DNS-Namen aus dem Benutzernamen, mit welchem sich der Benutzer authentifiziert und der vorkonfigurierten Domäne bzw. der individuellen Domäne für den entsprechenden Benutzer (siehe Table users, groups und Backends) zusammen (getrennnt durch einen Punkt). Auch kann ein Benutzername dabei einen „.“ enthalten, bspw. folgendermaßen:

1. „test.benutzer“ loggt sich von IP 10.5.0.2 ein
2. Die Domäne ist von IronGate her auf „.foo.com“ konfiguriert
3. Der Prozess assembliert den A NAME Eintrag test.benutzer.foo.com sowie den IN PTR-Eintrag 2.0.5.10.in-addr.arpa
4. Der Prozess setzt den dynamischen Eintrag mit einer TTL (Time-to-live) Zeit von 30 Sekunden unter Verwendung von nsupdate

Schematisch würde ein dynamisches Update mit einem ausgelagerten DNS-Server und einer folgenden A NAME -¿ IP Anfrage wie in Abbildung 2.30 aussehen.

Wichtig in diesem Zusammenhang ist es, dass ein Benutzername, welcher für einen Dynamic-DNS Eintrag verwendet werden soll, keine Zeichen enthält, welche im DNS nicht erlaubt sind (Umlaute, Underscores, Doppelpunkte etc.).

2.4.14 MAC-bound Forwarding

Herkömmliche Firewalls kennen das Filtern nach MAC-Adresse und das Filtern nach der IP-Adresse. Beide Mechanismen jedoch schützen vor einer IP-Spoofing Attacke bzw. einer MAC-Spoof-Attacke nicht. IronGate verwendet daher ein Feature der iptables, welches es ermöglicht, eine Regel nur im Zusammenhang zwischen einer IP-Adresse und einer MAC-Adresse anzulegen. So wird das Routing im FILTER-Table nur in Kombination dieser beiden Werte erlaubt.

Zuerst wird die MAC-Adresse über das externe Tool ARP importiert (indem ARP nach der MAC zur vom Client verwendeten IP-Adresse gefragt

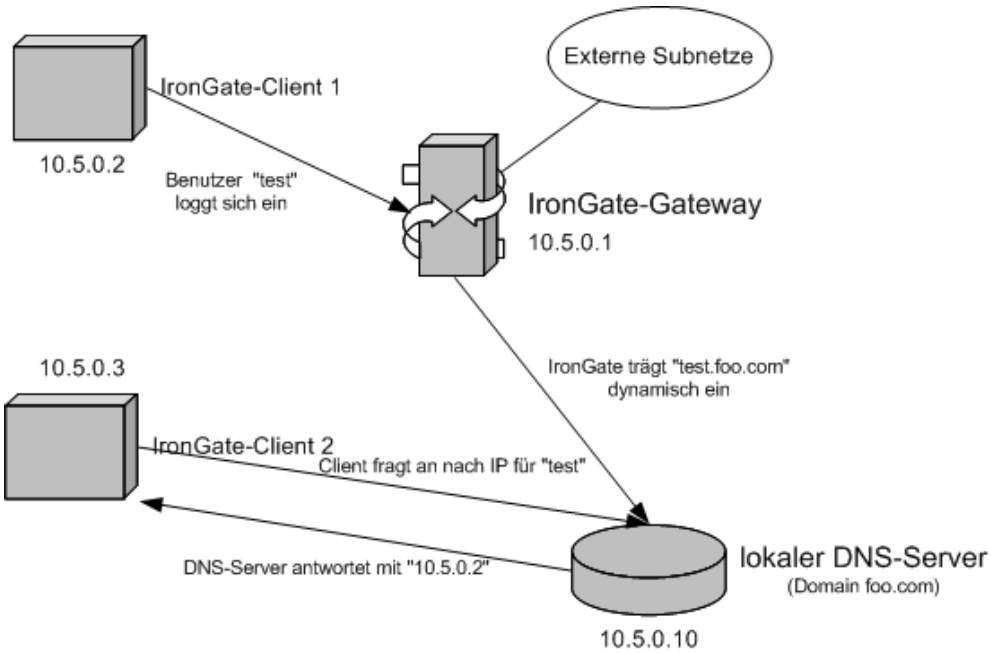

Abbildung 2.30: *Dynamisches DNS*

wird). Daraufhin wird die entsprechende Regel angelegt, welche folgende Angriffe verhindert:

- Reine IP-Spoofing Attacken können nicht durchgeführt werden, da Pakete von einer gefälschten IP-Adresse nicht von der in den Regeln erwarteten MAC-Adresse kommen würden
- Reine MAC-Spoofing Attacken können ebenso nicht durchgeführt werden, da da Pakete von einer gefälschten MAC-Adresse nicht von der in den Regeln erwarteten IP-Adresse kommen würden
- Eine Fälschung beider Adressen scheitert an der Konfiguration der Switches (wie eine reine MAC-Fälschung) ebenso wie an der geringen Zeit (maximal zehn Sekunden), die dem Angreifer bleibt, um die Verbindung zu übernehmen, wobei das gespoofte System die doppelte MAC-Adresse sofort bemerken würde, daraufhin den Stack deaktiviert, was einen Verbindungsabbruch zu Folge hat
- Ein MAC-Hijacking in Form von ARP-Flooding mit Übernahme der IP-Adresse scheitert ebenfalls an der zu geringen verfügbaren Zeit

2.4.15 heuristische Usage-Detection

Um über die Tätigkeiten der Benutzer ein ungefähres Wissen zu haben, besitzt IronGate zwei Heuristiken, welche sich an den Paket- und Bytezählern

der iptables-Ketten orientieren. Folgende Werte werden hierfür ausgewertet:

- Packetcount SYN - Die Anzahl der SYN-Pakete des Benutzers, welche innerhalb des letzten Keep-Alive-Zyklus den Router passierten
- Packetcount UDP - Die Anzahl der UDP-Pakete des Benutzers, welche innerhalb des letzten Keep-Alive-Zyklus den Router passierten
- Throughput insgesamt - Die Anzahl an Bytes an Up- und Download-Traffic innerhalb des letzten Keep-Alive-Zyklus

Anhand eines auf Erfahrungswerten basierenden Algorithmus werden diese Werte gegenübergestellt und aus ihnen jeweils eine Art der Leitungsverwendung und ein Load errechnet. Die Abbildungen 2.30 und 2.32 zeigen die möglichen Verwendungen und Loads im Zusammenhang mit den gesammelten Werten.

Packetcount SYN	**Packetcount UDP**	**Throughput insg.**	**Type of Use**
<20		<30.000	„surfing“
>20		>30.000	„sharing“
>100		<10.000	„scanning“
<20		>20.000	„leeching“
	>20	<20.000	„gaming“
	>20	>20.000	„streaming“
<10		<2.000	„idle“
		else	Undefinierbar

Abbildung 2.31: *Zusammenhang zwischen UDP, SYN und Traffic*

Throughput insg.	**Load**
<10.000	„low“
<30.000	„medium“
<60.000	„high“
else	„extreme“

Abbildung 2.32: *Zusammenhang zwischen Traffic und Load*

Die Heuristik deckt weder jeden Anwendungsfall ab noch kann sie besonders genau sein, gibt jedoch gerade bei relevanten Aktionen (Scannen, Spielen, Leechen) aktuellen Aufschluss über die Aktivitäten der Benutzer. Eine Kopplung der Heuristik mit einer Information des Systemadministrators (bspw. beim Scannen) wäre denkbar, jedoch sollte für eine wirkliche Intrusion Detection bzw. Abuse Detection ein speziell darauf ausgerichtetes Tool verwendet werden.

2.4.16 Accounting

IronGate speichert sämtliche Informationen über eine Session (alle gesammelten Informationen vom Zeitpunkt des Logins bis zum Zeitpunkt der Verbindungstrennung) im Table AuthLog zwischen (vgl. die internen Datenbanken).

Beim Accounting wird für jeden Benutzer (DISTINCT username) eine Zusammenfassung dieser authlog-Daten erstellt (aufsummiert) bzw. mit einem fixierten Preis direkt abgerechnet - daraufhin können Rechnungen für die einzelnen Benutzer entweder per Mail versandt oder auch ausgedruckt werden.

Das momentane Accounting per Web-Interface sieht lediglich das Aufsummieren der Authlog-Einträge in einem gewissen Zeitabstand vor (definierbar), sowie das Ausdrucken der Aufsummierung. Aufgrund der offenen Datenbankschnittstelle kann jedoch quasi jede Accounting-Software vorgeschaltet werden oder auch auf einfache Art und Weise in bpw. ein LaTeX-Dokument geparsed werden, um dann als Serienbrief ausgedruckt zu werden.

2.4.17 Prozess-Steuerung und -Recovery

Wenn eine Systemüberlastung oder eine verlorene Datenbankverbindung auftritt, können zwei Fälle eintreten:

- Der Benutzer kann nach wie vor mit dem Internet arbeiten, wird jedoch in dieser Session nicht mehr accounted

- Der Benutzer verliert die Verbindung zum Server, kann sich auch nicht mehr einloggen

Beide Fälle sind entweder für den Benutzer oder für die Administration untragbar. Daher verfügt IronGate über einen Prozess-Watchdog (eigener, unabhängiger Prozess), welcher in einem Zyklus von 15 Sekunden den Online-Table überprüft, und einen Prozess im Falle des Falles zu beenden und die von ihm eingetragenen Online-Informationen auf den Auth-Table zu übertragen (siehe Abbildung 2.33).

Das Verfahren basiert darauf, dass jeder Kindprozess im Laufe eines Zyklus die Variable „active“ auf 1 setzt (welche im Online-Table gespeichert wird). Der Watchdog überprüft ca. alle 2-3 Zyklen (je nach Keep-Alive Zeit des Client), ob diese Variable für jeden Prozess auf 1 ist und setzt sie daraufhin auf 0. Kommt ein Prozess nicht mehr dazu, die Variable innerhalb von 2-3 Zyklen der anderen Prozesse wieder auf 1 zu setzen, so leitet der Watchdog-Prozess ein Auslog-Verfahren ein, bei welchem alle relevanten Informationen vom Online-Table in den Authlog-Table übertragen werden und der betreffende Prozess mittels TERM bzw. KILL aus dem System entfernt

wird, da davon ausgegangen werden muss, dass der Prozess aus einem gewissen Grund hängt. Im Authlog wird dies als „process terminated“ verzeichnet (als Auslog-Message).

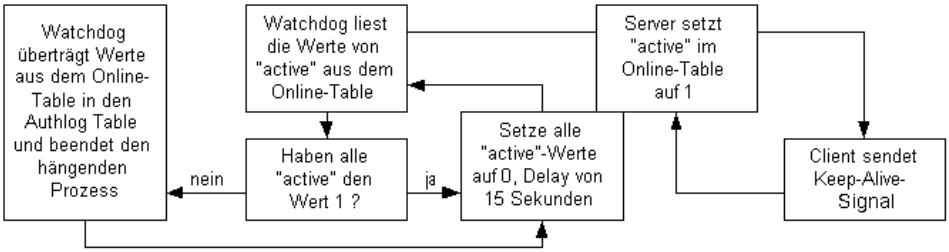

Abbildung 2.33: *Schema des Watchdog-Prozesses*

2.4.18 Der Logoff-Vorgang

Während der Logoff-Sequenz werden, wie bereits in den entsprechenden Kapiteln erwähnt, sämtliche Variablen des Online-Tables in den Authlog-Table zur späteren Auswertung in Form des Accountings oder Usertrackings übertragen. Darüberhinaus werden sämtliche Regeln und Filter gelöscht bzw. deaktiviert, das System quasi in den Zustand vor dem Login zurückversetzt.

Darüber hinaus wird der Grund des Logoffs angegeben - im entsprechenden Feld wird auch der Grund eines nicht geglückten Logins verzeichnet. Die Gründe sind im Detail folgende:

- *ip already used* - Die IP, von der aus sich jemand einloggen möchte, ist bereits von einem anderen Benutzer in Verwendung
- *no username given* - Der Benutzer hat keinen Benutzernamen angegeben
- *user not existing* - Der Benutzer existiert weder in den internen Tabellen noch auf einem Backend (ungültiger Benutzername)
- *licences exceeded* - Es ist die maximale Anzahl der lizenzierten Benutzer bereits online
- *user disabled* - Der Benutzeraccount wurde deaktiviert
- *username already in use* - Der Benutzername ist bereits von einem anderem Prozess in Verwendung
- *mac already loggedin* - Es ist bereits eine Maschine mit der verwendeten MAC-Adresse eingeloggt

- *logintime exception* - Der Benutzer versucht, sich zu früh oder zu spät einzuloggen
- *logindate exception* - Der Benutzer versucht, sich zu früh oder zu spät (in Tagen) einzuloggen
- *voldown exceeded* - Das maximale Downloadvolumen wurde erreicht
- *volup exceeded* - Das maximale Uploadvolumen wurde erreicht
- *vol exceeded* - Das maximale Gesamtübertragungsvolumen wurde erreicht
- *time exceeded* - Die maximale Online-Zeit wurde überschritten
- *salterror* - Es trat ein Fehler während der Verschlüsselung auf (der Salt wurde nicht korrekt zurückgegeben)
- *wrong userpass* - Benutzername/Kennwort Kombination ist inkorrekt (falsches Kennwort)
- *logintime overrun* - Die maximale Online-Zeit wurde während der Session überschritten
- *volup overrun* - Das maximale Upload-Transfervolumen wurde während der Session überschritten
- *voldown overrun* - Das maximale Download-Transfervolumen wurde während der Session überschritten
- *vol overrun* - Das maximale insgesamte Transfervolumen wurde während der Session überschritten
- *serverside logout* - Der Benutzer wurde serverseitig ausgeloggt (durch den Administrator)
- *process terminated* - Der handelnde Prozess wurde vom Watchdog terminiert (s.o.)

Parallel zum Authlog-Table Eintrag wird der Benutzer über das Client-Fenster aus über den Grund des Logoffs informiert

2.4.19 Login-Restrictions

Wie schon beschrieben, kann der Administrator zahlreiche Restriktionen vorgeben, welche auf einen Benutzer angewandt werden können (entweder in Form von default-Einstellungen, Benutzerkonfiguration, Klassenkonfiguration oder Gruppenkonfiguration bzw. durch Einträge im Backend). Im

Detail werden diese Restriktionen im Kapitel „Backend-Ansteuerung“ abgehandelt.

Um nun überprüfen zu können, ob ein Benutzer ein Zeit- oder Volumslimit überschritten hat, müssen sämtliche Einträge einen Benutzer betreffend im Authlog-Table aufsummiert werden und mit den Maximal-Werten verglichen werden. Dabei ist auch das Datum der Einträge relevant, da es Unterscheidungen zwischen Tagen, Wochen, Monaten und insgesamt bei den Limitierungen gibt. Wie die Berechnung der Restzeiten und -volumen im Detail abläuft, wird im Kapitel der Implementierung besprochen.

2.4.20 Lizenzverwaltung

IronGate verfügt über eine sehr einfache Lizenzverwaltung, welche lediglich die maximale Anzahl der gleichzeitig sich online befindlichen Benutzer definiert (Variable „licences“). Wird diese Variable überschritten, so wird der Benutzer, welcher sich einloggt, zurückgewiesen. Auf diese Art und Weise bzw. mit einer verschlüsselten Lizenzierung kann IronGate auf Lizenzbasis vertrieben werden.

2.4.21 Automatische Client-Konfiguration

Um einen möglichst hohen Benutzerkomfort zu erzielen, verfügt der IronGate-Client über eine Prozedur, welche auf dem lokalen IronGate-Router nach einem automatischen Konfigurationsskript sucht. Dies dient in erster Linie dem Handover, welches ansonsten nicht funktionieren könnte (da der Benutzer nach jedem Subnet-Wechsel den Client neu konfigurieren müsste) und in zweiter Linie auch der Wartbarkeit lokal installierter IronGate-Clients, welche auf diese Art und Weise nicht (um-)konfiguriert werden müssen.

Das Skript bzw. die Konfigurationsdatei muss hierzu unter dem Dateinamen „irongate-client.conf“ auf dem root-directory des Webservers gespeichert sein und folgende Variablen enthalten (jeweils in einer eigenen Zeile):

- IP des Servers (x.x.x.x)
- Port des Server (n)
- Delays zwischen den Keep-Alives (n)

Findet der Client dieses File beim Zugriff auf „www.google.com“ (was durch den HTTP-Redirect durch DNAT auf den lokalen Webserver umgeleitet wird), konfiguriert er sich selbst nach dieser Datei. Wird die Datei nicht gefunden, wird auf das eigene Konfigurationsfile bzw. in dritter Instanz auf die manuelle nicht statische Konfiguration zurückgegriffen.

2.4.22 Client-Subnet-Handover

Wenn ein WLAN-Client von einem Subnet ins nächste (Layer-3 Wechsel) wandert, so bricht die Verbindung zum IronGate-Server ab. Ohne eine Handover-Funktion müsste der Benutzer den Client manuell neu einloggen lassen, was Zeit und Komfort kostet.

Daher hat IronGate für diesen Fall eine Handover-Routine implementiert (client-seitig), welche eine Verbindungsunterbrechung erkennt, bei Neuvergabe der IP den Client automatisch mit dem serverseitig liegenden Script an den neuen Router anpasst und die Verbindung wieder aufbaut (siehe Abbildung 2.34). Die Zeit hierzu beträgt ca. 8 Sekunden (siehe Tests). Dies stellt zwar kein reguläres Handover dar, ermöglicht aber ein rasches Subnetübergreifendes Arbeiten mit IronGate.

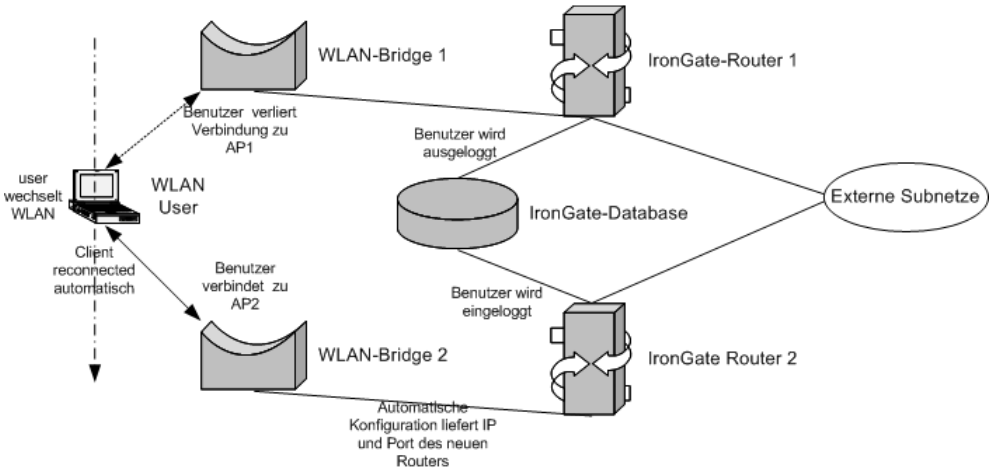

Abbildung 2.34: *Client-Handover mit IronGate*

2.5 Erweiterte Module/Funktionen

2.5.1 IPv6, mobileIPv6

Grundsätzlich ist Irongate als Router-Ersatz dazu prädestiniert, auch auf das Protokoll IPv6 aufzusetzen, um v.a. in modernen Provider-Infrastrukturen eingesetzt werden zu können. Durch den Einsatz eines v6-Stacks würde der Router auch die erweiterten Möglichkeiten, noch ganz unabhängig von IronGate, nutzen können, die IPv6 bietet. In diesem Kapitel sei einmal auf die Umlegung der bisherigen Möglichkeiten von IronGate auf IPv6 eingegangen. Grundsätzlich ist eine Erweiterung von IronGate auf IPv6 erst dann sinnvoll, wenn das komplette Projekt in die Objektorientierung (Java, C++) umgesetzt wurde, da es ansonst nur mehr mit erheblichem Aufwand wartbar bleiben würde. Daher gehen die weiteren Ausführungen davon aus, dass Java bereits als Sprache für den Server verwendet wird.

IronGate ist ein durchgehend auf IPv4 ausgelegtes Projekt, weshalb es unmfangreicher Änderungen bedarf, um den Server an IPv6 anzupassen. Im folgenden sind diese aufgelistet:

- Der bisherige Client muss auch IPv6-Adressen und Sockets akzeptieren und auf diese zugreifen können - dazu müssten die Java-Klassen für IPv6-Sockets in die Implementierung mit einbezogen werden und die Konfigurations- sowie Autoconfiguration-Dialoge und Funktionen entsprechend angepasst werden

- Die Autoconfiguration-Scripts des Servers müssten in zweifacher Ausführung vorliegen, um eine automatische Konfiguration auch für IPv6-Netze zu ermöglichen

- Die Konfiguration der Server-Interfaces müsste auf IPv6 umgestellt werden

- Es müsste eine eigene Klasse für einen IPv6-Benutzer angelegt werden, da sich seine Port- sowie IP-Einstellungen maßgeblich ändern würden

- Eine weitere Klasse explizit für ip6tables (iptables für IPv6) müsste angelegt werden, dieses neue Paket installiert sowie die Klasse daran angepasst werden (Klasse firewallv6 e.g.)

- Das Paket iproute2, aus welchem heraus IronGate das Programm ttcfür das CBQ verwendet, ist bereits IPv6-fähig - dennoch müsste eine eigene Klasse angelegt werden, welche die Steuerung des CBQ für IPv6 übernimmt, da sich die übergebenen Variablen v.a. in Bezug auf die IPv6-Adresse unterscheiden werden

- Das Konfigurationsskript des Servers muss modifiziert werden, da die interne und externe Subnetklassifizierung sich ändern würde - u.U. müssten zwei Konfigurationsskripts erstellt werden, anhand derer Existenz der Server sich auf IPv4, IPv6 oder beide Protokolle einstellen kann

- Es müsste eine neue Klasse für das IPv6 nsupdate erstellt werden, da sich das Dynamic Update beim IPv6 ändert

- Ein modifiziertes ARP müsste installiert werden, welches wiederum einer eigenen Klasse bedarf, da das ARP beim IPv6 andere IP-Adressen erwartet bzw. andere Outputs zurückgibt

- Der Packetlogger könnte u.U. leicht angepasst werden müssen, falls sich die Header-Informationen der ip6tables LOG-Funktion ändern

- Die Klassen, welche die Paket- und Bytezähler der einzelnen Chains auslesen, müssten ebenfalls umgeschrieben werden bzw. eigene IPv6-Klassen erstellt werden, da ip6tables vermutlich andere Rückgabevariablen beim Auslesen liefert als die bisherigen iptables

Im Falle einer Java-Implementierung müsste neben grundsätzlichen Systemänderungen (IPv6-Stack, IPv6-Routing) mit ca. 5 neuen Klassen gerechnet werden, sowie einer Modifikation des Clients, wobei das Web-Interface entgegen vollkommen unverändert bleiben könnte.

2.5.2 Diameter-Backend

Diameter, als Weiterentwicklung des RADIUS-Protokolls bzw. dessen Nachfolger, ist wie RADIUS ein AAA-Protokoll zur Kommunikation mit einer zentralen Accouting-Datenbank. Die momentane Verfügbarkeit an Bibliotheken zur Implementierung einer Diameter-Schnittstelle für IronGate ist eher gering, rein theoretisch wäre es für IronGate jedoch problemlos möglich, Diameter ebenso wie RADIUS zur Kommunikation mit entsprechenden Backends zu verwenden. IronGate arbeitet jedoch nicht mit Tickets, was bei einer Implementierung berücksichtigt werden müsste.

2.5.3 Java-Portierung Server

Der IronGate-Server ist momentan in Perl 5.8 implementiert. Perl eignet sich zwar herausragend für einfaches und schnelles Implementieren (beinahe schon Rapid Developement) bzw. zur Prototypenerstellung von Softwareprojekten und Machbarkeitsanalysen. Da IronGate im Laufe der Jahre jedoch immer weiter gewachsen ist, machen sich schon die Probleme einer rein prozeduralen Programmierung an mehreren Ecken bemerkbar, allen voran die Möglichkeit der echten Modularisierung.

Bisherige Recherchen ergaben, dass die Verarbeitungsgeschwindigkeit von Java in etwa der von Perl ebenbürtig ist, was an den bisherigen Messergebnissen und Performance-Erwartungen also nichts ändern wird. Konkret würde der Portierungsaufwand relativ hoch sein, da Java sehr viel strenger mit Datentypen umgeht und Perl die Typ-Umwandlungen selbst übernimmt (dynamische Datentypen). Desweiteren müssten neue Lösungen für die Backend-Implementierung gefunden werden sowie das Threading-Modell von Java durchleuchtet werden.

3 Die Implementierung

Auf den folgenden Seiten wird auf die Details der Implementierung eingegangen, aufbauend auf die im Kapitel über den Ablauf behandelten Routinen. Dabei wird vor allem auf die Ansteuerung der externen Programme wert gelegt sowie auf die speziell für IronGate entwickelten Algorithmen in ihrer implementierten Form.

3.1 Die externen Programme im Detail

Die folgenden Absätze zeigen Ausschnitte aus den Manuals der einzelnen von IronGate angesteuerten externen Programme unter Linux. Dadurch werden die in der Implementierung gezeigten Programmzeilen und Übergabeparameter verständlicher.

3.1.1 arp

Das Linux-Systemtool „arp“ dient zur Aufschlüsselung von IP-Adressen in MAC-Adressen und umgekehrt. In IronGate wird es verwendet, um die entsprechende MAC-Adresse zu den IPs der einzelnen Benutzer zu ermitteln, da die Programmiersprache Perl standardmäßig keine Bibliothek besitzt, welche Funktionen besäße, die wiederum die MAC-Adresse direkt als Rückgabewert liefern. Die MAC-Adresse ist jedoch vor allem für die Zuweisung an einzelne NICs und damit Maschinen der jeweiligen Benutzer sowie für die Anti-Spoof Mechanismen notwendig.

„arp“ kennt folgende für IronGate relevante Parameter (Auszug aus man 8 arp):

```
-n, --numeric:  shows  numerical  addresses  instead of trying to determine
symbolic host, port or user names

-a [hostname], --display [hostname]: Shows  the  entries of  the  specified
hosts. If the hostname parameter is not used, all entries will be displayed.
```

3.1.2 iptables

Das Kernel-Control Tool „iptables“ steuert die Netfilter-Paketfilter und -Manipulierungsfunktionen von Linux. Es wird verwendet, um Regeln zu setzen, zu verändern oder zu löschen. Die genaue Funktionsweise wurde eingehend in den vorderen Kapiteln erläutert, im folgenden werden die Parameter von iptables aufgezählt, welche für IronGate von Relevanz sind (Auszug aus man 8 iptables):

```
      -t, --table table
             This  option  specifies the packet matching table which the com-
```

```
       mand should operate on.  If the kernel is configured with  auto-
       matic module loading, an attempt will be made to load the appro-
       priate module for that table if it is not already there.

-A, --append chain rule-specification
       Append one or more rules to the end of the selected chain.  When
       the  source  and/or  destination  names resolve to more than one
       address, a rule will be added for each possible address combina-
       tion.

-D, --delete chain rule-specification
-D, --delete chain rulenum
       Delete one or more rules from the selected chain.  There are two
       versions of this command: the rule can be specified as a  number
       in  the  chain  (starting  at 1 for the first rule) or a rule to
       match.

-I, --insert chain [rulenum] rule-specification
       Insert one or more rules in the selected chain as the given rule
       number.   So,  if  the  rule  number is 1, the rule or rules are
       inserted at the head of the chain.  This is also the default  if
       no rule number is specified.

-L, --list [chain]
       List  all rules in the selected chain.  If no chain is selected,
       all chains are listed.  As  every  other  iptables  command,  it
       applies  to  the specified table (filter is the default), so NAT
       rules get listed by
        iptables -t nat -n -L
       Please note that it is often used with the -n option,  in  order
       to  avoid  long reverse DNS lookups.  It is legal to specify the
       -Z (zero) option as well, in which case  the  chain(s)  will  be
       atomically  listed  and zeroed.  The exact output is affected by
       the other arguments given. The exact rules are suppressed  until
       you use
        iptables -L -v

-F, --flush [chain]
       Flush the selected chain (all the chains in the table if none is
       given).  This is equivalent to deleting all  the  rules  one  by
       one.

-N, --new-chain chain
       Create  a  new user-defined chain by the given name.  There must
       be no target of that name already.

-X, --delete-chain [chain]
       Delete the optional user-defined chain specified.  There must be
       no  references  to  the chain.  If there are, you must delete or
       replace the referring rules before the chain can be deleted.  If
       no  argument  is  given,  it  will  attempt to delete every non-
       builtin chain in the table.

-p, --protocol [!] protocol
       The protocol of the rule or of the packet to check.  The  speci-
       fied protocol can be one of tcp, udp, icmp, or all, or it can be
       a numeric value, representing one of these protocols or  a  dif-
       ferent  one.   A  protocol  name  from  /etc/protocols  is  also
       allowed.  A ! argument before the protocol inverts  the  test.
       The  number  zero is equivalent to all.  Protocol all will match
       with all protocols and is taken as default when this  option  is
       omitted.
```

```
       -s, --source [!] address[/mask]
              Source specification.  Address can be either a network name, a
              hostname (please note that specifying any name to be resolved
              with a remote query such as DNS is a really bad idea), a network
              IP address (with /mask), or a plain IP address.  The mask can be
              either a network mask or a plain number, specifying the number
              of 1's at the left side of the network mask.  Thus, a mask of 24
              is equivalent to 255.255.255.0.  A ! argument before the
              address specification inverts the sense of the address. The flag
              --src is an alias for this option.

       -d, --destination [!] address[/mask]
              Destination specification.  See the description of the -s
              (source) flag for a detailed description of the syntax.  The
              flag --dst is an alias for this option.

       -j, --jump target
              This specifies the target of the rule; i.e., what to do if the
              packet matches it.  The target can be a user-defined chain
              (other than the one this rule is in), one of the special builtin
              targets which decide the fate of the packet immediately, or an
              extension (see EXTENSIONS below).  If this option is omitted in
              a rule, then matching the rule will have no effect on the
              packet's fate, but the counters on the rule will be incremented.

       -i, --in-interface [!] name
              Name of an interface via which a packet is going to be received
              (only for packets entering the INPUT, FORWARD and PREROUTING
              chains).  When the ! argument is used before the interface
              name, the sense is inverted.  If the interface name ends in a
              "+", then any interface which begins with this name will match.
              If this option is omitted, any interface name will match.

-o, --out-interface [!] name
              Name of an interface via which a packet is going to be sent (for
              packets entering the FORWARD, OUTPUT and POSTROUTING chains).
              When the ! argument is used before the interface name, the
              sense is inverted.  If the interface name ends in a "+", then
              any interface which begins with this name will match.  If this
              option is omitted, any interface name will match.

       -v, --verbose
              Verbose output.  This option makes the list command show the
              interface name, the rule options (if any), and the TOS masks.
              The packet and byte counters are also listed, with the suffix
              'K', 'M' or 'G' for 1000, 1,000,000 and 1,000,000,000 multipli-
              ers respectively (but see the -x flag to change this).  For
              appending, insertion, deletion and replacement, this causes
              detailed information on the rule or rules to be printed.

       -n, --numeric
              Numeric output.  IP addresses and port numbers will be printed
              in numeric format.  By default, the program will try to display
              them as host names, network names, or services (whenever appli-
              cable).

       -x, --exact
              Expand numbers.  Display the exact value of the packet and byte
              counters, instead of only the rounded number in K's (multiples
              of 1000) M's (multiples of 1000K) or G's (multiples of 1000M).
              This option is only relevant for the -L command.
```

```
       --source-port [!] port[:port]
              Source  port  or  port range specification. This can either be a
              service name or a port number. An inclusive range  can  also  be
              specified,  using  the  format  port:port.  If the first port is
              omitted, "0" is assumed; if the  last  is  omitted,  "65535"  is
              assumed.  If the second port greater then the first they will be
              swapped.  The flag  --sport  is  a  convenient  alias  for  this
              option.

       --destination-port [!] port[:port]
              Destination  port or port range specification.  The flag --dport
              is a convenient alias for this option.

       --tcp-flags [!] mask comp
              Match when the TCP flags are as specified.  The  first  argument
              is  the  flags which we should examine, written as a comma-sepa-
              rated list, and the second argument is a comma-separated list of
              flags which must be set.  Flags are: SYN ACK FIN RST URG PSH ALL
              NONE.  Hence the command
               iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST SYN
              will only match packets with the SYN flag set, and the ACK,  FIN
              and RST flags unset.

       --syn
              Only  match TCP packets with the SYN bit set and the ACK and FIN
              bits cleared.  Such packets are used to request  TCP  connection
              initiation;  for  example,  blocking  such  packets coming in an
              interface will prevent incoming TCP  connections,  but  outgoing
              TCP  connections will be unaffected.  It is equivalent to --tcp-
              flags SYN,RST,ACK SYN.  If the ! flag  precedes  the  "--syn",
              the sense of the option is inverted.

       --mac-source [!] address
              Match  source  MAC  address.    It  must  be  of  the  form
              XX:XX:XX:XX:XX:XX.   Note that this only makes sense for packets
              coming from an Ethernet device and entering the PREROUTING, FOR-
              WARD or INPUT chains.

       --mark value[/mask]
              Matches packets with the given unsigned mark value (if a mask is
              specified, this is logically ANDed with the mask before the com-
              parison).

       --tos tos
              The argument is either a standard name, (use
               iptables -m tos -h
              to see the list), or a numeric value to match.

--length length[:length]

       --log-level level
              Level of logging (numeric or see syslog.conf(5)).

       --log-prefix prefix
              Prefix log messages with the specified prefix; up to 29  letters
              long, and useful for distinguishing messages in the logs.

       --log-tcp-sequence
              Log  TCP sequence numbers. This is a security risk if the log is
              readable by users.
```

```
       --log-tcp-options
              Log options from the TCP packet header.

       --log-ip-options
              Log options from the IP packet header.

       --set-mark mark

       --set-tos tos
              You can use a numeric TOS values, or use
               iptables -j TOS -h
              to see the list of valid TOS names.

--to-source  ipaddr[-ipaddr][:port-port]
              which can specify a single new source IP address,  an  inclusive
              range  of  IP  addresses, and optionally, a port range (which is
              only valid if the rule also specifies -p tcp or -p udp).  If  no
              port  range  is  specified,  then source ports below 512 will be
              mapped to other ports below 512:  those  between  512  and  1023
              inclusive  will  be  mapped to ports below 1024, and other ports
              will be mapped to 1024 or above. Where possible, no port  alter-
              ation will occur.

       --to-destination ipaddr[-ipaddr][:port-port]
              which can specify a single new destination IP address, an inclu-
              sive  range of IP addresses, and optionally, a port range (which
              is only valid if the rule also specifies -p tcp or -p udp).   If
              no port range is specified, then the destination port will never
              be modified.

--to-ports port[-port]
              This  specifies  a  destination  port  or range of ports to use:
              without this, the destination port is never  altered.   This  is
              only valid if the rule also specifies -p tcp or -p udp.

%
```

3.1.3 iproute2

Das iproute2-Paket besteht im Groben aus zwei Tools, einerseits „ip“, welches der Modifikation der Routen-Tabellen dient und der Konfiguration der einzelnen Interfaces (der Nachfolger von „ifconfig“ und „route“) sowie dem Programm „tc“ für Traffic Control. IronGate verwendet ausschließlich „tc“ und konfiguriert über dieses nur den CBQ-Shaper (Funktionsweise siehe im entsprechenden Kapitel). Da „tc“ ein unheimlich mächtiges Werkzeug zur Konfiguration aller Traffic-Shaping und Priorizing Funktionen des Kernels und sämtlicher verfügbarer Algorithmen (bspw. PRIO, CBQ, TBF, RED etc.) ist, würde eine Aufzählung sämtlicher Parameter den Rahmen sprengen, darunter auch die Beschreibung der alleine für IronGate verwendeten Parameter und ihre Bedeutung. Das unter [10] erreichbare Manuskript geht detailliert auf alle Funktionen ein. Ganz grob verwendet IronGate die folgenden beiden Funktionen, um Regeln hinzuzufügen und zu entfernen:

```
add    Add a qdisc, class or filter to a node. For all entities, a par-
       ent  must  be  passed,  either by passing its ID or by attaching
```

```
       directly to the root of a device.  When creating a  qdisc  or  a
       filter,  it  can  be named with the handle parameter. A class is
       named with the classid parameter.

remove A qdisc can be removed by specifying its handle, which may  also
       be  'root'.  All  subclasses and their leaf qdiscs are automati-
       cally deleted, as well as any filters attached to them.
```

3.1.4 nsupdate

Nsupdate ist ein im ISC-Bind-Paket enthaltenes Tool zur dynamischen Aktualisierung von DNS-Zonendateien von anderen Maschinen aus. IronGate verwendet das Tool, um Benutzern während Ihrer Session A NAME und IN PTR-Records für ihre IP gemappt auf ihren Benutzernamen anzulegen. Der lokale DNS-Server wird aus der Auflösungskonfiguration des Systems (resolv.conf) entnommen. Die Konfiguration des Bind-Servers für dynamische Einträge wird detailliert in [11] gezeigt. Da lediglich Einträge gemacht und nach einer Session wieder gelöscht werden, verwendet IronGate folgende zwei Parameter:

```
update delete domain-name [ ttl ]  [ class ]  [ type  [ data... ]  ]
       Deletes  any  resource  records  named domain-name.  If type and
       data  is  provided,  only  matching  resource  records  will  be
       removed.   The  internet  class  is assumed if class is not sup-
       plied. The ttl is ignored, and is only allowed  for  compatibil-
       ity.

update add domain-name ttl [ class ]  type data...
       Adds  a  new  resource  record with the specified ttl, class and
       data.
```

3.1.5 gpg

Gpg ist eine freie Implementierung des PGP-Verschlüsselungssystems auf Basis des IDEA-Algorithmus. Um die Schlüssel auszulesen und an den Client weiterzuleiten sowie dessen verschlüsselte Strings wieder in Klartext umzuwandeln, verwendet IronGate das Tool „gpg“ und das File „irongate.key“ im Verzeichnis, von dem aus IronGate gestartet wurde. „gpg“ kennt eine Vielzahl an Parametern, von welchen folgende von IronGate verwendet werden:

```
--export  export keys
--yes  assume yes on most questions
--batch  batch mode never ask
-a  create ascii armoured output
-o  use as output file
--passphrase-fd 0 read password from file descriptor (0)=> Hard disk
--decrypt  decrypt data
```

3.1.6 syslog

Das Tool syslog aus der sysklogd-Suite ist der System-Log-Daemon, welcher über den Kernel und dessen Loglevel gespeist wird. Es konfiguriert sich aus der Datei /etc/syslog.conf selbst, diese muss jedoch speziell auf eine Named-Pipe (FIFO) konfiguriert sein. ironGate verwendet den Debug-Level für das Loggen der Pakete (Level 0), was normale System-Logs nicht betrifft, daher wird der System-Log-Daemon angewiesen, alle Pakete an diesen Loglevel in eine speziell konfigurierte Named-Pipe zu schreiben, welche wiederum vom IronGate-Packetlog-Daemon ausgelesen wird. Folgender Eintrag ist hierfür in /etc/syslog.conf nötig:

```
kern.=debug |/root/iptables.log.pipe
```

3.2 Der Server

Auf den folgenden Seiten wird der PERL-Code einzelner Routinen im IronGate-Server für Linux dargestellt und gegebenenfalls kommentiert. Es werden hierbei nur die interessantesten Codeteile aufgezeigt, und bei langen Listings nur Beispiele gegeben. Da IronGate prozedural programmiert ist und mit weit über 200 Variablen arbeiten muss, wird nicht jede Variable einzeln beschrieben - der Sinn ergibt sich aus der Verwendung und Bezeichnung.

3.2.1 Die Initialisierung

Beim Start des Servers werden die grundsätzlichen Regeln der Firewall und des Traffic-Shapings angelegt (Chains und Classes). Dies erfolgt über die beschriebenen Tools „tc“ und „iptables“. Folgende Befehle werden hierfür über Perl-„system“ an die Shell übergeben:

```
system "iptables -t nat -N irongate";
system "iptables -t mangle -N irongate";
system "iptables -N irongate";
system "iptables -t nat -I PREROUTING -j irongate";
system "iptables -t mangle -I PREROUTING -j irongate";
system "iptables -t mangle -I POSTROUTING -j irongate";
system "iptables -I FORWARD -j irongate";
system "iptables
-t nat -I irongate -s $internal_subnet -j DROP";
system "iptables
-I irongate -s $internal_subnet -j DROP";
```

```
system "tc qdisc add dev $external_if root
handle 1: cbq bandwidth 100Mbit avpkt 1000";
system "tc qdisc add dev $internal_if root
handle 1: cbq bandwidth 100Mbit avpkt 1000";
```

Um trotz der auf DROP gesetzten Regeln des PREROUTINGS eine Verbindung zwischen Client und Server aufbauen zu können sowie auch die HTTP-Redirection (neben dem notwendigen DHCP- und DNS-Redirecting) ermöglichen zu können, werden folgende Regeln gesetzt:

```
system "iptables -t nat -I irongate -p tcp --dport 80
-s $internal_subnet -j REDIRECT --to-port 80";
system "iptables -t nat -I irongate -p tcp --dport 80
-s $internal_subnet -j REDIRECT --to-port 80";
system "iptables -t nat -I irongate -p udp --dport 53
-s $internal_subnet -j REDIRECT --to-port 53";
system "iptables -t nat -I irongate -p tcp --dport
$server_port -s $internal_subnet -j REDIRECT --to-port
$server_port";
system "iptables -t nat -I irongate -p icmp -d
$internal_serverip -j ACCEPT";
system "iptables -t nat -I irongate -p udp
--dport 67 -j ACCEPT";
system "iptables -t nat -I irongate -p udp
--dport 68 -j ACCEPT";
```

Die nachfolgenden Programmzeilen initialisieren das Forking (Multiprozess-Handling) für aufbauende Verbindungen und initialisieren die Datenbankverbindung beim Eintreffen einer neuen TCP-Verbindung.

```
$server=IO::Socket::INET->new(LocalPort => $server_port, Type =>
SOCK_STREAM, Reuse => 1, Listen => 10) or die "socket failed: $!";

my $client;

my $goon = 1;
while ($goon)
{
while (($client,$client_address) = $server->accept)
```

```
{
next if $pid = fork;
die "fork: $!" unless defined $pid;

$dbh = DBI->connect('DBI:mysql:' . $irongate_mysql_db . '
:' . $irongate_mysql_server . ':' . $irongate_mysql_port,
$irongate_mysql_username, $irongate_mysql_password) or
die $DBI::errstr;
```

Ab diesem Zeitpunkt wurde die Verbindung zwischen einem Client- und einem Server hergestellt und die Handshake-Sequenz wird eingeleitet.

3.2.2 Ermitteln grundlegender Werte

Um anhand des Tools „arp“ die MAC-Adresse des Clients zu ermitteln, wird folgender Programmcode verwendet:

```
open ARP, $pathto_arp;
$mac = "";
while ($arpentry = <ARP>)
{
@splitarp = split(/ +/,$arpentry);
if ($splitarp[0] eq $ip)
{
$mac = $splitarp[2];
}
}
close ARP;
if ($mac eq '')
{
$mac = "remote";
}
```

Falls die Verbindung nicht aus dem gleichen Subnet erfolgt, kann aufgrund netzwerktechnischer Gegebenheiten nicht auf die MAC-Adresse rückgeschlossen werden, woraufhin diese als „remote“ deklariert wird.

Login-Zeit und Datum wird über die Systemzeit in UNIX-Seconds ermittelt. Diese Werte dienen dem Time Accounting und dem Logging:

```
($sec,$min,$hour,$mday,$mon,$year,undef,undef,undef) =
localtime(time);
```

```
$logintime = $hour . ":" . $min . ":" . $sec;
$logindate = ($year+1900) . "-" . ($mon+1) . "-" . $mday;
$currentunixtime = $sec + $min * 60 + $hour * 3600;
```

3.2.3 Handshaking und Authentifikation

Neben dem schon im Flussdiagramm dargestellten Ablauf der Authentifikation und des Handshakings, welcher auf einfacher Socketkommunikation beruht, dem Austausch von Strings, ist besonders die Erstellungs des Salts und die Entschlüsselung sowie der Schlüsselaustausch relevant. Der Public-Key wird mit folgenden Programmzeilen extrahiert und dem Client übermittelt:

```
system('gpg --export --yes --batch -a -o
./irongate.key irongate');
open(FH,"< irongate.key");
while($Zeile = <FH>)
{
print $client "$Zeile";
}
print $client "ENDE\n";
close(FH);
```

Der Salt wird erzeugt und an den Client übermittelt:

```
use integer;
$salt=0;
$salt=(rand(1000))/1;
no integer;
print $client "$salt\n";
```

Die vom Client empfangenen verschlüsselten Werte werden in einem File zwischengespeichert und über „gpg“ wieder entschlüsselt.

```
open(FH,"> verschl.cry");
$empfang = <$client>;
$empfang = substr($empfang,0,(length $empfang)-2);
while($empfang ne "ENDE")
{
$empfang = <$client>;
```

```
$empfang = substr($empfang,0,(length $empfang)-2);
print FH "$empfang\n";
}
close(FH);
system("echo $gpg_passphrase | gpg -o ./entschl.cry
--passphrase-fd 0 --yes --decrypt ./verschl.cry");
open(FH,"< entschl.cry");
```

3.2.4 Backend-Ansteuerung

Die als Schleife definierte Backend-Ansteuerung läuft nach dem Schema ab, die Übergabe-Parameter an die Backends über „system“ zu parsen und die Rückgabewerte aus dem Output des externen Scripts zu lesen. Bspw. der folgenden Codezeilen folgend:

```
$return_value = system("./irongate_module_ads.pl
$login_username $login_password");
if ($return_value == 0)
{
$username_ok = "yes";
$password_ok = "yes";
$username = $login_username;
$backend = "ads";
}
```

3.2.5 Limitberechnung

Nachdem über geschachtelte Routinen alle Informationen über den Benutzer, sofern er existiert, aus dem IronGate-Backend und etwaigen anderen Backend-Modulen gefetched wurden, erfolgt die Berechnung der Zeit- und Volumslimits. Zuerst wird überprüft, ob ein fixer Zeitrahmen (z.B. von 8-16h täglich) gegeben wurde.

```
if ($timelogout)
{
@logouttimesplit = split(/:/, $timelogout);
$bytimelimit = timelocal($logouttimesplit[0],
$logouttimesplit[1],$logouttimesplit[2],$mday,$mon,$year)
- time;
$logouttimeint = $logouttimesplit[2] + $logouttimesplit[1]
* 60 + $logouttimesplit[0] * 3600;
if ($bytimelimit < 0) {$bytimelimit = 0;}
if ($logouttimeint < $currentunixtime) {$loginexception
```

```
= "yes";}
}
if ($timelogin)
{
@logintimesplit = split(/:/, $timelogin);
$logintimeint = $logintimesplit[2] + $logintimesplit[1]
* 60 + $logintimesplit[0] * 3600;
if ($logintimeint > $currentunixtime) {$loginexception
= "yes";}
}
if ($datelogout)
{
@logoutdatesplit = split(/-/, $datelogout);
$bydatelimit = timelocal(0,0,0,$logoutdatesplit[2],
$logoutdatesplit[1],$logoutdatesplit[0]) - time;
$logoutdateint = timelocal(0,0,0,$logoutdatesplit[2],
$logoutdatesplit[1]+1,$logoutdatesplit[0]);
if ($bydatelimit < 0) {$bydatelimit = 0;}
if ($logoutdateint < time) {$logintimeexception = "yes";}
}
if ($datelogin)
{
@logindatesplit = split(/-/, $datelogin);
$logindateint = timelocal(0,0,0,$logindatesplit[2],
$logindatesplit[1]+1,$logindatesplit[0]);
if ($logindateint > time) {$logintimeexception = "yes";}
}
```

Es erfolgt die Aufsummierung der bisherigen „Verbräuche“ an Online-Zeit und Transfervolumen. Dies geschieht über eine Abzählung aller bisherigen Log-Einträge in einer Schleife und einer Zuweisung nach dem jeweiligen Datum.

```
$cursor = $dbh->prepare($authdataquery) or die $dbh->errstr;
$cursor->execute or (die $dbh->errstr, exit(-1));
$cursor->bind_columns(\$auth_volup, \$auth_voldown,
\$auth_onlinetime, \$auth_date);
while(($auth_volup,$auth_voldown,$auth_onlinetime,
$auth_date) = $cursor->fetchrow)
{
@auth_datesplit = split(/-/, $auth_date);
if (($auth_datesplit[0]) && ($auth_datesplit[1]) &&
```

```
($auth_datesplit[2]))
{
$auth_vol = $auth_volup + $auth_voldown;
$auth_date_unixtime = timelocal(0,0,0,
$auth_datesplit[2],$auth_datesplit[1],
$auth_datesplit[0]);
if ($auth_date_unixtime >= $dayunixtime)
{
$volupday_history =
$volupday_history + $auth_volup;
$voldownday_history =
$voldownday_history + $auth_voldown;
$volday_history =
$volday_history + $auth_vol;
$onlinetimeday_history =
$onlinetimeday_history + $auth_onlinetime;
}
if ($auth_date_unixtime >= $weekunixtime)
{
$volupweek_history =
$volupweek_history + $auth_volup;
$voldownweek_history =
$voldownweek_history + $auth_voldown;
$volweek_history =
$volweek_history + $auth_vol;
$onlinetimeweek_history =
$onlinetimeweek_history + $auth_onlinetime;
     }
if ($auth_date_unixtime >= $monthunixtime)
{
$volupmonth_history =
$volupmonth_history + $auth_volup;
$voldownmonth_history =
$voldownmonth_history + $auth_voldown;
$volmonth_history =
$volmonth_history + $auth_vol;
$onlinetimemonth_history =
$onlinetimemonth_history + $auth_onlinetime;
}
}
$volupoverall_history = $volupoverall_history + $auth_volup;
$voldownoverall_history =
$voldownoverall_history + $auth_voldown;
$voloverall_history = $voloverall_history + $auth_vol;
```

```
$onlinetimeoverall_history =
$onlinetimeoverall_history + $auth_onlinetime;
$runcounter++;
}
```

In Vergleichen wird daraufhin ermittelt, welcher der jeweilig geringste Wert der noch vorhandenen Kontingente ist, und dieser als gültig definiert.

3.2.6 Online-Überprüfung

Da ein Benutzer, eine IP oder eine MAC nur jeweils ein einziges Mal an einem IronGate-System angemeldet sein darf (andernfalls würden sich Regeln u.U. überschreiben oder ein Accouting fehlschlagen), wird vor dem Freischalten überprüft, ob diese Voraussetzungen zutreffen. Die Vergleiche für den Benutzer selbst oder auch die MAC-Adresse folgen hierbei folgendem Schema:

```
$onlinequery = "
SELECT ip
FROM online
WHERE ip = '$ip'
";
$cursor = $dbh->prepare($onlinequery);
$cursor->execute;
$ipused = "";
$ipdb = $cursor->fetchrow;
if ($ipdb) {$ipused = "yes";}
$cursor->finish;
```

3.2.7 User-ID Zuweisung

Wie im Kapitel über die Funktionen beschrieben, wird die Benutzer-ID bei jedem Einlogvorgang nach einem speziellen Algorithmus erneut vergeben. Dieser ist folgendermaßen ausgeführt:

```
$getfreeidquery= "
SELECT MIN(userid)
FROM freeids
";

$cursor = $dbh->prepare($getfreeidquery);
```

```
$cursor->execute;
$userid = $cursor->fetchrow;
$cursor->finish;

if ($userid ne "")
{
$deletefreeidquery= "
DELETE
FROM freeids
WHERE userid = '$userid'
";
$cursor = $dbh->prepare($deletefreeidquery);
$cursor->execute;
$cursor->finish;
}
else
{
$getnewidquery = "SELECT MAX(userid) FROM online";
$cursor = $dbh->prepare($getnewidquery);
$cursor->execute;
$userid = $cursor->fetchrow;
$cursor->finish;

$userid = $userid + 1;
}
```

Nach einer Sitzung wird die verwendete User-ID wieder in den Pool der „freeIDs“ übergeleitet, um für weitere Sessions zur Verfügung zu stehen (andernfalls würde sich die nächste verfügbare ID auf eine sehr hohe Zahl aufschaukeln, was zur Folge hätte, dass die iptables-Chains und die Traffic-Control versagen).

```
$dbh->do("INSERT INTO freeids (userid) VALUES ('$userid')");
```

3.2.8 Dynamic-DNS Update

Unter Verwendung von „nsupdate“ werden die dynamischen A NAME und IN PTR Records gesetzt, nachdem diese aus dem Benutzernamen und der Domain assembliert wurden.

```
$dnsname = $username . $domain;
```

```
@ipsplit = split(/\./, $ip);
$ptrip = $ipsplit[3] . "." . $ipsplit[2] . "." . $ipsplit[1] .
"." . $ipsplit[0] . ".in-addr.arpa";

open(NSUP, $pathto_nsupdate);
print NSUP "update add $dnsname $dns_ttl in a $ip\nupdate add
$ptrip $dns_ttl in ptr $dnsname\n\n";
close(NSUP);
```

Beim Ausloggen werden die dynamischen DNS-Einträge wieder aus den Zonendateien, abermals unter Zuhilfenahme von „nsupdate“ entfernt.

```
open(NSUP, $pathto_nsupdate);
print NSUP "update delete $dnsname 0 in a $ip\nupdate delete
$ptrip 0 in ptr $dnsname\n\n";
close(NSUP);
```

3.2.9 Firewall- und Traffic-Control

Nachdem ein Benutzer alle Kriterien erfüllte, wird er als Online eingetragen und für ihn sämtliche Regeln lt. seinem Konto auf die Firewall und die Traffic-Control sowie das Accounting übertragen. Dies geschieht anhand der Tools „iptables“ und „tc“.

```
if ($mac eq "remote")
{
system "iptables -I irongate$userid -j ACCEPT -s $ip";
}
else
{
system "iptables -I irongate$userid
-j ACCEPT -m mac --mac-source $mac -s $ip";
}
system "iptables -I irongate$userid -j ACCEPT -d $ip";

system "iptables -t nat -N irongate$userid";
system "iptables -t nat -I irongate -j irongate$userid -s $ip";
system "iptables -t nat -I irongate -j irongate$userid -d $ip";
system "iptables -t nat -I irongate$userid -j ACCEPT -s $ip";
system "iptables -t nat -I irongate$userid -j ACCEPT -d $ip";
system "iptables -t mangle -N irongate$userid";
```

```
system "iptables -t mangle -I irongate -j irongate$userid
-s $ip";
system "iptables -t mangle -I irongate -j irongate$userid
-d $ip";
system "iptables -t mangle -I irongate$userid 1 -s $ip
-d ! $server_subnet -j MARK --set-mark $userid";
system "iptables -t mangle -I irongate$userid 2 -d $ip
-s ! $server_subnet -j MARK --set-mark $userid";
system "iptables -I irongate$userid -p tcp --syn";
system "iptables -I irongate$userid -p udp";
system "iptables -I irongate$userid -s $ip";
system "iptables -I irongate$userid -d $ip";
```

Da IronGate nicht den (wenig rückwärtskompatiblen) Multiport-Filter verwendet, fügt es der Reihe nach offene und geblockte Ports, sofern eingetragen, in die entsprechende Chain ein.

```
if ($blockedports)
{
$rulecounter = 0;
@portssplit = split(/;/, $blockedports);
while ($portssplit[$rulecounter])
{
system "iptables -I irongate$userid
-s $ip -p tcp --dport $portssplit[$rulecounter] -j DROP";
$rulecounter++;
}
}
elsif ($openports ne "")
{
$rulecounter = 0;
@portssplit = split(/;/, $openports);
system "iptables -t nat -I irongate$userid -j DROP -s $ip";
while ($portssplit[$rulecounter])
{
system "iptables -t nat -I irongate$userid
-s $ip -p tcp --dport $portssplit[$rulecounter] -j ACCEPT";
print "Regel hinzugefuegt !\n";
$rulecounter++;
}
}
```

Der Vorgang wiederholt sich beim Abmelden des Benutzers mit dem Unterschied, dass die Regeln anstatt mit -I angelegt mit -D gelöscht werden.

Die Traffic-Control wird folgendermaßen eingerichtet:

```
$weight = int($bwup/10);
$weight = $weight . "Kbit";
$bwup = $bwup . "Kbit";
system "tc class add dev $external_if parent 1:0 classid
1:$userid cbq bandwidth $bwup rate $bwup allot 1514 weight
$weight prio 8 maxburst 20 avpkt 1000 bounded isolated";
system "tc filter add dev $external_if parent 1:0 protocol
ip prio $priority handle $userid fw flowid 1:$userid";

$weight = int($bwdown/10);
$weight = $weight . "Kbit";
$bwdown = $bwdown . "Kbit";
system "tc class add dev $internal_if parent 1:0 classid
1:$userid cbq bandwidth $bwdown rate $bwdown allot 1514
weight $weight prio 8 maxburst 20 avpkt 1000 bounded
isolated";
system "tc filter add dev $internal_if parent 1:0 protocol
ip prio $priority handle $userid fw flowid 1:$userid";
```

Die Regeln werden nach Beendigung der Session wieder auf gleiche Weise wie die Chains wieder entfernt, anstatt ein „add“ zu verwenden, werden die Regeln per „del“ gelöscht.

3.2.10 Traffic-Counter

Die Traffic-Counter der einzelnen Chains werden während jedes Zyklus abgefragt und die Differenz zur letzten Session berechnet. Dies dient während der Keep-Alive-Sequenz dem Traffic-Shaping und dem Volumen-Limit, beim finalen Auslesen dem Accouting. Das Auswerten der Chain-Werte erfolgt folgendermaßen:

```
$bytecountup_last = $bytecountup;
$bytecountdown_last = $bytecountdown;
$bytecount_last = $bytecountup+$bytecountdown;

open BYTECOUNT, "iptables -nvxL irongate$userid |";
$race = 0;
```

```
while ($bytecount = <BYTECOUNT>)
{
$race++;
@splitnvxl = split(/ +/,$bytecount);
if ($race == 3) {$bytecountdown = $splitnvxl[2];
$packetcountdown = $splitnvxl[1];}
if ($race == 4) {$bytecountup = $splitnvxl[2];
$packetcountup = $splitnvxl[1];}
if ($race == 5) {$packetcountudp = $splitnvxl[1]};
if ($race == 6) {$syncount = $splitnvxl[1]};
}
close BYTECOUNT;

$throughputup = int(($bytecountup-$bytecountup_last)/$delay);
$throughputdown = int(($bytecountdown-$bytecountdown_last)/$delay);
$throughput = $throughputup+$throughputdown;
```

Das Auslesen der Byte- und Packetcounter am Ende der Session erfolgt gleichermaßen. Mit den so gesammelten Werten wird auch die Usage der Verbindung und der Load berechnet, wie in den entsprechenden Kapiteln beschrieben. Die Implementierung sieht folgendermaßen aus:

```
if ($throughput < 10000) {$load = "low"}
elsif ($throughput < 30000) {$load = "medium"}
elsif ($throughput < 60000) {$load = "high"}
else {$load = "extreme";}
$packetcountsyn = $syncount - $packetcountsyn;
if ($packetcountsyn > 50) {$abnormal="yes";}

if (($packetcountsyn < 20) && ($throughput < 30000))
{$typeofuse = "surfing";}
elsif (($packetcountsyn > 20) && ($throughput > 30000))
{$typeofuse = "sharing";}
elsif (($packetcountsyn > 100) && ($throughput < 10000))
{$typeofuse = "scanning";}
elsif (($packetcountsyn < 20) && ($throughput > 20000))
{$typeofuse = "leeching";}
elsif (($packetcountudp > 20) && ($throughput < 20000))
{$typeofuse = "gaming";}
elsif (($packetcountudp > 20) && ($throughput > 20000))
{$typeofuse = "streaming";}
elsif (($packetcountsyn < 10) && ($throughput < 2000))
```

```
{$typeofuse = "idle";}
else {$typeofuse = "undeterm";}
```

3.3 Die Clients

Der Visual-Basic-Client, als Teil der Bakkalaureatsarbeit „Irongate“ [12], ist in dieser genau beschrieben. Der Java-Client sowie der Java-Applet-Client wird ausführlich in [REFENZ STAHL] dokumentiert.

3.4 Das Web-Interface

Die Implementierung des Web-Interfaces erfolgte ausschließlich in PHP4. Es wurden die Standard-mySQL-Schnittstellen ebenso verwendet wie übliche Programmkonstrukte. Da es sich lediglich um ein Verwaltungs-Frontend für die den mySQL-Tabellen liegenden Informationen handelt, beinhaltet es keine neuartigen Algorithmen.

4 Testläufe

Im Laufe der Entwicklung und nach Abschluss der Implementierung wurden zahlreiche Tests im Bereich der Performance und Belastbarkeit in Hinsicht vielerlei Aspekte durchgeführt, welche in diesem Kapitel zusammengefasst wiegegeben werden.

4.1 Testumgebungen

Als Testumgebungen dienten zwei 19“ Maschinen mit 1,6GHz Kernen als IronGate-Server, die auf Basis eines Linux LFS-Systems (siehe [13]) arbeiteten. Als NICs wurden D-Link Serverkarten (DX-580) verwendet. Die Softwareausstattung der Server umfasste folgende Pakete:

- Linux Kernel 2.6.3 mit Kernelpachtes (siehe Implementierung) - Systemkern
- Apache 2.0 mit PHP-Support - Webserver
- mySQL 4.0 - SQL-Server
- Bind 9 - Nameserver
- iproute2 und iptables bzw. iptables6 - Netfilter- und CBQ-Steuerungsprogramme

Als Clients kamen ein Desktop-PC und ein Notebook mit Orinoco-WLAN-Adapter zum Einsatz. Als Wireless Access Points wurden ein CISCO Aeronet 1500 Access-Point sowie ein Surecom WAP verwendet.

Für einen Teil der Tests wurde ein eigenes Script entwickelt, welches die Last des Servers reguliert und eine vorgegebene Anzahl an eingeloggten Clients simuliert (durch massive parallele Verbindungen). Das Tool wird im Kapitel der Implementierung näher erklärt. Um parallele Logins von einer Maschine aus zu simulieren, wurde zu diesem Zweck der mehrfache Login einer IP erlaubt.

4.2 Test- und Messergebnisse

Die Ergenisse der Tests, welche während und nach der Implementierung von IronGate durchgeführt wurden, werden durch Langzeiterfahrungen durch die Vorversion von IronGate (MOSIAP) ergänzt, welches zur Zeit der Schriftlegung dieses Dokumentes seit ca. 3 Jahren problemlos im Dauereinsatz ca. 100 Client-Maschinen versorgt. Die Erkenntnisse durch den Produktionseinsatz von MOSIAP erübrigen Langzeittests von IronGate, da IronGate die Architektur von MOSIAP übernommen hat und dessen Architektur bereits auf den Langzeiteinsatz ausgelegt war. Nähere Informationen zu MOSIAP sind unter [14] zu finden.

4.2.1 Belastungs- und Reaktionsmessungen

Zunächst wurde die Serverlast bei steigender Anzahl der Benutzer nebst der Speicherauslastung des Servers gemessen.

Hierfür wurde der in Abbildung 4.1 dargestellte Versuchsaufbau 1 herangezogen.

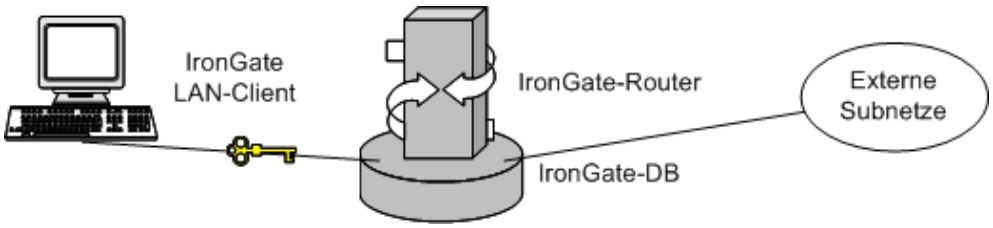

Abbildung 4.1: *Versuchsaufbau 1*

Anhand des für die Tests entwickelten Tools wurde die Anzahl der Benutzer stetig erhöht und jeweils die Speicher- und CPU-Auslastung gemessen. Die Messergebnisse zeigten, dass die Ansprüche an das System linear und streng proportional mit der Anzahl der Benutzer steigt.

Anhand des gleichen Versuchsaufbaues wurden auch die Reaktionszeiten des Servers bzgl. weiterer Login-Versuche gemessen. Hierbeit wurde auch unterschieden, ob die Authentifizierung verschlüsselt oder unverschlüsselt geschieht. Die unverschlüsselte Authentifizierung geht naturgemäß schneller vonstatten, da keine Ver- und Entschlüsselungsroutinen durchlaufen werden müssen und auch weder Keys noch verschlüsselte Strings übertragen werden müssen. Im Normalfall dauert der Login-Prozess in etwa zwei Sekunden länger, wenn ein verschlüsselter Login abläuft. Die Ergebnisse der Messung zeigen die Kurven in Abbildung 4.2.

Sie zeigen deutlich, dass der IronGate-Server stabil gegenüber der Anzahl der gleichzeitig sich online befindlichen Benutzer ist und die Reaktionszeit niemals deutlich sinkt, woraus sich auch schließen lässt, dass eine sehr starke (Multiprozessor-)maschine durchaus über 1000 Clients gleichzeit verwalten können sollte.

Um auch die Performance-Unterschiede während des Login-Prozesses zwischen einem IronGate-Server mit lokaler Datenbank und einem Server mit ausgelagerter Datenbank aufzuzeigen, wurde Versuchsaufbau 2 (Abbildung 4.3) erstellt und in diesem wiederum die Loginzeiten im verschlüsselten und unverschlüsselten Modus in Abhängigkeit von der Benutzeranzahl gemessen.

Hierbei zeigte sich, dass eine ausgelagerte Datenbank (im selben externen Subnet mit LAN-Latencies) kaum messbare Performance-Unterschiede hervorruft. Die Einlogzeiten verlängern sich aufgrund der etwas höheren Verzögerungs- und Reaktionszeiten um durchschnittlich 0,1-0,2 Sekunden, was Abbildung 4.4 veranschaulicht.

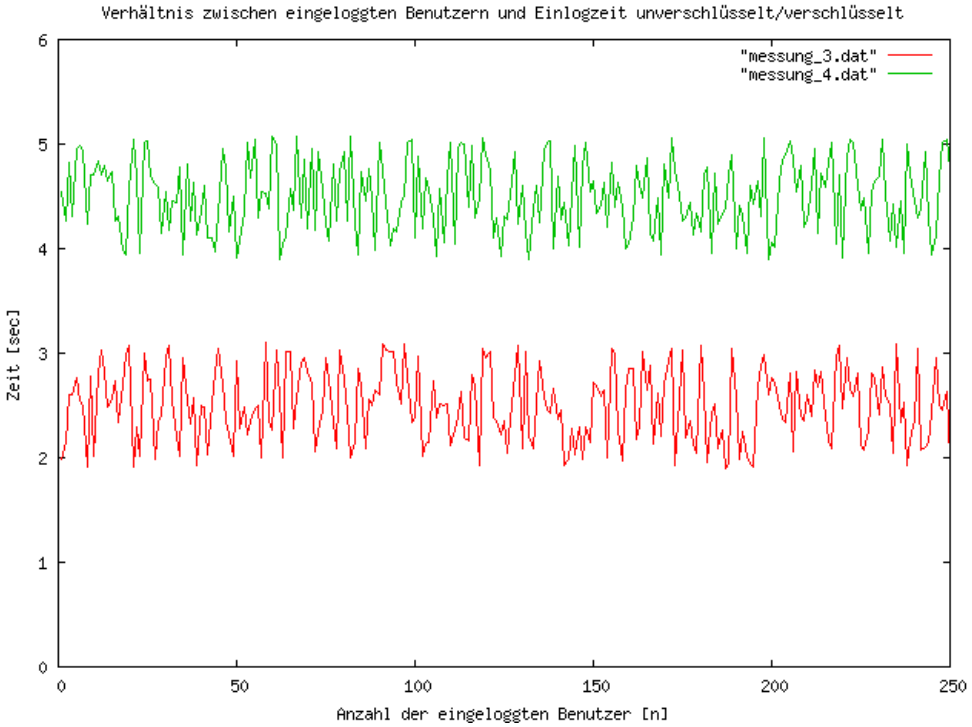

Abbildung 4.2: *Verhältnis zwischen eingeloggten Benutzern und Einlogzeit verschlüsselt/unverschlüsselt*

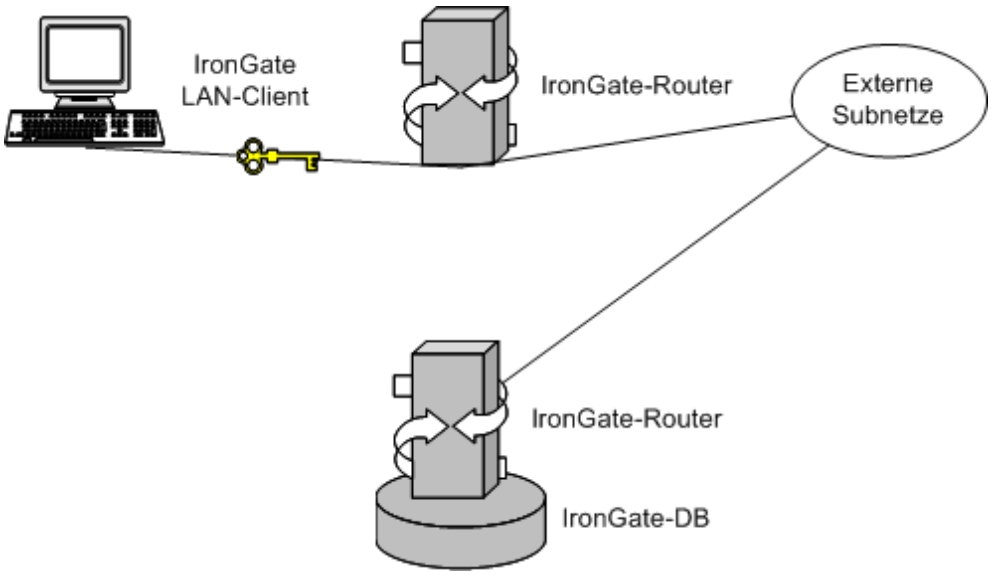

Abbildung 4.3: *Versuchsaufbau 2*

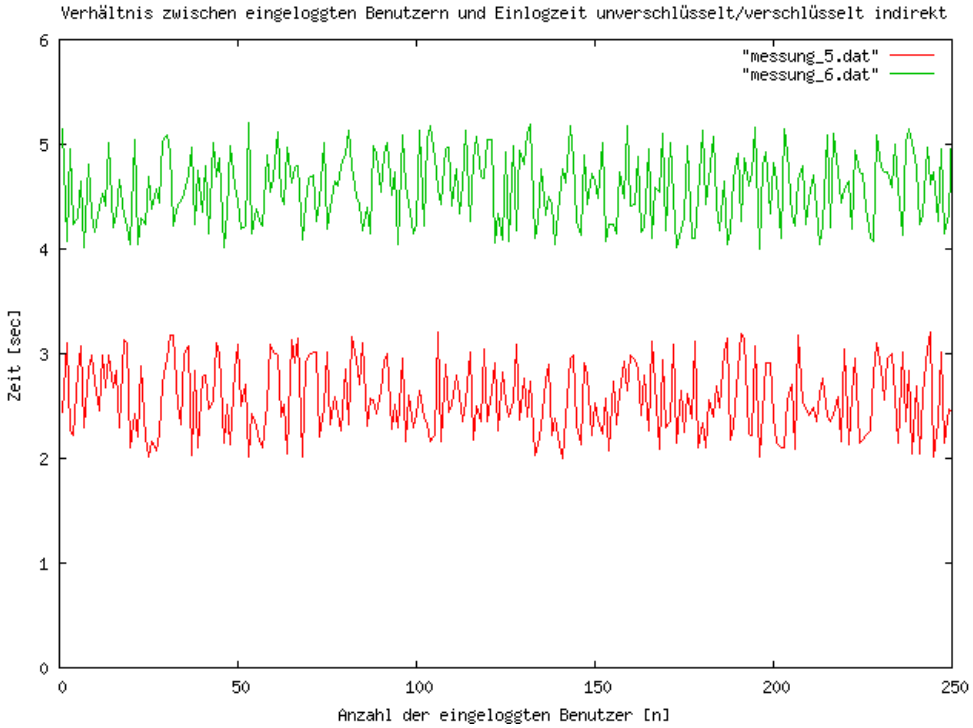

Abbildung 4.4: *Verhältnis zwischen eingeloggten Benutzern und Einlogzeit indirekt verschlüsselt/unverschlüsselt*

Bei den Tests der Reaktionszeiten wurde gemessen, wie schnell ein Login-Vorgang über einen IronGate-Server bis zur laufenden Session vor sich geht in Abhängigkeit von der Last des Servers in Form der Anzahl der Benutzer, welche gleichzeit auf dem Server eingeloggt sind.

Da IronGate bei jedem Login-Versuch die Werte des Authlog aufsummiert, wurde anhand des Versuchsaufbaues 1 auch überprüft, wie sehr die Anzahl der bereits erfolgten Logins Einfluss auf die Performance nimmt. Es stellte sich heraus, dass sich die Einlogzeiten nicht messbar ändern (siehe Abbildung 4.5, wenn mehr Werte aufsummiert werden müssen, woraus man schließen kann, dass es nicht notwendig ist, ein monatliches Accounting durchzuführen, wenn nicht gewünscht.

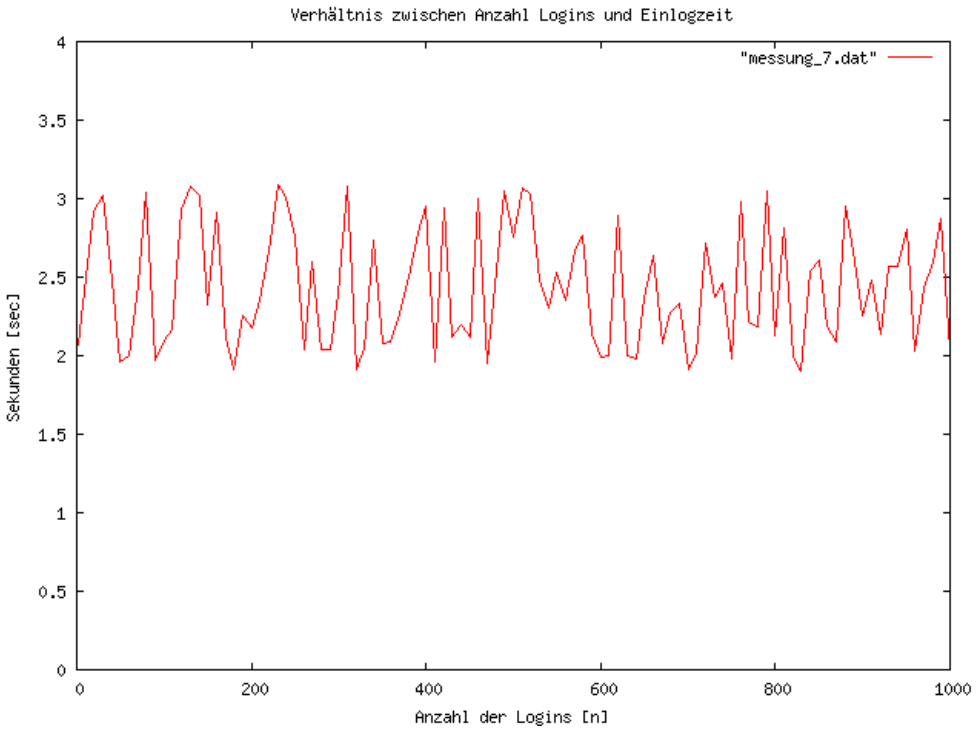

Abbildung 4.5: *Verhältnis zwischen Anzahl Logins und Einlogzeit*

4.2.2 Backend-Messungen

Bei den Backend-Messungen wurde ein Windows-2000 Server in die Testumgebung temporär integriert und auf dessen ADS zugegriffen. Es wurde getestet, um wieviel länger eine Authentifizierung bei der Verwendung eines Backends dauert als bei der alleinigen Verwendung der internen Datenbank. Hierfür wurde Versuchsaufbau 3 (Abbildung 4.6) als Messgrundlage verwendet.

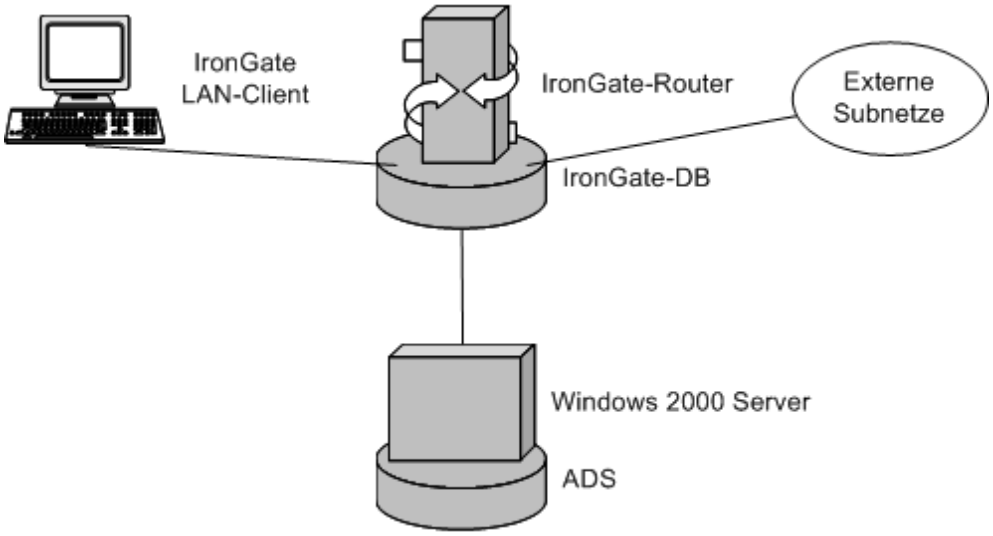

Abbildung 4.6: *Versuchsaufbau 3*

Das Ergebnis zeigt, dass quasi keinerlei Mehrzeit bei der Verwendung eines (ADS-)Backends auftritt, was in erster Linie auf eine sehr schnelle und schlanke Backend-Architektur und in zweiter Linie auf ein sauber und schnell programmiertes Interface auf Seiten des ADS-Servers schließen lässt.

4.2.3 WAN-Emulations-Messungen

Um das Verhalten von IronGate in WANs, stark überlasteten Netzwerken, bei schlechtem WLAN-Empfang usw. messen zu können, wurden mehrere Tests und Versuchsaufbauten mit WAN-Emulatoren durchgeführt. Das Verhalten in Client-Handover-Situationen zeigt das Folgekapitel, wohingegen Versuchsbau 4 und die dazugehörigen Messreihen das Verhalten mit nur einem IronGate-Router darstellen (vgl. Abbildung 4.8).

Auch hierbei wurde wieder unterschieden, ob eine verschlüsselte oder unverschlüsselte Verbindungsprozedur vorliegt. Der Unterschied der beiden Methoden verdeutlicht sich gerade hier, da bei einer verschlüsselten Verbindung weitaus mehr Pakete, welche bei schlechterer Verbindung eine höhere Laufzeit aufweisen, zwischen Client und Server ausgetauscht werden.

Die notwendige Zeit für den Login steigt hierbei linear an, wobei sie bei bspw. üblichen 50-100ms etwa eine Sekunde länger dauert, bei unüblichen aber möglichen 500ms bis zu 12 Sekunden (verschlüsselt) (vgl. Abbildung 4.9). Trotz der hohen Latenz arbeitet das Einlog-System jedoch zuverlässig - es gab keine Timeouts.

Die Situation, dass ein IronGate-Client die Verbindung zum Server in Form der Keep-Alive Signale verliert, tritt abhängig von den gesetzten Timeouts ein. Sendet der Client bspw. alle zwei Sekunden ein Keep-Alive-

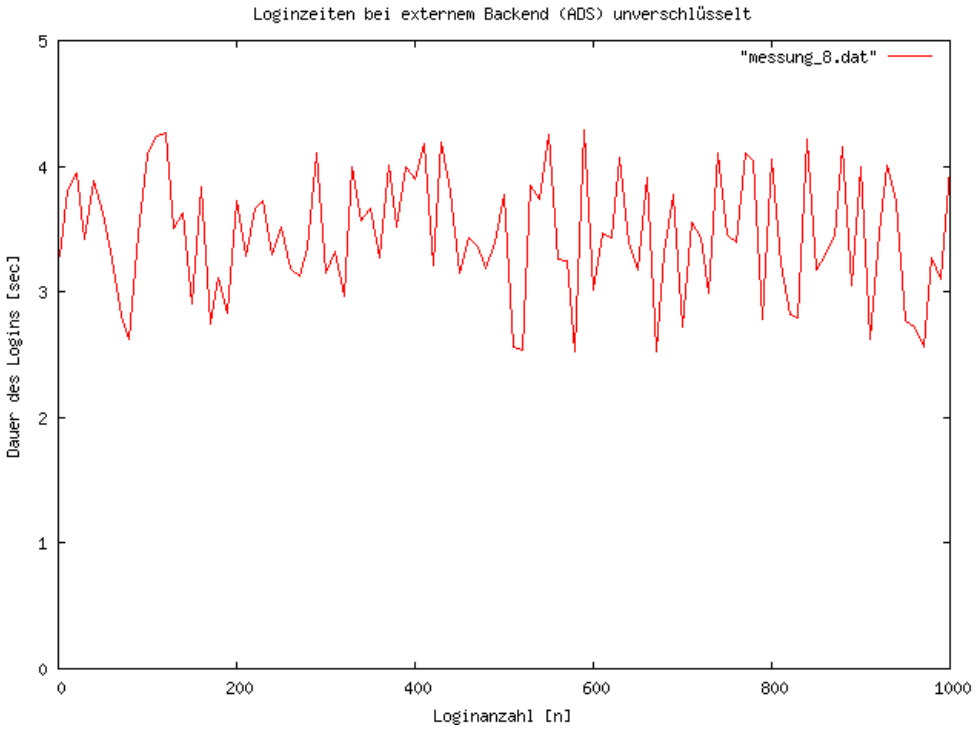

Abbildung 4.7: *Verhältnis zwischen Anzahl der Benutzer und Einlogzeit bei externem Backend*

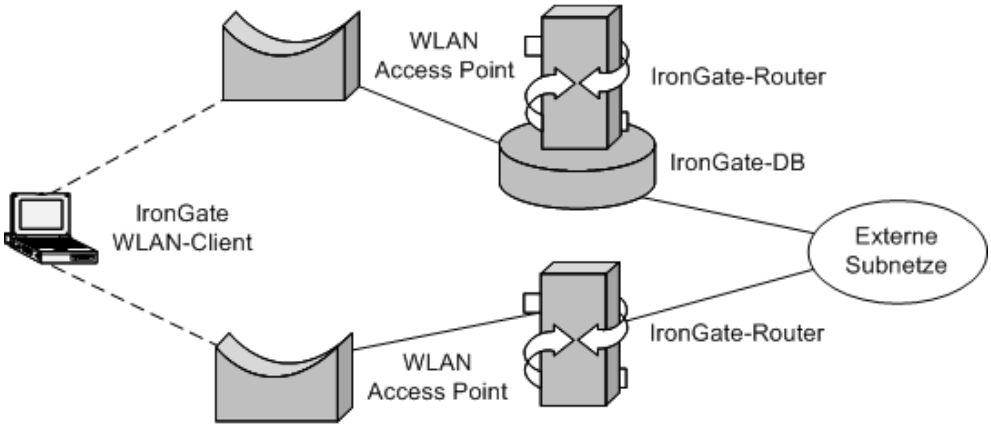

Abbildung 4.8: *Versuchsaufbau 5*

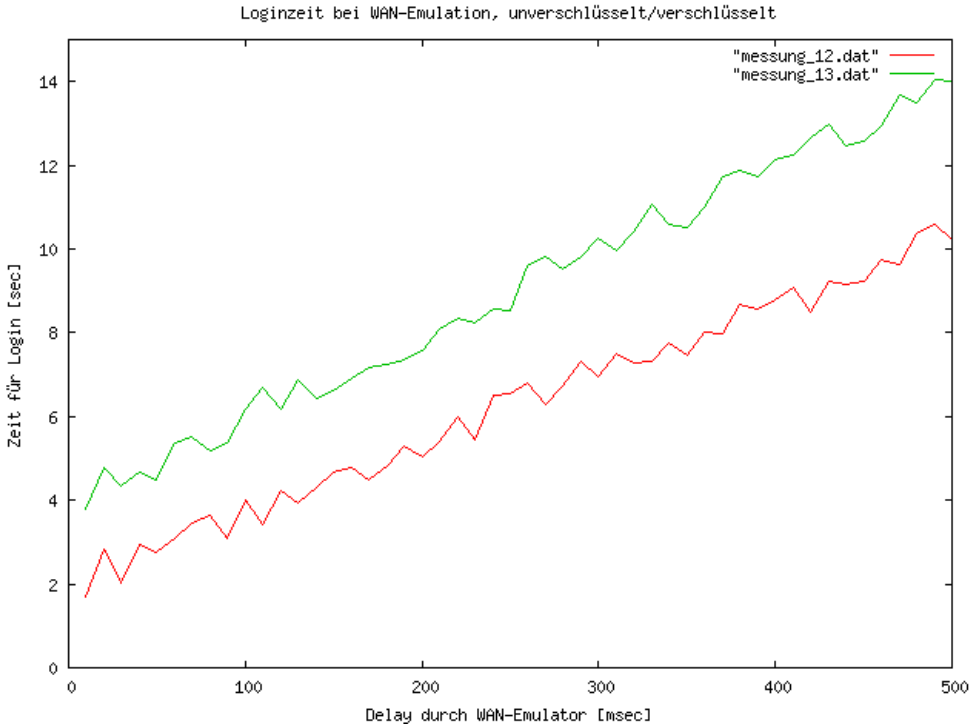

Abbildung 4.9: *Loginzeit bei WAN-Emulation verschlüsselt/unverschlüsselt*

Signal und der Server überprüft dessen Ankunft alle 4 Sekunden, so liegt die schlecht-möglichste Verbindung bei 2000ms Delay. Steigt die Verzögerung, wertet der Server den Client als ausgeloggt.

4.2.4 Handover-Messungen

Ein Schwerpunkt der Testreihen war das Verhalten des IronGate-Systems bei Client-Handover-Vorgängen, welche dann eintreten, wenn ein Client von einem Subnet in ein anderes wechselt.

Hierzu wurden zwei verschiedene Testumgebungen geschaffen, welche jeweils zwei IronGate-Router, die wiederum zwei Subnetze verwalten, sowie in dem einen Falle zwei WAN-Emulatoren für Messungen unter hoher Netzwerklast (Versuchsaufbau 6, Abbildung 4.11) und andererseits keine WAN-Emulatoren für den Test unter normalen Bedingungen bereitstellten (Versuchsaufbau 5, Abbildung 4.10).

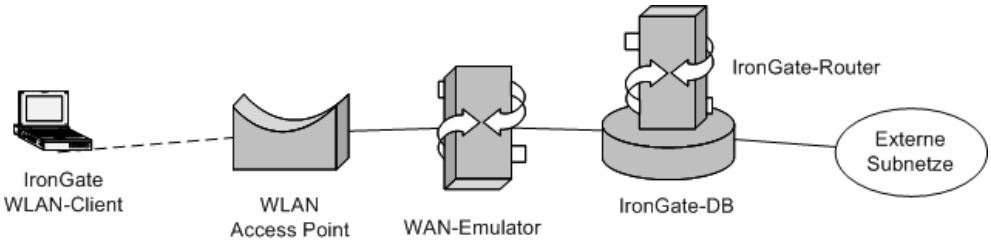

Abbildung 4.10: *Versuchsaufbau 5*

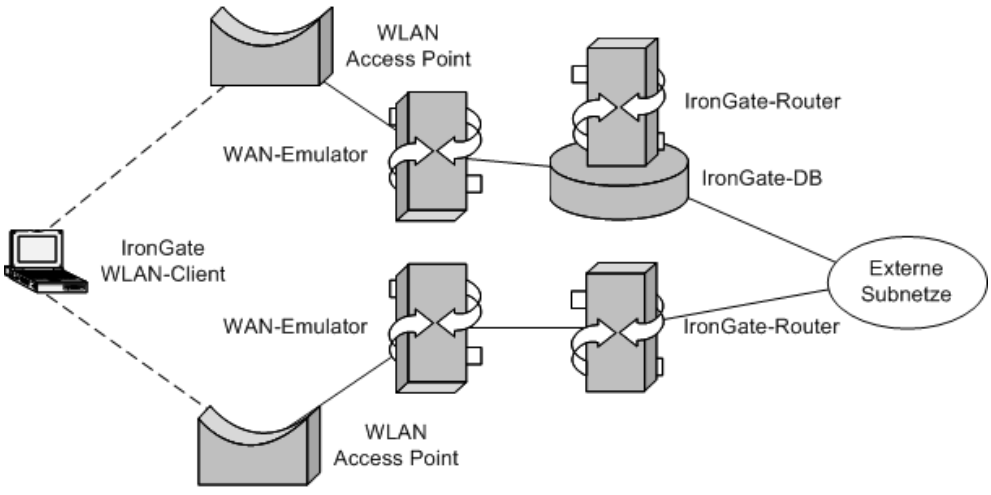

Abbildung 4.11: *Versuchsaufbau 6*

Die ersten Testreihen (Abbildungen 4.12 und 4.13) analysierten die Dauer des Re-Logins über einen längeren Zeitraum (100 Handover-Vorgänge)

um festzustellen, ob sich die Handover-Zeiten mit der Häufigkeit der Subnetzwechsel verändern (negativ oder positiv). Wiederum wurden einerseits verschlüsselte und andererseits unverschlüsselte Re-Login-Zeiten gemessen. Dabei stellte sich heraus, dass die Anzahl der Hops keinen Einfluss auf die Performance beim Subnetwechsel hat, unabhängig von der Art der Datenübetragung.

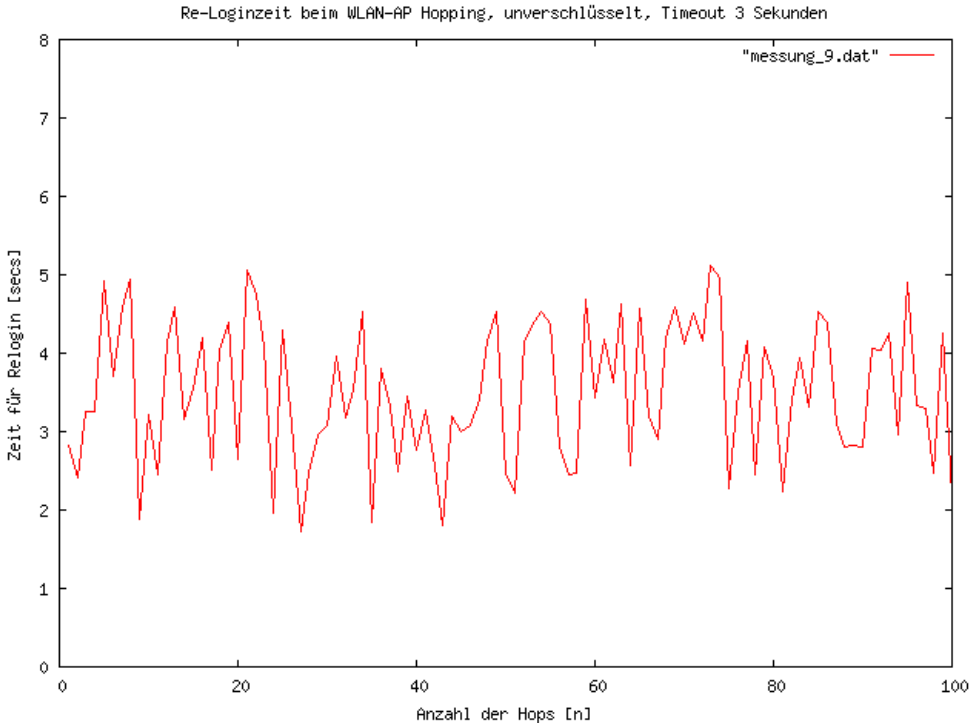

Abbildung 4.12: *Re-Loginzeit bei WLAN-Hopping unverschlüsselt*

Im nächsten Schritt wurde nachwievor in Versuchsaufbau 5 der Einfluss des Sessions-Timeouts auf die Handover-Zeiten gemessen, wobei sich erwarteterweise herausstellte, dass eine sehr geringe Timeout-Zeit einen sehr positiven Einfluss auf die Handover-Zeiten hat. Jedoch steigt die Last auf der Maschine bei mehreren gleichzeitigen Sessions bei Timeout-Zeiten kleiner einer Sekunde extrem an, was diese geringen Timeout-Zeiten nur theoretische möglich macht. Höhere Timeout-Zeiten bringen weniger Last sowie eine höhere Verbindungsstabilität mit sich, erhöhen jedoch ebensosehr auch die Trägheit der Handover-Vorgänge sowie die Session-Recovery-Geschwindigkeit.

Mit Versuchsaufbau 6 wurde nun gemessen, wie sich eine sehr hohe Netzwerklast (Annahme 300msec Latenz zum IronGate-Server) auf das Handover auswirkt (Abbildung 4.14). Auch bei diesen Messungen verhielt sich das System in etwa wie erwartet, indem ein Handover-Vorgang ca. soviel Zeit

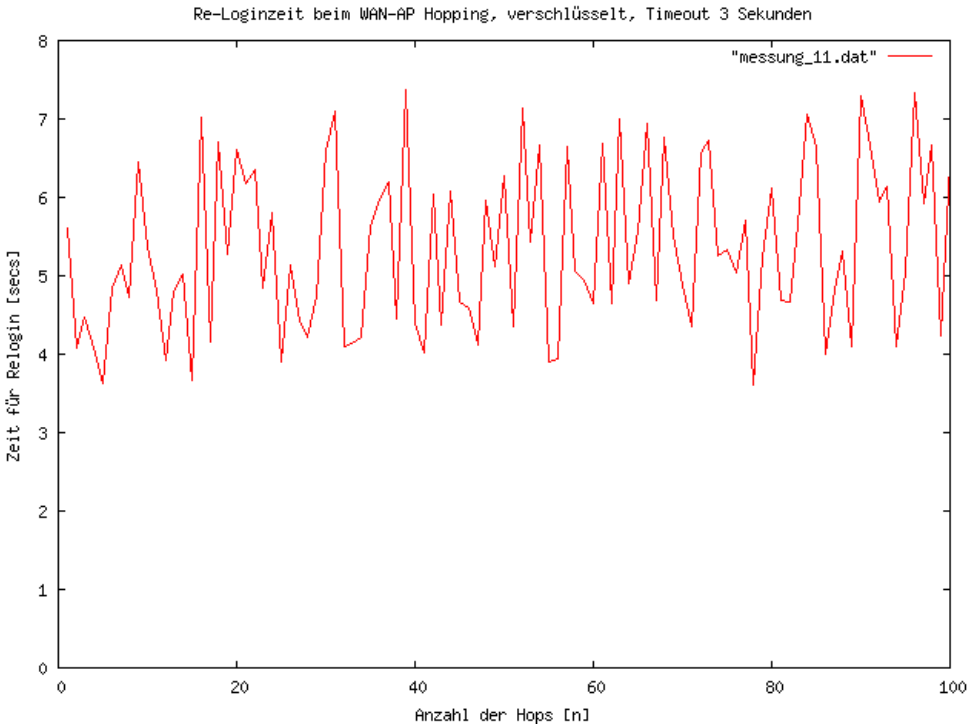

Abbildung 4.13: *Re-Loginzeit bei WLAN-Hopping verschlüsselt*

benötigte, wie ein normaler Login unter entsprechender simulierter Latenz plus der Zeit, die ein normaler verschlüsselter Handover-Vorgang einnimmt. Daraus lässt sich folgern, dass nicht nur normale Sessions, sondern auch Handover-Funktionen selbst unter hoher Last stabil sind. Dies ist im weiteren die Voraussetzung für den Echt-Betrieb in WLAN-Umgebungen, da in diesen ständig schwache Verbindungen herrschen, und zwar gerade in den Momenten, in denen in Handover fällig wird.

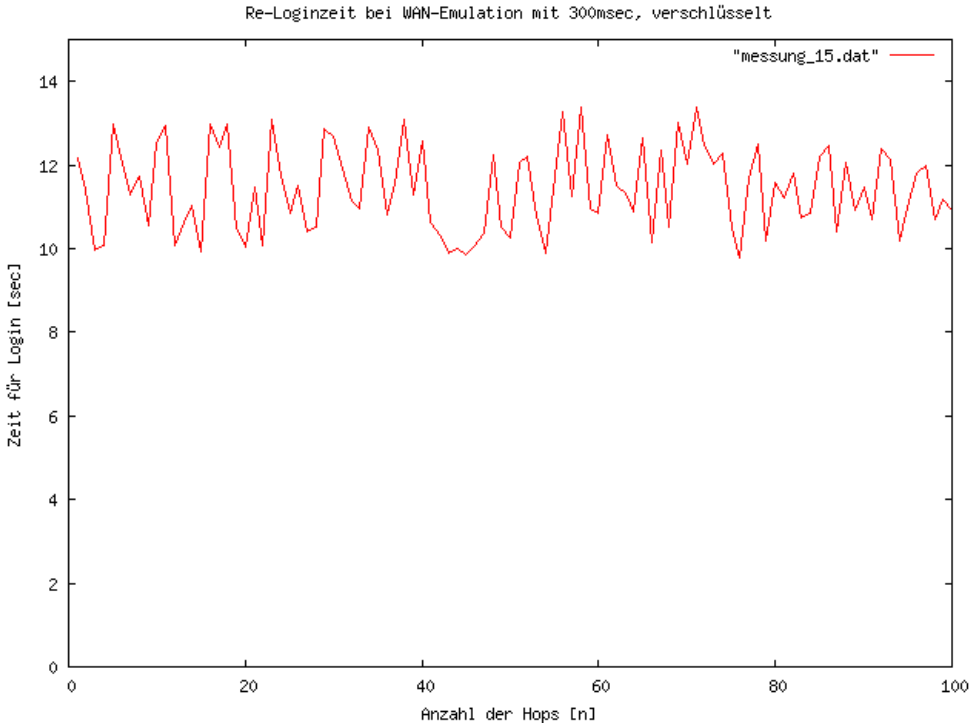

Abbildung 4.14: *Re-Loginzeit bei WLAN-Hopping mit WAN-Emulation verschlüsselt*

4.2.5 MIPL-Messungen

Um den Einfluss von IronGate auf Handover-Vorgänge mit MIPL zu messen, wurde außerhalb der IronGate-Subnets ein MIPL-Home-Agent installiert (Aufbau 7).

Zuerst wurden die IronGate-Router so konfiguriert, dass sie auch ohne Authentifikation routen. So wurden eine Reihe von MIPL-Handover-Messungen durchgeführt. Anschließend wurde IronGate gestartet, und die Messungen erneut durchgeführt. Bei aktiviertem IronGate jedoch musste der Client vorher über IPv4 das Handover von einem Subnet in das andere durchführen, bevor er sich erneut beim Home-Agent anmelden konnte. Dies

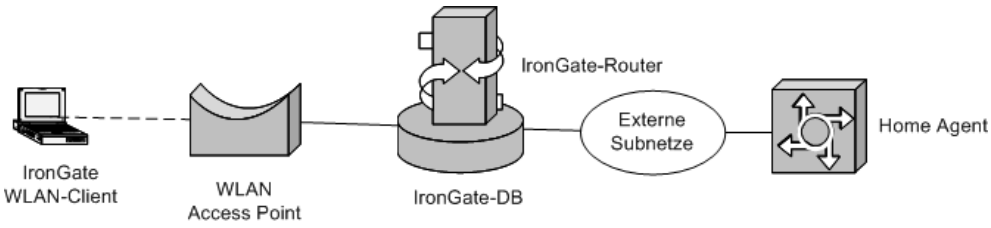

Abbildung 4.15: *Versuchsaufbau 7*

führte zu maximal 6 Sekunden Zeit, welche für ein MIPL-Roaming vennöten ist, im Gegensatz zu 1,5 Sekunden ohne IronGate (siehe Abbildung 4.16). Diese höheren Handover-Zeiten beim Einsatz von IronGate liegen in der jetzigen Architektur von IronGate, welche auf einer ganz anderen Ebene angesiedelt ist (vom Protokoll her) wie IPv6 Roaming-Mechanismen.

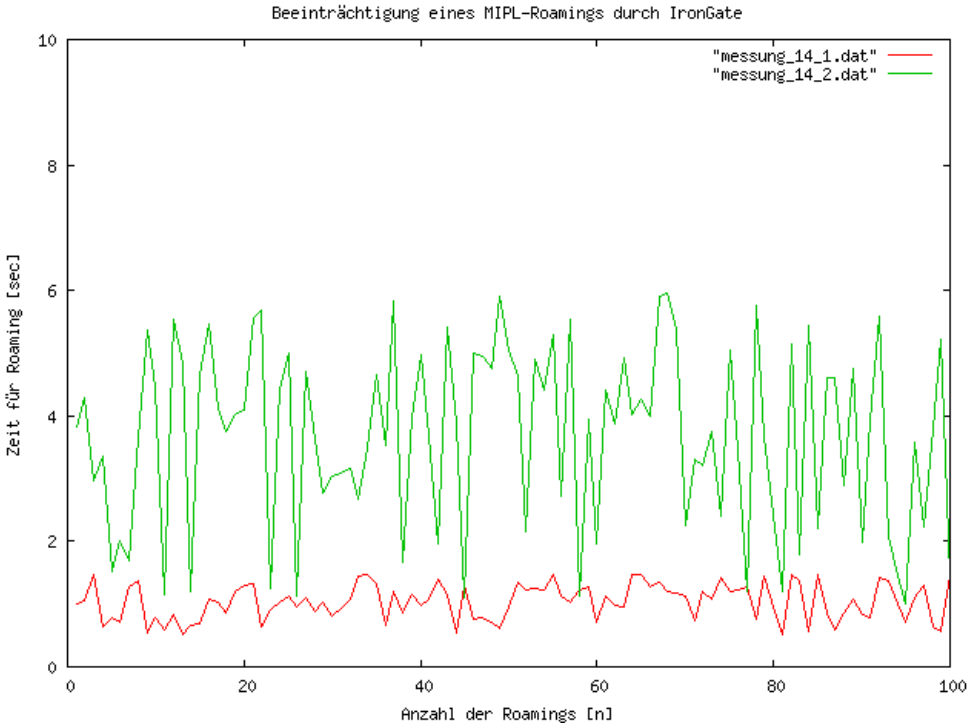

Abbildung 4.16: *Beeinträchtigung eines MIPL-Roamings durch IronGate*

Die hohe Varianz der Verzögerung beim Roaming ergibt sich durch das gesetzte Timeout. Es ist mehr oder weniger dem Zufäll überlassen, ob der Beginn eines Roaming kurz vor Ablauf des Keep-Alive-Timeouts einsetzt oder kurz nach einem Keep-Alive-Paket, welches den Timer zurücksetzt. So kann es bis zur Einleitung der Logout-Sequenz am einen IronGate-Router in

einem Fall beinahe 3 Sekunden (bei einem angenommenen Timeout von 3 Sekunden) dauern und in einem Fall nur wenige Millisekunden. So schwankten bei den Tests die durch IronGate verursachten Verzögerungen in einem Rahmen von etwa 3 Sekunden zuzüglich den in den früheren Messungen nachgewiesenen Schwankungen beim Login selbst (im Bereich von ca. 1,5 Sekunden).

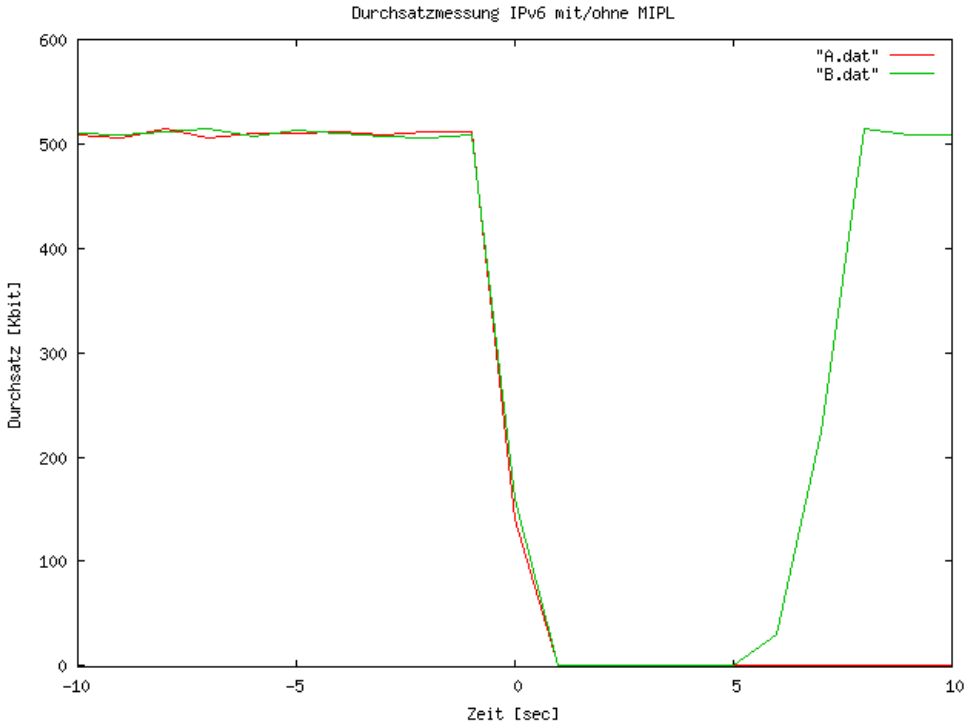

Abbildung 4.17: *Durchsatz bei Handover IPv6 mit/ohne MIPL*

Abbildung 4.17 zeigt das Verhalten von IronGate bei einem eingestellten Maximaldurchsatz von 512KBit/s und einem Handover zwischen zwei IronGate-Routern durch den Wechsel des Access Points. Messung A zeigt einfaches IPv6 ohne MIPL. Hierbei bricht der Durchsatz bei Beginn des Roamings ein der Transfer stoppt. Eine Wiederaufnahme des Transfers muss manuell eingeleitet werden. Messung B zeigt den Durchsatz bei Verwendung von IPv6 mit MIPL. Hierbei wird der Transfer nach dem erfolgten Handover vom Home-Agent korrekt umgeleitet und fortgesetzt.

4.3 Theoretische und praktische Limits

IronGate arbeitet sehr ressourcensparend durch das Teilen von Ressourcen beim Process-Forking, wodurch z.B. File- und Datenbankhandler, sofern global erstellt, nicht multipel angelegt werden. Daher kostet bei IronGate vor

allem die Kommunikation mit dem Client, den Backends sowie das Errechnen von Accounting-Informationen Ressourcen.

Theoretisch findet IronGate durch den Einsatz von Traffic-Shaping durch CBQ sein Limit. CBQ kennt nur maximal 65536 Klassen, was nur diese maximale Anzahl an Benutzern an einem IronGate-System erlaubt (wobei die Anzahl der daran beteiligten Maschinen keine Rolle spielt). Verzichtet man auf den Einsatz der Traffic-Control, so steigt die theoretische Maximalkapazität auf etwa 4 Milliarden an, sämtliche mögliche IPv4-Adressen. Das Maximum der Online-Zeit eines Benutzers ist durch den Wertebereich des „bigint"-Typs des mySQL-Datenbanksystems gesetzt, was jedoch die Lebensdauer einer Maschine überdauern dürfte. Das Maximum des Traffics einer Person in eine Richtung ist mit dem Byte-Counter der iptables verbunden, welcher als „longint" spezifiziert ist. Bei Überschreitung dieser beiden letztgenannten Werte versagt das Accounting, die Sitzung bleibt jedoch bestehen. Um ein regelmäßiges korrektes Accounting zu ermöglichen, beendet IronGate eine Session in jedem Falle nach einer Woche Online-Zeit bzw. einem Terabyte an übertragenen Daten. Die theoretische Maximalanzahl an verbundenen IronGate-Maschinen hängt ausschließlich mit der Leistung der zentralen mySQL-Datenbank und deren Anbindung zusammen, und ist theoretisch unbeschränkt.

Praktisch findet IronGate im Laufe der Tests bei einem Durchschnittsserver (1,5Ghz, 1GByte RAM) bei etwa 250 gleichzeitig zu verwaltenden Benutzern über eine Maschine sein Limit, welches durch die aufwendigen Accounting-Berechnungen und das stetige Refreshing des Online-Tables entsteht. Auch die stetig wachsende Zahl der iptables-Regeln führt bei einer derartigen Anzahl von Benutzer zu etwa 2500 Regeln, welche der Kernel verwalten muss, was den Prozessor belastet. Bei einer solchen Anzahl von Benutzers finden im Schnitt ca. 70 Zugriffe pro Sekunde auf die mySQL-Datenbank statt und etwa 3KByte/sec Traffic zwischen Client und Server (was nicht ins Gewicht fällt). CBQ kennt bei dieser Anzahl 500 Klassen, und auch das Shaping belastet den Kernel. Im Falle von IronGate wächst die Belastung linear mit den sich online befindenden Benutzern und die Verzögerung beim Login ebenfalls linear mit den bisherigen aufgezeichneten Sitzungen eines Benutzers. So müssen bei 1000 bisherigen Sitzungen auch 1000 Einträge vor dem Freischalten der Verbindung aufsummiert und ausgewertet werden, was viel Rechenzeit benötigt und (bei 1000 Einträgen) in etwa 200Kbyte an Datentransfer zwischen Authlog-Table und IronGate-Server erzeugt. Man kann in Folge sagen, dass eine Maschine mit doppelter Anzahl an Arbeitsspeicher und doppelter Rechengeschwindigkeit auch doppelt soviel Benutzer verwalten kann. Im praktischen Einsatz wird jedoch selten eine einzige IronGate-Maschine mehr als ein Class-C-Netz verwalten. Das Maximum an IronGate-Maschinen in einem Verbund liegt, wie auch das theoretische Maximum, an der Kapazität der mySQL-Datenbank. Wenn eine Datenbank 10.000 Schreib/Lesezugriffe pro Sekunde verarbeiten kann, und

eine IronGate-Maschine 100 S/L Zugriffe pro Sekunde erzeugt, so können problemlos 100 IronGate-Router aneinandergekoppelt sein. Wie die meisten praktischen Limits hängt auch dieser von den vorhandenen Ressourcen ab.

Der Packetlogger kennt keine theoretischen Limits außer denen der maximalen Anzahl an Einträgen, die mySQL in eine Tabelle erlaubt. Praktisch konnte eine Maschine mit 30 Millionen geloggten Einträgen (ca. 3Byte Daten) diese nach einigen Minuten Verarbeitungszeit noch auswerten. Es ist in jedem Falle zu empfehlen, in zeitgerechten Abständen den Packetlog zu bereinigen.

5 Praktischer Einsatz

5.1 Erfahrungswerte

Die bei den Tests und den seit 3 Jahren im Echtbetrieb des IronGate-Vorgänger MOSIAP (siehe [14] gesammelten Praxis-Werte sind in folgender Tabelle zusammengefasst, wobei sich diese auf ein System mit 1,5GHz CPU ohne Fremd-Backends und davon ausgehenden Hochrechnungen beziehen:

Maximum an IronGate-Maschinen im Verbund	100
Maximum an eingeloggten Benutzern pro Router	250
Maximum an eingeloggten Benutzern im Verbund	25.000
Maximum der eingtragenen Benutzer	unbegrenzt
Maximum des Traffics pro IronGate-Router	1GBit/sec
Maximum der Packetlog-Einträge	30.000.000
Einlog-Verzögerung	2-5 Sekunden
Roaming-Verzögerung	max. 8 Sekunden
CPU-Belastung Client	max. 3 Prozent
Prozess-Ausfälle pro Jahr	0
Watchdog-Einsätze pro Jahr	0

5.2 Migrationsvorgang

Ausgehend von einer aktuellen Linux-Distribution als Router sind folgende Einstellungen, Modifikationen und Packete vonnöten, um auf IronGate zu migrieren:

- Linux Kernel ab 2.6.3 mit Kernelpachtes (siehe Implementierung) und aktivierten Netfilter und Traffic-Shaping-Modulen (CBQ), sofern der Packetlog aktiv sein soll und das Traffic-Shaping eingesetzt wird

- Webserver mit den IronGate-Tutorien (welche auch die Clients beinhalten) als Standardseite muss eingerichtet sein

- mySQL ab Version 3.0 muss mit angelegten IronGate-Tabellen eingerichtet sein und den lokalen Zugriff erlauben

- GPG muss zur Verfügung stehen und das Schlüsselpaar erstellt, um verschlüsselte Verbindungen zu ermöglichen

- Bind Server ab Version 8.0 muss eingerichtet sein, sofern der Router als DNS fungiert und dynamische DNS-Updates erwünscht sind
- Die Pakete iproute2 und iptables müssen installiert sein, um Traffic-Shaping sowie Paketfilter konfigurieren zu können
- Die IronGate-Server-Sources müssen installiert sein und durch das Konfigurationsskript eingerichtet
- Das Routing zwischen mindestens zwei Interfaces muss aktiviert sein, wobei außer DNAT-Einträgen (Masquerading) keine Firewall-Regeln definiert sein dürfen

Eine Änderung der Client-Konfiguration oder deren Treiber muss nicht erfolgen. Software, welche bereits am Server läuft, kann ohne Einschränkungen weiterverwendet werden.

5.3 Bekannte Probleme und Lösungen

IronGate als Software selbst kennt nach eingehenden Tests keine Probleme oder Angriffsflächen, weder auf (Java)-Client-Seite noch auf Serverseite. Es gab jedoch Schwierigkeiten mit einer ausgelasteten Datenbank, woraufhin ein Prozess einfriert und als Zombie-Prozess endet, wogegen der Watchdog arbeitet, indem er den Benutzer zumindest wieder freischaltet. Auch hat IronGate das Problem, dass die Einlogzeit nach einer gewissen Zeit auch von der Anzahl der Accounting-Einträge im Authlog abhängt, und bei Extremwerten Zeitüberschreitungen seitens der Datenbank hervorrufen, was wiederum zu einem Prozess-Tod führt, und ein Einloggen unmöglich macht. Auch kann in so einem Falle der Watchdog nicht mehr helfen, da er von der Online-Datenbank abhängig ist, und diese „busy“ ist in Folge der Auslastung durch das Accounting. Dies trat im Praxiseinsatz nie auf, ist jedoch denkbar. Sehr wohl trat eine Überlastung des Servers durch einen überfüllten Packetlog ein, was das o.g. Problem hervorrufte. Es ist daher sehr wichtig, die stark dynamischen Tabellen bspw. von „cron“ regelmäßig einer Wartung zu unterziehen (Löschen veralteter Einträge bzw. Aufsummieren und Zusammenfassen selbiger etc.).

Darüber hinaus konnten beim Praxiseinsatz keine Fehler gefunden werden, die nicht bis zur momentanen Version behoben wurden.

6 Zusammenfassung

Wie in dieser Arbeit detailliert gezeigt, wurde mit IronGate ein sehr vielseitiges, flexibles und funktionales System entwickelt, welches für zahlreiche Probleme des Netzwerkmanagements eine effiziente und praktisch verwendbare Lösung darstellt. Hierbei wurden Protokolle und Techniken, welche traditionell strikt voneinander getrennt wurden, vereint und verknüpft integriert, was zu neuartigen Lösungswegen führte.

Probleme bei der Entwicklung ergaben sich vor allem durch den Versionsverlauf und der Herkunft von IronGate. Dadurch, dass das System ursprünglich nicht in dieser Größe geplant wurde, wurde das Erweitern teilweise erschwert. Auch war die Zielsetzung des zugrundeliegenden Systems kein Einsatz in einer mobilen Umgebung, weshalb teilweise eine sehr tiefgehende Neuentwicklung notwendig war. Diese Tatsachen führten auch zu der ungeplant längeren Entwicklungszeit. Auch die durch den Ursprung von IronGate eingesetzten prozeduralen Entwicklungsmethoden stießen hierbei an ihre Grenzen, auch bedingt durch den Einsatz einer nicht für Multi-Prozess-Anwendungen optimierten Sprache wie PERL. Für zukünftige Entwicklungen, bei denen Multithreading nicht ausgeschlossen ist, sollte eine Programmiersprache wie Java auch in Hinsicht der Modularität durch Objektorientierung vorgezogen werden.

Was einerseits ein großer Vorteil von IronGate ist, nämlich die Plattformunabhängigkeit durch die Trennung des Clients bzw. der Authentifizierungsmechanismen von der Treiberebene und der Implementierung auf einer höheren Schicht der Übertragung, ist gleichzeitig auch in Hinsicht auf die Mobilität ein Nachteil. So fällt die Geschwindigkeit beim Handover hinter die von betriebssystemspezifischen Implementierungen zurück, wofür systemsspezifische Treiber entwickelt werden müssten, was jedoch die plattformunabhängigen Pradigmen, unter welchen IronGate entwickelt wurde, verletzen würde. Das Ziel der Plattformunabhängigkeit, welches in jeder Hinsicht erreicht wurde, führt ausgleichend dazu, dass keine spezifische Mehrfachimplementierung notwendig ist und dadurch die Wartungskosten minimal sind.

Die der Entwicklung angeschlossenen Testläufe zeigten, dass IronGate ein sehr stabiles und belastbares System ist. Zusammen mit den Beobachtungen des zugrundeliegenden Systems im langjährigen Produktionsbetrieb kann behauptet werden, dass IronGate auch in jeder anderen Umgebung problemlos funktionieren würde. Abgesehen von den mobilen Gesichtspunkten ist die Performance und Effizienz sehr zufriedenstellend und in Anbetracht des ursprünglichen Einsatzbereiches erfüllen auch die Handover-Routinen die gestellten Anforderungen in jeglicher Hinsicht.

Literatur

[1] *„The GNU Privacy Guard“*, (http://www.gnupg.org)

[2] Matthew G. Marsh, *„Policy Routing Using Linux“*, March 2001

[3] Douglas E. Comer, *„Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture“*, Prentice Hall, 4th edition, January 2000

[4] Silvia Hagen, *„IPv6 Essentials“*, O'Reilly, July 2002

[5] Melissa Craft, *„Windows 2000 Active Directory“*, Syngress, October 2001

[6] Paul DuBois, *„MySQL, Second Edition“*, Sams, 2nd edition, January 2003

[7] Brian Arkills, *„LDAP Directories Explained: An Introduction and Analysis“*, Addison-Wesley Professional, 1st edition, February 2003

[8] Paul Albitz and Cricket Liu, *„DNS and BIND“*, O'Reilly, 4th edition , April 2001

[9] Michael Stahl, *„IronGate Java-Client Implementierung“*, Bakkalaureatsarbeit an der TU Wien, Institut für Breitbandkommunikation, January 2005

[10] Thomas Graf, Greg Maxwell, Remco van Mook, *„Linux Advanced Routing and Traffic Control“*, http://lartc.org

[11] Larry Wall, Tom Christiansen, Jon Orwant, *„Programming Perl“*, O'Reilly, 3rd edition, July 2000

[12] Bastian Preindl, *„IronGate Network Management“*, Bakkalaureatsarbeit an der TU Wien, Institut für Breitbandkommunikation, July 2003

[13] Gerard Beekmans, Matthew Burgess, *„LFS - Linux from Scratch“*, http://www.linuxfromscratch.org

[14] Bastian Preindl, *„MOSIAP - Multi Operating System Intranet Authentication Package“*, http://www.preindl.net/mosiap/ainacmosiap.ppt

Abbildungsverzeichnis

Glossar

LAN - Local Area Network
WLAN - Wireless LAN
WAN - Wide Area Network
MAN - Metropolitan Area Network
MOSIAP - Multi Operating System Intranet Authentification Package
SQL - Structured Query Language
AAA - Authentication, Authorization, Accounting
LDAP - Lightwigth Directory Access Protocol
ADS - Active Directory Service
TCP/IP - Transmission Control Protocol / Internet Protocol
MAC - Media Access Control
DHCP - Dynamic Host Configuration Protocol
ISP - Internet Service Provider
HF - High Frequency
NIS - Network Information Service
I/O - Input / Output
POSIX - Portable Operating System Interface based on uniX
UNIX - von UNICS, UNiplexed Information and Computing System
PIM - Personal Information Manager
OSI - Open System Interconnection
ISA - Internet Security Architecture
DNS - Domain Name Service
NDS - Novell Directory Service
TACACS - Terminal Access Controller Access Control System
PAM - Pluggable Authentication Modules.
QOS - Quality Of Service
HTML - HyperText Markup Language
RSA - Rivest, Shamir, Adleman (Erfinder)
GPG - GNU Privacy Guard
CBQ - Class Based Queuing
NetBIOS - Network Basic Input and Output System
PID - Process ID
IPC - InterProcess Communication
TERM - Termination, Terminal
HTTP - HyperText Transfer Protocol
SSL - Secure Socket Layer
HTTPS - HTTP Secured
DHTML - Dynamic HTML
PHP - PHP Hypertext Precompiler
CRLF - Carriage Return / Line Feed

OS - Operating System
DOS - Denial of Service
DB - DataBase
UDP - User Datagram Protocol
ICMP - Internet Control Message Protocol
NAI - Network Access Identifier
TOS - Type Of Service
ID - Identification
NAT - Network Address Translation
SNAT - Source NAT
DNAT - Destination NAT
ARP - Address Resolution Protocol
IPv6 - IP Protocol Version 6
PERL - Practical Extraction and Report Language
TBF - Token Bucket Filtering
RED - Random Early Detection
FIFO - First In First Out
WAP - Wireless Access Point
NIC - Network Interface Card/Connector
CPU - Central Processing Unit
MIPL - Mobile IPv6 for Linux
GNU - GNU is Not Unix
PGP - Pretty Good Privacy