FORSCHUNGSZENTRUM JÜLICH GmbH
Jülich Supercomputing Centre
D-52425 Jülich, Tel. (02461) 61-6402

Interner Bericht

Ein zuverlässiger und schneller Dateitransfer mit dynamischer Firewall-Konfiguration für Grid-Systeme

Thomas Oistrez

FZJ-JSC-IB-2008-02

März 2008

(letzte Änderung: 20.03.2008)

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Motivation eines neuen Dateitransfers für Grid-Umgebungen

Grid-Systeme wurden entwickelt, um Wissenschaftlern einen besseren und flexibleren Zugang zu den weltweit vorhandenen IT-Ressourcen zu verschaffen. Zur Lösung von Problemen in Wissenschaft und Forschung sind Computer heute unbedingt notwendig. Viele Entwicklungen sind nur durch Simulationen und Berechnungen auf Computersystemen möglich. Die dabei benötigte Rechenleistung steigt ständig an. Auch ein Supercomputer kann in vielen Fällen nicht alle Anforderungen eines Forschungsprojektes erfüllen. Es entstand daher bei Wissenschaftlern und Ingenieuren der Wunsch, weltweit vorhandene und auf viele Organisationen aufgeteilte Rechenleistung gemeinsam nutzen zu können. Die Grid-Technologie wurde entwickelt, um dies möglich zu machen.

1.1 Was ist ein Grid?

Ein Grid ist ein verteiltes System, welches den Nutzern verschiedene Ressourcen zugänglich macht, auch wenn diese auf mehrere, administrativ unabhängige Einrichtungen an unterschiedlichen Standorten verteilt sind. Diese Ressourcen sind Rechenleistung, Speicherkapazität und Daten, welche für ganze Arbeitsgruppen in einem Grid verfügbar gemacht und untereinander ausgetauscht werden können.

Die Rechenleistung der im Grid zusammengefassten Rechnersysteme wird allen Nutzern zugänglich gemacht, so dass deren anstehende Berechnungen immer auf dem System ausgeführt werden, welches zur Zeit die größten Kapazitäten frei hat oder in speziellen Fällen am Besten dafür geeignet ist. Kriterien für die Auswahl eines Systems sind neben der Rechenleistung und der Größe des Hauptspeichers z.B. das Vorhandensein spezieller Bibliotheken und benötigter Software sowie die Bandbreite und Qualität der Netzwerkverbindung zu diesem System. So erreicht man im Idealfall eine sehr gute Auslastung der vorhandenen Kapazitäten und eine schnellstmögliche Bearbeitung der anstehenden Aufgaben.

Bei wissenschaftlich-technischen Modellberechnungen fallen große Datenmengen an, die auf allen an der Berechnung beteiligten Systeme zur Verfügung stehen müssen, so dass eine große Menge an persistentem Speicher notwendig wäre. In einem Grid werden Speicher aber ebenso flexibel und ortsunabhängig genutzt wie die integrierten Rechner. Als Teil des Gesamtsystems ist jeder persistente Speicher von allen Systemen und für alle Nutzer zugänglich. Diese einzelnen Speicherkapazitäten können z.B. in Form von ganzen RAID-Systemen und Magnetspeichern in das Grid mit eingebracht werden. So kann die insgesamt zur Verfügung stehende Speicherkapazität flexibel und ortsunabhängig zwischen den Nutzern aufgeteilt werden, abhängig davon, wie groß die einzelnen Anforderungen der gerade laufenden Jobs sind.

Zudem ist ein einfaches Austauschen von Daten und berechneten Ergebnissen sinnvoll, um die Zusammenarbeit der weltweit verteilten Organisationen und Arbeitsgruppen zu verbessern. In einem sogenannten DataGrid können daher Daten, die bei Berechnungen im Grid angefallen sind, zwischen den Nutzern geteilt und öffentlich zugänglich gemacht werden. Dieses Prinzip wird oft mit den oben beschriebenen Konzepten kombiniert.

Die wichtigsten Eigenschaften, die ein Grid-System haben muss, sind demnach die Ortstransparenz sowie die Plattform- und Systemunabhängigkeit. Ortstransparenz bedeutet für ein Grid-System, dass es für den Benutzer keine Rolle spielt, wo sich ein bestimmtes Gerät befindet. Das Gleiche gilt auch für die Speichersysteme, auf die immer auf die gleiche Weise, unabhängig von ihrem weltweiten Standort, schreibend sowie lesend zugegriffen werden kann. Ein Rechner oder Speicher im Nachbarraum sollte genau so angesprochen werden können wie ein Rechner am anderen Ende der Welt. Auch die Plattformunabhängigkeit ist wichtig, damit

jedes zur Verfügung stehende System in das Grid integriert werden kann, unabhängig von dessen Hardware-Architektur und dem verwendeten Betriebssystem. Vom Arbeitsplatzrechner bis zum Supercomputer müssen alle Systeme gleich benutzbar sein.

Eine weitere Besonderheit eines Grids ist die dynamische Bereitstellung von Ressourcen. So können jederzeit einzelne Systeme in das Grid integriert oder entfernt werden, um das Gesamtsystem auf einem aktuellen, leistungsfähigen Stand zu halten. Durch diese Möglichkeit können neu erworbene Komponenten einfach und schnell in ein bestehendes Grid integriert werden. Damit ändern sich Anzahl und Standort der einzelnen Komponenten im Grid ständig.

Gängige Grid-Systeme nutzen daher sogenannte Middleware. Dies sind Softwarekomponenten, welche die oben genannten Eigenschaften implementieren. Eine Middleware basiert auf der jeweiligen Hard- und Software-Architektur und versteckt deren spezielle Eigenschaften vor den Nutzern und anderen Systemen. So kann auf alle Systeme auf die gleiche Weise zugegriffen werden. Die Middleware übernimmt auch die Kommunikation zwischen den Systemen, so dass der Nutzer keine Informationen über Standort und Art der einzelnen Komponenten des Grids haben muss. Außerdem werden Authentifizierung und Autorisierung von der Grid-Middleware übernommen.

Für Forschungsprojekte, an denen mehrere Organisationen beteiligt sind, ist es üblich sogenannte virtuelle Organisationen zu bilden. Das bedeutet, dass mehrere reale Organisationen (Unternehmen und Forschungseinrichtungen) bzw. einzelne Arbeitsgruppen dieser Organisationen sich zu einer einzigen virtuellen Organisation (VO) zusammenschließen. Die verschiedenen Mitglieder einer VO können sich an weltweit verteilten Standorten befinden und aus völlig verschiedenen Einrichtungen stammen. Einige reale Organisationen stellen Ressourcen für das Grid zur Verfügung, während andere keine eigenen Ressourcen haben, sondern nur als Nutzer auftreten. Dabei kann ein Unternehmen komplett der VO beitreten oder es nehmen nur eine Arbeitsgruppe oder einzelne Personen aus einer realen Organisation teil. Mit einer Grid-Middleware kann eine solche VO technisch realisiert werden. Eine VO hat durch zentrales Authentifizieren eine eigene, unabhängige Zugriffs- und Sicherheitspolitik, welche von den Richtlinien der realen Organisationen abweichen kann. Die Rolle einer Person in der VO muss nicht der Rolle in ihrer realen Organisation entsprechen. Abbildung 1.1 zeigt den Aufbau solcher VOs. Weitere allgemeine Informationen zu Grids bietet [1].

1.2 Anforderungen von Grid-Systemen an die Netz-Infrastruktur

Ein Grid-System, wie es im vorherigen Abschnitt beschrieben wurde, hat einige besondere Eigenschaften und Anforderungen an die zugrunde liegenden Netzwerke. Die meisten Anwendungen arbeiten sowohl mit großen Datenmengen, als auch mit vielen kleinen Dateien. Zuverlässige und stabile Verbindungen sind dazu ebenso wichtig wie hohe Transferraten, um ein effizientes Arbeiten zu ermöglichen. Ein einzelnes langsames Teilnetz kann schnell zu einem Flaschenhals für das gesamte Grid werden. Im Extremfall überwiegt die benötigte Zeit für Datentransfers gegenüber der eigentlichen Rechenzeit, was dem Gedanken der hohen Ressourcen-Auslastung und des effizienten Arbeitens widerspricht. Außer der benötigten Bandbreite spielt auch die Ortstransparenz eine wichtige Rolle für die Netzwerke. Der Datenaustausch erfolgt sowohl zwischen Client und Servern, als auch zwischen zwei Servern oder z.B. einem Storage-Area-Network (SAN). Die einzelnen Systeme sind dabei nicht unbedingt in der selben realen Organisation angesiedelt, sondern über die ganze VO verteilt. Datenver-

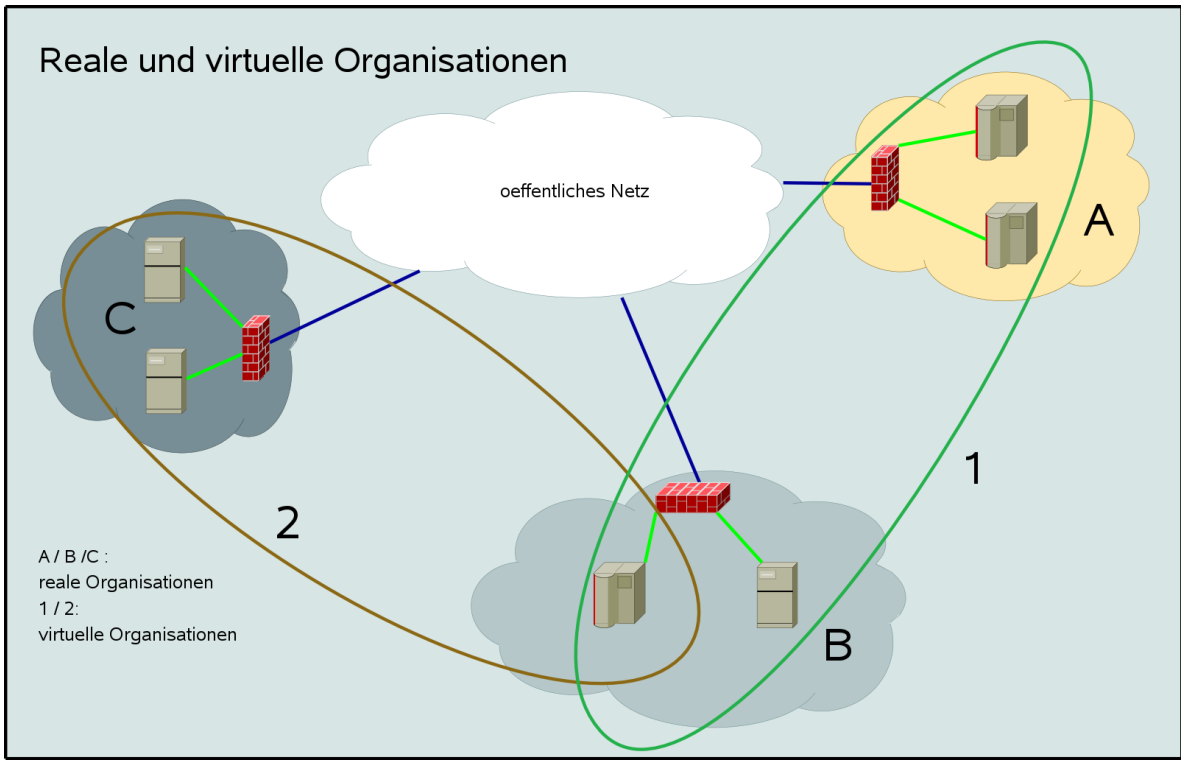

Abbildung 1.1: Reale und virtuelle Organisationen

bindungen müssen also über mehrere autonome Netze hinweg aufgebaut werden.

Besonders stark beeinflussen sich der Betrieb eines Grids und die Sicherheitsrichtlinien bei den einzelnen realen Organisationen. Unternehmen und auch Forschungseinrichtungen haben genaue Sicherheitsanforderungen an ihre Netzwerke. Ein unsicheres Netzwerk bedeutet eine ernsthafte Bedrohung für die ganze Organisation, weil z.B. Angreifer die Funktionen des gesamten Netzes beeinträchtigen oder geheime Daten stehlen können. Die besonderen Anforderungen eines Grids an die Erreichbarkeit der einzelnen Komponenten widerspricht diesem Sicherheitsgedanken der Unternehmen. Jede Komponente des Grids muss von allen anderen Komponenten aus erreichbar sein, die größtenteils außerhalb des eigenen Netzwerks liegen. Zwar besteht zwischen den einzelnen Teilnehmern eines Grids normalerweise ein Vertrauensverhältnis, welches den gegenseitigen Zugriff auf Ressourcen erlaubt, doch die entsprechende Konfiguration der Netzwerk-Komponenten ist schwierig. Zudem können in einem Grid transitive Vertrauensketten zwischen mehreren realen Organisationen aufgebaut werden. Das bedeutet, dass einem Unternehmen Zugriff und Nutzungsrechte auf Ressourcen eines anderen Untermehens eingeräumt werden, welches wiederum ein Vertrauensverhältniss zu einer dritten Organisation unterhält. So entstehen Zugriffsstrukturen, die dann aufgrund ihrer Komplexität nicht mehr sinnvoll auf Zugangsregeln abgebildet werden können. Es sollten im Idealfall nur die fremden Rechner Zugriff auf das eigene Netz erhalten, die zum Grid gehören. Und auch diese dürfen nicht beliebige Verbindungen in das eigene Netz aufbauen, sondern nur zu den Systemen, welche Teil des Grids sind. Zudem ändern sich die Gegebenheiten mit jedem Rechner, der neu in das Grid eingefügt oder daraus entfernt wird.

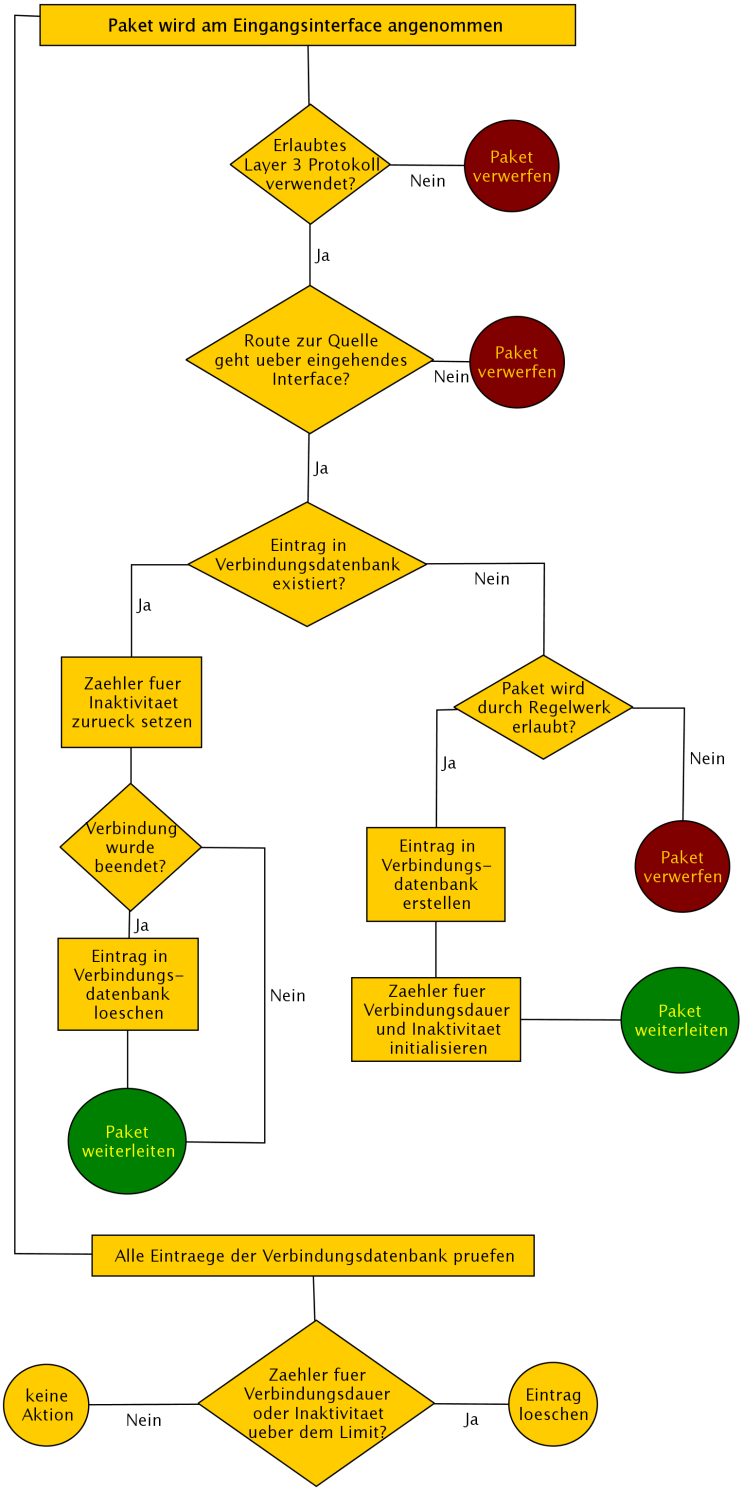

Abbildung 1.2: Funktionsweise einer Firewall

Die konkrete technische Umsetzung der Zugangsregeln findet in den Firewalls der Organisationen statt. Eine Firewall ist eine Netzwerk-Komponente, welche sämtlichen Datenverkehr zwischen dem lokalen Netz und der Außenwelt kontrolliert. Dabei wird nur der Datenverkehr durchgelassen, der laut Sicherheitsrichtlinie erlaubt ist. Zur Kontrolle der Datenpakete stehen verschiedene Tests zur Verfügung. Es kann z.B. anhand des verwendeten Layer-3-Protokolls und mit Hilfe der eigenen Routing-Tabellen nach ungültigen oder gefälschten Paketen gesucht werden. Die meisten Entscheidungen werden jedoch durch das vom Administrator konfigurierte Regelwerk, sowie bei Stateful-Packet-Filtern durch die Verbindungsdatenbank der Firewall getroffen. Die Abläufe in einer Firewall sind in Abbildung 1.2 grob dargestellt. Pakete, die zu bestehenden Verbindungen gehören, werden weitergeleitet. Bei den restlichen Paketen wird das Regelwerk daraufhin überprüft, ob eine neue Verbindung erlaubt wird. Das Regelwerk beschreibt einzelne Verkehrsklassen, die bei allen gängigen Firewalls durch die IP-Adressen und Ports der beteiligten Kommunikationspartner sowie der verwendeten Protokolle beschrieben werden. Für jede Kombination aus einem internen und einem externen Rechner muss also mindestens eine solche Regel existieren. Gerade in Grids entsteht so ein großes und kompliziertes Regelwerk. Manche der dort eingesetzten Protokolle, wie zum Beispiel GridFTP, verwenden zudem mehrere Ports gleichzeitig, wodurch im Regelwerk größere Port-Bereiche freigegeben werden müssen. Diese Art der Konfiguration ist nicht nur aufwändig und fehleranfällig, sie kann unmöglich werden, wenn die technischen Grenzen der Firewall erreicht werden. So ist der verfügbare Speicher für das Regelwerk und sonstige Konfigurationen beschränkt. Gängige Limits liegen bei 2 MB. Davon bleiben für das Regelwerk in günstigen Fällen noch 1,5 MB, was in etwa 20000 einzelnen Regeln entspricht. Dies ist ein Wert, der durchaus erreicht werden kann, wenn die Firewall ein größeres Firmennetz vom Internet trennen soll. In der Praxis werden daher oft viele Regeln zu einer einzelnen groben Regel zusammengefasst. Es werden zum Beispiel nicht einzelne Rechner eingetragen, sondern ganze Subnetze und große Port-Bereiche. Dies vereinfacht die Arbeit der Netzwerk-Administratoren erheblich und es verkleinert das Regelwerk, was auch der Performance der Firewall zugute kommt. Es werden aber gleichzeitig unnötige Angriffsflächen und Gefahren geschaffen, denn es sind mehr Rechner und Ports erreichbar als nötig. Je nachdem wie groß das Sicherheitsbedürfnis der jeweiligen Organisation ist, ist das ein inakzeptabler Zustand.

Eine Alternative zu direkten Verbindungen zwischen den zwei Kommunikationspartnern stellt das Tunneln der Verbindungen da. Dabei werden die Verbindungen zwischen zwei Endsystemen nicht direkt hergestellt, sondern z.B. durch einen http- oder ssh-Tunnel geleitet. Die realen Organisationen müssen dazu Tunnel-Server betreiben, die die verschiedenen Verbindungen ähnlich einem Proxi-Server entgegen nehmen und durch eine einzige Verbindung zur anderen Organisation leiten. Die Konzentration der Verbindungen auf einen zentralen Rechner sowie das Multiplexen und optionale Verschlüsseln der Daten machen dieses Verfahren langsamer als direkte Verbindungen. Zudem erzeugt man mit dem Tunnel-Server einen Single-Point-of-Failure und man bekommt Probleme mit eventuell eingesetzen Content-Filtern in den Firewalls. Diese erkennen, dass das eingesetzte Anwendungsprotokoll (z.B. http) keine zum Protokoll passenden Daten transportiert und verwerfen die Datenpakete. Tunnel-Techniken sind daher weniger flexibel und aufwändiger zu realisieren als direkte Verbindungen.

Insgesamt zeigt sich also, daß ein Grid kaum mit hohen Sicherheitsanforderungen vereinbar ist. Auf der einen Seite benötigt ein Grid eine Möglichkeit, um Daten schnell zwischen beliebigen Komponenten auszutauschen. Andererseits sollen nur wirklich benötigte Verbindungen erlaubt werden. Es sollten durch die verteilte Struktur des Grids keine zusätzlichen Gefahren beim Betrieb der lokalen Netze der einzelnen Teilnehmer entstehen. Mit den traditionellen

Verfahren zur Firewall-Konfiguration ist dieser Zustand nicht oder nur unter sehr hohem Aufwand erreichbar. Eine Lösung für dieses Problem wäre ein Verfahren, welches eine einfache, unkomplizierte und immer aktuelle Konfiguration der Firewalls erlaubt, ohne diese dabei weiter zu öffnen als unbedingt nötig. Im Wesentlichen ergeben sich dafür folgende grundsätzliche Anforderungen, um eine leistungsfähige und praxistaugliche Lösung zu schaffen:

1. Die Lösung muss sich nahtlos in bestehende Sicherheitskonzepte integrieren lassen, damit das bestehende Sicherheitsniveau nicht gefährdet wird und dessen Komplexität nicht unnötig ansteigt.

2. Die Lösung muss in jeder Art von Software einsetzbar sein, unabhängig davon, ob es sich um kommerzielle Produkte oder Open-Source handelt. Es dürfen also nur standardisierte und für jeden nutzbare Technologien benutzt werden.

3. Die erlaubten Zugriffe von außen müssen nicht nur örtlich, sondern auch zeitlich beschränkt sein, so dass Verbindungen nur bei wirklichem Bedarf möglich sind. Nur so kann jede unnötige Gefahr vermieden werden.

1.3 Dynamische Konfiguration von Firewalls

Wie der letzte Abschnitt bereits gezeigt hat, birgt ein statisches, manuell gepflegtes Regelwerk für Firewalls einige Probleme und Gefahren beim Betrieb eines Grids. Es sollte eine Möglichkeit gefunden werden, die größten Nachteile dieser Vorgehensweise zu umgehen. Das bedeutet, dass die Regeln ständig an den gerade vorherrschenden Zustand angepasst werden müssten. Das Regelwerk sollte also dynamisch und immer aktuell sein. Die ständigen Änderungen könnten nicht von einem Administrator gemacht werden, sondern sollten automatisiert ausgeführt werden. Die Aufgabe, die Firewall dynamisch zu konfigurieren, müsste daher von den Rechnern im eigenen Netz übernommen werden, denn grundsätzlich wissen die einzelnen Komponenten in einem Grid am Besten, wann sie eine Datenverbindung benötigen, und zu welchem anderen Rechner diese aufgebaut werden soll. Es liegt nahe, dass die Rechner im eigenen Netz der Firewall mitteilen, wann eine Verbindung von außen erwartet wird. Dieses Vorgehen würde die Sicherheit des lokalen Netzes nicht wesentlich gefährden, da die eigenen Rechner als vertrauenswürdig gelten. Zudem könnte die Möglichkeit einer solchen Konfiguration auf bestimmte Rechner beschränkt werden. Wichtig ist dabei, dass der eigene Rechner genaue Angaben über die Grid-Komponente machen kann, von der eine Verbindung erwartet wird. Sonst könnte eine angekündigte Verbindung von einem dritten missbraucht werden. Beide Seiten der Datenverbindung müssten also eindeutig identifiziert werden, im Normalfall durch ihre IP-Adresse und den genutzten Port. Die Firewall könnte dann eine Regel mit zeitlich begrenzter Lebensdauer erstellen, die genau auf die erwartete Verbindung abgestimmt wäre.

Ein Beispiel für eine erste Realisierung dieses dynamischen Konzepts ist das „Cooperative On Demand Opening“ (CODO) [3]. Diese zweiteilige Programmbibliothek ersetzt auf Client-Seite die normale Berkeley-Socket-Library durch eine eigene Implementierung. Diese erweiterte Version der Sockets kommuniziert beim Aufbau einer Verbindung erst mit einem Firewall-Agenten, dem zweiten Teil der Bibliothek. Dieser nimmt die Verbindungsdaten in Form von IP-Adressen und Ports entgegen und konfiguriert die eigene Firewall dynamisch. Außerdem leitet er die Informationen an den Firewall-Agenten des Ziel-Rechners weiter,

welcher ebenfalls eine entsprechende Konfiguration an dessen Firewall vornimmt. Die Kommunikation zwischen dem CODO-Klienten und den Firewall-Agenten geschieht verschlüsselt mittels X.509-Zertifikaten. CODO bietet zwar eine dynamische Konfiguration, wie sie gefordert wurde, hat aber auch einige Nachteile. So unterstützt der Firewall-Agent zur Zeit nur Netfilter-Firewalls (iptables). Er ist also nicht in kommerzielle Lösungen integrierbar. Von kommerziellen Herstellern sind bisher noch keine eigenen, marktreifen Implementierungen eines Firewall-Agenten für ihre Produkte vorgestellt worden. Selbst wenn solche Produkte verfügbar werden, wird es einige Zeit dauern, bis diese eine ausreichende Verbreitung gefunden haben. Außerdem macht die Verwendung eines solchen Agenten die Firewall selbst unsicher. Es ist eine wichtige Eigenschaft aller Firewalls, dass sie gegen Angriffe immun sind. Mit dem Firewall-Agenten bietet die Firewall aber einen eigenen Dienst an, auch wenn er sich auf einer dedizierten Machine befindet. Dies macht die Firewall angreifbar, weil sie sich bei einem erfolgreichen Angriff auf den Firewall-Agenten manipulieren lässt.

Es sind also von Seiten der Hardware-Hersteller bisher keine praxistauglichen und weitreichend verbreiteten Lösungen lieferbar, die alle nötigen Funktionen für sichere und effiziente dynamische Firewall-Konfiguration mitbringen. Wenn man eine dynamische Konfiguration erreichen will, muss man sich auf die Möglichkeiten beschränken, die mit jeder handelsüblichen Firewall verfügbar sind. Gesucht ist also ein Verfahren, welches herstellerunabhängig ist und sich leicht in jeder Art von Software implementieren lässt. Ein solches Verfahren stellt das UDP-Hole-Punching dar. Es hat nur geringe Anforderungen an die Konfiguration der Firewall, funktioniert in allen üblichen Szenarien und ist mit Hilfe der normalen Socket-API leicht zu implementieren. Im folgenden Kapitel wird das UDP-Hole-Punching detailliert beschrieben.

Kapitel 2

Das Konzept des UDP-Hole-Punching

2.1 Verhalten von Firewalls

Es ist zuerst ein genauerer Blick auf aktuelle Firewall-Technologien sinnvoll, um das Prinzip des UDP-Hole-Punching zu verstehen. Firewalls gibt es in verschiedenen Ausprägungen, die auf unterschiedlichen Protokoll-Ebenen operieren. Die meisten Firewalls arbeiten heute als sogenannte Stateful-Inspection-Engines. Sie sind sowohl als Software- als auch als Hardware-Komponenten verfügbar. Software-Firewalls werden z.B. als Personal-Firewalls auf Endsystemen eingesetzt, während zur Sicherung ganzer Netze Hardware-Komponenten mit spezieller Software eingesetzt werden. Diese sind in Netzwerk-Komponenten wie Switches und Routern integriert, oder als eigenständige Appliance verfügbar.

Ein einfacher Paket-Filter untersucht jedes einzelne Paket, das er weiterreichen soll und entscheidet anhand des vom Administrator vorgegebenen Regelwerks, ob das Paket durchgelassen wird oder ob es abgewiesen werden muss. Abgewiesene Pakete werden gelöscht, wobei je nach Konfiguration eine ICMP-Nachricht an den Sender verschickt werden kann. In IP basierten Netzen enthält jedes Paket einen IP-Header mit den IP-Adressen der Quelle und des Ziels. Das Regelwerk enthält die IP-Adressen und die verwendeten Ports der erlaubten Kommunikationspartner. Diese einfachen Paket-Filter betrachten also die Header der Ebenen 3 und 4 des ISO-OSI-Modells, um Quelle und Ziel eines Pakets zu bestimmen. Die verwendeten Transportprotokolle können ebenfalls in den Regeln bestimmt werden. Ein Beispiel für ein solches Regelwerk zeigt die folgende Tabelle. Sie regelt den Zugriff auf das interne Netz 192.168.0.0 indem sie nur bestimmte Verbindungen zuläßt. Erlaubt sind eingehende Mails über den SMTP-Port 25 an den Mailserver 192.168.1.2 (Zeile 1) sowie SSH-Zugriff von extern zu allen Rechner im internen Netz (Zeile 2). Interne Rechner dürfen nach extern nur Verbindungen zum HTTP-Port 80 öffen (Zeile 3) sowie zu einer speziellen Anwendung, welche auf Port 17070 wartet und von Rechnern im Netz 172.16.0.0 angeboten wird. Alle anderen Verbindungen werden unterbunden.

Zugang	IP Quelle	Port Quelle	IP Ziel	Port Ziel	Protokoll	Kommentar
erlaubt	*	*	192.168.1.2	25	TCP	eingehende Mails
erlaubt	*	*	192.168.0.0	22	TCP	ssh von außen
erlaubt	192.168.2.0	*	*	80	TCP	ferne Web-Server
erlaubt	192.168.2.0	*	172.16.0.0	17070	UDP	Anwendung

Tabelle 2.1: Beispiel eines Firewall-Regelwerks

Eine erweiterte Form der einfachen Paket-Filter sind die Stateful-Packet-Filter. Sie betrachten nicht jedes Paket einzeln, sondern können die Pakete einer festen Verbindung zwischen zwei Endpunkten im Netz zuordnen. Wie eine Verbindung definiert ist und wie sie erkannt und behandelt wird, hängt von dem verwendeten Transportprotokoll ab. Die beiden Transportprotokolle der TCP/IP-Familie sind das Transmission Control Protocol (TCP) und das User Datagram Protocol (UDP).

TCP: IP-basierte Netze transportieren Daten in einzelnen, unabhängigen Paketen. Diese enthalten die IP-Adressen als Informationen über den Absender und den Empfänger des Pakets. IP ist ein unzuverlässiger und verbindungsloser Datagrammdienst. Das bedeutet, dass IP keine Garantie dafür übernimmt, dass alle versendeten Pakete korrekt beim Empfänger ankommen. Sie können verloren gehen, dupliziert werden oder sich gegenseitig überholen. Aus diesem Grund wurde TCP entwickelt. TCP erweitert das IP-Protokoll, so dass es einen

zuverlässigen und verbindungsorientierten Transport von Daten zwischen zwei Prozessen auf verschiedenen Rechnern bietet. Es garantiert die Einhaltung der Reihenfolge, sowie das Erkennen von Duplikaten und von Paketverlusten. Außerdem ist es die Aufgabe eines jeden Transportprotokolls, die Pakete einem bestimmten Prozess auf dem jeweiligen Host zuzuordnen. Dazu sind einige Erweiterungen und Informationen nötig. Diese werden in einem zusätzlichen TCP-Header, welcher in den Nutzdaten des IP-Paketes enthalten ist, übermittelt. Die Zuordnung zu einzelnen Anwendungen auf den jeweiligen Rechnern geschieht durch Port-Nummern, welche einem Prozess exklusiv zugeteilt werden. TCP führt die sogenannten Sequenznummern ein, um die Einhaltung der Reihenfolge der Pakete und die Erkennung von Duplikaten, welche auf dem Weg durch das Netz entstehen können, sicher zu stellen. Jedes einzelne Paket bekommt dabei eine fortlaufende Nummer und ist somit eindeutig identifizierbar. Damit Pakete, welche auf dem Weg zum Ziel verloren gehen, erneut gesendet werden, teilt der Empfänger des Pakets den Erhalt durch ein Antwortpaket, ein sogenanntes ACK[1], mit. Ein ACK kann auch in einem Datenpaket enthalten sein, das in die andere Richtung gesendet wird. Pakete, auf denen nach einer bestimmten Zeit kein ACK erfolgt ist, werden erneut versendet. TCP bietet zusätzlich einen genau definierten Verbindungsauf und -abbau. In Abbildung 2.1 sind diese beiden Vorgänge dargestellt. Beim Aufbau handelt es sich um einen 3-Way-Handshake. Der Client sendet ein erstes SYN-Paket[2] aus, welches bereits eine erste Sequenznummer enthält. Der Server antwortet durch ein eigens SYN-Paket mit einer eigenen Sequenznummer, welches auch ein ACK für den Erhalt des ersten SYN-Paketes enthält. Der Client verschickt dann seinerseits nochmal ein ACK an den Server. Dadurch werden technisch gesehen zwei Simplex-Verbindungen etabliert, eine für jede Richtung. Diese beiden Teilverbindungen führen von einander unabhängige Sequenznummern die durch die beiden SYN-Pakete synchronisiert wurden. Der Abbau der Verbindung geschieht einzeln für jede Teilverbindung. Wenn einer der beiden Kommunikationspartner keine weiteren Daten mehr verschicken will, dann schickt er ein FIN-Paket[3]. Dieses wird von der anderen Seite bestätigt. Die TCP-Verbindung gilt als beendet, wenn dieser Abbau in beide Richtungen erfolgt ist. Ein Stateful-Packet-Filter kann die Informationen der TCP-Header auswerten und dadurch TCP-Verbindungen erkennen und analysieren. Die Firewall erkennt den 3-Way-Handshake als Verbindungsaufbau und merkt sich diese Verbindung bis zu ihrem Abbau. Es muss daher im Regelwerk auch angegeben werden, in welche Richtung eine erlaubte Verbindung aufgebaut werden darf. Pakete, die zwischen den beiden Endpunkten einer erlaubten Verbindung verschickt werden, können von der Firewall auf sinnvolle Sequenznummern kontrolliert werden, so dass nur zu dieser Verbindung gehörende Pakete akzeptiert werden. TCP implementiert noch eine Reihe weiterer Funktionen, die aber bei der Erkennung von Verbindungen keine Rolle spielen.

UDP: UDP ist ein wesentlich einfacheres Transportprotokoll. Es verzichtet auf feste Verbindungen und Zuverlässigkeit. Die Erweiterungen gegenüber IP beschränken sich im Wesentlichen auf das Einführen von Port-Nummern. Daher ist es nicht wirklich möglich, in einem UDP-basierten Datenaustausch eine Verbindung zu erkennen. Jedes Paket steht für sich und enthält nur IP-Adresse und Port-Nummer vom Sender und vom Empfänger als verwertbare Informationen. Für UDP-Verkehr erstellen Firewalls daher virtuelle Verbindungen. Wie TCP-Verbindungen können diese durch Firewall-Regeln erlaubt oder verboten werden. Eine

[1] ACK → „acknowledgment“. dt. Bestätigung
[2] SYN → „synchronize“. dt. Synchronisieren
[3] FIN → „finish “. dt. Abschluss

solche virtuelle Verbindung wird erstellt, wenn ein erstes UDP-Paket weitergeleitet wird. Die Verbindung besteht dann genau zwischen den Endpunkten dieses einen Paketes. Die Endpunkte sind dabei durch zwei Tupel bestehend aus IP-Adresse und Port-Nummer eindeutig identifiziert. Weitere Pakete, die zwischen denselben Endpunkten verschickt werden, werden als der Verbindung zugehörig betrachtet, solange die Firewall die virtuelle Verbindung in ihrer Verbindungsdatenbank führt. Beendet wird eine solche Verbindung automatisch nach dem letzten registrierten Paket bei Erreichen eines festgelegten Time-Out-Intervalls. So kann UDP-Verkehr erlaubt werden, solange er z.B. vom lokalen Netz aus durch ein erstes UDP-Paket initiiert wird. Pakete von extern werden nur zugelassen, wenn der Sender vorher ein Paket von intern erhalten hat, also wenn das von extern kommende Paket eine Antwort darstellt.

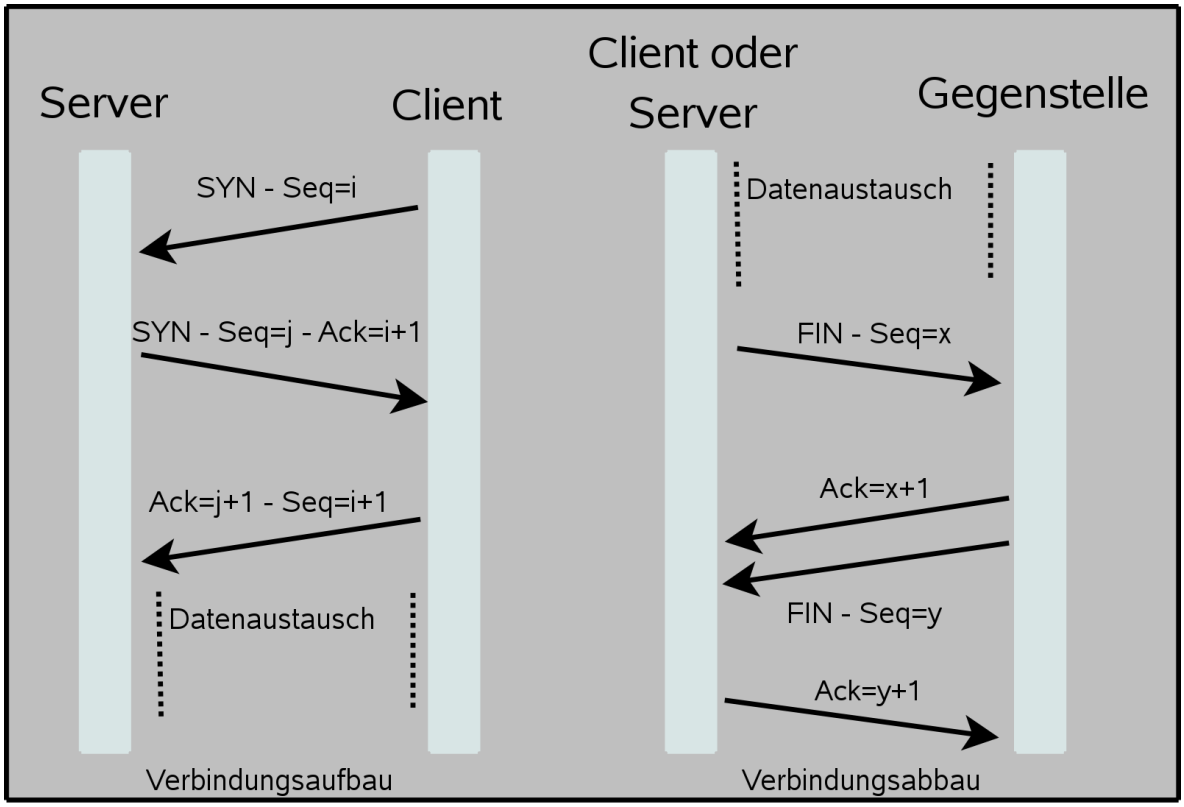

Abbildung 2.1: Auf- und Abbau einer TCP-Verbindung

2.2 Prinzip und Technik

Das UDP-Hole-Punching stellt eine Möglichkeit dar, UDP-Verbindungen im oben beschriebenen Sinne zwischen zwei Endgeräten zu etablieren, selbst wenn sie sich jeweils hinter einer Firewall befinden. Entscheidend ist, dass in der Firewall-Konfiguration keine von außen eingehenden Verbindungen erlaubt werden müssen, sondern nur ausgehender UDP-Verkehr.

Entwickelt wurde diese Technik ursprünglich, um UDP-Verkehr zwischen sowohl privat als auch geschäftlich genutzen PCs zu ermöglichen, z.B. für Voice-over-IP Anwendungen (VoIP). Dort existiert das Problem, dass die genutzten PCs meistens in privaten Netzen liegen, die

durch Router mit NAT-Funktionalität und integrierten Firewalls mit dem Internet verbunden werden. Die Firewalls lassen in ihrer Standardkonfiguration meistens keine eingehenden Verbindungen zu. Für Privatanwender ist es aus Sicherheitsgründen nicht ratsam, dies zu ändern. Zudem sind die einzelnen Rechner in privaten Netzen bei Verwendung der NAT-Technik nicht direkt ansprechbar, da sie keine öffentliche IP-Adresse haben. Es ist also kein direkter Verbindungsaufbau zwischen zwei Endpunkten möglich. Im Bereich des Grid-Computing sind die Voraussetzungen ähnlich, allerdings spielt NAT hier keine große Rolle. Rechner in Unternehmen und Forschungseinrichtungen, die sich an Grid-Projekten beteiligen, sind meistens weltweit eindeutig ansprechbar. Das UDP-Hole-Punching kann nicht nur für VoIP und ähnliche Dienste genutzt werden, sondern es ist auch bei Datentransfers in Grid-Systemen einsetzbar. Das wurde bereits im Rahmen des D-GRID Integrationsprojektes erkannt, einer Initiative zum Aufbau und Betrieb einer Grid-Plattform, welche 2004 vom Bundesministerium fur Bildung und Forschung initiiert wurde. Im Forschungszentrum Jülich wurde daraufhin ein grundlegendes Konzeptpapier [4] über den Einsatz von UDP-Hole-Punching in Grid-Systemen verfasst. Design und Implementierung eines entsprechenden Dateitransfers werden aber erst in der vorliegenden Arbeit behandelt.

Es sind gewisse Voraussetzungen in Bezug auf die Konfiguration der Firewalls nötig, um das UDP-Hole-Punching benutzen zu können. Da das Verfahren ausschließlich UDP verwendet, müssen für den Datentransfer keine TCP-Verbindungen erlaubt werden. Lediglich ausgehender UDP-Verkehr muss für die eigenen, an Dateitransfers beteiligten Systemen zugelassen werden. Im Sinne der beschriebenen virtuellen UDP-Verbindungen muss UDP-Verkehr erlaubt sein, welcher von internen Systemen initiiert wird. Bei dieser Konfiguration bleiben die eigenen Systeme für externe Verbindungsversuche weiterhin unerreichbar.

Kein beteiligtes System kann ein anderes System direkt ansprechen, deshalb muss es einen alternativen Weg zum Aufbau einer Verbindung geben. Daher ist der Einsatz eines zentralen anwendungsspezifischen Gateway-Dienstes, welcher zwischen den einzelnen Kommunikationspartnern vermittelt, wesentlicher Bestandteil des Verfahrens. Zu diesem Gateway baut jedes potenzielle Endsystem eine Kontrollverbindung auf. Wenn ein Dateitransfer ansteht, kann über diese Verbindung ein Informationsaustausch mit dem gewünschten Kommunikationspartner stattfinden. Übermittelt werden dabei von beiden Seiten die IP-Adresse und der Port des genutzten UDP-Sockets. Diese müssen zu Beginn des Informationsaustausches festgelegt werden. Sind die Sockets reserviert und die entsprechenden Informationen übermittelt, kann das eigentliche UDP-Hole-Punching stattfinden.

Da auf beiden Endsystemen die Adress-Informationen bekannt sind, kann nun einer von beiden ein erstes UDP-Paket an die andere Seite schicken. Das Paket wird von der eigenen Firewall weiter geleitet, denn dort ist ausgehender UDP-Verkehr erlaubt. Die Firewall richtet zudem eine virtuelle UDP-Verbindung zwischen den beiden Endpunkten dieses UDP-Paketes ein. Das Paket wird durch das Internet bis zur Firewall der Organisation geleitet, zu der das Zielsystem gehört. Die Firewall kann das Paket keiner bestehenden Verbindung zuordnen. Da eingehende UDP-Verbindungen für das Zielsystem nicht erlaubt sind, wird das Paket verworfen. Über die Kontrollverbindung kann dem Zielsystem nun mitgeteilt werden, dass das erste Paket versendet wurde. Dieses System fängt seinerseits an, UDP-Pakete an seinen Kommunikationspartner zu senden. Sie passieren die eigene Firewall und auch dort wird eine UDP-Verbindung für diese ausgehenden Pakete eingerichtet. Wenn die UDP-Pakete auf die Firewall des ersten Systems treffen, kann sie die Pakete der bestehenden Verbindung zuordnen, da sie von genau dem Endpunkt im Netz stammen, zu dem das ursprüngliche Paket verschickt wurde. Da nun in beiden Firewalls virtuelle UDP-Verbindungen bestehen,

können beliebig UDP-Pakete in beide Richtungen ausgetauscht werden. Diese Möglichkeit besteht, bis die Firewalls die virtuellen UDP-Verbindungen löschen. Das geschieht nach einem konfigurierbaren Zeitintervall, nachdem keine zur Verbindung gehörenden Pakete mehr von der Firewall registriert wurden.

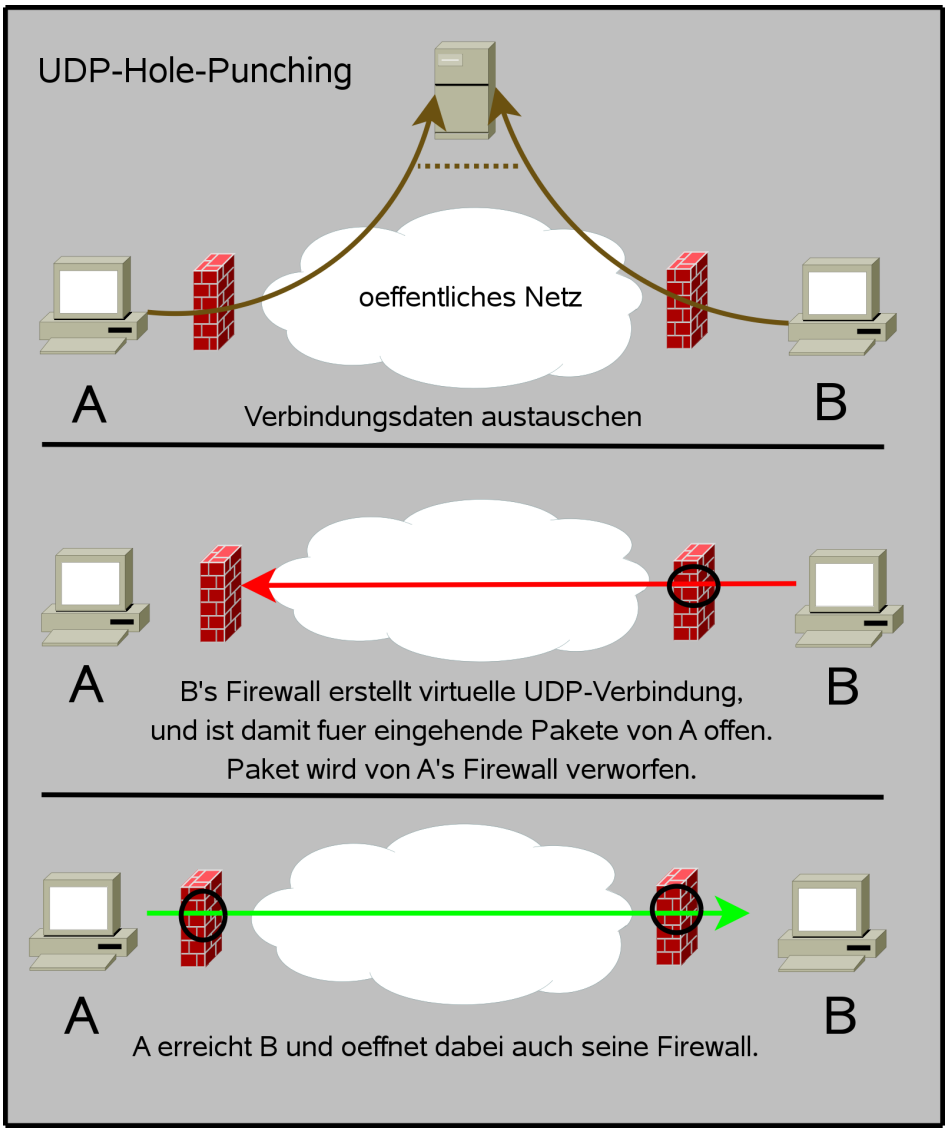

Abbildung 2.2: Prinzip des UDP-Hole-Punching

2.3 Proof of Concept

Im Rahmen dieser Arbeit wird ein Proof-Of-Concept entwickelt, um die Funktionstüchtigkeit des UDP-Hole-Punching zu zeigen und die Reaktionen verschiedener Firewalls genau zu studieren. Das Programmpaket besteht nur aus einer Server- und einer Client-Komponente. Diese tauschen die nötigen Informationen über eine TCP-Verbindung aus und führen dann das UDP-Hole-Punching und den Datentransfer durch. Auf einen Gateway-Dienst sowie Authentifizierung und Verschlüsselung wird hierbei verzichtet.

2.3.1 Funktionsbeschreibung

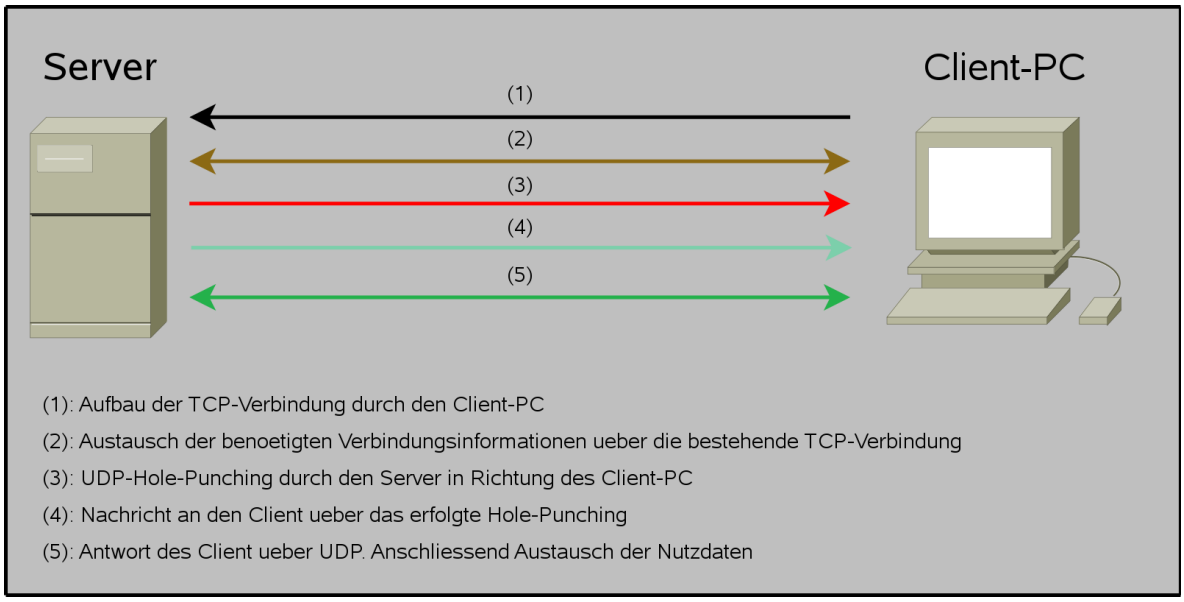

Abbildung 2.3: Ablauf des Dateitransfers im Proof-of-Concept

Der grundlegende Ablauf des Dateitransfers ist in Abbildung 2.3 dargestellt. Im folgenden werden die einzelnen Schritte genauer beschrieben.

Der Server soll TCP-Verbindungen an einem festen, öffentlich erreichbaren Port entgegennehmen. Initiiert wird ein Dateitransfer vom Client, welcher eine TCP-Verbindung zum Server aufbaut. Über diese Verbindung werden die benötigten UDP-Verbindungsdaten ausgetauscht. Das sind die IP-Adressen und Ports der beiden Sockets, die auf Server- bzw. Client-Seite für den anstehenden Transfer reserviert wurden. Anschließend kann der Server das eigentliche UDP-Hole-Punching durchführen. Er versendet dazu von dem bereits reservierten Port ein einzelnes UDP-Paket ohne Daten an den Endpunkt, der ihm vom Client mitgeteilt wurde.

Dieses Paket wird vermutlich von der Firewall des Client verworfen. Es kann aber theoretisch bis zum Client gelangen, falls dessen Firewall entsprechend geöffnet ist. Bei einer solchen Konfiguration ist das UDP-Hole-Punching zwar überflüssig, trotzdem muss dieser Fall bei der Entwicklung beachtet werden. Der Client ignoriert eingehende Pakete zu diesem Zeitpunkt komplett. Erst wenn der Server über die bestehende Kontrollverbindung mitteilt, dass das Hole-Punching durchgeführt wurde, wird der Client aktiv. Er versendet ein UDP-Paket an

den Server und schafft damit eine passende UDP-Verbindung in seiner eigenen Firewall. Der Server erhält das Paket und antwortet mit einem weiteren UDP-Paket. Dieses Vorgehen stellt einen einfachen Verbindungsaufbau auf Anwendungsebene dar. Über die Kontrollverbindung werden dann weitere Informationen zum Dateitransfer ausgetauscht. Das sind die Richtung des Transfers, sowie der Pfad und der Name der zu übertragenden Datei auf Server-Seite. Diese Informationen bekommt der Client beim Start als Argumente auf der Kommandozeile übergeben. Nun sind alle Voraussetzungen für den Dateitransfer geschaffen und die Nutzdaten werden in UDP-Paketen versendet. Anschließend kann auch die TCP-Verbindung von beiden Seiten beendet werden. Um auch über UDP-Sockets einen zuverlässigen Dateitransfer zu ermöglichen, wurde für den Transfer das auf Anwendungsschicht arbeitende UDT-Protokoll genutzt. Für das UDP-Hole-Punching spielt dieses Protokoll keine Rolle. Es wird in einem späteren Kapitel genauer beschrieben.

2.3.2 Testlauf

```
Regelwerk:

timeout udp 00:02:00

access-list outside extended permit tcp any host 10.0.1.4 eq 17070
access-list inside  extended permit udp any any
access-list outside extended deny   any any any
access-list inside  extended deny   any any any

Log:

Built inbound TCP connection 219025308
   for outside:10.0.0.6/52376 (10.0.0.6/52376)
   to inside:10.0.1.4/17070 (10.0.1.4/17070)

Built outbound UDP connection 219025309
   for inside:10.0.1.4/33101 (10.0.1.4/33101)
   to outside:10.0.0.6/32988 (10.0.0.6/32988)

Teardown TCP connection 219025308
   for outside:10.0.0.6/52376
   to inside:10.0.1.4/17070 duration 0:00:04 bytes 1396 TCP-FINs

Teardown UDP connection 219025309
   for inside:10.0.1.4/33101
   to outside:10.0.0.6/32988 duration 0:02:04 bytes 23154
```

Abbildung 2.4: Firewall-Log während eines Dateitransfers

Zur Analyse des Verhaltens dieses Proof-Of-Concept wurden Testläufe in verschiedenen Konfigurationen durchgeführt. Einer dieser Tests wird im Folgenden genau beschrieben und ausgewertet. Um die Vorgänge auf Netzwerk-Ebene genau beobachten zu können, wurden alle IP-Pakete, die zwischen den beiden beteiligten Systemen verschickt wurden, auf Server-Seite mit der Software Ethereal aufgezeichnet. Das von Ethereal erzeugte Protokoll wurde zur besseren Lesbarkeit umformatiert und eingefärbt. Es ist im Anhang vollständig abgedruckt. Abbildung 2.5 zeigt ein Diagramm der Abläufe. Im Folgenden wird auf die einzelnen Pakete anhand ihrer Nummer Bezug genommen, welche in der ersten Spalte des Protokolls steht und auch im Diagramm angegeben ist. Außerdem wurde der wesentliche Teil der Firewall-Konfiguration auf Server-Seite protokolliert, sowie die von der Firewall in der Zeit des Transfers erstellten Log-Einträge (Abbildung 2.4).

Der Server wurde auf einem Rechner mit der IP-Adresse 10.0.1.4 gestartet und erwartet TCP-Verbindungen an Port 17070, während der Client auf dem Rechner 10.0.0.6 ausgeführt wird. Das Subnetz, in welchem sich der Server befindet, wird durch eine Firewall geschützt. Wie im Regelwerk zu sehen ist, sind von außen nur TCP-Verbindungen nach Port 17070 des Servers erlaubt und von innen ist beliebiger UDP-Verkehr nach außen gestattet. Anderer Verkehr ist verboten und wird verworfen.

Der Client baut nun eine Kontrollverbindung zum Server auf. Das geschieht durch den bereits beschreibenen 3-Way-Handshake (Pakete 1 bis 3). Im Log der Firewall sieht man einen entsprechenden Eintrag „Built inbound TCP connection“. Anschließend werden über die bestehende TCP-Verbindung die UDP-Verbindungsdaten ausgetauscht. Jeder der beiden Kommunikationspartner teilt der Gegenstelle seine IP-Adresse und seinen Port für den Dateitransfer mit (Pakete 4 bis 8). Der Server kann nun das UDP-Hole-Punching ausführen. Dazu verschickt er ein einzelnes UDP-Paket ohne Daten an den Client (Paket 9). Im Protokoll ist erkennbar, dass der Client Port 32988 gewählt hat und der Server Port 33101. Die IP-Adressen sind die gleichen wie bei der Kontrollverbindung. Es könnten auch abweichende IP-Adressen zum Einsatz kommen, wenn die Systeme z.B. mehrere Netzwerk-Interfaces haben. Im Firewall-Log wird für dieses Paket eine virtuelle UDP-Verbindung angezeigt: „Built outbound UDP connection“. Der Client bekommt über die Kontrollverbindung die Nachricht, dass das UDP-Hole-Punching stattgefunden hat (Paket 10) und antwortet mit einem ersten UDP-Paket (Paket 11). Der Server antwortet mit einem weiteren UDP-Paket (Paket 12). Über die Kontrollverbindung werden weitere Informationen wie Dateiname und Dateigröße übermittelt.

Der eigentliche Dateitransfer geschieht mit dem Paketen 19 bis 43. Die Datei wird vom Server zum Client übertragen. Die Pakete 19, 37 und 43 stellen ACKs dar, die auf Anwendungsebene erzeugt werden, um die erforderliche Zuverlässigkeit zu erhalten. Nach dem Transfer findet ein einfacher, auf Anwendungsebene implementierter Verbindungsabbau statt (Pakete 44 bis 46). Die Kontrollverbindung kann jetzt abgebaut werden. Dies wird von der Firewall registriert: „Teardown TCP connection“. Das Löschen der virtuellen UDP-Verbindung in der Verbindungsdatenbank der Firewall geschieht nach einem Zeitintervall, welches hier auf zwei Minuten eingestellt ist: „Teardown UDP connection“.

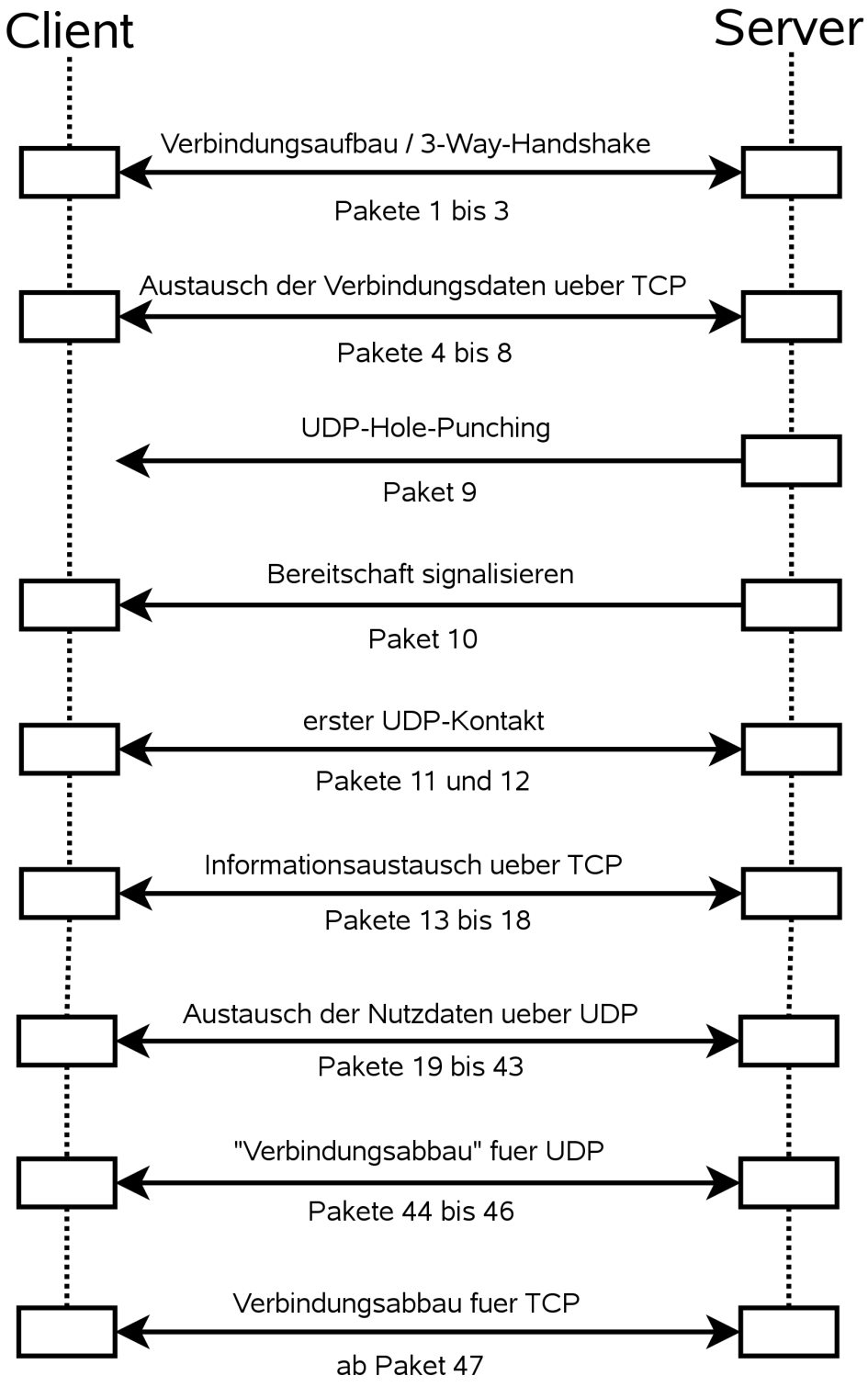

Abbildung 2.5: Mitschnitt des Netzwerkverkehrs eines Dateitransfers

2.4 Sicherheitskritische Bewertung des Verfahrens

Das beschriebene Verfahren wurde als Möglichkeit zum Verbindungsaufbau in Grids ausgewählt, weil es effizient ist und trotzdem einen sicheren Betrieb der beteiligten Systeme und Netzwerke erlaubt. Es macht daher Sinn, das Verfahren in Bezug auf Sicherheit genauer zu bewerten.

Im Folgenden sollen zuerst die sicherheitsrelevanten Vorteile des Verfahrens im Vergleich zu anderen üblichen Dateitransfer-Methoden und zu statischer Firewall-Konfiguration dargestellt werden:

1. Die Systeme müssen von außen nicht erreichbar sein. Eingehende TCP-Verbindungen oder UDP-Datenströme müssen in den Firewall-Regeln bis auf die erwähnte Kontrollverbindung nicht erlaubt werden, was zu einer erhöhten Sicherheit der Systeme im Vergleich zu anderen Verfahren, welche auf von außen erreichbare Ports angewiesen sind, führt.

2. Aus dem ersten Punkt ergibt sich, dass die Regelwerke der Firewalls kleiner gehalten werden können als beim Einsatz von Ports, die von außen erreichbar sein müssen. Das führt auch zu höherer Performance der Firewall.

3. Datenaustausch mit Systemen, welche außerhalb des eigenen Netzes liegen, ist immer nur für begrenzte Zeit möglich. Wenn beide beteiligten Systeme keine Daten mehr senden, dann wird die virtuelle UDP-Verbindung von der Firewall gelöscht und das „UDP-Loch“ ist wieder geschlossen. In den Firewall-Regeln konfigurierte Verbindungen sind dagegen immer erlaubt.

4. Eine virtuelle UDP-Verbindung gilt nur für genau eine Kombination aus zwei Endpunkten im Netz. „Endpunkte“ sind dabei durch ihre IP-Adresse und einen Port definiert. Das Öffnen der Firewall geschieht also mit größtmöglicher Genauigkeit. Es werden keine unnötigen Systeme oder Ports von außen sichtbar.

Ganz ohne statische Regeln in der Firewall-Konfiguration kommt das UDP-Hole-Punching aber nicht aus. Es muss im Regelwerk erlaubt werden, dass alle Rechner im eigenen Netz, die an Dateitransfers beteiligt sein könnten, UDP-Pakete nach außen schicken dürfen. Dabei stellt sich die Frage, wie genau beim Formulieren dieser Regel vorgegangen werden soll. Die einfachste Variante ist, grundsätzlich allen eigenen Rechnern oder zumindest betroffenen Teilnetzen beliebigen ausgehenden UDP-Verkehr zu erlauben. Die Erlaubnis kann aber auch genau auf die am Grid beteiligten Systeme beschränkt werden. Damit wird das Regelwerk wieder größer, und man muss neue Systeme von Hand eintragen.

Wie gefährlich die Erlaubnis für ausgehenden UDP-Verkehr ist, hängt von den Sicherheitsrichtlinien der jeweiligen Organisation ab. Im Allgemeinen gelten eigene Systeme als wesentlich vertrauenswürdiger als externe Systeme. Ausgehende Verbindungen werden von vielen Rechnern aus ohnehin erlaubt, um die Dienste des Internets nutzen zu können. Ausgehender UDP-Verkehr ist für viele wichtige Protokolle wie DNS oder NTP nötig.

Um das UDP-Hole-Punching für ungewollte Zwecke missbrauchen zu können, sind im wesentlichen die folgenden Szenarien denkbar:

- Die Anwendung, die das UDP-Hole-Punching benutzt, ist nicht fehlerfrei implementiert. Wenn eine Anwendung das Hole-Punching mit falschen Parametern benutzt, werden z.B. falsche oder unnötige Löcher in der Firewall geöffnet. Diese Löcher könnten benutzt werden, um ungewollte Informationen von außen an ein internes System zu senden.

- Ein Angreifer erlangt auf einem vom UDP-Hole-Punching unabhängigen Weg Kontrolle über ein System und kann dort eigene Anwendungen starten. Die Erlaubnis für ausgehenden UDP-Verkehr ist nicht auf bestimmte Anwendungen beschränkt und kann somit von Benutzern auch in bösartiger Absicht für Datentransfers genutzt werden.
- Durch Manipulation der IP-Header eines UDP-Paketes können Pakete in einen bereits laufenden Datenaustausch eingeschleust werden. Ein Angreifer kann die Adressen der beiden eigentlichen Kommunikationspartner in ein beliebiges UDP-Paket eintragen. Da UDP keine Sequenznummern oder ähnliche Mechanismen zur Identifikation der einzelnen Pakete benutzt, sind solche gefälschten Pakete nicht von korrekten Paketen zu unterscheiden. Dies kann z.B. für Denial-of-Service-Attacken genutzt werden.

Es sind entsprechende Gegenmaßnahmen zu treffen, um diese Gefahren abzuwenden. Ein Fehlverhalten der Anwendung muss durch ausführliche Tests vom Programmierer verhindert werden. Alle anderen beschriebenen Angriffsszenarien sind darauf angewiesen, dass das Netzwerk auf einem vom UDP-Hole-Punching unabhängigen Weg angreifbar ist. Die Endsysteme müssen vor Angriffen durch Personal-Firewalls, Viren-Scanner und ähnliche Sicherheitsprodukte geschützt werden. Auf manipulierte IP-Pakete kann bereits an der Firewall geachtet werden, z.B. durch Überprüfung der eigenen Routing-Informationen oder durch die Analyse des Datenverkehrs mit Intrusion-Prevention-Systemen (IPS). Die Vorraussetzungen für einen Angriff werden durch diese Maßnahmen so weit erschwert, dass das Verfahren als ausreichend sicher betrachtet werden kann.

Kapitel 3

Architektur einer allgemeinen Dateitransfer-Lösung

3.1 Architektur des Gesamtsystems

Es sind eine Reihe von weiteren Design-Entscheidungen nötig, um aus den bestehenden Erkenntnissen zum UDP-Hole-Punching eine praxistaugliche Dateitransferlösung zu erstellen. Es wurde bereits gesagt, dass Kontrollinformationen über ein Gateway-System augetauscht werden müssen. Somit sind drei verschiedene Komponenten zu entwickeln: ein Client, ein Server und ein Gateway. Die Komponenten werden im Folgenden einzeln beschrieben.

3.1.1 Die Programm-Komponenten

Die Gateway-Komponente wird in einem Grid nur einmal benötigt, bei beliebig vielen Servern und Clients. Das Gateway-System nimmt Verbindungen sowohl von Dateitransfer-Servern als auch von Clients entgegen. Diese reinen Kontrollverbindungen übermitteln nicht die zu versendenden Daten. Wegen der kleinen Datenmengen auf diesen Verbindungen hat deren Durchsatz keinen nennenswerten Einfluß auf die Performance des gesamten Dateitransfers. Da es nur einen einzigen Gateway gibt und dieser nur an einem einzigen öffentlichen Port erreichbar sein muss, kann diese Kontrollverbindung als eine gewöhnliche TCP-Verbindung etabliert werden, welche durch statische Firewall-Konfiguration erlaubt wird. Als von außen erreichbares System sollten für den Gateway strengere Sicherheitsrichtlinien gelten als für die anderen Komponenten. Der Gateway führt Informationen über alle Systeme, die eine Verbindung zu ihm unterhalten. Seine wichtigste Aufgabe besteht darin, Nachrichten, welche er von einem der verbundenen Systeme erhält, an ein anderes System weiterzuleiten. So können Clients und Server Informationen austauschen ohne direkt miteinander verbunden zu sein. Außerdem fällt die Authentifizierung der verbundenen Systeme in den Aufgabenbereich des Gateways. So muss sich ein Server-System beim Gateway authentifizieren, um Daten mit anderen Systemen austauschen zu dürfen. Dies ist insbesondere aus Sicherheitsgründen nötig, weil der Gateway einen dauerhaft von Rechnern außerhalb des eigenen Netzes erreichbaren Port bereitstellt. Weiterhin muss der Gateway eine Autorisierung vornehmen, falls dies nicht von einer anderen Grid-Komponente übernommen wird. So werden nicht nur Systeme sondern auch Benutzer authentifiziert, wodurch eine genaue Rechtevergabe möglich wird. Transfers kommen dann nur zwischen bestimmten Systemen zu Stande und auch nur, wenn der angemeldete Benutzer dazu berechtigt ist. Eine Grid-Middleware implementiert diese Funktionen auch für andere Bereiche als den Dateitransfer. Der Gateway muss die verwendeten Protokolle und Standards entsprechend unterstützen.

Ein Server muss für jedes System des Grids laufen auf dem Daten gespeichert sind. Diese Daten sollen im Sinne der Grid-Architektur für alle Nutzer zugänglich und somit jederzeit abrufbar und änderbar sein. Ein Server stellt diese Daten zur Verfügung, indem er sich beim Gateway als potenzieller Partner für Dateitransfers anmeldet. Dazu authentifiziert sich der Server wie oben beschrieben und hält die bestehende Kontrollverbindung dauerhaft offen. Anschließend wartet der Server ab, bis er über die Kontrollverbindung vom Gateway eine Anfrage für einen Dateitransfer erhält. Erreicht den Server eine solche Nachricht, kann er über die Verbindung zum Gateway weitere Informationen mit dem Kommunikationspartner austauschen, so dass die Voraussetzungen für ein UDP-Hole-Punching geschaffen werden. Sind alle nötigen Informationen vorhanden, wird unabhängig von der Kontrollverbindung zum Gateway ein direkter UDP-Datenaustausch mit dem anfragenden Client initiiert. Über diesen UDP-Verkehr werden dann die eigentlichen Dateien übertragen. Der Server muss mehrere Clients gleichzeitig als Concurrent-Server bedienen.

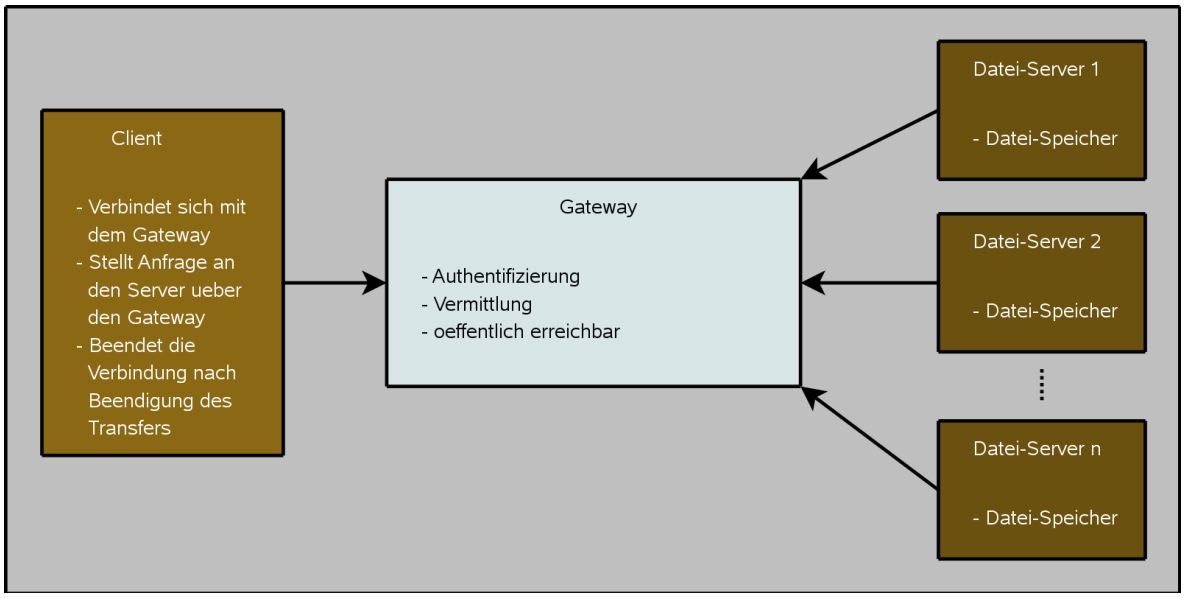

Abbildung 3.1: Architektur eines Dateitransfer-Programms

Clients sind in diesem Szenario eine kurzlebige Komponente. Sie werden gestartet, wenn ein Dateitransfer unmittelbar durchgeführt werden soll. Ein Client etabliert dazu eine Kontrollverbindung zum Gateway wie für den Server bereits beschrieben. Der Benutzer des Clients muss sich beim Gateway anmelden, damit dessen Autorisierung für den anstehenden Dateitransfer überprüft werden kann. Der Client wählt dann aus der Liste der am Gateway angemeldeten Server ein System aus mit dem er eine Datei austauschen will. Anschließend können die nötigen Kontrollinformationen ausgetauscht werden, um einen direkten UDP-Datenverkehr zwischen Client und Server zu ermöglichen. Außerdem bestimmt der Client welche Datei versendet werden soll und in welche Richtung der Transfer geht. Auch diese Informationen werden über die Kontrollverbindung ausgetauscht. Wenn alle nötigen Informationen durch den Gateway übermittelt wurden, kann der Client die Kontrollverbindung beenden. Sie ist für den eigentlichen Dateitransfer nicht mehr nötig. Stattdessen kann nun der eigentliche UDP-Verkehr erfolgen.

3.1.2 Authentifizierung, Autorisierung und Verschlüsselung

In einer praxistauglichen Dateitransferlösung zum Einsatz in Grid-Systemen spielen Authentifizierung und Verschlüsselung eine wichtige Rolle. Kein Grid kommt ohne eine leistungsfähige Rechteverwaltung aus. Daher sollen hier mögliche Implementierungen dieser Funktionen genauer betrachtet werden. Die Server und der Gateway müssen sich gegenseitig authentifizieren können, damit zu jedem Zeitpunkt eine Nichtabstreitbarkeit gegeben ist. Ein geeignetes Verfahren hierfür ist zum Beispiel die Verwendung von X.509-Zertifikaten. X.509 ist ein ITU-Standard [1] und das derzeit wichtigste und verbreitetste Verfahren für eine zertifikatsbasierte Public-Key-Infrastruktur. Diese maschinenbezogenen Zertifikate ermöglichen die eindeutige Identifizierung eines Systems, wodurch eine unerlaubte Nutzung des Gateways verhindert

[1] ITU → International Telecommunication Union

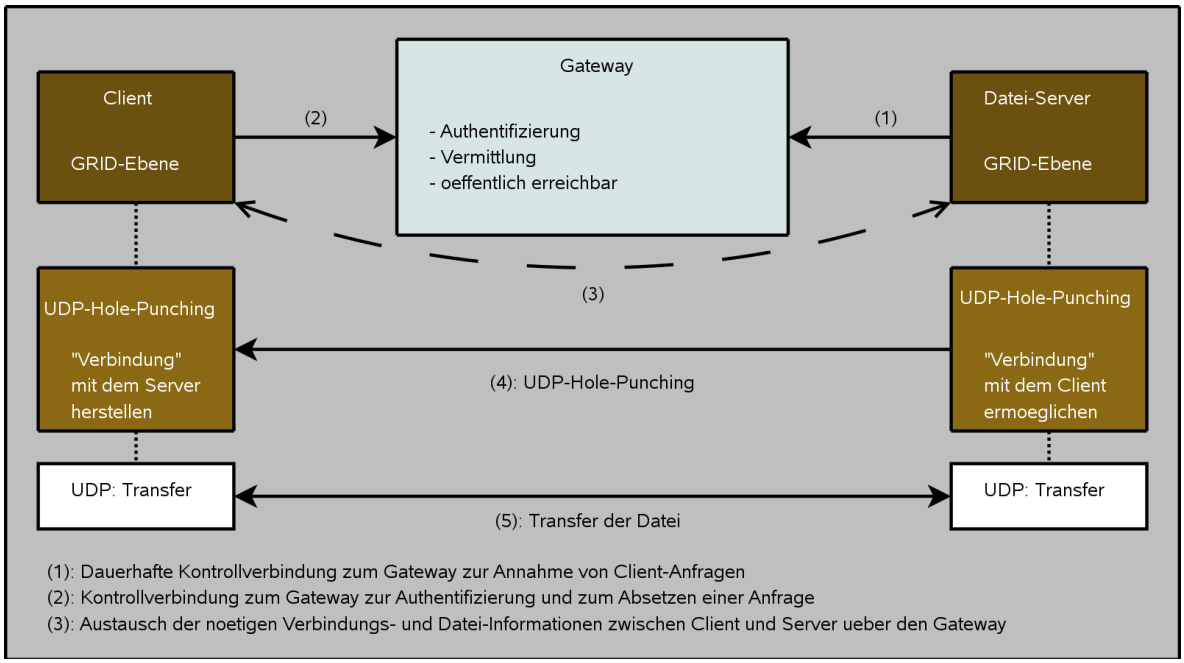

Abbildung 3.2: Ablauf eines Dateitransfers

werden kann. Es müssen allerdings nicht nur die Server- und Gateway-Systeme authentifiziert werden, sondern auch die Benutzer. Dazu können personenbezogene Zertifikate benutzt werden.

Nachdem Rechner und Personen identifiziert sind, muss eine entsprechende Rechteverwaltung möglich sein. Es muss entschieden werden können zwischen welchen Benutzern und Rechnern Dateitransfers überhaupt zulässig sind. Zudem muss festgelegt sein, welche Zugriffsrechte der Nutzer auf der jeweiligen Datei auf dem Server hat. Das kann geschehen, indem die Benutzernamen des Grids auf real existierende Benutzerkonten auf den Servern abgebildet werden. So ist die normale Verwaltung von Zugriffsrechten für einzelne Benutzer auf dem Server nutzbar.

Die Daten zur Autorisierung und Authentifizierung sollten nicht über unverschlüsselte Datenkanäle versendet werden, damit sie nicht von Außenstehenden mitgelesen werden können. Daher ist es ratsam, die Kontrollverbindungen zum Gateway zu verschlüsseln. Dies ist durch die eingesetzten Zertifikate bereits gewährleistet. Die üblichen Techniken zur Verschlüsselung ganzer TCP-Sessions sind SSL [2] und dessen Nachfolger TLS [3], welche X.509-Zertifikate nutzen. Damit ist nicht nur die Vertraulichkeit von Benutzernamen und Dateinamen gegeben, auch die ausgetauschten UDP-Verbindungsinformationen der Endgeräte können nicht mitgelesen werden. Ein potentieller Angreifer erfährt hier nichts über den Standort und die Adresse der einzelnen Grid-Komponenten.

Die eigentlichen Daten bleiben von dieser Verschlüsselung unberührt. Wenn der Inhalt der versendeten Dateien schützenswert ist, muss über eine Verschlüsselung des UDP-Verkehrs nachgedacht werden. SSL ist für UDP nicht anwendbar, daher muss die Verschlüsselung auf

[2] SSL → Secure Socket Layer
[3] TLS → Transport Layer Security

Anwendungsebene stattfinden. Hier könnte ein als ausreichend sicher empfundenes, symmetrisches Verfahren benutzt werden, wie z.B. DES oder AES. Symmetrische Verschlüsselungsverfahren sind weniger rechenaufwendig als asymmetrische Verfahren. Der gemeinsame Schlüssel muss allerdings sicher übertragen werden, was über die Kontrollverbindungen geschehen kann. Verschlüsselung ist aber in jedem Fall rechenintensiv und belastet damit die Performance der beteiligten Systeme. Die Menge der ausgetauschten Daten ist bei den Kontrollverbindungen nicht besonders hoch, weshalb ein höherer Rechenaufwand aufgrund des hohen Gewinns an Sicherheit akzeptiert werden sollte. Der eigentliche Dateitransfer ist allerdings auf hohe Durchsatzraten angewiesen und würde an Nutzen stark verlieren, wenn er durch eine aufwendige Verschlüsselung ausgebremst wird. Gerade Server-Systeme mit vielen gleichzeitigen Verbindungen könnten in ihrer gesamten Funktion stark behindert werden. Abhängig vom konkreten Einsatzzweck ist zwischen Durchsatz und Sicherheit abzuwägen.

3.2 Das UDT-Protokoll

Ein wichtiges Merkmal des UDP-basierten Datenaustausches wurde bisher noch nicht ausreichend diskutiert: UDP ist ein verbindungsloser und unzuverlässiger Datagramm-Dienst. Es können also Datagramme beim Versand verloren gehen, dupliziert werden oder ihre Reihenfolge ändern. Für einen Dateitransfer ist das natürlich nicht akzeptabel. Man muss sich darauf verlassen können, dass die Daten vollständig und korrekt übertragen werden. Wenn das Transportprotokoll diese Funktionalität nicht liefern kann, muss sie auf Anwendungsebene implementiert werden. Ideal ist dabei eine Funktionsbibliothek, welche basierend auf UDP-Sockets [2] eine den TCP-Sockets ähnliche API bereit stellt. Eine solche Funktionsbiliothek stellt UDT [4] dar.

3.2.1 Funktionen

UDT wird am „*National Center for Data Mining*“ der University of Illinois entwickelt. Es bietet eine API, die mit der normalen Socket-API für TCP-Sockets nahezu identisch ist. Intern werden aber nur UDP-Sockets benutzt. Die Entwickler beschreiben UDT wie folgt:

> „UDT is an application level data transport protocol for the emerging distributed data intensive applications over wide area high-speed networks“[5].

UDT ist in C++ implementiert. Es bietet einen eigenen Namensraum und orientiert sich streng an der normalen Socket-API. Es braucht daher keinen großen Aufwand zur Einarbeitung und kann mit nur wenigen Anpassungen in bestehende Projekte integriert werden. Die wesentlichen Funktionen und Stärken von UDT sind:

Anwendungsebene: Da UDT als Bibliothek auf Anwendungsebene implementiert ist, kann es von jeder Anwendung benutzt werden. Es sind keine Veränderungen im Kernel bzw. im TCP/IP-Stack der Systeme notwendig. Die Anwendungen brauchen auch keine administrativen Rechte auf den Systemen.

Systemunabhängig: Die Bibliothek ist sowohl für Linux als auch für Windows verfügbar.

[4] UDT → UDP-based Data Transfer

Open-Source: Die Bibliothek ist bis einschließlich Version 3.3 unter der „GNU Library General Public License“ (LGPL) verfügbar. Ab Version 4 wird sie unter der BSD Lizenz veröffentlicht. Damit ist eine ausreichende Nutzbarkeit sowohl für Open-Source-Projekte als auch für kommerzielle Produkte sichergestellt.

Verbindungen: UDT bietet einen wohl definierten Verbindungsauf- und abbau. Genau wie bei TCP gibt es listening-Sockets und Funktionen wie „connect“, „accept“ und „close“. Damit können feste Verbindungen zwischen zwei Endpunkten hergestellt und wieder beendet werden.

Sequenznummern: Genau wie TCP verwendet UDT Sequenznummern, um die richtige Reihenfolge der Pakete einhalten zu können. Zudem werden so duplizierte Pakete erkannt.

Bestätigungen: UDT verschickt ACKs für erhaltene Pakete, so dass verlorengegangene Pakete erkannt werden können. Diese werden dann vom Sender erneut verschickt, um eine vollständige Übertragung zu garantieren.

Stau- und Flusskontrolle: Wichtige Merkmale von TCP sind die Flusskontrolle mittels Sliding-Window-Technik und die Staukontrolle. Da UDP keine von beiden bereit stellt, muss UDT diese Aufgaben übernehmen.

3.2.2 Durchsatzraten und Fairness

UDT implementiert alle wichtigen Funktionalitäten von TCP, verhält sich in der Praxis aber anders. UDT wurde für ganz bestimmte Einsatzzwecke konzipiert und optimiert, und unterscheidet sich daher in einigen Punkten von TCP. Das Szenario, in dem UDT entwickelt wurde, ähnelt stark den in dieser Arbeit beschriebenen Gegebenheiten. Grid-Systeme werden als Anwendung für das Protokoll explizit genannt. Schnelle, für wenige Anwendungen reservierte Netze sind ebenfalls ein bevorzugter Einsatzort. Ziel der Entwickler war es, in dieser Umgebung die vorhandenen Kapazitäten so gut wie möglich auszulasten. Dazu sind im Vergleich zu TCP einige Änderungen notwendig. Diese betreffen hauptsächlich die Fluss- und Staukontrolle.

Neben dem Sliding-Window, welches sich bei TCP und UDT nicht wesentlich unterscheidet, muss auch ein „congestion window“ geführt werden. Dieses soll einer Überlastung des Netzwerks vorbeugen, indem es die Anzahl der versendeten, aber noch nicht bestätigten Pakete limitiert. Ist dieses Limit erreicht, dann wird das nächste Paket erst versendet, wenn ein anderes durch ein ACK vom Empfänger bestätigt wurde. Dieses Fenster reguliert automatisch die Übertragungsrate.

Bei TCP startet die Fenstergröße zu Beginn einer Verbindung mit dem Wert 1. Für jedes bestätigte Paket wird sie um 1 erhöht und verdoppelt sich somit innerhalb einer Roundtrip-Zeit. Dies wird fortgeführt, bis ein erster Paketverlust erkannt wurde. Dann wird das congestion window halbiert und anschließend nur noch linear erhöht. Diese Start-Phase wird Slow-Start genannt. Es folgt ein ständiger Wechsel von linearer Erhöhung und Halbierung bei Paketverlust, was in einem Diagramm zu der typischen Sägezahn-Struktur führt. Erkannt werden Paketverluste dabei durch duplizierte ACKs. Diese verschickt der Empfänger, wenn er auf ein Paket mit der Sequenznummer x wartet aber nur Pakete mit einer Nummer empfängt, die größer als x ist. Er bestätigt dann immer wieder das Paket x-1. Der Sender muss nach

drei dieser ACKs davon ausgehen, dass das Paket x verloren ging. Eine andere Möglichkeit zur Erkennung eines Verlustes ist der Timeout. Wenn in einem bestimmten Zeitintervall kein ACK empfangen wurde, deutet das auf ein größeres Problem im Netz hin. Das congestion window wird daraufhin nicht halbiert, sondern auf 1 gesetzt. Es beginnt ein neuer Slow-Start.

Die Begrenzung der Durchsatzrate einer TCP-Verbindung anhand der Paketverluste kann auch auf mathematischem Weg beschrieben werden. Es gilt dabei folgende Formel:

$$BW <= (\frac{MSS}{RTT}) * (\frac{1}{\sqrt{p}})$$

mit: BW = Durchsatzrate ; MSS = Maximale Segmentgröße ; RTT = Roundtrip-Zeit ; p = Paketverlustrate. Transportprotokolle, die sich an diese Formel halten werden als *„TCP-friendly“* bezeichnet.

Diese Vorgehensweise löst Staus im Netz zwar effektiv auf, führt aber zu einer schlechten Ausnutzung der Ressourcen, denn bereits kleine Störungen können die Datenrate stark einbrechen lassen. Zudem hat man das Problem, dass man bei der Übertragung vieler kleinerer Dateien nie über die Slow-Start-Phase hinauskommt, wenn man für jede Datei eine eigene Verbindung nutzt. Auch in nahezu ungenutzten Netzen kommt es in regelmäßigen Abständen zu Paketverlusten, die nichts mit einer Stausituation zu tun haben.

Da die Staukontrolle bei UDT modular aufgebaut ist, kann man als Entwickler eigene Algorithmen implementieren. Diese werden über Callback-Interfaces in die UDT-Bibliothek eingebunden. Verwendet man in UDT den gleichen Algorithmus wie in TCP, bietet es optimale Fairness zu TCP-Strömen und erfüllt die Bedingungen für *„TCP-friendliness“*. UDT bringt bereits einen Algorithmus zur Staukontrolle mit, doch im Vergleich zu TCP geht dieser anders vor. Er erreicht bereits nach kurzer Zeit die maximale Datenrate des genutzten Netzes und reagiert auf vereinzelte Paketverluste weniger stark als TCP. Trotzdem verhalten sich mehrere UDT-Ströme in einem Netz fair zueinander. Das bedeutet, dass bei wirklicher Überlast die verfügbare Bandbreite gleichmäßig unter den Datenströmen aufgeteilt wird. TCP-Ströme werden von diesem Verfahren zwar weniger stark beeinträchtigt als es bei unkontrolliertem UDP-Verkehr der Fall wäre, aber „TCP-friendly“ ist das Verfahren nicht.

UDT ist also zum Einsatz in schnellen Netzen sehr gut geeignet und kann durch seine Modularität an die spezifischen Gegebenheiten optimal angepasst werden.

3.3 Zusammenfassung

In diesem Kapitel wurde eine mögliche Architektur für einen schnellen, zuverlässigen und firewallfreundlichen Dateitransfer skizziert. Der Einsatz eines Gateway-Dienstes erlaubt in Verbindung mit dem UDP-Hole-Punching eine einfache und sichere Konfiguration der Firewall. Server können flexibel zum Gesamtsystem hinzugefügt und wieder entfernt werden. Die Clients haben jederzeit Zugriff auf alle angemeldeten Server und der eigentliche Dateitransfer kann mit hoher Durchsatzrate auf direktem Wege erfolgen, ohne dass der Gateway belastet wird. Für höhere Verfügbarkeit ist der Einsatz von mehreren redundanten Gateways denkbar.

Wenn man eine solche Dateitransferlösung implementieren möchte, sind also drei Komponenten zu entwickeln: Der Gateway, der Server und der Client. Zudem muss für den praktischen Einsatz eine Zertifizierungsstelle zur Vergabe von Zertifikaten für Systeme und Nutzer eingerichtet werden. Die Zugriffsrechte innerhalb der VO müssen unter Berücksichtigung der real existierenden Nutzerkonten auf den Servern vergeben werden. So erhält man eine umfassende Lösung für Dateitransfers in verteilten Systemen.

Wenn der Dateitransfer aber mit einer bereits existierenden Grid-Middleware zusammenarbeiten soll, ist eine solche stand-alone-Variante nicht die beste Lösung. Grid-Software enthält standardmäßig Mechanismen zur Authentifizierung und Autorisierung. Auch eine Möglichkeit für Dateitransfers ist in einer Grid-Software normalerweise vorhanden. Will man eine solche Software um die hier beschriebenen Möglichkeiten erweitern, sollte der neue Dateitransfer direkt in die bestehenden Programme integriert werden. So können die existierenden Mechanismen mitbenutzt werden und man erspart sich die Implementierung von Standard-Features. Zudem ist durch die enge Verknüpfung der Software-Komponenten eine bessere und effizientere Zusammenarbeit möglich als bei einer stand-alone-Lösung.

Im folgenden Kapitel wird mit UNICORE eine existierende Grid-Middleware vorgestellt, die bereits modulare und erweiterbare Mechanismen zum Dateitransfer mitbringt. Es wird ein neuer Dateitransfermodus für UNICORE implementiert, der auf der vorgestellten Technik des UDP-Hole-Punching basiert und dem hier gezeigten Design folgt.

Kapitel 4

Implementierung des Verfahrens durch Integration in das UNICORE-System

4.1 Übersicht der UNICORE-Architektur

UNICORE [6] ist eine in Europa weit verbreitete Grid-Middleware, die zum großen Teil im Forschungszentrum Jülich als Open-Source Projekt entwickelt wird. Die Architektur der Software basiert auf einem modernen, modularen und erweiterbaren Design und auf verschiedenen offen Standards von diversen Standardisierungsorganisationen wie z.B. W3C, OASIS und IETF. Der wichtigste dieser Standards ist die „Open Grid Services Architecture“ (OGSA). Die wichtigsten Ziele beim Design der aktuellen Version 6 waren:

- Es soll ein kompletter und vollständig integrierter Software-Stack entwickelt werden. Dieser soll offen und erweiterbar sein und mit anderen Software-Komponenten interagieren können.
- Die Installation und Konfiguration der gesamten UNICORE-Software soll einfach und unkompliziert sein, um eine schnelle Inbetriebnahme zu ermöglichen.
- Es sollen nicht nur einfache Jobs verarbeitet werden können, sondern ganze Workflows. Diese werden vom Benutzer am Client erstellt.
- Es soll grafische Clients geben, die das Erstellen und Abschicken von Jobs und Workflows einfach und intuitiv ermöglichen.

Diese Anforderungen und das Erfüllen der diversen Standards führten zu einer Service-orientierten Struktur mit einem zentralen Gateway in jedem Grid. Der Zugriff auf Ressourcen im Grid geschieht bei UNICORE mittels WebServices. Dazu stellt jedes Server-System diverse Services zur Verfügung und publiziert diese in einer zentralen Registry. In einer solchen Registry werden alle nötigen Informationen über einen WebService gespeichert, wie z.B. ein Verweis auf dessen WSDL-Dokument[1]. Die Clients erfragen in der Registry die genauen Adressen der Services und schicken Nachrichten mit Anfragen und Aufträgen über den Gateway zu den Servern. Diese Nachrichten sind entfernte Methoden-Aufrufe, durch die neue WebService-Objekte auf den Servern erzeugt und mit Informationen versorgt werden. Die WebService-Objekte repräsentieren dann z.B. einen einzelnen Job, einen Workflow oder einen Dateitransfer. Die einzelnen Stufen der Kommunikation sind in Abbildung 4.1 vereinfacht dargestellt. Die verwendeten Techniken und die einzelnen Komponenten werden im Folgenden kurz beschrieben.

HTTP Das TCP-basierte HyperText Transfer Protocol (HTTP) dient zum Austausch von Nachrichten im Web. Es ist ein zustandsloses Protokoll, mit dem der Client eine Anfrage an den Server schickt und anschließend dessen Antwort erhält. UNICORE nutzt zum Austausch von HTTP-Nachrichten den WebServer Jetty 6.

XML Die eXtensible Markup Language (XML) ist ein Standard für eine einfache und flexible Auszeichnungssprache. Als Metasprache ist XML zur Strukturierung von Dokumenten und Daten geeignet und mittels Namensräumen und Schemata kann sie für Dokumente eines bestimmten Typs angepasst werden. XML dient als Basis für die hier beschriebenen SOAP-Nachrichten.

[1] WSDL → WebService Definition Language

SOAP Das Simple Object Access Protocol (SOAP) ist ein auf XML basierendes Nachrichtenformat. SOAP ist system- und plattformunabhängig und wird von WebServices zur Übermittlung von Methodenaufrufen und deren Rückgabewerten eingesetzt. Eine SOAP-Nachricht besteht aus einem optionalen Header mit Authentifizierungs- und Routing-Informationen und einem Message-Body. Zur Übermittlung kann im Prinzip jedes Kommunikationsprotokoll verwendet werden, UNICORE nutzt hierzu HTTP. Verarbeitet werden diese Nachrichten von einer SOAP-Engine. Diese Engine, bei UNIIORE wird XFIRE eingesetzt, codiert und decodiert die Nachrichten und reicht sie an die WebService-Komponente weiter.

WSRF Ein WebService wird von einem Service-Container verwaltet. Da die WebService-Objekte genau wie die Kommunikation zustandslos sind, ist ein solcher Container nötig, um den Clients einen konsistenten und zustandsbehafteten Zugriff auf die Objekte zu bieten. Diese Verwaltung übernimmt bei UNICORE das WebService Resource Framework (WSRF).

JEE5 UNICORE ist vollständig in Java implementiert. Dadurch wird es system- und plattformunabhängig und es können viele Features der Java Platform Enterprise Edition 5 (JEE5) genutzt werden, wie z.B. die Java Management Extensions.

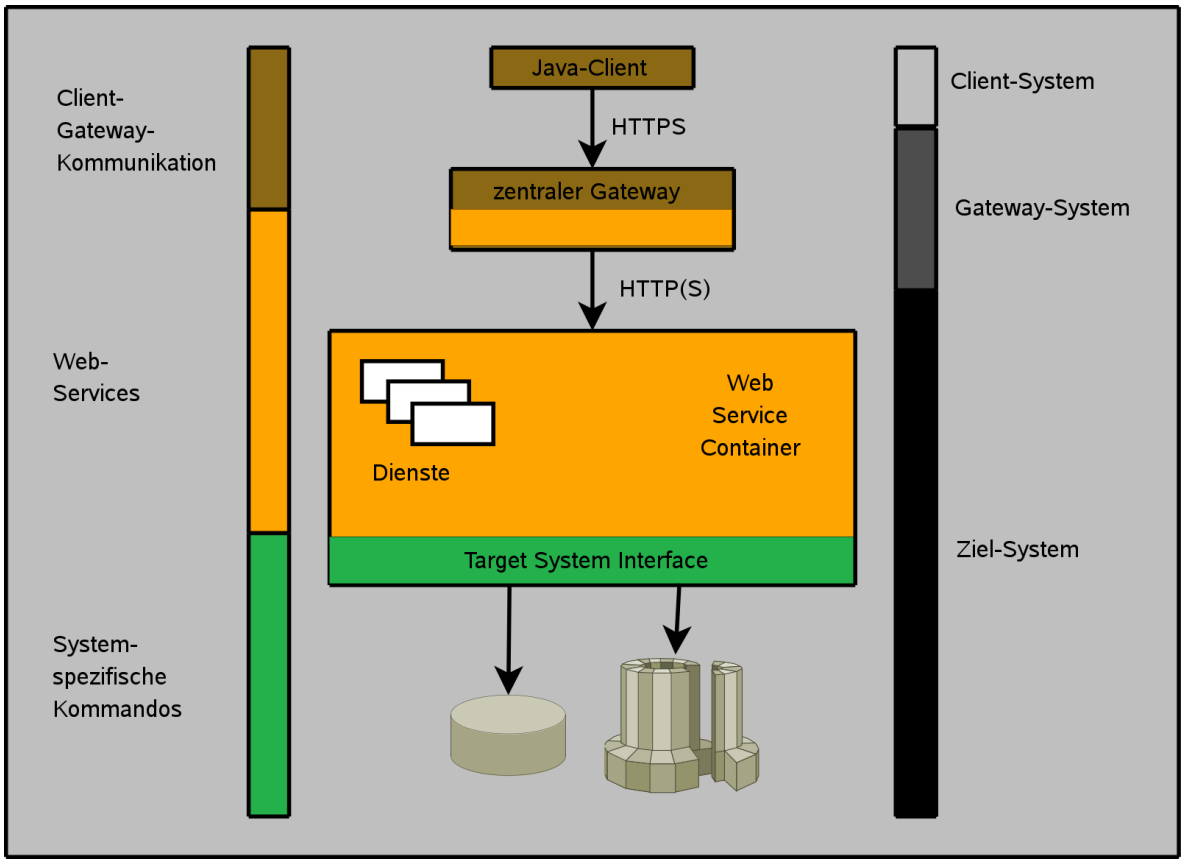

Abbildung 4.1: UNICORE Übersicht

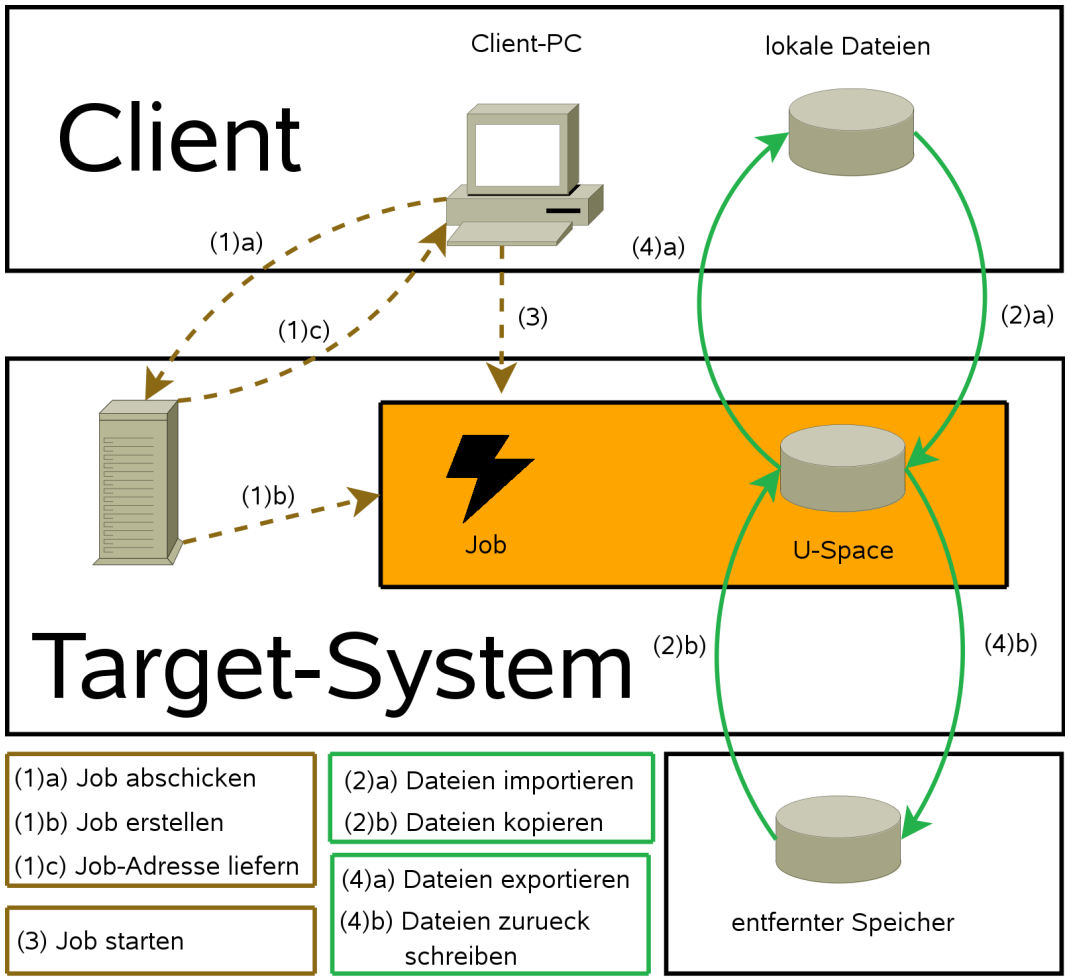

Abbildung 4.2: UNICORE Abläufe

Der Gateway und die Server bieten bereits ein umfangreiches Rechtemanagement inklusive Authentifizierung, Autorisierung und Verschlüsselung mittels Zertifikaten. Die Clients sprechen die Server über die bereit gestellten WebServices an. Name und Adresse dieser Services können von der zentralen Registry bezogen werden. Die wichtigsten Services einer UNICORE-Installation, die sogenannten Atomic Services, bieten Zugriff auf die Zielsysteme und deren Dateisysteme und bieten die Möglichkeit Jobs zu erstellen und zu verwalten. Zudem gehören Datei-Import und -Export zu den Atomic Services.

Wenn eine Datei zum oder vom Server kopiert werden soll, schickt der Client eine entsprechende SOAP-Nachricht und auf dem Server wird ein Dateitransfer-Objekt erstellt. Der Client erhält als Antwort einen Verweis auf dieses Objekt, so dass er es mit weiteren Nachrichten direkt ansprechen kann. Es können dann durch die Methoden des Objektes weitere Informationen und Daten ausgetauscht werden. Nach Beendigung des Transfers wird das Transfer-Objekt wieder zerstört. Dieser Ablauf wiederholt sich für jede einzelne Datei. Wann und wie solche Dateitransfers stattfinden zeigt Abbildung 4.2.

Ein Nutzer erstellt einen Job, bekommt einen Verweis auf das dafür erstellte Objekt und kann dann Dateien vom Client oder einem anderen entfernten Speicher des Grids importieren. Dazu wird für jeden Job ein sogenannter User Space (USpace) angelegt, der die Programme, Scripte und Datendateien enthält. Nach Beendigung des Jobs werden die Ergebnisse wieder zum Client oder anderen Speichern exportiert.

Die in UNICORE verfügbaren Dateitransfer-Methoden basieren alle auf einer Basis-Klasse, die die grundlegenden Funktionen eines WebService zur Verfügung stellt und den Austausch grundlegender Daten sowie eine Fehlerbehandlung bereitstellt. Zu jedem WebService gibt es eine entsprechende Klasse für den Client, die den Austausch der SOAP-Nachrichten übernimmt. Ein konkreter Service zum Dateitransfer wird von diesen Klassen abgeleitet. Dabei können weitere Methoden definiert werden, um für das Verfahren spezifische Informationen auszutauschen. Es gibt für UNICORE bereits mehrere dieser Dateitransfer-Klassen, von denen aber nur eine in der Praxis benutzt wird. Es handelt sich dabei um das ByteIO-Verfahren, ein einfaches, aber flexibles und sicheres Verfahren, das die zu übertragenden Daten in SOAP-Nachrichten verpackt und mit Methodenaufrufen versendet. Die Daten gehen hier den gleichen, verschlüsselten Weg zwischen Client, Gateway und Server wie die Kontrollinformationen. Der größte Nachteil dieses Verfahrens ist der mit maximal 400 KB/s [6] sehr geringe Durchsatz.

Im folgenden Abschnitt wird die Implementierung einer Alternative zu ByteIO besprochen. Diese leitet sich von der bereits erwähnten Basis-Klasse ab und setzt ein eigenes, auf den Erkenntissen der letzten Kapitel beruhendes Verfahren um.

4.2 Der neue Dateitransfer

Im Rahmen dieser Arbeit wird ein neuer Dateitransfer für UNICORE implementiert. Für den neuen Dateitransfer stellt UNICORE eine optimale Umgebung dar. Das Einbinden in UNICORE stellt durch das modulare Klassen-System kein Problem dar und viele der in Kapitel 3 beschriebenen Konzepte werden von UNICORE bereits umgesetzt. So gibt es einen zentralen Gateway, über den alle Kontrollinformationen ausgetauscht werden können. Dieser Austausch wird mit den beschriebenen X.509-Zertifikaten verschlüsselt. Benutzer und Systeme werden nicht bei jedem Dateitransfer authentifiziert und autorisiert, sondern einmalig vom Gateway. Die Verwaltung der WebService-Objekte geschieht durch den Service-Container, und die grundlegenden Funktionen, wie z.B. Übermittlung des Dateinamens, sind in einer allgemeinen Transfer-Klasse bereits implementiert. Die wirkliche Programmierarbeit beschränkt sich damit auf das Ableiten einer neuen Klasse von der allgemeinen Transfer-Klasse und das Implementieren der für das Verfahren spezifischen Funktionen.

Nachdem auf Client- und Server-Seite die Transfer-Objekte erstellt wurden, besteht der Ablauf des Verfahrens im Wesentlichen aus den in Abbildung 4.3 dargestellten Schritten. Die Funktion run() des Client wird aufgerufen und der Client bereitet den UDP-Port für den späteren Datenaustausch vor. Seine Verbindungsdaten schickt er als Parameter der Methode initUDT() an den Server. Dieser bereitet seinen eigenen UDP-Port vor und führt das UDP-Hole-Punching aus. Seine Verbindungsdaten sendet der Server als Rückgabewert der Methode zurück an den Client. Dieser kann nun eine UDT-Verbindung zum Server aufbauen, welcher am vorbereiteten Port mit einem UDT-ServerSocket auf die eingehende Verbindung wartet.

Eine Besonderheit der Implementierung bei diesem Verfahren ist die Tatsache, dass der WebService-Teil in Java geschrieben werden muss, während UDT in C++ geschrieben ist. Die-

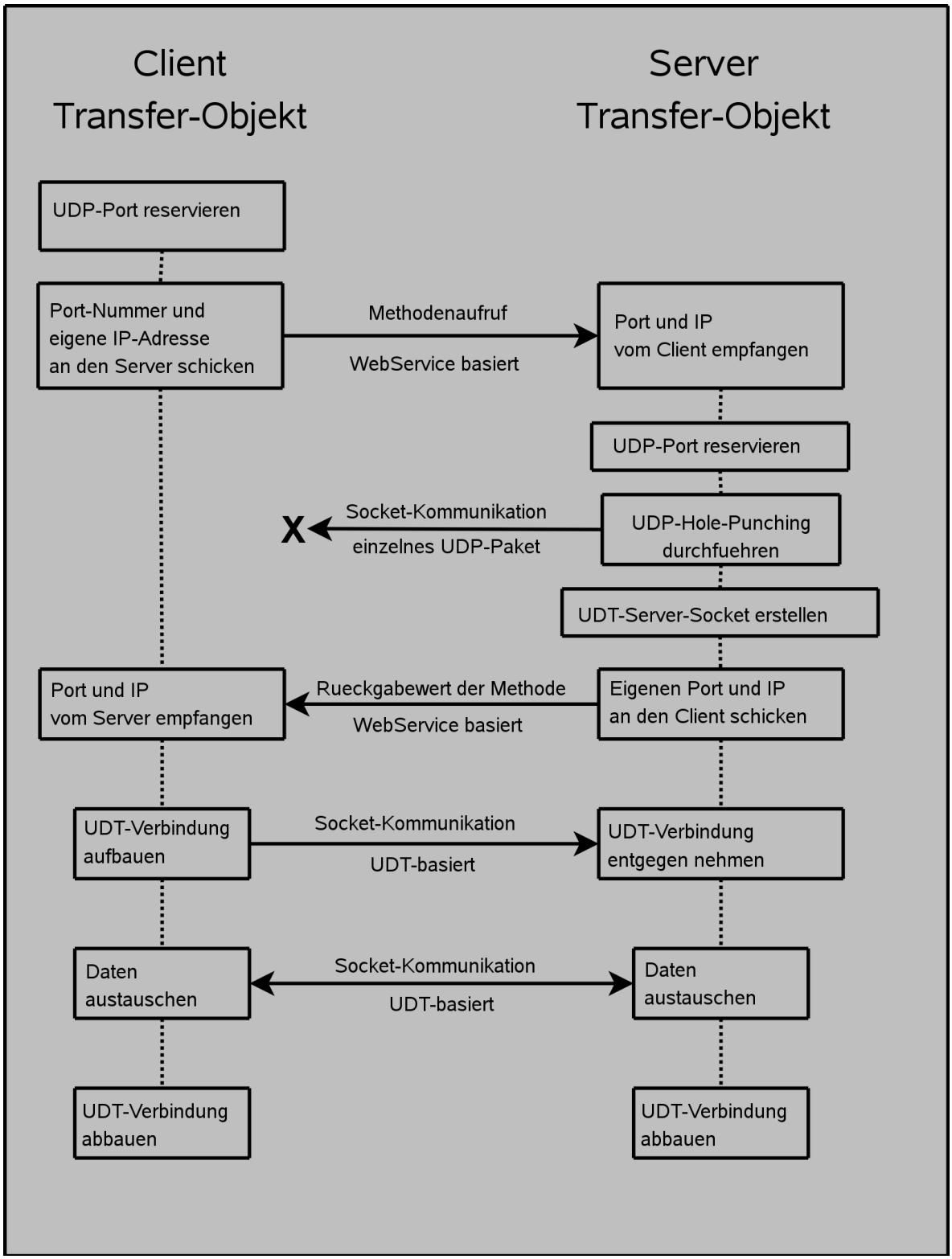

Abbildung 4.3: Ablauf des UNICORE-Dateitransfers

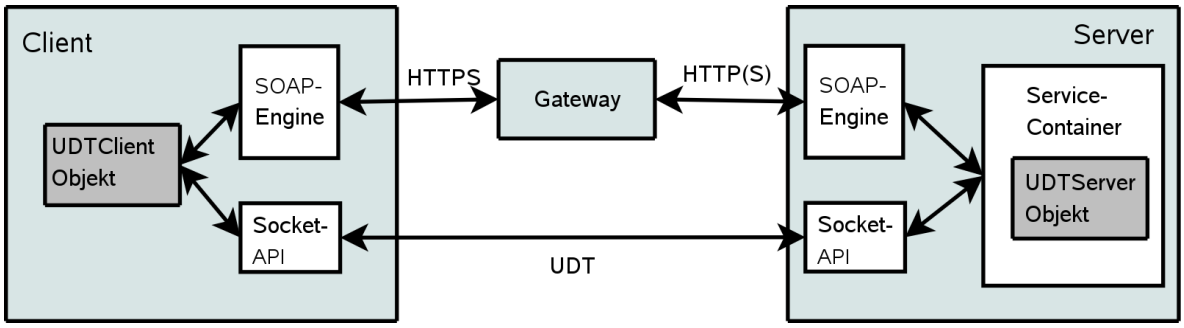

Abbildung 4.4: Integration der neuen Klassen in UNICORE

jenigen Teile der Implementierung, die UDT-Funktionen nutzen und das UDP-Hole-Punching durchführen, wurden daher in eine dynamische, in C++ implementierte Bibliothek ausgelagert. Die Java-Klassen greifen mit Hilfe des Java Native Interface (JNI) auf diese Bibliothek zu. JNI bietet die Möglichkeit, dynamische native Bibliotheken zu laden und darin enthaltene Funktionen von JAVA aus aufzurufen. Die Parameter der nativen Funktionen müssen dabei einer von JNI festgelegten Systematik folgen, damit sie Referenzen auf das aufrufende JAVA-Objekt entgegen nehmen können. Somit kann von der nativen Funktion aus auf die Methoden und Variablen des aufrufenden Objekts zugegriffen werden. Wer den neuen Dateitransfer einsetzt, muss darauf achten, dass die native Bibliothek für die eingesetzte Hardware und die Betriebssysteme verfügbar ist. Die folgenden Nassi-Schneidermann-Diagramme zeigen die Vorgänge in den einzelnen Methoden und Funktionen der Implementierung. Das sind die Methoden der Server-Klasse „UDTFileTransferImpl“ und der Client-Klasse „UDTClient“. Zudem benutzt der Server eine Klasse „UDTSession“ um die Verbindungsdaten zu verwalten. Der native Code wurde auf vier Funktionen aufgeteilt, je zwei für Client und Server. Die einzelnen Schritte können in vier Kategorien eingeteilt werden:

A Aufruf einer WebService-Methode bzw. Versand das Rückgabewertes einer WebServivce-Methode

B Aufruf einer nativen Methode von einer Java-Methode aus

C nativer C++-Code

D reiner Java-Code

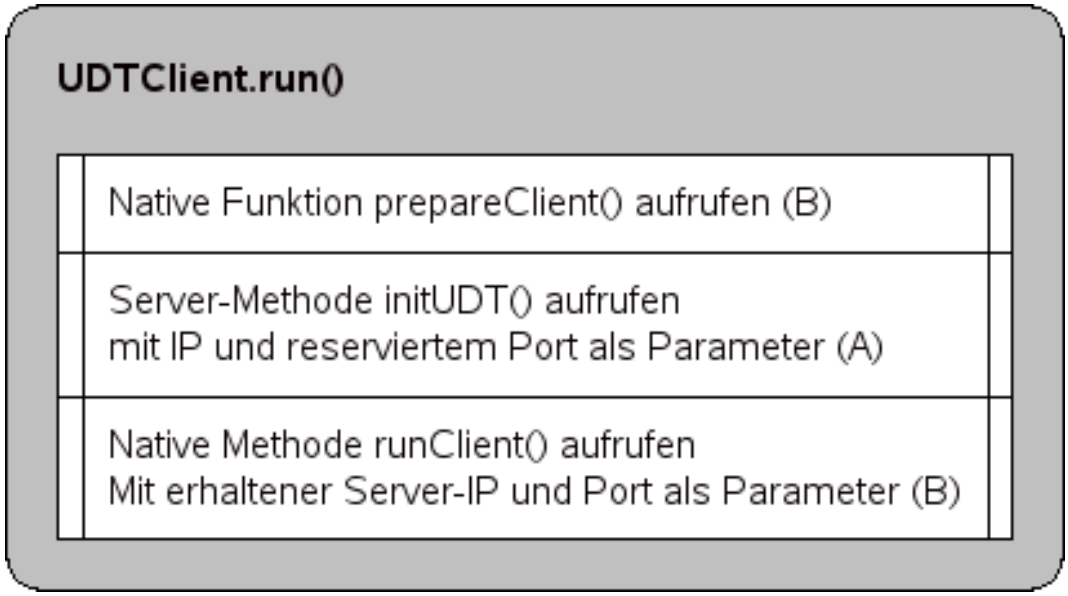

Abbildung 4.5: Methode UDTClient.run()

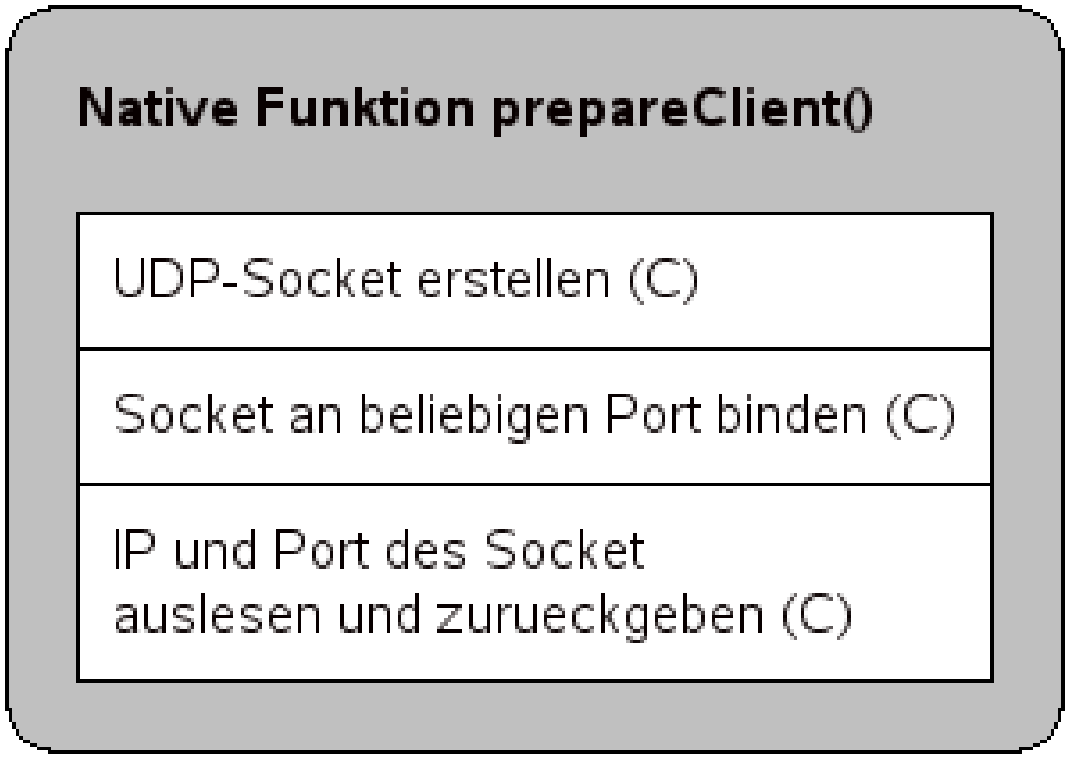

Abbildung 4.6: Funktion prepareClient()

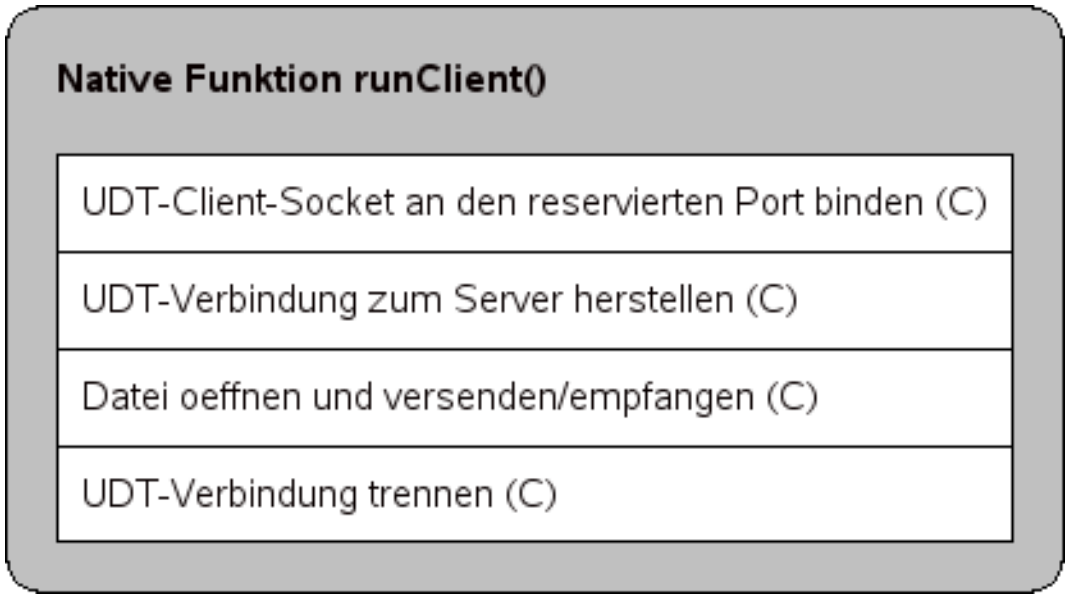

Abbildung 4.7: Funktion runClient()

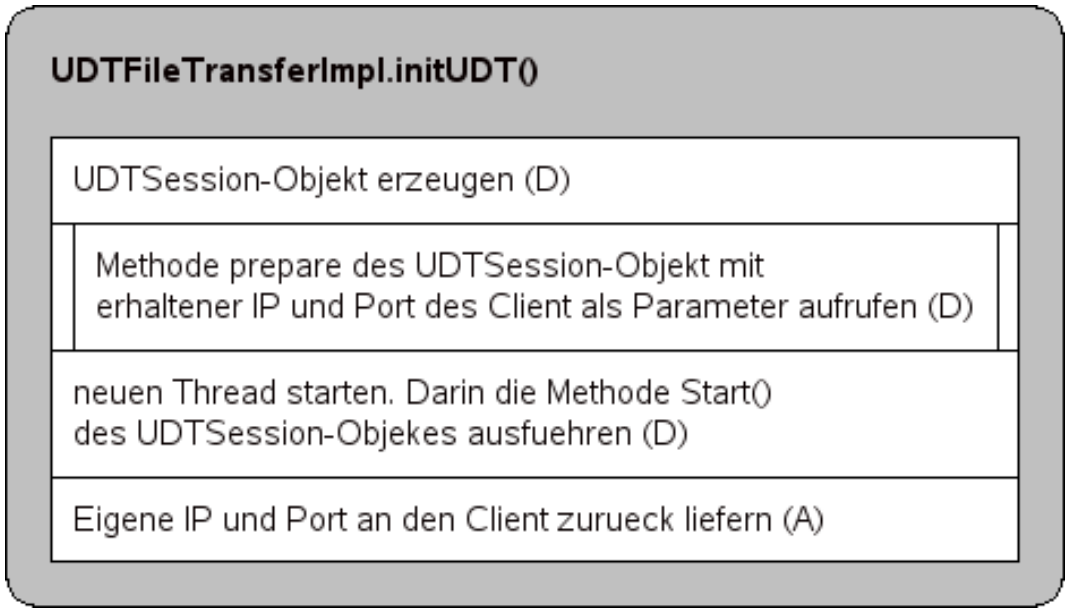

Abbildung 4.8: Methode UDTFileTransferImpl.initUDT()

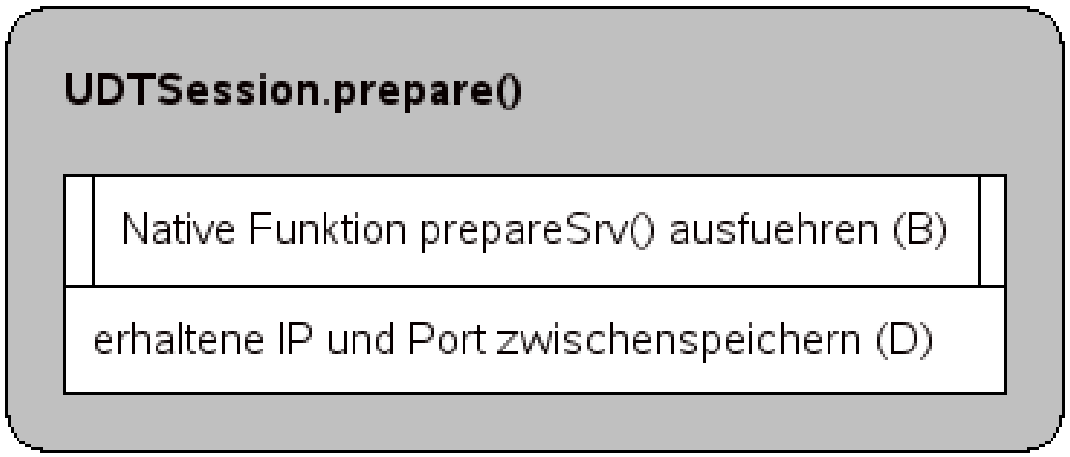

Abbildung 4.9: Methode UDTSession.prepare()

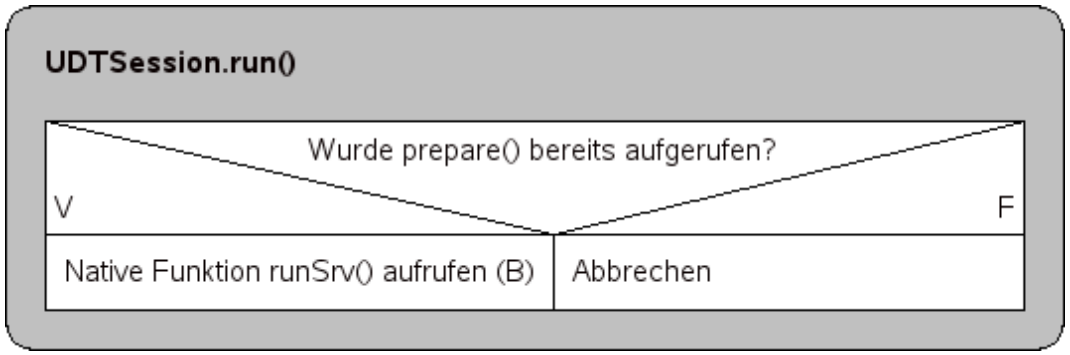

Abbildung 4.10: Methode UDTSession.run()

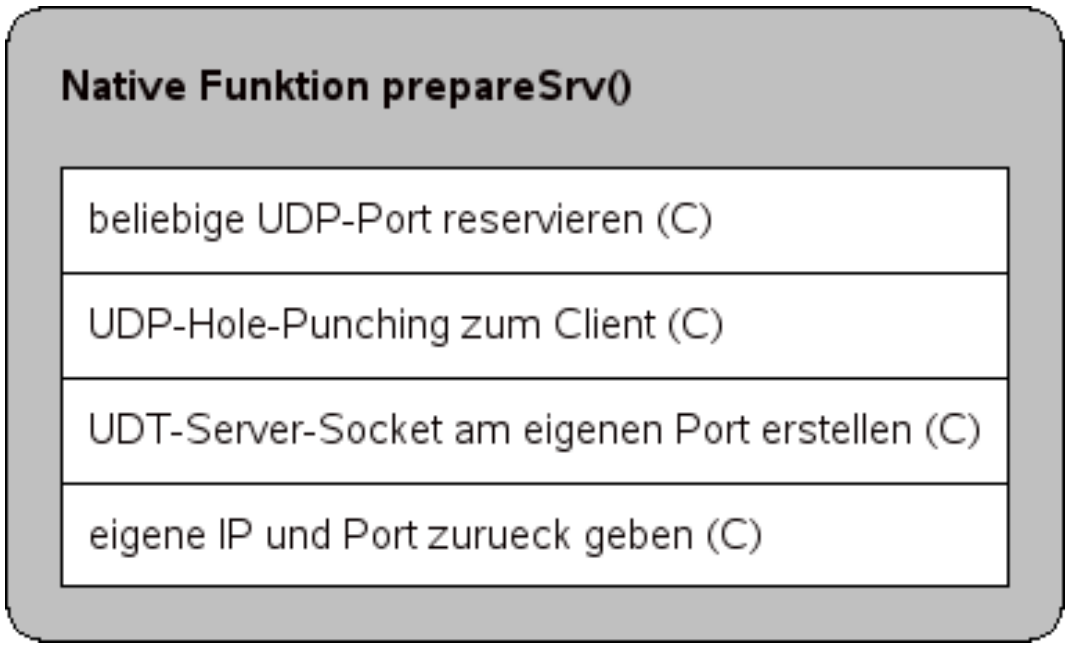

Abbildung 4.11: Funktion prepareSrv()

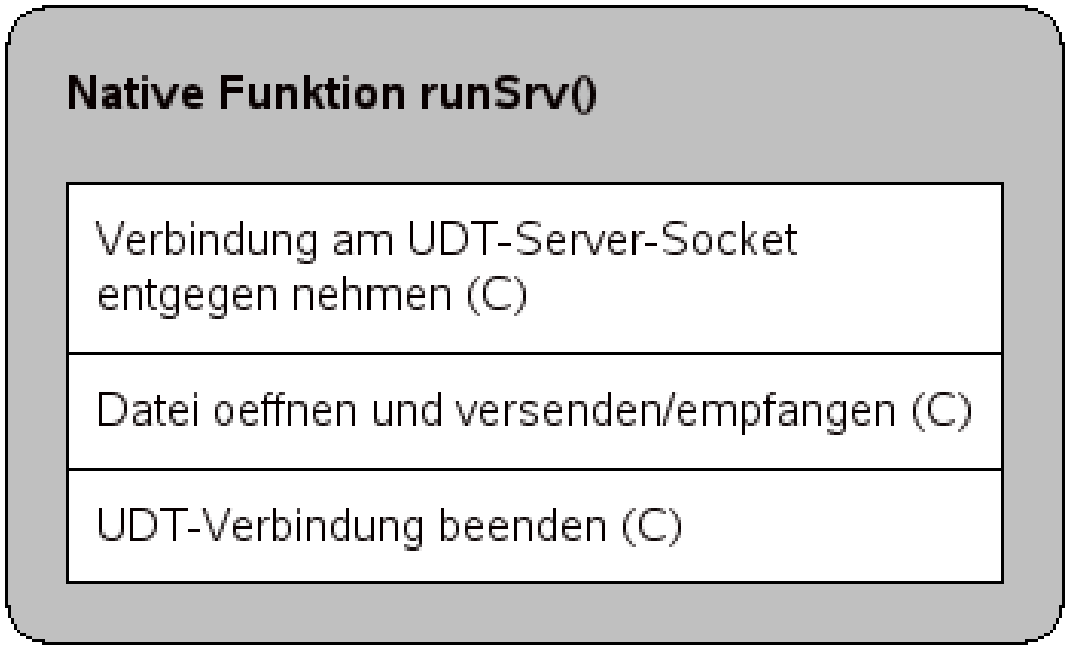

Abbildung 4.12: Funktion runSrv()

Kapitel 5

Vergleich des Verfahrens mit bestehenden Dateitransfer-Lösungen

5.1 GridFTP

GridFTP [7] ist ein auf FTP basierendes Dateitransfer-Protokoll. Es überträgt Dateien zwischen Client und Server oder zwischen zwei Servern über TCP-Verbindungen. Dazu werden Kontrollverbindungen und davon unabhängige Datenverbindungen aufgebaut. GridFTP wurde speziell für den Einsatz in Grid-Umgebungen entwickelt und unterscheidet sich daher in einigen Punkten von FTP. Die Kontrollverbindung wird verschlüsselt, während bei FTP alle Informationen im Klartext übertragen wurden. Das gewährleistet die in Grids geforderte Sicherheit. Außerdem werden für einen Dateitransfer nicht nur einer, sondern mehrere Datenkanäle geöffnet. Die zu übertragende Datei wird dann auf die parallelen Verbindungen aufgeteilt. Das erhöht die Durchsatzrate im Vergleich zu normalem FTP erheblich und macht den Transfer weniger anfällig für Störungen im Netz. GridFTP beherrscht „commandline pipelining“ und kann so mehrere Aktionen hintereinander ausführen ohne neue TCP-Verbindungen aufbauen zu müssen. Bekannte Grid-Middleware wie z.B. Globus nutzt GridFTP als Verfahren für Dateitransfers.

5.2 UNICORE-ByteIO

ByteIO ist das Standardverfahren zur Übertragung von Dateien in der Grid-Middleware UNICORE. Es nutzt die in UNICORE ohnehin vorhandene Möglichkeit, Informationen über Methodenaufrufe an WebServices zu übertragen. Dadurch braucht ByteIO keine eigenen Verbindungen zwischen den Endsystemen. Alle Daten werden in kleine Portionen aufgeteilt und als Parameter eines Aufrufs versendet. Dazu werden sie in eine SOAP-Nachricht verpackt und mit HTTP an den Gateway verschickt. Dieser leitet die Nachricht wiederum über HTTP an das Zielsystem weiter. So nutzt ByteIO automatisch die Verschlüsselung und Authentifizierung von UNICORE.

5.3 Vergleich

Durchsatz

Wie im vorhergehenden Kapitel bereits erwähnt, ist ByteIO mit maximal 400 KB/s sehr langsam. Die diversen Stufen der WebService-Architektur erfordern aufwändige Schritte zum Erstellen und Weiterleiten der Nachrichten, sowie anschließenden Auspacken der Daten. Die Durchsatzrate ist daher nicht durch das genutzte Netz begrenzt, sondern durch die verwendeten Protokolle.

GridFTP wurde unter anderem entwickelt, um die Durchsatzsprobleme von FTP zu umgehen. Beide benutzen TCP-Verbindungen. Die Nachteile von TCP in Bezug auf die Auslastung der Netzwerkhardware wurden bereits in Kapitel 3 behandelt. GridFTP erreicht hohe Netzauslastung, indem es mehrere TCP-Ströme gleichzeitig nutzt und die Daten gleichmäßig darauf aufteilt. Es kann insgesamt als effizient und schnell bezeichnet werden, die Verwendung von vielen Verbindungen geht allerdings auf Kosten der Fairness zum restlichen Datenverkehr im Netz. Ferner sinkt die Effizienz bei einer zu großen Anzahl von parallelen Verbindungen, da der steigende Verwaltungsaufwand den Durchsatz behindert.

Das Verfahren des UDP-Hole-Punching hat auf die Transfergeschwindigkeit keinen direkten Einfluss. Der Durchsatz und die Fairness werden nur durch das UDT-Protokoll bestimmt. Durch die modulare Staukontrolle ist UDT an jeden beliebigen Einsatzzweck anpassbar. Es

bietet bei Bedarf die nötige Fairness zu anderen UDT- oder TCP-Verbindungen und kann die zur Verfügung stehende Bandbreite optimal ausnutzen. Die folgende Tabelle zeigt die durchschnittlichen Ergebnisse der Durchsatzmessungen, die für UDT und GridFTP durchgeführt wurden. Die Messungen fanden im X-WIN-Netz des DFN zwischen der Universität Hannover und dem Forschungszentrum Jülich statt, an das die Endsysteme mit 1 GBit/s angebunden waren. Für UDT wurde die standardmäßig integrierte Staukontrolle genutzt.

GridFTP (1 Stream)	GridFTP (4 Streams)	UDT 3	UDT 4
81 MBit/s	294 MBit/s	700 MBit/s	930 MBit/s

Tabelle 5.1: Geschwindigkeiten der verschiedenen Transfer-Protokolle

Durch den Einsatz von UDT wurden parallel laufende TCP-Ströme stark unterdrückt. Im praktischen Einsatz sollte die Staukontrolle modifiziert werden, so dass eine ausreichende Fairness erreicht wird.

Sicherheit

ByteIO hat in Bezug auf die Sicherheit den großen Vorteil, dass es keine eigenen Verbindungen zwischen den Kommunikationspartnern braucht. In den Firewalls der beteiligten Organisationen brauchen daher für den Dateitransfer keine Rechner zugänglich gemacht zu werden. Die verwendete Kommunikation über WebServices ist für UNICORE ohnehin nötig. ByteIO stellt die geringsten Vorraussetzungen an die Netzwerk-Konfiguration und ist somit das sicherste Verfahren.

Der Einsatz von GridFTP erzwingt eine sehr unsichere Netzkonfiguration. Da die Kontrollverbindung im Gegensatz zu FTP verschlüsselt ist, können Firewalls den Datenverkehr nicht mitlesen und somit auch die Datenverbindungen keiner Kontrollverbindung zuordnen. Während bei FTP die Firewalls den Kontrollverkehr mitlesen können und daher Datenverbindungen bezogen auf die übermittelten Kontrollinformationen dynamisch an der Firewall freischalten können, ist das bei GridFTP nicht möglich. Es müssen an allen Systemen, die GridFTP nutzen, die verwendeten Ports von außen zugänglich sein. Durch die vielen parallelen Verbindungen sind sehr große Port-Bereiche betroffen. Wer GridFTP nutzt, muss an allen Systemen im Grid viele Ports für TCP-Verbindungen öffnen. Das macht GridFTP zu einem Sicherheitsrisiko.

Das UDP-Hole-Punching ist sicherer als GridFTP, weil keine eingehenden Verbindungen erlaubt werden müssen. Nur ausgehender UDP-Verkehr muss für die Grid-Komponenten zugelassen sein. In den meisten Einsatzszenarien ist das ein akzeptabler Zustand, der die internen Netze nicht übermäßig gefährdet. UDP-Hole-Punching kann als ausreichend sicher bezeichnet werden.

Ergebnis

Es hat sich gezeigt, dass beide bisher zur Verfügung stehenden Protokolle zum Dateitransfer in Grid-Systemen große Nachteile haben. ByteIO ist zwar ein sicheres und zuverlässiges Protokoll, doch der geringe Durchsatz macht es sehr unattraktiv. GridFTP nutzt die vorhandene Bandbreite gut aus, aber es erfordert eine sehr unsichere Konfiguration der Firewall. Damit wird sein Einsatz in kommerziell genutzten Grids und bei großen Organisationen nahezu

unmöglich.

Das Verfahren des UDP-Hole-Punching in Verbindung mit dem UDT-Protokoll hat sich als gute Alternative zu diesen beiden Protokollen herausgestellt. Es erlaubt eine Firewall-Konfiguration, die für die meisten Organisationen ausreichend sicher ist und bietet optimale Übertragungsraten. Da nur Standard-Technologien benutzt werden, ist das Verfahren in nahezu jede Software integrierbar, unabhängig von verwendeter Hardware, Betriebssystemen und dem Zweck des Datenaustausches.

Kapitel 6

Fazit

Zusammenfassung

Ziel dieser Arbeit war es, einen zuverlässigen, sicheren und schnellen Dateitransfer zum Einsatz in Grid-Systemen zu entwickeln. Dazu konnte das Verfahren des UDP-Hole-Punchings für den Einsatz in Grids angepasst werden. Seine Funktionstüchtigkeit wurde in Kapitel 2 gezeigt. Mit UDT wurde zudem ein Protokoll gefunden, das die nötige Zuverlässigkeit bietet. Zu einer umfassenden Lösung gehören aber auch noch andere Aspekte wie z.B. Rechteverwaltung. Daher wurde in Kapitel 3 das Design eines kompletten Software-Paketes entwickelt. Dass die vorgestellten Verfahren praxistauglich sind, zeigt die unkomplizierte Integration in die bestehende Software UNICORE.

Sicherheit

Das neue Verfahren bietet den großen Vorteil, dass es direkte Verbindungen zwischen den Kommunikationspartnern erlaubt, ohne die eigenen Systeme dauerhaft für externe Rechner erreichbar zu machen. Die erlaubten Verbindungen werden dynamisch, automatisch, sehr genau und nur temporär festgelegt. Dies ist die sicherste, zur Verfühgung stehende Methode, Dateitransfers direkt zwischen zwei Endsystemen zu ermöglichen.

Durchsatz

Die Verwendung von UDP als Basis macht eine UDT-Verbindung schneller als eine TCP-Verbindung. Durch die Modularität kann UDT zusätzlich an spezielle Szenarien angepasst werden. Damit ist UDT in den meisten Anwendungsfällen das schnellste Verfahren zum Datentransfer.

Ausführliche Tests haben aber gezeigt, dass die Durcshatzrate durch die Nutzung mehrerer paralleler UDT-Verbindungen weiter gesteigert werden könnte. Das ist dann der Fall, wenn der Dateitransfer eine Netzwerkleitung ganz für sich alleine hat und die verwendeten Router und Firewalls intern parallelisierte Hardware benutzen. Die Parallelisierung geschieht meist auf Verbindungsebene, weswegen bei nur einer aktiven Verbindung ein Teil der Hardware ungenutzt bleibt. Parallele Ströme könnten die Hardware in diesem speziellen Fall besser auslasten, wie es auch bei GridFTP geschieht.

Portabilität

Die vorgestellte Implementierung ist vollständig in die Software UNICORE integriert. Es ist aber nötig, die Bibliothek mit nativem Code für jedes verwendete Betriebssystem und jede Hardware-Plattform neu zu übersetzen.

Um das Verfahren noch flexibler zu machen, könnte in Zukunft eine Alternative zu JNI gesucht werden. Der Einsatz dieses Wrappers für nativen Code widerspricht der System- und Plattformunabhängigkeit von UNICORE. Diese ist aber ein wichtiges Feature. Eine Portierung von UDT nach JAVA erscheint daher interessant.

Ergebnis

Mit dem neuen Dateitransfer ist eine Lösung verfügbar, die die wichtigsten Vorteile der bisher genutzten Protokolle vereint, ohne ihre Schwächen zu übernehmen. Ein Abwägen zwischen Sicherheit und Geschwindigkeit ist nicht mehr nötig.

Anhang A

Protokoll des Testlaufs

1	0,000000	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [SYN]	Seq=0 Ack=0
2	0,000310	10.0.1.4	10.0.0.6	TCP	17070 > 52376 [SYN ACK]	Seq=0 Ack=1
3	0,000358	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [ACK]	Seq=1 Ack=1
4	1,433938	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [PSH ACK]	Seq=1 Ack=1
5	1,434091	10.0.1.4	10.0.0.6	TCP	17070 > 52376 [ACK]	Seq=1 Ack=16
6	1,434221	10.0.1.4	10.0.0.6	TCP	17070 > 52376 [PSH ACK]	Seq=1 Ack=16
7	1,434239	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [ACK]	Seq=16 Ack=10
8	1,434274	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [PSH ACK]	Seq=16 Ack=10
9	1,434622	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
10	1,434824	10.0.1.4	10.0.0.6	TCP	17070 > 52376 [PSH ACK]	Seq=10 Ack=25
11	1,434878	10.0.0.6	10.0.1.4	UDP	Source port: 32988	Destination port: 33101
12	1,436355	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
13	1,437800	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [PSH ACK]	Seq=25 Ack=17
14	1,475221	10.0.1.4	10.0.0.6	TCP	17070 > 52376 [ACK]	Seq=17 Ack=32
15	1,475244	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [PSH ACK]	Seq=32 Ack=17
16	1,475379	10.0.1.4	10.0.0.6	TCP	17070 > 52376 [ACK]	Seq=17 Ack=45
17	1,475526	10.0.1.4	10.0.0.6	TCP	17070 > 52376 [PSH ACK]	Seq=17 Ack=45
18	1,515616	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [ACK]	Seq=45 Ack=26
19	1,755636	10.0.0.6	10.0.1.4	UDP	Source port: 32988	Destination port: 33101
20	2,079234	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
21	2,079319	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
22	2,079343	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
23	2,079369	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
24	2,079374	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
25	2,079379	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
26	2,079471	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
27	2,079477	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
28	2,079482	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
29	2,079486	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
30	2,079491	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
31	2,079496	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
32	2,079503	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
33	2,079507	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
34	2,079522	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
35	2,079526	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
36	2,079531	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
37	2,095676	10.0.0.6	10.0.1.4	UDP	Source port: 32988	Destination port: 33101
38	2,095812	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
39	2,095875	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
40	2,095901	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
41	2,095909	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
42	2,095914	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
43	2,095946	10.0.0.6	10.0.1.4	UDP	Source port: 32988	Destination port: 33101
44	2,096073	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
45	2,096090	10.0.0.6	10.0.1.4	UDP	Source port: 32988	Destination port: 33101
46	2,096100	10.0.1.4	10.0.0.6	UDP	Source port: 33101	Destination port: 32988
47	2,096186	10.0.0.6	10.0.1.4	TCP	52376 > 17070 [FIN ACK]	Seq=45 Ack=26
48	2,135236	10.0.1.4	10.0.0.6	TCP	17070 > 52376 [ACK]	Seq=26 Ack=46

Abbildung A.1: Mitschnitt des Netzwerkverkehrs eines Dateitransfers

Literaturverzeichnis

[1] Cern, *Grid Cafe - The place for everybody to learn about the Grid*, gridcafe.web.cern.ch

[2] W. Richard Stevens, *UNIX Network Programming*, Prentice Hall PTR, Upper Saddle River, 1998

[3] S. Bon, B. Allcock, M. Livny, *CODO: Firewall Traversal by Cooperative On-Demand Opening*, 14th IEEE Symposium on High Performance Distributed Computing, Resaerch Triangle Park, Juli 2005, www.cs.wisc.edu/sschang/papers/CODO-hpdc.pdf

[4] Egon Gruenter, Markus Meier, Ralph Niederberger, Franz Petri, *Dynamic Configuration of Firewalls Using UDP Hole Punching*, D-Grid Integrationsbericht Fachgebiet 3-5, 2006

[5] Yunhong Gu, *UDT - UDP based Data Transfer: Breaking the Data Transfer Bottleneck*, udt.sourceforge.net, 2006

[6] UNICORE - Uniform Interface to Computing Resources, www.unicore.eu

[7] GridFTP - The globus alliance - www.globus.org/grid_software/data/gridftp.php