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

Egon Grünter, Markus Meier, Ralph Niederberger, Thomas Oistrez

{e.gruenter,m.meier,r.niederberger,t.oistrez}@fz-juelich.de

Abstract: Firewalls separieren Bereiche mit verschiedenen Sicherheitsanforderungen. Diese Hauptaufgabe führt zu Problemen in Bezug auf Erreichbarkeit und Geschwindigkeit bei diversen Anwendungen. Besonders bei verteilten Systemen, wie z.B. einem Grid, ist eine ungehinderte Kommunikation, welche für die Nutzung verteilter Ressourcen unerlässlich ist, nicht möglich. Zudem nutzen Grid-Anwendungen oft mehrere und dynamisch reservierte Ports parallel. Dies führt zu der Aufgabe, Firewalls dynamisch zu konfigurieren. Dieser Artikel zeigt eine auf UDP hole punching basierende Lösung und beschreibt die Implementierung eines UNICORE Service, welcher diese Technologie nutzt um schnelle, direkte Dateitransfers durchzuführen.

1 Firewalls und Filterung von Datenverkehr

Firewalls werden genutzt um Bereiche mit verschiedenen Sicherheitsanforderungen von einander zu trennen. Die Hauptaufgabe besteht im Schutz der Rechner-Ressourcen vor unerlaubtem Zugriff und Missbrauch. Zur Erfüllung dieser Aufgabe verarbeitet die Firewall verschiedene Informationen, um zu entscheiden, ob eine Nachricht die Firewall passieren darf oder nicht. Der Firewall-Administrator definiert ein Regelwerk, welches die Implementierung der jeweiligen Sicherheitsrichtlinie darstellt. Der Netzwerkverkehr wird in Klassen von Paketen aufgeteilt, die weitergeleitet werden, und solche, die zurückgewiesen werden. Dieses Regelwerk dient als Basis für verschiedene Tests, die jedes eingehende Paket durchläuft. Die Firewall überprüft IP Addressen und Port Nummern des jeweiligen Protokoll-Headers. Sogenannte Statefull Inspection Engines nutzen außerdem den Verbindungsstatus.

Firewalls speichern Statusinformation haupsächlich für TCP-Ströme da TCP ein verbindungsorientiertes Protkoll darstellt. Verbindungen könne vier verschiedene Zustände haben: halb offen, offen, halb geschlossen, geschlossen. Es macht Sinn, zwischen den aktuellen Zuständen einer Verbindung zu unterscheiden. Zudem ist TCP ein zuverlässiges Protokoll. Das bedeutet, dass ein Paket, welches durch fehlende Bestätigungen vom Empfänger als verlohren erkannt wurde, nochmals gesendet wird. Weitere Informationen hierzu finden sich in [Ste94].

Das User Datagramm Protocol (UDP) ist ein Transportprotokoll und ist weder zuverlässig noch verbindungsorientiert. Eine Anwendung, die UDP nutzt, muss sicherstellen, dass die Daten erfolgreich übertragen wurden. Obwohl bei UDP keine Verbindungen existieren,

nutzen Firewalls einen einfachen Mechanismus, um Verbindungen zu simulieren. Abbildung 1 zeigt einen Client hinter einer Firewall, der UDP Pakete nach außen senden darf und ein UDP Datagram an einen DNS Server außerhalb des Firmennetzwerkes sendet.

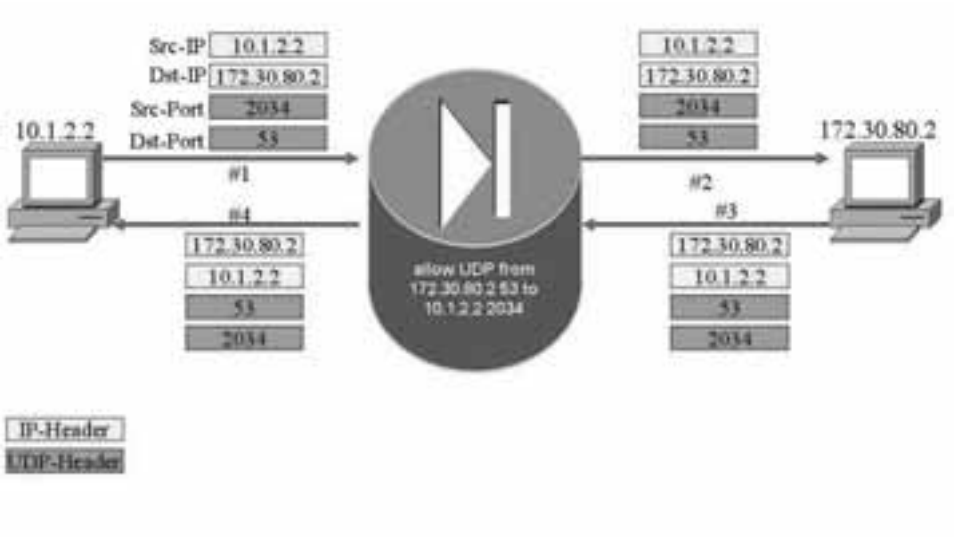

Abbildung 1: Firewalls und UDP

Der Client generiert das UDP Datagram und sendet es an die Firewall (#1). Die Firewall kontrolliert das Datagram und leitet es an das Ziel weiter (#2). Anhand der gesammelten Informationen fügt die Firewall einen Eintrag zur Verbindungstabelle hinzu, der die IP-Addressen und Port-Nummern von Quelle und Ziel enthält. Dies führt zu der folgenden, dynamisch generierten Zugangsregel, welche Antwortpakete des Servers an den Client erlaubt:

```
allow UDP from 127.30.80.2 port 53 to 10.1.2.2 port 2034
```

Diese ist für eine bestimmte, konfigurierbare Zeit gültig. Die Regel stellt sicher, dass innerhalb der festgelegten Zeit die Antworten vom Server (#3) die Firewall passieren können (#4).

2 Grid Anwendungen und Firewalls

Ein Grid ist ein verteiltes System, welches seinen Nutzern Zugriff auf Rechenzeit, Speicherplatz und verteilt vorliegende Daten bietet. Es bildet einen Bund aus verschiedenen, geographisch verteilten unabhängigen Organisationen, auch als virtuelle Organisation bezeichnet. Auf die verfügbaren Ressourcen kann statisch oder dynamisch zugegriffen werden, je nach Anforderung der Nutzer bzw. der Anwendung.

Viele Grid-Anwendungen benötigen hohe Bandbreiten und kurze Verzögerungszeiten. Zudem werden große Datenmengen übertragen, die das Ziel schnell und zuverlässig erreichen müssen. Ein Ansatz dazu ist die Nutzung mehrerer paralleler TCP-Verbindungen. GridFTP [web06] ist ein Beispiel hierfür: Es setzt voraus, dass mehrere Verbindungen zwischen Sender und Empfänger hergestellt werden. Das Protokoll sieht vor, dass ein Server, der GridFTP nutzt, auf einer Vielzahl von Ports von außen erreichbar ist. Das statische und somit langfristige Öffnen großer Portbereiche (in der Regel 5000 Ports) an der Firewall führt zu ernsten Sicherheitsproblemen, da die Ports von bösartiger Software genutzt werden können, während GridFTP sie nicht verwendet. GridFTP ist ein schnelles Verfahren, aber durch die statische Port-Öffnung an der Firewall ist seine Verwendung in den meisten produktiven Umgebungen nicht gerne gesehen.

Zur Erhöhung der Sicherheit scheint eine dynamische Konfiguration von Firewalls Sinnvoll. Diese sollte folgende Anforderungen erfüllen:

1. Sie kann reibungslos in die existierende Sicherheitsrichtlinien integriert werden.
2. Sie kann in Open-Source und kommerziellen Produkten genutzt werden.
3. Sie erlaubt die Kommunikation zwischen den beiden Partnern nur für die minimal nötige Dauer.

Zur Zeit gibt es verschiedene Lösungen um eine Firewall dynamisch zu konfigurieren. Diese sind entweder proprietär und herstellerspezifisch wie in den Cisco PIX Produkten [Cis], oder sie unterstützen nur bestimmte Arten von Firewalls. CODO [SAL05] ist eine solche Lösung. Neue universielle Ansätze zur Konfiguration und neue Protokolle zur Signalisierung befinden sich in der Entwicklung, sind aber noch nicht fertiggestellt bzw. noch unzureichend verbreitet. Dieser Artikel stellt einen neuen Ansatz zur dynamischen Firewall-Konfiguration vor. Er nutzt den Mechanismus des UDP hole punching, einer üblichen Technik im NAT [Sch06] Bereich. Der Ansatz ist leicht zu implementieren und mit allen zur Zeit üblichen Firewalls sofort einsetzbar. Da das Verfahren nur mit UDP funktioniert, und somit in Kombination mit etablierten Protokollen wie GridFTP nicht funktioniert, ist auch die Wahl eines geeigneten Transfer-Protokolls nötig.

3 UDP hole punching

UDP hole punching ist eine Methode zur Herstellung bidirektionaler UDP Verbindungen zwischen zwei durch Firewalls getrennte Internet Hosts. Das Verfahren basiert auf der Simulation von UDP Verbindungen durch Firewalls. Eingesetz wird UDP hole punching bereits seit längerem in VoIP- und P2P-Software. Hier dient es der Herstellung von direkten Verbindungen zwischen Endpunkten, die sich oft hinter NAT-Routern befinden. NAT-Router sind nur schwierig dynamisch zu konfigurieren, da sie die Portnummern und IP-Adressen der weitergereichten Pakete ändern und die Endsysteme somit ihre eigenen Verbindungsdaten nicht kennen. Im Grid-Umfeld spielt NAT keine besondere Rolle, so dass das Verfahren hier zu der im folgenden beschriebenen Methode vereinfacht werden

kann, was einerseits der Robustheit und der Sicherheit zu Gute kommt, andererseits einen Einsatz mit NAT-Routern aber unmöglich macht.

Es gelten folgende Voraussetzungen zur Nutzung von UDP hole punching:

- Die lokale Firewall erlaubt ausgehende UDP Verbindungen
- Die lokale Firewall simuliert diese UDP Verbindungen wie in Kapitel 1 beschrieben
- Es existiert ein zentraler Relay Server

Der zentrale Server ist ein wesentlicher Bestandteil des Konzeptes. Jeder Client verbindet sich zu diesem Server mit einer dauerhaften TCP Verbindung. Im Folgenden wird beispielhaft beschrieben, wie zwei Hosts aus verschiedenen Sicherheitsbereichen eine Verbindung mittels UDP hole punching herstellen (Abbildung 3). Der Initiator (Client A) sendet eine Nachricht an den Relay Server (Nachricht #1). Diese enthält die Information, das Client A mit Client B kommunizieren will und dafür einen bestimmten UDP Port, z.B. 4711, nutzen wird. Der Server benachrichtigt Client B über den Verbindungswunsch und übersendet die IP Addresse und den UDP Port von Client A (#2). Client B sendet seinen eigenen UDP Port, z.B. 8822 an den Server und sendet gleichzeitig ein erstes UDP Paket von Port 8822 an den Port 4711 des Client A (#3).

Die lokale Firewall von Client B leitet das Paket weiter und erstellt einen Eintrag in der Verbindungsdatenbank sowie die dynamische Zugangsregel, die eine Antwort von Client A zulässt. Die Firewall von Client A verwirft das Paket, da es sich hier um eine von außen initiierte Verbindung handelt. Der Server informiert Client A über die TCP Verbindung, dass Client B an Port 8822 erreichbar ist (#4).

Client A sendet jetzt ein UDP Paket von Port 4711 an B's Port 8822 (#5). A's lokale Firewall erstellt einen entsprechenden Verbindungseintrag. A's Paket wird von B's Firewall an B weitergeleitet, da die ursprünglich von B ausgegangene Verbindung in B's Firewall noch gültig ist. Jetzt ist eine Verbindung zwischen A und B aufgebaut, obwohl die Regelwerke beider Firewalls normalerweise keine eingehenden Verbindungen zulassen würden.

Bei diesem Verfahren ist nur UDP basierter Datenaustausch möglich. Da bei der Nutzung von TCP die Sequenznummern von der Firewall mit überwacht werden, und diese sich bei unterschiedlichen Verbindungen unterscheiden, kann TCP als Transportmechanismus nicht genutzt werden.

4 Ein Dateitransfer auf Basis von UDP hole punching

Für einen Dateitransfer, der auf UDP hole punching basiert, stellt die Grid-Software UNICORE [UNI06] eine ideale Umgebung dar. Viele der im letzten Kapitel beschriebenen Konzepte sind bereits realisiert. Es gibt bereits ein Gateway für alle Kontrollnachrichten, die mit X.509 Zetifikaten verschlüsselt werden. Daher passt sich das Verfahren sehr gut in die Archtitektur ein. Ein dedizierter Vermittlungsserver für das UDP hole punching ist

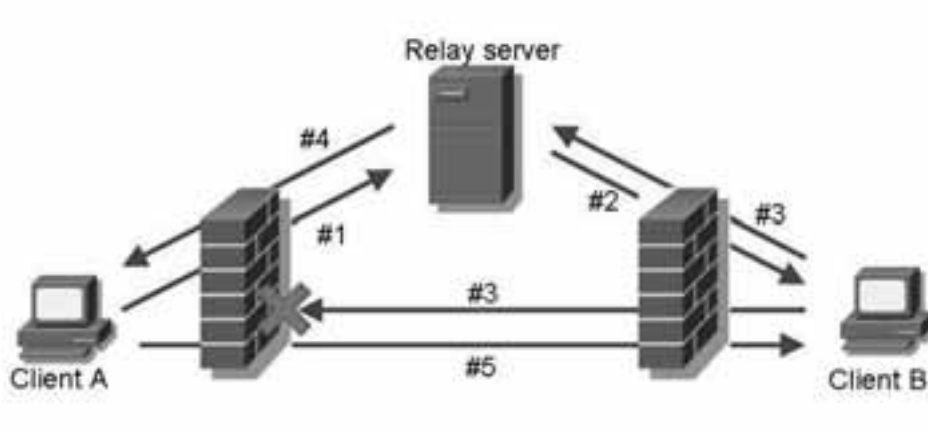

Abbildung 2: UDP hole punching

durch die Nutzung des UNICORE Gateways nicht mehr nötig. Benutzer und Geräte werden zentral autorisiert und authentifiziert. Die existierenden Dateitransfers in UNICORE basieren alle auf vordefinierten Basis-Klassen die grundlegende Funktionen und Fehlerbehandlung implementieren. Eine neue Transfermethode wird von diesen Klassen abgeleitet und erweitert sie um die für das Verfahren spezifischen Methoden. Daher besteht die Implementierung des Transfers aus der Bereitstellung der Methoden zum Austausch der Verbindungsdaten wie IP-Addresse und UDP-Port über das UNICORE Gateway, zum Versand des hole-punching Paketes und zum Datenaustausch über UDP-Pakete.

Eine neue Aufgabe ergibt sich aus der Verschlüsselung von UDP-Verkehr. Eine Möglichkeit ist die Verschlüsselung durch einen einfachen symmetrischen Algorithmus. Da die TCP-Kontrollverbindung zwischen Client und Server über das Gateway mit X.509 Zertifikaten verschlüsselt ist, kann der gemeinsame Schlüssel darüber ausgetauscht werden. Diese Verschlüsselung des Datentransfers sollte optional sein, um bei unsensiblen Daten eine höhere Geschwindigkeit bei gleichzeitig geringerer CPU-Belastung errreichen zu können.

Eine weitere Aufgabe ergibt sich aus den Anforderungen der Grid-Anwendung an den Dateitransfer. Dieser sollte schnell und zuverlässig sein. Da bei reiner UDP Kommunikation keine zuverlässige Übertragung garantiert ist, muss die Anwendung sicherstellen, dass alle Pakete in der richtigen Reihenfolge ankommen. Die Grid-Middleware braucht also ein Protokoll auf Anwendungsebene, das Zuverlässigkeit implementiert.

Dies kann vom `UDP-based Data Transfer Protokoll` (UDT) übernommen werden. UDT benutzt UDP als Transportprotokoll und ist somit „UDP hole punching“ fähig, garantiert aber Zuverlässigkeit durch eine zusätzliche Protokollschicht. Das folgende Unterkapitel beschreibt die allgemeinen Eigenschaften von UDT [GG04].

4.1 UDP-based Data Transfer Protocol (UDT)

Entwickelt wurde UDT am National Center for Data Mining der University of Illinois at Chicago [GG04]. Ziel war es, ein Protokoll zu entwickeln, das besonders auf schnelle moderne Netze zugeschnitten ist und diese vollständig ausnutzen kann. UDT ist in C++ implementiert. Es bietet einen eigenen Namensraum und orientiert sich streng an der standard Socket-API. Es braucht daher keinen großen Aufwand zur Einarbeitung und kann mit nur wenigen Anpassungen in bestehende Projekte integriert werden. Die wesentlichen Eigenschaften 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.

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.

Die wichtigen Eigenschaften und Funktionen Verbindungsorientiertheit, Zuverlässigkeit, Stau- und Flusskontrolle implementiert UDT auf Anwendungsebene mit den gleichen Mechanismen wie TCP. 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. UDT bringt bereits einen Algorithmus zur Staukontrolle mit. Er erreicht bereits nach kurzer Zeit die maximale Datenrate des genutzten Netzes und reagiert auf vereinzelte Paketverluste wesentlich weniger stark als TCP. Trotzdem verhalten sich mehrere UDT-Ströme in einem Netz fair zueinander. 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. Im praktischen Einsatz empfiehlt sich daher die Verwendung einer alternativen Staukontrolle oder die Limitierung der Bandbreite durch Quality-of-Service Regeln in den Routern. Dies gilt besonders bei der Nutzung öffentlicher Netze, in denen TCP-freundliches Verhalten Pflicht ist. Oft werden im Grid-Umfeld aber auch dedizierte Netze für den Grid-Verkehr genutzt. Dort ist UDT mit der standard-Staukontrolle besonders gut geeignet.

5 Die Gesamtarchitektur

Nachdem das UDP hole punching und seine Verwendung im Grid-Umfeld im letzten Kapitel beschrieben wurden, kann nun die Implementierung des neuen Dateitransfers im Gesamtsystem genauer betrachtet werden.

Die Implementierung basiert auf den beiden UNICORE-Klassen „FileTransferClient“ und „FileTransfer“ die nur durch eine zusätzliche WebService-Methode „initUDT“ erweitert wird. Wenn ein Transfer gefordert wird, dann erstellt die UNICORE-Middleware die Transferobjekte (WebService-Ports) auf Client- und Server-Seite. Der Transfer-Prozess besteht dann aus den folgenden Schritten. Der Client bereitet zuerst die Verbindung vor, indem er seinen UDT-Socket fest an einen zufällig gewählten Port bindet. Dann ruft er die Methode „initUDT“ des Servers auf. Diese Aufrufe werden nicht direkt getätigt, sondern auf dem Umweg über das Gateway. Seine IP und Port-Nummer sendet der Client als Parameter der WebService-Methode. Der Server bereitet seinen eigenen UDT-Server-Socket vor und führt das UDP hole punching aus, indem er ein leeres UDP-Paket an die IP-Port-Kombination des Client sendet. Dann sendet er seine eigene IP und Port-Nummer als Rückgabewert der Methode an den Client zurück. Der Client kann nun mit seinem UDT-Socket eine Verbindung zum Server aufbauen. Über diese Verbindung wird der Inhalt der zu transportierenden Datei mittels einfacher Send-Recieve-Funktionen übertragen. Informationen wie Dateiname und Dateigröße werden unabhängig vom Transfer-Verfahren von UNICORE festgestellt und übermittelt.

Da UDT in C++ geschrieben ist, UNICORE aber in JAVA, ist eine hybride Lösung nötig. Ein erster Ansatz bestand aus der Verwendung des JAVA Native Interface (JNI) für den Aufruf der UDT-Methoden. Ein flexibleres Verfahren stellt aber die Auslagerung des Transfers in eine eigene, in C++ geschriebene Anwendung dar. Diese wird von den Transfer-Objekten in einem zusätzlichen Prozess gestartet. Die Übergabe der Verbindungsparameter und zusätzlicher Informationen erfolgt durch die Umleitung und Manipulation der Standardein- und -ausgabe durch das JAVA-Objekt. Das UDP hole punching wird ebenfalls in dieser Anwendung durchgeführt. Die Anwendung muss bei diesem Vorgehen für das entsprechende System kompiliert sein. Sie ist nicht System- und Hardwareunabhängig. Durch die Trennung kann das Verfahren noch leichter in Kombination mit anderer Middleware genutzt, oder so sogar von Hand bedient werden.

Der Dateitransfer kann als Alternative zu den bisher üblichen Dateitransfermethoden in UNICORE genutzt werden. In produktiven Netzen zeigt er in Bezug auf Netzwerksicherheit und Geschwindigkeit die oben beschriebenen Vorteile gegenüber anderen Verfahren.

Für den Einsatz von GridFTP ist, verglichen mit dem neuen Verfahren, eine weniger sichere Konfiguration nötig, wärend die UNICORE Verfahren ByteIO und BFT die Dateien nicht direkt zwischen den Systemen austauschen, sondern durch WebService-Calls bzw. zusätzliche HTTP-Verbindungen über das Gateway. Sie sind wesentlich langsamer als direkter Austausch, da das Gateway in der Regel einen Bottleneck bildet.

6 Zusammenfassung

Firewalls sind zur Sicherung eines Netzwerkes absolut erforderlich. Der Einsatz von Sicherheitsregeln auf den Firewalls führt aber zu einer Einschränkung der Konnektivität. Grid-Anwendungen sind hier von besonders betroffen. Sie benötigen hohen Durchsatz und geringe Latenzen. Die Nutzung paralleler Verbindungen ist in diesem Zusammenhang

üblich. Dazu werden große Port-Bereiche an den Firewalls statisch geöffnet, was eventuell zu Sicherheitsproblemen führen kann. Dynamische Konfiguration kann dieses Problem mildern.

In diesem Bericht wurde eine Lösung vorgestellt, die eine Firewall mittels UDP hole punching auf sichere Weise dynamisch konfigurieren kann. Da dies nur mit UDP möglich ist wurde UDT als zuverlässiges, UDP basiertes Transport-Protokoll genutzt. Das Konzept konnte für die Verwendung im Grid Bereich angepasst und vereinfacht werden. Eine praxistaugliche Implementierung für UNICORE wurde vorgestellt. Diese Lösung bietet ein in vielen Organisationen ausreichendes Sicherheitsniveau.

Zu beachten ist, dass der standardmäßig mitgelieferte Algorithmus zur Staukontrolle für öffentliche Netze zu aggressiv vorgeht und die anderen Verbindungen im Netz stark unterdrückt. Ein faireres Verfahren sollte je nach Einsatzzweck genutzt werden. Alternativ kann die verwendete Bandbreite durch Quality of Service Regeln in den Routern der beteiligten Netze begrenzt werden.

Das Verfahren ist durch die modulare Implementierung leicht in anderen Grid-Anwendungen einsetzbar und funktioniert mit allen üblichen Firewalls. Es kann als Übergangslösung dienen, bis „echte“ dynamisch konfigurierbare Firewalls verfügbar sind bzw. neuartige Protkolle mit entsprechenden Features standardisiert und verbreitet wurden.

Literatur

[Cis] Cisco. *Cisco Security Appliance Command Line Configuration Guide - For the Cisco ASA 5500 Series and Cisco PIX 500 Series Software Version 7.2*.

[GG04] Y. Gu and R.L. Grossmann. *UDT: A transport protocol for data intesive applications*. University of Illinois at Chicago, August 2004.

[SAL05] S. Son, B. Allcok, and M. Livny. CODO: Firewall Traversal by Cooperative On-Demand Opening. In *14th IEEE Symposium on High Performance Distributed Computing (HPDC14)*, http://www.cs.wisc.edu/ sschang/papers/CODO-hpdc.pdf, July 2005.

[Sch06] J. Schmidt. Der Lochtrick - Wie Skype & Co. Firewalls umgehen. *Ct*, 17:142 ff, 2006.

[Ste94] W. Richard Stevens. *TCP/IP Illustrated*. Addison Wesley, 1994.

[UNI06] UNICORE. UNICORE Grid computing Technology, August 2006. http://www.unicore.eu/.

[web06] Globus Toolkit website. *http://www.globus.org/toolkit/docs/4.0/data/gridftp*, August 2006.