Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Project Report

Projekt Bericht

Lutz Behnke

Wotan

Fachbereich Informatik
Department of Computer Science

Lutz Behnke

Thema Projekt Bericht
Wotan

Stichworte
Peer2Peer Storage Persistenz

Kurzzusammenfassung
Wotan ist ein Prototyp für eine neue Art Dateien zu handhaben, abzulegen und in Netzwerken zu verteilen.

Lutz Behnke

Title of
Wotan

Keywords
Peer2Peer Storage Persistenz

Abstract
Wotan is the prototype for a new way to handle, store and distributed files across networks.

Inhaltsverzeichnis

Kapitel 1

Vision

Selbst bei weiterer Gültigkeit des Moore'schen Gesetzes, ist der Speicherplatz von mobilen Geräten immer noch nicht groß genug, da mit den Möglichkeiten auch die Anforderungen steigen. Handys, die Filme in Fernsehqualität aufzeichnen können oder hunderte, wenn nicht tausende von Bildern speichern können, sowie eine Titelbibliothek für den MP3-Spieler enthalten, die selbst den verschrobensten Plattensammler erblassen lassen. Natürlich ist der Speicher immer genau dann voll wenn man das eine ultimativ wichtige Bild machen möchte.

Die Vision von Wotan ist es den verfügbaren Speicherplatz aller, in einem Kontext vorhandenen Geräte, allen Benutzern und ihren Geräten zur Verfügung zu stellen. Die Abgrenzung des Kontextes geschieht dabei durch die Verfügbarkeit von Netzwerken und kann je nach Situation das eigene Personal Area Network (PAN, z.B. Bluetooth), Local Area Network (LAN, z.B. WLAN oder Ethernet) und womöglich auch das Internet, bzw. verteilte Arbeitsgruppen darin, enthalten.

Bei der Betrachtung von Komponenten die im Bereich Ubiqutous-, Wearable- und Pervasive-Computing zu Einsatz kommen, wie z.B. PDAs/XDAs, SmartPhones und anderen Geräten mit minimaler Benutzerschnittstelle zeigte sich auch, das ein neuer Ansatz für Datei-Ablagen nicht nur die Ressourcen für den Anwender transparent verwalten müssen, sondern auch den Zugriff zwischen Myriaden von Bausteinen regeln sollten, ohne das der Anwender eingreift. Und dennoch die Sicherheit heutiger Ansätze zumindest erreichen, besser noch *stark* verbessern.

Die Anforderung heutiger Desktop-Systeme, das der Anwender sich merken muss, auf welchem Gerät und in welchem Ordner er eine Datei abgelegt hat, ist an sich schon verbesserungs fähig, betrifft in der Hauptsache aber nur wenige Geräte (z.b. Den Arbeitsplatz-PC, der PDA und der PC zu Hause). In einer ubiquitären/-pervasiven Umgebung (*Ubiqutous/Pervasive Environment*, UPE) aber gibt es eine große Anzahl von Geräten die der Anwender nicht einmal unbedingt als Computer erkennen kann und auf die er, vor allem in mobilen Kontexten, auch möglicherweise keinen Einfluss nehmen kann. Daher muss eine minimale Anforderung an eine Lösung eine strikte Umsetzung des Zero-Configuration Anspruches sein.

Kapitel 2

Umsetzung

Im Rahmen wurde eine mögliche technische Umsetzung des Projektes Erarbeitet und wurden grundsätzliche Design Entscheidungen untersucht, welche die Architektur einer Lösung entscheidend bestimmen.

Grundsätzlich ergibt sich aus dem Problem der Verwaltung von Dateien mittels minimaler Benutzerschnittstellen und vielen Geräten (*"Auf welchem USB-Stick hatte ich bloß die neueste Version des Vortrages"* die Notwendigkeit die heute hauptsächlich verwendete Schreibtisch-und-Aktenschrank Metapher neu zu überdenken. Andere Ansätze bedingen eine grundlegend andere Formen der Dateiablage:

Keine Ablagehierarchie Es gibt keine Ordner, gemeinsame Ordner oder Projekt-Ablagen mehr. Auch müssen Dateien nicht notwendig dem Anwender präsentierte Namen haben, jedenfalls nicht als Bestandteil des Dateisystems. Sondern bezeichnende Labels kann jeder Benutzer selbst bestimmen und müssen durch Applikationen verwaltet werden, die oberhalb des Dateisystems angesiedelt sind.

Name und Ort sind unabhängig Der Speicherort einer Datei bestimmt nicht ihre Identifikation. Dies bezieht sich sowohl auf Ablage-Hierarchien (die nicht in vorhanden sind), als auch auf Medien oder Geräte. Im Gegenzug ändert eine Datei auch nicht ihre Identifikation wenn ihr Speicherort verändert wird.

Identifikation nach Inhalt Applikationen müssen nicht mehr den Inhalt einer Datei austauschen, sondern nur ihren Namen. Die Übertragung von Daten wird für alle Anwendungen von einer Komponente realisiert. Diese verwaltet auch Redundanz (um Ausfallsicherheit zu realisieren). Da sich der Name ändert, wenn die Datei geändert wird sind Versionskollisionen ausgeschlossen[1].

Keinerlei Speicherung von Metadaten Der ständige Anstieg, an zu verwaltenden Dokumenten hat einen immer größeren Satz an Metadaten in Dateisys-

[1] Dieser Mechanismus wird schon heute in GIT eingesetzt, dem Versionskontroll-System mit dem die Quellen des Linux Kernels verwaltet werden.(http://git.or.cz)

temen nötig gemacht. Wenn die Verwaltung der Metadaten aber als Aufgabe eines Dokumenten-Verwaltungssystems verstanden wird, so kann das System zu Speicherung der Nutzdaten von dieser Last befreit werden.

Transparente Verwaltung von Speicherknoten Egal ob durch schnelle Wechsel von Knoten in Ad-Hoc Netzen oder durch die schiere Anzahl der Knoten in Firmen-Netzen, als auch durch den mangelnden technischen Sachverstand des Benutzers ist eine Konfiguration von Speicherorten nicht praktikabel. Auch wenn diese Funktion durch den Prototypen noch nicht vollständig unterstützt wird, dann ist das automatische Entdecken und Verwalten von Speicherorten durch Wotan ein zentrales Element einer vollständigen Umsetzung.

Trennung von Kontrolle und Verarbeitung Die generelle Verfügbarkeit einer Datei in einem Kontext erlaubt eine Handhabung von Dateien analog zum Konzept des *late binding* in einigen Programmiersprachen: Die Datei wird erst beim öffnen auf die Zielkomponente übertragen. *Policies* können natürlich Caching und Pre-Fetching durchführen um die Effizienz des Systems zu optimieren. Aber dies ist weder Bestandteil des Prototypen noch des grundsätzlichen Verhaltens von Wotan .

Die hohe Anzahl von Einzelgeräte in einer UPE bedingt aber auch zwingend geringe Kosten für die einzelnen Komponenten. Dies wird sicherlich von vielen Anbietern auch als Entwicklungsziel angestrebt um die Produkte für den Anwender attraktiv zu machen. Damit wird aber sicherlich auch die Haltbarkeit der einzelnen Komponenten leiden. Damit muss eine UPE transparent mit dem Ausfall von Komponenten umgehen können. Dies verstärkt nicht nur die Forderung nach Zero-Configuration Elementen, sondern auch den konsequente Verzicht auf Strukturen die zu Single-Points-of-Failure werden können.

Für Wotan wurde auf eine Replikation von Daten auf multiplen Knoten gesetzt. Da eine Replikation entweder durch komplexen Algorithmen synchronisiert werden muss oder die Aktualität von lokalen Kopien einer Datei nicht garantieren kann, ändert sich in Wotan der Name einer Datei wenn deren Inhalt geändert wird. Damit ist der Identifikator einer Datei in Wotan eine Repräsentation ihres Inhaltes und nicht ihres Speicherortes.

Aus Sicht der Anwendung ist der Mangel an Ablagehierarchie kein unbedingter Nachteil. Werden bestimmte Inhalte gesucht und es ist kein spezieller Dateihandle bekannt (z.B. in einem Bild-Archiv), dann werden halt die Inhalte durchsucht. Dies wird z.B. für eMails schon bei Google-Mail[**?**] schon heute erfolgreich eingesetzt. Auch stellt Microsoft den Suchdienst als eine der wichtigsten Neuerungen von Windows Vista dar.

Kapitel 3

Das Projekt

3.1 Ziele des Projektes

Neben der Erarbeitung der Konzeptdetails in 2 und dem Vergleich mit existierenden Lösungen [**?**] hatte das Projekt folgende Ziele und Liefergegenstände:

- Entwicklung eines Protokolls zum Austausch von Dateien zwischen den Knoten und Beschreibung der Formate der beteiligten Daten-Objekte.
- Implementation eines Prototypen der obiges Protokoll umsetzt. Dabei sind alle Punkte zu identifizieren, in denen Schwächen oder Einschränkungen aufgrund des Prototyp-Charakters in der Implementation enthalten sind. Auch Funktionen die nicht implementiert wurden, die zur Umsetzung der Vision aber notwendig sind, sind zu dokumentieren.
- Durchführung der Implementation nach Software-Engineering Gesichtspunkten. Während durch den Umstand das nur eine Person an dem Projekt arbeitet eine Fehlerverfolgung nicht notwendig ist, so ist neben einem Versionsmanagement vor allem auch die Erstellung einer Testumgebung und der notwendigen Unit-Tests umzusetzen.
- Weiterführung der

3.2 Ergebnisse

3.2.1 Prototyp

Der Prototyp ist in Java 5.0 implementiert und in jeder entsprechenden Standard Edition (SE) Laufzeitumgebung ablauffähig.

Die Abbildung 3.1 zeigt eine grobe Übersicht über die Architektur von Wotan . Lokale Anwendungen sprechen immer mit dem Broker-Dienst, der auf jeder Maschine laufen muss. Je nach Fähigkeiten und lokalen Ressourcen kann der Broker nun Dateien lokal cachen oder versuchen zu anderen Brokern im Kontext abgeben.

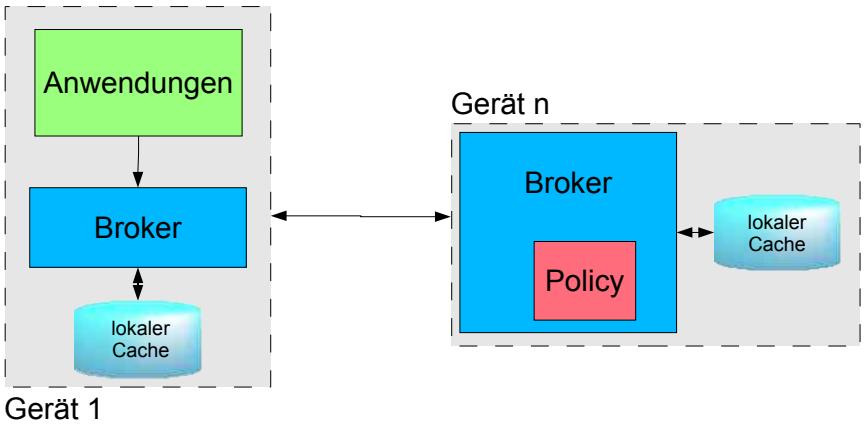

Abbildung 3.1: Architektur von Wotan

Alle Intelligenz, die zum Auffinden von anderen Knoten und dem Aushandeln von Ressourcen benötigt wird, ist im Broker enthalten.

Dabei ist jeder Broker im Netzwerk als Server sichtbar um Anfragen entgegen zu nehmen. Wird der Broker von einem Knoten kontaktiert, mit dem er bisher noch keine eigene Netzwerk-Verbindung unterhält, so eröffnet der Broker im Gegenzug sofort eine Verbindung. Dadurch lässt sich das Protokoll besonders einfach gestalten, denn alle Operationen können aus der Sicht des Clients entworfen werden. Wenn der Server selbst Daten erhalten möchte, so stellt er diese Anforderungen über die Zusätzliche Verbindung, in der er selbst der Client ist.

- Eine Datei besteht aus einer Menge an *Chunks*. Diese sind alle von gleicher Größe und werden durch anhand ihres SHA-384 Hashes identifiziert. Die Informationen, welche Chunks zu einer Datei gehören (durch eine Liste der Hashes) und wie viele Byte des letzten Chunks Nutzdaten und was Padding ist, werden durch einen File-Control-Chunk (FCC)festgelegt. Für diesen FCC wird wiederum der SHA-384 Hash ermittelt. Dies ergibt den Namen der Datei.

- Alle Dateien haben eine weltweit eindeutige Identifikation, die sich aus ihrem Inhalt ergibt. Diese Identifikation ist nicht für Menschen, sondern ausschließlich für Maschinen gedacht und wird auch nur von diesen verwaltet.

- Während die Schnittstelle des Prototypen noch den exakten Aufbau des Handles auf Datei-Objekte verbirgt, und nur einen abstrakten Handle zur Verfügung stellt, so hat sich doch im Laufe der Entwicklung der Einsatz dieser Identifikation, einer Sequenz von 48 Byte Länge als praktikabel und ausreichen herausgestellt. Dieser Hash wird in Kombination von einem Gruppenschlüssel, der `group-id` verwendet. In einer Gruppe sind alle Chunks von gleicher Größe und benutzen den gleichen Hash-Algorithmus.

Während der Umsetzung des Prototypen hat sich zeigte sich das das Verhalten einer Wotan Implementation durch einige wenige Zeilen Code bestimmt werden können, in denen das Verhalten des Brokers beschrieben werden kann. Diese Policy entscheidet welche Dateien und Chunks an andere Broker versandt werden, welche von anderen eingefordert werden.

Im Prototypen ist bisher nur eine Policy implementiert. Sie repliziert alle verfügbaren Daten auf alle erreichbaren Knoten.

3.2.2 Protokoll

Das Wotan Protokoll ist im Anhang enthalten. Während der Laufzeit des Projektes wurde das Protokoll immer kürzer statt länger, da sich viele Funktionen, wie die Verwaltung von Zugriffsrechten und komplexe Operationen wie sie aus dem FTP bekannt sind, entweder als nicht notwendig erschienen oder der Vision von Wotan widersprachen Z.B. beschäftigt sich ein großer Teil von FTP im Navigieren und Verwalten von Verzeichnishierarchien und der Manipulation von Dateinamen. Alle diese Funktionen sind in Wotan nicht existent.

Was von FTP aber inspiriert wurde wahr die Verteilung von unterschiedlichen Aufgaben auf unterschiedliche FTP-Verbindungen. Während FTP Kontrolle und Datenübertragung auf verschiedene Kanäle verteilt, sind es bei Wotan die unterschiedlichen Rollen welche die Teilnehmer einnehmen.

3.3 Schwierigkeiten

Ein Großteil der Zeit während der Projektlaufzeit wurde für die Implementation von Byte-genauen Netzwerk Operationen in Java sowie die Beschäftigung mit der `java.nio` Pakete zur Implementation von asynchronen Netzwerk-Anwendungen benötigt. Die von Java zur Verfügung gestellten Beispiele sind nicht sonderlich aussagekräftig und die Dokumentation verbesserungswürdig.

Dem Prototypen mangelt es immer noch an Stabilität. So können bestimmte Race-Conditions zu einer Invalidierung der Select/Wait Operationen der `java.nio` Pakete führen, was einen Busy-Loop zu folge hat, der alle verfügbare Rechenleistung der Maschine verbraucht.

3.4 nicht erreichte Ziele

Die Umsetzung der Anforderungen an ingenieurmäßiges Arbeiten ist am Widerstand der ausgewählten Werkzeuge gescheitert. Es ist mir bis heute nicht gelungen den Remote-Agent für das *Test and Performance Tools Plattform*(TPTP) zur Ausführung zu bringen. Alle Anleitungen und Tutorials beschäftigen sich damit Applikation Server zu testen, aber einfache `Hello-World` Beispiele, die auch mögliche auftretende Fehler diskutieren, fehlen leider.

Auch die Implementation des Prototypen direkt auf einer PDA-Zielplattform ist an dem Aufwand, eine neue Programmiersprache (`C#`) zu lernen gescheitert. Diese Sprache hätte nämlich neben der eigentlichen Sprache auch die Einarbeitung in eine weitere komplexe Werkzeug-Umgebung und – wahrscheinlich noch aufwendiger – die dazugehörige Laufzeitbibliothek. Eine kurze Sondierung ob sich die Plattform auch unter Linux entwickeln ließe brachte eher schlechte Ergebnisse. Zum einen ist Mono nicht .Net. Die Alternative währe eine Verwendung des Visual Studio via Remote-Desktop Verbindung. Hier zeigte sich das entfernte Darstellung für Windows keine native Lösung ist, und selbst Anwendungen von Microsoft selbst bei einer Übertragung über das Remote-Desktop Protokoll Darstellungs-Artefakte produzieren. Diese Lösung ist für Remote-Administration geeignet, für Remote-Softwareentwicklung aber nicht.

3.5 Fazit

Der Prototyp ist noch nicht von ausreichender Reife um festzustellen ob das Konzept von Wotan in produktiven Lösungen einsetzbar ist. Erst die Entwicklung von intelligenteren Policies kann dies zeigen. Auch ein Trust- und Verschlüsselungs-Konzept ist notwendig um zumindest den Grad an Leistungsfähigkeit von normalen Zugriffskontrollmechanismen zu erreichen.

Für die weitere Entwicklung existiert eine lange Liste von Verbesserungen die in den nächsten Wochen umgesetzt werden. Dennoch zeigt die kontinuierliche Beschäftigung mit dem Thema eine Reihe von Anknüpfungspunkten, sowohl im der ursprünglichen Zielbereich, den mobilen Geräten, als auch bei der Umsetzung von kollaborativen Scenarien wie sie im Projekt Aura der Carnegie-Mellon Universität und der ETH Zürich untersucht werden.

3.6 Ausblick

Das Thema werde ich im Rahmen der Abschlussarbeit des Masterstudienganges weiterbearbeiteten. Dafür stehen einige konkrete Arbeiten für die Zukunft an um Wotan als Lösung für den praktischen Einsatz untersuchen zu können.

- Entwicklung von Policies die den REDIRECT Fehlercode implementieren
- Policies die Vertrauen und Verantwortung für die Ablage von Chunks mit ihren Peers aushandeln. Diese Anforderung war einer der Hauptgründe für die Einführung der `community_id`, um Gruppen, innerhalb denen eine Vertrauens- und Verantwortungsstruktur aufgebaut werden kann.
- Die Definition für Struktur der FCCs wird gerade von einer einfachen Java serialisierungs Struktur, welche nicht Implementations-neutral ist (oder welche eklatante Teile der Reflection- und Marschaling-Konstrukte von Java

RMI reproduzieren müssten) in eine stark vereinfachte, bit-genau definierte, und vor allem Plattform-neutral Form übertragen.

- Mit dem neuen FCC Format werden dann auch Verschlüsselung und Signatur in Wotan Einzug halten. Der Umstand das die Chunks und die Dateien selbst durch Hash-Werte identifiziert und verwaltet werden sollte die Einführung von Signaturen stark vereinfachen, da die halbe Arbeit (das Berechnen der Hashes) schon erledigt wurde.

- Um möglich Speicherorte in größeren Kontexten (z.b. dem Internet) zu verwalten werden neue Funktionen benötigt. Als aussichtsreiche Kandidaten wurde während des Projektes Distributed Hash Tables (DHTs) untersucht. Diese sind zwar resistent gegen den kontinuierlichen Verlust von Knoten (und das Hinzufügen von einzelnen Knoten). Aber das Verhalten bei einem Kontext-Wechsel, bei dem potentiell alle Knoten ausgetauscht werden ist noch zu untersuchen.

  In kleinen Kontexten, z.B. einem PAN oder kleinen WLAN Umgebungen (<20 Knoten) können alle Peers sequentiell gefragt werden. Durch das Aufrechterhalten von TCP Verbindungen wird ein Flooding des Netzes vermieden, da die Broker nur die Änderungen ihrer Datenbasen kommunizieren.

Natürlich stellt ein Prototyp schon einen reichen Quell an Code dar, in dem eine Reihe von Abkürzungen und hart verdrahtete Vereinfachungen gemacht wurde. Diese sind zu großen Teilen im Source-Code identifiziert und müssen nun abgearbeitet werden.

Außerdem müssen noch einige Schwächen der Implementation im Bereich der Fehlerresistenz ausgemerzt werden.

Anhang A

Wotan Protokol

A.1 Einleitung

A.1.1 Protokollelemente

Das Protokoll sorgt in der Hauptsache für die Übetragung von Dateien (“Files”) die in Abschnitte (“Chunks”) aufgeteilt sind. Die Chunks werden durch das Ergebniss eines kryptographischen Digest auf ihren Inhalt identifiziert. Für jede Datei gibt es mindestens einen Kontroll-Chunk, der eine Liste der zur Datei gehöhrigen Chunks enthällt. Eine Datei kann im Prinzip aus beliebig vielen Chunks bestehen, ist in dieser Version des Protokols aber auf die Anzahl der Chunks begrenzt deren Identifikatoren (“Hashes”) in einem Chunk abgelegt werden können.

Jede Form von Metadaten, die für die Präsentation der Dateien für einen Menschen notwendig sind, z.B. ein Dateiname oder Typ des Inhaltes sind nicht Bestandteil dieses Standards.

A.2 Verbindungsaufbau

Durch einen Mechanismuß der nicht Bestandteil dieses Protokolles ist werden Gegenstellen identifiziert. Diese werden durch den Aufbau einer TCP Verbindung kontaktiert. Da jeder Knoten im System sowohl als Client als auch als Server im Socket-Modell auftritt wird eine kontaktierte Gegenstelle auch die Gegenverbindung aufbauen.

Der Mechanismuß zum identifizieren von Gegenstellen kann sowohl eine statische Liste von Teilnehmern einer Gruppe wie auch ein AdHoc-Discovery Mechanismuß wie UPnP, SDP oder Bluetooth-Discovery sein.

Danach handeln beide die zu übertragenen Daten aus. Das folgende Protokol ist aus Sicht des Clients definiert. Beide Knoten agieren nach dem gegenseitigen Verbindungsaufbau simultan als Client und führen das Protokol durch.

Die Entscheidung ob Daten angenommen oder abgeschickt werden sollen entscheidet eine Policy-Komponente in jedem Teilnehmer eigenständig. Diese Policy-

Komponente ist nicht Bestandteil dieses Protokolles.

Wenn eine Partei nicht zeitgerecht auf ein Kommando antwortet (Es kommt an beliebiger Stelle zu einem Timeout) wird die Verbindung abgebrochen kann erneut aufgebaut werden wenn die Gegenstelle noch erreichbar ist.

A.3 Kommandos

Alle Zeichen sind nach dem ANSII kodiert. Die Antwortkodes bestehen jeweils aus Dezimalzahlen die als String von drei Zeichen kodiert sind.

die mit **Send:** markierten Nachrichten werden vom Client and den Server gesandt, die mit **Reply:** markierten Nachrichten vom Server zu Client.

A.3.1 HELLO

Das HELLO-Kommando initiert die Kommunikation zwischen zwei Knoten und dient dem Abgleich der Versionsnummer, so das ein Teilnehmer unterschiedliche, evtl. inkompatible Versionen des Standards unterstützen kann.

Send: HELLO <Protokoll Version: 1 byte>

Reply: <Antwortkode: 3 byte>

Die Protokoll Version kann folgende Werte haben:

Version	*Byte Wert*
0.9	1

Der Antwortkode kann folgende Werte annehmen:

Kode	*Mnemonik*	*Bedeutung*
200	TREADY	Der Server ist bereit für weitere Schritte im Protokol und unterstützt die angebene Versionsnummer
403	VERFAIL	Die im HELLO angegebene Versionnummer wird vom Server nicht unterstützt
501	NETFAIL	Ein nicht näher spezifizierter Fehler in der Übertragung ist aufgetreten.

A.3.2 Send Chunk (SENDCH)

Send: SENDCH <Chunk Länge: Integer Wert, MSB, 4 Byte><Hash OID Länge: 1 byte><Hash Algorithmus, DER-kodierte ASN.1 OID> <chunk hash>

Reply: <Antwortkode 3 byte>

Wenn der Antwortkode “200” ist:
Send: <Chunk daten>

Wenn die Übertragung erfolgreich war:
Reply: <Antwortkode: TROK><chunk hash><Signature Algorithm ASN.1 OID><Signatur Daten>

Sonst:
Reply: <Antwortkode>[<Länge der Fehlerbeschreibung: 1 Byte>][<Fehlerbeschreibung, max 255 Byte>]

Die textuelle Fehlerbeschreibung sollten in Englisch gehalten sein.

Alg	*OID (ASN.1/DER)*	*Länge (Bit)*
SHA	1.3.14.3.2.18 (06:05:2B:0E:03:02:12)	20
SHA-1	1.3.14.3.2.26 (06:05:2B:0E:03:02:1A)	20
SHA-384	2.16.840.1.101.3.4.2.2 (06:09:60:86:48:01:65:03:04:02:02)	48
SHA-512	2.16.840.1.101.3.4.2.3 (06:09:60:86:48:01:65:03:04:02:03)	64

SHA-384 ist der bevorzugte Algorithmus. Die angegebenen OID Informationen stammen aus der Datei `dumpasn1.config` des dumpasn1 Programmes von Peter Gutmann.

Folgende Fehlerkodes können auftreten:

Kode	*Mnemonik*	*Bedeutung*
200	TREADY	Der Server ist bereit für weitere Schritte im Protokol und unterstützt die angebene Versionsnummer
201	TROK	Die Übertragung des Chunks war erfolgreich
301	HAVECH	Einen Chunk mit diesem Hash hat der Empfänger schon
401	NOHASH	Der Hash-Algorithmus ist nicht bekannt oder die Berechnung des Hashes ist aus anderen Gründen nicht möglich
402	NOSPACE	Empfänger hat keinen Platz oder ist zumindest nicht bereit Platz zu opfern.
501	NETFAIL	Ein, nicht näher spezifizierter Fehler in der Übertragung ist aufgetreten.
503	LFAIL	Ein, nicht näher spezifizierter lokaler Fehler ist aufgetreten

A.3.3 Retrieve Chunk (QUERYCH)

Ein Client fordert einen Chunk vom Server ab.

Send: QUERYCH <hash OID len: 1 byte> <hash algorithm ASN.1 OID><chunk hash>

Reply: <reply code 3 byte>[<vom Fehlerkode abhängige Daten>]

Folgende Fehlerkodes können auftreten:

Kode	*Mnemonik*	*Bedeutung*
200	TREADY	Der Server ist bereit den Chunk zu übertragen. Die Nutzdaten folgen dem Fehlerkode.
302	NOHAVE	der Chunk mit diesem Hash ist dem Server nicht bekannt
303	NOHAVENOW	der Chunk ist momentan nicht verfügbar. Er wird gerade von einem anderne Server/Medium besorgt und der Client kann, wenn er möchte, in der Zwischenzeit etwas anderes tun.
304	REDIRECT	Der Server ist nicht bereit, berechtigt oder sonst in der Lage den anderen Server, von dem er weis das dieser den Chunk hat, zu kontaktieren, aber der Client kann selbst die Verbindung aufbauen und die Daten holen.
401	NOHASH	Der Hash-Algorithmus ist nicht bekannt oder die Berechnung des Hashes ist aus anderen Gründen nicht möglich
501	NETFAIL	Ein, nicht näher spezifizierter, Fehler in der Übertragung ist aufgetreten.
503	LFAIL	Ein, nicht näher spezifizierter, lokaler Fehler ist aufgetreten

Wenn der Fehlerkode 200 beträgt sieht die Antwort des Servers wie folgt aus:

Reply: <reply code 3 byte><chunk length: integer value 4 bytes, little-endian><chunk data>

Wenn der Fehlerkode 304 beträgt sieht die Antwort des Servers wie folgt aus:

Reply: <reply code 3 byte><Addresse (IP) des andern Servers auf dem sich die Daten befinden können. 4 Byte>

A.3.4 Set File (SFILE)

Eigentlich ist eine Benachrichtigung des Peers über die Eigenschaft eines Chunk als Datei-Index nicht notwendig. Ein Zugriff auf eine Datei ist nur möglich, wenn eine Anwendung den Hashwert und damit den Identifier eines Index-Chunks verwaltet. Somit muss ein Server nicht wissen um was es sich bei einem Chunk handelt.

Allerdings kann es hilfreich sein, um die Offline-Effektivität von Clients zu verbessern, alle Chunks einer bekannten Datei auf den lokalen Puffer zu übertragen.

Dies ist nur möglich wenn der lokale Broker die Identifikatoren für alle Chunks einer Datei kennt.

Der Befehl `SFILE` muss von einem Server nicht implementiert werden, solange eine Anfrage mit dem korrekten Fehlercode (510:NOIMPL) beantwortet wird.

Der Befehl überträgt keine Chunk-Daten sondern markiert einen Chunk nur als Datei-Index.

Send: SFILE <Hash OID len: 1 byte><Hash Algorithmus ASN.1 OID><Chunk Hash>

Reply: <reply code 3 byte>

Kode	*Mnemonik*	*Bedeutung*
201	TROK	Der Hinweis auf den Datei-Index wurde akzeptiert.
401	NOHASH	Der Hash-Algorithmus ist nicht bekannt oder die Berechnung des Hashes ist aus anderen Gründen nicht möglich
501	NETFAIL	Ein, nicht näher spezifizierter Fehler in der Übertragung ist aufgetreten.
503	LFAIL	Ein, nicht näher spezifizierter lokaler Fehler ist aufgetreten
510	NOIMPL	Dieser Befehl ist nicht implementiert, die Information wird ignoriert

A.4 Dateistruktur

Chunks welche die Struktur einer Datei enthalten sind als Folge von Bytes wie folgt aufgebaut:

<Chunk-Größe, 4 Byte, little-endian> <Community-Id, 8 Byte, little-endian> <Länge der Datei, 8 Byte, little-endian> <Anzahl der Bytes in der Hash-Oid, 1 Byte> <Hash-Oid, variable Länge> <Anzahl der Chunks in der Datei, 4 Byte, littel-endian>

Nun folgen die Hashes der Chunks in der Datei. Ihre Länge wird durch den verwendeten Hash-Algorithmus bestimmt.

A.5 Notizen und Anmerkungen

In der vorliegenden Version enthält das Protokol noch einige Lücke und Schwächen. Ein paar Überlegungen für Weiterentwicklungen sind im folgenden notiert. Die Reihenfolge ist zufällig und stellt keine Gewichtung dar.

In der momentanen Version des Protokolles ist eine Verschlüsselung noch nicht spezifiziert[1], aber das eine der intendierten Klasse von Zielsystemen auch mobile

[1] "Sicherheit kommt später"

Geräte mit begrenzter Rechenleistung und Energievorrat sind, sollte asymetrische Crypto auf dem Client wenn möglich vermeiden werden. Es ist die Möglichkeit für den Einsatz von HMAC's zu untersuchen.

Im Moment sind noch keine Timeouts definiert[2]

Eine Übetragung von Gewichtungen von Dateien ist nocht nicht unterstützt. Da diese Informationen aber von der jeweiligen Policy abhängen, wird das Protokol wahrscheinlich noch um einen Policy spezifischen Datenblock erweitert werden müssen. Da potentiell jede Policy eigene Daten definieren muss, sollte der Mechanismuß entsprechend flexibel sein.

[2]Macht aber nix, die Referenzimplementation unterstützt auch noch keine Timeouts