Sicherheitsaspekte von verteilten Hashtabellen in Peer-to-Peer Routingprotokollen

Ullrich Pfefferlein
{ullrichp@cs.tu-berlin.de}

Seminar "Internetsicherheit" ,
Technische Universität Berlin

WS 2009/2010 (14. Januar 2010)

Zusammenfassung

Es gibt zahlreiche Untersuchungen über Sicherheitsaspekte von verteilten Hashtabellen in Peer-to-Peer Routingprotokollen. In diesem Papier wird untersucht, wie diese Aspekte in den Protokollen: CAN, Chord, Kademlia und Pastry berücksichtigt bzw. implementiert wurden.

1 Einleitung

Peer-to-Peer (kurz: P2P) ist als Schlagwort wohlbekannt und wird meist synonym für den Begriff *Filesharing* verwendet. Dabei steht "P2P" für ein Konzept, wie Resourcen (z.B. Dateien) in einem Netzwerk übertragen und gespeichert werden können. In neueren P2P-Systemen kommen dafür Routingprotokolle zum Einsatz die ohne zentrale Server (sog. Tracker) arbeiten und mit Hilfe von *verteilten Hashtabellen* (kurz: DHT, aus dem engl.: distributed hash table) den Knoten (Peer) finden, der für eine bestimmte Resource zuständig ist bzw. welcher diese gespeichert hat. Die Sicherheitsaspekte, die in diesem Papier untersucht werden sollen, leiten sich durch die Verwendung von verteilten Hashtabellen ab oder ergeben sich durch diese.

Die hier untersuchten Protokolle die verteilte Hashtabellen einsetzen sind: CAN[3], Chord[4], Kademlia[5] und Pastry[6]. Diese Protokolle kommen auch in P2P-Clients zum Einsatz. Der BitTorrent-Client[1] z.B. verwendet Kademlia für den trackerlosen Betrieb. Die anderen Protokolle finden in keinem der weit verbreiteten P2P-Clients Verwendung. Pastry findet in anderen Anwendungen (z.B. PAST oder SplitStream) Verwendung, Chord und CAN dagegen werden ehr im akademischen Umfeld eingesetzt und untersucht.

Es folgt eine kurze Einführung in verteilte Hashtabellen (Kapitel 2). Ausführlich erläutert werden die Sicherheitsaspekte in Kapitel 3. Es wird das Angreifermodell vereinbart und Abschnittsweise das Problem bzw. der Angriff und eine passende Lösung vorgestellt. Diese Auflistung basiert auf der Veröffentlichung von Emil Sit und Robert Morris[1]. In Kapitel 4 soll dann untersucht werden, wie diese Sicherheitsaspekte in den vier hier untersuchten Protokollen umgesetzt und berücksichtigt worden sind oder in wie weit Eigenschaften des Protokolls Vor- bzw. Nachteile diesbezüglich bringen.

[1] Gemeint ist der (zum Protokoll gleichnamige) Client der Firma BitTorrent Inc., siehe auch http://www.bittorrent.com/.

2 Verteilte Hashtabellen

In diesem Kapitel wird der Begriff *verteilte Hashtabelle* näher erläutert und die Verwendung dieser in Peer-to-Peer Routingprotokollen gezeigt. Die beiden zentralen Begriffe sind: Verteilung und Hashing. Es wird nun zunächst erläutert welche Rolle Hashing spielt und dann wie eine Verteilung dieser Hashtabelle erfolgt.

Hashing Über eine Hashfunktion (z.B. SHA1 oder RMD160) wird ein Hashwert von einem *Identifikator* (z.B. einem Dateinamen) gebildet. Dieser Hashwert liefert den Schlüssel unter welchem die Resource (die eigentlichen Daten zum Identifikator) im P2P-Netzwerk zu finden ist.

Verteilung Den Knoten in einem P2P-Netzwerk werden eindeutige IDs (man spricht von der Knoten-ID) zugeteilt. Sie werden dann (gemäß dem benutzten Protokoll) über diese ID in einer bestimmten Struktur angeordnet. Diese Struktur kann ein Ring sein (Chord), ein d-Torus (CAN), eine Baumstruktur (Kademlia) oder eine Struktur mit mehreren Ebenen (Pastry). Man spricht auch von einem *Overlay-Netzwerk*. Jeder Knoten ist darin für einen bestimmten "Bereich" zuständig.

Die Hashwerte werden dann mittels *konsistentem Hashing*[2] auf das Overlay-Netzwerk abgebildet. Dadurch existiert dann eine eineindeutige Zuordnung von Identifikatoren auf Knoten-IDs.

Einsatz einer verteilten Hashtabelle Zum Einsatz kommen verteilte Hashtabellen in Kombination mit einer *Speicher-API*. Dort werden Funktionen zum Speichern, Suchen und Löschen von $\langle Identifikator, Resource\rangle$-Paaren angeboten. Im Mittelpunkt steht die Suchfunktionalität die sich durch die Struktur des Overlay-Netzwerks und den eingesetzten Algorithmus definiert und in jedem Protokoll unterschiedlich ist. Gesucht wird immer nach einer Knoten-ID (und damit einem Knoten) die sich näher am Hashwert eines Identifikator befindet oder direkt dafür - im Sinne des Protokolls - zuständig ist.

3 Sicherheitsaspekte in verteilten Hashtabellen

Wie bei allen Netzwerkanwendungen die für nahezu jedermann, weltweit zur Verfügung stehen, ist es notwendig sich Gedanken über Sicherheit zu machen. Es wird immer teilweise bösartige Knoten geben, die vor einem bestimmten Hintergrund (Wahrung von Urheberrechten, Zensur, wirtschaftliche Interessen) entweder das Finden oder Erlangen zu verhindern oder die Existenz von Daten zu verschleiern versuchen werden. Im folgenden wird detailliert aufgelistet welche Sicherheitsaspekte untersucht werden.

Das Angreiffermodell sei wie folgt vereinbart: Der Angreifer kennt das eingesetzte Protokoll und kann das Verhalten seines Knotens nach belieben steuern. Er kann im Rahmen der Netzwerkkommunikation die Daten die durch bzw. über ihn geleitet werden lesen und verändern. Der Angreifer hat jedoch keine Möglichkeit die Kommunikation zwischen zwei entfernten Knoten zu lesen oder gar zu verändern und er hat nur begrenzt Zugriff auf Resourcen und Bandbreite.

I Falsche Routingweiterleitung Ein bösartiger Knoten könnte Routinganfragen die an ihn - im Sinne des Protokolls - gerichtet werden zu einem falschen oder nicht existierendem Knoten weiterleiten.

[2]Das ist eine Art des klassischen Hashing bei der sich das Mapping der Identifikatoren auf Knoten-IDs nicht oder nur geringfügig ändert, wenn neue Knoten hinzukommen oder Knoten das Netzwerk verlassen.

Lösung: Der *anfragende* Knoten muss den Suchprozess (nach)verfolgen können. Da in jedem Protokoll der nächste Schritt näher an das Ziel führen muss, kann der anfragende Knoten dieses Fehlverhalten feststellen und einen anderen Knoten befragen. Einschränkungen dieser Lösung könnten Erweiterungen im Protokoll sein, die es erlauben andere Knoten als nächsten zu kontaktieren, wenn dadurch die Latenz oder der Abstand, anders als durch das Protokoll vorgegeben, verkleinert werden kann. Dann müsste zusätzlich überprüft werden, ob diese Änderung regulär ist oder ob es sich um einen Fehler handelt. Die hier vorgestellte Lösung würde dadurch eingeschränkt.

II Falsche Zuständigkeit Ein bösartiger Knoten könnte im Suchprozess behaupten, dass ein beliebiger anderer (oder falscher) Knoten für einen Identifikator zuständig ist. Dieser Knoten würde dann die Existenz der Resource verneinen. Unter Umständen würde die Suche nach der Resource fehlschlagen.

Lösung: Einerseits sollte der anfragende Knoten sicherstellen, dass sich der ausgewiesene Knoten auch für den Identifikator zuständig fühlt. Das heißt, es muss überprüft werden, ob der Identifikator gemäß des Hashingalgorithmus zur gesuchten Resource passt. Und die Knoten-ID muss für andere Knoten *nachvollziehbar* und *sicher* zugewiesen werden. Das ist deshalb notwendig, da sonst nicht überprüft werden könnte, ob die Knoten-ID im Sinne des Protokolls zugewiesen worden ist.

III Gefälschte Routinginformation Da jeder Knoten seine Routingtabelle durch Anfragen an andere Knoten aufbaut, können Anfragen an einen bösartigen Knoten (der falsche Routinginformationen liefert) beim Update der eigenen Routingtabellen zu einer unbrauchbaren Routingtabelle führen. Dadurch würden dann Anfragen an falschen oder nicht existierende Knoten weitergeleitet werden, was wiederum zur Folge haben könnte, dass Resourcen nicht gefunden werden können.

Lösung: Es muss überprüft werden, ob die Knoten, die in die eigene Routingtabelle eingefügt werden sollen, existieren und die Eigenschaften des Protokolls eingehalten werden.

IV Bootstrapping Woher kann ein neuer Knoten wissen, ob der Knoten, den er kontaktiert, bösartig ist und ob er dem "richtigen" Netzwerk beitreten wird (siehe [1, Seite 4])? Einer oder mehrere Knoten könnten eine Partitionierung des P2P-Netzwerks anstreben, indem Sie sich als legitime Knoten ausgeben, dann jedoch nur falsche (oder von ebenfalls bösartigen Knoten) Routinginformationen weitergeben und so verhindern, dass der beitretende Knoten die anderen legitimen Teilnehmer oder bestimmte Resourcen finden kann.

Lösung: Der Zugang sollte nur über wohlbekannte und vertrauenswürdige Knoten erfolgen. Das steht jedoch im Widerspruch zu dem dezentralen Ansatz und würde bei einem großen Anteil bösartiger Knoten im P2P-Netzwerk die Partitionierung (und die dadurch resultierende Zensur oder Kontrolle) durch das Abschalten dieser zentralen Stellen erleichtern. Ein weiter Ansatz, um dieses Risiko zu minimieren würde in einer Public-Key-Infrastruktur liegen, um einerseits die Kommunikation zu verschlüsseln und andererseits die Authentizität der Knoten überprüfen zu können. Zufällige Routinganfragen sollten ebenfalls durchgeführt werden, um die eigene Sicht auf das P2P-Netzwerk mit der Sicht der anderen Knoten abzugleichen.

V Speicherverweigerung Ein bösartiger Knoten könnte die Existenz von Daten bestreiten, für die er im Sinne des Protokolls zuständig ist oder behaupten, die Daten zu besitzen, diese dann aber nicht an anfragende Knoten ausliefern.

Lösung: Replikation (Vermeidung von "single point of failure/responsibility"), anfragende Knoten müssen selbstständig Replikationen finden können (Invarianten des Proto-

kolls müssen eingehalten werden) und es müssen mindestens zwei Knoten nach dem selben Identifikator gefragt werden, um sicher zu sein, dass eine Resource nicht existiert. Replikation ist für sich genommen ein sehr komplexes Thema und wird deshalb nicht weiter erläutert.

VI Schnelles Verlassen und Beitreten Durch schnelles verlassen und beitreten von Knoten kann das Ausbalancieren und Verteilen der Daten (durch Replikate) eine große Bandbreite benachbarter oder anderer Knoten belegen. Das verringert die Effizenz und Performance des Systems und hat den maximalen Effekt, wenn der Knoten (der das Netzwerk verlässt bzw. beitritt) nicht an dem Datentransfer beteiligt ist.

Lösung: Die Daten auf jedem Knoten sollten in einem (kleinen) Rahmen gehalten werden, sodass eine Überlastung nur schwer eintreten kann.

Zusammenfassung In diesem Kapitel wurden sechs Probleme erläutert und jeweils eine oder mehrere Lösungen genannt. Gerade die Routing-Angriffe (falsche Routingweiterleitung, falsche Zuständigkeit, gefälschte Routinginformation) und das schnelle Verlassen und Beitreten in ein P2P-Netzwerk werden so täglich im normalen Gebrauch auftreten (gerade, wenn Knoten das Netzwerk verlassen) und werden durch das Protokolldesign abgefangen. Je mehr Knoten jedoch diese Angriffe durchführen können und wenn diese ihre Aktionen koordinieren, steigt die Wahrscheinlichkeit einen Teil des Netzwerks zu stören oder Informationen vor anderen Teilnehmern zu verbergen.

4 Analyse der Sicherheitsaspekte und Gegenmaßnamen in den untersuchten Protokollen

In diesem Kapitel wird für jeweils alle Aspekte aus Kapitel 3 und jedes der angesprochenen Protokolle (im einzelnen: CAN, Chord, Kademlia und Pastry) dargelegt, ob Angriffe über die erwähnten Sicherheitsaspekte durchführbar wären oder welche Maßnahmen im Protokolldesign diese unterbinden. Zusätzlich wird jeweils kurz der entsprechende Mechanismus oder Funktion erklärt und auf die entsprechende Stelle im Originalpapier verwiesen.

CAN Wie bereits erwähnt (Kapitel 2, Verteilung) erfolgt in CAN die Abbildung der Identifikatoren auf einen Punkt in einem d-Torus. Der Torus ist in Zonen aufgeteilt, wobei jeder Knoten für eine Zone (den dorthin gemappten Identifikatoren) zuständig ist.

Jeder Knoten hält und pflegt eine Routingtaballe, welche die direkten Nachbarn (von sich selbst) aus dem d-Torus enthält. Zusammen mit der IP-Adresse wird die Zone (in virtuellen Koordinaten) eines jeden Nachbarn gespeichert. Das Routing funktioniert durch das Weiterleiten der Anfrage an den Knoten (aus den eigenen Nachbarn), welcher dem Ziel in virtuellen Koordinaten am nächsten ist.

I Eine falsche Routingweiterleitung ist durchführbar, da durch Optimierung der Pfade - durch Messung der RTT (round trip time) - ohne Interaktion (bzw. Rückfrage) mit dem anfragenden Knoten der Zielknoten gesucht wird.[3, Kap. 3.3]

II Szenario II ist durchführbar, da der zweite Teil der Lösung nicht erfüllbar ist. CAN Knoten wählen sich selbst einen Punkt im d-Torus.[3, Kap. 2.2]

III Szenario III ist ebenfalls durchführbar, da es die RTT-Optimierung aus Performancegründen zulässt einen anderen Knoten, anstatt einen benachbarten, als nächsten auszuweisen. Dadurch kann die Invariante des Protokolls nicht mehr - durch den anfragenden Knoten - überprüft werden.

IV Bootstrapping wird durch das YOID-Protokoll[7] durchgeführt. Es wird angenommen, dass das CAN-Netzwerk unter einem wohlbekannten Namen über DNS aufgelöst werden kann und so neue Knoten eine IP-Adresse als Einstiegspunkt für das Netzwerk bekommen können.[3, Kap. 2.2]

V Durch sog. "zone overloading" und Replikation über mehrere Hashtabellen wird ein "single point of responsibility" unterbunden. Mehrere Knoten sind hierbei für die selbe Zone zuständig oder die Abbildung des Identifikators wird durch mehrere Hashtabellen auf unterschiedliche Knoten im Netz verteilt.[3, Kap. 3.4 + 3.5]

VI Beim Verlassen des Netzwerks ist der Knoten selbst dafür zuständig seine Daten an den am wenigsten belasteten Nachbarn abzugeben. Falls ein Knoten nicht mehr zu erreichen ist (und keine Übergabe stattgefunden hat), versuchen die benachbarten Knoten nach einem festgelegten Timeout die (ohne "zone overloading") nicht mehr besetzte Zone zu verwalten. Es kann vorkommen, dass das Netzwerk kurzzeitig inkonsistent wird, wenn viele benachbarte Knoten gleichzeitig ausfallen und weniger als die Hälfte der verbliebenen Nachbarn erreichbar ist. Es gibt jedoch Reparaturmechanismen in CAN, um eine weitere Fragmentierung zu unterbinden und die Konsitenz der Zonen wiederherzustellen.[3, Kap. 2.3 + 3.7]

Chord Das Overlay-Netzwerk in Chord ist Ringförmig und jeder Knoten hat neben dem direkten Nachfolger auch eine sog. *Finger-Tabelle*, um weiter entfernte Knoten ansprechen zu können. Neben den Knoten werden auch die Identifikatoren über die Hashfunktion auf dem Ring verteilt. Jeder Knoten ist für die Resourcen zuständig, die über die Identifikatoren, im Ring, *vor* ihm platziert werden.

Es kann vorkommen, dass einige Knoten mit Identifikatoren überlastet werden (wenn der Vorgänger weit entfernt ist). Deshalb wurden "virtuelle Knoten" eingeführt, die den eigentlichen physikalischen Knoten entlasten sollen. Bei einem Update der Routing-Tabelle wäre dann nur ein kleiner (virtueller) Teil der Knoten betroffen. Außerdem wird dadurch eine bessere Gleichverteilung der Knoten erreicht.

I Liefert ein Knoten in einem Chord-Ring bei der Suche nach dem Nachfolger eines Schlüssels die falsche Knoten-ID zurück, fragt der Suchende Knoten diesen nach der Resource. Der fälschlich befragte Knoten wiederum könnte feststellen (wenn er gutartig ist), dass er für diese Resource nicht zuständig ist und seinerseits in seiner Finger-Tabelle suchen, um die Anfrage an den nächst passenden Knoten weiterzuleiten.

Theoretisch könnte dieser Angriff eine Resource verbergen. Das gelingt jedoch nur, wenn alle anderen Knoten im Chord-Ring den bösartigen Knoten in ihrer Finger-Tabelle als nächsten (für die Resource zuständig) referenzieren und demzufolge alle Anfragen über diesen einen bösartigen Knoten laufen würden. Sobald ein anderer Knoten näher an der Resource ist und referenziert wird scheitert dieser Angriff.

II In Chord wird die IP-Adresse des Kontens gehasht, um eine Knoten-ID zu erzeugen. Auch hier gilt die gleiche Argumentation wie bei I. Dieses Angriffsszenario ist nur durchführbar, wenn viele Knoten um die gesuchte Resource herum unter der Kontrolle des Angreifers stehen.

III Knoten, die dem Chord-Ring beitreten, finden ihre Vorgängerknoten, indem sie einen "anderen" Knoten danach fragen. Anschließend wird überprüft, ob diese Knoten auch die Forderungen der Finger-Tabelle erfüllen und ob sie erreichbar sind. Damit ist dieser Angriff ausgeschlossen. Die Herkunft des "anderen" Knoten wird hierbei einer höheren Schicht überlassen.[4, Kap. 4.4]

IV Wie bereits erwähnt, wird das Finden von dem Chord-Ring zugehörigen Knoten der darüber liegenden Anwendung überlassen, die dieses Protokoll benutzt. Bei ei-

ner schlechten Implementierung kann dies also ausgenutzt werden, um den Ring zu partitionieren.

V Die Speicherverweigerung ist durchführbar, obwohl es sichere Knoten-IDs gibt. Ein Angreifer der viele virtuelle Knoten unter seiner Kontrolle hat (oder nur für viele Identifikatoren zuständig ist), könnte einen Teil der Resourcen verbergen. Es kommt hinzu, dass Replikation nicht explizit über das Protokoll geregelt wird und deshalb die Verfügbarkeit einer Resource nicht sichergestellt werden kann.[4, Kap. 4.2]

VI Das Chord-Protokoll ist entworfen worden, um Identifikatoren auf Knoten abbilden zu können. Es ist deshalb "nur" verantwortlich dafür, das Routing im Ring aufrecht zuerhalten. Weitergehende Verantworlichkeiten wie Authentifizierung, Caching oder Replikation ist der Anwendung überlassen (siehe [4, Kap. 3]). Es werden deshalb auch keine Resourcen im Netz auf andere Knoten übertragen. Lediglich die Zuständigkeiten ändern sich. Hier wurde jedoch gezeigt, dass das auch bei größeren Ausfällen das Routing gewährleistet werden kann.[4, Kap. 5.2 + 6]. Das schnelle Verlassen und Beitreten hat also keinen Effekt auf den Chord-Ring.

Kademlia Im Kademlia-Protokoll wird als zentrale Idee mittels XOR eine Abstandsfunktion definiert, um die Entfernung eines Knotens zu einem Identifikator zu ermitteln. Weiterhin werden die Knoten des Netzwerks als Blätter in einem binären Baum verstanden, in dem die Position eines Knoten durch den kürzesten Prefix seiner ID bestimmt ist.

Knoten-IDs und Identifikatoren werden über die selbe Hashfunktion auf diesen Baum abgebildet. Jeder Knoten speichert mehrere Routing-Tabellen (*k-buckets*, k ist eine obere Grenze der Anzahl), die die Adressen von Knoten enthalten, welche sich in einem anderen Teilbaum (definiert über die Abstandsfunktion) des Netzwerks befinden.

I Es werden immer - rekursiv - mehrere Knoten nach der Knoten-ID eines näher am Identifikator liegenden Knoten gefragt. Die Antworten werden über die Abstandsfunktion ausgewertet und die Knoten aus der Antwort, welche sich nun näher an dem gesuchten Identifikator befinden, befragt.

Liefert ein bösartiger Knoten eine falsche Knoten-ID zurück, findet diese nur Berücksichtigung, wenn der Abstand zum Identifikator klein ist. Der daraufhin nach der Resource gefragte Knoten wird a) entweder nicht antworten, weil er nicht existiert oder b) die Resource nicht liefern können. In beiden Fällen werden die nächsten, passenden Knoten-IDs (aus den vorherigen Antworten) verarbeitet und rekursiv nach einem näher an dem Identifikator liegenden Knoten befragt. Dieser Angriff kann den Suchprozess also nicht unterbrechen.[5, Kap. 2.2]

II Obwohl es keine sichere ID-Zuweisung gibt, kann die Argumentation von I herangezogen werden, um zu belegen, dass dieser Angriff keinen Schaden am System oder Suchprozess bewirken würde. Nur das "flächendeckende" belegen ganzer Hashregionen würde zu einem Erfolg für den Angreifer führen.

III Ein Knoten füllt seine Routing-Tabellen über die ihn erreichenden oder selbst verschickte Anfragen. Bei jeder neuen Knoten-ID wird geprüft, ob 1) der Knoten erreichbar ist (PING) und 2) wenn die Liste in welche der Knoten einsortiert werden würde voll ist, ob der am längsten in dieser Liste vorhandene Knoten nicht mehr erreichbar ist. Sobald eine dieser beiden Bedingungen nicht erfüllt ist, wird diese neue Knoten-ID verworfen und nicht in die eigene Routing-Tabelle übernommen. Falsche Routinginformationen werden also wenig Verbreitung finden.[5, Kap. 2.4]

IV Wie bei Chord wird das Finden eines dem Netzwerk angehörigen Knoten einer höherliegenden Anwendung überlassen. Es wäre also möglich, einen Knoten dazu zu bringen einem falschen Netzwerk beizutreten. Mit genügend anderen Knoten, die beim initialen Füllen der Routing-Tabelle als legitime Teilnehmer des Netzwerks

benannt werden, ist es möglich einen neuen Knoten zu täuschen und eine Partitionierung des Netzwerks zu erreichen.[5, Kap. 2.3]

V Die rekursive Suche nach einer Resource endet erst, wenn alle Knoten (die befragt wurden) keine Daten übermittelt haben oder ein Knoten die gewünschte Resource übermittelt hat. Weiterhin gibt es eine Caching-Funktion, in der Art, dass nach einer erfolgreichen Suche die Resource an dem Knoten gespeichert wird, der am nächsten zum Identifikator gelegen ist und keine Daten geliefert hat. Somit würde das hier beschriebene Szenario der Speicherverweigerung nur bei neuen Resourcen, die noch kein anderer Knoten bereitgestellt hat, funktionieren.[5, Kap. 2.3]

VI Betritt ein neuer Knoten das Netzwerk, muss er zuerst "lernen" welche anderen Knoten existieren. Diese lernen den neuen Knoten auch erst dadurch kennen. Der neue Knoten erhält dann nach und nach Resourcen von den umliegen (im ID-Raum, nicht physikalisch) Knoten, falls er dichter an einem Identifikator - als sie selbst - liegt. Beim Verlassen ist keine explizite Übergabe der Daten vorgesehen. Auch kann kein Knoten angewiesen werden Resourcen an einen anderen Knoten zu transferieren; die Knoten entscheiden dies selbst. Durch Schnelles Verlassen und Beitreten kann deshalb keine Bandbreitenüberlastung der anderen Knoten erreicht werden.

Eine Eigenschaft des Protokolls sei jedoch noch erwähnt: Jeder Knoten ist angewiesen nach einer bestimmten Zeitspanne (1 Stunde) die $\langle Identifikator, Resource\rangle$-Paare erneut bei anderen, ihm nahe liegenden, Knoten zu speichern. Könnte man diese Periode für viele Knoten synchronisieren, würde das Netzwerk periodisch eine sehr hohe Last aufweisen.[5, Kap. 2.5]

Pastry Pastry definiert ein kreisförmiges Overlay-Netzwerk (ähnlich wie Chrod) mit mehreren Ebenen. Die Identifikatoren werden bei dem *nummerisch* am nächst gelegenen Knoten gespeichert.

Es wird (ähnlich wie bei Kademlia) versucht, den zuständigen Knoten zu einem Identifikator zu finden, indem die Bits der Knoten-ID immer ähnlicher dem Identifikator gemacht werden. Außerdem wird eine "Nachbarschaftsbeziehung" definiert, die durch die höherliegende Anwendung implementiert wird. *Ping* oder *traceroute* sind bekannte Vertreter, um zu ermitteln, wie nah sich zwei Knoten sind.

I Da bei jedem Schritt die Knoten-ID näher am Identifikator liegen muss, würde eine falsche Weiterleitung im nächsten Schritt korrigiert werden oder die Invariante des Protokolls würde verletzt sein.

II Durch Replikate wird sichergestellt, dass eine Resource nicht nur von einem einzelnen Knoten geliefert werden kann. Diese Replikate werden beim Einfügen einer Resource über die sog. *Namensraummenge* (eine Menge von Knoten, deren ID einen gemeinsamen Prefix teilt) verteilt, sodass ein Angreifer alle Knoten eines Namensraums kontrollieren müsste, um eine Resource zu verbergen.[6, Kap. 2.2]

III Falsche Routinginformationen könnten sich kaum verbreiten, da der Prefix der Knoten-ID stimmen muss, um in die Routing-Tabelle aufgenommen werden zu können. Da die Knoten-ID jedoch vom Knoten selbst gewählt wird, verbirgt sich hier für einen Namensraum ein gewisses Risiko.[6, Kap. 2]

IV Bootstrapping wird auch hier wieder über einen "wohlbekannten" Knoten vereinbart, sodass eine schlechte Implementation dieser Funktionalität zu der besagten Partitionierung führen könnte.

V Durch Replikation ist sichergestellt, dass eine Resource nicht nur von einem Knoten abhängig ist. Siehe auch II.

VI Ähnlich wie im Kademlia-Protokoll müssen sich neue Knoten durch "join"-Nachrichten im System bekannt machen. Die Autoren von [6] zeigen selbst, dass es bei einer Anzahl von 100.000 Knoten ungefähr 1 KByte an RPC-Nachrichten benötigt, einen

Knoten bekannt zu machen. Auch in Pastry entscheidet jeder Knoten selbst, wann ein Knoten zur Nachbarschaftsmenge gehört und welche Resourcen repliziert werden müssen. Durch schnelles verlassen und beitreten eines Knotens kann also keine Überlastung erzeugt werden.

Zusammenfassung Die folgende Tabelle gibt einen Überblick über die hier genannten Ergebnisse. Ein "+" bedeutet, dass dieser Aspekt im Protokoll berücksichtigt wurde oder nicht ausgenutzt werden kann und ein Angriff keinen Erfolg hätte diesen auszunutzen. Ein "-" signalisiert, dass sich dieses Szenario durchführen lässt. Ein "?" zeigt an, dass ein erfolgreicher Angriff von mehr Faktoren als dem Protokolldesign abhängt und unter Umständen viele Resourcen (Geld, Knoten) und Bandbreite erfordert.

Tabelle 1: Übersicht über die Sicherheitsaspekte

	Protokolle			
	CAN	Chord	Kademlia	Pastry
I Falsche Routingweiterleitung	-	+	+	+
II Falsche Zuständigkeit	-	?	?	?
III Gefälschte Routinginformation	-	+	+	+
IV Bootstrapping	+	?	?	?
V Speicherverweigerung	+	-	+	+
VI Schnelles Verlassen/Beitreten	?	+	+	+

5 Zusammenfassung

In diesem Papier wurde untersucht, ob die in Kapitel 3 benannten Sicherheitsprobleme in verteilten Hashtabellen Auswirkungen auf die Protokolle CAN, Chord, Kademlia und Pastry haben und welche Eigenschaften des Protokolls ein Ausnutzen dieser Aspekte als Angriff unterbinden oder zulassen. Es hat sich gezeigt, dass gerade das Zuständigkeitsproblem (II) und Bootstrapping (IV) große Probleme bereiten kann. Die sichere und *überprüfbare* Zuweisung einer Knoten-ID überlassen alle Protokolle einer höher liegenden Anwendung. Für das Bootstrappingproblem wird nur in CAN eine Lösung vorgeschlagen; alle anderen Protokolle sprechen von einem "wohlbekannten" Knoten als Startpunkt für neue Knoten, was immer ein Risiko birgt.

Es bleibt zu zeigen, ob sich die Angriffsszenarien, die hier überprüft wurden, in der Praxis umsetzen lassen. Vor allem die Frage, ob sich diese für Zensur oder die Wahrung von Urheberrechten ausnutzen lassen, müsste weitergehend überprüft werden.

Literatur

[1] Emil Sit, R. Morris: *Security considerations for peer-to-peer distributed hash tables.*; 1st International Workshop on Peer-to-Peer Systems (Cambridge, Massachusetts 2002).

[2] Dan Wallach: *A survey of peer-to-peer security issues.*; In International Symposium on Software Security (Tokyo, Japan Nov. 2002).

[3] Sylvia Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker: *A scalable contentaddressable network.*; In Proceedings of ACM SIGCOMM (San Diego, California, Aug. 2001), pp. 161-172.

[4] Ion Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan: *Chord: A scalable peer-to-peer lookup service for internet applications.*; In Proceedings of ACM SIGCOMM (San Diego, California, Aug. 2001), pp. 149-160.

[5] Petar Maymounkov, D. Mazières: *Kademlia: A Peer-to-peer Information System Based on the XOR Metric.*; In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (Cambridge, Mar. 2002).

[6] Antony Rowstron, P. Druschel: *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.*; In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Nov. 2001).

[7] Paul Francis: *Yoid: Extending the Internet Multicast Architecture.*; Unpublished paper, available at http://www.aciri.org/yoid/docs/index.html, Apr. 2000.