Prosemninar

Peer-to-Peer Informationssysteme

bei

Professor Weikum

Thema

Privacy of Ideas in Peer-to-Peer Networks

Paper von Müller, Henrich, Eisenhardt

Write-up

von

Dieter Brunotte

Tutor:

Sebastian Michel

Inhaltsverzeichnis

1.Einführung/Motivation

Wissen ist Macht. Dies gilt vor allem in unserer modernen Informations- und Wissensgesellschaft. Dabei eröffnete vor allem das Internet neue Möglichkeiten bei der Informationssuche. Allerdings gibt man bei der Suche nach Informationen unfreiwillig sein Wissensbedürfnis preis. Dies gilt besonders, wenn man nach sehr spezifischen Informationen sucht. In vielen Anwendungsbereichen sollte dies jedoch vermieden werden, denn z.b. ein Forscher der eine neue Forschungsidee hat, möchte nicht, dass jemand anderes diese Idee erfährt, bevor er eine erste wissenschaftliche Abhandlung über dieses Thema publiziert hat. Denn schon das Downloaden eines Dokumentes zu einen bestimmten Thema durch einen bestimmten Nutzer kann Rückschlüsse auf die Idee zulassen. Deshalb ist es wünschenswert die Suche bzw. generelle Webdienste anonymisieren.

Allein das anonymisieren des Webdienstes ist aber nicht genügt. Auf der Suche nach Dokumenten gibt ein Benutzer eines solchen Webdienstes eine Query ein, eine Menge von Termen die die gewünschten Dokumente enthalten sollten. Diese Query bleibt bei der nur anonymisierten Suche dem Provider des Webdienstes natürlich nicht verborgen. Im Falle des Forschers, der eine neue Idee hat, wird natürlich vorher nachforschen wollen, ob diese Idee oder ähnliche wissenschaftliche Publikationen bereits existieren. Die Suchterme, die er als Query eingibt, geben meist jedoch die Idee des Forschers relativ präzise wieder und somit kann man die Query selbst als neue Idee ansehen. Somit sollte die Query dem Provider ebenfalls verborgen bleiben und wir brauchen einen Suchdienst, der diese Query ebenfalls noch zusätzlich verschleiert.

2.Privacy of ideas

2.1. Leakage

Unter Leakage versteht man das unbeabsichtigte Preisgeben einer Idee. Beispiel: Jemand sucht im Web bei einem Dienst wie CiteSeer oder NEC (internet search services für Publikationen in der Informatik) nach „Privacy and Web“, weil er auf diesem Gebiet Forschen will. Im Logfile des Providers (z.b. CiteSeer, NEC) erzeugt dies in der logfile des Providers folgende Logline:

```
138.15.10.13 - - [25/Sep/2002:20:05:58 +0200] \ "GET /cs?q=web+and+privacy"
```

Der Provider kann diese Logfiles nun benutzen, um Rückschlüsse auf die Idee der Nutzer zu ziehen. Hierbei sind vor allem die Queries interessant, die keine Ergebnisse zurückgeliefert hat, denn zu diesem Thema scheint es keine Publikationen zu geben und die zugrunde liegende Idee scheint neu zu sein.

2.2. Definition für Privacy of Ideas

Text Nachdem wir nun grob die neuen Anforderung formuliert haben, können wir die Privacy of Ideas definieren:

Definition: Ein Dienst bietet **Privacy of Ideas**, wenn er vollständig genutzt werden kann, ohne Informationen preiszugeben, die Rückschlüsseauf die Ideen des Nutzers zulassen.

Diese Forderung nach Privacy of Ideas ist nicht äquivalent zur Forderung nach Anonymität, denn wie bereits dargestellt, kann auch ein Nutzer, der anonym nach Informationen sucht, die Idee in Form seiner Suchquery zurücklassen. Daher muss zusätzlich zur Anonymität auch die Suchanfrage selbst verborgen werden, um solche Privacy leaks zu vermeiden.

2.3. Vorgehen für Privacy of Ideas

Um solche Privacy leaks zu vermeiden, können wir wie folgt vorgehen:

1. Anstatt eine einzige große Query durchzuführen, spalten wir diese in viele kleine Subqueries auf, die wir an Stelle der großen Query einzeln ausführen.
   Dabei sollte die Query jedoch möglichst so aufgeteilt werden, dass diese noch mindestenz ein Dokument als Ergebnis zurückliefert. Dies sollte deshalb der Fall sein, weil die Queries mit neuen Ideen wie unter Leakage beschrieben, diejenigen sind, die keine Dokumente zurückliefern und somit ist auch jede leere Subquery ein leak.
2. Die einzelnen Subqueries werden zeitlich dekorreliert versand.
   Dadurch wird sichergestellt, dass ein Angreifer, eine vollständige Query nicht aus zeitgleich ankommenden Subqueries wieder zusammenstellen kann
3. Die einzelnen Subqueries werden anonym versendet.
   Das anonymisierte Versenden von Nachrichten wird mittels Tarzan realisiert, das im nächsten Kapitel noch genauer beschrieben wird.
4. Das gewünschte Suchergebnis wird schließlich beim suchenden Peer selbst wieder aus den Ergebnissen der einzelnen Subqueries zusammengesetzt.
   Dadurch hat lediglich der suchende Peer das komplette Suchergebnis und die Privacy wird dadurch erhalten.

Ein erstes Baselinesystem , das dieses Vorgehen realisiert, ist Liane, das im 4.Kapitel näher beschrieben wird.

3.Basistechniken für Liane

3.1. Tarzan

Tarzan ist ein Peer-to-Peer Netzwerk Protokoll, das das anonymisierte Versenden von Daten in 2 Richtungen realisiert. Um den anonymisierten Datentransport zu realisieren, benutzt Tarzan Vorschlüsselungsverfahren, wie die asymmetrische Verschlüsselung. Bei dieser Art der Verschlüsselung wird für die Decodierung einer Nachricht ein anderer Schlüssel verwendet als bei der Encodierung. Dies hat den Vorteil, dass man den Schlüssel für die Encodierung (public key) jedem zugänglich machen kann, ohne zu riskieren, dass ein Angreifer nun meine codierten Nachrichten decodieren kann. Im Tarzannetzwerk hat jeder Peer einen eigenen public key, der jedem anderen Peer zugänglich ist, und einem passenden sekret key zur Dekodierung, der nur ihm selbst zugänglich ist. Beide keys werden beim Eintritt des Peers in das Netzwerk erzeugt.

Ein Peer N_{sender}, der einen Tarzan-anonymen Nachrichtenkanal zu einem Peer $N_{receiver}$ öffnen will, sucht sich dafür zufällig n Knoten $N_1,...,N_n$ aus dem Tarzannetzwerk aus. Eine Nachricht, die N_{sender} $N_{receiver}$ schicken will, verpackt N_{sender} in n Schichten asymmetrischer Verschlüsselung mit den public keys von $N_1,...,N_n$. Bei der Verschlüsselung geht N_{sender} wie folgt vor: er verpackt die an den Peer $N_{receiver}$ zu sendenden Daten zusammen mit der IP-Adresse von $N_{receiver}$ in ein Datenpaket und verschlüsselt mit dem public key von N_n. Dann nimmt er jeweils das gerade verschlüsselte Paket für einen Knoten verpackt dieses mit der IP-Adresse des Vorgängerknotens in ein Datenpaket und verschlüsselt es mit dem public key des Vorgängerknoten. Abb.1 skizziert den schematischen Aufbau einer Tarzan-Nachricht N_{sender} wie sie abschickt.

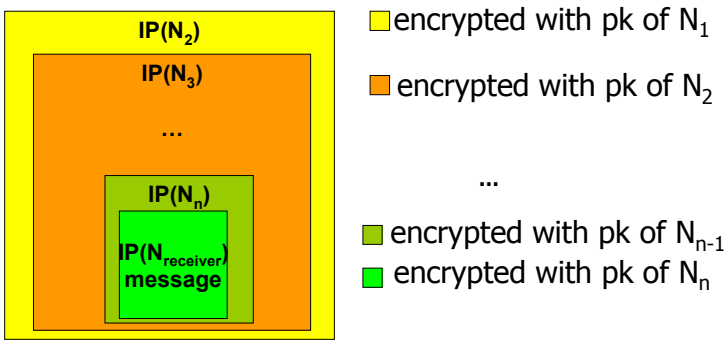

Abb.1

Dieses in n Schichten verschlüsselte Datenpaket sendet N_{sender} nun an N_1. N1 kann nun nur die erste Schicht mit seinem private key entschlüsseln und erhält so die IP-Adresse von N_2 und das in nun n-1 Schichten verschlüsselte Datenpaket für N_2. Dieser wiederum kann seinerseits eine weitere Schicht mit seinen sekret key entschlüsseln und ein in n-2 Schichten verschlüsseltes Datenpaket an N_3 senden. So entfernt jeder Knoten in der Tarzankette eine weitere Verschlüsselungsschicht, bis schließlich N_n die ursprüngliche Nachricht und die IP-Adresse von $N_{receiver}$ und kann die ursprüngliche Nachricht an $N_{receiver}$ weiterleiten.

Auf diese Weise können die Knoten N_1 bis N_n das eigentlich verschickte Datenpaket nicht lesen. Ebenfalls weis kein Knoten der wievielte Knoten der Tarzankette er ist, denn auch N_1 weiß nicht, ob sein Vorgänger, im Falle von N_1 der sendende Peer, N_{sender} oder einfach nur ein weiterer Peer in der Kette ist und somit ist für jeden Peer in der Tarzankette unklar welcher Peer der Sender der Nachricht ist, und die Anonymität bleibt bewahrt.

In der Regel ist es darüber hinaus notwendig, das N_{sender} von $N_{receiver}$ eine Antwort auf seine Nachricht erhält, wobei N_{sender} natürlich anonym bleiben muss, also $N_{receiver}$ muss eine Nachricht verschicken können, ohne zu wissen wohin er sie schicken muss. Dies Realisiert man, indem die Daten über die gleiche Tarzankette geschickt werden, aber in die andere Richtung, wobei jeder

Peer in der Tarzankette wieder eine Verschlüsselungsschicht hinzufügt. Hierbei werden allerdings andere symmetrische Schlüssel verwendet als auf dem Hinweg. Ist das Paket bei Peer N_{sender} angekommen, muss dieser es selbst wieder n mal entschlüsseln, um an die Nachricht zu gelangen. Damit wird der Arbeit für das Encodieren und Dekodieren zum größten Teil auf N_{sender} verlagert.

3.2 Invertierte Listen

Ein Dokument können wir als eine Menge von Wörtern betrachten, die in diesem Dokument vorkommen. Bei einer Keyword-Query sollen alle Dokumente gefunden werden, die diese Keywords enthalten. Es ist aber sehr ineffizient, für jede Query alle Dokumente nach den Keywords zu durchsuchen. Deshalb legt man nun für jedes Dokument eine Liste erstellt, die die Dokument bzw. eindeutig bestimmte Dokument-IDs enthält, in denen die Terme vorkommen. Dabei werden die Dokumente nach ihrer Ähnlichkeit zum Term gerängt, denn das Auftreten eines Terms sagt nicht unbedingt etwas über seine Wichtigkeit für das Dokument aus und können die Trefferdokumente so bereits vorsortieren.

Dies lohnt sich in der Praxis vor allem deshalb, weil öfter nach etwas gesucht wird als dass ein Dokument hinzugefügt oder modifiziert wird .

4.Liane

4.1. Funktionsweise von Liane

Um mit Liane eine anonyme, leakage freie Query durchzuführen, werden die bereits bestehen Systeme Tarzan, Chord und die invertierten Listen kombiniert. Dabei ist der Chord-Ring für das Verwalten der invertierten Listen zuständig. Chord realisiert das Lokalisieren gesuchter Daten in logarithmischer Zeit. Ein Peer, der eine Query $Q = \{q_1,..,q_m\}$ stellen möchte, muss dann die Chord-Knoten kontaktieren, die die invertierten Listen der Terme $q_1,..,q_m$ enthalten. Dies müssen allerdings nicht immer |Q| verschiedene Knoten sein.

Diese Kontaktierung der Knoten sowie das Lokalisieren der Listen muss natürlich anonym gehalten werden. Liane eröffnet für eine Query Q zunächst |Q| mit Tarzan anonymisierte Verbindung des anfragenden Peers zu |Q| zufällig ausgewählten Knoten im Netzwerk. Die Länge der Tarzankette soll |Q| sein, aber es spricht nichts gegen eine andere Länge, insofern die Tarzankette dann noch lang genug ist. Der anfragende Peer schickt dann pro Tarzan-Verbindung einen Queryterm über das Netzwerk. Ein Peer, der einen dieser Queryterme erhält, lokalisiert anstelle des anfragenden Peers nun mit Hilfe von Chord den Peer im Netzwerk, der die entsprechende invertierte Liste enthält. Der Peer mit der invertierten Liste wird nun an den anfragenden Knoten über die entsprechende Tarzanverbindung zurückgeschickt.

Nun weiss der anfragende Peer, wo welche Liste gespeichert ist und eröffnet neue Tarzanverbindungen, um diese darüber erneut anonymisiert herunterzuladen. Mit diesen invertierten Listen stellt sich der anfragende Peer selbst die Ergebnisdokumente der Gesamtquery zusammen. Somit kennt nur der anfragende Peer selbst das Ergebnis seiner Anfrage und hat anonym ohne Hinterlassen der Query selbst eine Anfrage ausgeführt.

4.2. Lianes Schwächen und Optimierung

Obwohl ich beschrieben habe, dass Liane Privacy of Ideas erfüllt, gibt es noch eine kleine, sehr theoretische Möglichkeit, Liane zu attackieren. Dafür ist es allerdings notwendig, dass der Angreifer im Lianenetzwerk von sehr viele Wörtern die invertierte Liste besitzt, und so vom Angreifer viele der Listen stammen, die der anfragende Peer benötigt. Der Angreifer kann dann einen Korrelationsanalyse durchführen. Aber hierfür gibt es einfache Abwehrmaßnahmen wie dummy-Anfragen (Anfordern einer nicht benötigten Liste über Tarzan) oder Cashing.

Die größere Schwäche von Liane ist aber die Resourcenverschwendung, die eine Liane-Query erzeugt. Hierfür sehen wir uns die Komplexität einer solchen Anfrage an:

$$Comp = (c_{connect} \cdot |Q| \cdot \log(|Q|) + c_{transfer} \cdot \sum_{\psi \in Q} |InvFileList(\psi)|)$$

Hierbei steht $c_{connect}$ für die Kosten zur Etablierung einer Verbindung, und $c_{transfer}$ für die Kosten ein Byte übers Netzwerk zu schicken. Und diese beinhalten auch mindestenz den Faktor |Q|, da die Daten über Tarzan-verbindungen geschickt werden.

Eine einfache Optimierung ist, dass wir die Query an stelle von Single term subqueries in Subqueries mit mehreren Termen aufspalten. Das Problem hierbei ist folgendes: machen wir die Subqueries zu groß, riskieren wir eine Leakage, denn diese kann bereits derart präzise sein, dass ihr Ergebnis leer ist; machen wir die Subqueries zu klein, kann die Resourcenverschwendung noch nicht stark genug reduziert werden. Um dieses Problem in den Griff zu bekommen, führen wir im nächsten Kapitel ein Kostenmaß ein, anhand dessen Liane optimiert wird.

4.3. Kostenmaß für Liane

Für das Kostenmaß werden wir 2 Hauptfaktoren betrachten:

1. c_{net}: Dies sind die Kosten für das Retrieval einer Dokumentreferenz über das Peer-to-Peer Netzwerk.
2. c_{leak} : Dies sind die Kosten, die uns durch ein Leakage unserer Idee entstehen. Wir können dies als den Wert unserer Idee ansehen.

Die Gesamtkosten für eine Query Q können wir wie folgt ausdrücken:

$$c_{total}(Q) = c_{leak} \cdot P(leak) + c_{net} \cdot |RR|$$

wobei |RR| die Anzahl der Gefundenen Dokumente in den Subqueries ist. Darüber hinaus ist c_{total} aber auch die Summe der Kosten für die einzelnen Subqueries und man kann dies wie folgt als Summe darstellen:

$$c_{total} = \sum_{j=1}^{m} c_{total}(Q_j)$$

Für eine bestimmte Query können wir jedoch die Wahrscheinlichkeit einer Leakage P(leak) wie folgt berechnen:

$$P(leak_{Q_j}) = \left(1 - \prod_{q_i \in Q_j} \frac{|InvFileList(q_i)|}{|Coll.|}\right)^{|Coll.|}$$

Hierbei ist die Wahrscheinlichkeit einer Leakage gleich der Wahrscheinlichkeit, dass in der Dokumentkollektion kein Dokument alle Terme der Query enthält, denn gerade diesen Fall sehen wir als Leakage an.

Die Formel für P(leak) lässt sich mit Hilfe der Formel $\lim_{N \to \infty} (1 + x/N)^N = e^x$ umformen, wenn wir N = |Coll.| und x = $-|Coll.| \cdot \left(\prod_{q_i \in Q_j} \frac{|InvFileList(q_i)|}{|Coll.|}\right)$ setzen,

denn normaler weise ist eine Kollektion groß genug, so dass wir bereits annähernd nahe genug bei e^x sind. So erhalten wir

$$P(leak_{Q_j}) \approx \mathrm{e}^{-|Coll.| \cdot \left(\prod_{q_i \in Q_j} \frac{|InvFileList(q_i)|}{|Coll.|} \right)}$$

und die Gesamtformel lautet nun:

$$c_{total}(Q_j) = c_{leak} \cdot \mathrm{e}^{-|Coll.| \cdot \left(\prod_{q_i \in Q_j} \frac{|InvFileList(q_i)|}{|Coll.|} \right)} + c_{net} \cdot |Coll.| \cdot \prod_{q_i \in Q_j} \frac{|InvFileList(q_i)|}{|Coll.|}$$

Das Produkt des 2. Summanden stellt hierbei die Zu erwartende Anteil an gesuchten Dokumenten in der Kollektion dar, multipliziert mit der Größe der Kollektion ergibt das die erwartete Größe der zu tranferierenden Daten.

Aus der umgeformten Formel kann man leicht ersehen, dass die Kosten, die ein Leakage unserer Idee verursachen, exponentiell mit der Anzahl der in den Subqueries gefundenen Dokumente sinken. Andererseits stiegen mit der Anzahl der gefundenen Dokumente aus den Subqueries die Tranferkosten. Um die Gesamtkosten gering zu halten, sollten es wegen der Leakage-Gefahr einerseits die Ergebnisse Subqueries möglichst groß sein, auf der anderen Seite sollte sie aber möglichst gering sein, um die über das Netzwerk zu versendende Datenmenge klein zu halten. Um die Gesamtkosten gering zu halten, muss nun eine geeignete „Kompromiss“-Partitionierung der Query in Subqueries gefunden werden.

5.Experimente mit Liane

Zu guter letzt haben die Athoren des Liane Papers noch einige Simulationen für Liane durchgeführt. Dabei simulierten sie ein PlanetP ähnliches Peer-to-Peer Netzwerk simuliert. Dabei benutzten sie als Kollektion 170000 News Artikel aus der Reuters collection. Jedes Dokument hat eine Größe von etwa 1-3 Kbyte und wurde von Stoppwötern befreit. Die Testqueries enthielten dabei eine Anzahl von n_k Querytermen, die in mindestenz k Dokumenten der Kollektion vorkommen müssen, und eine Anzahl von n_{all} Querytermen, die aus allen Termen der Kollektion ausgesucht wurden. Dabei wurde für jede Kombination an Werten von k, n_k, n_{all}, c_{leak}/c_{net} das Ergebnis von etwa 1000 Läufen gemittelt.

Zur Aufspaltung der Queries in Subqueries wurde laut Authoren ein einfacher Optimierungsalgorithmus verwendet, der aber nicht näher beschrieben wird. Dies ist wie ich finde, eine nicht ganz unerhebliche Lücke im Paper. Lediglich aus den Ergebnissen des ersten Experimentes kann man schließen, dass dieser das Verhältnis c_{leak}/c_{net} als Eingabe haben muss.

Als Ergebnis wurde zum einen in der Spalte Leaks/query group notiert, wie oft im Schnitt eine Leakage produziert wurde, also wie oft es eine Subquery ohne Ergebnisse gab. Die Spalte Communication savings gibt an, wieviel Datenverkehr prozentual im Vergleich zu den Single Term Queries eingespart wurde.

Im ersten Experiment wurde das Verhälnis c_{leak}/c_{net} variert. Dabei wurde festgestellt, das hohe Werte von c_{leak}/c_{net} die Anzahl der Leaks stark reduziert, schon bei 10000 war der Wert annähernd 0. Allerdings nimmt auch die Kommunikationersparnis rapide ab: Hatten wir bei kleinen Werten über 90%, waren es bei 10000 gerade mal 65%. (vgl. Tab 1).

k	n_{all}	n_k	c_{leak}/c_{net}	*Leaks/Query group*	*Communication Saing*
100	0	5	100000	0.027	0.560
100	0	5	10000	0.067	0.656
100	0	5	1000	0.397	0.831
100	0	5	100	1.596	0.988
100	0	5	10	1.620	0.987

Das 2.Experiment variert das Verhältnis von n_k und n_{all} . Mit dem Anteil seltener Terme sinkt die Kommunikationsersparnis - bei 100% seltenen Termen sogar fast auf 0. Allerdings sinken etwas unerwartet die Privacy leaks mit der Anzahl seltener Terme.

k	n_{all}	n_k	c_{leak}/c_{net}	*Leaks/Query group*	*Communication Saing*
100	5	0	100000	0.007	0.023
100	4	1	100000	0.027	0.115
100	3	2	100000	0.039	0.367
100	2	3	100000	0.052	0.475
100	1	4	100000	0.050	0.591

Das 3. und letzte Experiment zeigt, das man sehr viel Datenverkehr sparet, wenn die Suchterme in jeweils sehr vielen Dokumenten vorkommen. Die Privacy leak Raten schwanken leicht zwischen 0.5 und 0.1 (Jede 10.- 20. Query hat einen leak), aber bei extrem Hohen Raten sind sie bei 0.

k	n_{all}	n_k	c_{leak}/c_{net}	*Leaks/Query group*	*Communication Saing*
250	0	5	10000	0.086	0.808
500	0	5	10000	0.135	0.924
1000	0	5	10000	0.066	0.956
2000	0	5	10000	0.045	0.981
4000	0	5	10000	0.041	0.936
8000	0	5	10000	0.113	0.995
16000	0	5	10000	0.000	0.999

6.Zusammenfassung

Wir haben folgendes gesehen:

1. Was ist Privacy of Ideas und warum wir sie brauchen. Dabei lag die Idee zugrunde, dass die Terme einer Suchanfragen eine neue Idee wiedergeben.
2. Wir wollten dafür anonymisierten Datenverkehr, der mit Tarzan realisiert wird. Anonymiesiert allein ist aber nicht genug, da dies die Idee nicht verbirgt.
3. Deshalb wurde mit Liane ein erstes Baselinesystem entwickelt, das die Privacyof Ideas erfüllt. Diese System verband Tarzan, Chord und die invertierten Listen.
4. Dieses System war aber noch nicht optimal, da eine Anfrage einen immensen Datentranfer verursacht. Deshalb wurde es mit einem Kosten maß optimiert, das sowohl die Transferkosten als auch die Leakage einer Idee berücksichtigt.
5. Als letztes wurden dann noch Experimente mit Liane durchgeführt.

Literaturverzeichnis

[1] Müller, Henrich, Eisenhardt: Privacy of Ideas in P2P Networks. In: Gronau, Benger (edts.): JXTA Workshop. Potentiale, Konzepte, Anwendungen. GITO Verlag, Berlin, 2003.

[2] Eisenhardt: Privacy of Ideas in P2P Networks. Workshop JXTA. Potentiale, Konzepte, Anwendungen. Technische Universität Berlin, Germany, 11 November 2003.