Technische Universität Berlin
Seminar Network Measurement
Betreuer: Bernhard Ager und Jörg Wallerich

Bitmap Algorithms for Counting Active Flows on High Speed Links

Zusammenfassung

Der folgende Text ist eine Zusammenfassung der Veröffentlichung: "Bitmap Algorithms for Counting Active Flows on High Speed Links" von Cristian Estan, George Varghese und Mike Fisk für die Internet Measurement Conference 2003 [1]. Das Paper ist angefertigt für das Seminar "Network Measurement", an dem Lehrstuhl für Intelligente Netze und Management verteilter Systeme (INET), Research Group Prof. Anja Feldmann, an der Technischen Universität Berlin. Es wird hier eine Familie von Bitmapalgorithmen zur Zählung von Flows in Netzwerken beschrieben, die sowie wenig Speicher verbrauchen als auch geeignet sind für Hochgeschwindigkeitsoperationen.

Elisa Jasinska
jasinska@informatik.hu-berlin.de
4. Juli 2007

1 Einleitung

Der folgende Text beschreibt eine Familie von Bitmapalgorithmen zum Zählen unterschiedlicher Headerpattern, sogenannter “Flows”, auf Hochgeschwindigkeitslinks.

Das Zählen von Flows zu ihrer Ankunftszeit stellt eine grosse Herausforderung dar, vor allem in Anbetracht der ständig schneller werdenden Verbindungen. OC-768 Geschwindigkeiten (40Gbit/sec) werden demnächst eingeführt und die Standardisierung von 100Gbit Ethernet ist momentan ein wichtiges Thema unter grossen Netzwekbetreibern, wie zum Beispiel ISPs[1] oder IXPs[2]. Daher muss jegliche Art von Inspektion, zum Beispiel das Zählen von Paketen, welches Voraussetzung für das Zählen von Flows ist, ein hohes Maß an Effizienz aufweisen.

Ein “Flow” wird hier durch sich nicht verändernde Werte in bestimmten Feldern der Paket Header beschrieben. Nehmen wir zum Beispiel die Quell und Ziel IP Adresse um unseren Flow zu definieren, so würde die Anzahl unterschiedlicher Quell/Ziel IP Paare die Anzahl der möglichen Flows darstellen. Da in diesem Fall Verkehr zwischen zwei Hosts auftauchen und wieder verschwinden kann, wird zudem zwischen “aktiven Flows” und “inaktiven Flows” unterschieden. Hierfür muss ein Timeout definiert werden, also eine Zeitbegrenzung ab wann ein vorher aktiver Flow wieder als inaktiv gilt.

Unter anderem war das Ziel der Einführung von neuen Algorithmen ein möglichst geringer Verbrauch von Speicherressourcen, so dass man auf die Verwendung von langsamem DRAM[3] verzichten kann und stattdessen schnelleren SRAM[4] oder auch Prozessorregister ausreichend sind. Bei der Verwendung von Bitmaps muss zudem nur ein einzelnes Bit gesetzt werden und auf den üblichen Prozess von lesen, ändern und schreiben eines Wertes kann verzichtet werden.

In Kapitel zwei wird zunächst das Problem und die Motivation genauer beschrieben. Kapitel drei geht dann auf die Bitmapalgorithmen ein wärend sich dann Kapitel vier mit ihren statistische Eigenschaften genauer befasst. Kapitel fünf beschreibt Testergebnisse und Schlussfolgerungen und abschliessend werden in Kapitel sechs die Resultate noch einmal zusammengefasst.

[1] Internet Service Provider
[2] Internet Exchange Point
[3] Dynamic random access memory
[4] Static random access memory

2 Problembeschreibung

Ein naiver Ansatz zum Zählen von unterschiedlichen Flows eines Links lässt sich wie folgt darstellen: für jedes neue Quell/Ziel IP Paar wird ein Zähler inkrementiert, zudem wird eine Hashtabelle verwaltet, in welche die bekannten IP Paare geschrieben werden. Da die Größe eines Quell/Ziel IP Paares 64 Bit beträgt und man auf heutzutage aktuellen Hochgeschwindigkeitslinks Millionen von Flows antreffen kann, würde dieser Ansatz eine Menge schnell zugreifbaren Speichers benötigen.

Das Zählen von unterschiedlichen Flows in einem bestimmten Zeitintervall kann zu unterschiedlichen Zwecken eingesetzt werden. Ein Intrusion Detection System (IDS) zum Beispiel, welches Portscans erkennen will, könnte davon ausgehen, daß jede Quell IP Adresse, die mehr als 3 Flows in 12 Sekunden zu unterschiedlichen Ziel IP Adress und Port Paaren öffnet, einen Portscan durchführt. Da jedoch verschiedene Applikationen unterschiedliche Anforderungen an die Definition eines Flows haben können, wird hier jede mögliche Art von Kennung für einen Flow in Betracht gezogen (IP Adressen, Ports, etc.), die sogenannte “FlowID”.

Portscans

Wenn eine Quelle zu viele Verbindungen innerhalb eines kurzen Zeitraumes zu unterschiedlichen Zielen öffnet könnte man von einem Portscan ausgehen[5]. Das weit verbreitete Intrusion Detection System Snort [2] verwaltet zur Portscanerkennung eine Tabelle mit allen aktiven Verbindungen. Dies ist unnötiger Aufwand, da die meisten Verbindungen nicht zwingend Portscans sein müssen und die Anzahl von Quell IPs im Allgemeinen sehr hoch sein kann. Ein System, welches diese Zählung effizient realisiert, wäre hier wünschenswert.

Denial of Service Attacken

Die Anzahl von aktiven Flows kann ebenfalls zur Erkennung von Denial of Service Attacken (DoS)[6] verwendet werden, wie David Plonka mit seiner Software FlowScan [3] zeigt. Da die Quell-IP-Adressen bei DoS-Attacken oft beliebig gewählt werden, kann man bei einer hohen Anzahl Quell-IPs, die alle Verbindungen zu dem selben Ziel öffnen, davon ausgehen dass eine DoS-Attacke stattfindet. Plonkas Analysen beruhen jedoch auf NetFlow-Daten [4], welche in grossen Netz-

[5] Verteilte Portscans, die von unterschiedlichen Quellen ausgehen, sind ebenfalls möglich, hier allerdings nicht betrachtet.

[6] Denial of Service sind Attacken bei denen durch zum Beispiel übermässige Anfragen eines Dienstes dieser nicht mehr zum beabsichtigten Gebrauch zur verfügung steht.

werken nicht skalieren. Mit der Verwendung von schnelleren Bitmaps könnte man diese Methode verbessern.

Generelle Messungen

Generelle Messungen von Netzwerkverkehr können sinnvolle Informationen enthalten. Zum Beispiel ist die Messung von unterschiedlichen Protokollen interessant, um ein akkurates Bild über deren Verwendung zu erhalten. Ausserdem sind solche Analysen zur Dimensionierung von z.B. verschiedenen Routercaches nützlich.

Ausbreitungsrate von Würmern

Anfang August 2001 wurde versucht, durch das Sammeln von Paketen mit bestimmten Headerpattern den Wurm "Code Red" [5] zu verfolgen. Um die Ausbreitungsrate des Wurmes zu bestimmen, musste die Anzahl der unterschiedlichen Quell-IPs gezählt werden, dies wurde damals ineffizient durch eine Hashtabelle und Logfiles realisiert.

Paketscheduling

Einige existierende Schedulingalgorithmen versuchen, sämtliche weiterzuleitenden Flows auf faire Weise über die zu Verfügung stehende Bandbreite zu verteilen. Hierzu wird jedoch ebenfalls die Information benötigt, wie viele Flows überhaupt vorhanden sind, um den Vorrang gerecht zu bestimmen. Da es für Routinghardware auf Grund der limitierten Ressourcen und der grossen Anzahl an kleinen Flows nicht effizient ist, Flowzustände zu speichern, sind solche Geräte hierbei auf externe Informationen angewiesen. Die Möglichkeit diese Berechnungen effizient auf der eigentlichen Hardware durchzuführen würde Anfälligkeiten, wie zum Beispiel den Ausfall eines benötigten Informationssystems und die dadurch fehlende Aktualisierung der Daten, reduzieren.

3 Algorithmen

Die hier vorgestellte Familie von Bitmapalgorithmen aktualisiert über einen festen Zeitraum hinweg eine Bitmap und kalkuliert am Ende des Zeitraumes eine Schätzung für die Anzahl der aktiven Flows. Da keine Zustandsanalyse durchgeführt wird und der Wertebereich durch die Hashfunktion reduziert ist, es also zu Kollisionen kommen kann, sind alle Angaben Schätzungen, deren Genauigkeit allerdings durch Tests verifiziert wurde (siehe Kapitel 5).

Die Familie besteht aus drei Kernalgorithmen: Direct Bitmap, Virtual Bitmap und Multiresolution Bitmap sowie zwei daraus abgeleiteten Algorithmen. Direct Bitmap und Virtual Bitmap sind bekannte Algorithmen, die hier jedoch von grosser Bedeutung sind, da die neuen Algorithmen auf ihnen basieren. Daher werden sie hier erneut vorgestellt.

3.1 Direct Bitmap

Direct Bitmap ist ein einfacher Algorithmus, der die Anzahl der Flows auf einem Link abschätzt. Zunächst wird eine Bitmap mit Nullen initialisiert. Eine Hashfunktion wird auf die FlowID angewendet, wobei das Ergebnis dieser Funktion genau einer Position in unserer Bitmap entspricht, welche dann auf eins gesetzt wird. Alle Pakete die zu dem selben Flow gehören setzen die selbe Stelle in der Bitmap. Daher entspricht die Anzahl der gesetzten Felder zum Schluss der Anzahl der gesehenen Flows (siehe Abbildung 1).

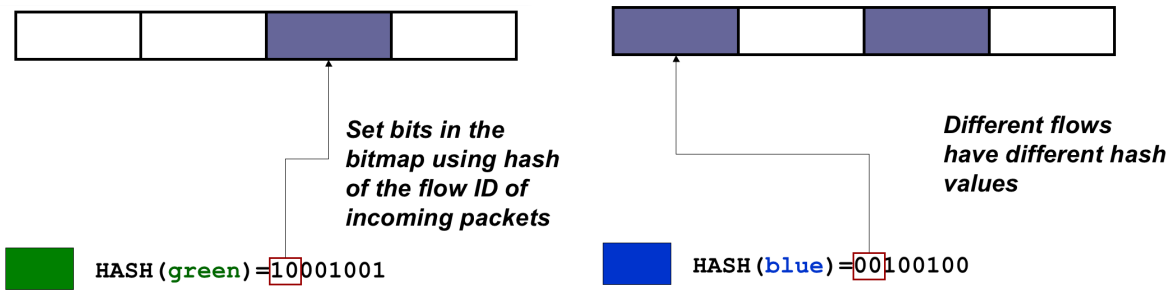

Abbildung 1: Direct Bitmap

Da jedoch die Reduktion des Wertebereiches durch die Hashfunktion dazu führen kann, dass zwei unterschiedliche FlowIDs zum gleichen Ergebnis führen (eine sog. Kollision, siehe Abbildung 2), ist eine Berechnung der Flowanzahl auf diese Weise ungenau. Um diese Ungenauigkeit zu minimieren, wird die Wahrscheinlichkeit einer Kollision in die Abschätzung der Flowanzahl mit einberechnet.

Diese Abschätzung weicht jedoch immer noch sehr von der Wirklichkeit ab, wenn die Bitmap voll wird, denn je mehr Bits belegt sind, desto höher ist die Wahrscheinlichkeit, dass Kollisionen aufgetreten sind. Die optimale Fülle der Bitmap um die Kollisionen korrekt abschätzen zu können beträgt ungefähr 50%. Die Bitmapgrösse müsste daher linear mit der Zahl der Flows skalieren, was wiederum viel Speicher verbrauchen würde.

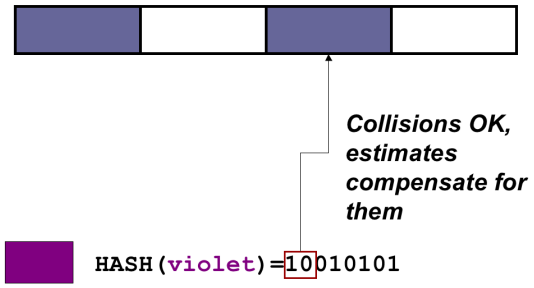

Abbildung 2: Direct Bitmap Kollision

3.2 Virtual Bitmap

Virtual Bitmap reduziert den Speicherverbrauch, indem es nur einen Teil der Bitmap speichert, den Direct Bitmap für ein akkurates Ergebnis benötigen würde, (siehe Abbildung 3) und die Anzahl der gesetzten Bits in der Bitmap hochrechnet. Man könnte dies auch als Abtasten des FlowID Raumes bezeichnen. Da der Speicherverbrauch der Bitmap gleich bleibt, wird verhältnismässig der Teil des FlowID Raumes, den wir betrachten, kleiner, je grösser die Anzahl der Flows ist. Dies benötigt allerdings Vorwissen über die Anzahl der zu betrachtenden Flows. Wenn die Abtastrate ungeschickt gewählt ist, führt das zu ungenauen Ergebnissen.

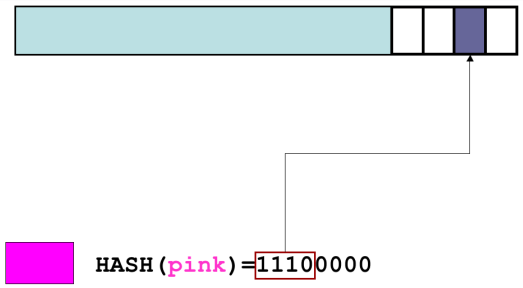

Abbildung 3: Virtual Bitmap

Angenommen die Virtuelle Bitmap speichert 1% des tatsächlich möglichen FlowID Raumes und wir haben 50 aktive Flows. Wenn keiner dieser Flows ein Bit in der Bitmap setzt, wird davon ausgegangen, dass kein Flow vorliegt. Wenn ein Bit gesetzt wird, würde von einer Anzahl von 100 Flows ausgegangen. Niemals jedoch hätten wir tatsächlich das Ergebnis 50 vorzuliegen.

3.3 Multiresolution Bitmap

Um das benötigte Vorwissen bei Virtual Bitmaps zu umgehen, könnte man mehrere verschiedene Virtual Bitmaps verwenden, jede mit dem gleichen Speicherverbrauch, aber unterschiedlichen Abtastraten (siehe Abbildung 4).

Abhängig von der Anzahl der gesetzten Bits benutzen wir im Endeffekt die Bitmap die uns das genauste Ergebnis liefern wird (also die, die ungefähr zu 50% gefüllt ist). Die Bitmap mit der geringsten Auflösung (in Abbildung 4 ganz unten) funktioniert prima bei wenigen Flows, je mehr Flows vorkommen desto besser ist es eine Bitmap mit höherer Auflösung zu wählen.

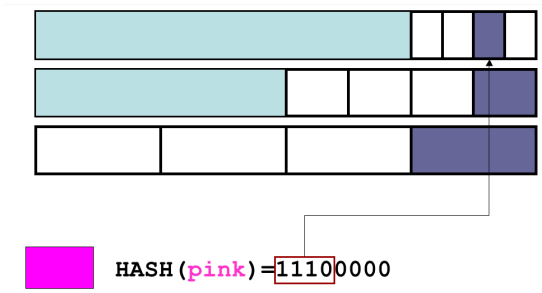

Abbildung 4: Multiple Bitmaps

Das Problem beim Benutzen mehrerer Bitmaps ist, dass wir nun mehr Speicherzugriffe benötigen, um alle Bitmaps zu aktualisieren und dass der gesamte Speicherverbrauch sich mit der Anzahl der Bitmaps vervielfältigt. Um dies effizienter zu gestalten wird die Multiresolution Bitmap eingeführt.

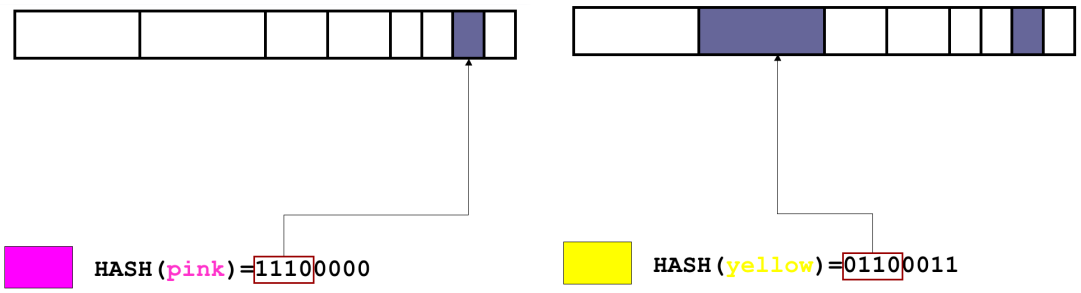

Abbildung 5: Multiresolution Bitmap

Die Multiresolution Bitmap besteht aus verschiedenen Teilen mit verschiedenen Auflösungen (siehe Abbildung 5). Sie vereint die Vorteile von multiplen Virtual

Bitmaps mit nur einem Speicherzugriff, da wir nun nicht mehr mehrere Bitmaps aktualisieren müssen.

Aus der Multiresolution Bitmap können wir Multiple Bitmaps herleiten, indem wir die Teile mit jeweils geringeren Auflösungen miteinander ODER-verknüpfen. In Abbildung 5 würde man zum Beispiel die letzten vier Felder ODER-verknüpfen und die zwei mit der nächst höheren Auflösung, um eine Bitmap mit der höhsten Auflösung zu erhalten. In den letzten vier Feldern ist ein Bit gesetzt, was nach der Verknüpfung wiederum zu einem gesetzten Bit führen würde. Die zwei Felder davor sind beide leer, somit ist das verknüpfte Feld ebenfalls leer.

Diesen Schritt kann man für beliebeige Auflösungen wiederholen und erhält somit wieder mehrere Bitmaps wie in Abbildung 4. Nun kann man wie schon bei den Multiplen Bitmaps jenen Abschnitt verwenden, welcher das genauste Ergebnis liefern wird.

3.4 Weitere Algorithmen

Aus den bisher vorgestellten Algorithmen wurden weitere Ableitungen definiert, die jeweils ein bestimmtes Problem adressieren. Diese Kombinationen werden hier nur kurz vorgestellt, da sie im Grunde auf den schon bereits erläuterten Algorithmen basieren.

Adaptive Bitmap

Adaptive Bitmap kombiniert die Genauigkeit von Virtual Bitmap mit der Robustheit von Multiresolution Bitmap wobei es sich auf eine stetige Anzahl von Flows verlässt.

Messungen haben ergeben, dass die Anzahl der Flows in verschiedenen Messintervallen sich nicht dramatisch verändert[7]. Adaptive Bitmap nutzt eine grosse Multiresolution Bitmap zur Ermittlung eines Richtwertes für die Anzahl der Flows, in einer weiteren Iteration wird dann die Genauigkeit einer kleinen Virtual Bitmap mit der Anzahl der vorher ermittelten Flows eingestellt.

Um zwei Schreiboperationen zu vermeiden, kann die Virtual Bitmap in die Multiresolution Bitmap integriert werden. An welcher Stelle genau in der Multiresolution nun die Virtual Bitmap plaziert wird, hängt von der vorherigen Kalkulation der

[7] Dieser Algorithmus eignet sich nicht für Analysen die eine drastische Veränderung von der Flowanzahl voraussetzen, wie zum Beispiel das Erkennen DoS Attacken.

Flowanzahl ab. Dieser Algorithmus zählt 0 bis 100 Millionen Flows mit einem durchschnittlichen Fehler von $< 1\%$ bei nur 2 Kbytes Speicherverbrauch.

Triggered Bitmap

Triggered Bitmap nutzt Direct Bitmap und Multiresolution Bitmap um den Speicherverbrauch zu reduzieren. Zunächst wird immer eine kleine Direct Bitmap verwendet, überschreiten die Bits in dieser eine bestimmte Anzahl, wird stattdessen eine grosse Multiresolution Bitmap eingesetzt.

Bei Portscans zum Beispiel öffnet eine Quelle sehr viele Verbindungen, diese wird dann in ein Multiresolution Bitmap geschrieben. Zum Ende werden die Anzahlen aus dem Direkt Bitmap mit denen aus dem Multiresolution Bitmap addiert. Das führt zu einem genauen Ergebnis, und generiert nur mehr Speicherverbrauch, wenn tatsächlich ein Portscan auftritt.

4 Statistische Eigenschaften der Algorithmen

Dieses Kapitel gibt einen Überblick über die statistischen Eigenschaften, bzw. die erwartete Genauigkeit der vorgestellten Bitmapalgorithmen. Jedoch ist immer je nach Anwedungszweck (Vorwissen über die zu messenden Daten, verwendete Hashfunktion, etc.) zu beachten, dass die Ergebnisse stark variieren können.

4.1 Direct Bitmap

Zunächst wird beschrieben, wie man aus einer Direct Bitmap die tatsächliche Anzahl der Flows berechnet, und welchen Fehler man dabei zu erwarten hat.

Anzahl der Flows

Um aus der Anzahl der gesetzten Bits in einem Direct Bitmap die Anzahl der Flows zu berechnen, müssen wir, wie schon in Kapitel 3.1. erwähnt, Kollisionen beachten.

Sei b die Grösse der Bitmap. Die Wahrscheinlichkeit, dass ein bestimmter Flow einem bestimmten Bit zugeordnet wird ist: $p = 1/b$. Sei n die Anzahl der aktiven Flows, so ist die Wahrscheinlichkeit, dass kein Flow zu einem bestimmten Bit zugeordnet wird $p_z = (1-p)^n \approx (1/e)^{n/b}$. Dies führt zu erwarteten Anzahl an nicht gesetzten Bits zum Ende eines Messintervalls $E[z] = bp_z \approx b(1/e)^{n/b}$. Mit der

Anzahl der Null-Bits in z können wir nun anhand von Gleichung (1) die Schätzung[8] $\hat{n}$ für die Anzahl der aktiven Flows berechnen.

$$\hat{n} = b \ln \left(\frac{b}{z} \right) \tag{1}$$

Erwarteter Fehler

Da die Anzahl der Kollisionen zufällig ist, liefert die oben stehende Gleichung nicht immer ein akkurates Ergebnis. Um die Genauigkeit des Algorithmus zu bestimmen, wollen wir feststellen, wie weit unser errechneter Wert $\hat{n}$ von der tatsächlichen Anzahl der Flows n abweicht.

Hierzu benötigen wir den Wert der Flowdichte ρ, welches die durchschnittliche Anzahl von Flows ist, die einem Bit zugeordnet werden. Mit Gleichung (2) kann nun die Standardabweichung des Quotienten $\hat{n}/n$ angenährt werden, also beschrieben mit welcher Wahrscheintlichkeit die Ungenauigkeit nicht groesser als erwartet sein wird.

$$SD \left[\frac{\hat{n}}{n} \right] \approx \frac{\sqrt{e^{\rho} - \rho - 1}}{\rho \sqrt{b}} \tag{2}$$

4.2 Virtual Bitmap

Die Berechnungen für die Virtual Bitmap basieren auf dem Fakt, dass eine Virtual Bitmap nur einen Teil einer Direct Bitmap darstellt und die gesetzten Bits einer Direct Bitmap gleich verteilt sind. Somit kann man aus der Menge der gesetzten Bits auf die Menge schliessen, die in einem Direct Bitmap gesetzt wäre und dann wiederum mit Gleichung 1 auf die Anzahl der Flows kommen.

Der durchschnittliche Fehler bei Virtual Bitmap kann durch eine geschickt gewählte Flowdichte minimiert werden. Wenn die Flowdichte zu gering ist führt das zu Fehlern auf Grund des Abtastens, ist die Flowdichte zu hoch entstehen Fehler durch Kollisionen. Optimalerweise sollte die Flowdichte $\rho_{optimal} = 1.593624$ betragen, was bedeutet, dass $20,3\%$ der Bits in der bitmap nicht gesetzt sein sollten.

[8] Maximum-Likelihood-Methode siehe [8].

4.3 Multiresolution Bitmap

Multiresolution Bitmap verhält sich ähnlich wie Virtual Bitmap, zu jeder Teilauflösung können wir einzeln die Flowanzahl und den Fahler berechnen. Durch optimale Wahl eines Teiles (welches wiederum ein Virtual Bitmap dastellt, also wieder $\rho_{optimal} = 1.593624$ gilt) können wir wie in Kapitel 4.2. beschrieben die Flowanzahl und den durchschnittlichen Fehler berechnen.

5 Ergebnisse

Um die errechnete Genauigkeit der Algorithmen zu überprüfen und den geringeren Speicherverbrauch festzustellen, wurden Versuchsreihen durchgeführt. Im folgenden wird auf die Genauigkeit der Fehlerberechnung am Beispiel von Virtual Bitmap eingegangen und der Vergleich des Speicherverbrauchs bei der Portscanerkennung von Triggered Bitmap, Snort und Wahrscheinlichkeitsberechnungen gezeigt.

5.1 Versuchsumgebung

In den Testreihen wurden 3 verschiedene Datensätze verwendet: einer von CAIDA [6], erfasst am 6. August 2001 an einem OC-48 Link und zwei vom MOAT Projekt [7] erfasst an zwei Knotenpunkten von Universitäten ans Internet. Das Messintervall wurde auf 5 Sekunden festgelegt, da es grösser als die maximale Roundtrip Zeit ist die wir im Internet erwarten. Ein Flow wird in allen Experimenten als ein 5-Tupel definiert, bestehend aus: Quell und Ziel IP Adresse, Quell und Ziel Port und Protokoll. In allen Tests wurden CRC-basierte Hashfunktionen verwendet.

5.2 Fehlerberechnung Virtual Bitmap

Der erwartete Fehler von Virtual Bitmaps wurde mit tatsächlichen Fehlerraten aus den Tests verglichen (siehe Abbildung 6). Die Tests haben ergeben, dass die theoretische Berechnung des Fehlers meist eine leicht negativere Darstellung liefert, als man es in der Wirklichkeit antrifft, der Fehler also geringer ausfällt als erwartet.

Die genausten Ergebnisse erhält man bei einer Flowdichte von $\rho = 1.6$, wie schon in Kapitel 4.2. beschrieben.

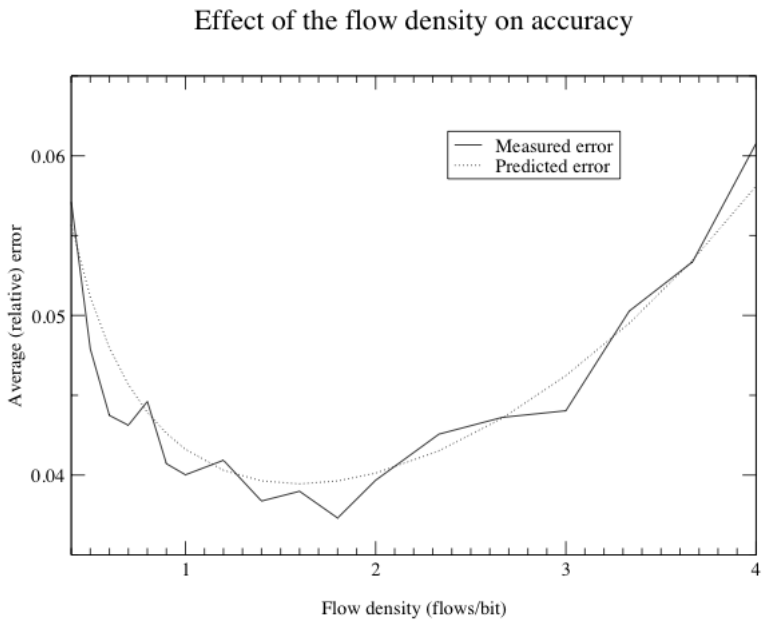

Abbildung 6: Virtual Bitmap Error

5.3 Portscanerkennung Triggered Bitmap

Um ein praxisnaheres Beispiel zu zeigen, wurde der Speicherverbrauch bei der Portscanerkennung von Triggered Bitmap im Vergleich zu Snort und theoretischen Berechnungen untersucht.

In der Definition eines Portscans wurde hierbei die gleiche Schranke gesetzt wie Snort in seiner Defaulteinstellung benutzt: eine Quelle wird als Portscanner identifiziert, wenn sie 4 Verbindungen in einem 12 Sekunden Messintervall öffnet.

Die Triggered Bitmap wurde wie folgt konfiguriert: eine Direct Bitmap mit der Groesse von 4 Bytes und eine Multiresolution Bitmap mit 11 unterschiedlichen Auflösungen, jede mit der Groesse von 4 Byte (die letzte Aufösung war 8 Byte gross). Die Multiresolution Bitmap wurde nach einer Belegung von 8 Bit in der Direct Bitmap eingesetzt.

Die theoretischen Berechnung erfolgten auf der Grundlage eines ähnlichen Speicherverbrauches wie ihn die hier vorgestellte Multiresolution Bitmap hätte.

Der Speicherverbrauch durch Snort wurde ebenfalls theoretisch kalkuliert. Dazu wurde nicht das tatsächliche Modell von Snort verwendet, sondern die Menge der

Daten bei der Kallkulation reduziert. Snort verwenden ineffiziente Strukturen wie zum Beispiel verkettete Listen, in dem Testaufbau wurde nur von 8 Bytes für die Quell IP Adresse und einem Zähler und 9 Bytes für für die Ziel IP, Quell Port, Ziel Port und Typ ausgegangen.

Intervall	Snort	Berechnung	Triggered bitmap
12 Sec.	1968K	2474K	381K
600 Sec.	50791K	22876K	5725K

Abbildung 7: Speicherverbrauch Vergleich

Abbildung 7 zeigt die Ergebnisse des Tests zu Speicgerverbrauch. Es ist zu entnehmen, dass die Tiggered Bitmap 1/8 bis 1/4 des Speicherverbrauches einer Multiresolution Bitmap benötigen würde. Im Vergleich zu Snort verbraucht die Triggered Bitmap durchschnittlich 1/9 bis 1/6 des Speichers.

6 Zusammenfassung

Dieses Paper stellt eine Reihe von Bitmapalgorithmen vor, die das Problem des Zählen von aktiven Flows auf Hochgeschwindigkeitslinks adressieren, jedoch auch für andere Anwendungen durchaus geeignet wären. Durch die Benutzung von Bitmaps wird dabei die Schnelligkeit wie auch geringer Speicherverbrauch garantiert.

Direct Bitmap und Virtual Bitmap wurden im veröffentlichten Paper zum ersten mal im Kontext von Flowberechnungen analysiert, waren aber vorher schon durchaus bekannt. Multiresolution Bitmap wurde in dem referenzierten Paper [1] zum ersten mal eingeführt. Dieser Algorithmus, oder dessen Kombination mit den anderen, bieten eine solide und anpassungsfähige Grundlage für Programmierer sowie Hardwarehersteller, die in diversen Kontexten ihren Speicherbedarf reduzieren wollen.

Literatur

[1] Cristian Estan, George Varghese & Mike Fisk. Bitmap Algorithms for Counting Active Flows on High Speed Links, Paper: http://www.imconf.net/imc-2003/papers/p327-estan.ps , Presentation: http://www.cs.ucsd.edu/users/cestan/papers/FlowCountingBitmaps.ppt , Internet Measurement Conference 2003.

[2] Snort Intrusion Detection System
http://snort.org/ .

[3] David Plonka's FlowScan,
http://net.doit.wisc.edu/~plonka/FlowScan/ .

[4] Cisco - NetFlow,
http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html .

[5] CAIDA Analysis of Code-Red,
http://www.caida.org/analysis/security/code-red/ .

[6] CAIDA,
http://www.caida.org/home/ .

[7] MOAT,
http://moat.nlanr.net/ .

[8] Maximum-Likelihood-Methode,
http://de.wikipedia.org/wiki/Maximum-Likelihood-Methode .