Filtry Blooma

Janina Mincer-Daszkiewicz

Systemy rozproszone

MSUI, II rok

Materiały i rysunki zaczerpnięto z następujących źródeł

- Wikipedia, http://en.wikipedia.org/wiki/Bloom_filter
- Krzysztof Kotuła, seminarium SR 2009/10

  http://students.mimuw.edu.pl/SR/SR-MONO/filtry-blooma.pdf
- Michael Mitzenmacher, *Compressed Bloom Filters*, http://www.eecs.harvard.edu/~michaelm/NEWWORK/postscripts/cbf2.pdf
- Michael Mitzenmacher, *Network Applications of Bloom Filters: A Survey*, http://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf
- Robert Gilbert, Kerby Johnson, Shaomei Wu, Ben Y. Zhao, Haitao Zheng, *Location Independent Compact Routing for Wireless Network*, http://www.cs.ucsb.edu/~ravenben/publications/pdf/tarp-mobishare06.pdf
- T. Kruk, P.Tobiś, Zastosowanie filtrów Blooma do wyznaczania tras ataków sieciowych, http://www.cert.pl/PDF/secure2004/kruk-tobis.pdf

Wprowadzenie

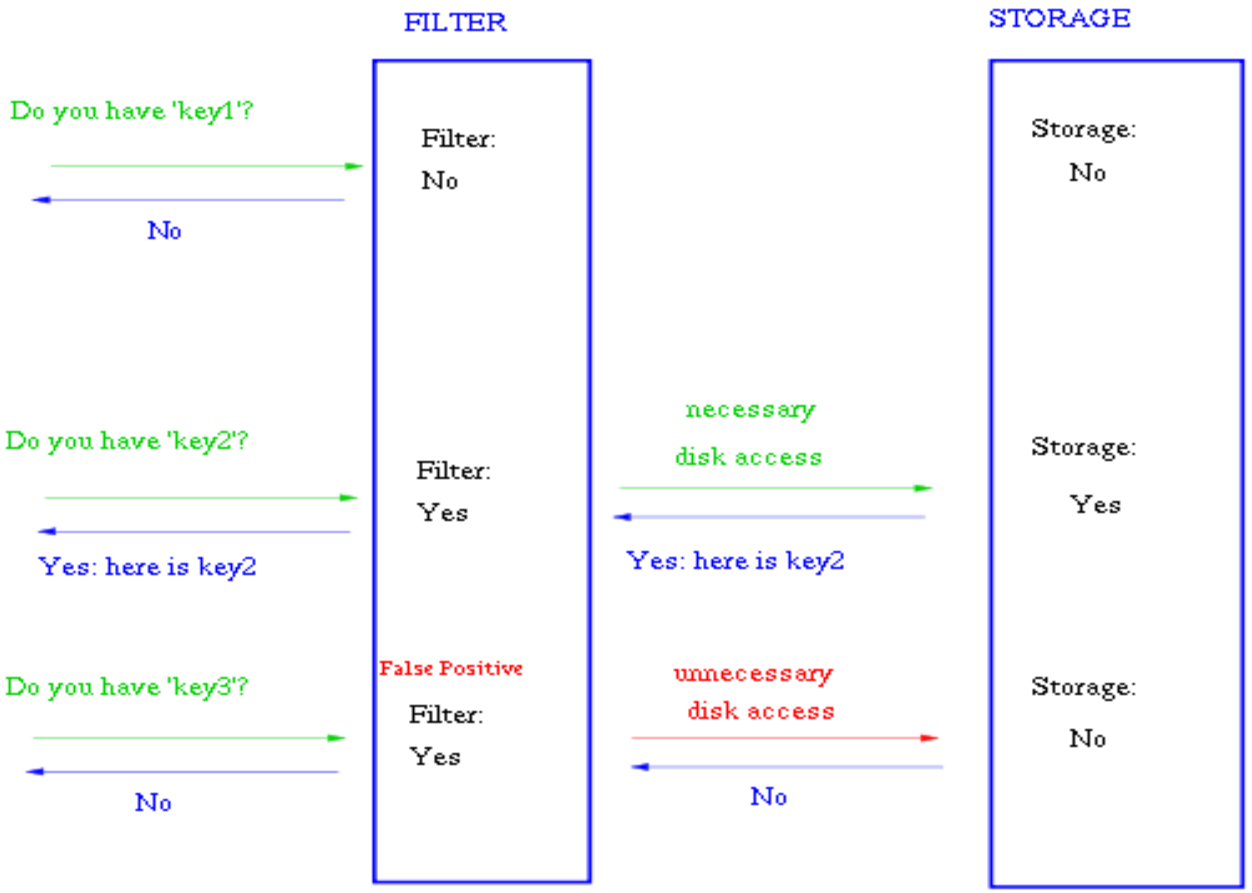
- Filtry Blooma to struktura danych wynaleziona przez Burtona Howarda Blooma w 1970 roku
- Potrafi odpowiedzieć na pytanie czy element należy do zbioru
- Jest probabilistyczna, czyli ... czasem się myli
- Za to jest wydajna pod względem zużycia pamięci
- Elementy można dodawać, nie można usuwać
- Im więcej jest elementów w zbiorze, tym większe prawdopodobieństwo błędu

Hm, to jaki z tego praktyczny pożytek?

Podstawowe cechy

- Dozwolone operacje:
  - Wstawienie elementu do zbioru
  - Zapytanie czy element należy do zbioru
  - W wersji podstawowej – nie ma usuwania
- Wykorzystuje losowe funkcje haszujące
- Jeśli algorytm odpowie na zapytanie **NIE**, to się **nie myli**
- Jeśli jednak odpowie **TAK**, to **może kłamać** (ang. *false positive*)
- W ogólności zatem korzystanie z filtru Blooma nie uwalnia nas od jawnego przechowywanie zbioru

Przykład zastosowania



Filtr Blooma pozwala na uniknięcie niepotrzebnych odczytów dyskowych, z wyjątkiem przypadków gdy kłamie (*false positive*)

Implementacja

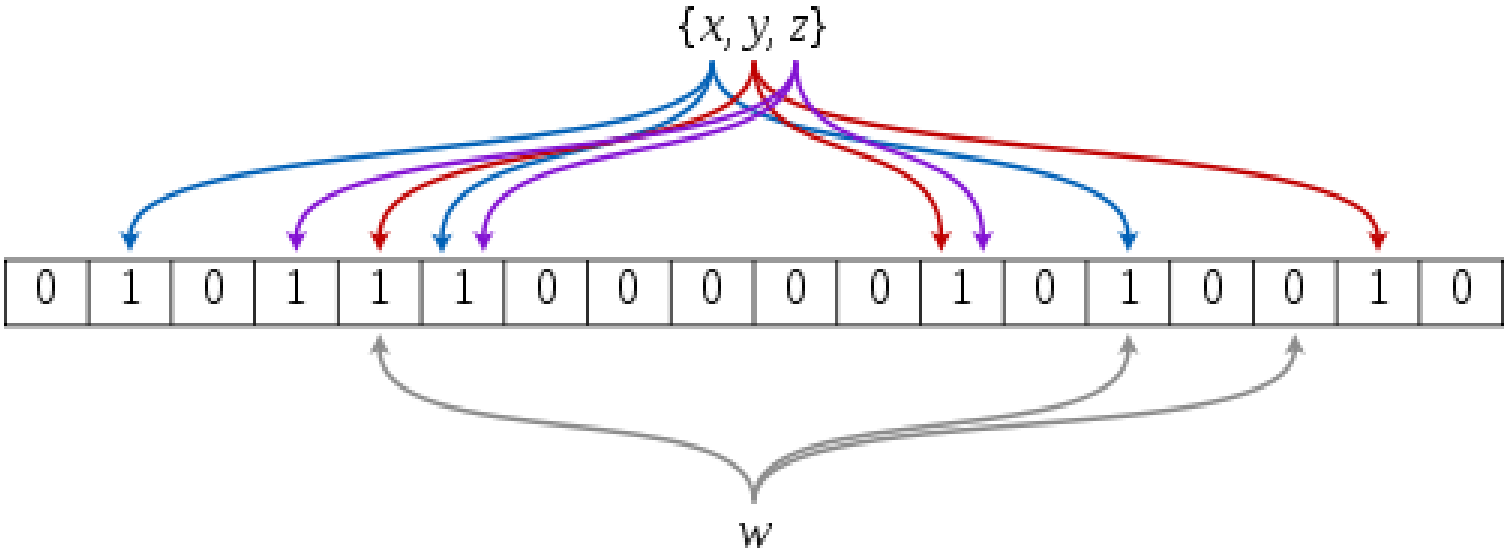
- Filtr Blooma to ***m*-bitowa tablica**
- Korzysta się z ***k* niezależnych, losowych funkcji haszujących** o wartościach z przedziału [0, *m*), które z równym prawdopodobieństwem odwzorowują elementy zbioru w pozycje w tablicy
- Parametry *m* i *k* oraz **jakość** funkcji haszujących determinują efektywność działania filtru Blooma
- Na początku wszystkie bity są zgaszone (zbiór jest pusty)
- W wersji podstawowej bitów nigdy nie gasimy i nie przejmujemy się tym, że mogą zostać wielokrotnie ustawione (tracimy zatem możliwość usuwania)

Operacje

- Wstawienie elementu do zbioru:
  - oblicza się wartości wszystkich *k* funkcji,
  - zapala wyznaczone *k* bitów
- Sprawdzenie czy element należy do zbioru
  - oblicza się wartości wszystkich *k* funkcji,
  - jeśli co najmniej jeden z wyznaczonych bitów jest zgaszony, to element **na pewno** nie należy do zbioru,
  - wpp – nie wiadomo, tzn. jest szansa, że element jest w zbiorze
- Gdy $k = 1$ mamy to czynienia z prostą tablicą haszującą

Dobry filtr Blooma będzie się mylić z małym prawdopodobieństwem

Przykład



Źródło: Wikipedia.org

- $k = 3$, $m = 18$
- Kolorowe strzałki wyznaczają wartości funkcji haszujących dla elementów x, y, z
- Elementu w nie ma w zbiorze, ponieważ jedna z funkcji wskazuje na zero w tablicy

Usuwanie

- Jeden bit może być zapalony wielokrotnie, przy wstawianiu różnych elementów
- A zatem usuwając jeden element nie można po prostu zgasić odpowiadających mu bitów
- Inaczej algorytm zacząłby kłamać mówiąc NIE, gubiąc tym samym jedną z kluczowych cech

Problem można próbować obejść ...

Usuwanie z drugim filtrem

- Mamy dwa filtry Blooma (niekoniecznie tego samego rozmiaru)
- Elementy wstawiamy do pierwszego filtru
- Usuwanie polega na wstawieniu obiektu do drugiego filtru
- Zapytanie sprawdza zatem, czy element został wstawiony oraz czy nie został usunięty
- Średnio wymaga dwukrotnie więcej pamięci
- Średnio dwukrotnie dłużej trwa zapytanie
- Drugi filtr jest także probabilistyczny, zatem może kłamać twierdząc, że element został usunięty

... i najważniejsze

usuniętego elementu nie daje się ponownie wstawić!

Usuwanie z licznikiem

- Pomysł: skoro jeden bit gubi informację o tym, ile razy został zapalony, to zastąpmy go licznikiem
- Zapalanie bitu to zwiększanie wartości o jeden, gaszenie to zmniejszanie o jeden
- Zapytanie polega na sprawdzeniu, czy wyznaczone *k* liczników ma wartości dodatnie
- Niestety dla każdej pozycji przeznaczona jest z góry określona liczba bitów, zatem grozi przepełnienie licznika – w efekcie znowu tracimy informacje ile razy licznik został zwiększony
- Stosując tę metodę trzeba znać oszacowanie maksymalnej liczby wstawień elementu

... i najważniejsze

tablica jest kilkakrotnie większa niż w wersji pierwotnej

Właściwości filtru

- Filtr zużywa stałą ilość pamięci
- Operacja wstawiania jest zawsze wykonalna (ale sukcesywnie rośnie prawdopodobieństwo błędu)
- Filtr potrafi reprezentować zbiór pełny (same 1)
- Koszt wstawienia i zapytania jest stały: O(k)
- Złożoność obliczeniowa zapytania nie zależy od m
- Nie potrzeba dodatkowej pamięci na żadne wskaźniki
- Obliczanie k niezależnych funkcji można realizować równolegle

Za niskie i stałe zużycie pamięci płacimy wzrostem prawdopodobieństwa błędu

Niestandardowe operacje

- Suma dwóch filtrów – OR dwóch wektorów bitowych
- Przecięcie dwóch filtrów – AND dwóch wektorów bitowych
- Dwukrotne zmniejszenie filtru – OR dwóch połówek wektora
- Powiększanie filtru i dopełnienie zbioru nie są już takie proste

Właściwości matematyczne

- Konstruując filtr mamy pole do manewru na dwóch płaszczyznach:
  - liczbie bitów w tablicy – za mało oznacza dużą szansę błędu, za dużo nadmierne zużycie pamięci
  - Liczbie funkcji haszujących – za mało oznacza dużą szansę błędu, za dużo też
- Niech:
  - m – rozmiar tablicy (liczba bitów w filtrze)
  - k – liczba funkcji haszujących
  - n – liczba wstawionych elementów

Cel: przy ustalonej liczbie n znaleźć wartości m i k minimalizujące prawdopodobieństwo błędu

Trochę matematyki – 1

- Zakładamy, że funkcja haszująca wybiera każdą wartość z równym prawdopodobieństwem
- P (dany bit zostanie zapalony przez daną funkcję) $= 1/m$
- P (dany bit nie zostanie zapalony przez daną funkcję) $= 1 - 1/m$
- P (dany bit nie zostanie zapalony przez żadną funkcję) $= (1 - 1/m)^k$
- P (dany bit nie zostanie zapalony przy wstawianiu n elementów) $= (1 - 1/m)^{kn} \approx e^{-kn/m}$
- P (dany bit zostanie zapalony) $= 1 - (1 - 1/m)^{kn}$

Trochę matematyki – 2

- Weźmy teraz element nie należący do zbioru
- Żeby **filtr skłamał**, wszystkie k bity wyliczone przez funkcje haszujące muszą być zapalone
- Szansa na to wynosi $P = (1 - (1 - 1/m)^{kn})^k$
- Stąd ostatecznie: $\boldsymbol{P \approx (1 - e^{-kn/m})^k}$
- Zgodnie z intuicją:

  szansa błędu rośnie wraz z liczbą wstawianych elementów

  i maleje wraz ze wzrostem rozmiaru tablicy bitowej

- Przy ustalonym n i m minimum tej funkcji wynosi

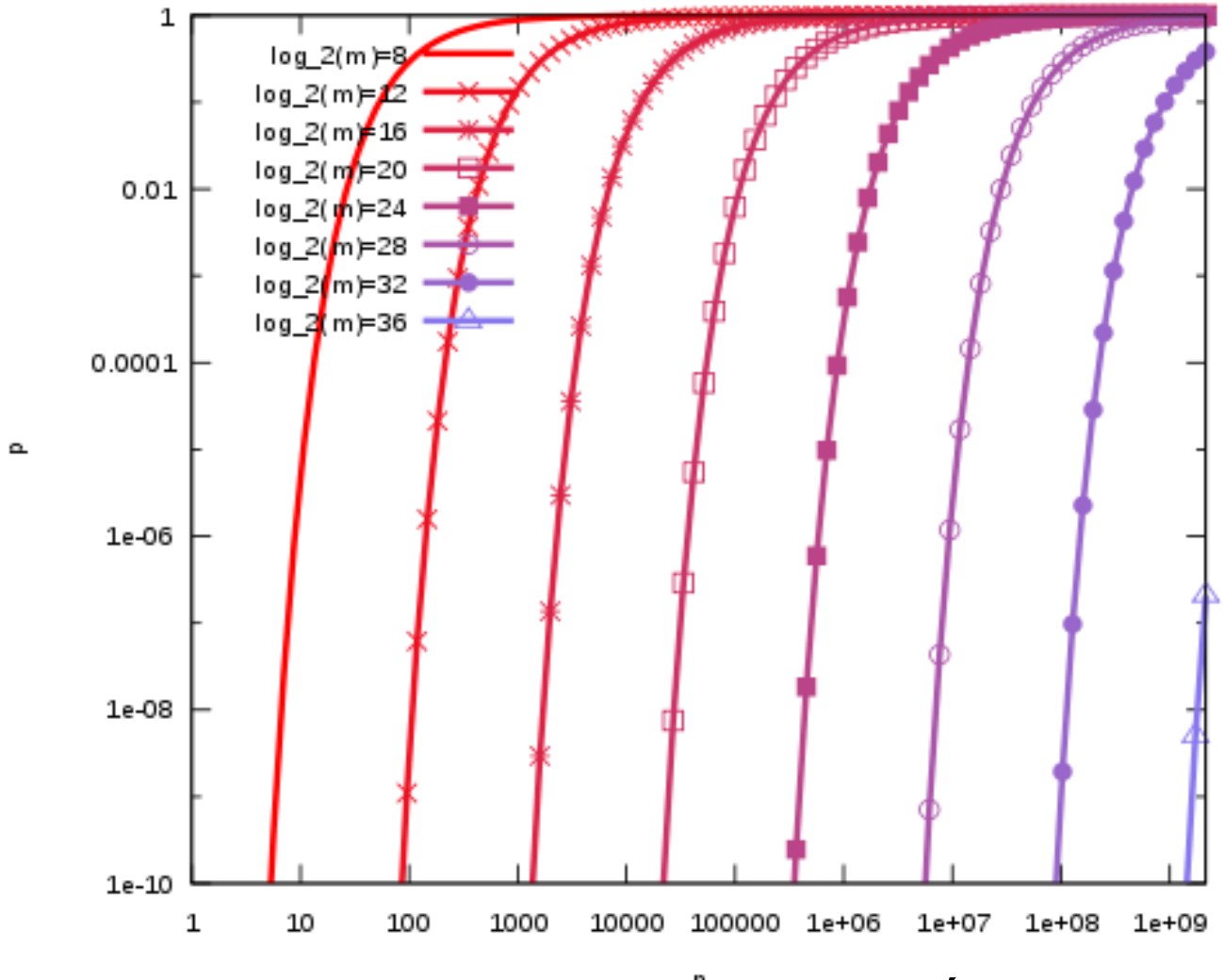
  $k = m/n \ln 2 \approx 0.693\, m/n$

Trochę matematyki – 3

- Szansa błędu wynosi wówczas $1/2^k \approx 0.6185^{m/n}$
- Dla danego n i P – pożądanego prawdopodobieństwa błędu, wymagana liczba bitów m wynosi:

  $\boldsymbol{m = -(n \ln P) / (\ln 2)^2}$
- ***Wniosek***: żeby utrzymać ustalone prawdopodobieństwo błędu (przy optymalnym k), m musi rosnąć liniowo zależnie od n (i być kilkukrotnie większe)
- $m = 8n \rightarrow P \approx 0.0214$
- $m = 12n \rightarrow P \approx 0.0031$
- $m = 16n \rightarrow P \approx 0.0005$
- Nie wolno przy tym zapominać, że k musi być całkowite

Trochę matematyki – podsumowanie



Błąd w zależności od n i m, k optymalne. Źródło: Wikipedia

Filtry Blooma z kompresją

- Jeśli filtr Blooma jest wysyłany siecią, to można by go dodatkowo skompresować
- W przedstawionej analizie braliśmy pod uwagę funkcje rozsiewające bity równomiernie
- Po wstawieniu dużego zbioru elementów mamy praktycznie losowy wektor 0-1 (biały szum)
- Taki ciąg danych niestety słabo się kompresuje
- Podejście pierwotne: zminimalizować prawdopodobieństwo błędu względem k przy ustalonym m i n
- Ponieważ dysponujemy kompresją – uwalniamy m na rzecz nowego czynnika z
- Teraz mamy szansę zyskać na kompresji, ale co na tym stracimy?

Filtry Blooma z kompresją - obliczenia

- n – liczba wstawianych elementów
- m – liczba bitów w tablicy nieskompresowanej
- k – liczba funkcji haszujących
- z – żądany rozmiar tablicy skompresowanej (w bitach)
- P – prawdopodobieństwo, że dany bit jest zgaszony (bity są niezależne)
- Zadanie: Dla ustalonej wartości n i z znaleźć wartości m i k minimalizujące prawdopodobieństwo błędu
- Posługując się wcześniej obliczonymi wartościami mamy:

  $z = m$, $k = m/n \ln 2$, zatem $P \leq 0.6185z/n$, gdy $P = 1/2$
- Naturalnie da się lepiej, można nawet pokazać, że $P = 1/2$ maksymalizuje błąd kompresowanego filtru!

Filtry Blooma z kompresją – wyniki 1

m/n	8	14	92
z/n	8	7.923	7.923
k	6	2	1
P	0.0216	0.0177	0.0108
$\approx$ 8 bitów na element			

m/n	16	28	48
z/n	16	15.846	15.829
k	11	4	3
P	0.000459	0.000314	0.000222
$\approx$ 16 bitów na element			

Źródło: *Compressed Bloom Filters*, Michael Mitzen

Filtry Blooma z kompresją – wyniki 2

m/n	8	12.6	46
z/n	8	7.582	6.891
k	6	2	1
P	0.0216	0.0216	0.0215
Współczynnik błędu ≈ 0.0216			

m/n	16	37.5	93
z/n	16	14.666	13.815
k	11	3	2
P	0.000459	0.000454	0.000453
Współczynnik błędu ≈ 0.00045			

Źródło: *Compressed Bloom Filters*, Michael Mitzen

Filtry Blooma z kompresją – wydajność obliczeniowa

- Otrzymana struktura po spakowaniu zajmuje tę samą ilość pamięci
- Niestety bieżące korzystanie z niej wymaga ciągłej kompresji/dekompresji
- Żeby uniknąć ogromnego narzutu czasowego stosujemy kompresję fragmentami
- Przy okazji niestety pogarszamy współczynnik kompresji

Zastosowania – dawniej 1

- Sprawdzanie pisowni (tj. reprezentacja słownika)
- Przechowywanie zbioru niepoprawnych haseł w UNIX-ie
- Operacja semi-join w rozproszonej bazie danych (przykład):
  - Baza A chce wysłać do bazy B listę miast, w których koszt życia wynosi > 50000
  - Baza B natomiast chce odesłać do A listę osób żyjących w takich miastach
  - Zamiast przesyłać listę miast z A do B można wysłać filtr Blooma
  - W zamian B odsyła listę potencjalnych odpowiedzi, z których należy odrzucić błędy

Zastosowania – dawniej 2

- Filtr dla dziennika transakcji bazy danych:
  - W bazie trzymany jest zapis zmian z danego dnia
  - Chcąc odczytać rekord sprawdzamy najpierw, czy został dzisiaj zmodyfikowany
  - Zamiast sięgać do dziennika można odpytać przygotowany filtr Blooma
  - Błąd spowoduje co najwyżej niepotrzebny odczyt z dziennika

Zastosowania – przykłady

- **Google BigTable** używa filtru Blooma żeby zredukować operacje dyskowe dla nieistniejących wierszy i kolumn
- **Venti** (sieciowy system pamięci, ang. *network storage system*) używa filtru Blooma do wykrywania przechowywanych elementów
- Używane przez **Squid** (serwer proxy i demon schowka) – o tym zastosowaniu za chwilę dokładniej

Zastosowania – schowek w serwerach proxy

- Serwery proxy buforują strony internetowe
- Jeśli pewien serwer chce odczytać stronę i nie ma jej we własnej pamięci, to odpytuje inne
- Normalnie serwery mogłyby wymieniać się listami adresów URL
- Używając filtrów Blooma zamiast takich list oszczędzają pasmo
- Serwer trzyma filtr Blooma dla każdego innego serwera
- Rozgłaszanie aktualnego filtru odbywa się co jakiś czas
- Ponieważ schowek zmienia się dynamicznie, często zdarzać się będą zapytania o wyrzuconą stronę
- W praktyce takich błędów będzie znacznie więcej niż tych
- wynikających z pomyłki filtru
- A zatem omylność filtrów Blooma prawie nie wpływa na wydajność serwerów

Zastosowania – lokalizacja serwerów w sieciach P2P

- Wykorzystanie filtrów Blooma jest w zasadzie identyczne, jak w serwerach proxy
- Dla dużych, zmiennych sieci zaleca się użycie rozproszonych tablic haszujących, które poprawiają skalowalność
- Błąd filtru znowu powoduje tylko zbędne zapytanie

Zastosowania – uzgadnianie zbiorów

- Węzeł *B* chce pobrać od węzła *A* elementy, których nie ma
- Jeśli *A* dysponuje zbiorem S_A, a *B* zbiorem S_B, to przesłany powinien zostać zbiór $S_A - S_B$
- Wystarczy, że *B* wyśle do *A* filtr Blooma reprezentujący S_B
- Kłamstwo filtru spowoduje niepełne uzgodnienie (część elementów nie zostanie wysłana)
- W sieciach P2P zbiory S_X mogą (i nawet powinny) składać się z kawałków plików

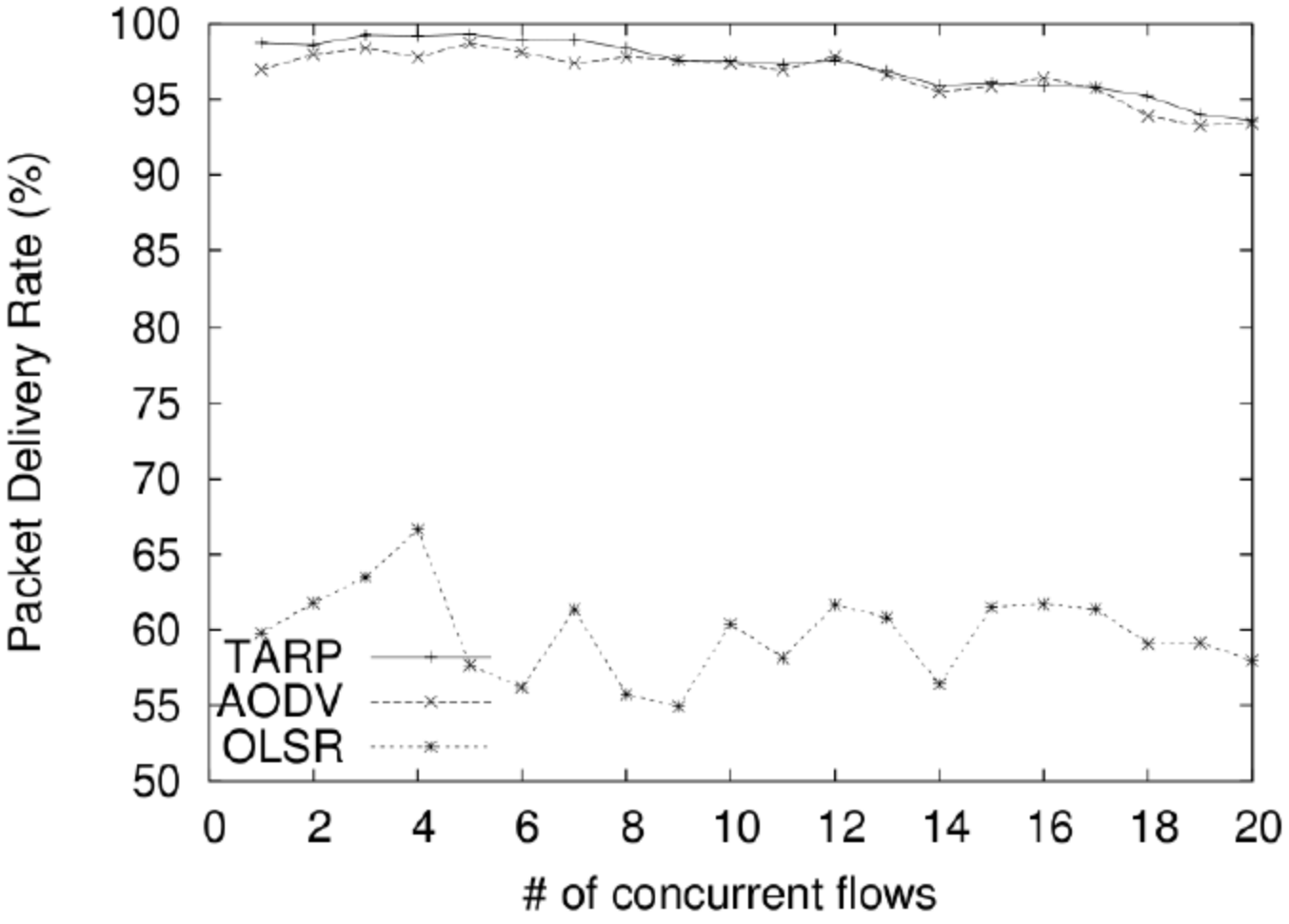
Zastosowania – rutowanie

- Chcemy móc odnajdywać obiekty, które znajdują się niedaleko w sieci (w sensie liczby węzłów pośredniczących)
- Dla każdego połączenia i odległości (do wyznaczonego limitu *d*) tworzymy filtr Blooma
- W ten sposób (z pewnym błędem) możemy docierać do danego obiektu po najkrótszej ścieżce
- Błąd filtru skutkuje sięgnięciem po zasób odleglejszy niż *d* kroków

Zastosowania – rutowanie testy notacja

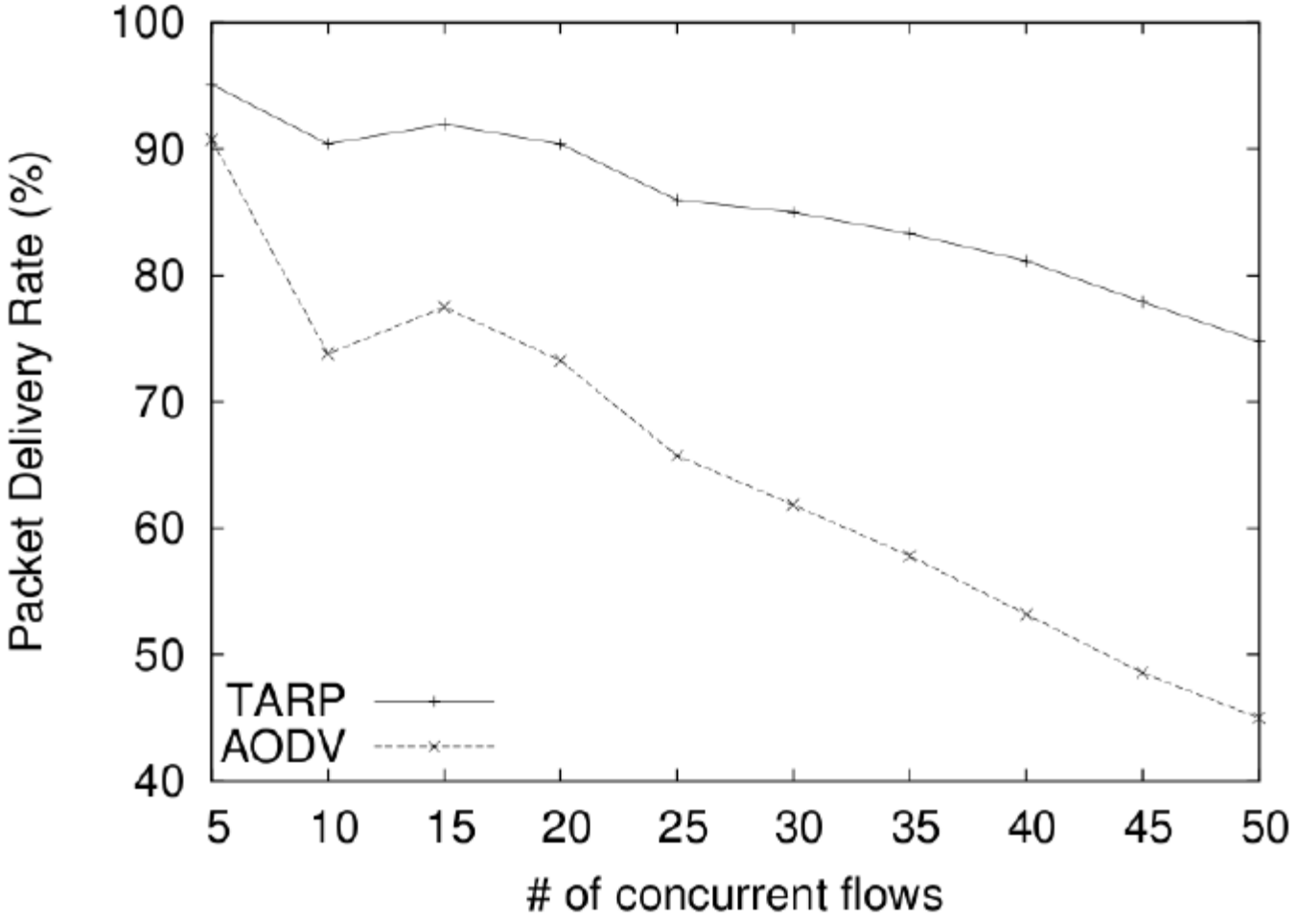
- Praca: *Location Independent Compact Routing for Wireless Networks*
- Badania wykonane na sieci wifi
- OLSR – protokół na bieżąco badający istniejące trasy
- AODV – protokół budujący i utrzymujący trasy na bieżące potrzeby
- TARP – protokół posługujący się filtrami Blooma mniej-więcej jak opisano

Zastosowania – rutowanie wyniki 1



100 węzłów w sieci

Zastosowania – rutowanie wyniki 2



500 węzłów w sieci

Zastosowanie – śledzenie ataków w sieci

- Ataki sieciowe są na porządku dziennym, bo:
  - Dostęp do internetu jest powszechny
  - Duże natężenie ruch w sieci utrudnia znalezienie hackera
  - *Spoofing* – możliwość fałszowania adresu źródłowego pakietu
- Popularne metody ataku:
  - *Flooding* – zapychanie ofiary masową liczbą pakietów, pośrednio (poprzez komputery zombie) lub bezpośrednio
  - *Skanowanie* – np. program nmap (narzędzie do eksploracji sieci i audytów bezpieczeństwa)
  - Wykorzystywanie błędów w systemach zdalnych np. przez przepełnienie bufora

Zastosowanie – ataki w sieci – ochrona

- Metody ochrony przed *floodingiem* i *skanowaniem* sprawdzają się w obrębie sieci lokalnych
- Obrona przed atakiem z zewnątrz wymaga znajomości źródła
- Szczególnie trudne do zidentyfikowania są ataki krótkie
- Rejestrowanie całego ruchu w sieci jest niewykonalne:
  przy łączu 1 Gbps w trakcie 1 min pełnego ruchu
  rejestrowanie 20-bajtowych nagłówków IP
  wymagałoby prawie 5 GB pamięci...

... tu wkraczają filtry Blooma

Zastosowanie – ataki w sieci – ochrona

- Cel: zbudować system monitorujący ruch w poszczególnych węzłach sieci, ograniczyć prawdopodobieństwo błędów
- Liczba filtrów:
  - Co najmniej jeden na każdy interfejs sieciowy urządzenia
  - Za mało – informacje będą szybko nadpisywane
  - Za dużo – wyszukiwanie źródła będzie trwało zbyt długo, a prawdopodobieństwo błędu się powiększy
  - Każdy filtr rejestruje jedynie ruch wychodzący lub przychodzący – pozwoli to odtworzyć trasę pakietu
  - Ewentualnie dodatkowy filtr do śledzenia pakietów generowanych przez urządzenie (szczególnie, gdy jest *końcówką* sieci)

Zastosowanie – ataki w sieci – ochrona
Wyznaczanie trasy pakietu

- Dany węzeł otrzymuje zapytanie o konkretny pakiet:
  - jego początkowe X bajtów (wielkość powinna być z góry ustalona),
  - unikalny identyfikator (np. numer),
  - szacowany odstęp czasu, jaki upłynął od zarejestrowania pakietu u sąsiada
- ... po czym odpytuje swoje filtry:
  - jeśli nie znalazł pakietu, to kończy swoje zadanie
  - jeśli znalazł, to odpowiada i przekazuje pytanie do odpowiednich sąsiadów

Zastosowanie – ataki w sieci – ochrona
Wyznaczanie trasy pakietu cd

- Ze względu na omylność filtrów jedno urządzenie może dostać dwukrotnie zapytanie o ten sam pakiet
- Przed takim zapętleniem chroni numerowanie zapytań
- Dla bezpieczeństwa warto stosować szyfrowanie i uwierzytelnianie
- Należy też pamiętać, że pewne pola nagłówka IP ulegają zmianie podczas podróży przy zapytaniu trzeba je ignorować
- Aby pokonać NAT, ruter będący jego wyjściem musi umieć prawidłowo przekazać zapytanie do podsieci

Zastosowanie – ataki w sieci – ochrona
Testy

- Filtry Blooma nasycono 270000 pakietami z sieci lokalnej klasy C
- Zbadano również różne algorytmy liczenia skrótów jako bazy do tworzenia funkcji haszujących
- *Wnioski:*
  - żeby ruter mógł wytrzymać obciążenie obliczeniowe, wymagane są specjalizowane układy sprzętowe
  - jak również dużo (szybkiej) pamięci dyskowej
  - ...a to kosztuje ☹

Zastosowanie – ataki w sieci – ochrona
Testy – wyniki

		Sieć dostępowa	Ethernet 10BaseT	Fast Ethernet	Gigabit Ethernet
Przepustowość	[Mbps]	2	10	100	1000
	[kpps]	5,4	27	272	2717
Zasoby dla całych ramek		18 MB	89 MB	889 MB	8,9 GB
Zasoby dla filtrów		< 1MB	7,5 MB	60 MB	960 MB
Zysk [%]		95	92	93	89
Parametry filtrów	*m*	2^16	2^19	2^22	2^26
	k	8	13	11	17
	p	3,6e-3	9,4e-5	6,0e-4	7,0e-6

Wykorzystane zasoby przy 1 minucie rejestrowania
40 B nagłówków IP