CoralCDN

Aleksander Jurkowski

Uniwersytet Warszawski

aj262944@students.mimuw.edu.pl

March 2, 2011

Co to jest CoralCDN

CDN

Content Delivery Network - system rozproszony mający na celu dostarczanie określonych zasobów do użytkowników

CoralCDN

CoralCDN jest rozwiązaniem służącym do udostępniania treści internetowych dostępnych dla dowolnej liczby użytkowników, niezależnie od posiadanych własnych zasobów serwerowych.

Założenia CoralCDN

- Wolny dostęp - każdy może używać CoralCDN
- Nie ma potrzeby rejestracji użytkowników
- Brak autoryzacji
- Nie wymaga skomplikowanej konfiguracji by go użyć

CoralCDN - [podstawowe] użycie

Założenia

Mamy stronę internetową na serwerze, np example.com wyświetlającą treści i chcemy skorzystać z CoralCDN by uodpornić się na różne zagrożenia (o tym później).

Konfiguracja

Dokonujemy zamiany example.com -> example.com.nyud.net

I ... działa

W szczególności możemy przekierować w ten sposób np tylko najbardziej obciążającą nas część strony.

Pojekt CoralCDN

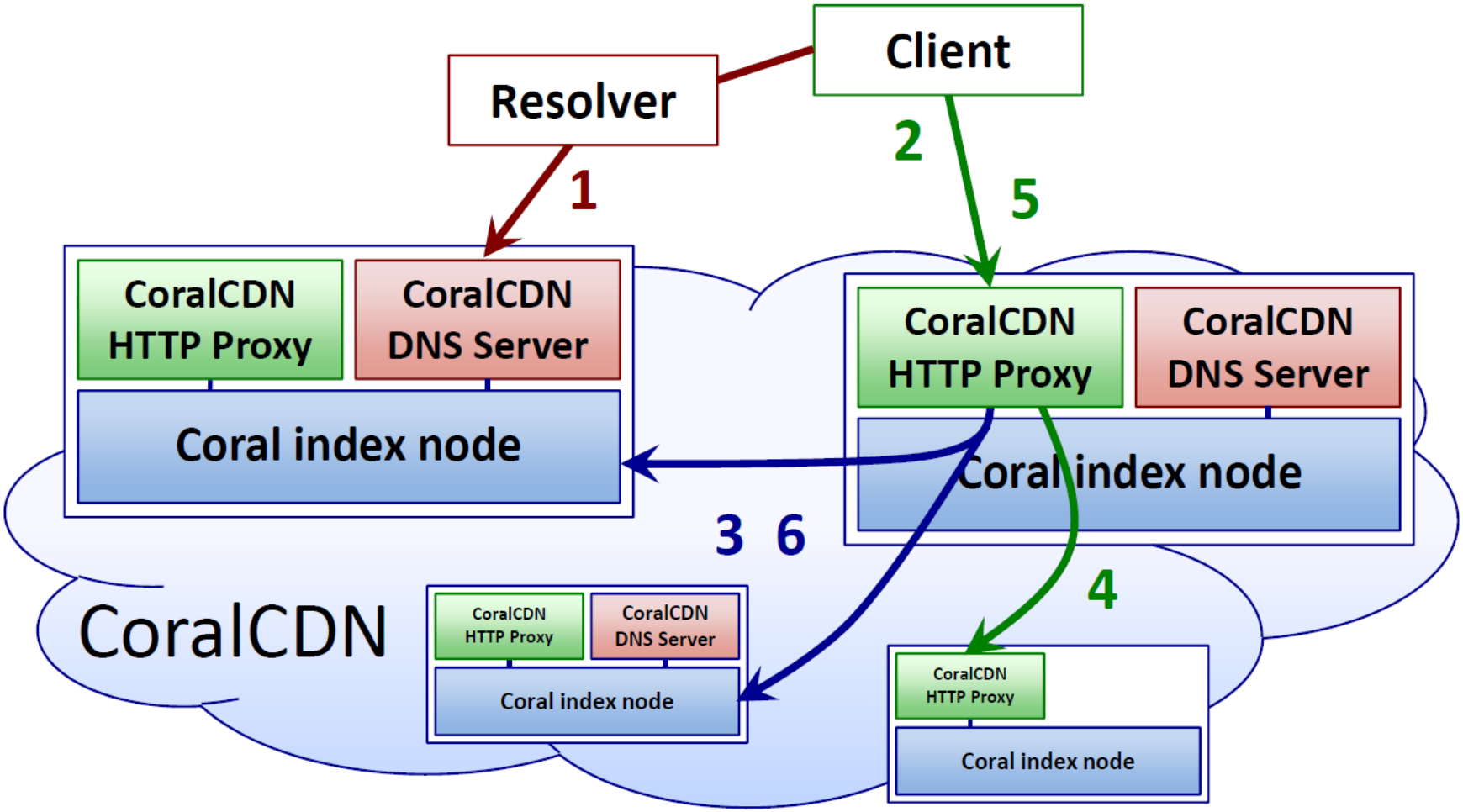
Najważniejsze części CoralCDN

- HTTP proxy - pojedyncze węzły, obsługujące zapytania użytkowników
- Serwery DNS CoralCDN zarządzające domeną `nuyd.net`
- Infrastruktura indeksująca i klastrowanie węzłów

Terminologia

Warstwa indeksująca nazywana jest *Coral* a cały system *CoralCDN*.

Schemat działania



Schemat działania

1. Rozwiązywanie DNS - nazwa z sufiksem (example.com.nyud.net) CoralCDN jest rozwiązywana przez klienta przy użyciu serwerów CoralCDN. Serwer CoralCDN sprawdza czas latencję klienta i na tej podstawie przyporządkowuje odpowiednie serwery DNS i HTTP proxy.
2. Przetwarzanie zapytań HTTP - klient wysyła zapytanie HTTP do jednego ze zwróconych HTTP proxy. W przypadku gdy proxy ma podaną strone w swoim cache to jest ona wysyłana w odpowiedzi klientowi, w przeciwnym przypadku proxy próbuje znaleźć inne mające dane treści.
3. Wyszukiwanie treści w cache innych proxy - HTTP proxy wyszukuje URL w warstwie indeksującej

Schemat działania

4. Jeśli proxy znajdzie węzeł zawierający dany URL, to pobiera go z niego, w przeciwnym przypadku pobiera z oryginalnego serwera (example.com)
5. HTTP proxy zapisuje na swoim dysku stronę i wysyła ją użytkownikowi
6. HTTP proxy rejestruje referencję do siebie w warstwie indeksującej, informującą, że od teraz ma cache z URL

Warstwa indeksująca

Teoria

Struktura indeksująca *Coral* jest blisko związana z rozproszonymi tablicami hashującymi (DHT), takimi jak Chord czy Kademlia.

Przestrzeń identyfikatorów

Klucze są kodowane do przestrzeni identyfikatorów (zupełnie abstrakcyjnych), identyfikatory węzłów są umieszczone w tej samej przestrzeni. W ten sposób dostajemy czas wyszukiwania na poziomie $O(log(n))$, dla systemu o n węzłach

Kademlia

Ogólnie

Kademlia jest przykładem DHT, w którym wyszukiwanie zajmuje $O(log(n))$, w systemach o n węzłach.

Identyfikatory

- identyfikatory są związane z przechowywanymi wartościami (jest bezpośrednie mapowanie plik => id np przez hash)
- węzeł o danym ID posiada informacje skąd można pobrać zasoby o danym hashu

Kademlia

Odległość

- w Kademlia odległość to różnica ID a nie odległość fizyczna
- odległość dwóch ID to ich alternatywa wykluczająca
- klucze i identyfikatory mają ten sam format - można tak samo mierzyć odległość dla węzłów i zasobów
- identyfikatory węzłów to duże liczby losowe, unikalne (jak GUID) - sąsiedzi w ID nie muszą być blisko

Kluczowe cechy "odległości":

- Identyczność nierozróżnialnych: $\forall A \mid d(A, A) == 0$
- Symetria: $\forall A.B \mid d(A, B) == d(B, A)$
- Nierówność trójkąta: $\forall A, B, C \mid d(A, B) + d(B, C) \geq d(A, C)$

Kademlia - uproszczony opis

- załóżmy, że klucz jest liczbą 128-bitową
- każdy węzeł przechowuje długość klucza list, w tym przypadku 128
- każdy wpis na liście to dane niezbędne do zlokalizowania węzła, typowo <IP, port, ID>
- listy są zależne od "odległości" od danego węzła, dokładnie do listy numer k trafiają węzły mające te same bity $1..k-1$, ale różne na pozycji k

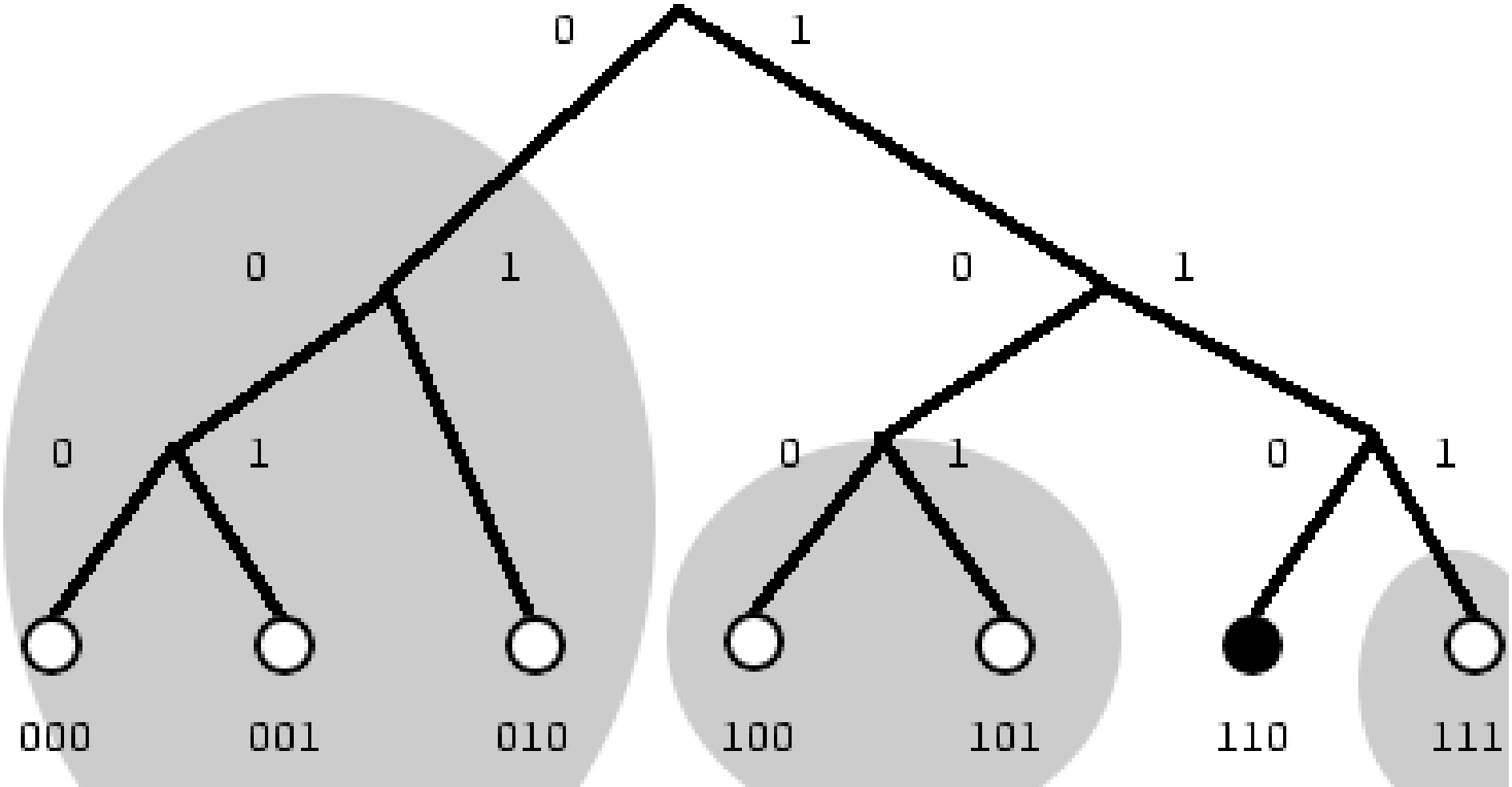
Mamy:

- Do pierwszej listy trafi $\frac{1}{2}$ węzłów - połowa węzłów ma ten sam pierwszy bit
- Do drugiej listy trafi $\frac{1}{4}$ węzłów
- ...

Kademlia - uproszczony opis

- wpisy do list "sąsiadów" są dodawane w miarę ich poznawania w sieci => ich zawartość zmienia sie dynamicznie wraz z siecią
- listy są ograniczone z góry do długości k, co oznacza, że początkowe listy mają pełną reprezentację, a dalsze po k wpisów o "sąsiadach"
- w literaturze jest spotykane określenie *k-kubełki* na owe listy

Kademlia



Kademlia

Fakt

Węzeł, który jest podłączony od dłuższego czasu ma duże prawdopodobieństwo pozostać jeszcze długo połączony.

Wracamy do list

- poznajemy nowy węzeł
- jeśli odpowiadający mu k-kubełek jest pełny, to:
  - węzeł, z którym ostatni kontakt był najwcześniej jest sprawdzany
  - jeśli nie odpowiada, to:
    - stary węzeł jest usuwany
    - nowy trafia na jego miejsce
  - jeśli stary węzeł odpowiada, to
    - nowy wpis trafia na listę dodatkową, z której skorzystamy w przypadku, gdy okaże się, że któryś z węzłów na liście nie odpowiada.
- jeśli k-kubełek nie jest pełny to dodajemy nowy wpis

Kademlia - znajdowanie węzłów

1. węzeł chce znaleźć inny o znanym ID
2. wybiera ze swoich k-kubełków k węzłów o ID najbliższych poszukiwanemu
3. wysyła zapytanie do tych k węzłów o listę węzłów o ID bliższych szukanemu
4. gdy dostaje odpowiedź, to aktualizuje zbiór k najbliższych szukanego klucza
5. goto 3

STOP: jeśli w otrzymane listy nie poprawiają tej już posiadanej
Gdy ten proces się kończy to mamy k najbliższych węzłów szukanemu.

Kademlia

Znajdowanie zasobów

- liczymy hash od danego zasobu
- szukamy tego klucza używając poprzedniego algorytmu, kończąc gdy tylko znajdziemy węzeł, który przechowuje ten zasób

Redundancja

Żeby nie tracić danych potrzebne jest trzymanie ich w wielu kopiach

- węzeł co jakiś czas kontaktuje się z k-najbliższymi sąsiadami i wysyła im swoje zasoby
- zasoby, które są bardzo często pobierane są kopiowane również poza k-najbliższych sąsiadów, co jest określane jako *cache*
- ilość i TTL zapisów *cache* zależy od popularności zasobu

Kademlia - przyłączanie się do sieci

- żeby przyłączyć się do sieci trzeba znać jakiś węzeł, który aktualnie w niej jest
- trzeba posiadać unikalny id (można go wygenerować) - ma być stały przez cały pobyt w sieci

Procedura:

1. węzeł wyszukuje siebie u węzła z sieci, o który zna
2. w ten sposób poznaje sąsiadów po drodze od znanego węzła do siebie i wypełnia początkowe k-kubełki
3. nowo poznane węzły są pytane o odpowiednio odległe losowe liczby, co pozwala wypełnić dalsze k-kubełki

Warstwa indeksująca

Specyficzne DHT

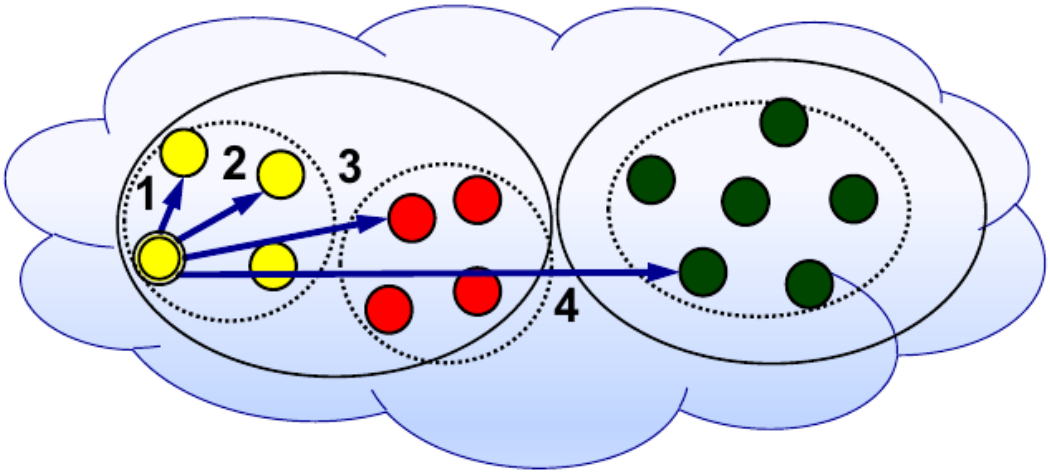
Ze względu na cele CoralCDN nie są potrzebne tak silne więzy spójności jak w tradycyjnym użyciu - wpis w cache może nie zostać znaleziony nawet jeśli istnieje jakiś węzeł go przechowuje.

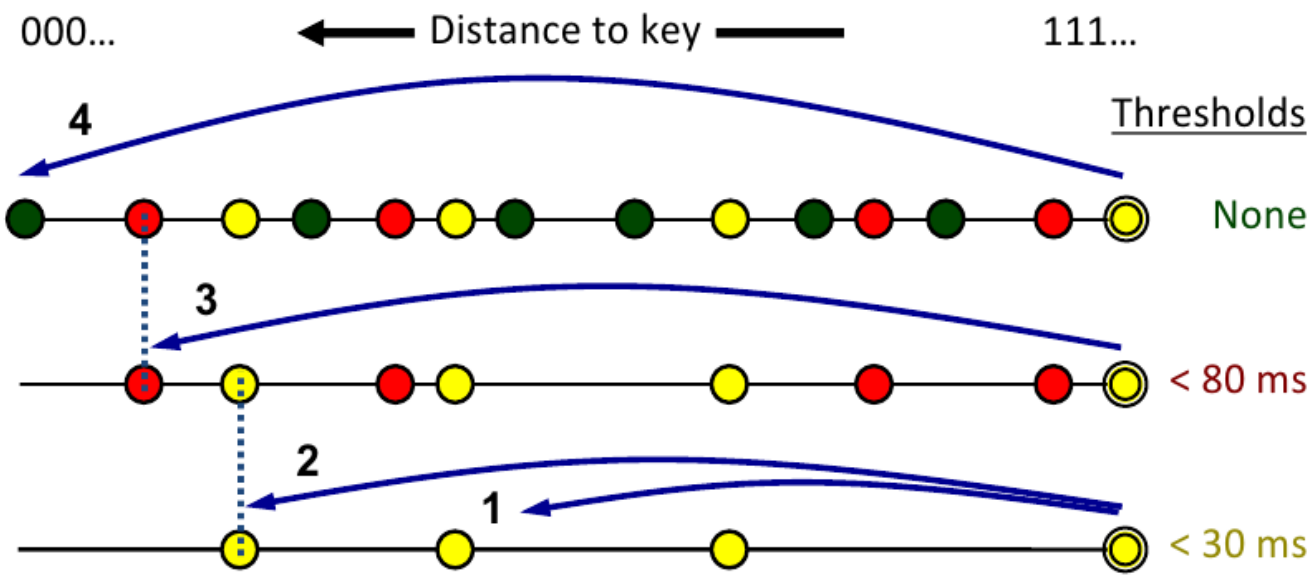
Semantyka

Operacje:

- *get* - znajduje HTTP proxy zawierające wpisy w cache o podanym URL
- *put* - rejestruje wpis cache o URL, dodatkowo zawiera czas życia (TTL), po którym staje się nieważny

Warstwa indeksująca - klastry





Warstwa indeksująca - klastry

Klastry

W celu zwiększenia lokalności wyszukiwania z danego węzła sieć została podzielona na klastry:

- poziom 2 - węzły o RTT < 30ms
- poziom 1 - węzły o RTT < 80ms
- poziom 0 - globalnie

Wyszukiwanie działa tak jak w Kademlia, z tym, że najpierw jest wykonywane tylko na poziomie 2, potem poziomie 1 i gdy się nie uda to na poziomie 0.

HTTP proxy

Flash crowd

Nagły wzrost zapotrzebowania do pewnej strony. Przykładem tego może być nagłe zainteresowanie określonym URL po umieszczeniu linku do strony na popularnym portalu.

Flash crowd w CoralCDN

Kiedy zapotrzebowanie na jakieś źródła nagle wzrasta, to HTTP proxy w CoralCDN układają się w drzewo, propagując treści od siebie. To pozwala na zminimalizowanie ilości zapytań do oryginalnego serwera (i chroni przed jego przeciążeniem).

HTTP proxy

Jak to działa?

By zapewnić taki efekt CoralCDN stosuje 2 techniki:

- natychmiastowa propagacja - kiedy HTTP proxy pobiera większy obiekt, to zaczyna wysyłać jego części natychmiast (nie czeka na pobranie całości)
- optymistyczne referencje - HTTP proxy umieszcza referencję do siebie z krótkim TTL (30s) w warstwie indeksującej gdy tylko posiada część danych. W miarę przychodzenia następnych części wpisy są aktualizowane, a gdy posiada całość, to zwiększa TTL swojej referencji.

Wdrożenie CoralCDN

PlanetLab

PlanetLab jest eksperymentalną siecią, składającą się głównie z serwerów akademickich mającą na celu testowanie projektów rozproszonych. PlanetLab jest darmowe, ale by móc z niego skorzystać konieczne jest dostępu od instytucji będącej w konsorcjum PlanetLab. Aby być w konsorcjum PlanetLab instytucje muszą spełnić szereg wymagań, z czego najważniejsze jest udostępnienie serwera (2 węzłów) o odpowiedniej mocy.

Wdrożenie CoralCDN

CoralCDN został wdrożony na serwerach PlanetLab, typowo działa na 300-400 serwerów, z czego 70-100 to serwery DNS. CoralCDN jest w pełni systemem rozproszonym - mimo że PlanetLab daje możliwości scentralizowanego działania, CoralCDN korzysta z tego tylko do zarządzania oprogramowaniem i jego aktualizacjami.

Użycie

Sposoby używania CoralCDN

W praktyce zaobserwowano 4 główne cele używania CoralCDN:

1. Odzyskiwanie starych materiałów - niektórzy klienci korzystają z CoralCDN by uzyskać dostęp do stron internetowych, które już nie są dostępne. Istnieją nawet wtyczki do przeglądarek, mające na celu wskazywanie nieaktywnych stron i jako jedno ze źródeł używają one właśnie CoralCDN
2. Udostępnianie mało popularnych zasobów - ogromna liczba URLi odnosi się do niepopularnych stron - podawane są linki do CoralCDN nawet dla mało powszechnych materiałów. Ponadto istnieją klienci, używający CoralCDN jak "tradycyjnego" proxy.

Użycie

Sposoby używania CoralCDN

1. Udostępnianie popularnych zasobów w długim czasie
2. Uodparnianie się na nagłe wzrosty popularności określonych zasobów - powszechną praktyką jest np umieszczanie “coralized” linków na popularnych portalach zamiast bezpośrednich odnośników

Praktyka

Jak to się sprawdza ?

Okazuje się, że CoralCDN nie jest najlepszym rozwiązaniem dla pierwszych trzech z wymienionych celów - zgodnie z zamysłem projektu, sprawdza się najlepiej przy tzw “flash crowds”

Udostępnianie nieaktualnych stron

HTTP proxy z założenia nie propagują zapamiętanych treści w celu odostępniania starych stron. Zwykle TTL jest ustawiony na 12h, HTTP proxy będzie udostępniać stare zasoby maksymalnie 24h po ich zdeaktualizowaniu.

Praktyka

Mało popularne treści

Dla niepopularnych treści użycie CoralCDN mija się z celem:

- Zmniejszenie obciążenia serwera jest pomijalne
- Zwiększenie opóźnienia połączeń
- "Tradycyjne" otwarte HTTP proxy o wiele lepiej sprawdza się w roli proxy

Praktyka

Popularne treści w długim okresie

CoralCDN jest o wiele zbyt skomplikowanym rozwiązaniem dla tego problemu.
Fakty:

1. 0.01% najpopularniejszych URL wykorzystuje 49% całego ruchu w CoralCDN
2. 14MB wystarczy żeby przechować 0.01% najpopularniejszych URLi
3. każdy węzeł ma do dyspozycji $3GB$ pamięci - wystarczy na 85% zapytań
4. 70% zapytań do HTTP proxy jest obsługiwane przez lokalny cache
5. obsługa 7% zapytań wykorzystuje komunikacje między proxy

Dla najpopularniejszych URL komunikacja między proxy nie jest potrzebna.

Akamai

Akamai to komercyjny CDN, jego celem jest właśnie dostarczanie treści długookresowo.

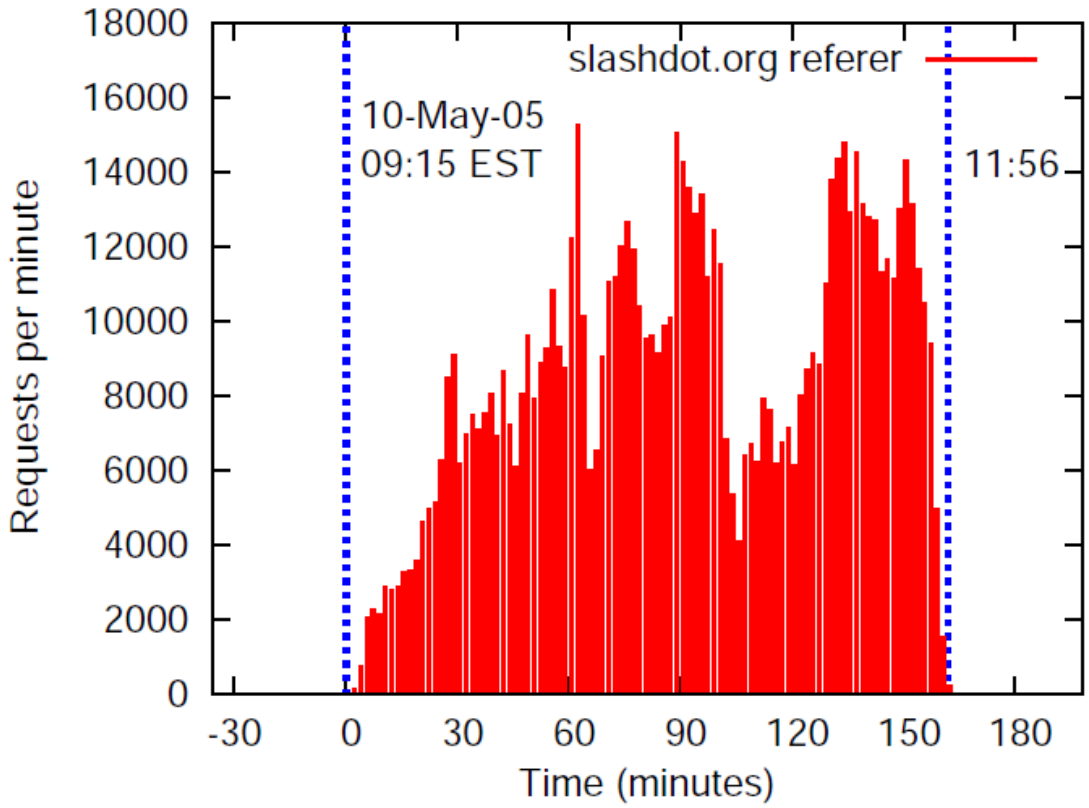
- obsługuje komercyjne serwisy - znacznie większe zasoby serwerowe oryginalnych serwerów
- węzły w Akamai współpracują tylko w ramach klastrów po 1000
- każdy z klastrów niezależnie odwołuje się do oryginalnego serwera

To rozwiązanie okazuje się w zupełności wystarczające do tego celu.

Flash crowds

Obserwacja

Flash crowd formuje się w ciągu minut a nie sekund - system ma więc sporo czasu na dostosowanie się do nowych warunków.



Flash crowds - wnioski

1. flash crowds występują tylko dla bardzo niewielkiej ilości URL
2. ruch sieciowy domen, dla których flash crowds występują jest niewielkim odsetkiem całości ruchu
3. na przestrzeni sekund bardzo rzadko zdarzają się wzrosty popularności

CoralCDN

Dla obsługi flash crowds aktualna forma komunikacji jest w zupełności wystarczająca

Praktyka - wyścig

Problem

Przy wysokiej współbieżności CoralCDN może odwołać się niepotrzebnie wiele razy do źródłowego serwera.

Źródło problemu

Scenariusz:

1. wiele węzłów równocześnie wykonuje *get* na tym samym kluczu
2. poszukiwany klucz nie występuje w warstwie indeksującej
3. wszystkie wyszukiwania w indeksie mogą się nie udać
4. wtedy wszystkie te węzły rozpoczną pobieranie danych ze źródła

Praktyka - wyścig

Pocieszenie

Praktycznie wszystkie systemy DHT mają ten problem.

Proste rozwiązanie

Nie przejmujemy się tym problemem, bo jak się okazało ilości zapytań przy flash crowds nie narastają w ciągu sekund tylko minut.

Wyścig - próba rozwiązania

Nowe rozwiązanie

put + *get* umieszcza klucz w indeksie oraz zwraca pierwsze wyniki dla powiązanego klucza.

- klucze teraz są umieszczane jeszcze przed rozpoczęciem pobierania ze źródła
- jeśli pobieranie się nie uda, to ten wpis w indeksie zmniejsza wydajność systemu

Wniosek

To rozwiąznie było testowane przez kilka miesięcy w 2005 roku i nie jest kontynuowane.

Publiczne API

Z punktu widzenia użytkownika

API CoralCDN dla administratora strony zapewnia:

- przezroczystość - działa z niezmodyfikowanymi, nieprzekonfigurowywanymi, nieświadomymi obecności CoralCDN klientami
- głębokie cachowanie - zawarte obrazki itp są uzyskiwane przez CoralCDN
- pełna kontrola serwera - nie odbiera możliwości logowania czy kontroli jak przekazywane są treści
- nie przeszkadza reklamom - serwisy reklamowe, czy analizy użycia działają prawidłowo przez CoralCDN

Publiczne API - interface DNS

Alternatywa

Nie używać sufiksu .nyud.net tylko zamieniać example.com => nyud.net/example.com/

- upraszcza parsowanie
- nie potrzeba podsystemu DNS

ale...

- Forum

Publiczne API - interface DNS

- Można zamienić wszystkie linki programowo

ale...

- treści generowane przez javascript - rozwiązanie się komplikuje
- serwisy reklamowe mogą mieć z tym problemy

Publiczne API - integracja z serwerem

Strategia

Serwer example.com odpowiada klientom “302 redirect” do example.com.nyud.net

Koszt odpowiedzi “redirect” jest bardzo mały w porównaniu do faktycznego wysłania całej strony

Apache

```
RewriteEngine on
RewriteCond %{HTTP_USER_AGENT} !^CoralWebPrx
RewriteCond %{QUERY_STRING} !(^|&)coral-no-serve$
RewriteRule ^(.*)$ http://%{HTTP_HOST}.nyud.net
                                %{REQUEST_URI} [R,L]
```

Publiczne API

Przekierowanie tylko z określonych źródeł

```
RewriteCond %{HTTP_REFERER} slashdot\.org [NC,OR]
RewriteCond %{HTTP_REFERER} digg\.com [NC,OR]
RewriteCond %{HTTP_REFERER} blogspot\.com [NC]
```

Inne ciekawe użycie

flood.com.nuyd.net.nyud.(...).nyud.net - zapytań jest tyle ile sufiksów

Publiczne API

Dynamiczne przekierowywanie

Popularnym rozwiązaniem jest przekierowywanie do CoralCDN tylko przy dużych obciążeniach.

Dodatkowe funkcje

Na wniosek użytkowników wzbogacono API - np nagłówek `X-Coral-Control` pozwala określić czy serwisy, które przekroczyły limit sieci w CoralCDN mają być odsyłane bezpośrednio do serwera czy mają zwracać błąd.

Bezpieczeństwo

Limitowane funkcjonalności:

- wspierane są tylko zapytania GET i HEAD (nie POST czy CONNECT)
- zapytania CONNECT mogłyby potencjalnie wysyłać spam przez SMTP
- brak wsparcia dla HTTPS
- HTTP proxy kasują wszystkie ciasteczka z nagłówków
- miał miejsce 1 atak na jedną z 5 najpopularniejszych stron, która przesyłała hasła plain-text przez GET

Atak brutalny przez CoralCDN

- dla każdego zapytania musi nie udać się wyszukiwanie DHT
- opóźnienie z DHT działa tutaj jak bariera przed zbyt częstym łączeniem z oryginalnym serwerem
- CoralCDN nie daje pełnej anonimowości - ustawia nagłówki Via i X-Forwarded-For

Limit użycia sieci

- maksymalny rozmiar pliku w CoralCDN - 50MB
- strony fałszowały nagłówki Content-Length, oznaczając połączenia jako trwałe
- HTTP proxy mierzą transfer i blokują “nieuczciwych” klientów

Blokowanie domen

Blacklist

HTTP proxy regularnie pobiera globalną listę zablokowanych domen i nie obsługuje ich.

Powody blokowania domen:

- podejrzenie o phishing
  - CoralCDN nie zmienia treści
  - www nie ma protokołu potwierdzania (jak S-HTTP)
  - SSL potwierdza identyfikacje, ale wymaga dodania serwerów CoralCDN do zaufanych

Blokowanie domen

- łamanie praw autorskich
  - poleceniu DMCA towarzyszy często zgłoszenie materiałów chronionych prawami autorskimi w CoralCDN
  - bez informowania CoralCDN, po usunięciu danych z oryginalnego serwera, treści znikną po maksymalnie 24h

Blokowanie domen - obchodzenie zabezpieczeń

- pewne serwisy przyznają dostęp do chronionych na podstawie adresu IP klienta
- otwarte proxy na takiej liście => wolny dostęp dla każdego
- administratorzy tych serwisów czasem dość bezmyślnie zwiększają pulę uprawnionych IP dla wygody klientów
- w przypadku CoralCDN nie ma takiej potrzeby - IP klienta jest przekazywane
- niestety spora część stron nie korzysta z tej łatwo dostępnej informacji

Blokowanie domen - obchodzenie zabezpieczeń

Udany atak

Cel: uzyskanie dostępu do jednej z blokowanych stron przez CoralCDN (dokładniej czasopismo akademickie)

1. stworzono wpis w DNS o losowej nazwie (losowo.evil.com) wskazujący na adres IP serwera z interesującymi treściami
2. losowy adres nie był blokowany przez CoralCDN
3. CoralCDN pobrał dane i udostępnił nieuprawnionemu użytkownikowi

Proste rozwiązanie - sprawdzanie pola Host w zapytaniu.

Bezpieczeństwo - nazwy

Funkcje nazw domen dla przeglądarek internetowych:

1. wskazują skąd pobierać dane po ich rozwiązaniu na adres IP
2. dają zrozumiałą dla człowieka nazwę organizacji z jaką się kontaktują (jak np "common name" w SSL)
3. wyznaczają ograniczenia wynikające z bezpieczeństwa np dla komunikacji aplikacji webowych

Bezpieczeństwo - nazwy

SOP

Same Origin Policy - specyfikuje jak skrypty z domeny mogą modyfikować dane przeglądarki:

- ciasteczka
- okna przeglądarki, ramki itp
- dostęp do URL przez XmlHttpRequest (AJAX)

Domyślnie wszystko jest dozwolone w ramach tej samej domeny:

- prywatne dane nie mogą być wysłane do innego serwera
- nie można zrobić trywialnego "auto-kliknięcia" w serwisach reklamowych jak np Google AdSense

Bezpieczeństwo - nazwy

- zgodnie z SOP www.example.com ma dostęp ciasteczek example.com
- w CoralCDN to przestaje być bezpieczne - example.com**.nyud.net** i hack.com**.nyud.net**
- CoralCDN kasuje wszystkie ciasteczka z nagłówków
- często zarządzanie ciasteczkami odbywa się przez javascript (w przeglądarce) - wtedy tam pozostają
  1. example.com.nyud.net tworzy ciasteczko przy pomocy javascript
  2. klient wchodzi na hack.com.nyud.net
  3. HTTP proxy kasuje ciasteczko z nagłówka, ale pozostaje ono w przeglądarce
  4. można wysłać teraz XmlHttpRequest z poziomu javascript z hack.com.nyud.net, kodujący zawartość ciasteczka w URL

SOP

Te ataki działają nawet jeśli CoralCDN funkcjonuje na w pełni zaufanych węzłach !

Zasada najmniejszego uprzywilejowania

Każdy element systemu informatycznego ma dostęp tylko do informacji i zasobów, które są niezbędne do spełnienia wyznaczonego celu lub zadania.

Twórcy CoralCDN twierdzą, że tutaj problem leży po stronie SOP a nie CoralCDN

- używanie tylko 2 ostatnich członów nazwy do identyfikacji pomija zbyt wiele informacji
- HTTP proxy mogłyby wykrywać dokładnie uprawnioną domenę (tutaj example.com.nyud.net)

SOP

- SOP okazuje się też w pewnych przypadkach zbyt restrykcyjny
- Współczesne rozwiązania (javascript, HTML5, Flash) próbują obchodzić SOP np dodaje się zewnętrzne treści przez javascript jak mapy Google Maps

Wniosek

Twórcy CoralCDN pokazują, że SOP powinien zostać dopasowany do dzisiejszych potrzeb i powinien pozwalać na elastyczną kontrolę np przez możliwość zdefiniowania reguł dostępu.

Awarie źródłowych serwerów

Scenariusz:

1. popularny portal umieszcza link do mało popularnego serwera
2. użytkownik umieszcza alternatywny link - “skoralizowany” URL serwera na portalu

Możliwe problemy:

- źródłowy serwer staje się niedostępny przed pobraniem danych przez jakiekolwiek HTTP proxy
- CoralCDN już ma kopię strony w cache, ale przekroczyła ona swoje TTL, a źródłowy serwer nie odpowiada
- CoralCDN ma nieaktualną kopię strony, a zapytania do źródłowego serwera dają tylko częściowe odpowiedzi

Reagowanie na awarie

- CoralCDN może otrzymać ogromną ilość zapytań o nieaktywny URL:
  - nieudaje się znalezienie wpisu DNS
  - następuje TCP timeout
  - ...
- HTTP proxy i serwery DNS zapamiętują mają lokalnie "negatywny cache" dla powtarzających się porażek
- bez tego rozwiązania zdarzało się zużyć całe zasoby przy flash crowd do nieaktywnej strony
- CoralCDN dostaje czasem zapytania do stron nieaktywnych od kilku lat

Awarie - udostępnianie nieaktualnych wersji

- CoralCDN nie zastępuje poprawnych wartości nieprawidłowymi
- po wygaśnięciu treści proxy wykonuje zapytanie

  `If-Modified-Since`
- serwer może odpowiedzieć błedem (lub nie odpowiedzieć) - 0.5% tak się kończy
- proxy nie zastępuje strony wersją z błędem, ale wysyła nieaktualną wersję (maksymalnie przez 24h po TTL)

Awarie - zastępowanie tylko pełnymi kopiami

- HTTP proxy ma już nieaktualną wersję strony
- serwer źródłowy jest pod bardzo dużym obciążeniem
- proxy zaczyna pobierać nową, ale transfer nie przychodzi w całości (przynajmniej nie od razu)
- pobierany plik jest zapisywany w pamięci tymczasowej
- zasób jest zastępowany przez nowy dopiero gdy tamten uda się w pełni pobrać

Awarie - wątpliwości

- czy 403 (Forbidden) znaczy, że zasoby nie są już udostępniane czy że serwer jest tymczasowo przeciążony ?
- czy 404 (File Not Found) oznacza, że pliku faktycznie nie ma (np zgłoszenie DMCA), czy tymczasowe (błąd)
- czasem serwery używają odpowiedzi 200 (Success) i wysyłają w HTML komunikat o błędzie
  - w tej sytuacji CoralCDN zastąpi treści błedem
  - wini się tutaj złe domyślne konfiguracje środowisk jak np PHP

Awarie - wątpliwości

Załóżmy, że rozpoznajemy dokładnie błędy, które mają charakter tymczasowy
Jak długo udostępniać nieaktualną wersję ?

- hard-coded: maksymalnie 24h
- można ustawić w nagłówkach wartość max-stale by skonfigurować zachowanie CoralCDN

Morał

"Maintain the status quo unless improvements are possible"

Zarządzanie CoralCDN

- podobną filozofią kieruje się zarządzanie systemem CoralCDN:
  - uaktualnianie stanów węzłów
  - uruchamianie/zatrzymywanie serwisów
  - instalacja/uaktualnienie oprogramowania
- jedna z awarii CoralCDN była spowodowana zwracaniem nieoczekiwanych wartości przez serwer zarządzający
- od tego czasu informacje dotyczące zarządzania są zapisywane aż węzeł otrzyma nową, poprawną instrukcję

Zasoby sieciowe

- komercyjne CDN z reguły zwiększają swoje zasoby sieciowe gdy jest to potrzebne
- CoralCDN ma do dyspozycji tylko PlanetLab
  - po tsunami w Azji w grudniu 2004 (przed youtube) CoralCDN zostało użyte do umieszczenia ogromnej ilości amatorskich filmów
  - PlanetLab nie limitował wtedy użycia sieci
  - CoralCDN zaczął wykorzystywać ogromne ilości zasobów PlanetLab
  - zarządzający węzłami zażądali limitów pod groźbą odłączenia się od sieci
  - uzgodniono, że CoralCDN może użyć dziennie do 10GB transferu per węzeł (PlanetLab silver)

Zasoby sieciowe - limity

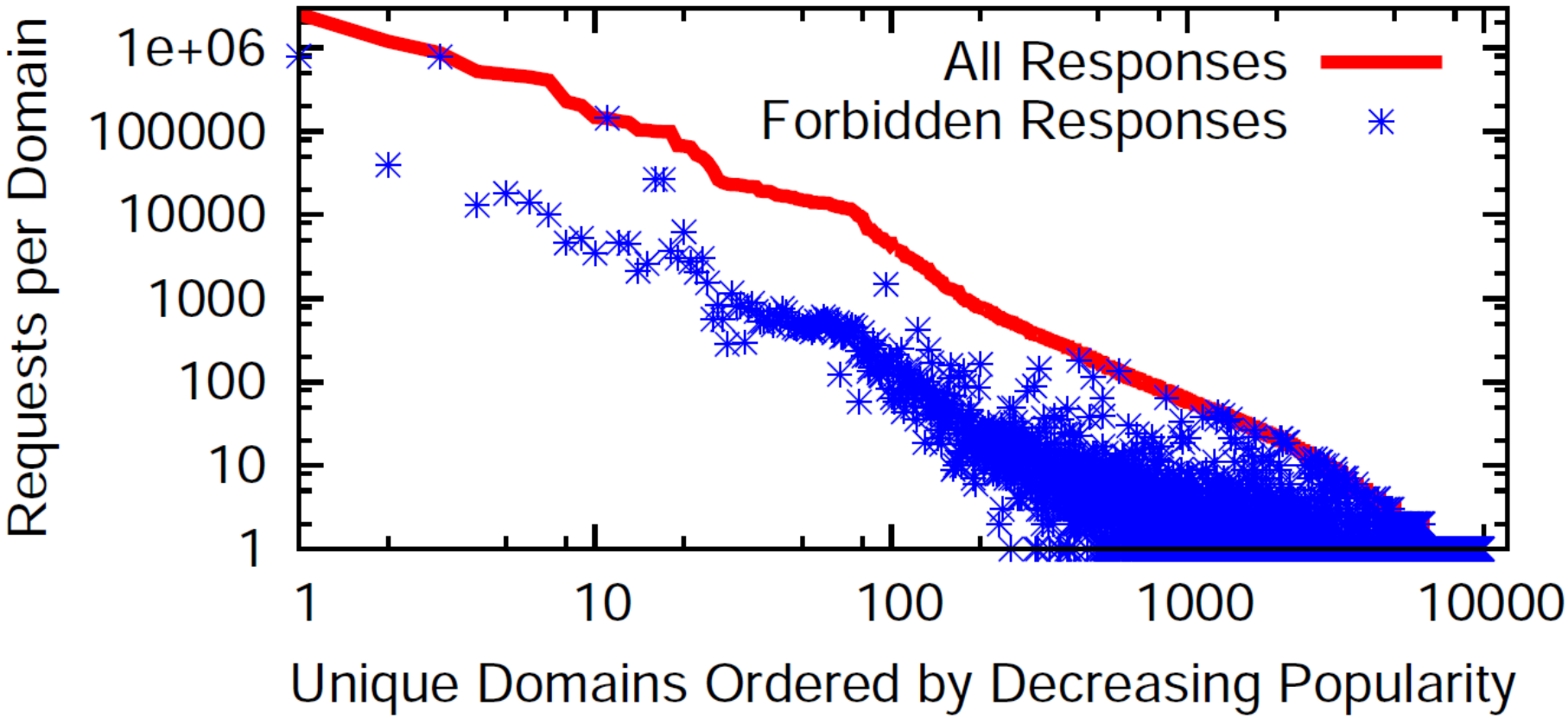
Proste rozwiązania

- odłączanie proxy gdy przekroczy swój limit
- ograniczenie transferu w dłuższym czasie

Wymagania:

- sprawiedliwy przydział sieci między różne domeny
- odporność na flash crowds

Zasoby sieciowe - limity



Zasoby sieciowe - limity

- sprawiedliwy podział sieci może być bardzo kosztowny - CoralCDN obsługuje 10K domen dziennie
- mała część domen zużywa większość transferu
- wystarczy limitować bardzo małą część domen
- konieczne jest rozróżnianie stałego wysokiego zużycia od nagłego wzrostu (flash crowd)

Zasoby sieciowe - limity

Algorytm ogólnie:

1. twardy limit transferu 1000MB na HTTP proxy przez godzinę
2. oczekiwany transfer przez godzinę 400MB
3. dynamiczny limit per domena - statyczny doprowadziłby do bardzo małego użycia sieci

Zasoby sieciowe - limity

Liczenie użycia:

- w trakcie każdej epoki (1h) proxy zapamiętuje ilość danych wysłanych przez domenę
- proxy liczy średnią przez tydzień oraz średnią ze wszystkich domen

Własności:

- przydzielanie limitów domenom opiera się na długich okresach
- flash crowd uzyska potrzebne zasoby
- użycie sieci jest śledzone tylko dla domen należących do 100 pierwszych
- traktowanie domen spoza 100 pierwszych jako używających 0 pozwala alokować w razie potrzeby dodatkowe zasoby na “flash crowds”

Zasoby sieciowe - limity

Działanie:

- kiedy domena przekracza dostępny dla siebie transfer na godzinę, to CoralCDN odpowiada 403 (Forbidden)
- jeśli proxy przekracza swój ogólny limit transferu (wszystkie domeny), to przekierowuje każde zapytanie do oryginalnego serwera i wyłącza się tymczasowo z sieci

Limity w praktyce

- zapotrzebowanie transferu dziennie przekracza 10TB
- faktyczny transfer HTTP wynosi 2TB dziennie (po odrzuceniu sporej liczby zapytań)
- druga co do popularności domena zawiera grafiki mniejsze niż 10KB -> 3.3% zapytań jest odmawianych
- trzecia co do popularności domena składa się z większych treści, materiały wideo, wygaszacze ekranu 5MB każdy -> 89% jest odrzucanych

Konfigurowalność:

- można zwiększyć wagę historycznego transferu przy liczeniu limitów dla domeny
  - zwiększyłoby to zasoby do mitygowania flash crowds
  - CoralCDN nie nadawałby się do długookresowego serwowania popularnych materiałów - sztywne limity

Problem z limitami

- limity liczone są per węzeł
- domena, której klienci pochodzą z różnych regionów trafia do większej ilości proxy
- domena popularna lokalnie używa mniejszego zbioru węzłów
- lokalna domena otrzyma istotnie mniej zasobów
- autorzy CoralCDN nie traktują tego jako poważny problem, ale twierdzą, że jest to ciekawy temat do badań

Niestabilna wydajność

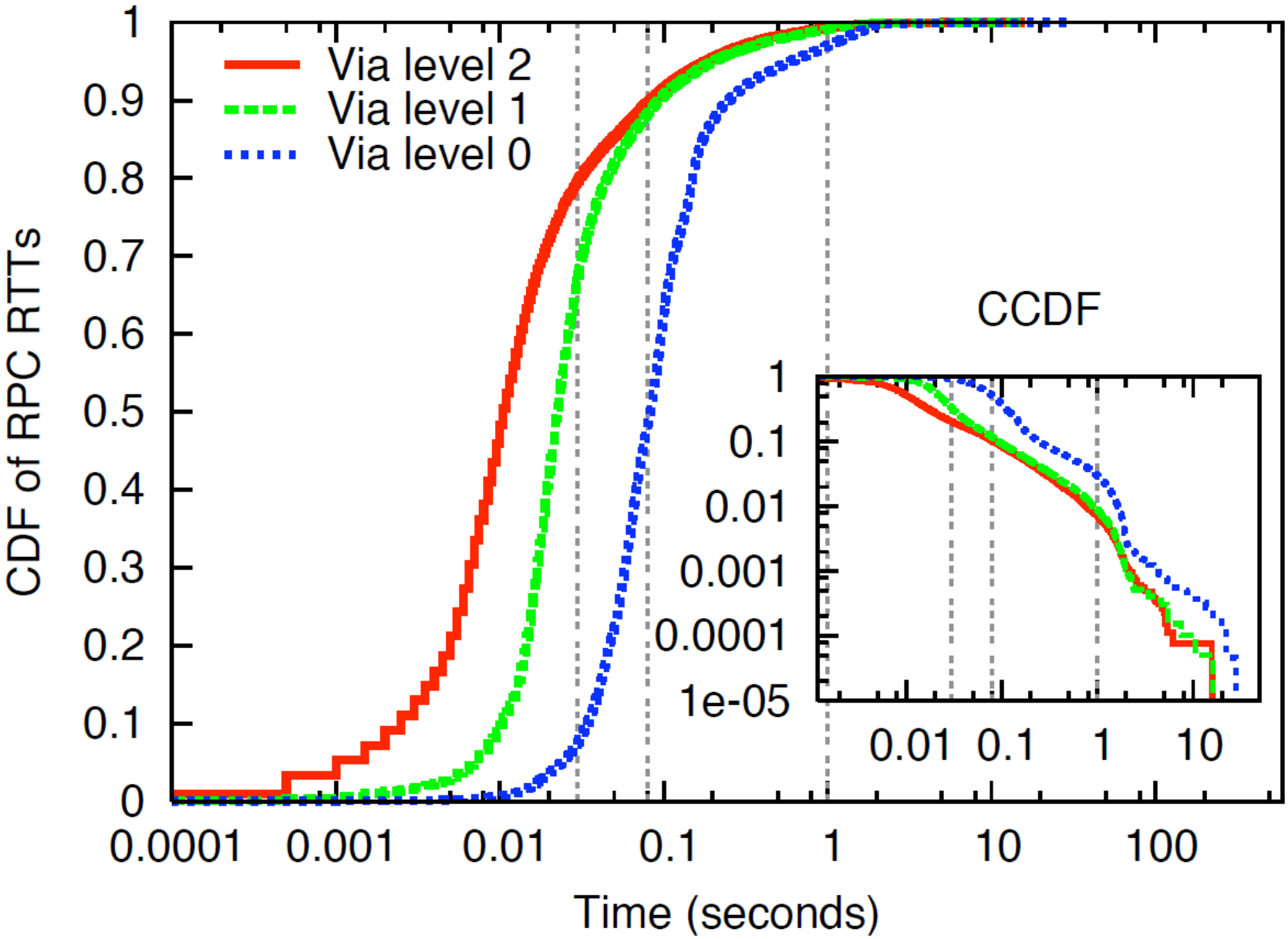
Fakt

CoralCDN działa na serwerach PlanetLab, które są obciążone różnymi innymi zadaniami, dlatego ich wydajność nie jest stała. Praktycznie w każdym systemie rozproszonym musimy liczyć się z podobnym problemem.

DHT w CoralCDN

CoralCDN jest podzielony na klastry w oparciu o RTT, ale nie jest to stała liczba - zarządzanie tym wymaga specjalnych mechanizmów.

DHT w CoralCDN



DHT w CoralCDN

Zasady:

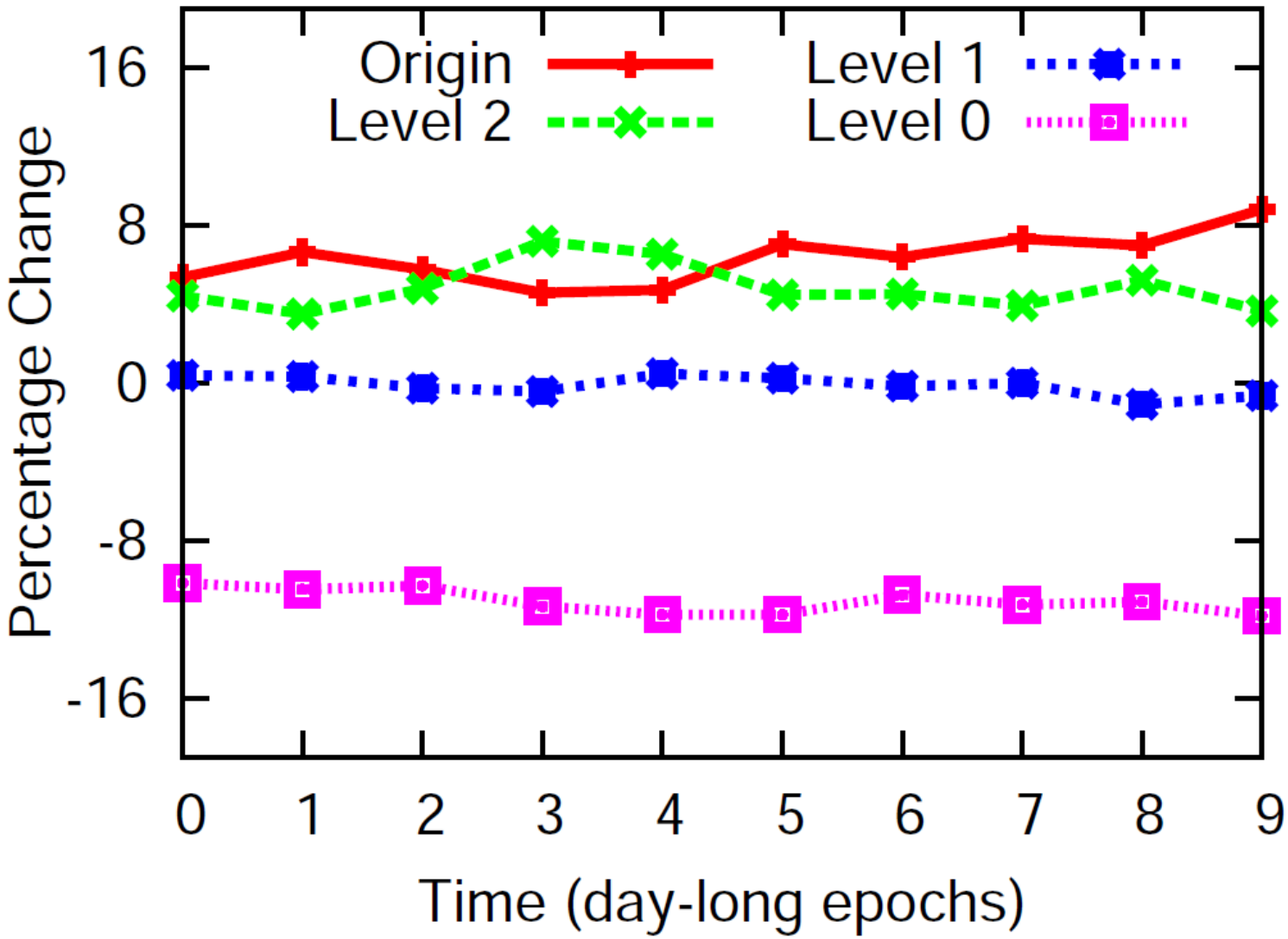
- węzeł przyłącza się do klastra jeśli RTT do 85% jego członków jest niższy niż limit (30ms - poziom 2, 80ms poziom 1)
- węzeł przechodzi do mniejszego klastra jeśli mniej niż 70% członków aktualnego ma odpowiednie RTT
- odchodzący węzeł stworzy własny klaster (singleton) tylko jeśli $> 50\%$ sąsiadów ma zbyt duże RTT

Wniosek

Przyłączenie się do klastra jest obwarowane wysokim progiem, ale by go opuścić warunki muszą znacznie się pogorszyć.

- 1% RPC zajął 1s - nawet w tym samym klastrze ! Równoległe RPC pozwala zmniejszyć znaczenie tego problemu.

Zwiększenie lokalności



Zwiększenie lokalności

W celu zmniejszenia ilości globalnych wyszukiwań, połowa węzłów przy wyszukiwaniu w warstwie indeksującej robi to tylko na poziomie 2 i 1.

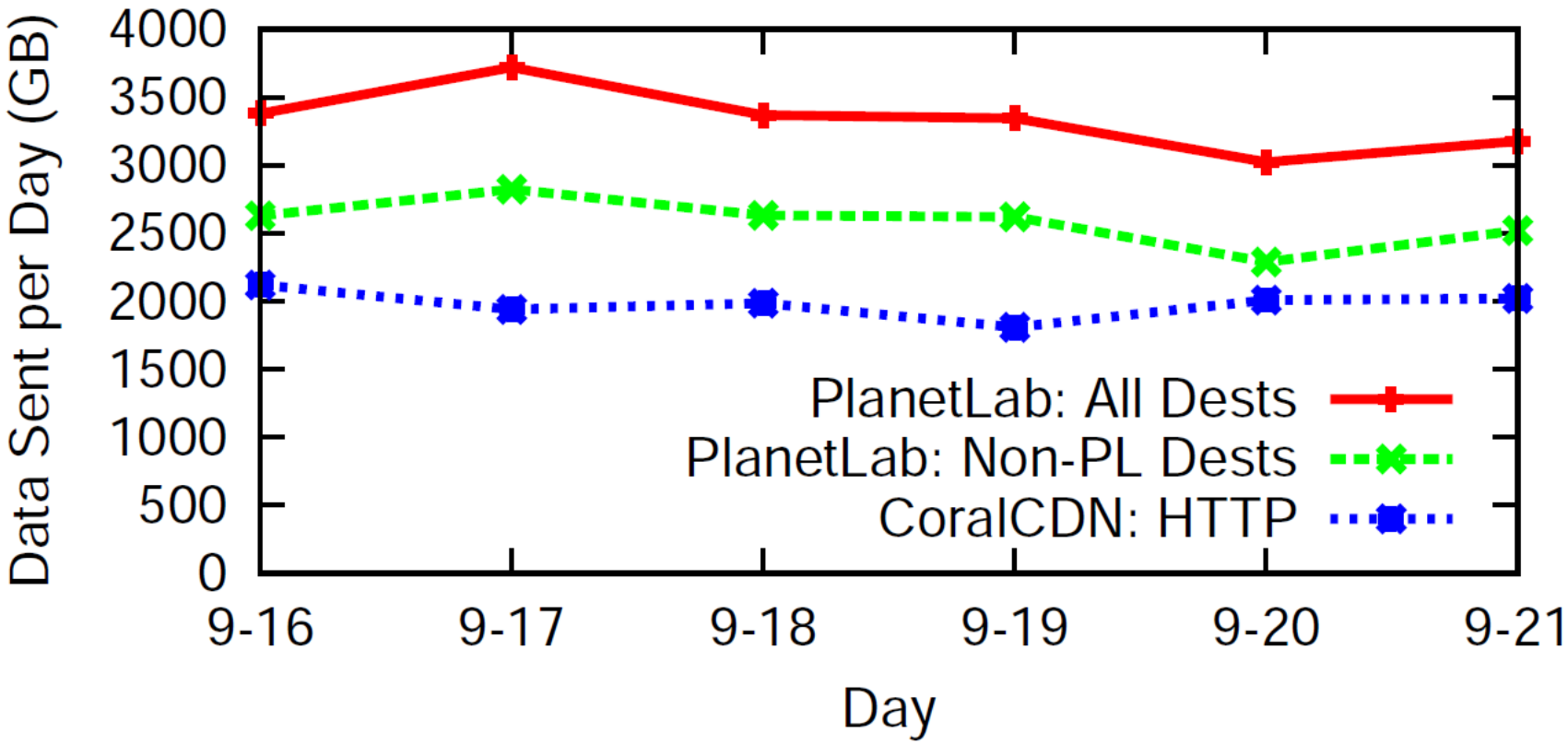
- ilość globalnych wyszukiwań spadła o 12%
- globalny indeks jest wciąż potrzebny i spełnia swoją rolę - 0.56% zapytań nieudanych na originalnym serwerze jest rozwiązywana

Platformy

PlanetLab:

- kiedy serwer przekracza 80% swojego transferu - 17.2GB na dzień, to system zaczyna limitować przepustowość sieci
- CoralCDN używa sztywnego limitu 10GB dziennie na węzeł
- okazuje się, że codziennie 5-10 (z 300-400) serwerów wysyła więcej niż 13.8GB
- CoralCDN liczy tylko poprawne transfery HTTP, pomija DNS, DHT RPC i nieudane zapytania
- statystyki kernela są 50%-90% wyższe niż CoralCDN

Platformy


Data Sent per Day (GB)
4000
3500
3000
2500
2000
1500
1000
500
0
9-16
9-17
9-18
9-19
9-20
9-21
Day
PlanetLab: All Dests
PlanetLab: Non-PL Dests
CoralCDN: HTTP

Platformy

Fakt

Dokładny pomiar użycia sieci jest bardzo trudny lub kosztowny z poziomu aplikacji użytkownika.

Postulaty twórców CoralCDN:

- udostępnienie dokładnych statystyk użycia sieci przez system operacyjny
- umożliwienie skonfigurowania polityki zarządzania użyciem zasobów
  - CoralCDN w tym wypadku dla sieci ustawiłby priorytet DNS i DHT nad transferem HTTP

Platformy

- z punktu widzenia systemu operacyjnego liczy się tylko krótki okres - dla aplikacji jak CoralCDN ważne są dane z długiego okresu
- zapotrzebowanie na takie dane występuje też w komercyjnych rozwiązaniach
  - Akamai udostępnia tylko przybliżone użycie sieci
  - Amazon w czerwcu 2009 uruchomił usługę Cloud-Watch pozwalającą monitorować użycie zasobów na poziomie pojedynczej maszyny (VM)

Odporność na błędy

- wąskim gardłem w niezawodności CoralCDN jest początkowe rozwiązanie adresów DNS
- 10-12 serwerów DNS CoralCDN jest zarejestrowanych w serwerach gTLD zarządzających domeną *.net*
- awaria jednego z nich powoduje wydłużenie opóźnień w rozwiązywaniu nazw "coralized" u klienta
- CoralCDN reaguje dynamicznie na awarie wewnątrz sieci - serwery *.net* są poza systemem
- klient może przechowywać IP odpowiadający nazwie (np by utrudnić ataki na SOP - zamianę IP) - awaria jednego węzła uniemożliwi mu połączenie jak i reakcję ze strony CoralCDN

Odporność na błędy

Rozwiązania:

1. można dynamicznie aktualizować listę serwerów w domenie *.net*, RFC2136 - specyfikuje do tego protokół (rzadko wspierany)
2. wysyłanie adresów IP poprzez anycast, protokołami BGP lub OSPF (protokoły routujące, pomiędzy AS)
3. użycie mechanizmów reagowania na błędy z poziomu sieciowego (np VIP/DIP)

Usprawiedliwienie

Żaden z wymienionych mechanizmów jest wspierany przez PlanetLab czy gTLD.

Amazon EC2 wprowadziło w marcu 2008 mechanizm "Elastic IP Adresses", pozwalający na ukrycie mechanizmów reagowania na błędy (ARP spoofing) dla maszyn wirtualnych.

Wnioski

- CoralCDN jest udanym przykładem działającego, otwartego rozwiązania typu CDN, z wygodnym API
- wydaje się, że jest zbyt skomplikowane - prostsze rozwiązanie mogłoby działać równie dobrze i szybciej się uruchamiać
- problemy bezpieczeństwa wynikają nie tylko z faktu użycia niezaufanych węzłów (SOP)
- można stworzyć więcej usług na podstawie dodawania sufiksów do nazwy (udekorować URL)
- jedynie podpisywanie zawartości da pełne bezpieczeństwo treści
- CoralCDN zakłada, że oryginalny serwer jest zawsze przeciążony, w praktyce spora część serwerów dynamicznie podejmuje decyzję o tym, czy przekierować ruch - potrzebny jest bogatszy interface po stronie CDN

References


Michael J. Freedman, Experiences with CoralCDN: A Five-Year Operational View, Princeton University


http://en.wikipedia.org/wiki/Kademlia


http://pl.wikipedia.org/wiki/Zasada_najmniejszego_uprzywilejowania

The End