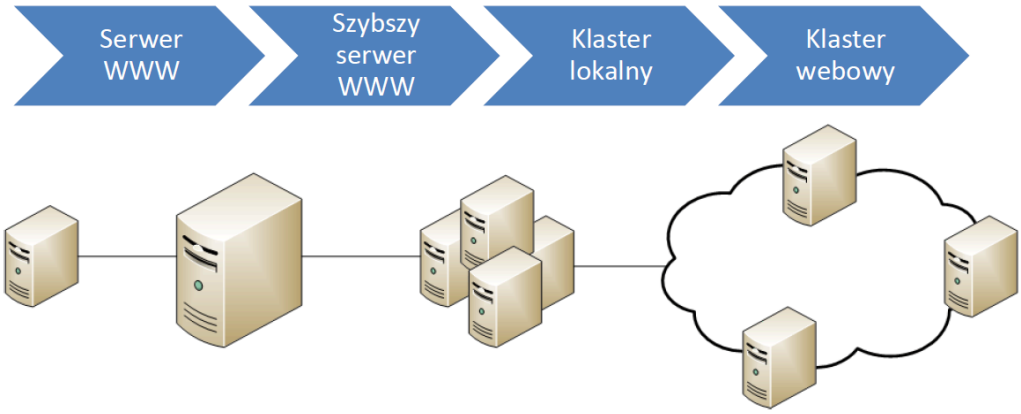
Systemy webowe - laboratorium 4

Symulacja Sieci Dystrybucji Treści z wykorzystaniem pakietu symulacyjnego CDNSim

Informatyka WIZ PWr
semestr zimowy 2014/2015

Przygotował:
Zespół dydaktyczny Zakładu Rozproszonych Systemów Komputerowych
Instytutu Informatyki Politechniki Wrocławskiej



Rysunek 1: Ewolucja systemów webowych

1 Sieci dystrybucji treści

1.1 Wstęp

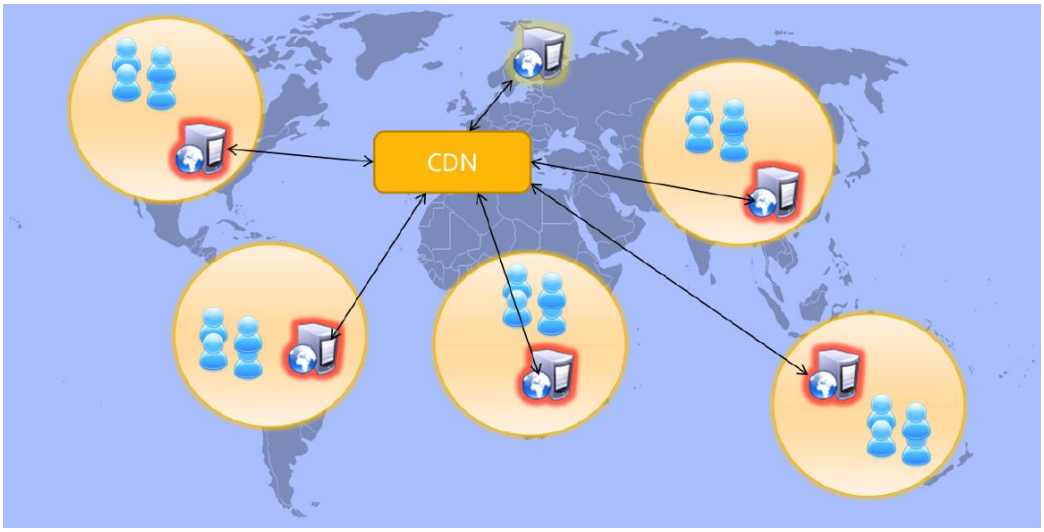
Wraz z gwałtownym rozwojem Internetu na początku lat 90-tych ubiegłego wieku, szybko okazało się, że pojedynczy serwer nie jest i nigdy nie będzie w stanie obsłużyć dużej liczby użytkowników. Społeczność IT stanęła przed dużym problemem, jakim niewątpliwie była wykładniczo rosnąca liczba użytkowników sieci. Pierwsze próby sprostania wyzwaniu polegały na wyposażaniu serwerów WWW w szybkie dyski, duże pamięci i najnowsze (jak na tamte czasy) procesory.

Kolejnym krokiem było zmuszenie do współpracy wielu maszyn ściśle kooperujących ze sobą – czyli klastrów. Jednakże nawet najmocniejszy klaster nie jest w stanie wyeliminować opóźnienia jakie jest konsekwencją maksymalnej szybkości propagacji fali na łączach szkieletowych Internetu. Innym czynnikiem przemawiającym za poszukiwaniem nowych rozwiązań było pojawienie się efektu *Flash Crowd* (lub *Slash Dot*)[1]. W kontekście Internetu *Flash Crowd* utożsamiany jest z nagłym zwiększeniem ruchu webowego i liczby zapytań do serwera WWW. Zjawisko to ma następujące cechy:

- wzrost liczby zapytań jest olbrzymi (na ogół powoduje zapchanie serwera),
- wzrost liczby zapytań jest szybki, ale nie błyskawiczny, najczęściej trwa od kilkunastu minut do kilkunastu godzin,

[1] *Flash Crowd* to termin, który pierwszy raz pojawił się w powieści fantastyczno-naukowej amerykańskiego pisarza – Larrego Nivena. W wyimaginowanym świecie ludzie z łatwością mogli teleportować się do miejsc, w których działo się właśnie coś ciekawego tworząc tłumy gapiów.

- jest powodowane przez niewielki procent treści przechowywanej na serwerze (na przykład przez stronę główną portalu z newsami), wąskim gardłem jest przepustowość łącza serwera.



Rysunek 2: Sieć CDN jest globalnie rozproszoną, wydajną infrastrukturą sieciową

Przykładami zjawisk Flash Crowd mogą być: wzmożony ruch po atakach terrorystycznych w 2001, który zablokował serwery CNN i BBC lub kwiecień 2010 kiedy po katastrofie smoleńskiej strony `wp.pl` i `gazeta.pl` przestały odpowiadać. Z czasem zaczęto lokalizować klastry w różnych punktach naszego globu (tzw. klastry webowe). Posiadanie wielu klastrów w różnych centrach kolokacyjnych ma następujące zalety:

- treść znajduje się bliżej użytkowników co skutkuje redukcją opóźnienia,
- cała infrastruktura jest odporniejsza na ataki typu DDoS,
- jakość dostarczonej treści może być wyższa.

Jak również posiada pewne minusy, takie jak:

- utrudnione zarządzanie skomplikowaną strukturą,
- koszt posiadania wielu serwerowni,
- utrudniona współpraca urządzeń różnych producentów.

Rozwiązaniem kompletnym, integrującym klastry webowe, systemy monitorujące, billingujące i zarządzające stały się **Sieci Dystrybucji Treści**.

Zbiór geograficznie rozproszonych urządzeń sieciowych, współpracujących ze sobą w sposób transparentny w celu dostarczenia użytkownikom wysokiej jakości treści nazywamy ***Siecią Dystrybucji Treści*** *(Content Delivery/Distribution Network).*

Największe firmy na rynku takie jak Akamai czy Limelight Neworks posiadają kilkaset własnych centrów kolokacyjnych i kilkadziesiąt tysięcy serwerów. Swoje dane na serwerach CDN lokalizują Adobe, Microsoft, Audi czy też Yahoo. Mistrzostwa świata 2010 były obsługiwane przez Akamai, które chwali się, że obsługuje 20% ruchu w Internecie. W tym przypadku można było przewidzieć zachowanie użytkowników ponieważ zdarzenie był zaplanowane. Jak rozlokować serwery gdy tego nie wiemy? Czy replikować całą treść czy tylko jej część? W jaki sposób przekierować użytkowników do serwera, który obsłuży ich żądania w najkrótszym czasie? Między innymi na te pytania trzeba odpowiedzieć planując wykorzystanie sieci CDN.

1.2 Elementy

Serwery webowe są tylko jednym z elementów Sieci Dystrybucji Treści. W sieci CDN partycypują na ogół trzy strony: użytkownicy, właściciel treści oraz dostarczyciel usług CDN. Poniżej podana została terminologia użyta w dalszej części instrukcji:

1. **treść** (*ang. content*) – składa się z zakodowanej zawartości oraz opisu w postaci metadanych, może to być dokument HTML, plik wideo lub audio;

2. **operator treści** (*ang. content provider*) – jest to właściciel treści, na ogół duża firma świadcząca usługi dla milionów użytkowników (na przykład Microsoft, CNN, BBC, MTV);

3. operator sieci dystrybucji treści - jest stroną dostarczającą swoją infrastrukturę operatorom treści, największe firmy na rynku to Akamai, Limelight Networks oraz MirrorImage;

4. **użytkownicy** (*ang. end users*);

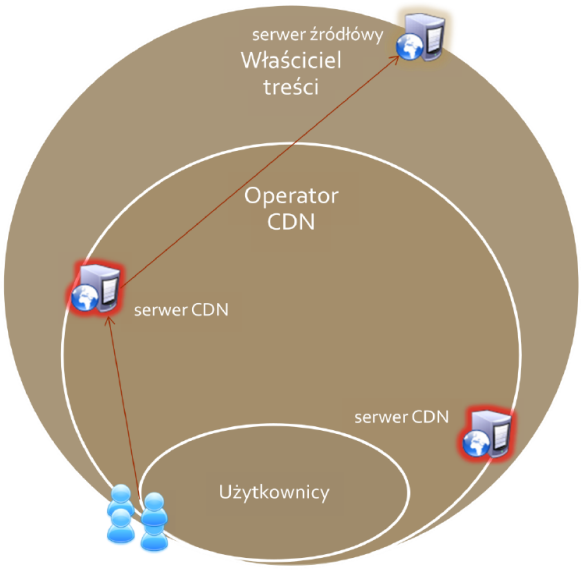
5. **serwer źródłowy** (*ang. origin server*) – jest maszyną lub klastrem maszyn, na którym publikowana jest pierwotna wersja treści, operator sieci dystrybucji treści musi zapewnić mechanizmy replikacji i dystrybucji każdej zmiany na serwery partycypujące w sieci CDN;

6. **serwer replik** (*ang. replica server, surrogate*[2]) – jest serwerem lub klastrem serwerów należących do operatora sieci dystrybucji treści, na którym przechowywane są pliki, które nie podlegają częstym modyfikacjom;

7. **serwer keszujący** (*ang. cache server, caching server*) – serwer przechowywujący treści, które często się zmieniają (np. personalizowane strony WWW, strony główne portali), podstawową różnicą między serwerem replik i keszującym jest częstość zmian;

8. **klaster webowy** (*ang. web cluster*) – jest grupą współpracujących serwerów webowych, w kontekście sieci dystrybucji treści, rozproszonych na dużym obszarze.

Najczęściej serwer replik i serwer keszujący są fizycznie jednym urządzeniem a powierzchnia dyskowa dzielona jest pomiędzy obie funkcjonalności zgodnie z przyjętymi założeniami.



Rysunek 3: Poglądowe zależności między stronami partycypującymi w sieci CDN

Operator (właściciel) treści wykupuje usługi u dostawcy usługi CDN. Dzięki temu może lokalizować swoją treść "blisko" użytkownika. Przykładowo duży portal informacyjny w Polsce mógłby zdecydować się na zlokalizowanie swoich serwerów poprzez sieć CDN w kilku dużych miastach Polski takich jak: Łódź, Poznań czy Warszawa i przekierowywać użytkowników do nich zamiast do głównego serwera. Można powiedzieć, że chodzi tu o równoważenie obciążenia w

[2]surrogate z ang. zastępca

rozproszonym systemie oraz zapewnienie użytkownikowi jak najwyższej jakości usługi (najkrótszy czas ładowania strony, wyższa jakość materiałów wideo, etc.). Dokładne działanie sieci CDN zostanie przedstawione w kolejnym podpunkcie.

1.3 Działanie

Istnieje 5 podstawowych kwestii związanych z działaniem Sieci Dystrybucji Treści. Są to:

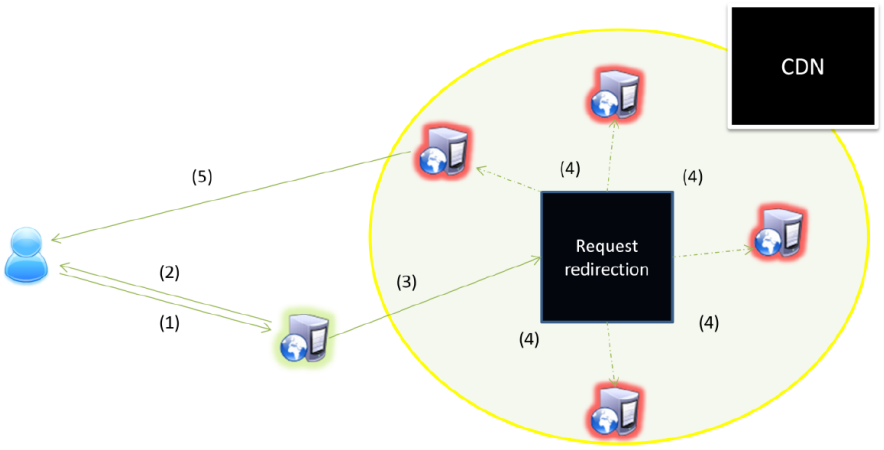
- **wybór i sposób dostarczenia treści** (*ang. content selection and delivery*) – odpowiada na pytanie: którą część treści wybrać do rozprowadzenia w Sieci Dystrybucji Treści jeśli mamy ograniczenia na wielkość przestrzeni dyskowej?
- **lokalizacja serwerów CDN** (*ang. surrogate placement*) – odpowiada na pytanie: w których lokalizacjach rozmieścić serwery, aby jakość treści była jak najwyższa a czas jej dostarczenia jak najkrótszy?
- **outsourcing treści** (*ang. content outsourcing*) – odpowiada na pytanie: co zrobić jeśli treści nie ma na serwerze oraz na których serwerach umieścić konkretną treść?
- **organizacja i zarządzanie serwerami kesz** (*ang. cache organization and management*) – odpowiada na pytanie: czy serwery kesz powinny tworzyć hierarchię i jaki algorytm keszowania powinien być użyty?
- **ruting żądań** (*ang. request routing*) – odpowiada na pytanie: gdzie przekierować dane żądanie lub ciąg żądań, aby czas ich obsługi był jak najkrótszy?

W ramach laboratorium zajmiemy się badaniem algorytmów starających się rozwiązać powyższe problemy. Porównamy cztery proste algorytmy rutingu żądań HTTP.

1.3.1 Ruting żądań

Działanie systemu CDN od otrzymania żądania do jego realizacji można opisać w kilku krokach przedstawionych na rysunku 4.

1. Użytkownik wysyła żądanie http, np. GET /example.com.
2. Żądanie dzięki systemowi DNS dociera do serwera źródłowego, który odpowiada wysyłając szkielet strony.
3. Żądania o obiekty wbudowane są przekierowywane do wnętrza sieci CDN.



Rysunek 4: Ruting żądań

4. Moduł odpowiedzialny za rozdzielenie żądań podejmuje decyzje o ich realizacji na konkretnych maszynach.[3]

5. Obiekty wbudowane wysyłane są do klienta, transfer zostaje ukończony.

Na laboratorium zbadamy 4 algorytmy dystrybucji żądań i outsourcingu treści.

- **Najbliższy serwer** (*ang. closest surrogate*) - żądanie dociera do serwera oddalonego o najmniejszą liczbę 'hopów' IP (w naszym przypadku - ruterów). Jeśli żądanego obiektu nie ma na tym serwerze, kooperuje on z innymi i pobiera obiekt do własnej pamięci z najbliższego serwera alternatywnego a następnie zwraca obiekt klientowi.

- **Najbliższe źródło** (*ang. closest origin*) – ponownie żądanie wędruje do najbliższego serwera. W przypadku gdy żądanego obiektu nie ma na nim, jest on pobierany do pamięci z najbliższego serwera źródłowego.

- **Losowy** (*ang. random surrogate*) – żądanie wędruje do losowego serwera a w przypadku gdy danego obiektu nie ma na serwerze jest on pobierany do pamięci z innego, przypadkowo wybranego serwera alternatywnego, który posiada żądaną treść.

- **Z równoważeniem obciążenia** (*ang. load balance surrogate*) – żądanie jest przekierowane do najbliższego serwera. W przypadku gdy jego obciążenie grozi przeciążeniem systemu (obciążenie waha się w granicach 95%)

[3]Decyzja może być podjęta bezpośrednio przez serwer źródłowy. W takim przypadku najczęściej stosuje się metodę nadpisania linków w dokumentach (X)HTML.

jest ono wysyłane do najmniej obciążonego serwera alternatywnego. Jeśli obiektu nie ma na serwerach keszujących jest on pobierany na podobnej zasadzie z serwerów źródłowych.

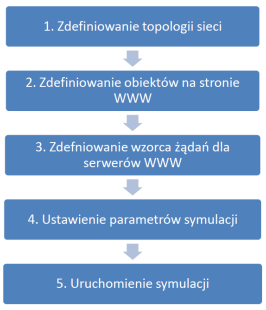
Wymienione wyżej algorytmy będziemy badać w środowisku symulacyjnym CDNSim.

2 Pakiet CDNSim

Pakiet CDNSim jest symulatorem sieci komputerowych opartym na środowisku OMNET++ [4]. Jest on, na ten moment, jedynym wiarygodnym narzędziem pozwalającym porównywać różne rozwiązania w obszarze Sieci Dystrybucji Treści. Pakiet CDNSim posiada interfejs graficzny pozwalający na konfigurację parametrów dotyczących symulacji. Najważniejsze z nich to:

- maksymalna przepustowość łączy,
- liczba połączeń obsługiwanych przez serwer,
- liczba klientów,
- liczba serwerów keszujących,
- liczba serwerów źródłowych.

Ustawienia pojedynczej symulacji zapisywane są do pliku symulacji (*ang. bottle*) aby zautomatyzować procedurę dla wielu symulacji różniących się ustawieniem parametrów. Schemat na rysunku 5 pokazuje kroki jakie należy wykonać, aby stworzyć pojedynczy plik symulacji.

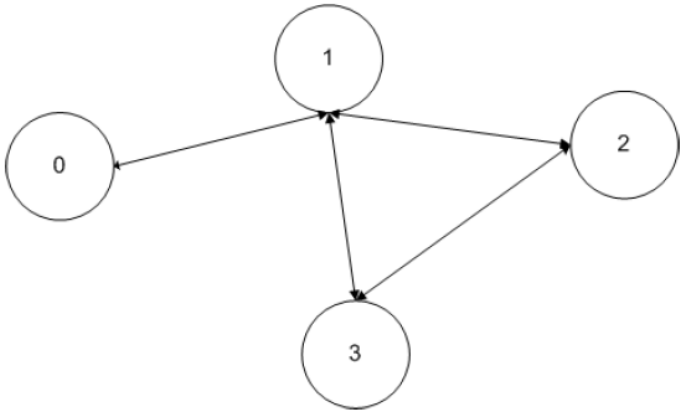


Rysunek 5: Schemat przygotowania symulacji

[4] `http://www.omnetpp.org/`

2.1 Topologia sieci

Topologia sieci jest grafem połączonych ruterów. Serwery (kesz i źródłowe) oraz klienci będą dołączeni do sieci w sposób przypadkowy (znany tylko autorom) z tą różnicą, że dla tej samej topologii za każdym razem otrzymamy to samo rozmieszczenie serwerów. Topologie możemy opisać jako graf $G = (V, E)$, gdzie V jest zbiorem ruterów $v \in \{0 \dots N\}, N > 0$, a N jest numerem ostatniego rutera. Wszystkie rutery muszą być numerowane kolejno, w przeciwnym razie otrzymamy graf rozłączny. Relacja określa połączenia między ruterami, $(v_1, v_2) \in E$, jeśli istnieją obustronne połączenia między ruterami v_1 i v_2. Nie ma potrzeby definiować połączeń z dwóch stron. Dla przykładu topologia:
$G = (0, 1, 2, 3), (0, 1), (1, 2), (2, 3), (1, 3)$ opisuje następującą sieć przedstawioną na rysunku 6.



Rysunek 6: Przykładowe połączenie ruterów

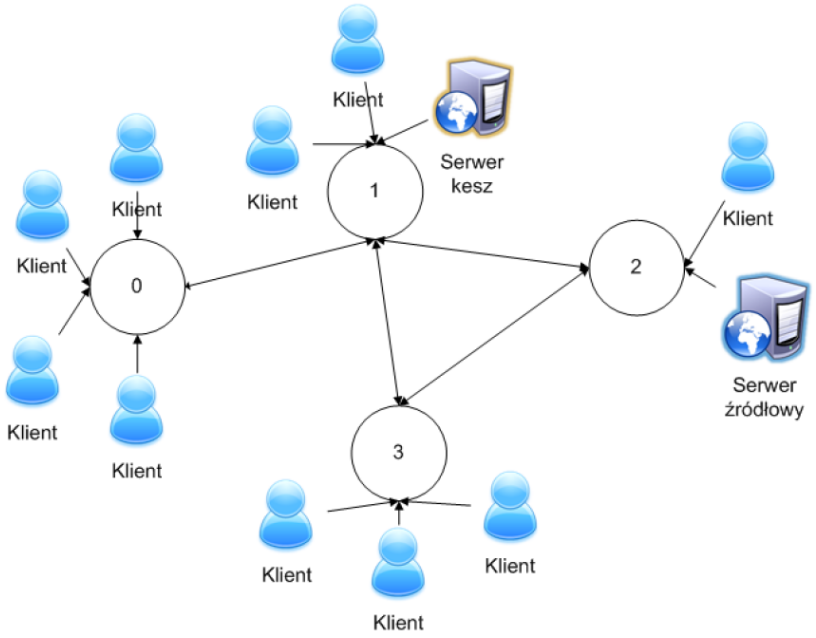
Plik topologii musi być zgodny z formatem **„int int/n”**, czyli w naszym przypadku w pliku topologii powinien znaleźć się taki oto zapis przedstawiony na rysunku 7.

```
0 1
1 2
2 3
1 3
```

Rysunek 7: topologia_1.txt

Na laboratorium pliki topologii nazywać będziemy topologia_1.txt, topologia_2.txt, ..., topologia_X.txt, gdzie X jest liczbą symulacji, które będziemy przeprowadzać (jeśli rzecz jasna korzystają z różnych topologii).

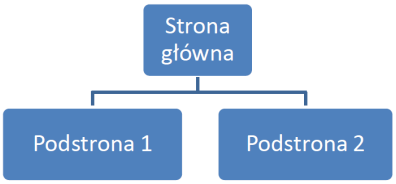
Klienci, serwery źródłowe oraz serwery kesz/replik są dołączone do topologii w sposób losowy. Zakładając, że posiadamy 1 serwer źródłowy, 1 serwer kesz oraz 10 klientów końcowa sieć użyta w symulacji może wyglądać tak jak na rysunku 8:



Rysunek 8: Przykładowa sieć w symulacji z dołączonymi klientami i serwerami

2.2 Obiekty na stronie WWW

Strona WWW zakodowana w języku (X)HTML może zawierać elementy wbudowane (*ang. embedded objects*). CDNSim pozwala na określenie rozmiaru tych obiektów używając ponownie zapisu **„int int/n”**. Pierwsza liczba to identyfikator obiektu a druga to jego rozmiar w bajtach. W symulatorze nie definiujemy bezpośrednio całych stron WWW, to znaczy takich, które same w sobie mają zagnieżdżone obiekty. Zamiast tego definiujemy pojedyncze obiekty (ponieważ każdemu obiektowi odpowiada kolejne zapytanie HTTP np.: GET) a fakt powiązania w ramach jednej strony ujmiemy przy opisywaniu wzorca ruchu dla wybranego serwisu WWW. W celu zobrazowania sobie tych kwestii załóżmy, że nasz serwis WWW składa się z 3 stron (rys. 9). Kody źródłowe stron zgodne ze specyfikacją HTML 5 przedstawione są na rysunku 10.



Rysunek 9: Struktura prostego serwisu

Strona główna

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html><html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Strona główna</title>
  </head>
  <body>
    <p>Paragraf 1</p>
    <div>
           <img src="obrazek1.jpg"/>
    </div>
    <figure>
           <img src="obrazek2.jpg">
           <figcaption>Pierwszy obrazek w HTML 5</figcaption>
    </figure>
  </body>
</html>
```

Podstrona 1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html><html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Podstrona 1</title>
  </head>
  <body>
           <img src="obrazek3.jpg"/>
           <img src="obrazek4.jpg"/>
           <img src="obrazek5.jpg"/>
  </body>
</html>
```

Podstrona 2

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html><html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Podstrona 2</title>
  </head>
<body>
         <img src="obrazek6.jpg"/>
         <img src="obrazek7.jpg"/>
         <img src="obrazek8.jpg"/>
</body>
</html>
```

Rysunek 10: Kody źródłowe stron zgodne ze specyfikacją HTML 5

Mamy zatem 3 strony HTML 5, zawierające łącznie 8 obiektów wbudowanych. Dodatkowo jako obiekt będziemy liczyć też szkielet strony (przykład na rysunku 11).

ID	Obiekt	Rozmiar (bajty)
0	Strona główna	1000
1	Podstrona 1	890
2	Podstrona 2	890
3	obrazek1.jpg	5000
4	obrazek2.jpg	8000
5	obrazek3.jpg	12000
6	obrazek4.jpg	12000
7	obrazek5.jpg	20000
8	obrazek6.jpg	20000
9	obrazek7.jpg	23000
10	obrazek8.jpg	12000

Rysunek 11: Rozmiary rozpatrywanych obiektów

Pliki stron WWW na laboratorium nazywać będziemy strona_1.txt, strona_2.txt, strona_X.txt, gdzie X ponownie jest liczbą różnych serwisów, które chcemy przetestować. Plik opisu strony dla naszego, małego serwisu prezentuje się tak jak na rysunku 12.

```
0 1000
1 890
2 890
3 5000
4 8000
5 12000
6 12000
7 20000
8 20000
9 23000
10 12000
```

Rysunek 12: strona_1.txt

Do wygenerowania ruchu będziemy musieli znać również powiązania między obiektami w ramach jednej strony. Posłuży nam do tego plik o nazwie hierarchia_1.txt o formacie **„int int/n”**, gdzie pierwsza liczba to identyfikator strony a druga identyfikator obiektu (jak w strona_1.txt) . Dla naszej strony wygląda on tak jak na rysunku 13.

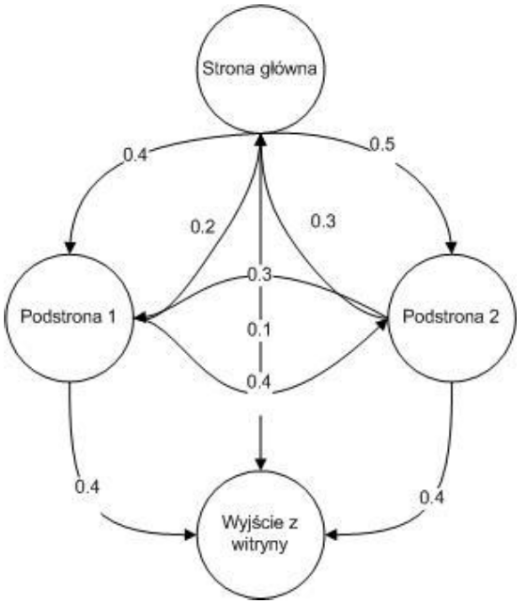
2.3 Generowanie ruchu WWW

Zachowanie użytkowników odwiedzających stronę może być opisane na kilka sposobów. Jednym z nich jest graf przejść, który określa prawdopodobieństwa przejścia z jednej strony na drugą. Taki opis ruchu WWW wykorzystywany jest

```
0 0
0 3
0 4
1 1
1 5
1 6
1 7
2 2
2 8
2 9
2 10
```

Rysunek 13: hierarchia_1.txt

najczęściej kiedy nie mamy do dyspozycji statystyk odwiedzin bądź logów serwera. Suma prawdopodobieństw przejścia z jednej strony na drugą (bądź wyjścia ze strony) musi być równa 1 dla każdej z nich.



Rysunek 14: Przykładowy wzorzec zachowania użytkownika

Dodatkowo każda ze stron może mieć pętlę zwrotną, która może oznaczać odświeżenie strony. Dla uchwycenia wzorca zachowania użytkownika ta pętla nie jest istotna, może być jednak istotna w naszym przypadku ponieważ odświeżenie strony pociągnie za sobą ponowne pobranie treści[5]. W powyższym modelu za-

[5]Jeśli założymy, że ani przeglądarka ani system operacyjny nie keszują zawartości strony. Jest to pewne uproszczenie ponieważ na ogół przeglądarki internetowe przechowują pobraną treść w pamięci podręcznej

łożyliśmy też, że użytkownik zawsze zaczyna serwowanie w serwisie od strony głównej. Można by pominąć do założenia wprowadzając dodatkowy węzeł (np. START) prowadzący do innych stron, który wskazywałby od jakiej strony użytkownik zaczyna sesję. Na laboratorium grafy przejść będziemy nazywać kolejno graf_1.txt, graf_2.txt, etc. Format pliku to **„int int double/n”**, gdzie pierwsza liczba oznacza węzeł początkowy (id strony jak w hierarchia_1.txt), druga węzeł końcowy a trzecia prawdopodobieństwo przejścia. Użyjemy dwóch węzłów specjalnych, aby zasymulować początek (-1) oraz koniec sesji (-2). Nasz graf przejść miałby w tej notacji postać jak na rysunku 15.

```
-1 1 1
1 2 0.4
1 3 0.5
1 -2 0.1
2 1 0.2
2 3 0.4
2 -2 0.5
3 1 0.3
3 2 0.3
3 -2 0.4
```

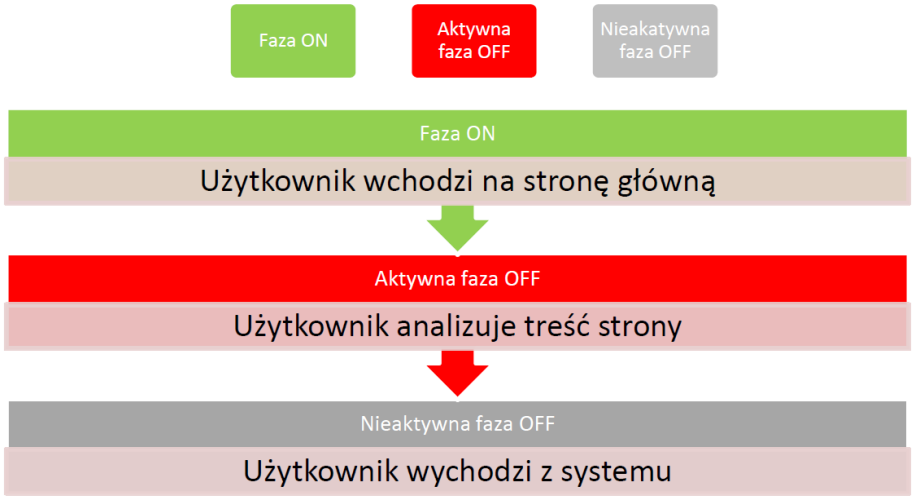
Rysunek 15: graf_1.txt

W tej instrukcji skorzystamy z grafu przejść, aby symulować zachowanie użytkowników (ruch WWW). Będą nam do tego potrzebne dwie rzeczy: wspomniany wcześniej graf przejść oraz model określający czas przebywania użytkownika w witrynie. Czas jaki spędza użytkownik na stronie może być opisany rozkładami prawdopodobieństwa w modelu ON-OFF. Model ON-OFF zakłada, że użytkownik aktywnie działa w fazie ON oraz czeka w fazie OFF. Faza OFF dodatkowo dzieli się na aktywną fazę OFF i nieaktywną fazę OFF. W czasie aktywnej fazy OFF użytkownik analizuje treść (na przykład czyta artykuł na stronie) a w czasie nieaktywnej fazy OFF nie korzysta z naszego portalu (na przykład odszedł od komputera, ale ma włączoną przeglądarkę lub też zamknął okno przeglądarki). Odpowiada to przejściu w grafie do węzła ‘Wyjście z witryny’.

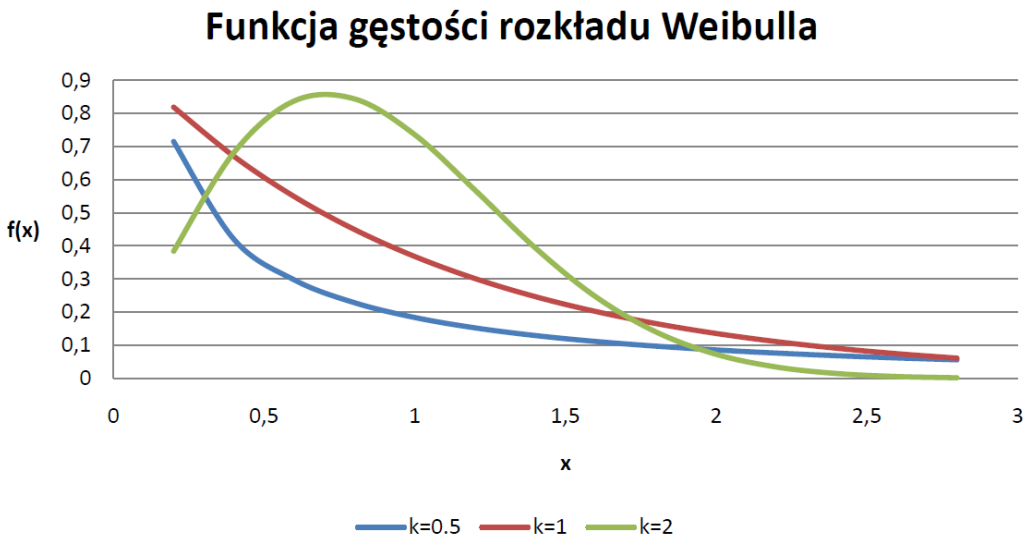
Faza ON zostanie zasymulowana przez CDNSim (Pobranie obiektu). Aktywną fazę OFF zasymulujemy korzystając z rozkładu Weibulla. Funkcja gęstości tego rozkładu wygląda następująco:

$$f\left(x;\lambda,k\right)=\begin{cases}\frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1}e^{-\left(\frac{x}{\lambda}\right)^{k}} & x\geq 0\\ 0 & x<0\end{cases} \tag{1}$$

Gdzie k jest parametrem kształtu a λ parametrem skali. Przykładowy wykres funkcji gęstości rozkładu Weibulla przedstawiony jest na rysunku 17.



Rysunek 16: Model ON-OFF



Rysunek 17: Funkcja gęstości rozkładu Weibulla

Wartość oczekiwana rozkładu opisana jest wzorem:

$$\lambda\Gamma\left(1+\frac{1}{k}\right) \tag{2}$$

W przypadku gdy $1+\frac{1}{k}$ jest liczbą całkowitą wartość oczekiwana funkcji wynosi:

$$\lambda\left(\frac{1}{k}\right)! \tag{3}$$

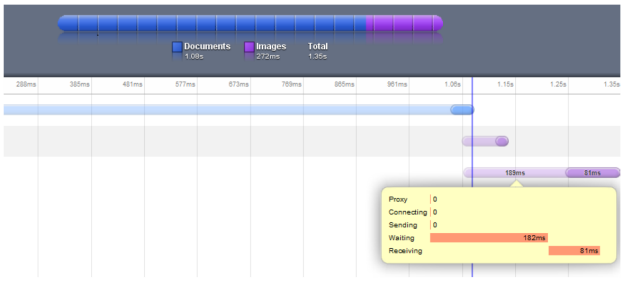
Na laboratorium przyjmiemy $k=1$ a zmieniać będziemy tylko parametr λ. Nieaktywną fazę OFF (czas przetwarzania informacji przez użytkownika i wyświetlania jej na ekranie) zasymulujemy rozkładem Pareto. Funkcja gęstości rozkładu Pareto dane jest wzorem:

$$p\left(x\right) =\propto k^{\propto}x^{-\propto-1},\ \propto,\ k>0,\ x\geq k \tag{4}$$

gdzie k jest najmniejszą wartością zmiennej losowej. W naszym przypadku $k=1$ może oznaczać, że użytkownik będzie analizował treść strony przynajmniej 1 sekundę. Wartość oczekiwana (czyli oczekiwany czas przyswajania treści) opisuje następujący wzór:

$$\frac{\alpha k}{\alpha-1} \tag{5}$$

Ostatnią rzeczą, jakiej musimy być świadomi to to, że CDNSim symuluje żądania na poziomie obiektu a nie strony. Jeśli wchodzimy na stronę główną w naszym przypadku pobrane zostaną 2 obrazki i szkielet strony. Wygląda to następująco[6]:



Rysunek 18: Diagram czasu pobierania strony głównej naszego przykładowego portalu w HTML5

[6]Zrzut ekranu z narzędzia diagnostycznego przeglądarki Google Chrome

Pierwszy krok to pobranie szkieletu strony (HTML5) a drugi to przetworzenie dokumentu według DOM[7] i pobranie elementów zagnieżdżonych. Jak widać obrazki nie są pobrane w tym samym czasie. Jednym z powodów jest fakt struktury drzewiastej dokumentu, program wysyłający żądania trafia na elementy wbudowane w różnych momentach czasu. Na laboratorium zasymulujemy ten fakt rozdzielając żądania o obiekty statyczne o wybrany, stały okres czasu. Dodatkowo, aby nie wysłać żądań o obiekty wbudowane przed ściągnięciem szkieletu strony, wykorzystamy wzór na przepustowość TCP/IP, aby obliczyć czas ściągania strony głównej (będzie to minimalny czas rozdzielający żądanie o stronę główną i pierwszy obiekt wbudowany). Wzór ten jest następujący:

$$THR = \frac{C \times MSS}{RTT \times \sqrt{p}} \tag{6}$$

gdzie p to prawdopodobieństwo utraty pakietu w sieci TCP a MSS to największy rozmiar segmentu TCP przesyłany w sieci bez fragmentowania. $C = 1.22$ jest stałą. Dla uproszczenia przyjmiemy $RTT = 100\,ms$, $p = 0.0004$, $MSS = 1460$ (dla sieci IP4). Czas pobierania szkieletu strony głównej o wielkości 1000 bajtów wynosił będzie zatem:

$$\frac{1000}{\frac{1{,}22 \times 1460}{0{,}1 \times \sqrt{0{,}0004}}} = 1,12\,ms \tag{7}$$

Wiedząc już jakie zależności panują między stronami naszego portalu oraz jak użytkownik będzie zachowywał się na każdej z nich, możemy zasymulować zachowanie pojedynczego użytkownika. Aby to zrobić wykorzystamy zebrane wiadomości.

Do dyspozycji mamy witrynę WWW składającą się ze stron HTML. Każda strona WWW składa się ze szkieletu oraz obiektów wbudowanych o określonych rozmiarach. Zachowanie użytkownika opisane jest grafem przejść. W momencie wejścia na stronę użytkownik ściąga szkielet strony a następnie obiekty wbudowane. Po ściągnięciu całej strony użytkownik czeka w fazie aktywnej OFF (ten czas modelujemy rozkładem Weibulla). Następnie przechodzi na kolejną stronę (według prawdopodobieństwa w grafie przejść). Jeśli przechodzi na stronę serwisu cała sytuacja się powtarza, jeśli zaś wychodzi z systemu jego nieobecność zamodelujemy rozkładem Pareto.

Format pliku opisującego ruch to **„double int int/n”**. Gdzie pierwsza liczba określa moment czasu, druga identyfikator użytkownika a trzecia żądany obiekt.

Użytkownik „5” (identyfikator to druga liczba) wchodzi na stronę główną, następuje wysłanie żądania:

[7] Document Object Model

0 5 1000

czekamy 1.12s i ściągamy dwa obiekty wbudowane (obrazek1.jpg i obrazek2.jpg) wysyłając żądania:

1120 5000
1130 8000

przyjęliśmy, że stała związana z przechodzeniem po drzewie DOM to 10 ms. Teraz użytkownik analizuje treść ściągniętej strony a więc czekamy 'odcinek czasu' związany ze ściągnięciem elementów wbudowanych oraz czas modelowany przez rozkład Weibulla (na przykład z wartością oczekiwaną 2000 ms). Po tym kroku następuje przejście do kolejnej strony lub wyjście z systemu. Wyjście z systemu oznacza, że użytkownik połączy się ponownie ze stroną po czasie modelowanym rozkładem Pareto (na przykład z wartością oczekiwaną 100 000 ms). Do laboratorium dołączony został skrypt w Pythonie pozwalający na automatyzację działania wielu użytkowników według przedstawionego opisu. Wystarczy zdefiniować parametry:

- grafu przejść (graf_1.txt),
- rozkładów,
- liczbę użytkowników,
- czas aktywności (czas całej symulacji).

Aby nie wszyscy użytkownicy wysyłali pierwsze żądanie w tym samym czasie, prawdopodobieństwo dołączenia kolejnego użytkownika rośnie liniowo wraz z czasem aż do zdefiniowanego punktu, w którym wynosi 1.

Jak możemy zauważyć symulacja ruchu w serwisie webowym jest w tym przypadku dość uciążliwa. Pliki opisujące ruch nazywać będziemy ruch_1.txt, ruch_2.txt, etc.

```
0 5 1000
0 1120 5000
0 1130 8000
```

Rysunek 19: Plik wzorca ruchu, ruch_1.txt

Aby skorzystać z dołączonego skryptu należy zdefiniować pliki:

- strona_1.txt (opis rozmiarów obiektów),
- hierarchia_1.txt (opis rozmieszczenia obiektów na stronach),
- graf_1.txt (opis grafu przejść),

oraz podać parametry rozkładów i liczbę użytkowników. Parametry potrzebne do wygenerowania pliku ustawiamy w skrypcie ‘generuj_ruch.py’, który kopiujemy do katalogu z poprzednio zdefiniowanymi plikami. Parametry jakie trzeba zdefiniować znajdują się na początku skryptu.

```
#BEGIN USTAWIENIA
#Liczba uzytkowikow N, i maksymalny czas symulacji [s] - t, lw - maksymalna liczba rownolegle otwartych polaczen
N = 100
t = 1000
lw = 2
#ustawienia parametrow lacz, rtt [s]
stratnosc = 0.0004
rtt = 0.2
#Faza Active OFF (Weibull), E = L*(1/k)!= L jeśli k = 1, [s]
W_k = 1
W_L = 2
W_E = W_L
#Faza Inactive OFF (Pareto), E = alfa*k/(alfa+1), k = minimalna wartosc zmiennej losowej [s]
W_k = 1
W_E = 20
W_alfa = W_E/(W_E - W_k)
#Symulacja dodatkowego czasu miedzy pobraniem szkieletu strony a obiektami wbudowanymi (Pareto), [ms]
W2_k = 2
W2_E = 10
W2_alfa = W2_E/(W2_E - W2_k)
#END USTAWIENIA
```

Rysunek 20: Ustawienia dla generacji ruchu WWW

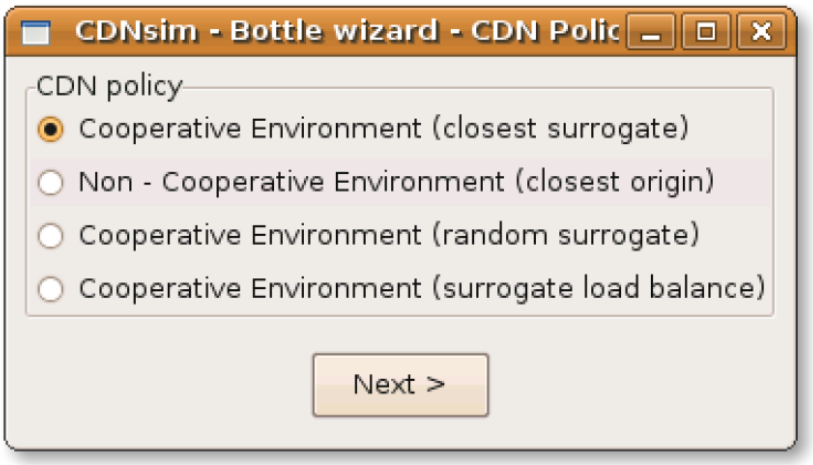
Należy zmienić:
N – liczbę użytkowników w systemie,
t - czas „trwania” ruchu WWW,
lw – liczba wątków jakie równolegle uruchamia przeglądarka przy pobieraniu strony WWW,
stratność – stratność łącza,
rtt – średni czas RTT między klientem a serwerem,
W_L – oczekiwany czas „przetwarzania strony” przez użytkownika $[s]$,
W_E – oczekiwany czas bezczynności użytkownika po wyjściu z systemu $[s]$.
Przykładowe ustawienia widoczne są na rysunku 21.
Po uruchomieniu skryptu wygenerowany zostanie plik ruch_1.txt, którego użyjemy w symulacji. Na wyjściu skryptu pojawią się też podstawowe parametry ruchu.

```
Wygenerowano 2896 żądań o łącznym rozmiarze 8881.09 MB, 8.90 MB/s w okresie czasu 1000 [s]
```

Rysunek 21: Przykładowe ustawienia

2.4 Ustawienie parametrów symulacji

Mając zdefiniowany szkielet strony, topologie sieci oraz wzorzec ruchu czasu przejść do symulacji[8]. Przechodzimy do katalogu, w którym został zainstalowany symulator i uruchamiamy GUI poleceniem:
`python CDNSim.py`
W pierwszym oknie musimy wybrać algorytm rutingu żądań (zostawiamy domyślne ustawienia).



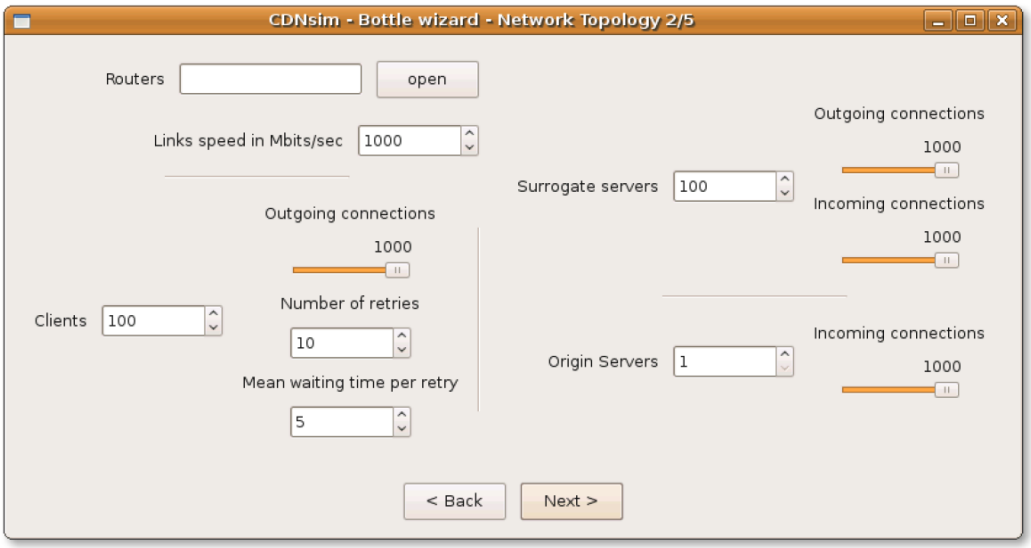
Rysunek 22: Wybór algorytmu

W kolejnym konfigurujemy parametry symulacji oraz wskazujemy plik z topologią sieci.

Liczba klientów musi być zgodna z tą jaką podaliśmy w skrypcie generującym ruch WWW. Poniżej przedstawiono znaczenia parametrów symulacji:

- Routers - ścieżka do pliku topologia_1.txt,
- Links speed – maksymalna przepustowość łączy w sieci (każde łącze ma taką samą),
- Surrogate servers – liczba serwerów kesz/replik,
- Clients: outgoing connections – maksymalna liczba połączeń utrzymywanych przez klienta,

[8] `http://oswinds.csd.auth.gr/~cdnsim/docs/tut1/index.html`



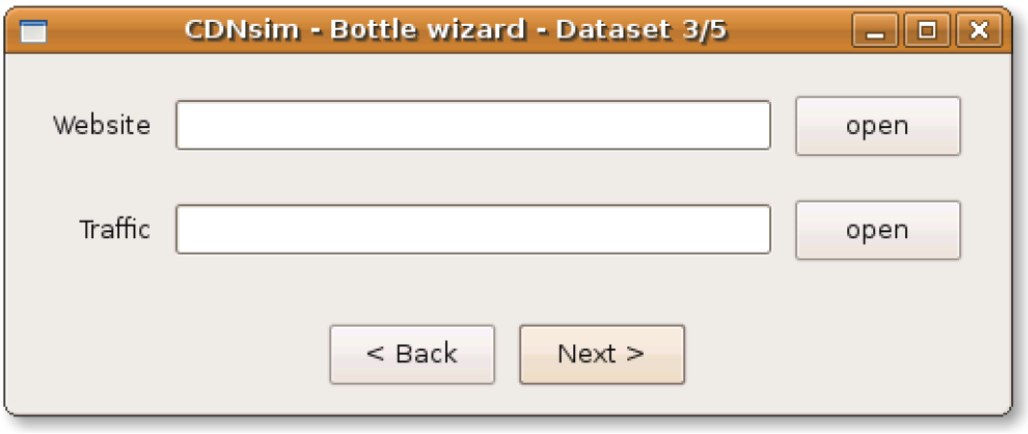
Rysunek 23: Konfiguracja parametrów symulacji

- Clients: number of retries – liczba ponowień żądania w przypadku odmowy czy też błędu poprzedniego,
- Clients: mean waiting time per retry – średni czas oczekiwania przed ponowieniem żądania,
- Surrogate servers: outgoing connections – liczba połączeń jakie może utrzymywać serwer do innych serwerów (na przykład w przypadku nie znalezienia treści w pamięci podręcznej i konieczności pobrania jej z innego serwera),
- Surrogate server: incoming connections – liczba połączeń jakie może obsłużyć serwer, jeśli zostanie przekroczona serwer przestanie odpowiadać,
- Origin servers – liczba serwerów źródłowych, czyli takich, które posiadają kompletną treść.

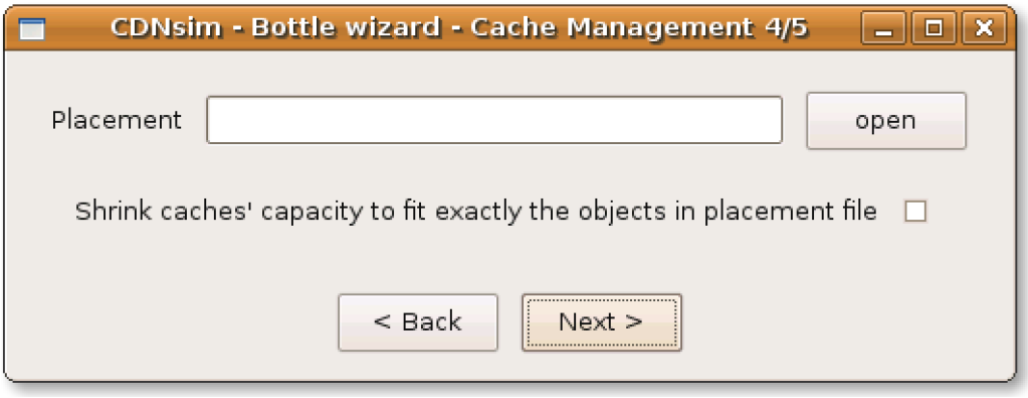
Następnie ładujemy plik opisujący stronę oraz wzorzec ruchu WWW jaki chcemy przetestować, powinny być to pliki strona_1.txt oraz ruch_1.txt.

Teraz czas na konfigurację naszych serwerów. W tym kroku możemy rozdzielić obiekty w sposób arbitralny między (jeszcze przed początkiem symulacji).

Plik składa się z rekordów, każdy po 5 wartości. Rekordów powinno być tyle ile serwerów kesz/replik. Struktura rekordu jest następująca:



Rysunek 24: Dołączanie plików opisujących stronę i wzorzec ruchu WWW



Rysunek 25: Rozdzielenie obiektów

```
0 // numer serwera - od 1 do N-1
1 // ma być zawsze 1
NULL //NULL lub nazwa pliku z obiektami początkowo
ulokowanymi na serwerze
LRU // polityka pamięci kesz, Last Recently Used, czyli
zamianie ulegają najdłużej nie używane obiekty
1200 // wielkość pamięci w bajtach
```

Nie podanie pliku z rozmieszczeniem obiektów oznacza, że serwer początkowo będzie miał pusty dysk i z czasem zapełni się obiektami. Nie będzie obiektów, których nie można usunąć więc host będzie działał jako serwer kesz a nie replik. Wielkość pamięci powinna być ustalona w odniesieniu do całkowitej wielkości serwisu. Odpowiedni plik nazwiemy rozmieszczenie_1.txt a wygenerować go możemy przy pomocy skryptu ‘rozmieszczenie.py’. Podobnie jak w przypadku

poprzedniego skryptu konfigurujemy kilka parametrów znajdujących się na jego początku.

```
#BEGIN USTAWIENIA
N = 10 #liczba serwerów
S = 10000 #wielkość pamięci w bajtach
#END USTAWIENIA
```

Rysunek 26: Konfiguracja parametrów na początku

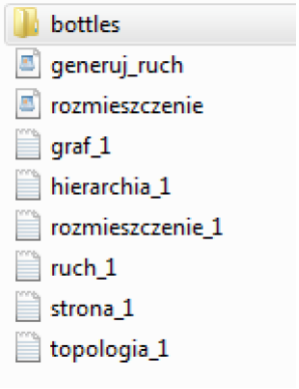
Oczywiście wielkość pamięci musi być mniejsza niż wielkość naszego serwisu. Ostatnim krokiem będzie zapisanie ustawień eksperymentu do pliku. Nazwy plików powinny być następujące: bottle_1.txt, bottle_2.txt, etc.

Rysunek 27: Zapisanie ustawień eksperymentu do pliku

Możemy powtórzyć część lub wszystkie kroki i stworzyć nowy plik symulacji w tym samym katalogu. Jeśli szczęśliwie dotarliśmy do tego momentu, nasz katalog eksperymentu powinien prezentować się jak na rysunku 28.

Skrypty generuj_ruch.py oraz rozmieszczenie.py są udostępnione na maszynie wirtualnej w katalogu “skrypty” umieszczonym na pulpicie. Wszystkie eksperymenty należy wykonywać równocześnie. Aby to zrobić wykorzystamy skrypty dostarczone z pakietem CDNSim.

```
cd /home/user/Desktop/CDNsim/CDNsim/CDNsimbatch/
./batch.sh KATALOG_BOTTLES
```



Rysunek 28: Katalog eksperymentu

gdzie: KATALOG_BOTTLES oznacza katalog, w którym zapisany został plik bottle z rozszerzeniem .tgz np. bottle-1-txt.tgz

Po wykonaniu symulacji należy wykonać polecenie przetwarzające wyniki.

```
cd /home/user/Desktop/CDNsim/CDNsim/CDNsimstats
./stats.sh KATALOG_BOTTLES
/home/user/Desktop/CDNsim/CDNsim/CDNsimstats/stats.py
/home/user/Desktop/CDNsim/CDNsim/CDNsimstats/util.py
```

Plik z wynikiem symulacji powinien znajdować się w katalogu bottles.

Dla każdego pliku otrzymamy szereg statystyk o podanym formacie jak na rysunku 29:

Pole	Wartość	Opis
Bottle	Przykład	Nazwa pliku
Client Side Measurements		**Pomiary po stronie klientów**
Completed requests	300	Liczba obsłużonych żądań
Aborted requests	12	Liczba nieobsłużonych żądań
Retries	33	Liczba ponowień
Not enough outgoing connections	3	Zbyt mała liczba wolnych połączeń
Mean response time	0.18	Średni czas odpowiedzi
Surrogate side measurements		**Pomiary po stronie serwerów**
Hit ratio percentage	5.9	Procent obiektów znalezionych w keszu
Byte hit ratio percentage	8.99	Procent treści znalezionej w keszu
Not enough outgoing connections	3	Zbyt mała liczba wolnych połączeń
Not enough incoming connections	5	Zbyt mała liczba wolnych połączeń
Failed pulls	8	Nieudana liczba pobrań treści z innych serwerów
Origin side measurements		**Pomiary po stronie serwera źródłowego**
Not enough incoming connections	2	Zbyt mała liczba wolnych połączeń
Surrogate Server utilities		**Wykorzystanie sieci**
Active surrogate servers	13	Liczba serwerów obsługujących ruch
Mean surrogate Server utility	0.4	Stosunek upload/download

Rysunek 29: Plik statystyk

3 Sprawozdanie nr 3 do wykonania

UWAGA! Wszystkie pliki powinny znajdować się w jednym katalogu.
Za poprawnie napisanie całego sprawozdania można otrzymać maksymalnie 10 punktów. Jeżeli zaś ktoś preferuje rozwiązać tylko część zadań w ramach sprawozdania, to punktacja za częściowe rozwiązanie jest następująca:

- Za poprawne rozwiązanie zadań od 1 do 5 można otrzymać maksymalnie 3 punkty.
- Za poprawne rozwiązanie zadania 6 można otrzymać maksymalnie 3 punkty.
- Za poprawne rozwiązanie zadania 7 i 8 można otrzymać maksymalnie 3 punkty.
- Za poprawne rozwiązanie całego zadania 9 można otrzymać maksymalnie 1 punkt.

1. Proszę zdefiniować prosty serwis webowy (inny niż przykładowy), składający się z 4 stron (strona główna + 3 podstrony) w specyfikacji zgodnej z

HTML 5. Strony powinny być sprawdzone pod kątem zgodności z draftem W3C[9]. Serwis powinien posiadać 12 obiektów wbudowanych (o łącznym rozmiarze ponad 4 MB).
Plik opisu strony proszę zapisać pod nazwą strona_1.txt. Pod spodem proszę opisać strukturę serwisu w postaci grafu lub drzewa. Strukturę w notacji **„int int/n”** proszę umieścić w pliku hierarchia_1.txt oraz pod spodem w sprawozdaniu.

2. Proszę zdefiniować topologię sieci i zapisać ją w pliku topologia_1.txt. Pod spodem proszę naszkicować odpowiedni graf i podać opis w postaci odpowiedniej relacji (topologia powinna zawierać przynajmniej 10 ruterów).

3. Proszę naszkicować graf przejść między stronami (według własnego pomysłu) i zapisać jego strukturę w notacji **„int int double/n”** w pliku graf_1.txt oraz w sprawozdaniu.

4. Dla każdej strony proszę wyliczyć minimalny czas ściągania przyjmując wartości parametrów (stratność =0,0004 i RTT = 100 ms).

5. Proszę wygenerować ruch WWW korzystając z dostarczonego skryptu i zapisać go w pliku ruch_1.txt, ruch_2.txt, etc. Proszę podać parametry użyte do generowania ruchu.

6. Proszę sprawdzić charakterystyki systemu w zależności od liczby użytkowników w systemie. Liczba użytkowników powinna być tak dobrana, aby w końcu przeciążyć system (podane liczby są orientacyjne). Aby usprawnić pracę proszę utworzyć 3 pliki eksperymentu i wykonać je w tym samym czasie. Proszę uzupełnić tabelę.

[9] http://validator.w3.org/

Pole	Eksperyment 1	Eksperyment 2	Eksperyment 3
Liczba użytkowników	5	20	100
Algorytm rutingu			
Czas symulacji [s]			
Bottle			
Client Side Measurements			
Completed requests			
Aborted requests			
Retries			
Not enough outgoing connections			
Mean response time			
Surrogate side measurements			
Hit ratio percentage			
Byte hit ratio percentage			
Not enough outgoing connections			
Not enough incoming connections			
Failed pulls			
Origin side measurements			
Not enough incoming connections			
Surrogate Server utilities			
Active surrogate servers			
Mean surrogate Server utility			

7. Dla liczby użytkowników równej 60-80% liczby przeciążającej system proszę przy tych samych ustawieniach środowiska porównać wydajność 4 różnych algorytmów rutingu.

Pole	Algorytm 1	Algorytm 2	Algorytm 3	Algorytm 4
Liczba użytkowników	(0.5 – 0.8) x LMAX			
Algorytm rutingu				
Czas symulacji [s]				
Bottle				
Client Side Measurements				
Completed requests				
Aborted requests				
Retries				
Not enough outgoing connections				
Mean response time				
Surrogate side measurements				
Hit ratio percentage				
Byte hit ratio percentage				
Not enough outgoing connections				
Not enough incoming connections				
Failed pulls				
Origin side measurements				
Not enough incoming connections				
Surrogate Server utilities				
Active surrogate servers				
Mean surrogate Server utility				

8. Dla najlepszego algorytmu z poprzedniego punktu proszę spróbować oszacować maksymalną liczbę użytkowników, jaką może obsłużyć nasza sieć CDN.

9. Proszę odpowiedzieć na następujące pytania:

   - Co świadczy o przeciążeniu systemu?
   - Jakie kroki można przedsięwziąć aby obsłużyć większą liczbę użytkowników?
   - Jaki wpływ na zachowanie systemu ma lub może mieć graf przejść?