Symulacja działania protokołu RIP

Autor:
Radosław Mikołajczyk

Programowanie Współbieżne i Rozproszone

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Spis treści

1 Tytuł programu **2**

2 Dane studenta **2**

3 Data oddania programu **2**

4 Cel programu - krótki opis **2**

5 Opis i schemat struktury zadaniowej programu **3**
5.1 Dodawanie bezpośredniego sąsiada 3
5.2 Broadcasting 4

6 Żądanie zdalnego obliczenia liczby pierwszej **5**

7 Informacje o stosowanych pakietach zewnętrznych (niestandardowych) **6**

8 Informacje o zastosowaniu specyficznych metod rozwiązania problemu **6**
8.1 Algorytm obliczania liczb pierwszych 6
8.2 RIP 7

9 Krótka instrukcja obsługi **8**
9.1 API 8
9.2 Instrukcje 9

10 Testy, przykłady **10**

11 Możliwe rozszerzenia programu **13**

12 Ograniczenia programu **13**

13 Inne informacje **14**

1 Tytuł programu

Program nosi tytuł ***Symulacja protokołu RIP***.

2 Dane studenta

Dane poniżej:
Imię i Nazwisko: Radosław Mikołajczyk
Rok studiów: trzeci
Kierunek studiów: informatyka
Nr grupy dziekanatowej: 2b
Nr indeksu: 303272

3 Data oddania programu

Data oddania projektu: 19 stycznia 2021

4 Cel programu - krótki opis

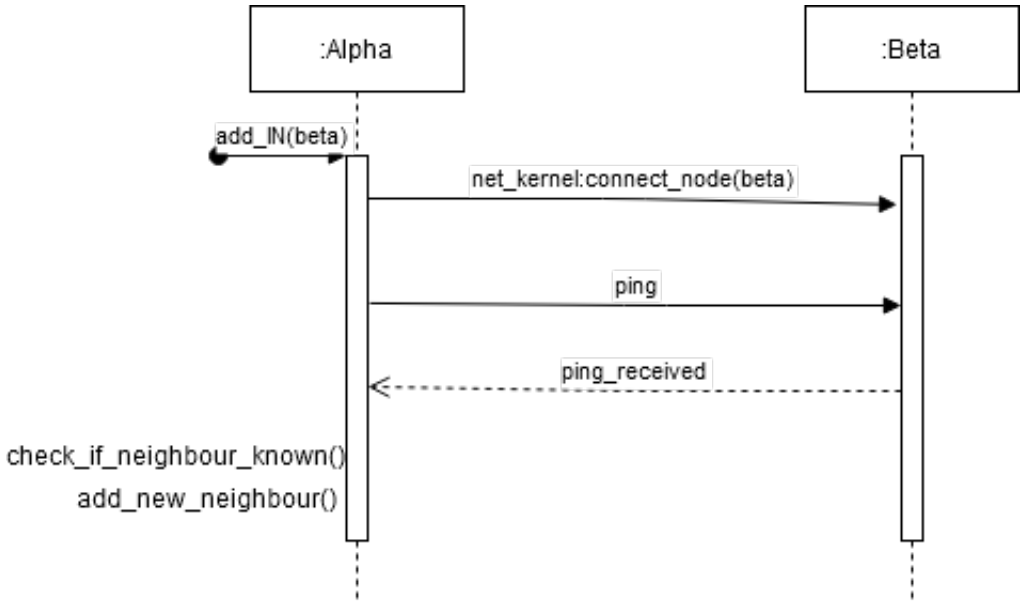
Jest to serwer, który komunikuje się z własnymi replikami poprzez protokół RIP przez różne węzły Erlanga. Może wysłać żądanie do zdalnego węzła, aby obliczyć n-tą liczbę pierwszą, a kiedy otrzyma odpowiedź od danego węzła, drukuje ją na konsoli. Podczas działania programu stale odświeżana jest tablica routingu.

5 Opis i schemat struktury zadaniowej programu

5.1 Dodawanie bezpośredniego sąsiada

Po uruchomieniu węzłów:

1. Wywołujemy funkcję API add_IN () aby połączyć dwa węzły.
2. Węzły erlanga próbują się połączyć ze sobą.
3. Alpha wysyła komunikat ping do węzła Beta, aby sprawdzić, czy naprawdę istnieje.
4. Beta odpowiada na komunikat ping.
5. Alpha sprawdza, czy węzeł Beta istnieje już na liście sąsiadów (tablicy routingu).
6. Beta jest zapisywana w tablicy routingu Alpha jako sąsiad.



5.2 Broadcasting

Gdy węzły Alpha i Beta zostaną połączone ze sobą jako sąsiedzi:

1. Okresowy zegar sygnalizuje Alpha, aby zaktualizował swoją tablicę routingu (lub możemy wywołać funkcję API update_RT).

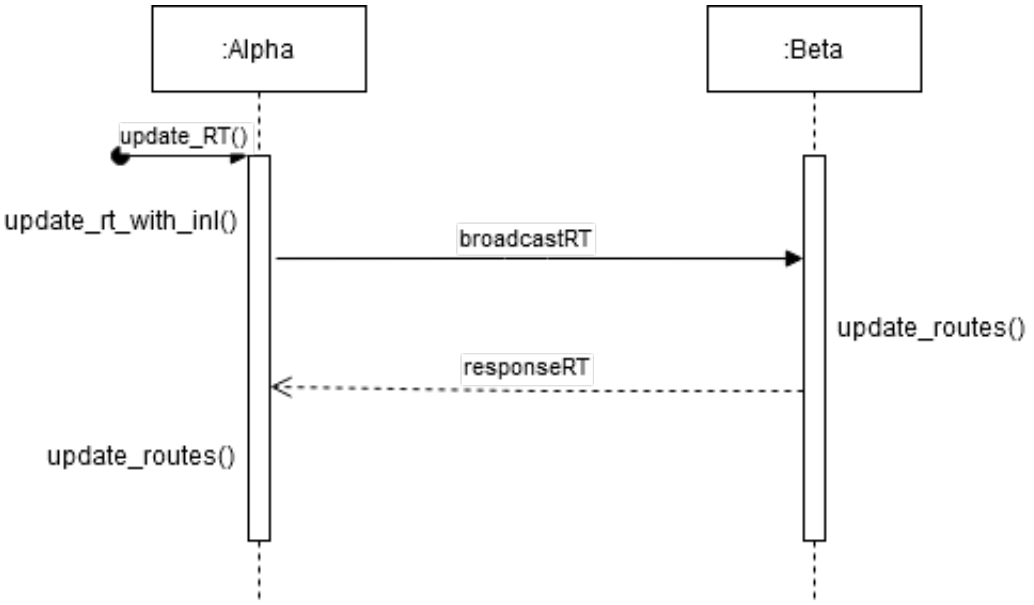
2. Alpha aktualizuje swoją własną tablicę routingu z ostatnio dodanymi bezpośrednimi sąsiadami.

3. Alpha rozgłasza swoją tablicę routingu do wszystkich swoich bezpośrednich sąsiadów.

4. Beta odbiera broadcast, aktualizuje własną tablicę routingu o otrzymaną tablicę. Dodaje nowe trasy i optymalizuje istniejące.

5. Beta odpowiada na broadcast Alpha z własną tablicą routingu.

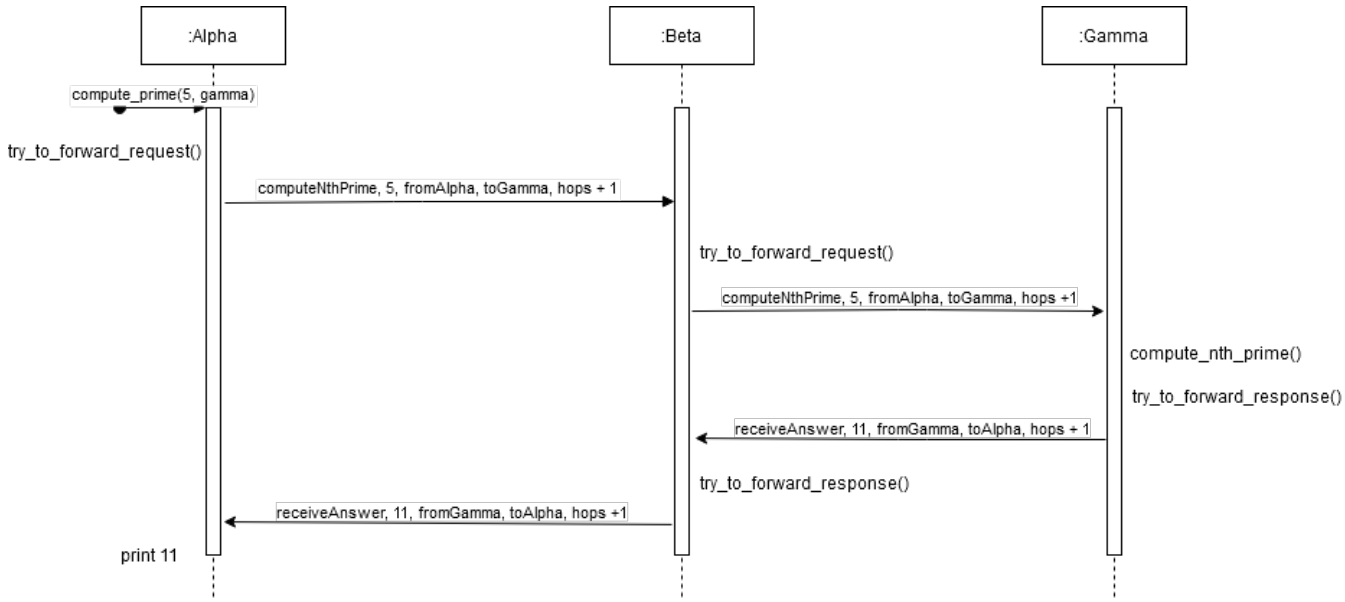
6. Alpha otrzymuje odpowiedź i aktualizuje swoją tablicę routingu za pomocą tabeli otrzymanej z węzła Beta.



6 Żądanie zdalnego obliczenia liczby pierwszej

Gdy Alpha, Beta i Gamma znają trasy w swojej topologii:

1. Alpha żąda, aby Gamma obliczyła liczby pierwsze, wywołując funkcję API compute_prime().
2. Alpha sprawdza, czy powinien przekazać żądanie (czy jest adresatem wiadomości, przeskoki nie przekraczają limitu).
3. Alpha wysyła żądanie do swojego sąsiedniego węzła Beta na trasie w kierunku Gamma. Nazwy węzłów źródłowych i docelowych są przechowywane w wiadomości. Liczba przeskoków również jest przechowywana.
4. Beta sprawdza, czy powinna przekazać żądanie.
5. Beta wysyła żądanie do Gamma.
6. Gamma jest odbiorcą żądania. Wykonuje zadanie, oblicza liczbę pierwszą.
7. Gamma sprawdza, czy powinien przekazać odpowiedź.
8. Gamma wysyła odpowiedź do węzła Beta na drodze w kierunku Alpha. Nazwy węzłów początkowych, docelowych oraz liczba przeskoków są przechowywane w odpowiedzi.
9. Beta sprawdza, czy powinna przekazać odpowiedź.
10. Beta wysyła odpowiedź do węzła Alpha.
11. Alfa jest odbiorcą odpowiedzi. Wypisuje informacje, że otrzymał zdalnie obliczoną liczbę pierwszą.



7 Informacje o stosowanych pakietach zewnętrznych (niestandardowych)

Program nie stosuje żadnych pakietów zewnętrznych. Moduł API (prime_server.erl) używa zachowania *OTP* gen_server.

8 Informacje o zastosowaniu specyficznych metod rozwiązania problemu

8.1 Algorytm obliczania liczb pierwszych

Zadaniem obliczeniowym jest znalezienie N-tej liczby pierwszej, zaczynając od 2. Liczby pierwsze są przechowywane na liście.

1. Pierwsze dwie liczby pierwsze 2 i 3 są stałe.
2. Po liczbie 3 każdy przyrost o 2 jest kandydatem. (Nie ma sensu sprawdzać liczb parzystych).

3. Dla każdego kandydata X przechodzimy przez liczby pierwsze znajdujące się już na liście. Sprawdzamy, czy każda liczba pierwsza P jest czynnikiem X.

4. Przestajemy sprawdzać liczby pierwsze P na liście, które są większe niż pierwiastek kwadratowy kandydata sqrt (X).

5. Jeśli nie znajdziemy czynników pierwszych X, X jest liczbą pierwszą. Dodajemy X do listy i przechodzimy do następnego kandydata X + 2.

6. Jeśli lista osiągnie rozmiar N, zatrzymujemy się i zwracamy N-tą liczbę pierwszą.

8.2 RIP

Zadanie mające na celu komunikację między węzłami odzwierciedlające działanie protokołu RIP.

1. Każdy proces ma pseudonim, który odpowiada nazwie węzła jego powłoki Erlanga. W ten sposób procesy wzajemnie się identyfikują.

2. Każdy proces ma maksymalnie 3 bezpośrednich sąsiadów. Procesy mogą komunikować się tylko z bezpośrednimi sąsiadami za pośrednictwem rozproszonych wiadomości Erlanga.

3. Każdy proces przechowuje tablicę routingu, zaczynając od swoich bezpośrednich sąsiadów. Zapisują, do jakiego procesu mogą dotrzeć w sieci i do którego sąsiada muszą najpierw wysłać wiadomość, aby dotrzeć do celu.

4. Każdy proces rozgłasza swoją pełną tablicę routingu. Kiedy proces Alpha otrzymuje tabelę kontaktów z Beta, integruje ją ze swoją własną tabelą. Następnie Alpha może dotrzeć do wszystkich sąsiadów Beta poprzez Betę.

5. Broadcast odbywa się okresowo, co 1 minutę, z przypadkowym opóźnieniem około 0-10 sekund. Element losowy służy zapobieganiu synchronizacji przez węzły ich transmisji, co skutkuje impulsami ruchu.

6. Za każdym razem, gdy proces otrzymuje wiadomość skierowaną do innego procesu, próbuje przekazać wiadomość do celu na podstawie własnej tablicy routingu.

7. Za każdym razem, gdy proces otrzymuje wiadomość, która jest skierowana do niego, wykonuje to, o co prosi komunikat (w tym przypadku oblicza liczby pierwsze), a następnie próbuje odpowiedzieć procesowi pochodzenia na podstawie tablicy routingu.

8. Każde przekazywanie wiadomości nazywa się przeskokiem. Wiadomości śledzą liczbę przeskoków. W przypadku osiągnięcia maksymalnego limitu wiadomość jest uznawana za utraconą. Ma to na celu uniknięcie nieskończonych pętli w przypadku błędów.

9 Krótka instrukcja obsługi

9.1 API

Interfejs przeznaczony do korzystania z programu to moduł prime_server.erl.

- **start_link(Nickname)** – Rozpoczyna działanie serwera. Nickname musi być pojedynczym słowem oraz być unikalny dla każdej instancji tego procesu w klastrze. W celu zatrzymania serwera należy go zawiesić.

- **add_IN ({Ref, Node})** – Łączy się ze zdalnym węzłem i przechowuje go jako bezpośredniego sąsiada. Jeśli jest już trzech bezpośrednich sąsiadów, odmawia przyjęcia nowych. Jednego sąsiada można dodać tylko raz. Bycie sąsiadem jest jednokierunkowe, inny proces nie zapisze tego procesu automatycznie. Ref to zarejestrowane odniesienie do procesu w innym węźle. Node to nazwa atomu zdalnego węzła.

- **add_IN (Node)** – Uproszczona wersja funkcji, w której odniesieniem do procesu jest nazwa modułu (prime_server), automatycznie rejestrowana przez OTP podczas uruchamiania.

- **compute_prime(N, DestName)** – wysyła żądanie do procesu o nazwie DestName w celu obliczenia podanej N-tej liczby pierwszej.

- **printINL(), printRT()** – drukuje listę najbliższych sąsiadów lub tablicę routingu do konsoli w celu debugowania.

- **update_RT()** – licznik czasu regularnie wyzwala aktualizację tablicy routingu. Można wywołać funckję w danym węźle w celu szybszej aktualizacji tablicy.

9.2 Instrukcje

Poniżej krótka instrukcja obsługi programu:

- Uruchom erlanga w wielu instancjach powłoki. Jeśli uruchamiasz klaster rozproszony na różnych hostach, upewnij się, że widzą się one w sieci lokalnej, a adres w nazwie każdego węzła odpowiada adresowi IP ich hosta. Np:

```
erl -name alpha@192.168.0.18 -setcookie mycookie
erl -name beta@192.168.0.19 -setcookie mycookie
erl -name gamma@192.168.0.20 -setcookie mycookie
```

- Skompiluj kod API w każdym węźle.

```
cd('~/path/to/project/src').
c(prime_server).
```

- Uruchom serwer, podając nazwę węzła jako parametr. Konieczne jest, aby nazwa węzła była poprawna, aby inne węzły mogły go zaadresować. Np:

```
prime_server:start_link('alpha@192.168.0.18').
```

- Połącz węzły i utwórz sieć (w dowolnej preferowanej topologii), określając bezpośrednich sąsiadów każdego węzła. Pamiętaj, że połączenia w tym programie są jednokierunkowe. Jeśli chcesz, aby dwa węzły widziały się nawzajem z obu kierunków, musisz dodać oba jako sąsiadów. Poniżej przykład połączenia w jednym z węzłów.

```
prime_server:add_IN('beta@192.168.0.19').
```

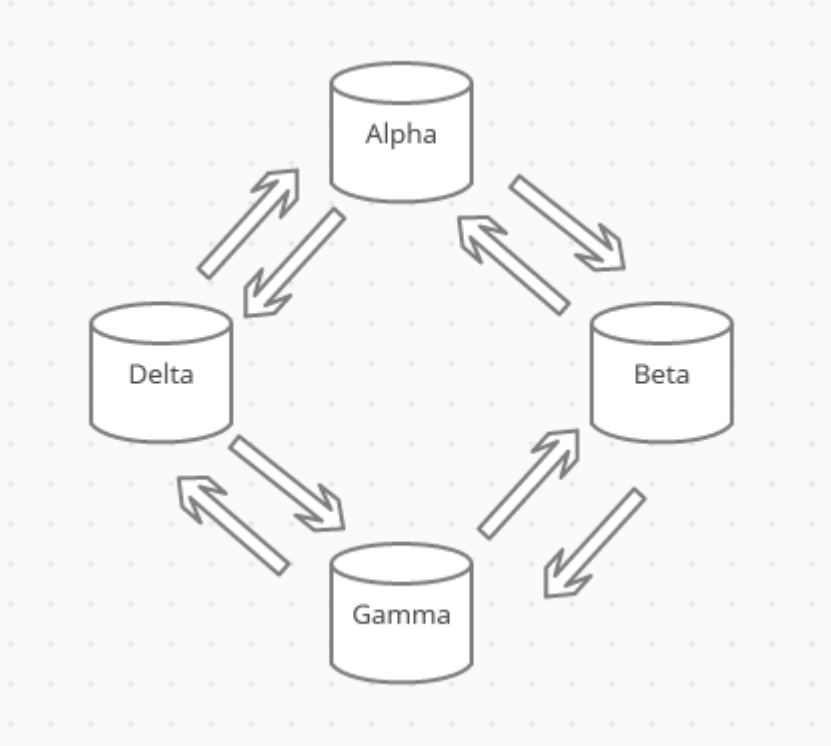
- Poczekaj, aż węzły automatycznie się odnajdą i wytyczą trasy umożliwiające zdalne połączenie. Powinno to zająć kilka minut, w zależności od liczby węzłów i topologii.

- Zażądaj obliczenia liczby pierwszej w węźle zdalnym. Jeśli twój węzeł zna trasę do zdalnego celu i odwrotnie, cel zna również trasę do twojego węzła początkowego, odpowiedź powinna zostać natychmiast odebrana i wydrukowana w konsoli. Np:

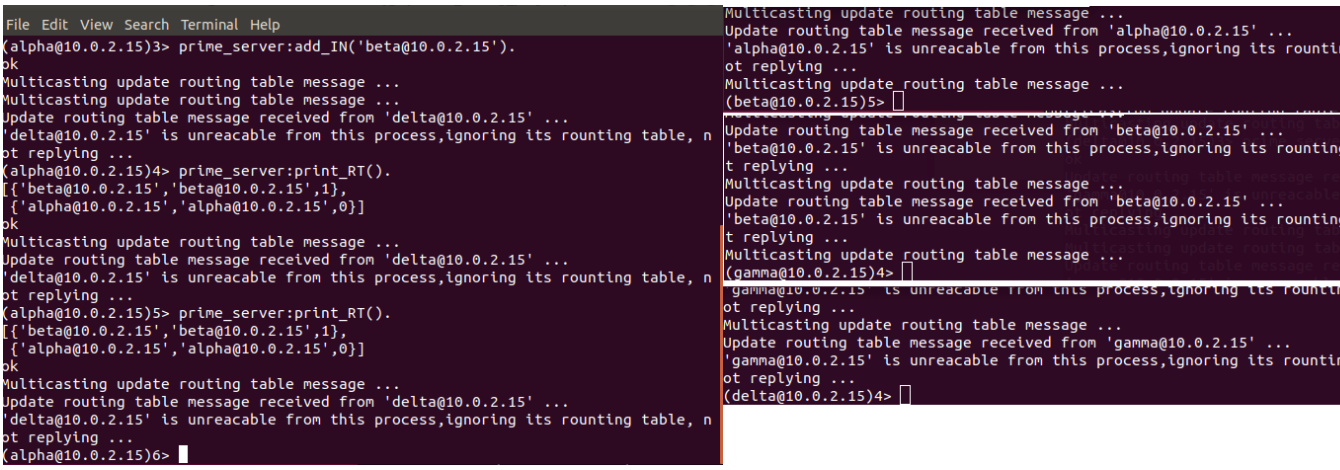
```
prime_server:compute_prime(7, 'gamma@192.168.0.20').
```

10 Testy, przykłady

W poniższym przykładzie uruchamiamy 4 węzły alpha, beta, gamma, delta i łączymy je w cykl.

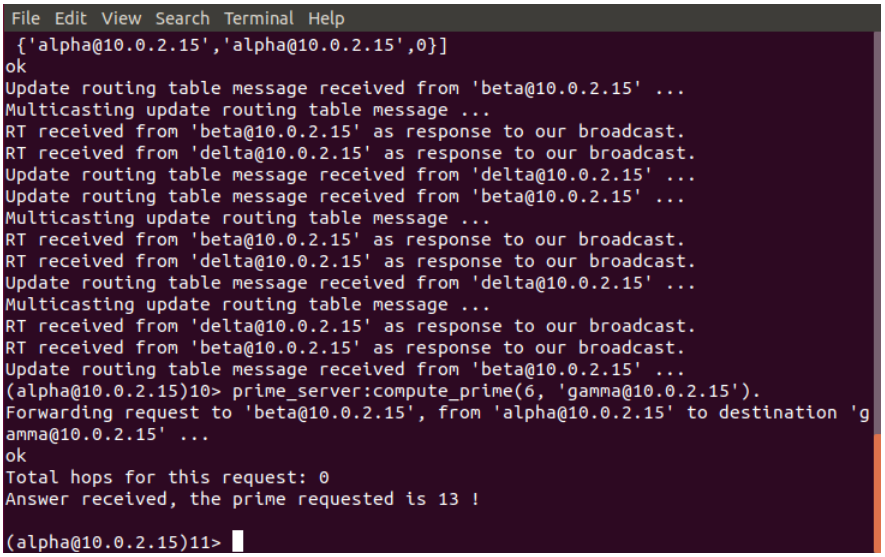


Uruchamiamy cztery węzły.

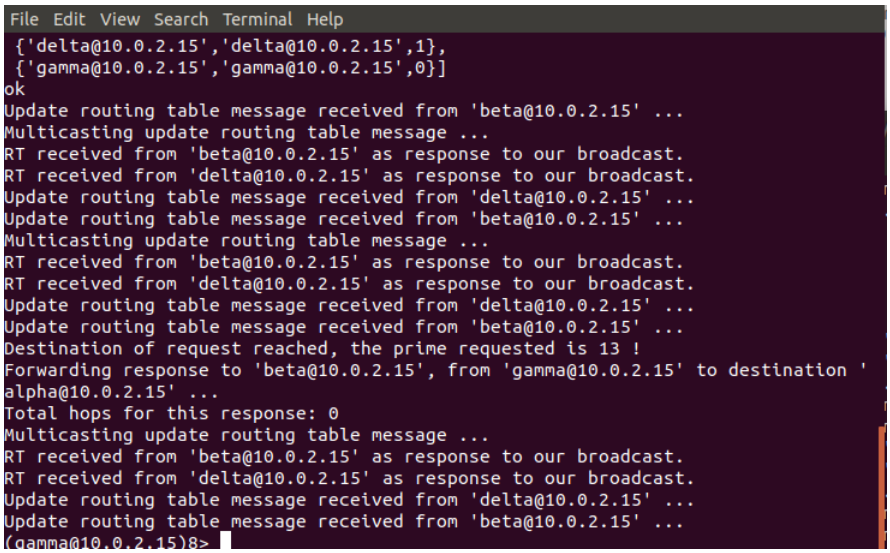


Tworzymy wszystkie połączenia określone w naszej topologii i czekamy, aż węzły się nawzajem wykryją. Możemy zobaczyć trasy każdego węzła, drukując ich tablice routingu za pomocą funkcji prime_server:print_RT()

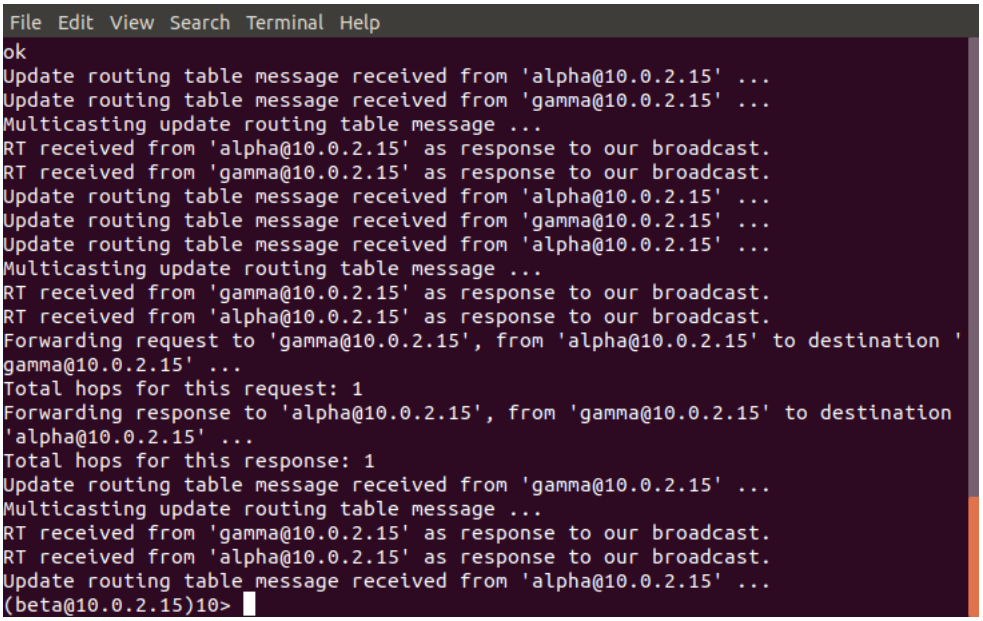
Teraz, gdy trasy są znane, możemy wysłać zdalne żądanie. Z alpha prosimy gamma, aby obliczyć 6-tą liczbę pierwszą. W węźle alpha wiadomość jest wysyłana do węzła beta. Odpowiedź wraca natychmiast. Oznacza to, że routing zadziałał, wiadomość trafiła do gamma i z powrotem.



W gamma (węzeł docelowy) wiadomość zostaje odbierana, obliczanie liczby pierwszej jest wykonywane. gamma wysyła odpowiedź na swojej drodze do pierwotnego węzła alpha przez węzeł beta.



Węzeł beta działał jako mediator zarówno dla oryginalnej wiadomości, jak i odpowiedzi, przekazując je do miejsca docelowego w oparciu o własną tablicę routingu, zliczając przeskoki.



Ten sam przykład można powtórzyć z dużo większą liczbą węzłów o innej topologii.

11 Możliwe rozszerzenia programu

Na ten moment wykrywanie sieci obejmuje tylko rozbudowę. Można by rozszerzyć program o to, że węzły wykrywają, czy inny węzeł w ich tablicy routingu umiera i nie jest już dostępny.

12 Ograniczenia programu

Następujące ograniczenia są zapisywane na stałe w dowolny sposób:

- Jeden węzeł może mieć maksymalnie 3 bezpośrednich sąsiadów.
- Węzeł jest uważany za nieosiągalny, jeśli trasa do niego jest dłuższa niż 15 przeskoków (co de facto jest zgodne ze standardem protokołu RIP)

- Domyślnie maksymalna całkowita liczba węzłów serwera obsługiwanych wynosi 16 384.

13 Inne informacje

Opierając się na protokole RIP routery podejmują następujące działania:

1. Żądają aktualnych informacji o routingu do innych routerów i na ich podstawie aktualizują tablice routingu.
2. Odpowiadają na podobne żądanie innych routerów.
3. W ściśle określonych przedziałach czasu rozsyłają informacje o swojej obecności, informując inne routery o aktualnej konfiguracji połączeń międzysieciowych.
4. W przypadku wykrycia zmian w konfiguracji sieci rozsyłają stosowną informację.

Protokół RIP wysyła komunikaty uaktualniające w określonych, stałych przedziałach czasu, na przykład co 30 s, lub w przypadku pojawienia się zmian w topologii sieci. Router po przyjęciu uaktualnionego routingu, które dotyczy zmian jednego z wejść, uaktualnia tablicę routingu (routing table), by odzwierciedlić nową trasę. Wartość miary przypisanej danej ścieżce wzrasta o jeden i jako następny skok jest wskazany nadawca. Routery RIP utrzymują do miejsca przeznaczenia tylko najlepszą trasę, to jest trasę z najmniejsza liczbą skoków. Router niezwłocznie po uaktualnieniu swojej tablicy routingu wysyła informacje o zmianie do pozostałych routerów w sieci. Są one wysyłane niezależnie do regularnie wysyłanych uaktualnień.

W celu dostosowania się do szybkich zmian topologii sieci protokół RIP wyposażono, podobnie jak i inne protokołu routingujące, w mechanizmy stabilizujące. Na przykład, by zapobiec skutkom błędnej informacji o routingu, w protokole RIP zaimplementowano mechanizmy split-horizon i hold-down. Powstaniu pętli zapobiega ograniczenie - na trasie pomiędzy źródłem a miejscem przeznaczenia - liczby skoków (do 15).