Sprawozdanie nr. 4

Wojciech Karol Rafałowski

09.06.2018, Wrocław

1 Definicje

1.1 Protokół UDP

Polega na tym, że pakiety są wysyłane bez kontroli przepływu. UDP jest używany w przypadku, gdy prędkość przepływu danych jest wyższego priorytetu niż ich stuprocentowa poprawność. Wykorzystuje się go między innymi w transmisjach na żywo, gdzie dozwolone jest porzucenie opóźnionych danych, aby szybciej otrzymać obecne dane.

1.2 Protokół TCP

Pakiety po wysłaniu są śledzone, czy dane nie zostały zgubione, uszkodzone lub zduplikowane. Wykonuje się to poprzez dostarczenie potwierdzenia odbioru do nadawcy. W przypadku gdy odbiorca otrzyma niepoprawne dane, nadawca ponownie je wysyła. TCP jest używany między innymi na stronach internetowych lub chatach, gdzie istotne jest otrzymanie danych w kolejności ich wysyłania.

2 Założenia sprawozdania

Kody źródłowe, podane na stronie dra Łukasza Krzywieckiego napisane w języku Java, które symulują zachowanie transferu danych w sieci należy przerobić tak, aby uwzględniały błędy w transmisji, takie jak: duplikowanie wysłanego pakietu, opóźnienie przesyłu oraz błąd w odbiorze pakietu. Aby uwzględnić te defekty, należy zmodyfikować programy `Z2Receiver.java` oraz `Z2Sender.java`.

3 Zasady działania projektu

Kolejne klasy symulują działanie pewnych mechanizmów w transferze pakietów w sieci.

3.1 `Z2Packet.java`

Ta klasa jest strukturą, która nosi w sobie informację o pakiecie danych wysyłanych przez pozostałe klasy. Możliwia zapisywanie i odczytywanie liczb całkowitych do tablicy bajtów.

3.2 `Z2Sender.java`

Ta klasa ma dwa główne parametry: numer portu, do którego przesyła dane otrzymane przez drugi port. Działanie obiektu tej klasy polega na nasłuchiwaniu wejscia funkcją `System.in.read()` oraz zapisywaniu otrzymanych danych do obiektu klasy `Z2Packet`. Cały pakiet jest potem wysyłany do odpowiednieg portu. Następnie wątkowi odpowiedzialnemu za to jest nadane polecenie `sleep` przez czas nadany w finalnym polu klasy.

Wątek odpowiedzialny za odbiór danych działa analogicznie, z tą różnicą, że każdą daną wywietla na standardowym wyjsciu zaraz po zapisaniu jej do obiektu klasy `Z2Packet`.

3.3 `Z2Receiver.java`

Ta klasa ma dwa główne parametry: numer portu, który nasłuchuje przesył pakietów, oraz drugi, który okresla, gdzie ma być wysłane potwierdzenie. Obiekt tej klasy jest odpowiedzialny wypisanie na standardowym wyjsciu danych otrzymanych w pierwszym porcie. Tak działa jej klasa wewnętrzna, dziedzicząca po `Thread`. Następnie zapisuje to do obiektu klasy `Z2Packet` i wysyła na drugi port.

3.4 `Z2Forwarder.java`

Ta klasa symuluje przesył pakietu. Okresla prawdopodobieństwo tego, czy dane mogą nie dojsć do celu oraz tego, czy mogą zostać zduplikowane. Jedna z klas wewnętrznych, dziedzicząca po `Thread` nasłuchuje przesył danych i każdą porcję dodaje do obiektu klasy `Packet`, jesli wylosowana dowolnie liczba jest mniejsza niż prawdopodobieństwo "zagubienia" danych lub dodaje jej kopię, jesli druga wylosowana w ten sposób liczba jest większa niż szansa zduplikowania danych. Otrzymany w ten sposób pakiet jest wysyłany do okreslonego portu.

4 Modyfikacja projektu

Aby klasy działały w ten sposób, że uwzględniają błędy transmisji, należy zmienić kod klas `Z2Sender` oraz `Receiver`.
Na wysyłanie danych nie można wpłynąć w tej klasie, ponieważ to, czy poprawnie otrzyma je obiekt nasłuchujący wpłynie klasa `Z2Forwarder`, która dopiero obsłuży wysłany pakiet. Modyfikacja tej klasy przyniesie sens tylko, jesli będzie to zmiana kodu odpowiedzialnego za odbiór, czyli wątku `ReceiverThread`. W pętli `while(true)` jest wykonywane przechwytywanie danych, zapisywanie ich do obieku klasy `Packet` i wyswietlanie ich na standardowym wyjsciu. Modyfikacja kodu polegała na tym, że Wyswietlanie wyniku odbywało się w drugiej pętli, z kolei w pierwszej następowała tylko weryfikacja danych. Jesli zostały odebrane dwa elementy o takiej samej wartosci i numerze sekwencyjnym, drugi z nich był pomijany. Ponadto, po zakończeniu transmisji elementy pakietu są sortowane według numeru sekwencyjnego. Jesli w ciągu znaków brakuje któergos, jest wysyłana informacja do portu, z którego nadeszła wiadomosć o ponownym wysłaniu danych razem z numerem, którego brak zarejestrowano.
Analogicznie wygląda poprawa kodu klasy `Z2Receiver`.
Należy wspomnieć o istotnym fakcie, że nie następuje żądanie o przerwaniu transmisji. Zostaje wyswietlony na standardowym wyjsciu komunikat o błędzie, jednak przesył trwa nadal.

Takie działania symulują protokół TCP.

5 Przykładowe uruchomienie

Po skompilowaniu projektu uruchomiono program `.exe` z parametrami określającymi numery portów. Wykorzystano te same dane, jakie zostały użyte w poleceniu, mianowicie:

```
java Z2Receiver 6002 6003 & java Z2Forwarder 6001 6002 &
java Z2Forwarder 6003 6000 & java Z2Sender 6000 6001
```

gdzie `plik.txt` jest plikiem tekstowym z danymi, które podlegają transmisji. W konsoli otrzymano informacje pokazujące niedoskonałość transferu.

6 Wnioski

- W Internecie pakiety wysyłane przez nadawcę mogą być tracone, przybywać z różnymi opóźnieniami, w zmienionej kolejności czy zduplikowane.
- Poprawna implementacja retransmisji i potwierdzenia odbioru zapewnia bezstratną komunikację mimo niepewnego łącza.
- W przypadku aplikacji działających w rzeczywistym czasie (chaty video, gry online) stosuje się protokół UDP, który pomija oczekiwanie na wszystkie pakiety, dzięki czemu działa sprawniej niż TCP.
- TCP jest przydatny wtedy, gdy największe znaczenie ma kompletność pakietów.