LABORATORIUM SIECI WIELO-USŁUGOWE

BADANIE SPRAWNOŚCI PROTOKOŁU TCP

POLITECHNIKA WARSZAWSKA

INSTYTUT TELEKOMUNIKACJI

Warszawa, 2015

Spis treści

1 Wstęp

Celem ćwiczenia jest zapoznanie się z mechanizmami protokołu TCP (*Transmission Control Protocol*) oraz zbadanie efektywności protokołu do sterowania przekazem danych przy różnych scenariuszach działania sieci telekomunikacyjnej.

2 Protokół TCP

Protokół TCP jest protokołem warstwy transportowej, wymagającym zestawienia połączenia. Bloki danych przekazywane w ramach protokołu TCP nazywamy segmentami. Dzięki zastosowaniu specjalnej metody sterowania (*flow control*) przekazywaniem segmentów, protokół TCP posiada zdolność dostosowania szybkości wysyłania danych do sieci zależnie od warunków ruchowych sieci i stanu odbiornika.

Protokół TCP umożliwia dupleksową wymianę segmentów pomiędzy stroną nadającą (nadajnikiem) i stroną odbierającą (odbiornikiem). Dla zapewnienia przekazu segmentów bez błędu, TCP działa zgodnie z systemem ze sprzężeniem zwrotnym decyzji (*closed loop*). Oznacza to, że odbiornik potwierdza prawidłowy odbiór segmentów za pomocą potwierdzeń (tzw. ACK – *acknowledgement*) wysyłanych do nadajnika. Wiadomość potwierdzająca jest przenoszona w nagłówku segmentu TCP, dzięki czemu możliwe jest przesyłanie potwierdzeń wraz z danymi wysyłanymi do strony przeciwnej (mechanizm „*piggybacking*"). Jednakże dodatkowe opóźnienia, wynikające z obecności mechanizmu sprzężenia zwrotnego, powodują, iż TCP nie nadaje się do obsługi aplikacji wymagających transmisji w czasie rzeczywistym (np. VoIP). Typowymi aplikacjami korzystającymi z protokołu TCP są: WWW, FTP, poczta elektroniczna, telnet itp.

2.1 Format segmentu TCP

Podstawową jednostką danych protokołu TCP jest segment, którego budowa została przedstawiona na rysunku 1. Znaczenie poszczególnych pól jest następujące:

- Numer portu nadawcy – 16-bitowy numer portu TCP identyfikujący aplikację wysyłającą dane;
- Numer portu odbiorcy – 16-bitowy numer portu TCP identyfikujący aplikację po stronie odbiorczej;
- Numer sekwencyjny – odnosi się do pierwszego bajtu w polu danych aktualnie wysyłanego segmentu i wskazuje pozycję tego bajtu w strumieniu danych generowanych przez aplikację (protokół TCP jest zorientowany strumieniowo, co oznacza, iż kontrola ilości przesyłanych danych opiera się na numeracji bajtów, a nie np. segmentów);
- Potwierdzenie – określa numer pierwszego oczekiwanego bajtu po stronie odbiorczej, potwierdzając tym samym poprawny odbiór przez stronę odbiorczą wszystkich bajtów o numerze mniejszym o jeden od numeru przenoszonego w polu potwierdzenia;
- Długość nagłówka - podaje długość nagłówka (włącznie z polem Opcje) w postaci wielokrotności 32 bitów;
- Rezerwa - pole zarezerwowane do przyszłych zastosowań;
- Pole kontrolne - zawiera sześć bitów używanych jako flagi, których znaczenie jest następujące:
  - URG (*Urgent Pointer*) - aktywne pole wiadomości pilnej;
  - ACK (*Acknowledgement Number*) - aktywne pole potwierdzenia;
  - PSH (*Push*) - żądanie natychmiastowego wysłania danych zgromadzonych w nadajniku TCP (nawet jeśli jest ich zbyt mało do stworzenia „długiego" segmentu);
  - RST (*Reset*) – informuje o konieczności zresetowania połączenia;

- SYN (*Synchronize*) - synchronizacja numerów sekwencyjnych podczas nawiązywania połączenia; pole Numer Sekwencyjny zawiera wówczas tzw. numer początkowy ISN (*Initial Sequence Number*), co oznacza, iż pierwszy bajt transmitowanych danych będzie miał numer ISN+1;
- FIN (*Finished*) – zakończenie przekazu danych;

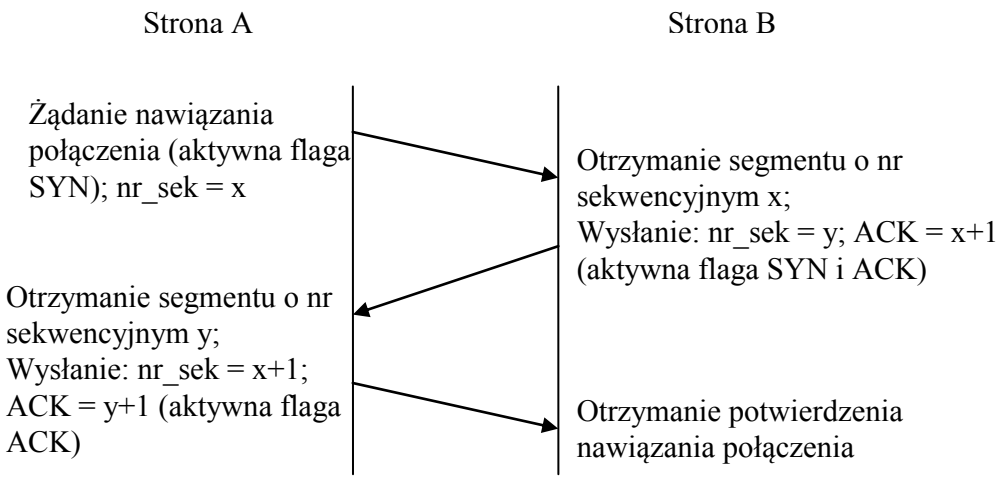
- Okno - maksymalna liczba bajtów, która może być przyjęta przez odbiorcę, licząc od numeru podanego w polu potwierdzenia; 16-bitowa długość tego pola ogranicza standardowy maksymalny rozmiar okna do 65535 bajtów;
- Wskaźnik wiadomości pilnej - wskazuje ostatni bajt pilnych danych (wymagających „specjalnego" odbioru) w przesyłanym segmencie; pole to jest ważne, jeśli ustawiona jest flaga URG;
- Opcje - pole to ma zmienną długość i jest stosowane do przesyłania dodatkowych informacji; najczęściej stosowanymi opcjami są:
  - MSS (*Maximum Segment Size*) – umożliwia stacjom nawiązującym połączenie TCP negocjację maksymalnej długości segmentu, wyrażonej w bajtach; opcja ta może być wysyłana jedynie w segmentach inicjujących połączenie (z aktywną flagą SYN);
  - opcja przeskalowania rozmiaru okna (W*indow Scale Option*);
  - opcja ustawiania znaczników czasu (*Timestamp Option*);
- *Padding* - pole dopełnienia, dzięki któremu długość nagłówka TCP jest zawsze równa wielokrotności 4 bajtów;
- Dane – pole danych użytkownika.

Rysunek 1. Segment danych protokołu TCP

2.2 Ustanowienie połączenia TCP i rozłączenie

Protokół TCP dostarcza warstwie aplikacji usługi zorientowanej połączeniowo, dlatego też, zanim nastąpi transmisja danych, konieczne jest zestawienie połączenia pomiędzy dwoma systemami końcowymi. Celem fazy nawiązania połączenia jest zagwarantowanie, iż obydwie strony są gotowe do przekazu informacji – nastąpiła wymiana numerów portów używanych przez stronę przeciwną, numerów sekwencyjnych wykorzystywanych do kontroli kolejności przesyłanych

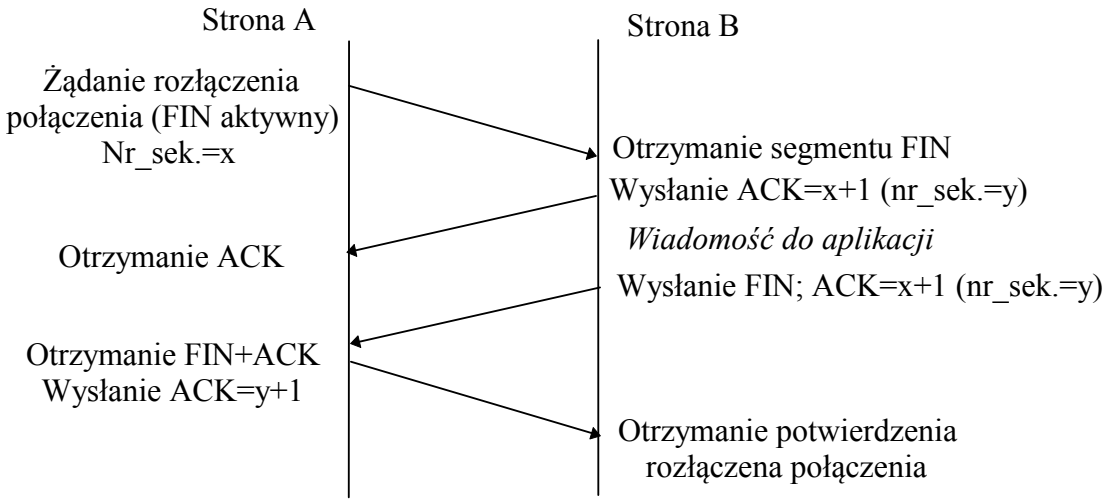
danych, jak również (opcjonalnie) uzgodnienie maksymalnego rozmiaru przesyłanych segmentów. Przy nawiązywaniu połączenia protokół TCP korzysta z mechanizmu trzyetapowego porozumienia (*three-way handshake*).



Rysunek 2. Proces nawiązywania połączenia protokołu TCP

Proces nawiązywania połączenia przebiega następująco (Rysunek 1): strona A informuje stronę B o chęci nawiązania połączenia poprzez wysłanie segmentu z aktywną flagą SYN, zawierającego początkowy numer sekwencyjny ISN równy *x* (losowa liczba z przedziału od 0 do 2^{32}-1). W odpowiedzi strona B wysyła segment SYN zawierający jej własny ISN, równy w tym przypadku *y*, oraz potwierdzenie dla strony A (ACK=x+1). Strona A potwierdza poprawne odebranie segmentu SYN wysyłając do strony B segment z aktywną flagą ACK i polem potwierdzenia ustawionym na wartość y+1. Odebranie przez stronę B segmentu potwierdzającego oznacza, iż obie strony posiadają uzgodnione numery sekwencyjne i są gotowe do wymiany danych.

Proces rozłączenia połączenia przebiega w podobny sposób i opiera się na zmodyfikowanym mechanizmie trzyetapowego porozumienia. Strona B, po otrzymaniu segmentu z aktywną flagą FIN, wysyła do strony A potwierdzenie (ACK=x+1) poprawnego odebrania segmentu FIN, informując jednocześnie aplikację o żądaniu zakończeniu połączenia. Po odebraniu informacji zwrotnej od aplikacji, stacja B wysyła segment FIN do stacji A. W momencie otrzymania segmentu potwierdzającego od stacji A połączenie zostaje zakończone.



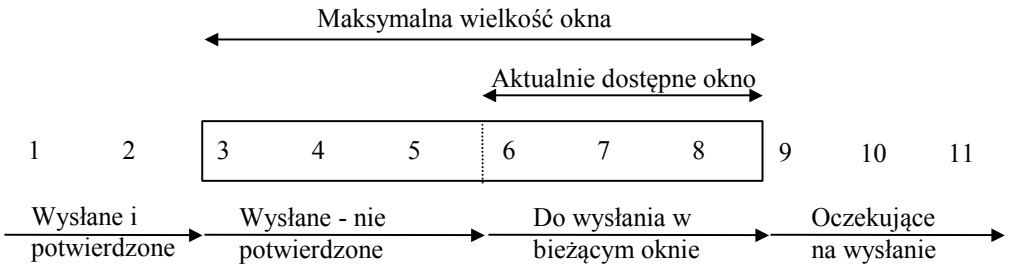
Rysunek 3. Proces rozłączenia połączenia protokołu TCP

2.3 Mechanizmy sterowania przekazem danych

TCP, dzięki specjalnej metodzie sterowania przekazywaniem segmentów (*flow control*), umożliwia stronie nadawczej dostosowanie szybkości transmitowania danych do warunków panujących w sieci oraz aktualnego stanu obciążenia odbiornika. Wykorzystywane w tym celu mechanizmy zostały opisane poniżej.

2.3.1 Mechanizm okna

Podczas transmisji strumienia danych protokół TCP wykorzystuje mechanizm przesuwającego się okna (*sliding window*), którego zasada działania przedstawiona jest na Rysunek 4. Idea przesuwającego się okna polega na kontrolowaniu ilości danych (segmentów) przebywających w danej chwili w sieci (czyli takich, które zostały już wysłane, ale jeszcze nie potwierdzone przez stronę odbiorczą). Jeżeli ilość ta osiągnie wartość równą rozmiarowi okna, nadajnik wstrzymuje transmisję kolejnych segmentów do momentu odebrania potwierdzeń od strony odbiorczej. Po otrzymaniu potwierdzeń, nadajnik wyznacza aktualnie dostępne okno (czyli ilość danych, jakie mogą zostać w danej chwili wysłane) jako różnicę pomiędzy maksymalnym rozmiarem okna a ilością danych już wysłanych, ale jeszcze nie potwierdzonych. Maksymalny rozmiar okna odpowiada maksymalnej ilości segmentów, które mogą zostać wysłane do sieci bez potwierdzenia.



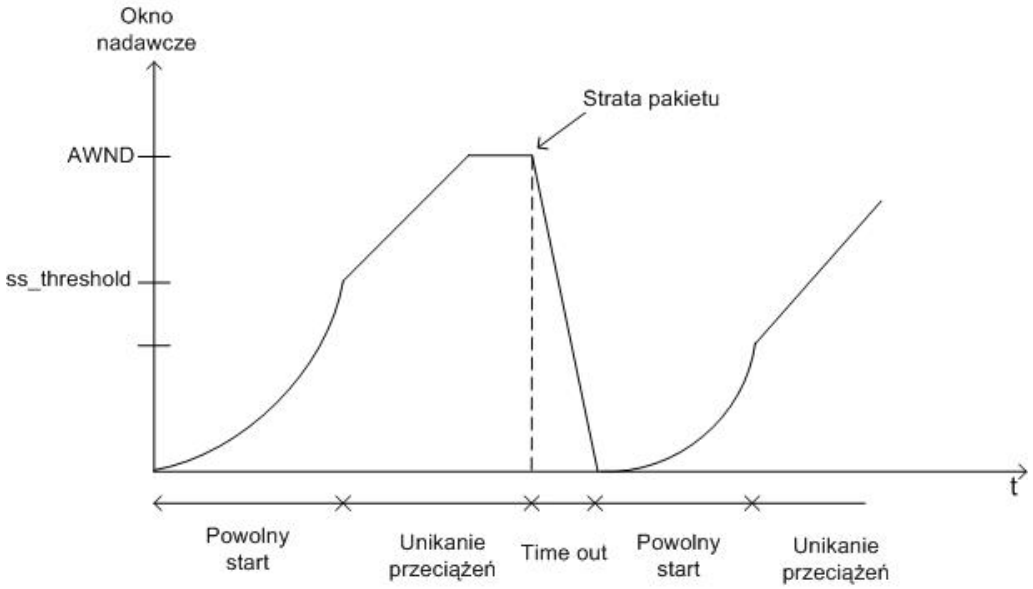
Rysunek 4. Mechanizm przesuwającego się okna

2.3.2 Powolny start i unikanie przeciążenia

W trakcie transmisji danych można wyróżnić następujące fazy pracy protokołu TCP: fazę powolnego startu (*slow start*) oraz unikania przeciążenia (*congestion avoidance*). Wymagają one utrzymywania dla każdego połączenia dwóch zmiennych: rozmiar okna przeciążenia - *cwnd* (*congestion window*) oraz próg fazy unikania przeciążenia - *ss_threshold*.

Po nawiązaniu połączenie nadajnik TCP znajduje się w fazie **wolnego startu**. Wartość zmiennej *ss_threshold,* stanowiąca granicę pomiędzy rozpatrywanymi fazami pracy protokołu, ustawiona jest wówczas na maksymalny rozmiar okna, wynoszący 65535 bajtów. Wartość *cwnd* (również wyrażona w bajtach), początkowo równa 1*MSS (*Maximum Segment Size*), jest zwiększana o jeden MSS po odebraniu każdego poprawnego potwierdzenia. Aby ograniczyć szybkość narastania bieżącego okna przeciążenia, po osiągnięciu wartości *ss_threshold* następuje spowolnienie zwiększania jego wartości i protokół przechodzi do fazy **unikania przeciążenia**. W tej fazie rozmiar okna *cwnd* jest zwiększany o jeden MSS dopiero po potwierdzeniu całego okna, czyli po czasie RTT.

Rysunek 5 przedstawia przykładowy przebieg zmian rozmiaru okna nadawczego w czasie, z zaznaczeniem przedstawionych powyżej dwóch faz działania protokołu TCP.



Rysunek 5. Fazy działania protokołu TCP

Po każdorazowym wysłaniu segmentu danych, strona nadawcza ustawia licznik czasu retransmisji (*timeout*). Jeśli w czasie wyznaczonym przez wartość tego licznika nie zostanie odebrane potwierdzenie, nadajnik zakłada, iż wysłane segmenty zostały stracone (w przeciążenia sieci bądź błędów transmisji). Podejmowane jest wówczas następujące działanie:

- niepotwierdzone segmenty zostają przesłane powtórnie (retransmisja typu *Go-Back-N*),
- rozmiar okna przeciążenia, *cwnd*, jest zmniejszana do jednego segmentu,
- zmienna *ss_threshold* jest redukowana o połowę,
- licznik czasu retransmisji, *timeout*, jest podwajany.

Zastosowanie algorytmów powolnego startu i unikania przeciążenia ma na celu wykrycie i reakcję protokołu na stan przeciążenia sieci. Należy pamiętać, że protokół TCP realizuje także funkcję sterowania przepływem, która ma za zadanie niedopuszczenie do zalania odbiornika wiadomościami z nadajnika. W tym celu odbiornik przesyła do nadajnika informację

o maksymalnej ilości danych, jaką może od niego przyjąć – jest to tzw. okno rozgłaszane (*advertised window size - awnd*). Nadajnik wysyłając dane do sieci bierze pod uwagę szerokość okna przeciążenia *cwnd* (czyli stan sieci) oraz okna rozgłaszanego *awnd* (czyli stan odbiornika). Rozmiar okna nadawczego wyznaczany jest jako:

$$okno\ nadawcze = min\ \{cwnd\ ,\ awnd\} \tag{1}$$

2.3.3 Pomiar czasu obiegu RTT oraz wyznaczanie licznika czasu retransmisji.

Czas pomiędzy nadaniem pierwszego bitu wysłanego segmentu, a odebraniem odpowiadającego mu potwierdzenia, określany jest jako tzw. czas obiegu RTT (*Round Trip Time*). Próbki czasu RTT mierzone są w momentach odebrania kolejnych potwierdzeń. Wartość średniego czasu RTT jest oszacowywana jako średnia krocząca z próbek uzyskanych z kolejnych pomiarów:

$$RTT(n) = \alpha * RTT(n-1) + (1-\alpha) * M \tag{2}$$

gdzie:

M – wartość ostatniej zmierzonej próbki RTT,
$RTT(n-1)$ - wartość oszacowania średniego RTT z poprzedniego kroku,
α - współczynnik o wartościach z zakresu $0<\alpha<1$ (wartość zalecana, $\alpha=0.9$)

Na podstawie oszacowanej wartości średniego czasu obiegu RTT obliczana jest wartość licznika czasu retransmisji, *timeout*. Prawidłowe określenie tego czasu ma duże znaczenie dla sprawnego działania protokołu. Zbyt mały czas *timeout* może skutkować niepotrzebnym powtarzaniem niektórych segmentów, podczas gdy ustawienie zbyt dużej wartości powoduje, że w momencie straty segmentu nadajnik zbyt długo oczekuje bezczynnie na możliwość retransmisji. Znane są dwie metody obliczania czasu *timeout*.

Metoda 1.
Na podstawie obliczonego czasu RTT, wartość licznika czasu retransmisji wyznacza się zgodnie z następującym wzorem:

$$timeout(n) = RTT(n) * \beta \tag{3}$$

gdzie β jest współczynnikiem o rekomendowanej wartości 2.

Metoda 2.
Według nowszych zaleceń, przy obliczaniu wartości licznika czasu retransmisji należy dodatkowo wziąć pod uwagę stopień zmienności mierzonych próbek RTT, czyli ich odchylenie w stosunku do oszacowania średniego RTT. W każdym kroku działania algorytmu (tj. po każdym obiegu pętli RTT), oprócz obliczenia wartości oszacowania RTT zgodnie ze wzorem (2), wykonuje się następujące obliczenia:

$$\begin{aligned} e(n) &= |\ RTT(n) - M\ | \\ m(n) &= \delta * m(n-1) + (1-\delta) * e(n) \\ timeout(n) &= RTT(n) + b * m(n) \end{aligned} \tag{4}$$

gdzie :

$e(n)$ – odchylenie n-tej próbki od oszacowanego średniego RTT
$m(n)$ – uśredniona wartość $e(n)$ z n próbek

δ – współczynnik o wartościach z zakresu $0<\delta<1$ (zazwyczaj $\delta=0.75$)
b – współczynnik, dla którego zwykle przyjmuje się wartość 4

Bardzo istotnym zagadnieniem mającym wpływ na efektywność działania protokołu TCP jest tzw. rozdzielczość zegara. W typowych implementacjach przyjmuje się wartość rozdzielczości zegara w granicach 300-500ms.

Obliczanie czasu *timeout* przy pomocy zależności przedstawionych powyżej napotyka na trudności w przypadku retransmitowanych segmentów. Wynika to z faktu, iż TCP nie jest w stanie odróżnić ACK odnoszących się do segmentów z identycznymi numerami sekwencyjnymi (nadajnik nie wie, czy dane potwierdzenie dotyczy pakietu pierwotnego, czy retransmitowanego). W celu rozwiązania tego problemu stosuje się algorytm Karna, który mówi, iż przy wyznaczaniu czasu retransmisji nie należy brać pod uwagę próbek RTT z retransmitowanych segmentów.

2.3.4 Opóźnione potwierdzenia

Typowo, strona odbiorcza wysyła potwierdzenie po odebraniu każdego kolejnego segmentu danych. Aby nie dopuścić do przesyłania przez sieć pakietów zawierających jedynie potwierdzenia i dzięki temu zwiększyć efektywność transmisji danych, często stosuje się tzw. mechanizm opóźnionego potwierdzenia. Polega on na tym, że wysłanie potwierdzenia jest wstrzymywane do momentu, kiedy TCP otrzyma z wyższych warstw dane do przesłania w przeciwnym kierunku, do których informacja związana z potwierdzeniem mogłaby być dołączona (tzw. *piggybacking*). Opóźnienie to zwykle nie może przekroczyć wartości granicznej, wynoszącej 200 lub 500ms.

2.3.5 Algorytm Nagla

Algorytm *Nagla* pozwala uniknąć niekorzystnego zjawiska wysyłania bardzo krótkich segmentów, np. zawierających tylko jeden znak. Polega on na tym, że pojedyncze bajty napływające z wyższych warstw są buforowane i wysyłane później w jednym segmencie. Opisany mechanizm jest szczególnie efektywny w sieciach WAN, gdzie czasy od momentu wysłania segmentu do odebrania potwierdzenia są długie. Powyższy algorytm na żądanie aplikacji może być wyłączony.

2.3.6 Algorytm szybkiej retransmisji i szybkiego odtwarzania

Algorytm **szybkiej retransmisji** (*fast retransmit)* umożliwia nadawcy wcześniejszą retransmisję zgubionego segmentu, bez oczekiwania na upłyniecie licznika czasu retransmisji. Algorytm ten opiera się na założeniu, iż otrzymanie przez nadawcę co najmniej trzech potwierdzeń dotyczących tego samego segmentu (tzw. zdublowane potwierdzenia) jest oznaką straty jednego lub kilku segmentów. W takim przypadku nadawca retransmituje stracony segment niezależnie od stanu licznika retransmisji *timeout*.
Po retransmisji zgubionego segmentu wartość parametru *ss_threshold* ustawiana jest zgodnie z równaniem:

$$ss\_threshold = max \{ FlightSize/2, 2*MSS \} \quad (5)$$

gdzie:

FlightSize –wartość *cwnd*, pomniejszona o liczbę potwierdzonych do tej pory segmentów
MSS – maksymalny rozmiar segmentu

Algorytm **szybkiego odtwarzania** (*fast recovery)* jest rozszerzeniem algorytmu *fast retransmit*. Algorytm *fast recovery* steruje wysyłaniem pakietów od momentu retransmisji zgubionego pakietu (zgodnie z algorytmem *fast retransmit*), aż do chwili, gdy nie będą przychodziły zdublowane komunikaty ACK. Podstawowym celem zastosowania algorytmu szybkiego odtwarzania jest skrócenie czasu koniecznego na retransmisję straconych pakietów a po ich retransmisji przejście do fazy unikania przeciążeń. Stracone segmenty są retransmitowane z większą szybkością gdyż w fazie tej nadawca po początkowym zmniejszeniu okna przeciążenia może je następnie zwiększać z każdym odebranym potwierdzeniem (patrz rysunek nr 7).

Szczegóły działania algorytmu *fast recovery* (wraz z poprzedzająca go fazą *fast retransmit*) są następujące:

1. W momencie otrzymania trzech identycznych ACK, nadawca wysyła brakujący segment i ustawia następujące wartości dla parametrów TCP:

$$ss\_threshold = max\ \{ FlightSize/2,\ 2*MSS\} \qquad (6)$$
$$cwnd = ss\_threshold + 3*MSS$$

2. Po odebraniu każdego zdublowanym ACK nadawca zwiększa wartość zmiennej *cwnd* o jeden *MSS*. Pozwala to na „sztuczne" zwiększenie wartości okna nadawczego i nadanie nowych segmentów (jeżeli takie oczekują).
3. W momencie otrzymania oczekiwanych (nie zdublowanych) potwierdzeń, algorytm *fast recovery* kończy swoje działanie. Transmisja przechodzi do fazy *congestion avoidance,* z oknem *cwnd* ustawionym na wartość parametru *ss_threshold*.

Podsumowując, dodanie do protokołu TCP algorytmu *fast retransmit* pozwala retransmitować zagubiony segment natychmiast po otrzymaniu zduplikowanych potwierdzeń, czyli wcześniej, niż wynikałoby to z czasu określonego przez parametr *timeout*. Natomiast wprowadzenie algorytmu *fast recovery* powoduje, iż po stracie pakietu TCP szybciej retransmituje stracone segmenty i nie rozpoczyna ponownej transmisji od fazy *slow start*, ale przechodzi od razu do fazy *congestion avoidance*.

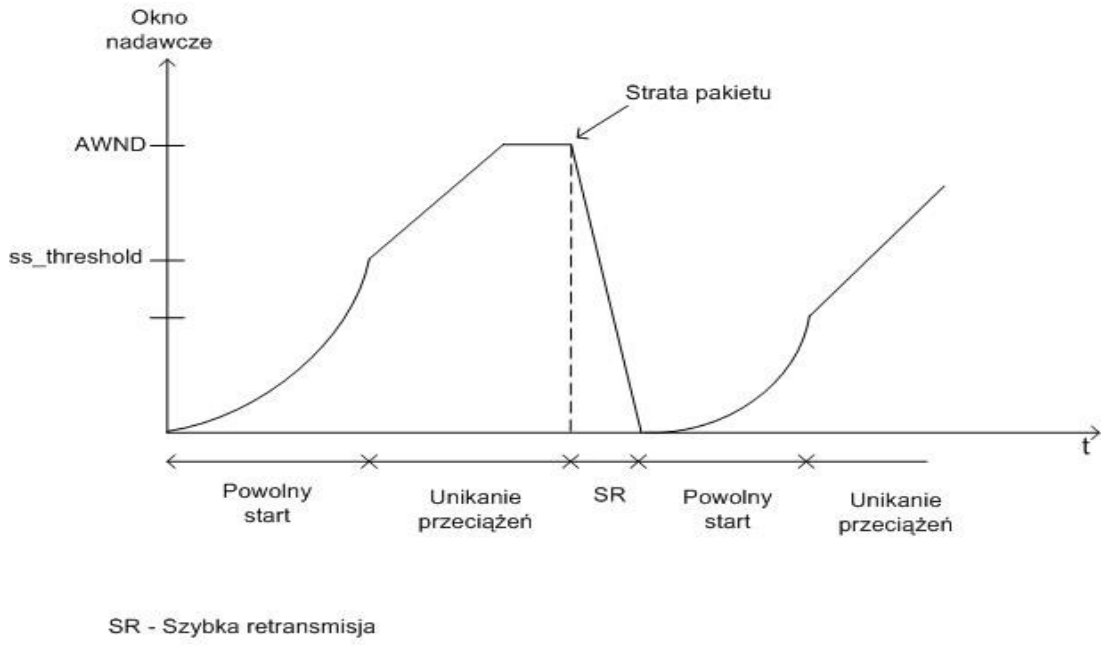
3 Przegląd implementacji TCP

Od momentu powstania protokół TCP podlega ciągłym modyfikacjom, których zadaniem jest poprawa efektywności jego działania. Efektem tego są kolejne implementacje protokołu, z których trzy opisane zostały poniżej.

3.1 TCP Tahoe

TCP Tahoe 4.3 BSD był pierwszą implementacją TCP, w której użyty został algorytm *fast retransmit*. TCP Tahoe może znajdować się w jednej z trzech faz (Rysunek 6): *slow-start, congestion avoidance, fast retransmit.* Pierwszą fazą po nawiązaniu połączenia jest *slow-start*. Po osiągnięciu przez okno nadawcze wartości *ss_threshold*, połączenie przechodzi do fazy *congestion avoidance*. Reakcja TCP na stratę segmentu zależy od tego, w jaki sposób ta strata została wykryta:

- w przypadku upłynięcia licznika czasu retransmisji, połączenie przechodzi do fazy *slow-start*;
- w przypadku straty wykrytej poprzez otrzymanie trzech kolejnych zduplikowanych ACK, TCP Tahoe realizuje algorytm *fast retransmit*, a następnie przechodzi do fazy *slow-start.*

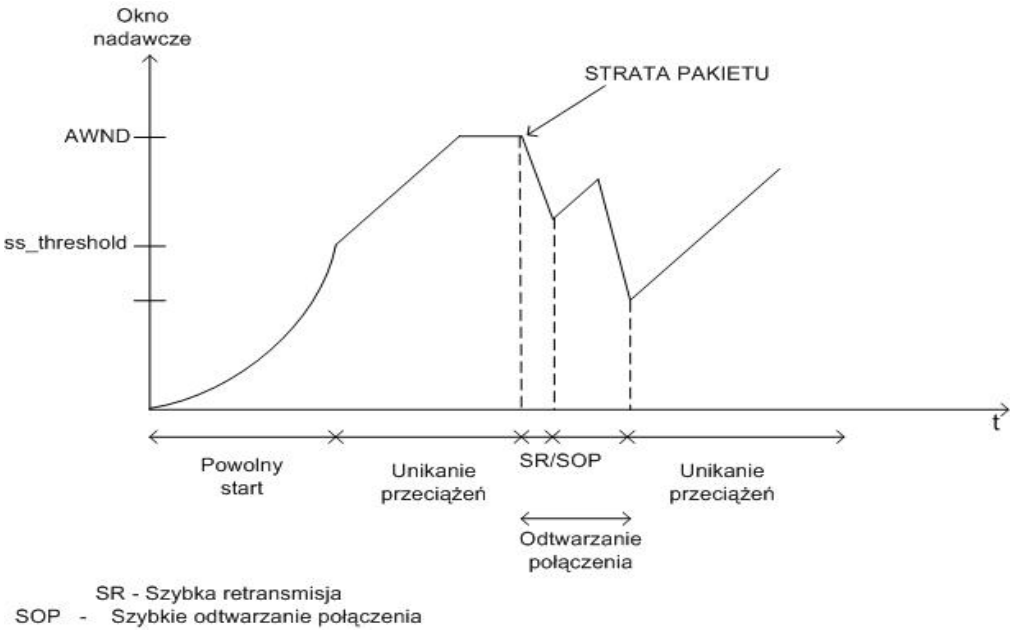


Rysunek 6. Fazy działania TCP Tahoe

3.2 TCP Reno

Implementacja TCP Reno rozszerza funkcjonalność TCP Tahoe poprzez dodanie algorytmu szybkiego odtwarzania (*fast recovery*). W tym przypadku, sposób reakcji na stratę segmentu jest następujący:

- jeżeli nadajnik wykryje stratę segmentu w wyniku upłynięcia czasu określonego parametrem *timeout*, następuje powrót do fazy *slow-start*, podobnie jak w TCP Tahoe;
- w przypadku straty wykrytej poprzez otrzymanie trzech kolejnych zduplikowanych ACK, TCP Reno najpierw realizuje algorytm *fast retransmit*, a po nim *fast recovery*. W momencie zakończenia algorytmu *fast recovery*, TCP Reno kontynuuje nadawanie pakietów w fazie *congestion avoidance* (Rysunek 7).



Rysunek 7. Fazy działania TCP Reno

Podsumowując, podstawowa różnica pomiędzy implementacjami Tahoe i Reno ujawnia się w sposobie reakcji na stratę pakietu i polega na zastosowaniu w Reno algorytmu *fast recovery*, co pozwala przejść od razu do fazy *congestion avoidance*, z pominięciem fazy *slow start*. Wspólną cechą implementacji Tahoe i Reno jest zastosowanie mechanizmu *fast retransmit* oraz retransmisji typu *Go-Back-N*.

3.3 TCP Sack (Selective Acknowledgement)

TCP Sack stanowi rozszerzenie wersji TCP Reno o opcję selektywnej retransmisji straconych pakietów. Jeżeli używanie tej opcji zostanie uzgodnione przez obie strony podczas fazy zestawiania połączenia, to w przypadku straty pakietu odbiornik informuje nadawcę, które pakiety zostały poprawnie dostarczone a których brakuje. Dzięki temu nadawca retransmituje jedynie te pakiety, które nie dotarły do odbiorcy. Implementacja ta jest wykorzystywana ją m.in. przez systemy operacyjne (np. Windows 2000 i XP, Solaris 8 i 9, Linux - jądro 2.4).

4 Parametry określające sprawność przekazu danych

Goodput – szybkość bitowa ruchu przenoszonego przez sieć, mierzona na poziomie aplikacji. Praktyczne wyznaczenie wartości parametru *goodput* polega na zmierzeniu ilości danych odebranych przez odbiornik na poziomie aplikacji w zadanym przedziale czasu. Jednostką *goodputu* jest [bit/s].

Throughput – szybkość bitowa ruchu wysyłanego do sieci przez nadajnik, mierzona na poziomie warstwy sieci. Praktyczne wyznaczenie parametru *throughput* polega na zmierzeniu ilości danych wysłanych przez nadajnik (na poziomie pakietów IP) w zadanym przedziale czasu. Jednostką *throughputu* jest [bit/s].

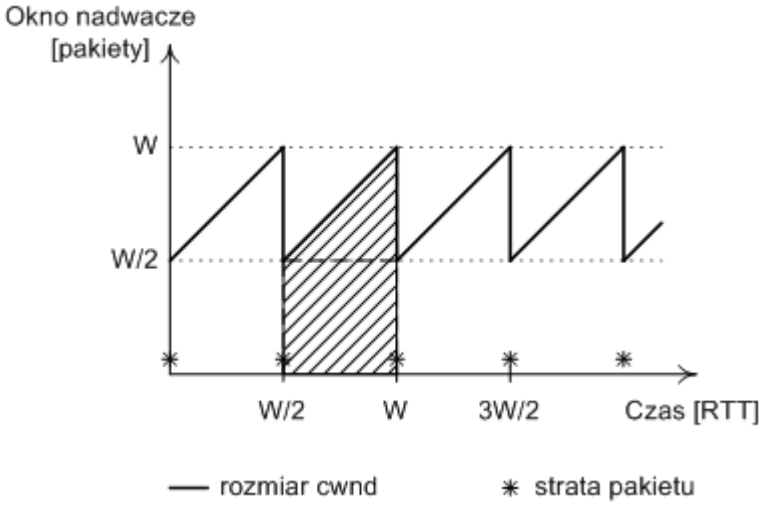
Z powyższych definicji wynikają następujące różnice pomiędzy *throughputem* a *goodputem*:
- do *throughputu* wliczane są pakiety retransmitowane, podczas gdy do *goodputu* nie

- do *throughputu* wliczana jest ilość informacji przenoszona w nagłówkach, podczas gdy do *goodputu* nie.

4.1 Uproszczony model umożliwiający wyznaczenie throughputu

W pracy [5] przedstawiony został uproszczony model mający na celu określenie szybkości bitowej połączenia TCP pracującego w fazie unikania przeciążenia oraz korzystającego z algorytmu *fast retransmit* i *fast recovery* (np. TCP Sack).
Równanie pozwalające wyznaczyć *throughput* uzyskano poprzez estymację średniego rozmiaru okna nadawczego, przy czym, w celu uproszczenia obliczeń, przyjęte zostały następujące założenia:

- RTT ma stałą wartość, wynikającą jedynie z czasów propagacji (zakładamy niewielkie obciążenie sieci, zatem czas kolejkowania pakietów w węzłach sieci jest pomijalnie mały);
- redukcja okna nadawczego następuje jedynie w wyniku odbierania przez nadajnik zdublowanych potwierdzeń ACK (wówczas rozmiar okno nadawcze zmniejszany jest o połowę); nie występują natomiast zdarzenia związane z przekroczeniem wartości wyznaczonej przez licznik czasu retransmisji, zatem nadajnik nie wchodzi w fazę *slow start*;
- straty pakietów występują ze stałą częstością p (0<p<1); straty te zostały aproksymowane poprzez założenie, iż po każdej poprawnej transmisji średnio 1/p pakietów następuje strata pakietu (przykład: p=0,01 oznacza iż średnio na każde sto przesłanych pakietów przypada jeden pakiet stracony).



Rysunek 8. Zmiana rozmiaru okna TCP przy okresowych stratach pakietów

Przyjmując, iż rozpatrywane połączenie TCP zakończyło już fazę powolnego startu i przeszło do fazy unikania przeciążenia, oraz biorąc pod uwagę przedstawione wyżej założenia, zachowanie nadajnika TCP możemy podzielić na następujące cykle: okno nadawcze *cwnd* narasta do pewnej wartości maksymalnej, oznaczonej jako W (nadajnik wysyła dane z coraz większą szybkością), następnie, po przesłaniu 1/p pakietów, występuje strata pakietu, *cwnd* ustawiane jest na wartość W/2, po czym rozpoczyna się kolejny cykl – okno nadawcze znów narasta do wartości W (Rysunek 8). Ponieważ zakładamy, iż TCP pracuje w stanie unikania przeciążenia, rozmiar okna zwiększa się o jeden segment po czasie RTT, zatem czas potrzebny do zmiany wartości *cwnd* od W/2 do W jest równy RTT*(W/2). Całkowita liczba przesłanych w tym czasie segmentów wynosi (pole figury zaznaczonej na Rysunek 8):

$$\left(\frac{W}{2}\right)^2 + \frac{1}{2}\cdot\left(\frac{W}{2}\right)^2 = \frac{3}{8}W^2$$

Zgodnie z założeniami, w każdym cyklu zostaje przesłanych 1/p pakietów, zatem $3/8*W^2 = 1/p$. Z równania tego otrzymujemy:

$$W = \sqrt{\frac{8}{3p}}$$

Średni throughput obliczamy jako stosunek ilości danych przesłanych w jednym cyklu (rozmiar segmentu pomnożony przez liczbę wysłanych segmentów) do czasu trwania cyklu:

$$throughput = \frac{dane\_przeslane\_w\_jednym\_cyklu}{czas\_trwania\_cyklu} = \frac{MSS \cdot \frac{3}{8} W^2}{RTT \cdot \frac{W}{2}} = \frac{MSS \cdot \sqrt{\frac{3}{2}}}{RTT \cdot \sqrt{p}}$$

$$throughput \approx 1.22 \cdot \frac{MSS}{RTT \cdot \sqrt{p}}$$

5 Literatura dodatkowa

[1] W.R. Stevens, „Biblia TCP/IP" tom 1, Wydawnictwo RM 1998
Rozdziały 17-21

[2] A. Tanenbaum, "Computer Networks", 3rd edition, Prentice-Hall 1996,
Rozdział 6.4, "The Internet Transport Protocols"

[3] W. Stallings, "Data and Computer Communications", Prentice-Hall, 1997
Rozdział 17, "Transport Protocols"

[4] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", IETF Request For Comments RFC 2581, dostępne na stronie www.ietf.org

[5] M. Mathis, J. Semke, J. Mahdavi, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", Computer Communications Review, volume 27, number 3, July 1997

6 Przykładowe pytania na kolokwium

1. Przedstawić proces nawiązywania i rozłączenia połączenia protokołu TCP.
2. Opisać mechanizm przesuwającego się okna.
3. Na czym polegają algorytmy powolnego startu i unikania przeciążenia?
4. Na czym polegają algorytmy *fast retransmit* i *fast recovery*?
5. Co rozumiemy pod pojęciem czasu RTT?
6. Podaj podstawowe różnice pomiędzy implementacjami TCP Tahoe, Reno i Sack.

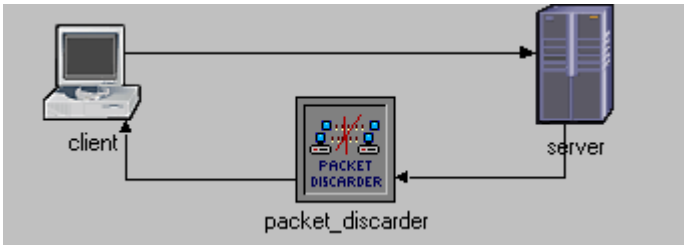
7 Zadania

Poniższe zadania należy wykonać korzystając z programu Riverbed Modeler Academic Edition.

7.1 Zadanie 1, badanie zmian okna nadawczego w czasie

Po uruchomieniu programu Riverbed Modeler należy wczytać projekt *„STIN_TCP”*. Scenariusze symulacji potrzebne do wykonania zadania pierwszego to (*„Scenarios->Switch to scenario”*): *„Tahoe_with_one_drop”*, *„Reno_wih_one_drop”* oraz *„SACK_with_one_drop”*.

Ćwiczenie jest wykonywane w topologii testowej z jednym serwerem i jednym, bezpośrednio do niego podłączonym klientem FTP (Rysunek 9). Moduł *„packet_discarder”* wymusza stratę jednego pakietu w trakcie symulacji, co pozwala na zaobserwowanie reakcji nadajnika TCP na stratę jednego pakietu.



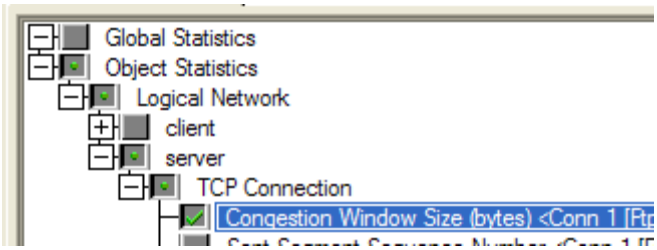
Rysunek 9. Topologia testowa do zadania 1

Celem zadania jest zapoznanie się z charakterem zmian rozmiaru okna nadawczego TCP w poszczególnych fazach nadawania tj. w fazie *slow-start, congestion-avoidance, fast retransmit, fast recovery*.

W sprawozdaniu należy zamieścić i skomentować wykres ewolucji okna nadawczego TCP w funkcji czasu, dla trzech implementacji protokołu TCP: **TCP Tahoe, TCP Reno, TCP Sack**. Podpisać na wykresie: wartość ***ss_threshold***, maksymalną wartość okna nadawczego ***(AWND)***, poszczególne fazy nadawania tj. ***slow start***, ***congestion avoidance***, ***fast retransmit***, ***fast recovery***.

7.1.1 Jak odczytać wykres zmian okna nadawczego w czasie?

Z menu głównego należy wybrać *„Results->View Results”* i rozwinąć dostępne opcje tak jak to przedstawia Rysunek 10. Po wyświetleniu okna *„Congestion Window Size”* należy zaznaczyć myszką obszar wykresu, który chcemy obejrzeć w zbliżeniu.



Rysunek 10. Odczytanie statystyki rozmiaru okna nadawczego w czasie

7.2 Zadanie 2, badanie wpływu maksymalnego rozmiaru okna

Zadanie to należy wykonać korzystając ze scenariusza symulacji *„Zadanie_2”*. Topologia testowa składa się z serwera z jednym, bezpośrednio podłączonym do niego klientem FTP (Rysunek 11). Klient FTP ściąga z serwera FTP plik o rozmiarze 10 MB.

Rysunek 11. Topologia testowa do zadania 2

Celem ćwiczenia jest zaobserwowanie wpływu rozmiaru okna nadawczego na szybkość bitową transferu danych za pomocą protokołu TCP. Ze względu na to, że w badanej sieci nie występują straty, okno nadawcze połączenia TCP szybko uzyskuje maksymalny rozmiar, równy AWND, i utrzymuje go aż do końca przesłania pliku.

Należy sporządzić wykres wartości parametru ***Goodput*** w funkcji maksymalnego rozmiaru okna rozgłaszanego przez klienta (AWND). Ponieważ maksymalne rozgłaszane okno równe jest pojemności bufora odbiorczego klienta, zmianę rozmiaru AWND trzeba realizować poprzez odpowiednie ustawienie pojemności bufora odbiorczego klienta FTP (w programie OPNET należy otworzyć okno edycji parametrów klienta FTP (*„Edit attributes”*) i zmienić wartość parametru *„Receive buffer (bytes)”* w grupie parametrów *„TCP parameters”*).

Pozostałe parametry konfiguracji:

MSS=1kB

Czas propagacji na łączu = 0.05 s

Przepływność łącza C=155 Mbit/s

Wartość parametru ***goodput*** należy obliczyć wg poniższego wzoru :

$$Goodput = \frac{file\_size[b]}{download\_time[s]} \tag{7}$$

Przykładowa postać tabeli wyników.

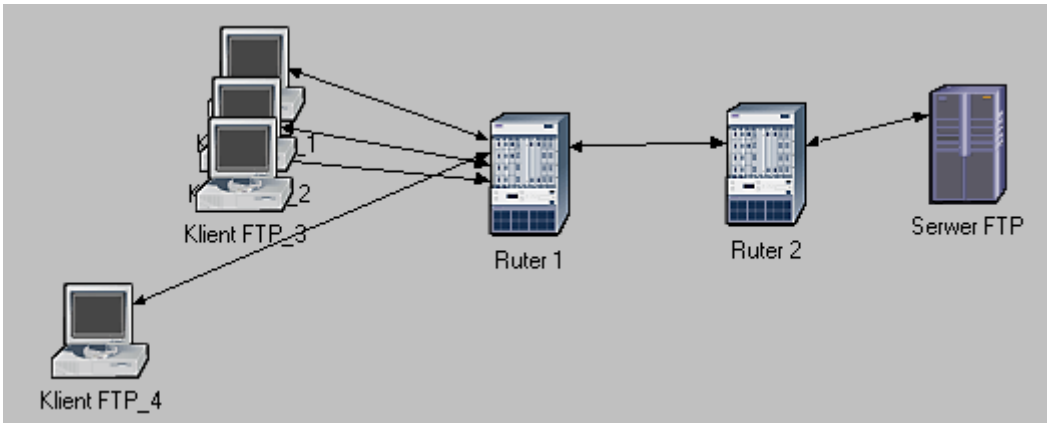
Rozmiar rozgłaszanego okna (AWND) [kB]	Czas przesłania pliku [s]	Goodput [bit/s]
4		
16		
32		
64		

7.2.1 Sposób wyznaczania wartości parametru goodput uzyskanej w symulacji

W celu obliczenia goodputu należy odczytać czas przesłania pliku FTP, tj. po symulacji otworzyć okno przeglądania wyników: *„Results->View results”* i odczytać wykres *„Object Statistics -> Logical Network->Client FTP->Download Response Time”*. Wartość parametru goodput można policzyć dzieląc rozmiar pliku (10MB) przez czas jego przesłania. Należy przy tym pamiętać, że jednostką goodputu powinien być [bit/s], więc otrzymaną wartość należy odpowiednio przeliczyć.

7.3 Zadanie 3, badanie współdzielenia zasobów pomiędzy kilkoma połączeniami TCP

Zadanie to należy wykonać korzystając ze scenariusza symulacji *„Zadanie_3”*. Topologia testowa (Rysunek 12) jest topologią typu *bottleneck* i składa się z dwóch ruterów, serwera i czterech klientów FTP.



Rysunek 12. Topologia testowa do zadania 4

Wszyscy klienci w tym samym momencie rozpoczynają ściąganie pliku o rozmiarze 10MB z serwera FTP. Ze względu na to, że łącze pomiędzy klientem FTP_4 a ruterem 1 ma dłuższy czas propagacji, zestawione połączenia TCP różnią się wartością RTT.

Celem zadania jest zbadanie sposobu, w jaki cztery jednocześnie aktywne połączenia TCP uzyskują dostęp do wspólnych zasobów łącza. Należy zmierzyć wartości parametru ***goodput*** osiągane przez klientów FTP_1, 2, 3 i 4. Ze względu na to, iż połączenie zestawione przez klienta FTP_4 będzie charakteryzować się innym czasem RTT, wyniki pozwalają zaobserwować wpływ tego czasu na sprawność protokołu TCP. Czas RTT można zmieniać w sposób pośredni, poprzez odpowiednie ustawienie czasu propagacji na łączu pomiędzy klientem FTP_4 a ruterem 1.

Pozostałe parametry konfiguracji:

MSS=1kB
Przepływność łącza bottleneck, C= 2Mbit/s
AWND=64kB (równe pojemności bufora odbiorczego po stronie klienta)
Czas propagacji na łączach FTP_1,2,3-ruter1 = 0.017s

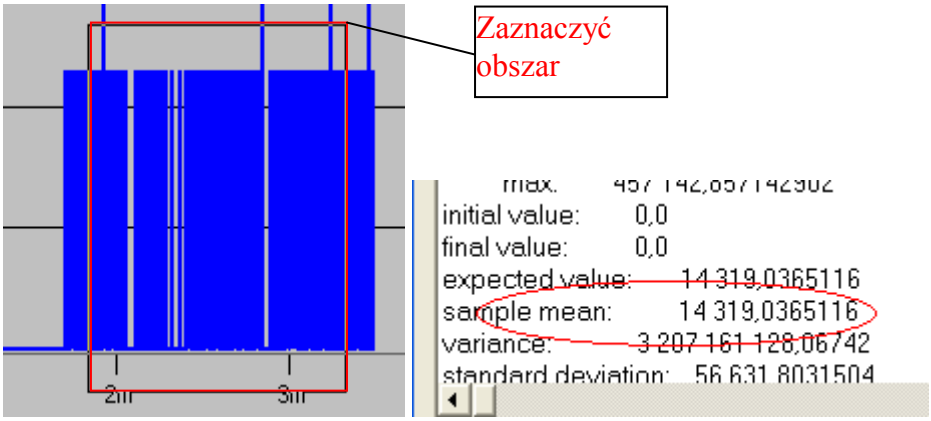
Przykładowa postać tabeli wyników.

Czas propagacji łącza FTP_4 – ruter1 [s]	Średni czas RTT czwartego źródła TCP [s]	Goodput 1 [Mbit/s]	Goodput 2 [Mbit/s]	Goodput 3 [Mbit/s]	Goodput 4 [Mbit/s]
0.017					
0.07					
0.117					

7.3.1 Jak obliczyć wartość parametru goodput uzyskaną w symulacji?

Otworzyć okno przeglądania wyników: *„Results->View results”* i odczytać wykres *„Logical network->Client FTP_4->TCP->Traffic received (bytes/sec)”*. Po otworzeniu okna tego

wykresu, należy zaznaczyć myszką część wykresu przedstawiającą fragment zmierzony w trakcie przekazu danych, tak jak to pokazuje Rysunek 13. Z tak otrzymanego wykresu należy obliczyć wartość średnią (*„sample mean”*), klikając prawym przyciskiem na krawędzi wykresu i wybierając *„Show statistic data”*. Uwaga: otrzymana w ten sposób wartość jest wyrażona w bajtach/sek.

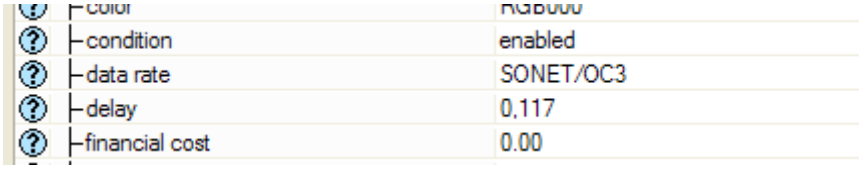


a) b)

Rysunek 13. a) Zaznaczenie części wykresu i b) odczytanie wartości średniej

7.3.2 Jak zmienić czas propagacji na łączu?

Czas propagacji można zmienić, klikając na danym łączu prawym przyciskiem myszki i wybierając opcję *„Advanced Edit Attributes”*. Czas propagacji, wyrażony w sekundach, należy ustawić w polu *„Delay”* (Rysunek 14).



Rysunek 14. Ustawienie czasu propagacji na łączu