Selektywne powtarzanie (SP)

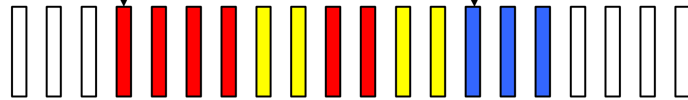
- odbiorca *selektywnie* potwierdza poprawnie odebrane pakiety
  - buforuje pakiety, gdy potrzeba, w celu uporządkowania przed przekazaniem warstwie wyższej
- nadawca retransmituje tylko pakiety, dla których nie odebrał ACK
  - nadawca ma zegar dla każdego niepotwierdzonego, wysłanego pakietu.
- okno nadawcy
  - N kolejnych numerów sekwencyjnych
  - określa, jakie pakiety mogą być wysłane bez potwierdzenia

SP: okna nadawcy i odbiorcy

początek okna
(pocz_okna)

nadawca

następny numer sekwencyjny
(nast_num)



rozmiar okna: N

już potwierdzony

gotowy, nie wysłany

wysłany, nie potwierdzony

nie używany

potwierdzony, w buforze

gotowy do odebrania

oczekiwany, nie otrzymany

nie używany

rozmiar okna: N

odbiorca

SP: nadawca i odbiorca

nadawca

dane od wyższej warstwy:

- jeśli w oknie jest wolny numer sekwencyjny, wyślij pakiet

timeout(n):

- retransmituj pakiet n, ustaw ponownie zegar

ACK(n) pakietu w oknie:

- zaznacz pakiet jako odebrany i wyłącz zegar
- jeśli n jest początkiem okna, przesuń okno do następnego niepotwierdzonego pakietu

odbiorca

pakiet n z okna

- wyślij ACK(n)
- nie w kolejności: do bufora
- w kolejności: przekaż (także przekaż uporządkowane pakiety z bufora), przesuń okno do następnego nieodebranego pakietu

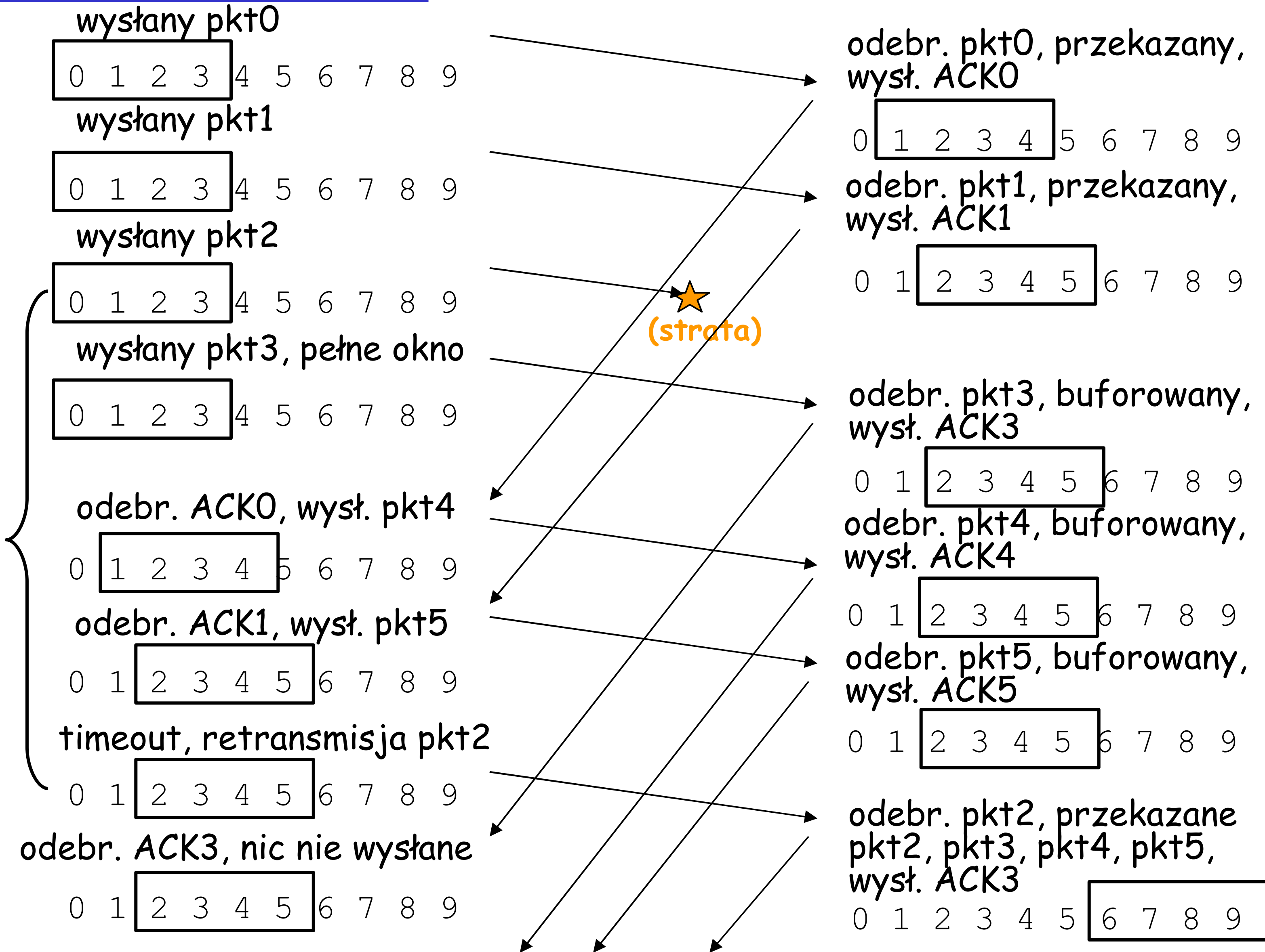
pakiet n z N pakietów przed oknem

- potwierdź ACK(n)

wszystkie inne pakiety:

- ignoruj

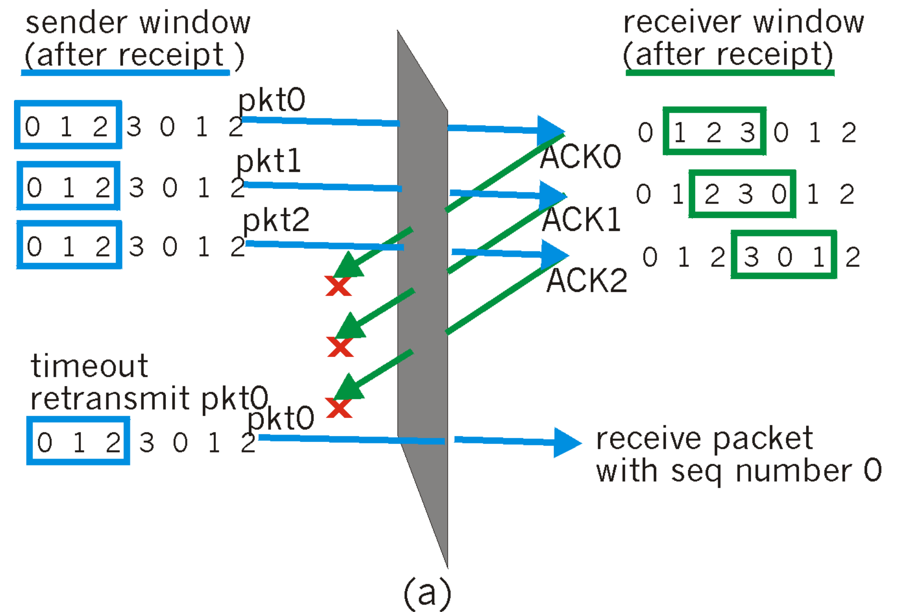
SP w działaniu



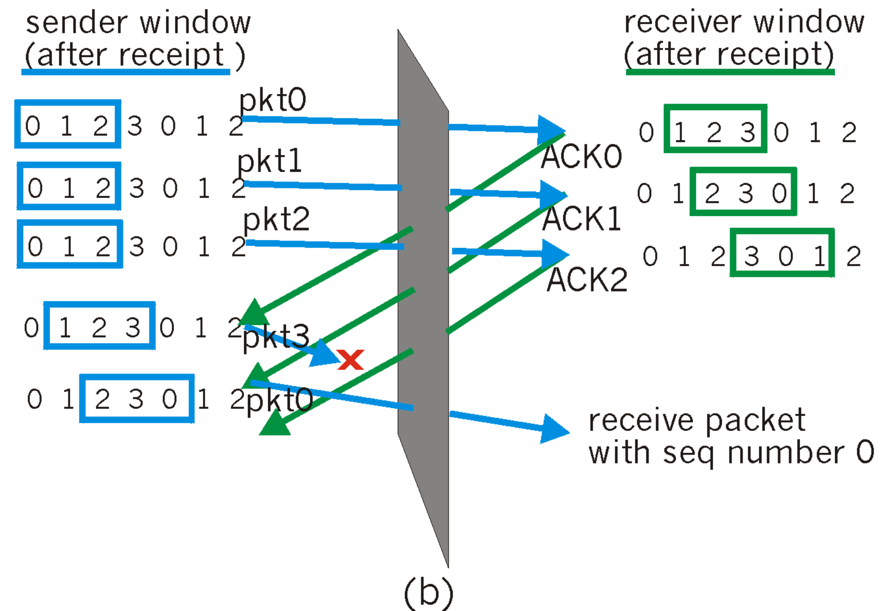
SP: dylemat

Przykład:

- numery: 0, 1, 2, 3
- rozmiar okna = 3



- odbiorca nie odróżni obu scenariuszy!
- niepoprawnie przekaże podwójnie dane w (a)



Pytanie: jaki jest związek pomiędzy ilością numerów sekwencyjnych a rozmiarem okna?

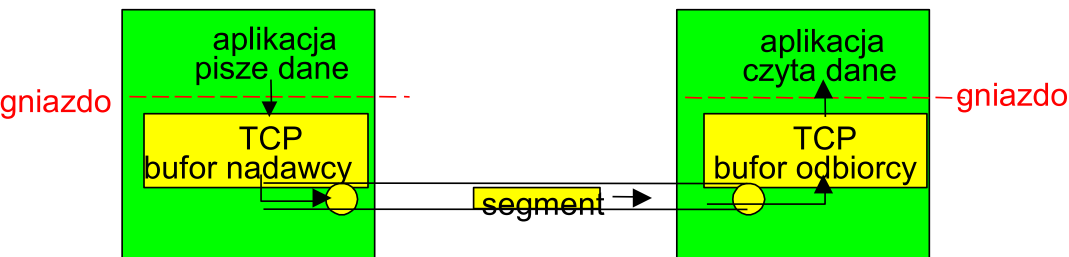
Mapa wykładu

- Usługi warstwy transportu
- Multipleksacja i demultipleksacja
- Transport bezpołączeniowy: UDP
- Zasady niezawodnej komunikacji danych
- Transport połączeniowy: TCP
  - struktura segmentu
  - niezawodna komunikacja
  - kontrola przepływu
  - zarządzanie połączeniem
- Mechanizmy kontroli przeciążenia
- Kontrola przeciążenia w TCP

TCP: Przegląd

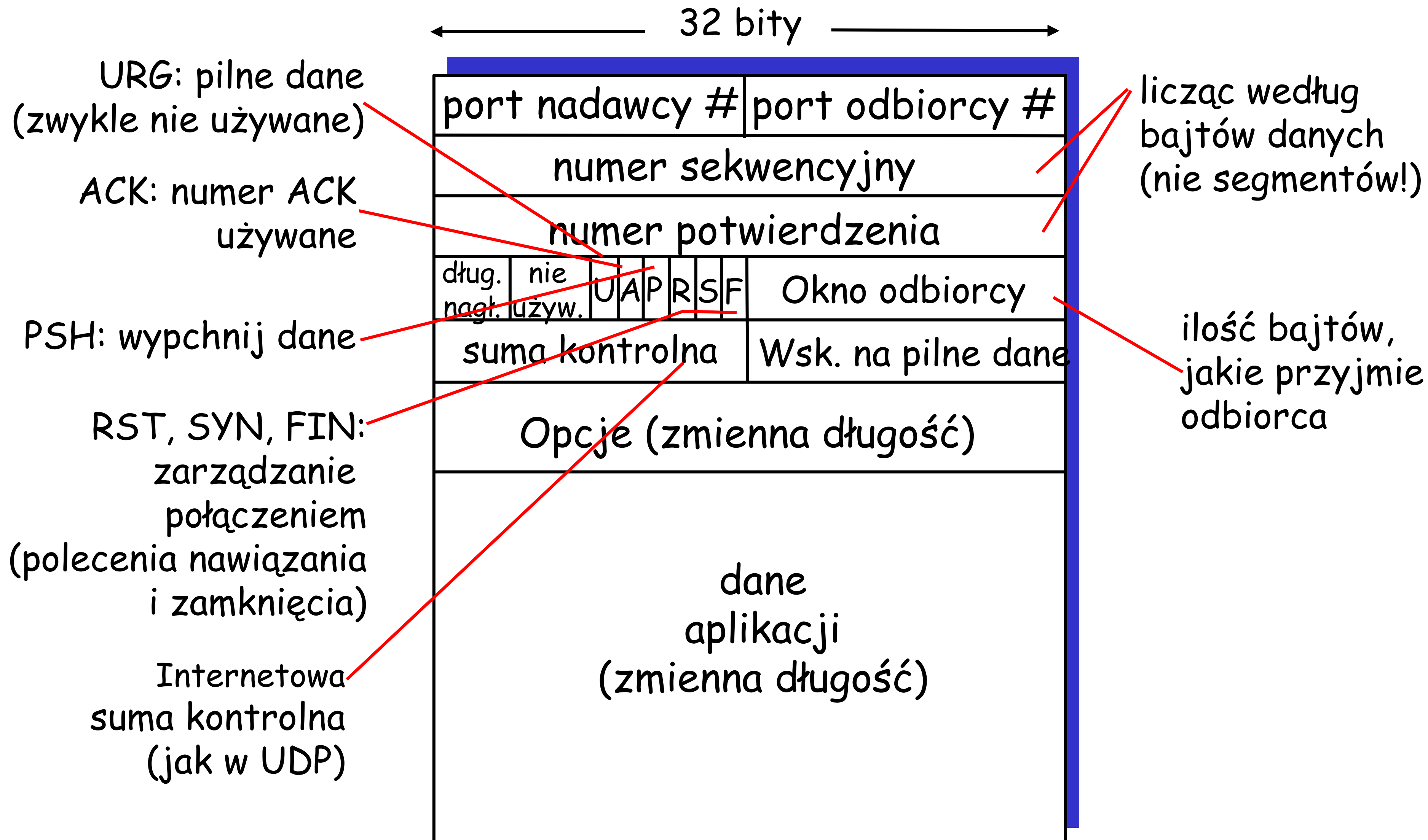
RFC: 793, 1122, 1323, 2018, 2581

- koniec-koniec:
  - jeden nadawca, jeden odbiorca
- niezawodny, uporządkowany *strumień bajtów:*
  - nie ma "granic komunikatów"
- wysyłający grupowo:
  - kontrola przeciążeń i przepływu TCP określają rozmiar okna
- *bufory u nadawcy i odbiorcy*



- komunikacja "full duplex":
  - dwukierunkowy przepływ danych na tym samym połączeniu
  - MRS: maksymalny rozmiar segmentu
- połączeniowe:
  - inicjalizacja (wymiana komunikatów kontrolnych) połączenia przed komunikacją danych
- kontrola przepływu:
  - nadawca nie "zaleje" odbiorcy

Struktura segmentu TCP



TCP: numery sekwencyjne i potwierdzenia

Numery sekwencyjne:

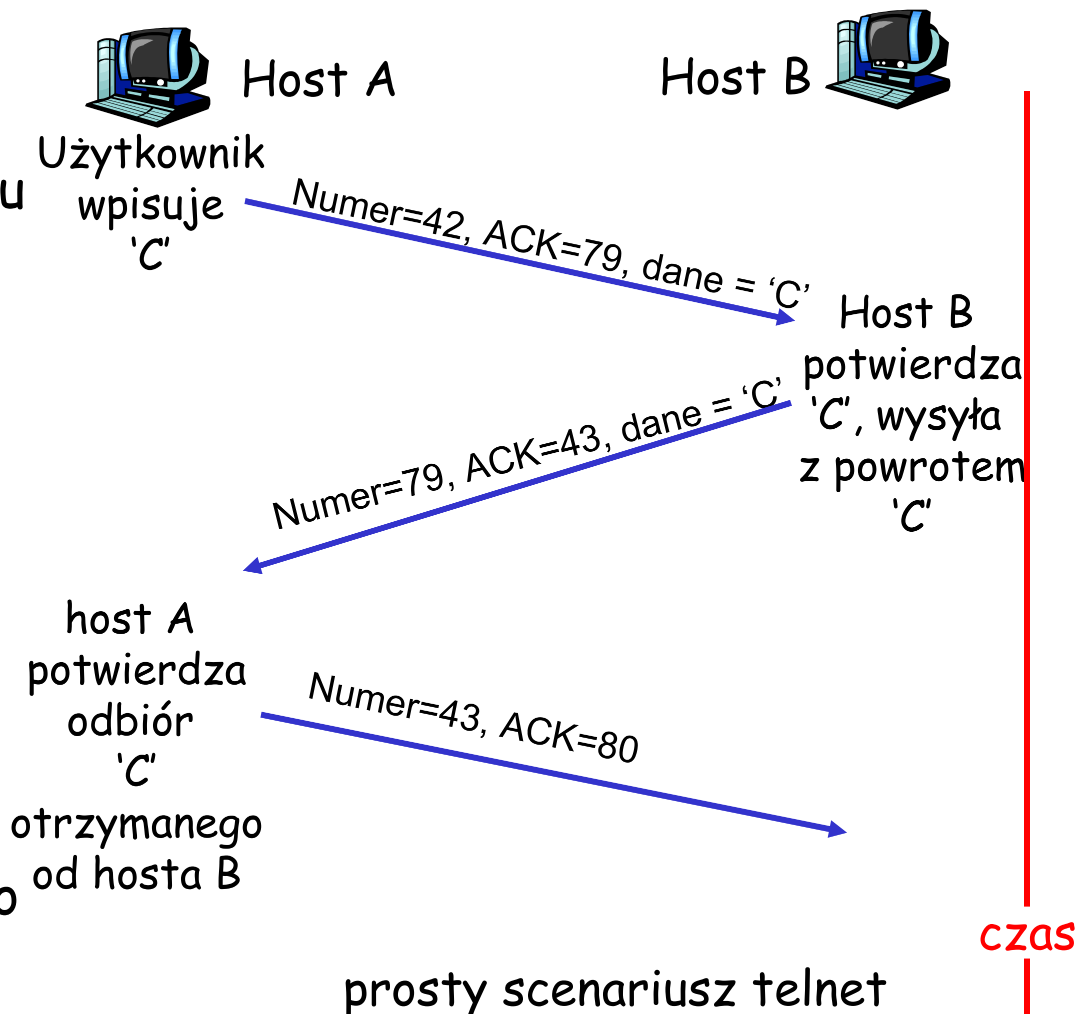
- numer w "strumieniu bajtów" pierwszego bajtu danych segmentu

Potwierdzenia:

- numer sekwencyjny następnego bajtu oczekiwanego od drugiej strony
- kumulatywny ACK

Pytanie: jak odbiorca traktuje segmenty nie w kolejności

- O: specyfikacja TCP tego nie określa: decyduje implementacja



TCP: czas powrotu (RTT) i timeout

Pytanie: jak ustalić timeout dla TCP?

- dłuższe niż RTT
  - ale RTT jest zmienne
- za krótki: za wczesny timeout
  - niepotrzebne retransmisje
- za długi: wolna reakcja na stratę segmentu

Pytanie: jak estymować RTT?

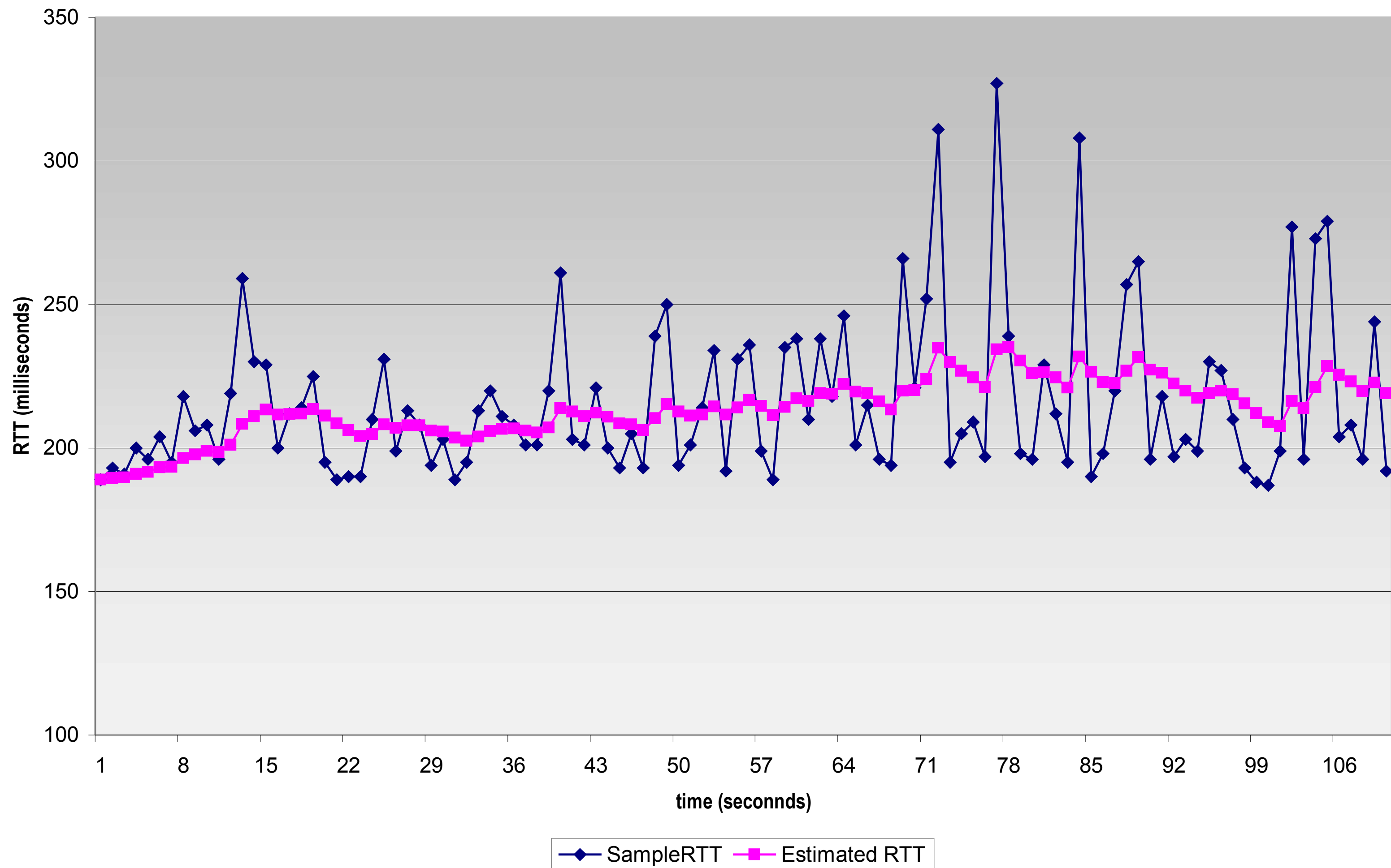
- **MierzoneRTT**: czas zmierzony od wysłania segmentu do odbioru ACK
  - ignorujemy retransmisje
- **MierzoneRTT** będzie zmienne, chcemy mieć "gładsze" estymowane RTT
  - średnia z wielu ostatnich pomiarów, nie tylko aktualnego **MierzoneRTT**

TCP: czas powrotu (RTT) i timeout

EstymowaneRTT = (1- α)*EstymowaneRTT + α*MierzoneRTT

- Wykładnicza średnia ruchoma
- wpływ starych pomiarów maleje wykładniczo
- typowa wartość parametru: $\alpha = 0.125$

Przykład estymacji RTT:



TCP: czas powrotu (RTT) i timeout

Ustawianie timeout

- **EstymowaneRTT** dodać "margines błędu"
  - im większa zmienność **MierzoneRTT**, **tym** większy margines błędu
- najpierw ocenić, o ile **MierzoneRTT** jest oddalone od **EstymowaneRTT**:

```
BłądRTT = (1-β)*BłądRTT +
                 β*|MierzoneRTT - EstymowaneRTT|

(zwykle, β = 0.25)
```

Ustalanie wielkości timeout:

```
Timeout = EstymowaneRTT + 4*BłądRTT
```

Mapa wykładu

- Usługi warstwy transportu
- Multipleksacja i demultipleksacja
- Transport bezpołączeniowy: UDP
- Zasady niezawodnej komunikacji danych
- Transport połączeniowy: TCP
  - struktura segmentu
  - niezawodna komunikacja
  - kontrola przepływu
  - zarządzanie połączeniem
- Mechanizmy kontroli przeciążenia
- Kontrola przeciążenia w TCP

Niezawodna komunikacja TCP

- TCP tworzy usługę NPK na zawodnej komunikacji IP
- Wysyłanie grupowe segmentów
- Kumulowane potwierdzenia
- TCP używa pojedynczego zegara do retransmisji

- Retransmisje są powodowane przez:
  - zdarzenia timeout
  - duplikaty ACK
- Na razie rozważymy prostszego nadawcę TCP:
  - ignoruje duplikaty ACK
  - ignoruje kontrolę przeciążenia i przepływu

Zdarzenia nadawcy TCP:

Dane od aplikacji:

- Utwórz segment z numerem sekwencyjnym
- numer sekwencyjny to numer w strumieniu bajtów pierwszego bajtu danych segmentu
- włącz zegar, jeśli jest wyłączony (zegar działa dla najstarszego niepotwierdzonego segmentu)
- oblicz czas oczekiwania: `Timeout`

Timeout:

- retransmituj segment, który spowodował timeout
- ponownie włącz zegar

Odbiór ACK:

- Jeśli potwierdza niepotwierdzone segmenty:
  - potwierdź odpowiednie segmenty
  - włącz zegar, jeśli są brakujące segmenty

Nadawca TCP (uproszczony)

```
 Nast_Num = 1
Pocz_Okna  = 1
loop (forever)
{
 switch( zdarzenie )
 {
  zdarzenie: Dane od aplikacji
      stwórz segment z numerem Nast_Num;
      if ( zegar wyłączony )
          włącz zegar;
      przekaż segment do warstwy IP;
      Nast_Num = Nast_Num + length( dane );

  zdarzenie: Timeout
      retransmituje niepotwierdzony segment
            o najniższym numerze sekwencyjnym;
      włącz zegar;

  zdarzenie: Odbiór ACK potwierdzającego pakiet y
      if  ( y > Pocz_Okna )
      {
         Pocz_Okna = y;
         if ( są niepotwierdzone segmenty w oknie )
            włącz zegar;
      }
  }
 }  /* end of loop forever */
```

Komentarz:

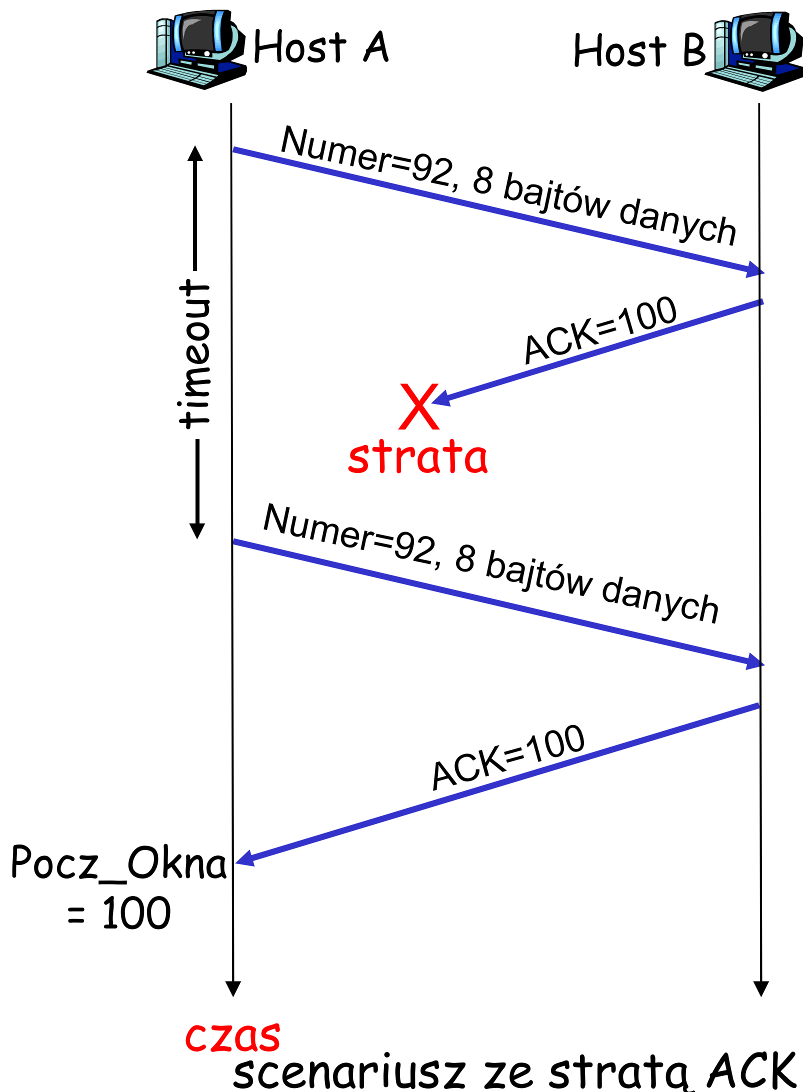
- Pocz_Okna - 1: ostatni potwierdzony bajt

Przykład:

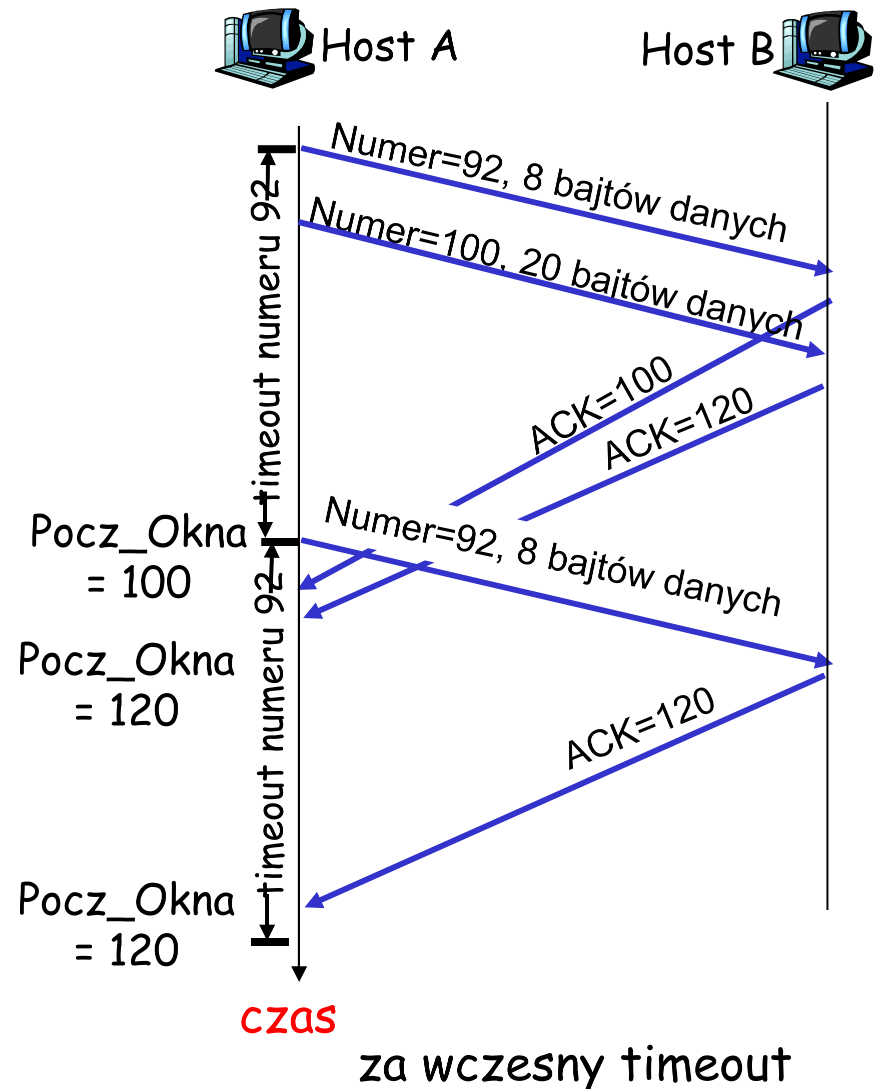
- Pocz_Okna - 1 = 71;

y= 73, więc odbiorca chce bajty powyżej 73;
y > Pocz_Okna, więc nowe dane zostały potwierdzone

TCP: scenariusze retransmisji

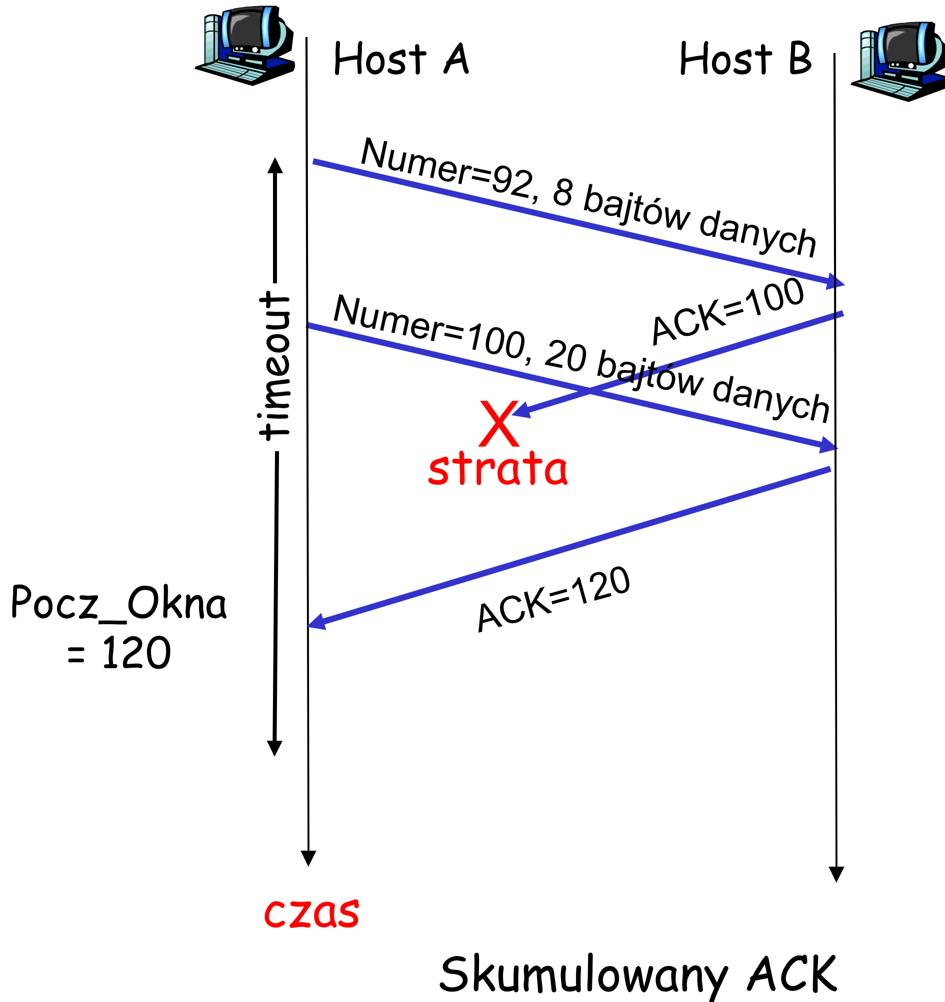


scenariusz ze stratą ACK



za wczesny timeout

TCP: scenariusze retransmisji (c.d.)



Skumulowany ACK

Wysyłanie ACK w TCP [RFC 1122, RFC 2581]

Zdarzenie u odbiorcy TCP	Akcja odbiorcy TCP
Odbiór segmentu o oczekiwanym numerze w kolejności. Wszystkie poprzednie dane już potwierdzone	Opóźnione ACK. Czekaj do 500 ms na następny segment. Jeśli go nie ma, wyślij ACK.
Odbiór segmentu o oczekiwanym numerze w kolejności. Jeden inny segment nie był potwierdzony	Wyślij natychmiast skumulowane ACK, potwierdzające oba segmenty.
Odbiór segmentu nie w kolejności o za wysokim numerze. Wykrycie luki w danych	Wyślij natychmiast duplikat ACK, w którym podany jest numer oczekiwanego bajtu
Odbiór segmentu, który całkiem lub częściowo wypełnia lukę.	Jeśli segment leży na początku luki, wyślij natychmiast ACK.

Szybkie retransmisje

- Okres oczekiwania na timeout jest długi:
  - długie czekanie na retransmisje
- Rozpoznaj stracone segmenty przez duplikaty ACK.
  - Nadawca często wysyła segmenty jeden tuż po drugim
  - Jeśli segment zginie, może być wiele duplikatów ACK.
- Jeśli nadawca otrzyma 3 duplikaty ACK dla tych samych danych, zakłada, że następny segment po potwierdzonych danych zginął:
  - szybkie retransmisje: wyślij segment zanim nastąpi timeout

Algorytm szybkich retransmisji:

zdarzenie: **Odbiór ACK potwierdzającego pakiet y**

```
if  ( y > Pocz_Okna )
{
   Pocz_Okna = y;
   if ( są niepotwierdzone segmenty w oknie )
       włącz zegar;
}
else
{
    zwiększ licznik duplikatów ACK dla y;
    if  ( licznik duplikatów ACK dla y == 3 )
       retransmituj segment y;
}
```

duplikat ACK dla już potwierdzonego segmentu

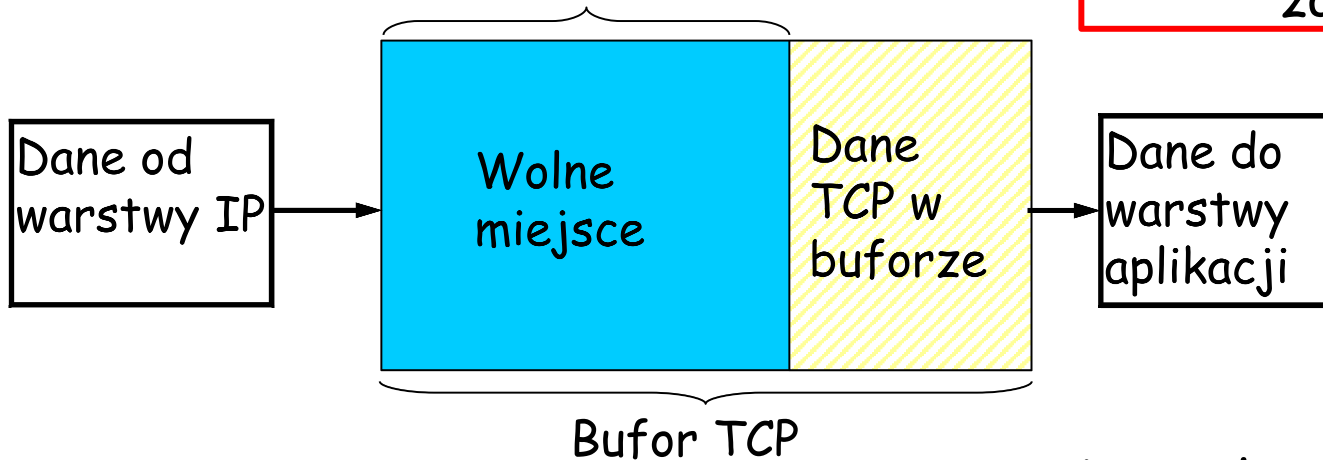
szybka retransmisja

Mapa wykładu

- Usługi warstwy transportu
- Multipleksacja i demultipleksacja
- Transport bezpołączeniowy: UDP
- Zasady niezawodnej komunikacji danych
- Transport połączeniowy: TCP
  - struktura segmentu
  - niezawodna komunikacja
  - kontrola przepływu
  - zarządzanie połączeniem
- Mechanizmy kontroli przeciążenia
- Kontrola przeciążenia w TCP

Kontrola przepływu TCP

- odbiorca TCP ma bufor odbiorcy:

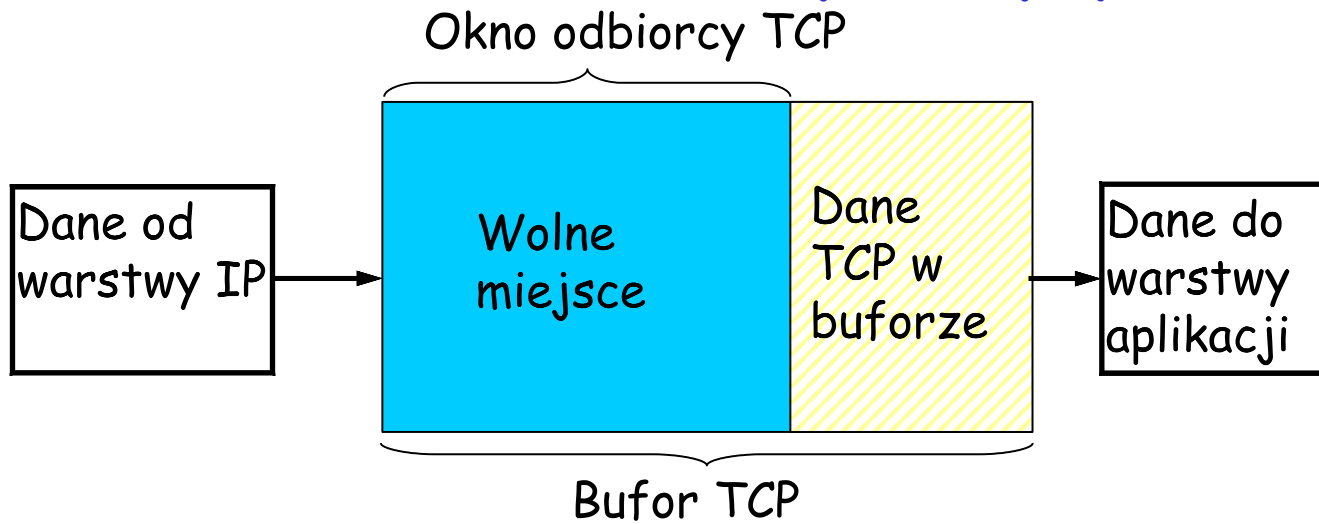


kontrola przepływu
nadawca nie "zaleje" bufora odbiorcy przesyłając za dużo i za szybko

- Proces aplikacji może za wolno czytać z bufora

- usługa dopasowania prędkości: dopasuje prędkość wysyłania do prędkości odbioru danych przez aplikację

Jak działa kontrola przepływu TCP



Załóżmy, że odbiorca wyrzuca segmenty nie w kolejności.

- Odbiorca ogłasza wolne miejsce umieszczając jego wielkość w segmentach ACK
- Odbiorca ogranicza rozmiar okna do wolnego miejsca
  - gwarantuje, że bufor nie zostanie przepełniony

Mapa wykładu

- Usługi warstwy transportu
- Multipleksacja i demultipleksacja
- Transport bezpołączeniowy: UDP
- Zasady niezawodnej komunikacji danych
- Transport połączeniowy: TCP
  - struktura segmentu
  - niezawodna komunikacja
  - kontrola przepływu
  - zarządzanie połączeniem
- Mechanizmy kontroli przeciążenia
- Kontrola przeciążenia w TCP

Zarządzanie połączeniem TCP

Przypomnienie: Nadawca i odbiorca TCP, tworzą "połączenie" zanim wymienią dane

- inicjalizacja zmiennych TCP:
  - numery sekwencyjne
  - bufory, informacja kontroli przepływu (rozmiar bufora)
- *klient:* nawiązuje połączenie

```
Socket clientSocket = new
 Socket("hostname","port

 number");
```

- *serwer:* odbiera połączenie

```
Socket connectionSocket =
 welcomeSocket.accept();
```

Trzykrotny uścisk dłoni:

Krok 1: host klienta wysyła segment SYN do serwera
- podaje początkowy numer sekwencyjny
- bez danych

Krok 2: host serwera odbiera SYN, odpowiada segmentem SYNACK
- serwer alokuje bufory
- określa początkowy numer

Krok 3: klient odbiera SYNACK, odpowiada segmentem ACK, **który może zawierać dane**

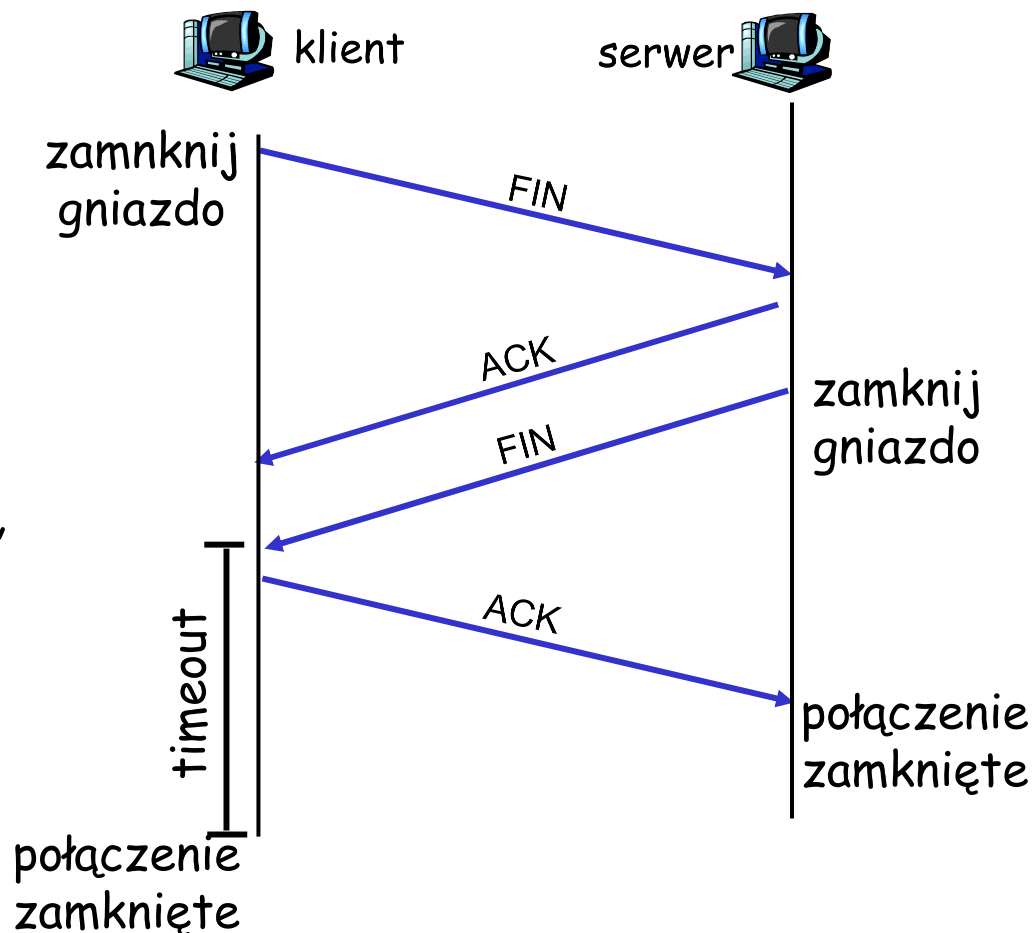
Zarządzanie połączeniem TCP (c.d..)

Zamykanie połączenia:

klient zamyka gniazdo:
```
clientSocket.close();
```

Krok 1: Host klienta wysyła segment TCP FIN do serwera

Krok 2: serwer odbiera FIN, odpowiada segmentem ACK. Zamyka połączenie, wysyła FIN.



Zarządzanie połączeniem TCP (c.d.)

Krok 3: klient odbiera FIN, odpowiada segmentem ACK.

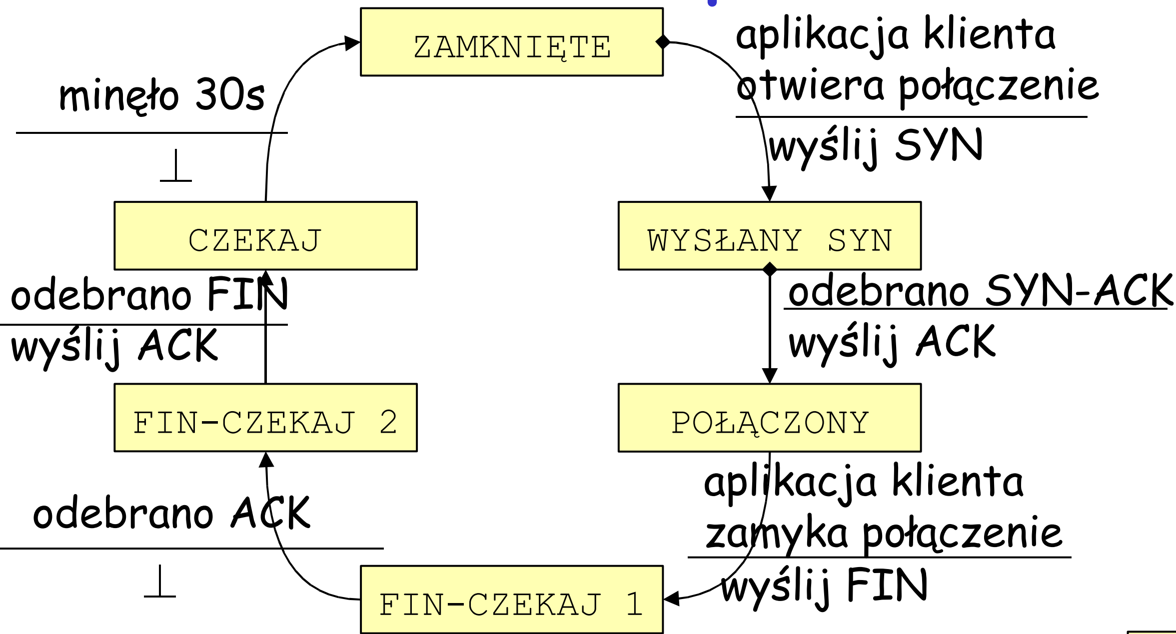
- Rozpoczyna oczekiwanie- będzie odpowiadał ACK na otrzymane FIN.

Krok 4: serwer odbiera ACK. Połączenie zamknięte.

Uwaga: z małymi zmianami, obsługuje jednoczesne FIN.

klient
serwer
zamknij gniazdo
FIN
ACK
zamknij gniazdo
FIN
ACK
timeout
połączenie zamknięte
połączenie zamknięte

Zarządzanie połączeniem TCP (c.d..)



Cykl życia
klienta TCP

Cykl życia
serwera TCP

ZAMKNIĘTE

aplikacja serwera
tworzy gniazdo
czekające

⊥

SŁUCHANIE

odebrano SYN

wyślij SYN-ACK

ODEBRANY SYN

odebrano ACK

⊥

POŁĄCZONY

Odebrano FIN

wyślij ACK

ZAMKNIJ
CZEKAJ

⊥

wyślij FIN

OSTATNI ACK

odebrano ACK

⊥

Mapa wykładu

- Usługi warstwy transportu
- Multipleksacja i demultipleksacja
- Transport bezpołączeniowy: UDP
- Zasady niezawodnej komunikacji danych
- Transport połączeniowy: TCP
  - struktura segmentu
  - niezawodna komunikacja
  - kontrola przepływu
  - zarządzanie połączeniem
- Mechanizmy kontroli przeciążenia
- Kontrola przeciążenia w TCP

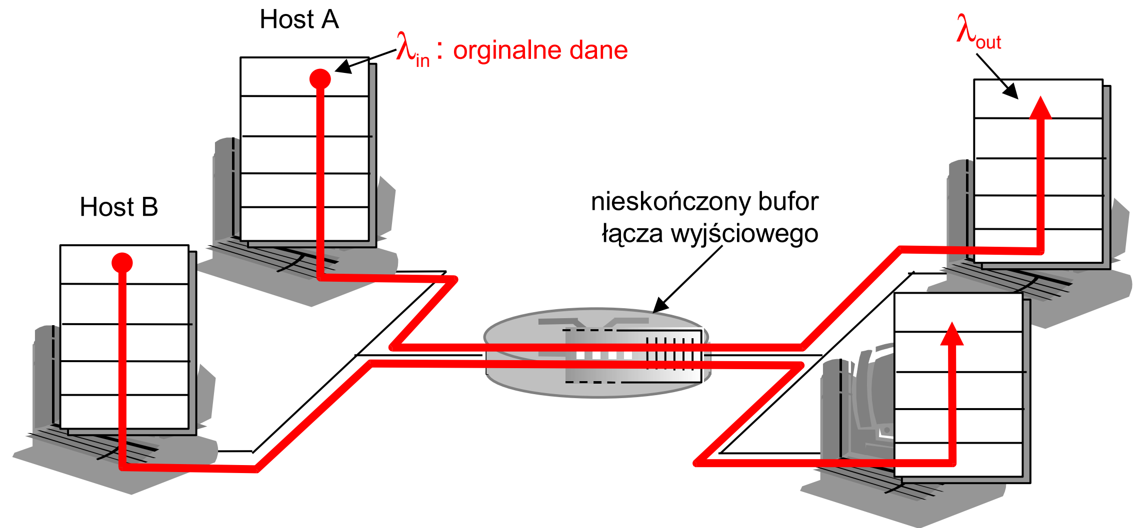
Zasady kontroli przeciążenia

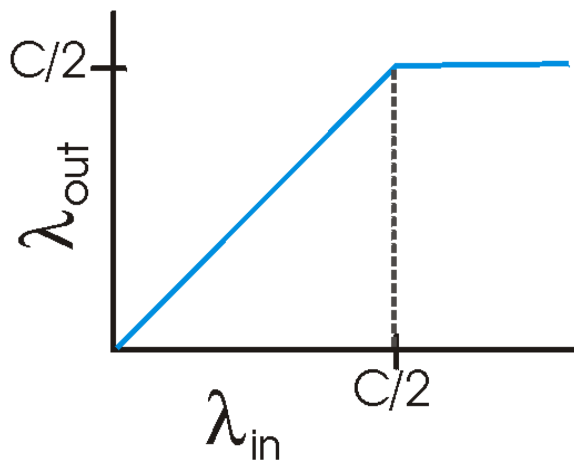
Przeciążenie:

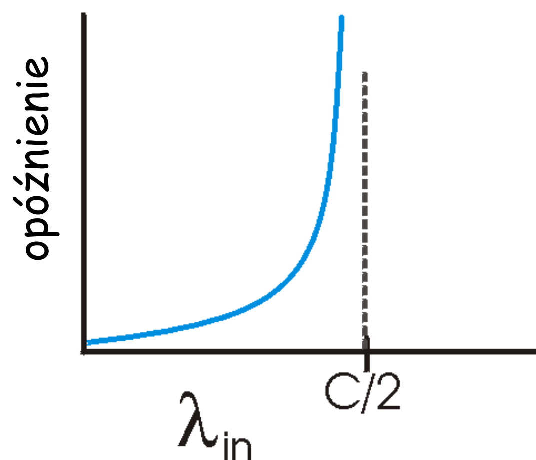
- nieformalnie: “za wiele źródeł wysyła za wiele danych za szybko, żeby *sieć* mogła je obsłużyć”
- różni się od kontroli przepływu!
- objawy przeciążenia:
  - zgubione pakiety (przepełnienie buforów w ruterach)
  - duże (i zmienne) opóźnienia (kolejkowanie w buforach ruterów)
- jeden z głównych problemów sieci!
- konsekwencja braku kontroli dostępu do sieci

Przyczyny/koszty przeciążenia: scenariusz 1

- dwóch nadawców, dwóch odbiorców
- jeden ruter, nieskończone bufory
- bez retransmisji



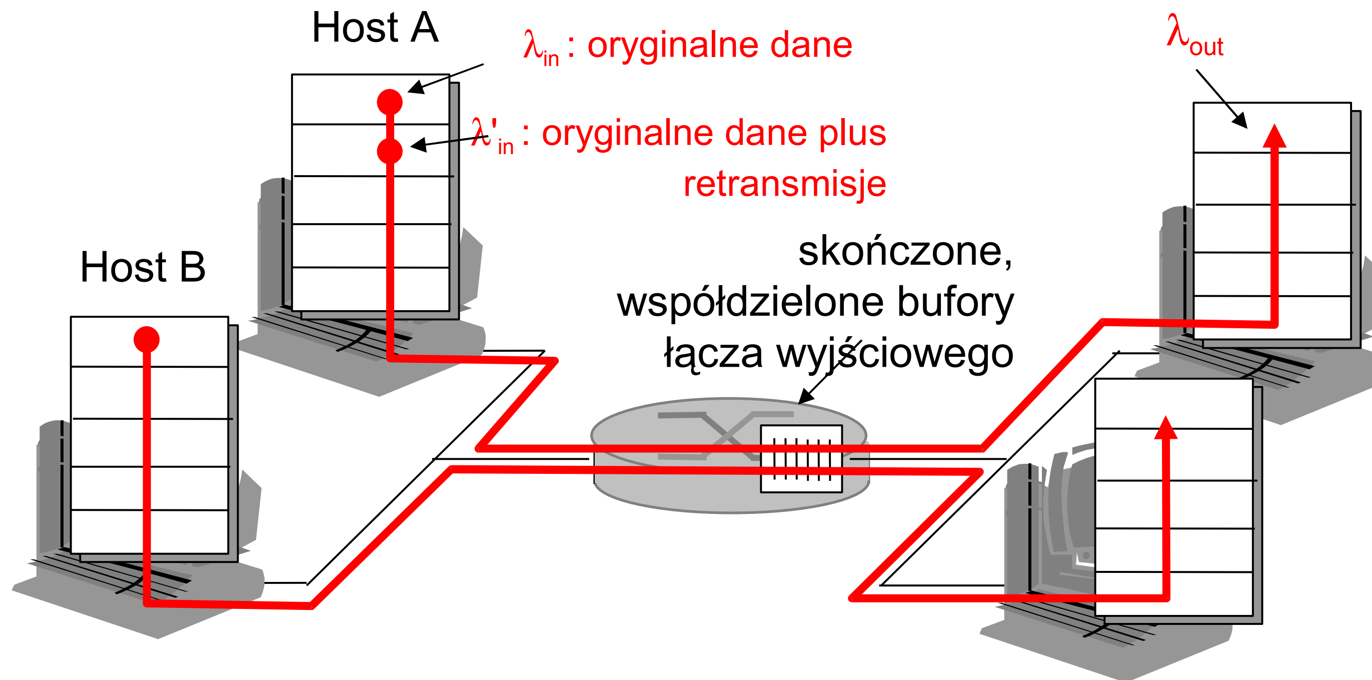




- duże opóźnienia przy przeciążeniu
- maksymalna dostępna przepustowość

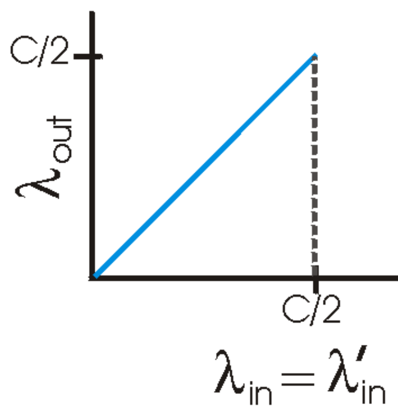
Przyczyny/koszty przeciążenia: scenariusz 2

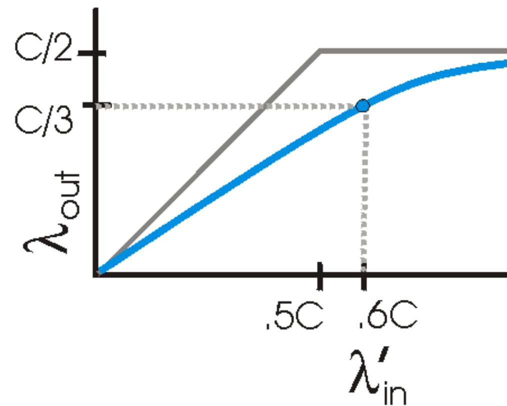
- jeden ruter, *skończone* bufory
- retransmisje straconych pakietów

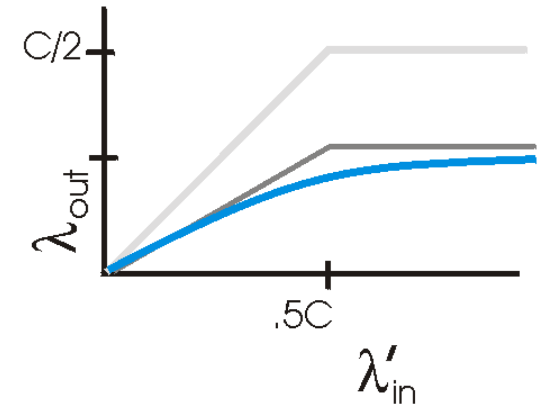


Przyczyny/koszty przeciążenia: scenariusz 2

- zawsze: $\lambda_{in} = \lambda_{out}$ (goodput)
- "doskonałe" retransmisje tylko po stracie: $\lambda'_{in} > \lambda_{out}$
- retransmisje opóźnionych (nie straconych) pakietów zwiększa λ'_{in} (w porównaniu z doskonałymi) dla tego samego λ_{out}







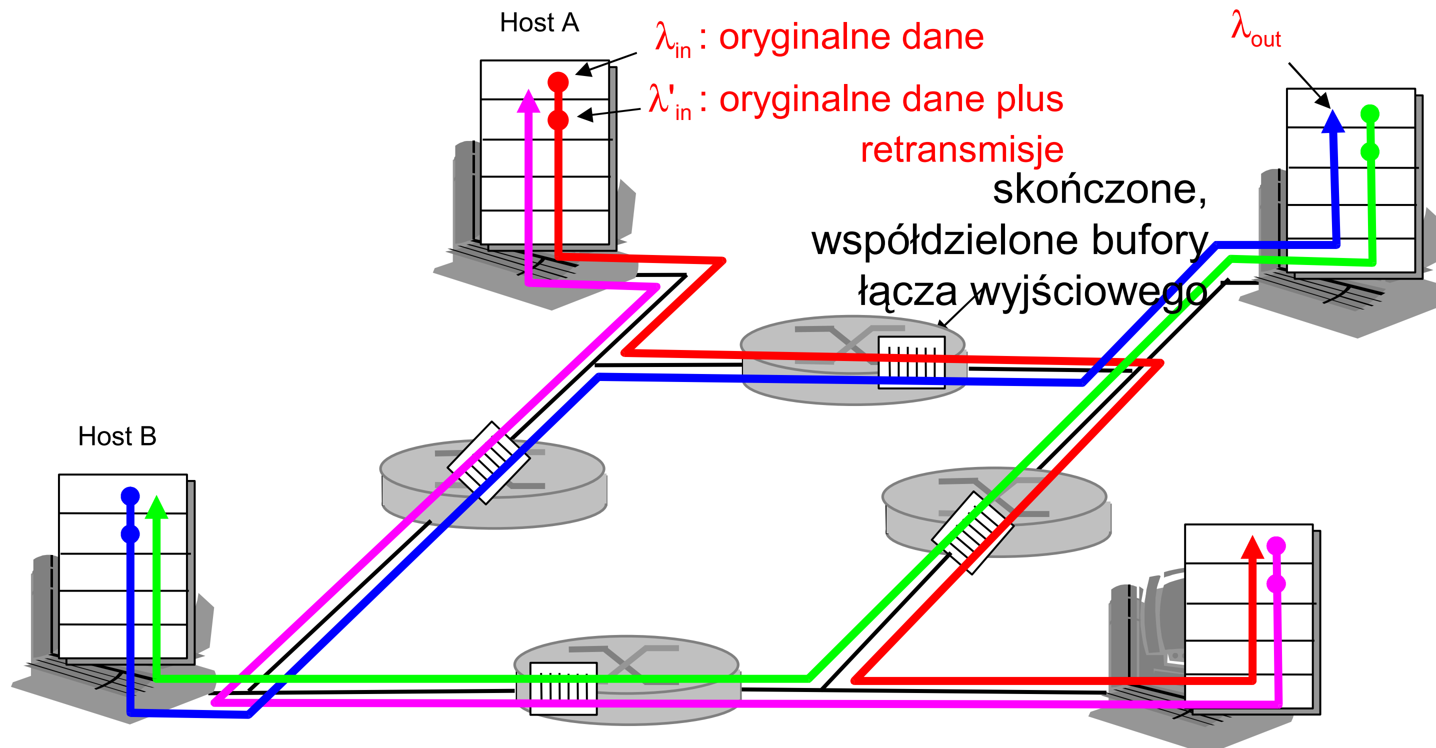
"koszty" przeciążenia:

- więcej pracy (retransmisje) na określony "goodput"
- niepotrzebne retransmisje: łącze przesyła wiele kopii pakietu

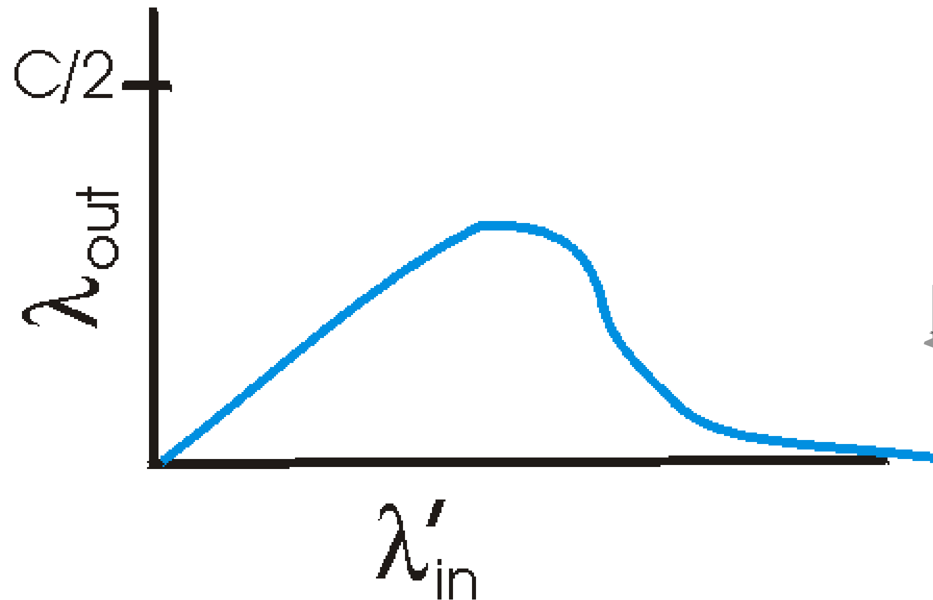
Przyczyny/koszty przeciążenia: scenariusz 3

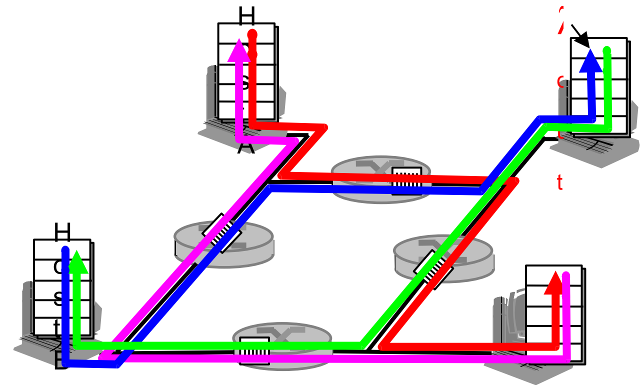
- czterech nadawców
- dłuższe ścieżki
- timeout/retransmisje

Pytanie: co nastąpi, gdy λ_{in} oraz λ'_{in} wzrosną?



Przyczyny/koszty przeciążenia: scenariusz 3





Jeszcze jeden "koszt" przeciążenia:

- gdy pakiet ginie, przepustowość "w górę ścieżki" zużyta na ten pakiet została zmarnowana!

Metody kontroli przeciążenia

Dwa rodzaje metod kontroli przeciążenia:

Kontrola przeciążenia typu koniec-koniec:

- brak bezpośredniej informacji zwrotnej od warstwy sieci
- przeciążenie jest obserwowane na podstawie strat i opóźnień w systemach końcowych
- tą metodą posługuje się TCP

Kontrola przeciążenia z pomocą sieci:

- rutery udostępniają informację zwrotną systemom końcowym
  - pojedynczy bit wskazuje na przeciążenie (SNA, DECbit, TCP/IP ECN, ATM)
  - podawana jest dokładna prędkość, z jaką system końcowy powinien wysyłać

Studium przypadku: kontrola przeciążeń w usłudze ABR sieci ATM

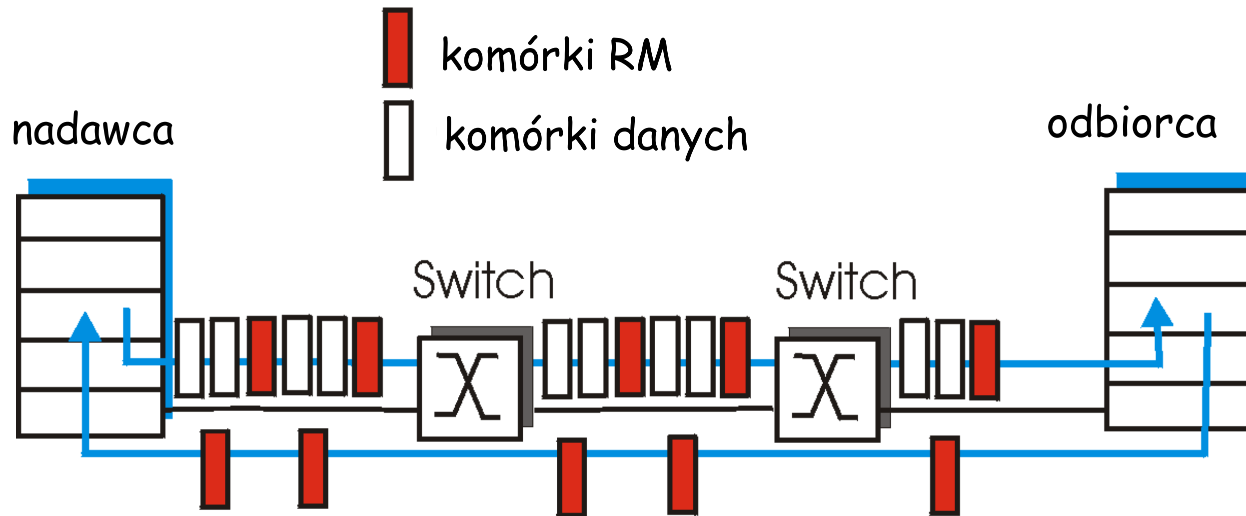
ABR: available bit rate:

- "usługa elastyczna"
- jeśli ścieżka nadawcy jest "niedociążona":
  - nadawca powinien używać dostępną przepustowość
- jeśli ścieżka nadawcy jest przeciążona:
  - nadawca jest ograniczany do minimalnej gwarantowanej przepustowości

Komórki RM (resource management):

- wysyłane przez nadawcę, przeplatane z komórkami danych
- bity w komórce RM ustawiane przez przełączniki sieci ("*z pomocą sieci*")
  - bit NI: nie zwiększaj szybkości (lekkie przeciążenie)
  - bit CI: wskazuje na przeciążenie
- komórki RM zwracane są do nadawcy przez odbiorcę bez zmian

Studium przypadku: kontrola przeciążeń w usłudze ABR sieci ATM



- dwubajtowe pole ER (explicit rate) w komórce RM
  - przeciążony switch może zmniejszyć wartość ER w komórce
  - z tego powodu, nadawca ma minimalną dostępną przepustowość na ścieżce
- bit EFCI w komórkach danych: ustawiany na 1 przez przeciążony switch
  - jeśli komórka danych poprzedzająca komórkę RM ma ustawiony bit EFCI, odbiorca ustawia bit CI w zwróconej komórce RM

Mapa wykładu

- Usługi warstwy transportu
- Multipleksacja i demultipleksacja
- Transport bezpołączeniowy: UDP
- Zasady niezawodnej komunikacji danych
- Transport połączeniowy: TCP
  - struktura segmentu
  - niezawodna komunikacja
  - kontrola przepływu
  - zarządzanie połączeniem
- Mechanizmy kontroli przeciążenia
- Kontrola przeciążenia w TCP

Kontrola przeciążenia w TCP

- metoda koniec-koniec (bez pomocy sieci)
- nadawca ogranicza prędkość transmisji:

  OstatniWysłanyBajt-OstatniPotwierdzonyBajt ≤ RozmiarOkna

- Z grubsza,

$$\text{prędkość} = \frac{\text{RozmiarOkna}}{\text{RTT}} \text{ [Bajt/s]}$$

- **RozmiarOkna** jest zmienny, funkcja obserwowanego przeciążenia sieci

Jak nadawca obserwuje przeciążenie?

- strata = timeout *lub* 3 zduplikowane ACK
- nadawca TCP zmniejsza prędkość (**RozmiarOkna**) po stracie

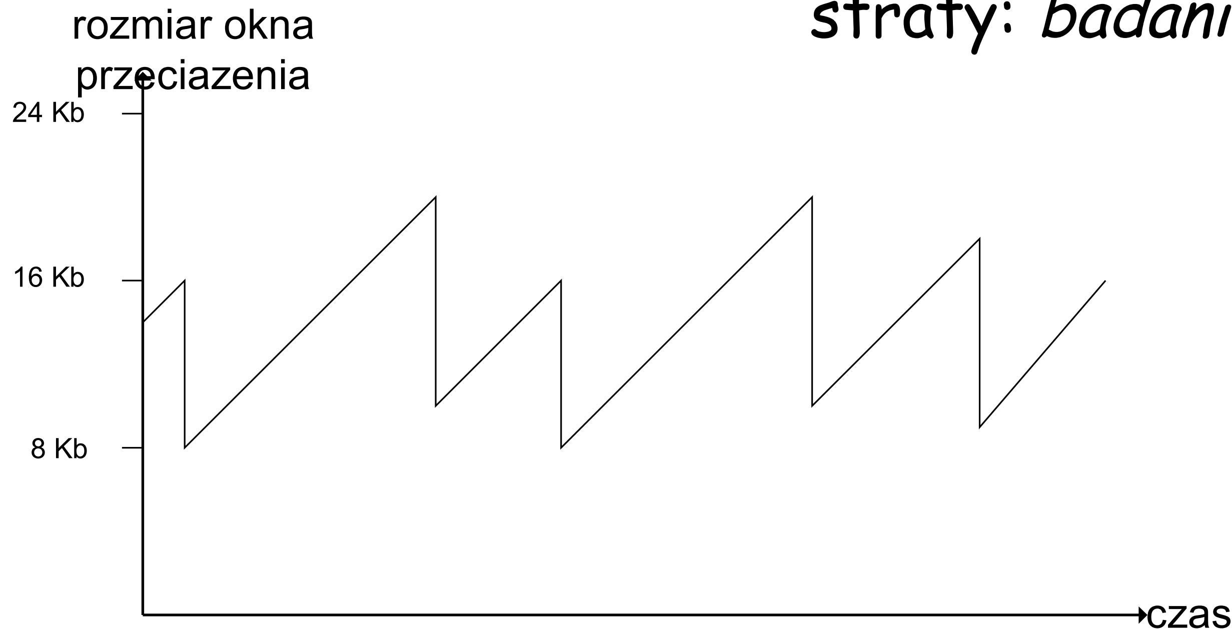
trzy mechanizmy:

- AIMD
- powolny start
- konserwatywne po zdarzeniu timeout

Mechanizm AIMD w TCP

multiplikatywne zmniejszanie: dziel **RozmiarOkna** przez dwa po stracie

addytywne zwiększanie: zwiększ **RozmiarOkna** o 1 rozmiar segmentu po każdym RTT bez straty: *badanie sieci*



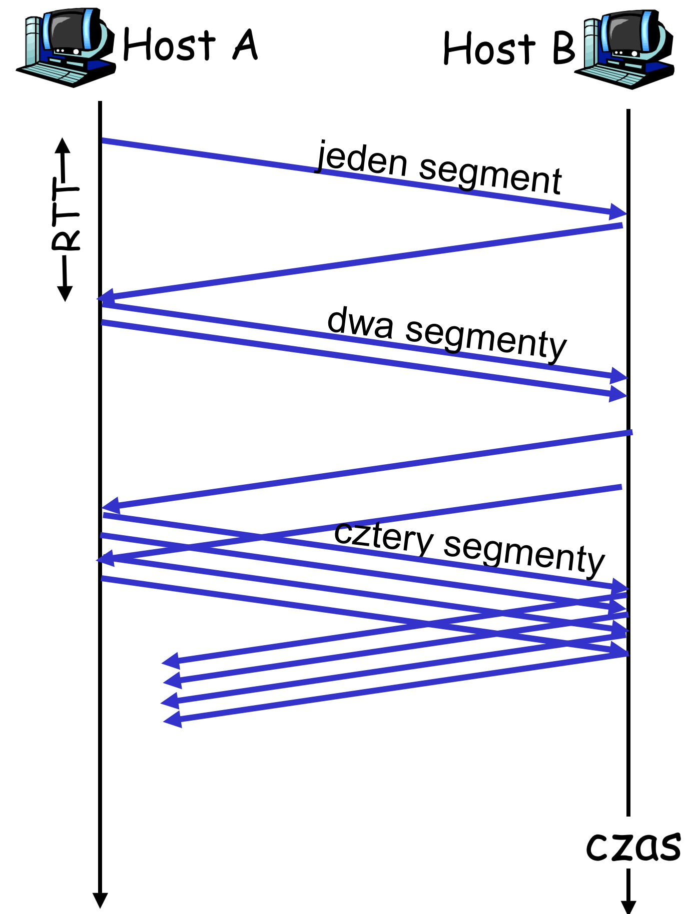
Długotrwałe połączenie TCP

Powolny start TCP

- Po nawiązaniu połączenia, **`RozmiarOkna`** = 1 segment
  - Przykład: Segment = 500 b oraz RTT = 200 ms
  - początkowa prędkość = 2.5 kb/s
- dostępna przepustowość >> rozmiarSegmentu/RTT
  - pożądane jest szybkie przyśpieszenie komunikacji
- Na początku połączenia, rozmiar okna jest mnożony przez dwa aż do wystąpienia pierwszej straty
- Potem zaczyna działać mechanizm AIMD

Powolny start TCP (c.d.)

- Po nawiązaniu połączenia, zwiększaj prędkość wykładniczo do pierwszej straty:
  - podwajaj **RozmiarOkna** co RTT
  - osiągane przez zwiększanie **RozmiarOkna** po otrzymaniu ACK
- <u>Podsumowanie:</u> początkowo, prędkość jest mała, ale szybko przyśpiesza



Udoskonalenie powolnego startu

- Po 3 zduplikowanych ACK:
  - **RozmiarOkna** dzielimy przez dwa
  - okno zwiększane liniowo
- Ale: po zdarzeniu timeout:
  - **RozmiarOkna** ustawiany na 1 segment;
  - okno z początku zwiększane wykładniczo
  - do pewnej wielkości, potem zwiększane liniowo

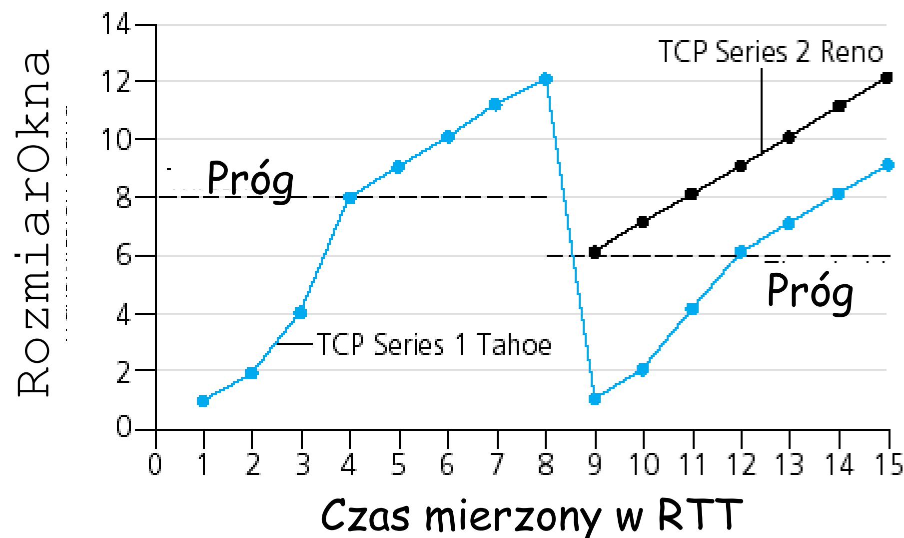
Uzasadnienie:

• 3 zduplikowane ACK wskazują, że sieć dostarcza niektóre segmenty
• timeout przed 3 zduplikowanymi ACK jest "bardziej alarmujący"

Udoskonalenia (c.d.)

Pytanie: Kiedy przestać zwiększać okno wykładniczo, a zacząć zwiększać liniowo?

Odpowiedź: Gdy **RozmiarOkna** osiągnie połowę swojej wartości z przed zdarzenia timeout.



Implementacja:

- Zmienny *Próg*
- Po stracie, Próg jest ustalany na 1/2 wartości RozmiarOkna tuż przed stratą

Podsumowanie: kontrola przeciążenia TCP

- Gdy **RozmiarOkna** poniżej wartości **Próg**, nadawca w stanie powolnego startu, okno rośnie wykładniczo.
- Gdy **RozmiarOkna** powyżej wartości **Próg**, nadawca w stanie unikania przeciążenia, okno rośnie liniowo.
- Po *trzech zduplikowanych ACK*, **Próg = RozmiarOkna/2** oraz **RozmiarOkna = Próg**.
- Po zdarzeniu timeout, **Próg = RozmiarOkna/2** oraz **RozmiarOkna** = 1 segment.

Podsumowanie: kontrola przeciążenia TCP

Zdarzenie	Stan	Akcja nadawcy TCP	Komentarz
Odebranie ACK za niepotwier-dzone dane	Powolny Start (PS)	RozmiarOkna = RozmiarOkna + 1 segment, If (RozmiarOkna > Próg) stan = “Unikanie Przeciążenia”	Po upływie RTT, RozmiarOkna się podwaja
Odebranie ACK za niepotwier-dzone dane	Unikanie Przeciążenia (UP)	RozmiarOkna = RozmiarOkna + (1 segment / RozmiarOkna)	Liniowy wzrost RozmiarOkna o 1 segment po upływie RTT
Strata zaobserwowana przez 3 zduplikowane ACK	PS lub UP	Próg = RozmiarOkna / 2, RozmiarOkna = Próg, Stan = “Unikanie Przeciążenia”	Szybka poprawa przez multiplikatywne zmniejszanie. RozmiarOkna nie będzie mniejszy niż 1 segment.
Timeout	PS lub UP	Próg = RozmiarOkna/2, RozmiarOkna = 1 segment, Stan = “Powolny Start”	Wchodzimy w Powolny Start
Zduplikowany ACK	PS lub UP	Zwiększ licznik ACK dla potwierdzonego segmentu	RozmiarOkna ani Próg nie zmieniają się.

Przepustowość TCP

- Jaka jest średnia przepustowość TCP jako funkcja rozmiaru okna oraz RTT?
  - Ignorujemy powolny start
- Niech W będzie rozmiarem okna w chwili straty.
- Gdy okno ma rozmiar W, przepustowość jest W/RTT
- Zaraz po stracie, okno zmniejszane do W/2, przepustowość jest W/2RTT.
- Średnia przepustowość: $\frac{3}{4}$ W/RTT

Przyszłość TCP

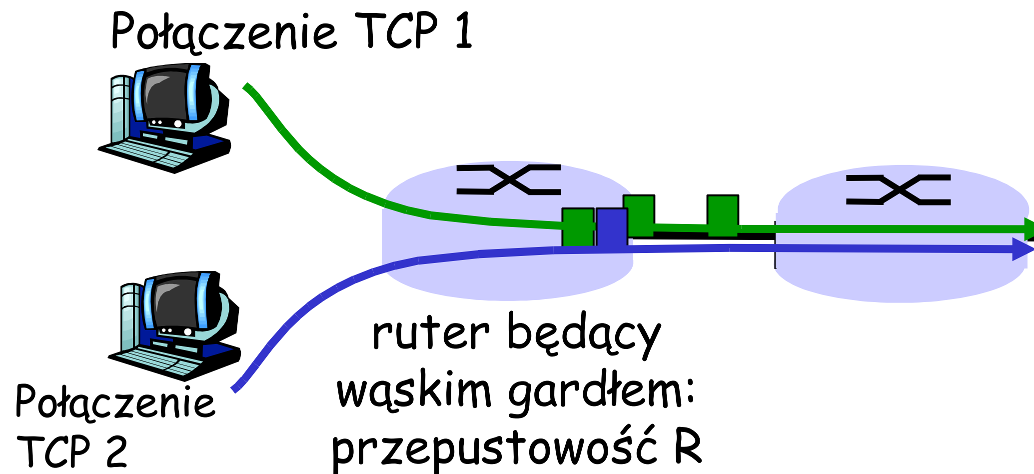
- Przykład: segment 1500 bajtów, RTT 100ms, chcemy przepustowość 10 Gb/s
- Potrzeba rozmiaru okna W = 83 333 segmentów
- Przepustowość jako funkcja częstości strat L:

$$\frac{1.22 \cdot segment}{RTT\sqrt{L}}$$

- □L = $2 \cdot 10^{-10}$ *Bardzo wyśrubowana!*
- Potrzeba nowych wersji TCP dla bardzo szybkich sieci!

Sprawiedliwość TCP

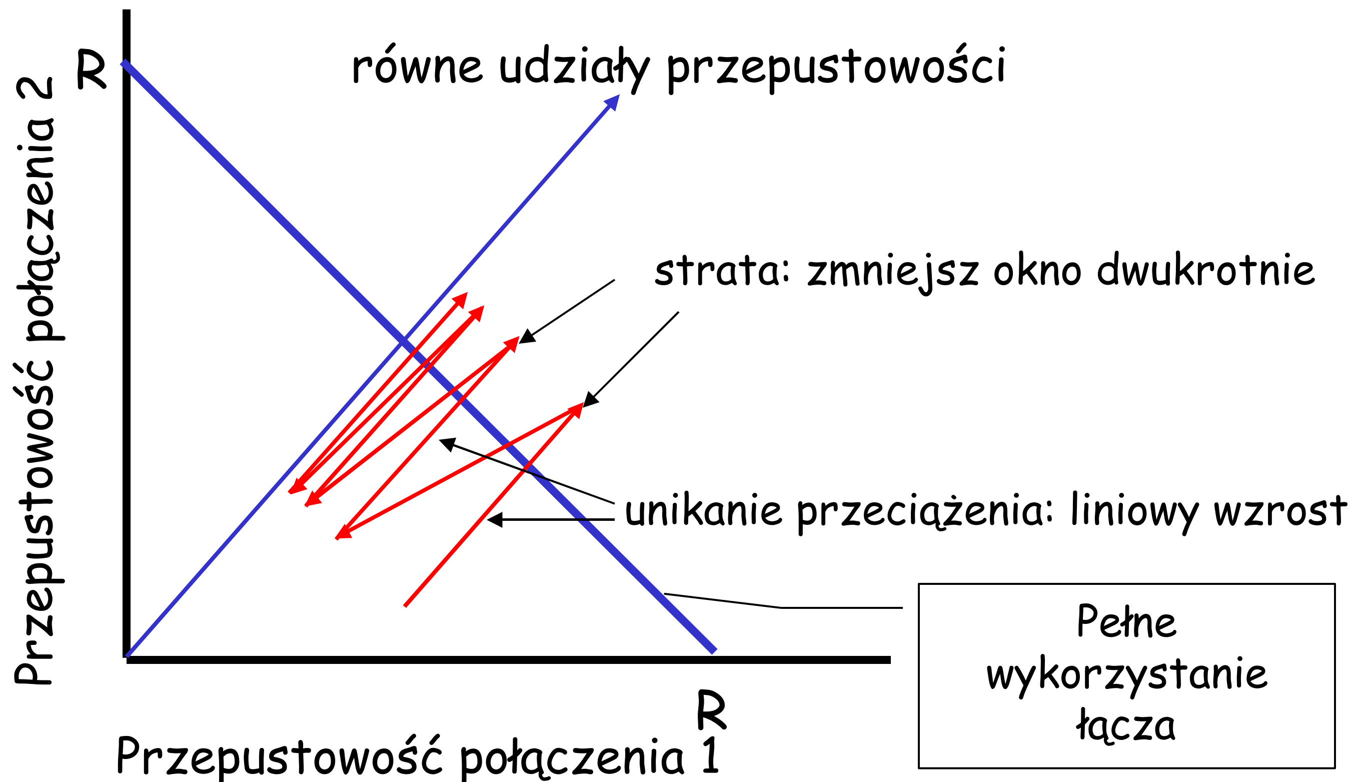
Sprawiedliwy cel: jeśli K połączeń TCP dzieli to samo łącze o przepustowości R będące wąskim gardłem, każde połączenie powinno mieć średnią przepustowość R/K



Dlaczego TCP jest sprawiedliwe?

Dwa konkurujące połączenia:

- addytywne zwiększanie daje nachylenie 1, gdy rośnie przepustowość
- multiplikatywne zmniejszanie zmniejsza przepustowość proporcjonalnie



Więcej o sprawiedliwości

Sprawiedliwość i UDP

- Komunikacja strumieniowa nie używa TCP
  - nie chce zmieniać prędkości nadawania zgodnie z kontrolą przeciążeń
- W zamian używa UDP:
  - śle audio/wideo ze stałą prędkością, toleruje straty pakietów
- Obszar badań: Komunikacja strumieniowa "TCP friendly"

Sprawiedliwość i równoległe połączenia TCP

- nic nie powstrzyma aplikacji przed nawiązaniem równoległych połączeń pomiędzy 2 hostami.
- Robią tak przeglądarki WWW
- Przykład: łącze o przepustowości R obsługuje 9 połączeń;
  - nowa aplikacja prosi o 1 połączenie TCP, dostaje przepustowość R/10
  - nowa aplikacja prosi o 11 połączeń TCP, dostaje przepustowość R/2 !

Podsumowanie warstwy transportu

- mechanizmy usług warstwy transportu:
  - multipleksacja, demultipleksacja
  - niezawodna komunikacja
  - kontrola przepływu
  - kontrola przeciążenia
- w Internecie:
  - UDP
  - TCP

Co dalej:

- opuszczamy "brzeg" sieci (warstwy aplikacji i transportu)
- wchodzimy w "szkielet" sieci