
Transport

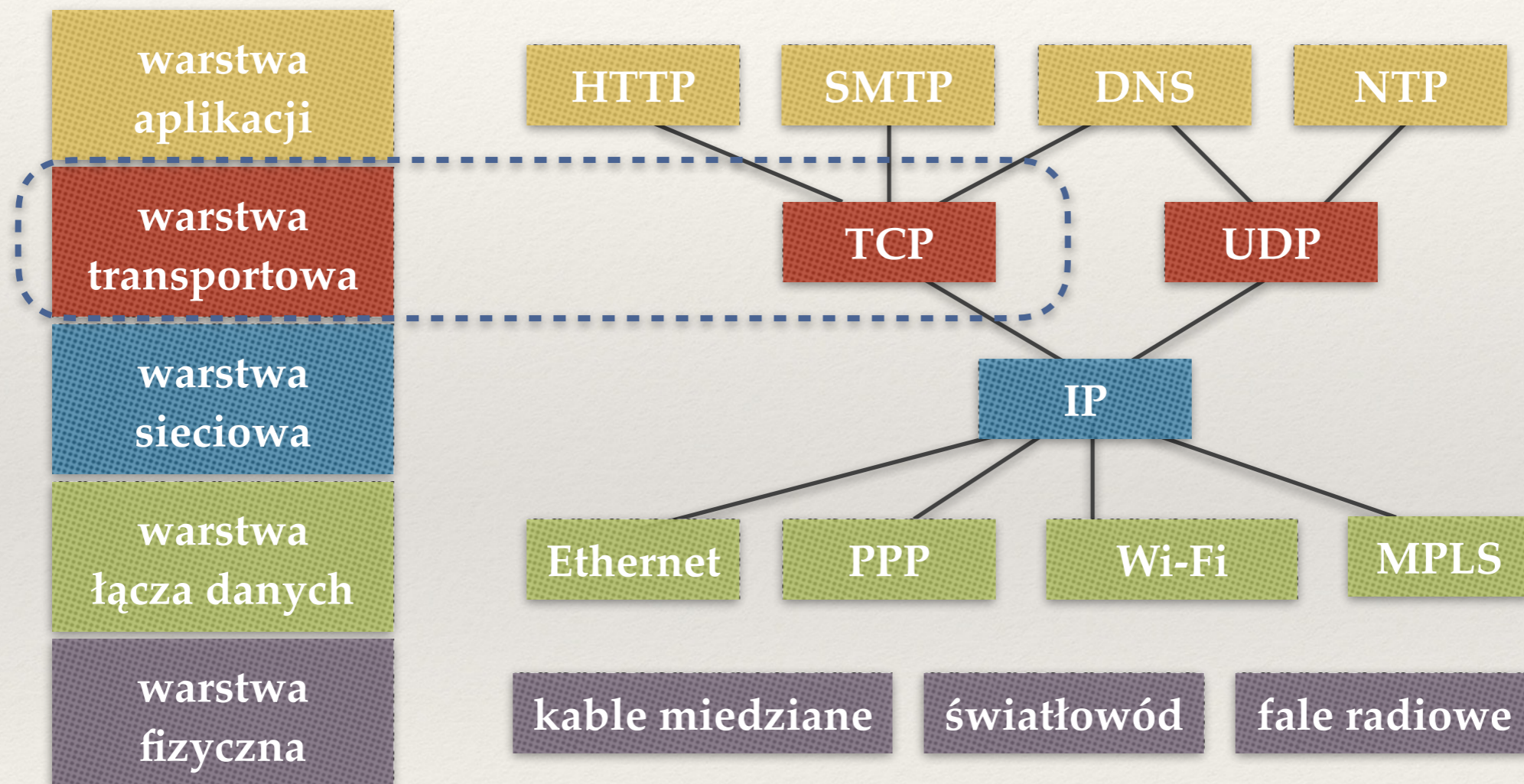
część 1: niezawodny transport

Sieci komputerowe

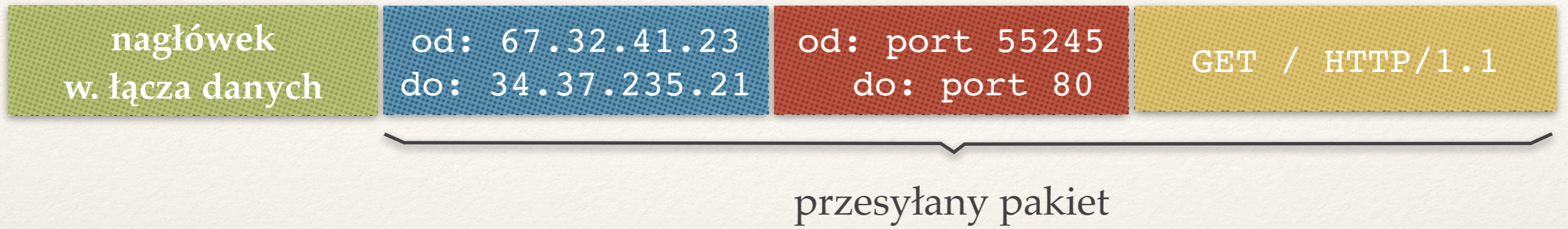
Wykład 6

Marcin Bieńkowski

Protokoły w Internecie



Internetowy model warstwowy



Globalne dostarczanie danych pomiędzy aplikacjami.

Globalne dostarczanie danych pomiędzy komputerami.

Lokalne dostarczanie danych pomiędzy komputerami.

Porty

❖ Port:

- ♦ liczba 16-bitowa;
- ♦ identyfikuje aplikację wewnątrz danego komputera
→ multipleksowanie wielu strumieni danych w jednym ciągu pakietów.

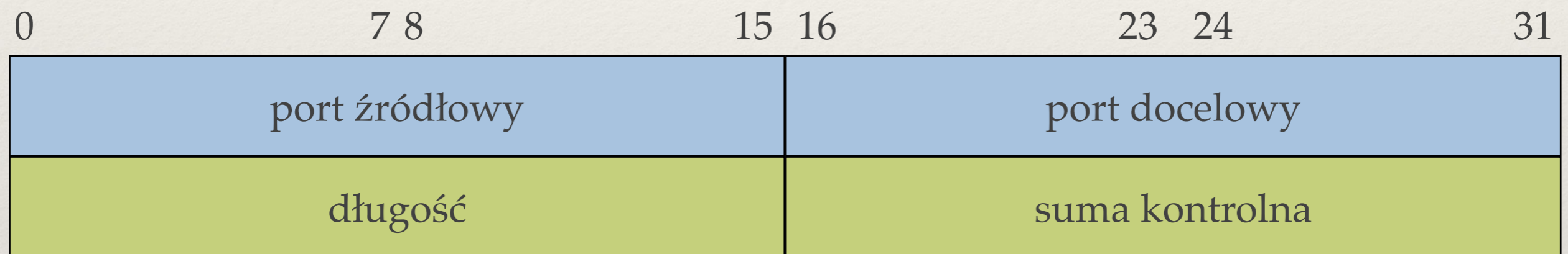
❖ W nagłówku warstwy transportowej znajduje się m.in.:

- ♦ port źródłowy;
- ♦ port docelowy.

Zawodny transport

UDP (User Data Protocol)

Najprostszy protokół warstwy transportowej.



Gwarancje UDP (1)

- ❖ Takie same jak IP.
 - ♦ Tylko zasada dołożenia wszelkich starań (*best effort*).

- ❖ Pakiety mogą zostać:
 - ♦ uszkodzone,
 - ♦ zgubione,
 - ♦ opóźnione,
 - ♦ zamienione (kolejność),
 - ♦ zduplikowane (przez wyższe lub niższe warstwy).

**I'D TELL YOU
A JOKE ABOUT
UDP
BUT YOU MIGHT
NOT GET IT**

Gwarancje UDP (2)

- ❖ Czy te negatywne wydarzenia są częste?
- ❖ Czy mogą wydarzyć się również w przypadku lokalnie przesyłanych pakietów?

Gwarancje UDP (2)

- ❖ Czy te negatywne wydarzenia są częste?
- ❖ Czy mogą wydarzyć się również w przypadku lokalnie przesyłanych pakietów?

`udp_client_flood.c`
kod (z obsługą błędów) na stronie wykładu

demonstracja

Gwarancje UDP (2)

- ❖ Czy te negatywne wydarzenia są częste?
- ❖ Czy mogą wydarzyć się również w przypadku lokalnie przesyłanych pakietów?

`udp_client_flood.c`
kod (z obsługą błędów) na stronie wykładu

demonstracja

- ❖ **Kontrola przepływu:**

- ◆ Nadawca powinien dostosowywać prędkość transmisji do możliwości odbiorcy.
- ◆ Nie wysyłać danych, których odbiorca nie ma gdzie przechować.

Gdzie wykorzystujemy

Zawodny transport:

- ❖ Dla małych ilości danych (DNS, DHCP).
- ❖ Proste urządzenia (TFTP do aktualizacji firmware).
- ❖ Szybka reakcja (gry, interaktywny streaming audio (*voice over IP*), interaktywny streaming video).
- ❖ Pełna kontrola nad danymi (NFS).

Niezawodny transport:

- ❖ Duże ilości danych (HTTP(S), nieinteraktywny streaming audio / video).

Niezawodny transport: segmentacja

Segmentacja

Niezawodne przesyłanie ciągu bajtów:

- ❖ Zadanie warstwy transportowej.
- ❖ Wymaga dzielenia ciągu bajtów na **segmenty**.
 - ◆ W UDP użytkownik musi to robić sam.

Segmentacja

Niezawodne przesyłanie ciągu bajtów:

- ❖ Zadanie warstwy transportowej.
- ❖ Wymaga dzielenia ciągu bajtów na **segmenty**.
 - ◆ W UDP użytkownik musi to robić sam.
- ❖ **Dlaczego nie przesyłamy wszystkiego w jednym segmencie?**

Słowo o nazewnictwie

warstwa
transportowa

datagramy (gdy użytkownik sam dzieli na części, np. UDP)
segmenty (gdy warstwa dzieli na części, np. TCP)

warstwa
sieciowa

pakiety

warstwa
łącza danych

ramki

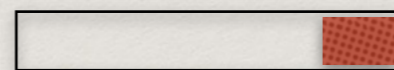
Niekonsekwentnie używane nazwy.

- ❖ Powszechnie stosowane „datagramy IP”.
- ❖ „Segment TCP” często oznacza same dane TCP (bez nagłówka TCP), np. w definicji MSS (*maximum segment size*).

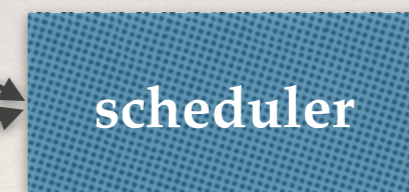
Dlaczego segmentacja jest potrzebna?

- ❖ **Maksymalny rozmiar segmentu (MSS).**
 - ♦ $MSS = MTU - \text{rozmiar nagłówka IP} - \text{rozmiar nagłówka TCP}$.
- ❖ **Dlaczego nie zwiększymy MTU (i MSS)?**
 - ♦ Większa szansa na zakłócenia (sieci bezprzewodowe).
 - ♦ Duże pakiety: problem w szeregowaniu w kolejce wyjściowej routera (małe pakiety mają duże opóźnienie).

strumień o wysokim priorytecie

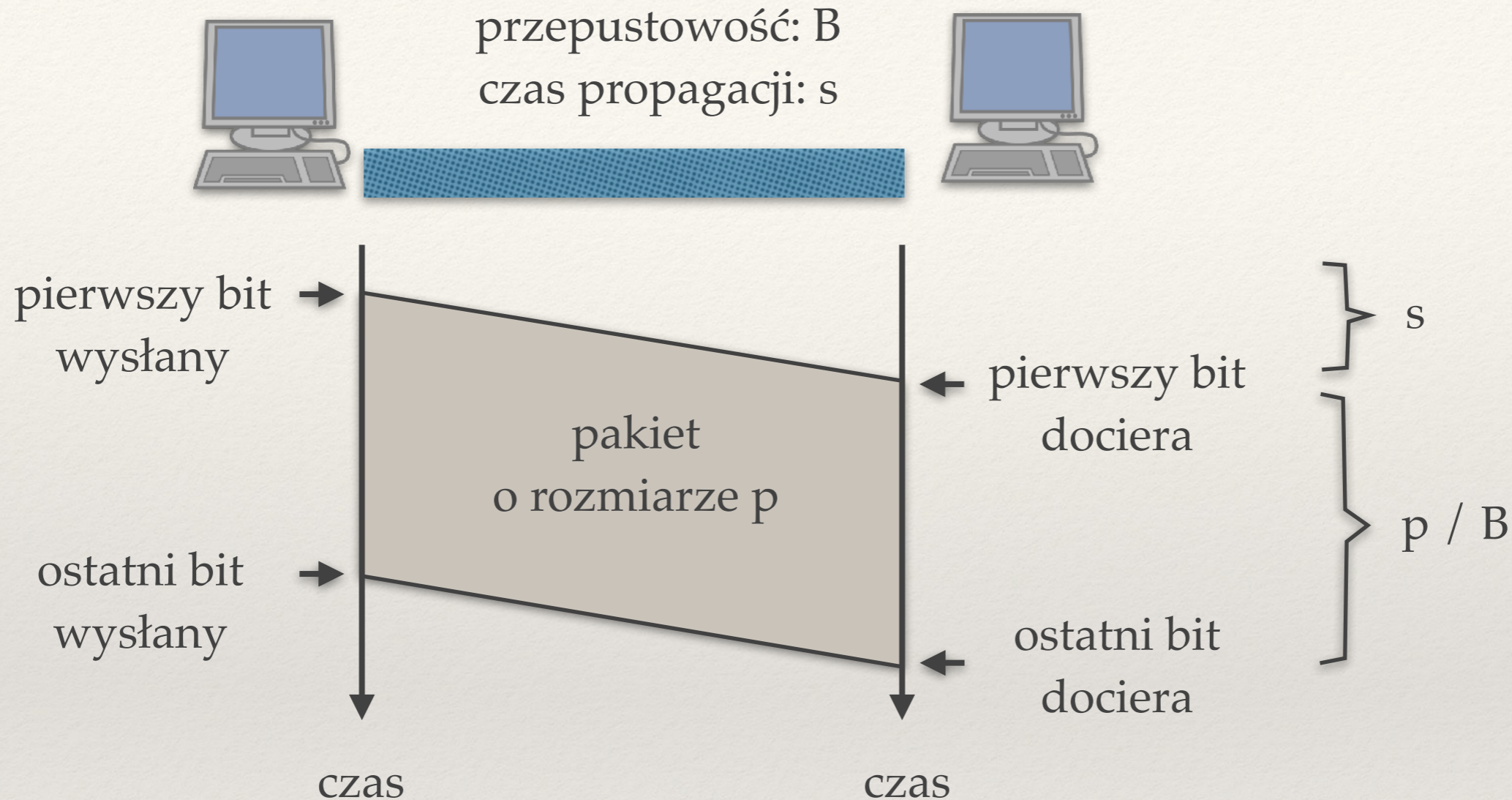


strumień o niskim priorytecie



- ♦ Główna przyczyna: **mniejsze opóźnienie przy długich ścieżkach.**

Opóźnienie pakietu na łączu (przypomnienie)



Opóźnienie na pojedynczym łączu = $s + p / B$.

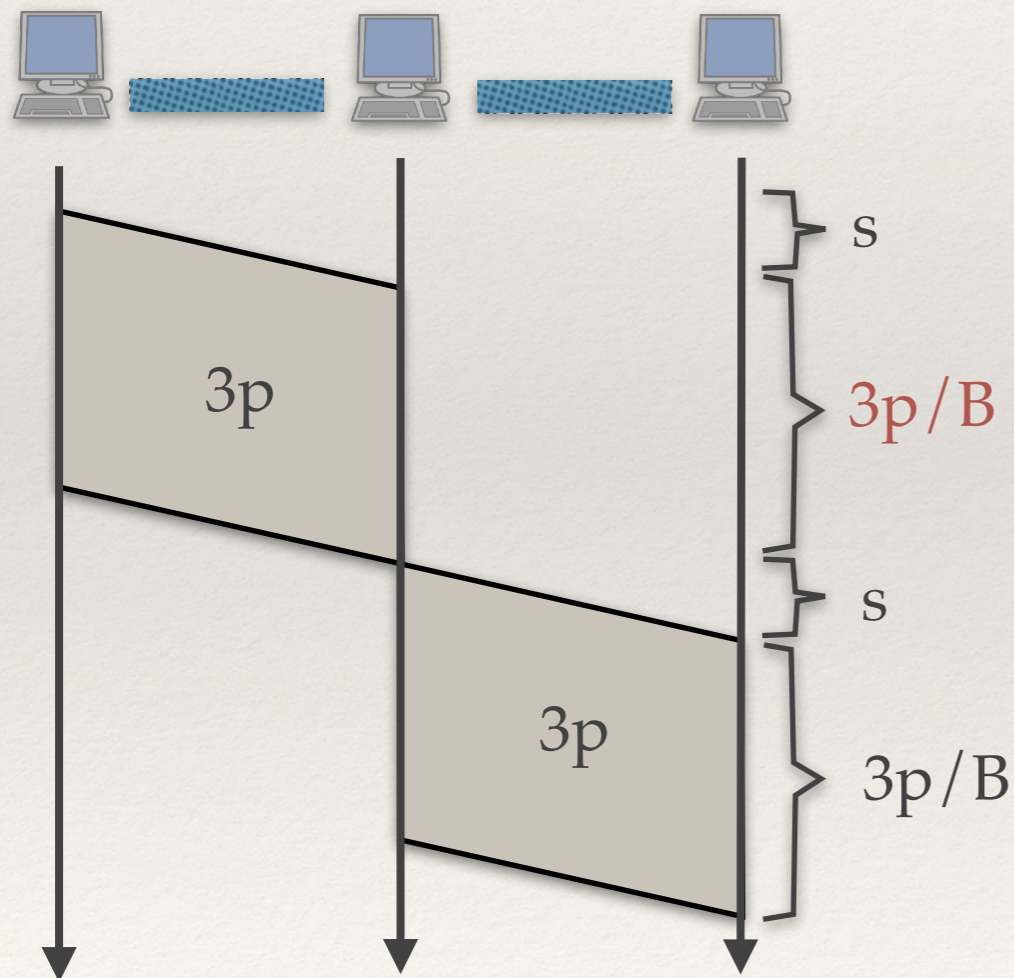
- ❖ Ignorujemy czas kolejowania pakietu w buforze.

Opóźnienie pakietu na ścieżce

Dla dowolnego łącza: przepustowość = B , czas propagacji = s .

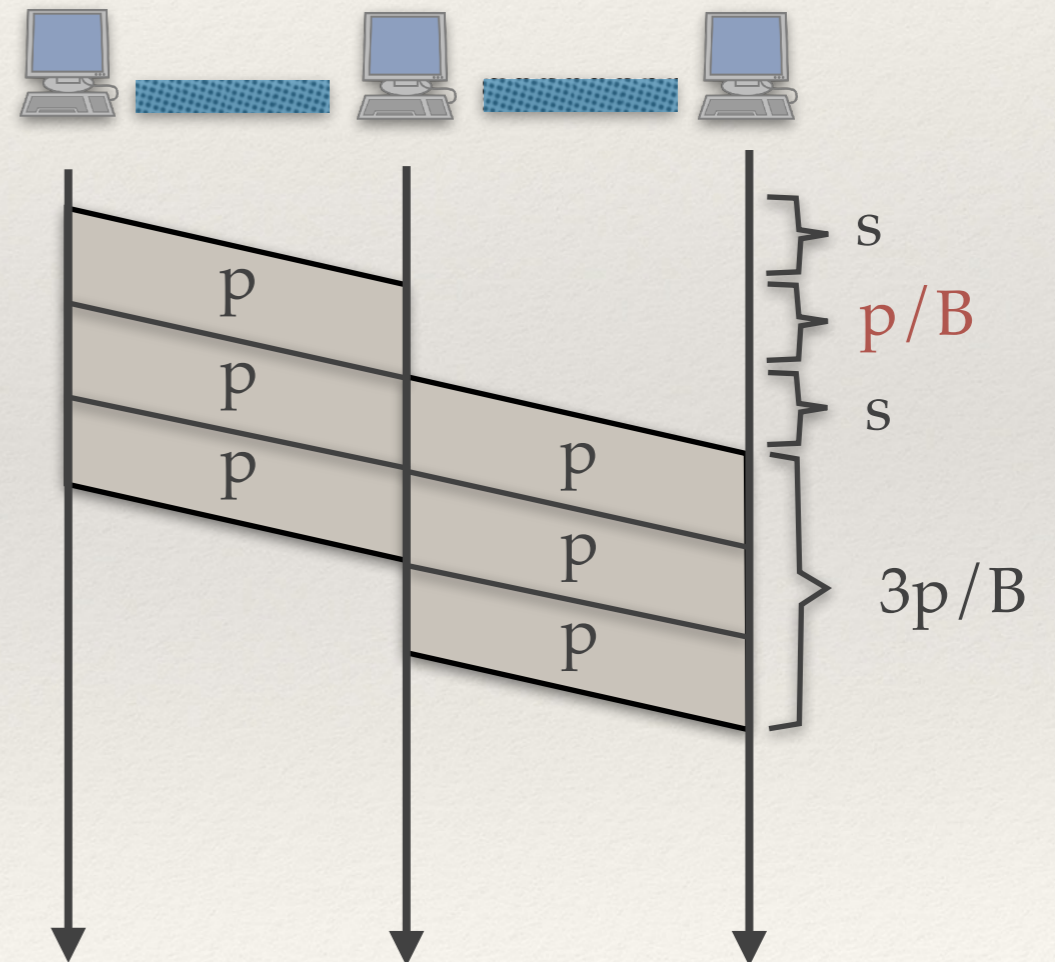
pakiet o rozmiarze $3p$

opóźnienie = $2s + 6(p/B)$



3 pakiety o rozmiarze p

opóźnienie = $2s + 4(p/B)$



Niezawodny transport: ARQ

ARQ = Automatic Repeat reQuest

- ❖ Zajmiemy się niezawodną transmisją jednokierunkową.
 - ✦ Od nadawcy do odbiorcy.
 - ✦ W drugą stronę identyczny mechanizm.
- ❖ ARQ
 - ✦ Wysyłanie „do skutku“ (do otrzymania potwierdzenia).
 - ✦ Niezawodny transport na bazie zawodnej usługi przesyłania pakietów.
 - ✦ Odbiorca może wysyłać informacje zwrotne (otrzymałem pakiet / zwolnij / przyspiesz / ...)
- ❖ Założymy, że strumień bajtów jest już posegmentowany.

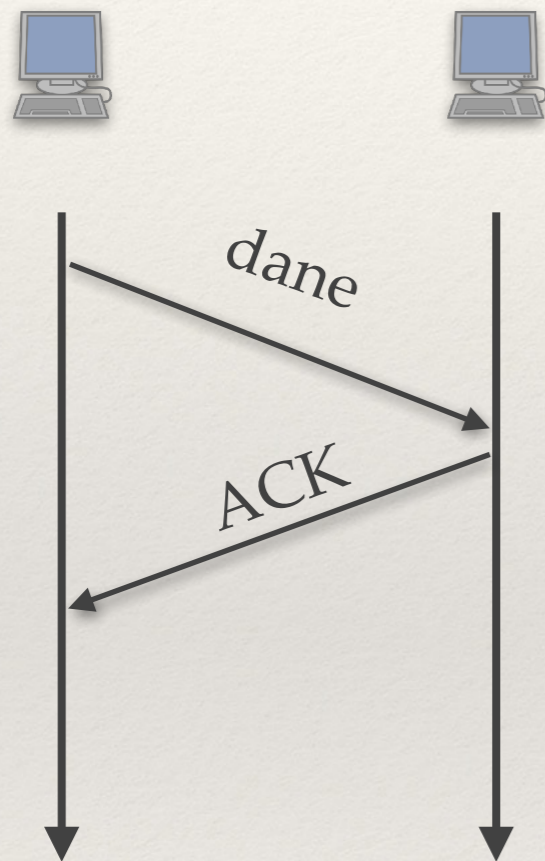
Dostępne mechanizmy

- ❖ **Sumy kontrolne:** możemy wykrywać, czy segment został uszkodzony.
- ❖ **Potwierdzenia (ACK):** małe pakiety kontrolne potwierdzające otrzymanie danego segmentu.
- ❖ **Timeout (przekroczenie czasu oczekiwania):**
 - ♦ jeśli nie otrzymamy potwierdzenia segmentu przez RTO (*retransmission timeout*);
 - ♦ RTO to dynamicznie ustalana wartość (funkcja bieżącego RTT).

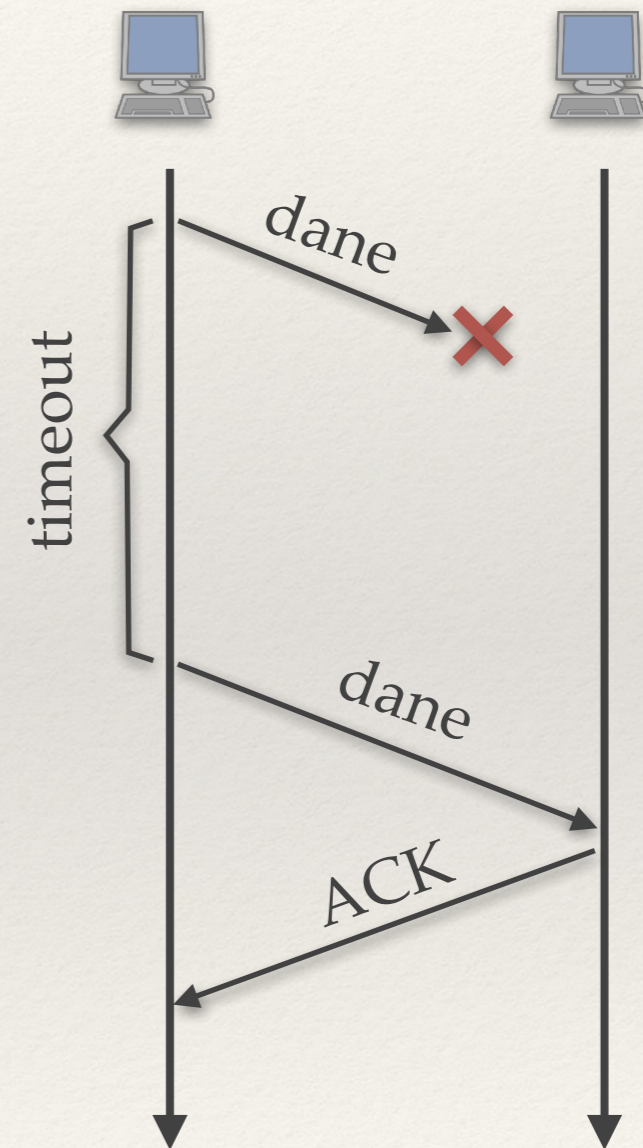
ARQ: Stop-and-Wait

Stop-and-wait: typowe wykonania

1. bez utraty pakietów

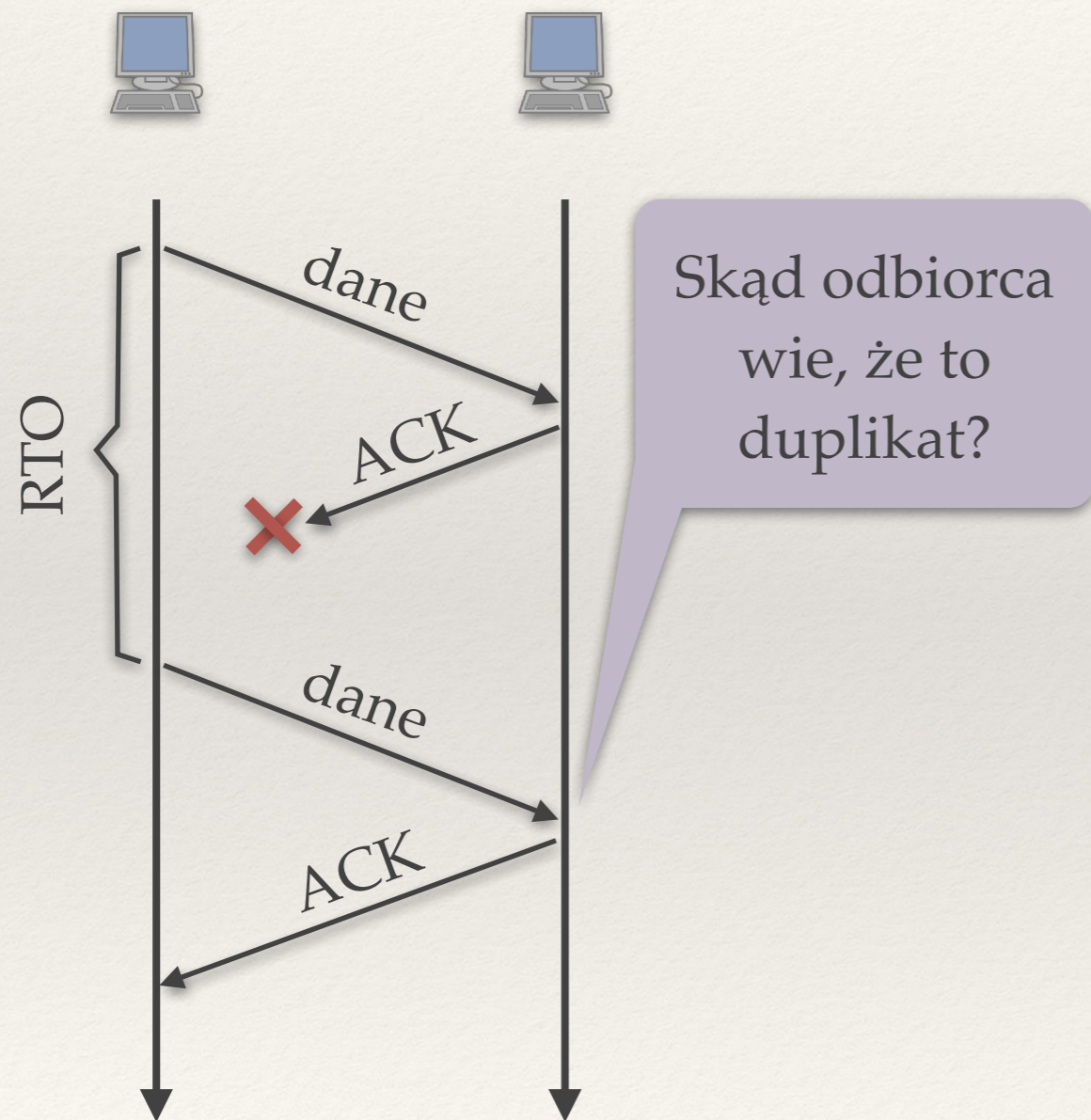


2. utrata danych



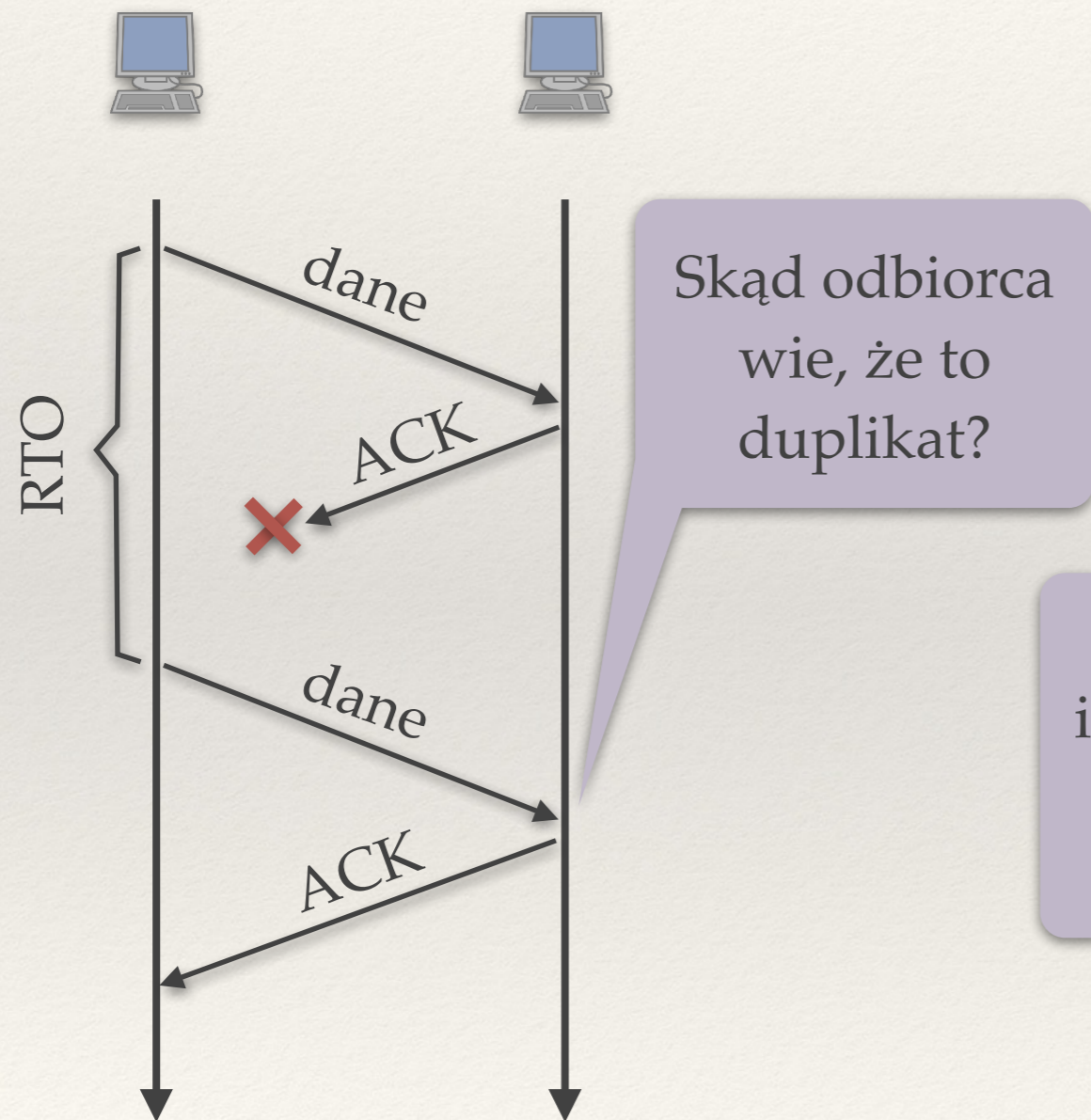
Stop-and-wait: problemy

3. utrata ACK

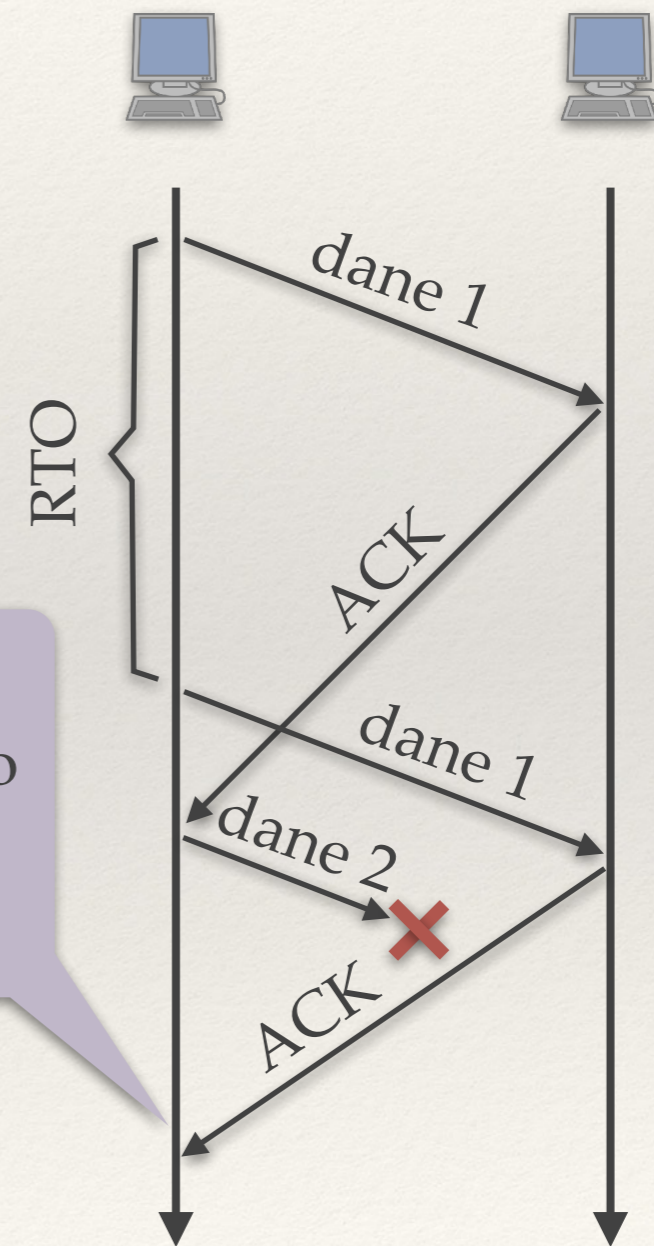


Stop-and-wait: problemy

3. utrata ACK

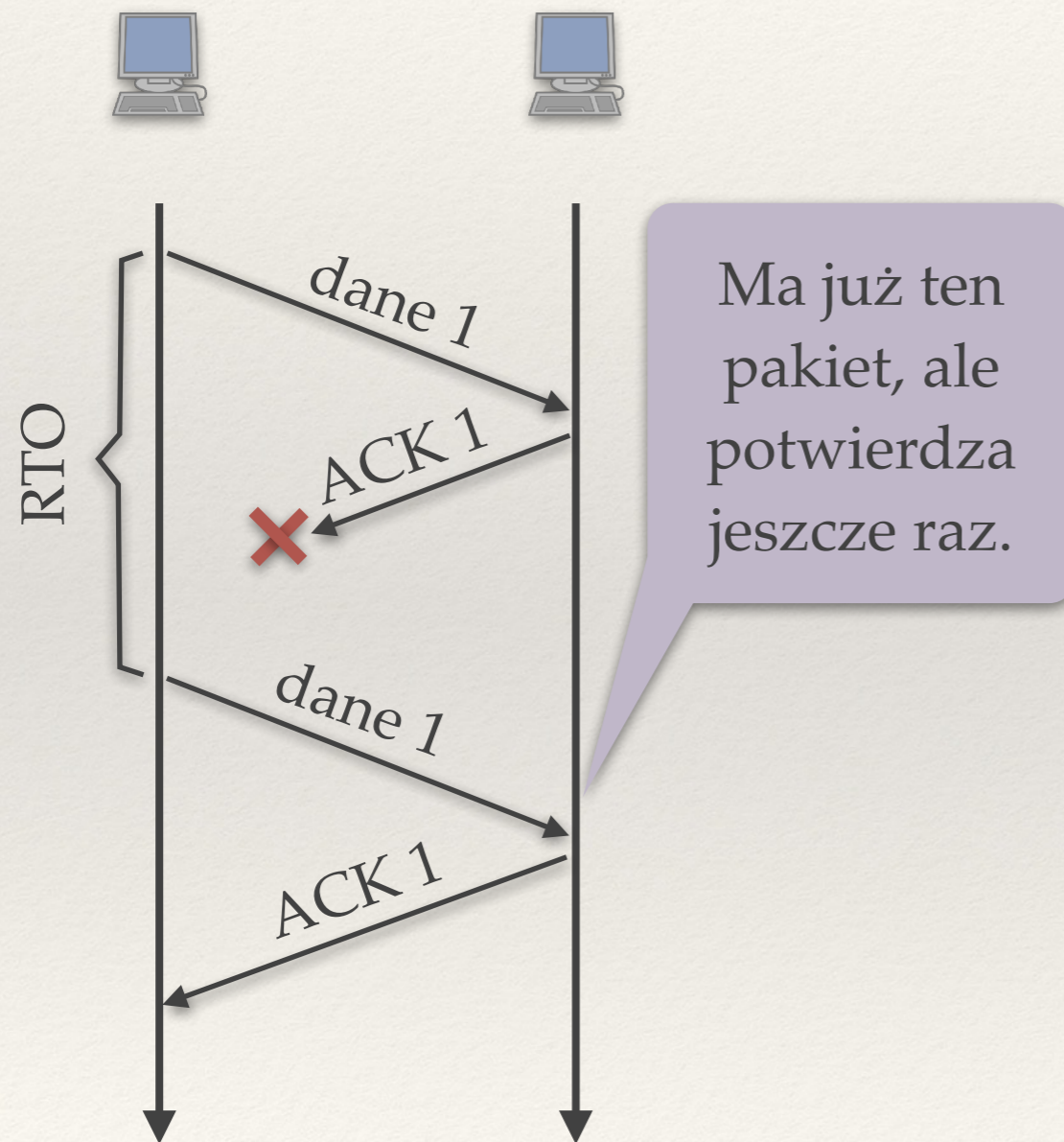


4. opóźnienie ACK + utrata danych



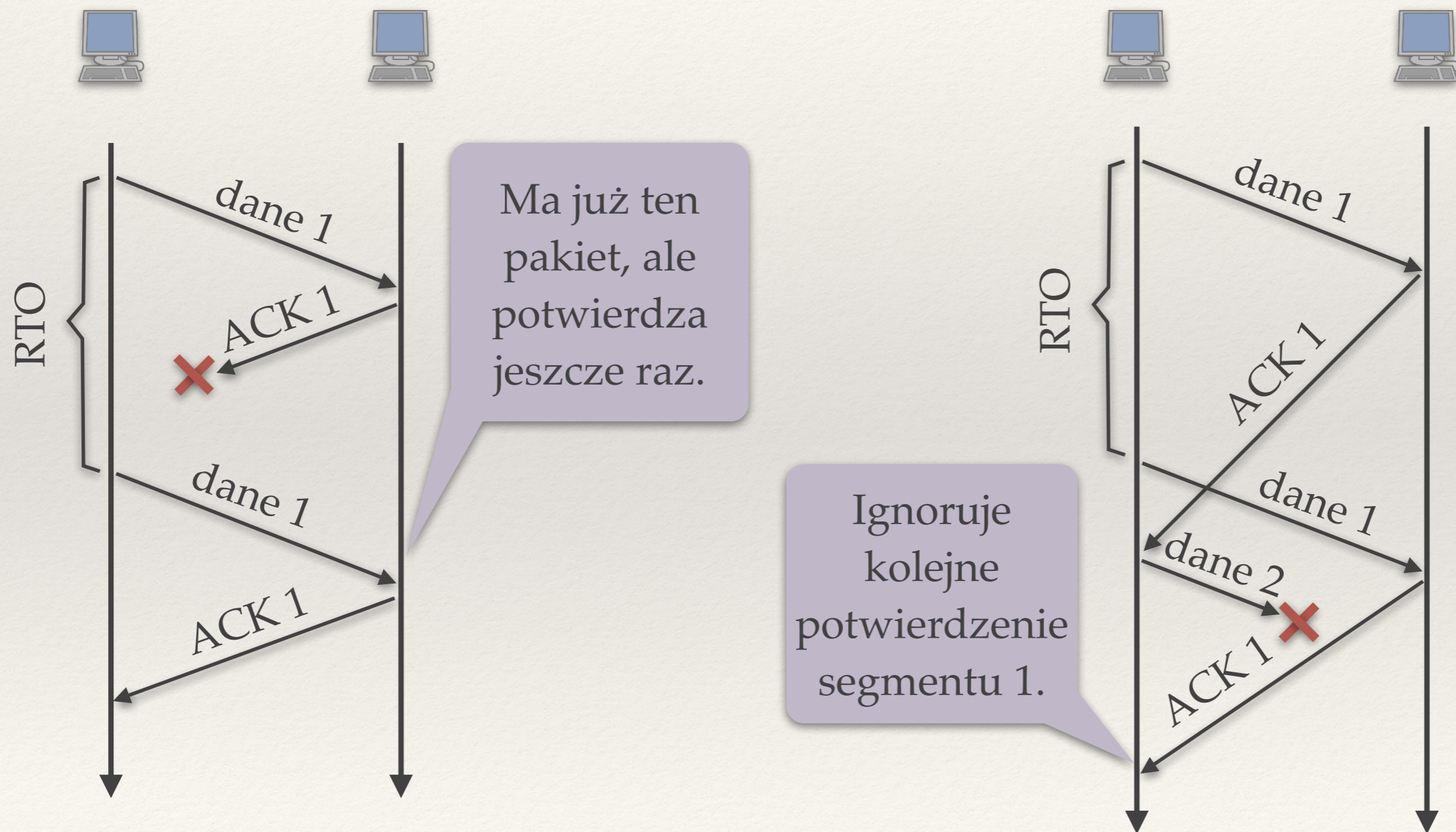
Dodajemy numery sekwencyjne

- ❖ Każdy segment jest numerowany.
- ❖ ACK zawiera numer potwierdzonego segmentu.



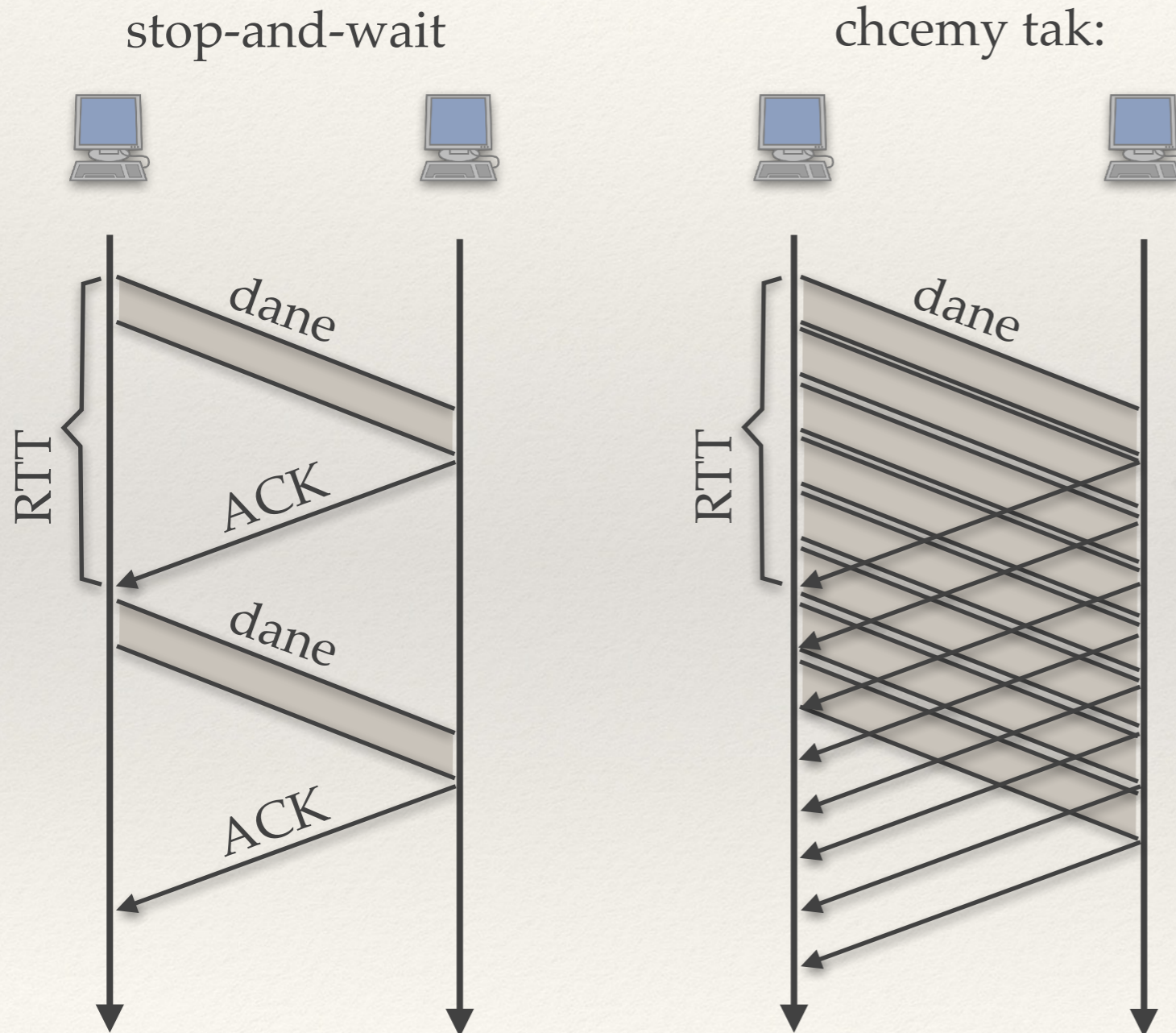
Dodajemy numery sekwencyjne

- ❖ Każdy segment jest numerowany.
- ❖ ACK zawiera numer potwierdzonego segmentu.



Stop-and-wait + numery sekwencyjne

Działa, ale przy długich łączach o dużej przepustowości wykorzystuje ułamek ich możliwości.



Nadawca powinien móc nadawać bez czekania na ACK co najmniej przez RTT.

ARQ: Okno przesuwne

Okno przesuwne nadawcy

- ❖ Klasa algorytmów wykorzystująca przesuwne okno nadawcy.
 - ♦ Cechy wspólne.
 - ♦ Algorytm Go-Back-N.
 - ♦ Algorytm wybiórczego powtarzania (SR, *selective repeat*).
- ❖ Implementacja w TCP.

Cechy wspólne: co robi nadawca

- ❖ SWS (*sender window size*) = rozmiar okna nadawcy.
- ❖ Niezmiennik: segmenty od 1 do LAR są potwierdzone, a LAR+1 nie.
- ❖ Otrzymanie ACK dla segmentu może przesunąć okno w prawo.




Cechy wspólne: co robi nadawca

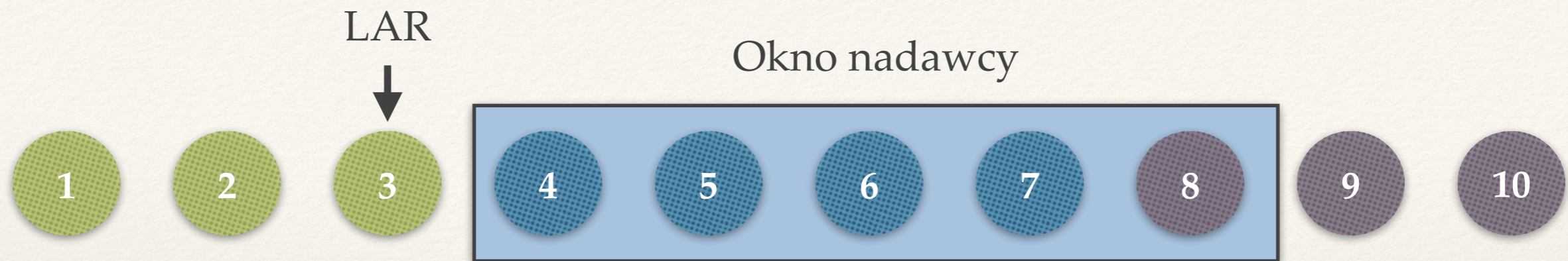
- ❖ SWS (*sender window size*) = rozmiar okna nadawcy.
- ❖ Niezmiennik: segmenty od 1 do LAR są potwierdzone, a LAR+1 nie.
- ❖ Otrzymanie ACK dla segmentu może przesunąć okno w prawo.
- ❖ Nadawca wysyła niewysłane segmenty z okna, jeśli są.
- ❖ W zależności od algorytmu czasem też ponawia niepotwierdzone.
- ❖ Dla każdego segmentu X nadawca pamięta $sent(X)$ = ostatni czas jego wysłania.



Okno przesuwne nadawcy

- ❖ Klasa algorytmów wykorzystująca przesuwne okno nadawcy.
 - ♦ Cechy wspólne. 
 - ♦ Algorytm Go-Back-N.
 - ♦ Algorytm wybiórczego powtarzania (*selective repeat*).
- ❖ Implementacja w TCP.

Go-Back-N: co robi odbiorca



Algorytm dla odbiorcy:

- ❖ Odbiorca ma segmenty do P włącznie.
- ❖ Odbiorca akceptuje otrzymany segment S , tylko jeśli $S = P + 1$.
- ❖ Odbiorca odsyła ACK, jeśli $S \leq P + 1$.

Co nadawca wnioskuje z otrzymaniem ACK dla segmentu S :

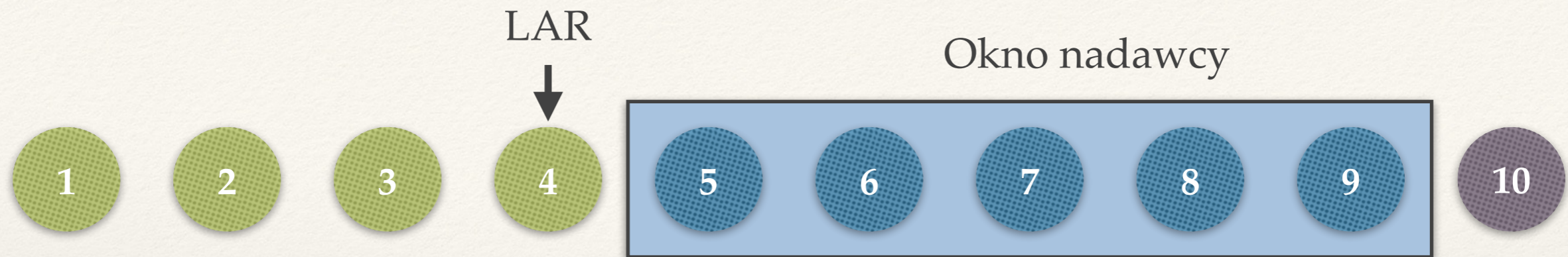
- ❖ Odbiorca ma wszystko do S włącznie \rightarrow można zaktualizować $LAR := S$

Go-Back-N: co robi nadawca





- ❖ Niezmiennik w Go-Back-N: okno nadawcy zawiera wyłącznie niepotwierdzone segmenty.
- ❖ W czasie $sent(LAR+1) + RTO$ wysyłamy ponownie wszystkie segmenty z okna.

Go-Back-N: co robi nadawca



- ❖ Niezmiennik w Go-Back-N: okno nadawcy zawiera wyłącznie niepotwierdzone segmenty.
- ❖ W czasie $sent(LAR+1) + RTO$ wysyłamy ponownie wszystkie segmenty z okna.
- ❖ Prosta implementacja nadawcy, trywialna odbiorcy.
- ❖ Jeśli $SWS = 1$, to Go-Back-N = Stop-and-Wait.

Okno przesuwne nadawcy

- ❖ Klasa algorytmów wykorzystująca przesuwne okno nadawcy.
 - ♦ Cechy wspólne. 
 - ♦ Algorytm Go-Back-N. 
 - ♦ Algorytm wybiórczego powtarzania (*selective repeat*).
- ❖ Implementacja w TCP.

SR: odbiorca też ma okno

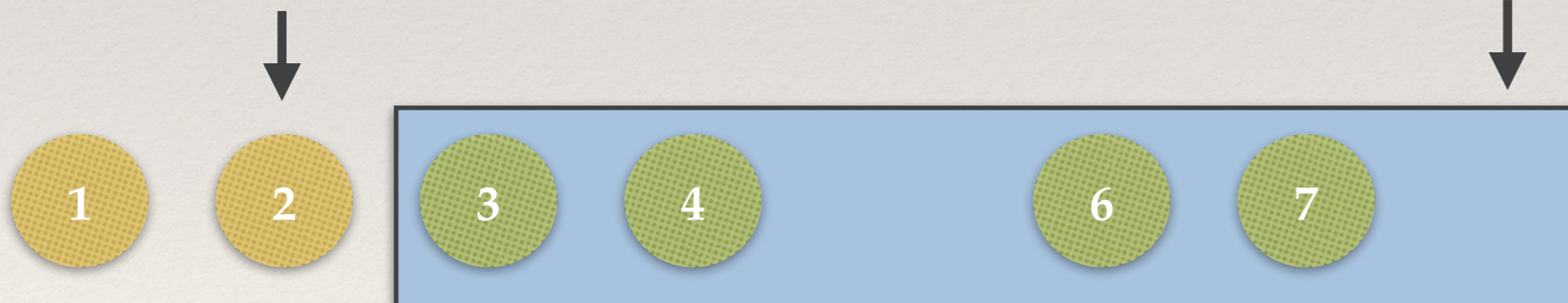
- ❖ RWS (*receiver window size*) = rozmiar okna odbiorcy.
- ❖ Okno to bufor na:
 - ◆ segmenty (jeszcze) nieprzeczytane,
 - ◆ segmenty, które przyszły „poza kolejnością”.

SR: odbiorca też ma okno

- ❖ RWS (*receiver window size*) = rozmiar okna odbiorcy.
- ❖ Okno to bufor na:
 - ♦ segmenty (jeszcze) nieprzeczytane,
 - ♦ segmenty, które przyszły „poza kolejnością”.
- ❖ Odbiorca potwierdza segment S jeśli $S \leq \text{LSR} + \text{RWS}$.

LSR (*last segment read*)

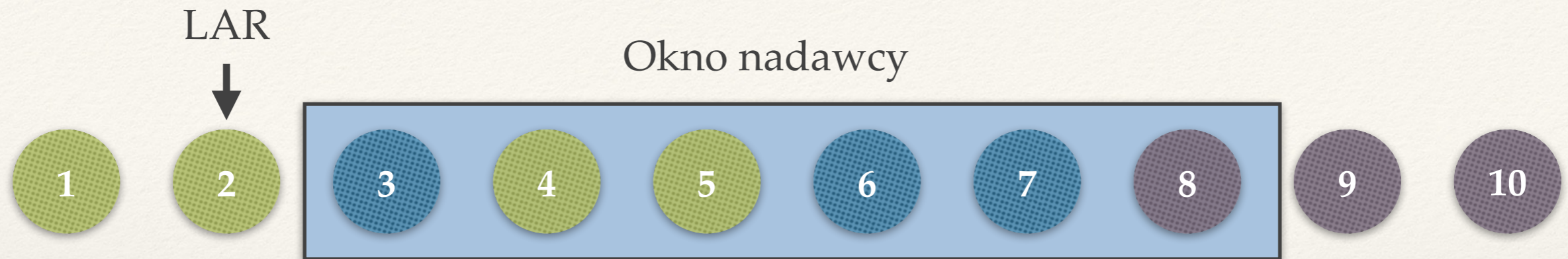
LSR + RWS



● odebrane, potwierdzone, przeczytane przez aplikację za pomocą `read()`

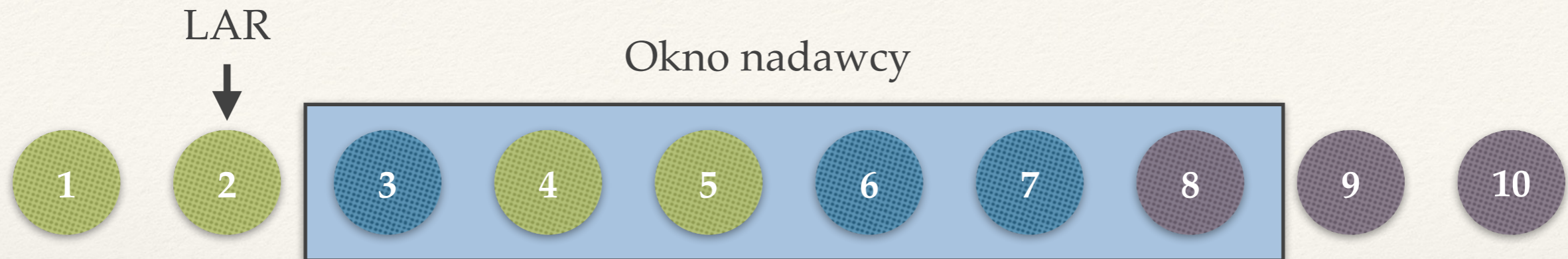
● odebrane (i potwierdzone), ale nieprzeczytane przez aplikację

SR: co robi nadawca



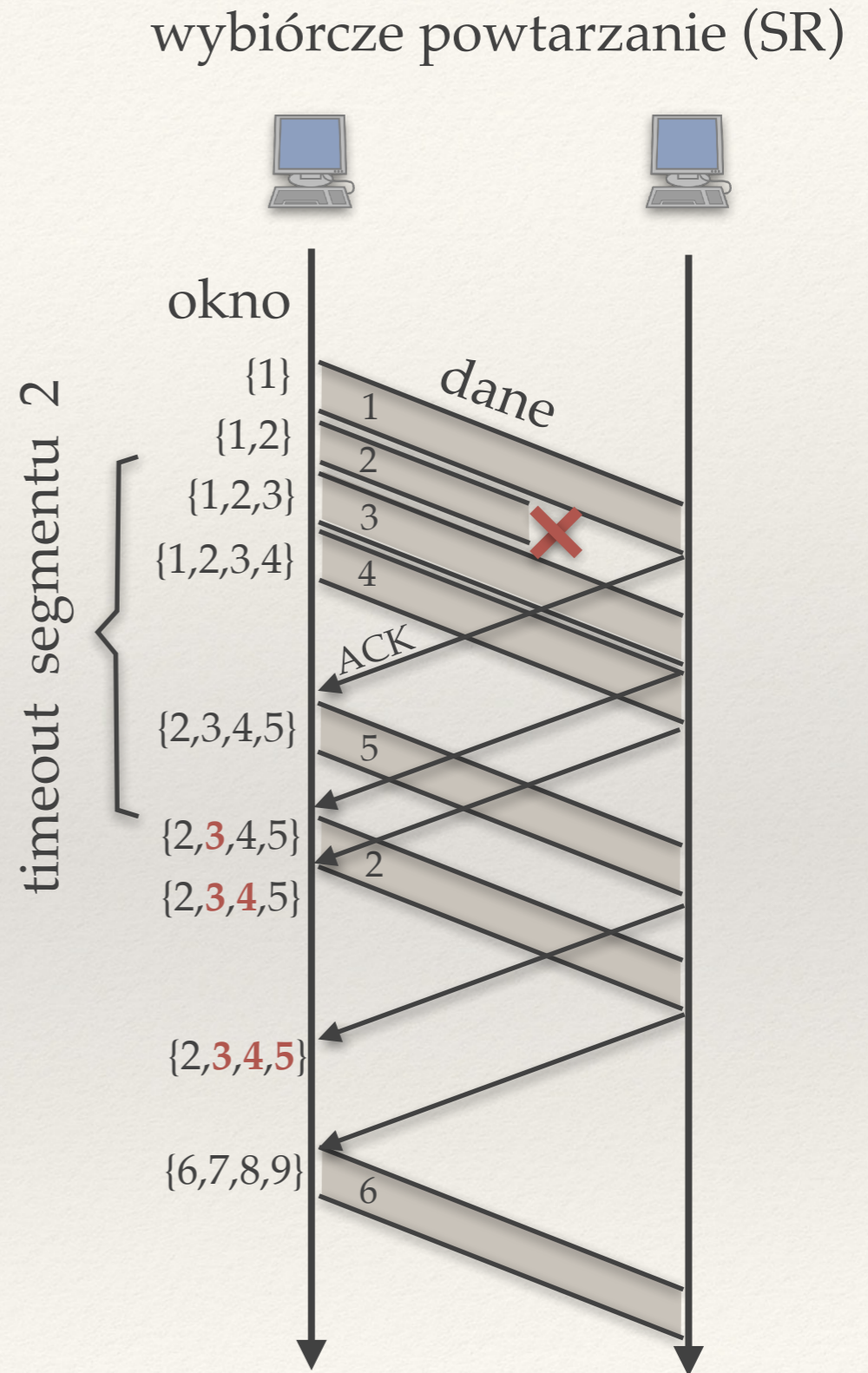
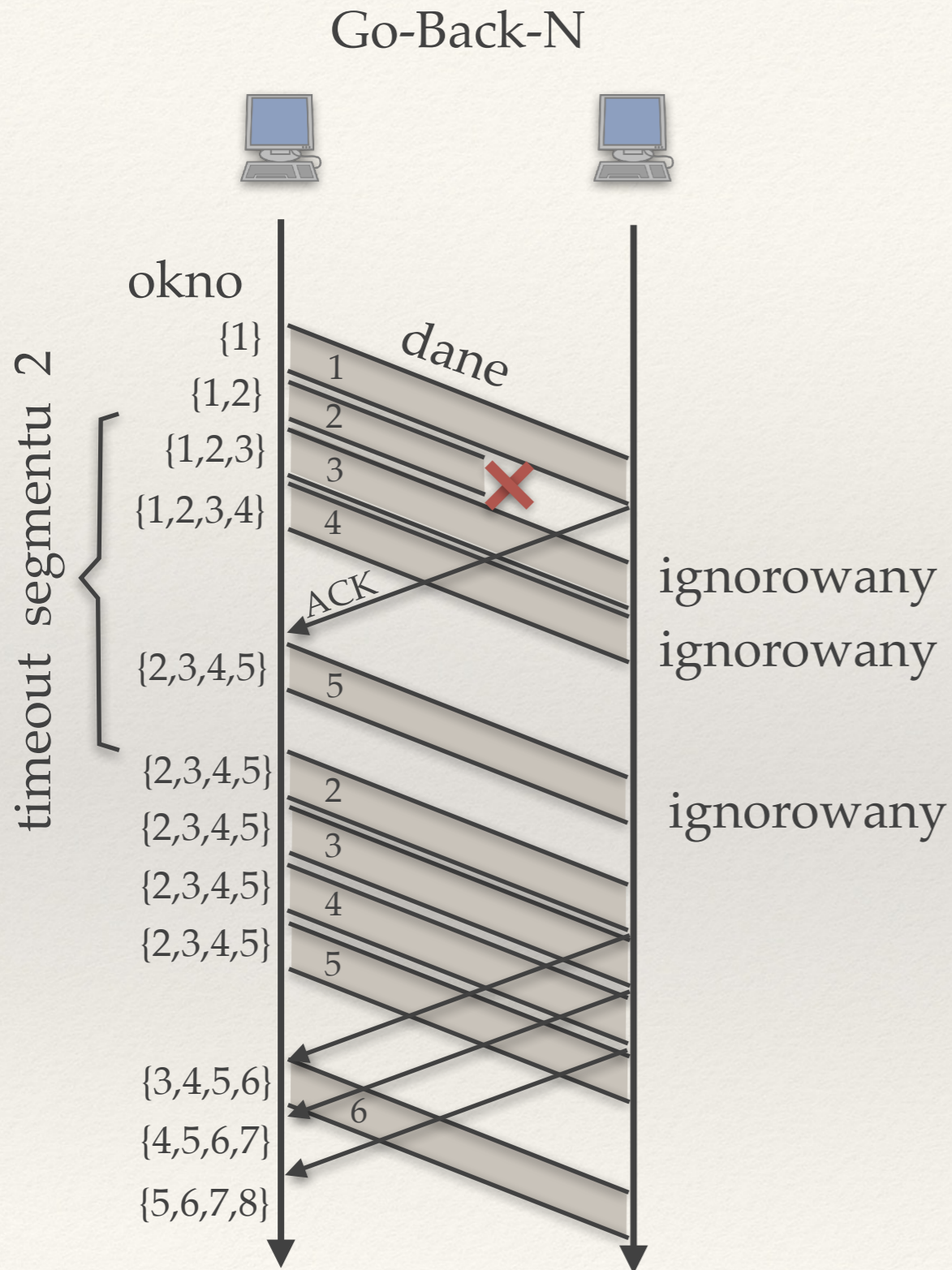
- ❖ Otrzymanie ACK $Y \rightarrow$ sprawdzamy, czy możemy przesunąć okno.
 - ✦ Cecha wspólna, ale ACK Y nie oznacza, że odbiorca ma wszystko do Y włącznie jak w Go-Back-N.
- ❖ Dla każdego segmentu X z okna:
 - ✦ wyślij X ponownie w czasie $sent(X) + RTO$.

SR: co robi nadawca






- ❖ Otrzymanie ACK $Y \rightarrow$ sprawdzamy, czy możemy przesunąć okno.
 - ✦ Cecha wspólna, ale ACK Y nie oznacza, że odbiorca ma wszystko do Y włącznie jak w Go-Back-N.
- ❖ Dla każdego segmentu X z okna:
 - ✦ wyślij X ponownie w czasie $sent(X) + RTO$.
- ❖ Jeśli $RWS = 1$, to Selective-Repeat = Go-Back-N.

Go-Back-N vs. SR, SWS = 4, utrata segmentu



Okno przesuwne nadawcy

- ❖ Klasa algorytmów wykorzystująca przesuwne okno nadawcy.
 - ♦ Cechy wspólne. 
 - ♦ Algorytm Go-Back-N. 
 - ♦ Algorytm wybiórczego powtarzania (*selective repeat*). 
- ❖ Implementacja w TCP.

ARQ: TCP

Liczymy w bajtach

- ❖ TCP dzieli strumień bajtów na segmenty.
- ❖ Podział jest dynamiczny i może być modyfikowany później:
 - ◆ Aplikacja może dostarczać małe dane do wysyłki.
 - ◆ MTU na ścieżce może się zmienić.
 - ◆ Rozmiar okna może się zmieniać (przyszłe wykłady).

Upraszczamy

- ❖ **Pełna implementacja SR jest skomplikowana:**
 - ♦ Konieczność pamiętania czasu wysyłania poszczególnych zakresów bajtów.
 - ♦ Konieczność retransmisji w określonych momentach.

Upraszczamy

- ❖ **Pełna implementacja SR jest skomplikowana:**
 - ♦ Konieczność pamiętania czasu wysyłania poszczególnych zakresów bajtów.
 - ♦ Konieczność retransmisji w określonych momentach.
- ❖ **TCP:**
 - ♦ Okno odbiorcy jak w SR.
 - ♦ Nadawca ma timeout tylko dla segmentu $LAR+1$ (jak w GBN).
 - ♦ **Potwierdzenia skumulowane:**
 - ♦ „ACK n ” oznacza „mam wszystko do bajtu $n-1$ włącznie”.
 - ♦ Umożliwia przesunięcie okna o wiele segmentów naraz.

Upraszczamy

❖ Pełna implementacja SR jest skomplikowana:

- ✦ Konieczność pamiętania czasu wysyłania poszczególnych zakresów bajtów.
- ✦ Konieczność retransmisji w określonych momentach.

❖ TCP:

- ✦ Okno odbiorcy jak w SR.
- ✦ Nadawca ma timeout tylko dla segmentu $LAR+1$ (jak w GBN).
- ✦ **Potwierdzenia skumulowane:**
 - ✦ „ACK n ” oznacza „mam wszystko do bajtu $n-1$ włącznie”.
 - ✦ Umożliwia przesunięcie okna o wiele segmentów naraz.

animacje

TCP (Transmission Control Protocol)

- ❖ Bajty w strumieniu są numerowane.
- ❖ ACK zazwyczaj wysyłany w pakiecie razem z danymi w drugą stronę.

0		7	8		15	16		23	24		31
port źródłowy						port docelowy					
numer sekwencyjny (numer pierwszego bajtu w segmencie)											
numer ostatniego potwierdzanego bajtu + 1											
offset	000	ECN	U-A-P-R-S-F			oferowane okno					
suma kontrolna						wskaźnik pilnych danych					
dodatkowe opcje, np. potwierdzanie selektywne											

Potwierdzenia opóźnione

- ❖ Cel: zmniejszenie liczby pakietów tylko z ACK.
- ❖ Czy są dane do wysłania w drugą stronę?
 - ♦ Tak → zawsze wysyłamy ACK razem z tymi danymi.
 - ♦ Nie → wymuszamy określony czas (ułamek RTT) pomiędzy kolejnymi ACK.
- ❖ Przy potwierdzeniach skumulowanych takie podejście przekazuje tę samą informację.

Szybka retransmisja

- ❖ Wysyłamy segmenty po 1000 bajtów każdy.
- ❖ Co oznacza ciąg potwierdzeń:
ACK 1001, ACK 2001, ACK 3001, ACK 3001, ACK 3001, ACK 3001, ... ?

Szybka retransmisja

- ❖ Wysyłamy segmenty po 1000 bajtów każdy.
- ❖ Co oznacza ciąg potwierdzeń:
ACK 1001, ACK 2001, ACK 3001, ACK 3001, ACK 3001, ACK 3001, ... ?
- ❖ Odbiorca dostał segmenty 1, 2 i 3:
 - ♦ wygenerowały ACK 1001, ACK 2001, ACK 3001.
- ❖ Następnie dostał trzy segmenty o numerach 5 lub większych:
 - ♦ wygenerowały ACK 3001 („czekam na segment 4”).

Szybka retransmisja

- ❖ Wysyłamy segmenty po 1000 bajtów każdy.
- ❖ Co oznacza ciąg potwierdzeń:
ACK 1001, ACK 2001, ACK 3001, ACK 3001, ACK 3001, ACK 3001, ... ?
- ❖ Odbiorca dostał segmenty 1, 2 i 3:
 - ✦ wygenerowały ACK 1001, ACK 2001, ACK 3001.
- ❖ Następnie dostał trzy segmenty o numerach 5 lub większych:
 - ✦ wygenerowały ACK 3001 („czekam na segment 4”).
- ❖ Wniosek: odbiorca nie ma segmentu 4.
- ❖ Mechanizm **szybkiej retransmisji**: wysyłamy segment 4 jeszcze raz bez czekania na jego RTO.

Kontrola przepływu


Okno odbiorcy


Odbiorca wysyła rozmiar wolnego miejsca w oknie odbiorcy w polu **oferowane okno**.

- ❖ Przy każdym ACK.
- ❖ To inna informacja niż ACK: segmenty mogą być potwierdzone na bieżąco, ale jeśli aplikacja wolno czyta z okna, to oferowane okno jest małe.

Okno odbiorcy



 odebrane, potwierdzone, przeczytane przez aplikację za pomocą `read()`

 odebrane (i potwierdzone) ale nieprzeczytane przez aplikację

Odbiorca wysyła rozmiar wolnego miejsca w oknie odbiorcy w polu **oferowane okno**.

- ❖ Przy każdym ACK.
- ❖ To inna informacja niż ACK: segmenty mogą być potwierdzane na bieżąco, ale jeśli aplikacja wolno czyta z okna, to oferowane okno jest małe.

Oferowane okno

- ❖ Oferowane okno = wolne miejsce w buforze odbiorcy.
- ❖ Nadawca zmienia SWS (rozmiar swojego okna) na rozmiar oferowanego okna.
- ❖ Zachowawcze działanie: być może część wysłanych i niepotwierdzonych pakietów jest już w buforze odbiorcy.



Oferowane okno

- ❖ Oferowane okno = wolne miejsce w buforze odbiorcy.
- ❖ Nadawca zmienia SWS (rozmiar swojego okna) na rozmiar oferowanego okna.
- ❖ Zachowawcze działanie: być może część wysłanych i niepotwierdzonych pakietów jest już w buforze odbiorcy.



demonstracja

Informacje dodatkowe

Przykładowe komplikacje

- ❖ Oferowaliśmy okno = 0 i aplikacja zwolniła miejsce?
→ Wyślij osobno rozmiar okna (bez ACK).
- ❖ Nie mamy danych do wysłania w drugą stronę?
→ Opóźnij wysyłanie ACK.
- ❖ Oferowane okno jest mniejsze niż MSS?
→ Czekaj z nadawaniem.

Obliczanie RTO

- ❖ Obliczamy średnie RTT ważone wykładniczo czasem.
 - ♦ $\text{avg-RTT} = \alpha \times \text{avg-RTT} + (1 - \alpha) \times \text{zmierzone-RTT}$.
 - ♦ RTT segmentów stałe \rightarrow avg-RTT zbiega do tej wartości.
 - ♦ W podobny sposób mierzymy wariancję RTT (var-RTT).

- ❖ $\text{RTO (retransmission timeout)} = 2 \times \text{avg-RTT} + 4 \times \text{var-RTT}$.

SACK (Selective ACKs)

- ❖ Współcześnie TCP obsługuje również potwierdzanie wybiórcze (*selective ACKs*).
- ❖ Nagłówek TCP zawiera wtedy:
 - ♦ „zwykłego” skumulowane ACK,
 - ♦ dodatkowe potwierdzenie kilku regionów bajtów.
- ❖ Bardziej precyzyjna informacja, przydatna zwłaszcza przy szybkiej retransmisji.

Gdzie implementować?

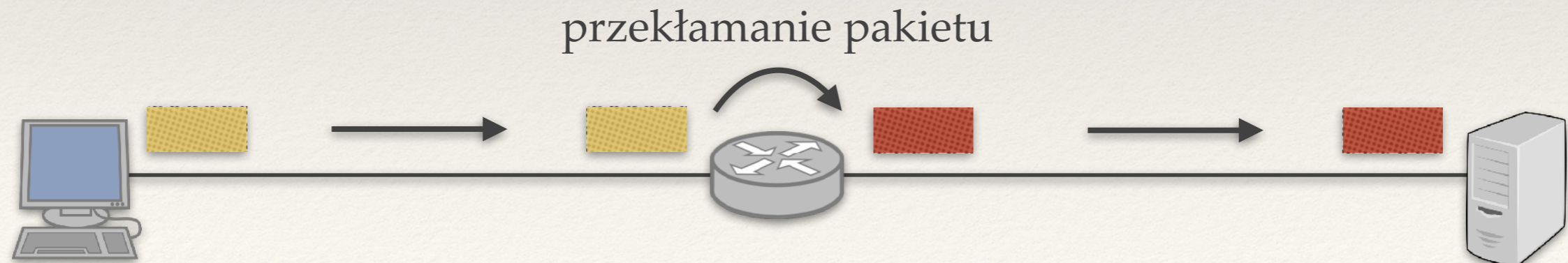
Gdzie implementować?

Warstwa sieciowa:

- ❖ Implementowana na routerach i urządzeniach końcowych.

Warstwa transportowa:

- ❖ Czy niezawodność dostarczania zapewniać na każdym łączy?
- ❖ Problem: błąd może nie być związany z łączyem.
- ❖ Niezawodność dostarczania i tak musi być kontrolowana na urządzeniach końcowych.



Zasada end-to-end

Niezawodne przesyłanie danych musi być zaimplementowane na urządzeniach końcowych w warstwie transportowej, a urządzenia pośrednie i niższe warstwy...

- ❖ ... mogą w tym pomagać. (**słaba wersja zasady**)
- ❖ ... nie powinny się tym zajmować. (**silna wersja zasady**)

Słaba wersja zasady: TCP + WiFi

- ❖ WiFi traci 20-80% pakietów.
- ❖ TCP działa dobrze, jeśli łącza są w miarę niezawodne (bo reaguje na utratę pakietów zwolnieniem wysyłania).
- ❖ WiFi wprowadza potwierdzanie i retransmisję na poziomie warstwy łącza danych (to nie są zadania warstwy łącza danych).
- ❖ Krótkoterminowe korzyści, ale być może trudności w przyszłości przy wprowadzeniu innej wersji warstwy transportowej.

Lektura dodatkowa

- ❖ Kurose & Ross: rozdział 3.
- ❖ Tanenbaum: rozdział 6.
- ❖ Dokumentacja TCP: <https://web.archive.org/web/20211226010252/http://www.networksorcery.com/enp/protocol/tcp.htm>
- ❖ Interaktywne animacje:
 - ♦ Go-Back-N: https://media.pearsoncmg.com/ph/esm/ecs_kurose_compnetwork_8/cw/content/interactiveanimations/go-back-n-protocol/index.html
 - ♦ SR: https://media.pearsoncmg.com/ph/esm/ecs_kurose_compnetwork_8/cw/content/interactiveanimations/selective-repeat-protocol/index.html

Zagadnienia

- ❖ Co może stać się z przesyłanym ciągiem pakietów IP podczas zawodnego i niezawodnego transportu?
- ❖ Co to jest kontrola przepływu?
- ❖ Czym różnią się protokoły UDP i TCP? Podaj zastosowania każdego z nich.
- ❖ Co to jest segmentacja? Dlaczego segmenty mają ograniczoną wielkość? Rozwiń skrót MSS.
- ❖ Jak nazywają się jednostki danych przesyłane w kolejnych warstwach?
- ❖ Jak małe pakiety zmniejszają opóźnienie przesyłania danych?
- ❖ Jak protokoły niezawodnego transportu wykrywają duplikaty pakietów i potwierdzeń?
- ❖ Opisz algorytm Stop-and-Wait. Jakie są jego wady i zalety?
- ❖ Do czego służą numery sekwencyjne w niezawodnym protokole transportowym?
- ❖ Opisz algorytmy Go-Back-N i wybiórczego powtarzania (*selective repeat*).
- ❖ Dlaczego istotne jest potwierdzanie odbioru duplikatów segmentów?
- ❖ Co to jest RTO? Na jakiej podstawie jest ustalana wartość RTO?
- ❖ Co to są potwierdzenia skumulowane?
- ❖ Jakie mechanizmy niezawodnego transportu i kontroli przepływu implementowane są w protokole TCP?
- ❖ Na czym polega mechanizm opóźnionych potwierdzeń?
- ❖ Co oznaczają pola „numer sekwencyjny” i „numer potwierdzenia” w nagłówku TCP?
- ❖ Co to jest okno oferowane? Jak pomaga w kontroli przepływu?
- ❖ Na czym polega mechanizm szybkiej retransmisji? Kiedy się go wykorzystuje?
- ❖ Czy warstwa transportowa implementowana jest na routerach? Dlaczego?
- ❖ Sformułuj słabą i silną zasadę end-to-end.