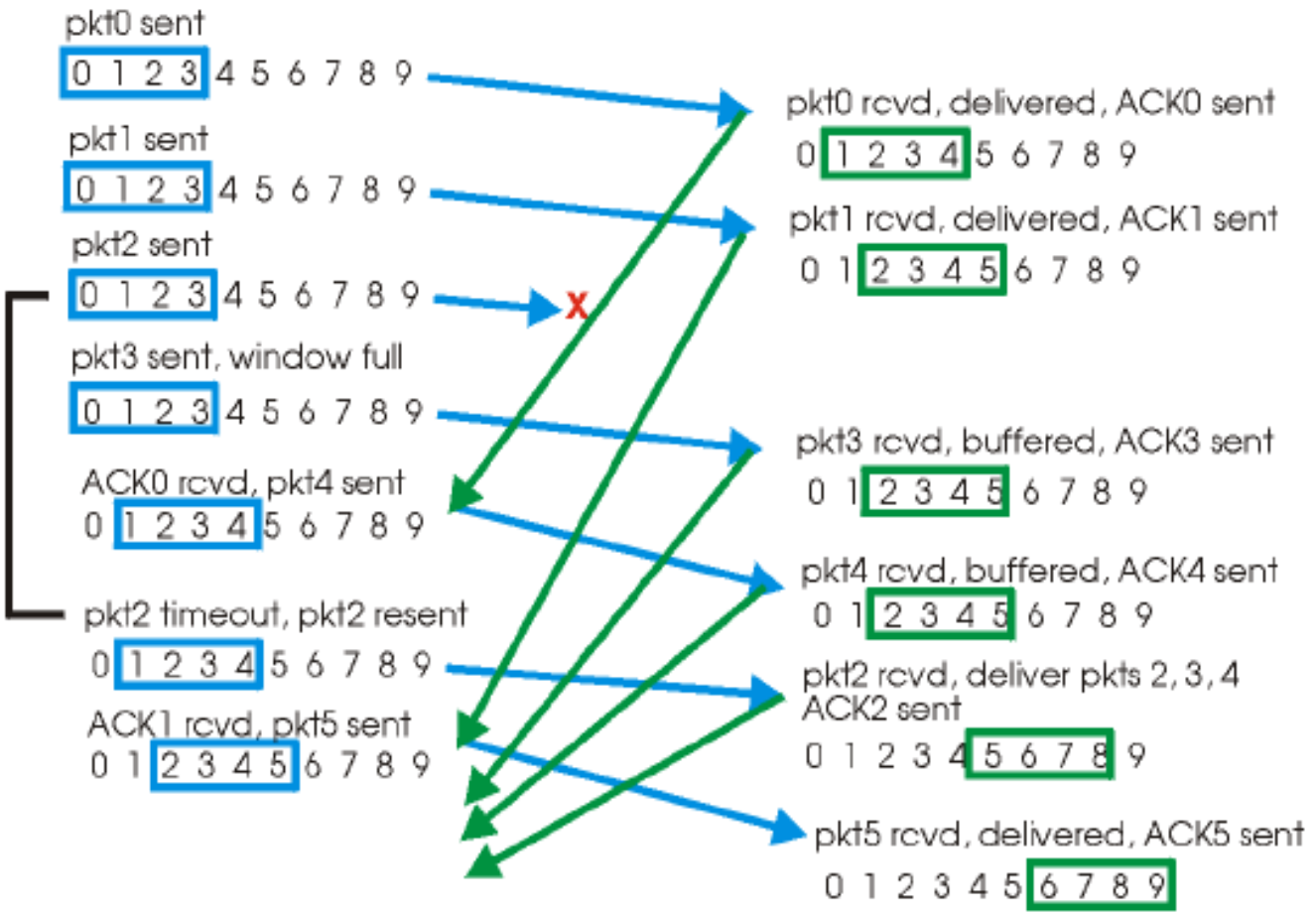
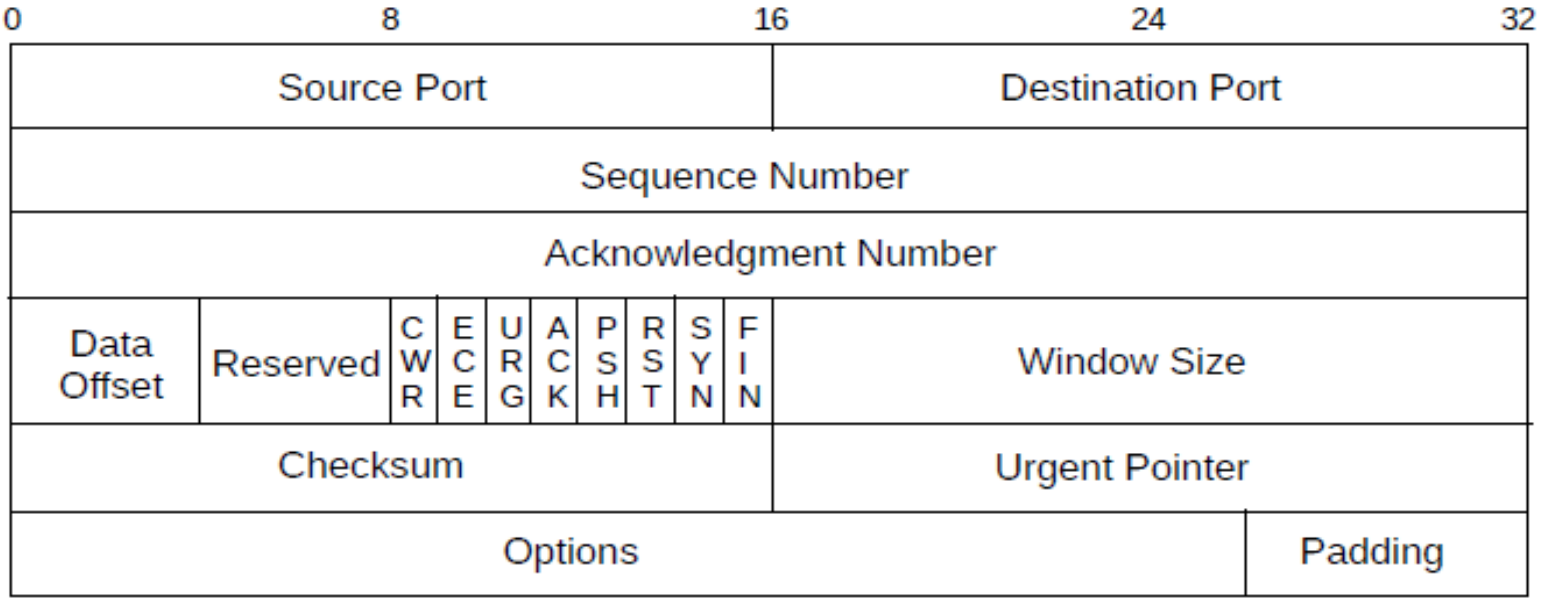
TCP

Co już wiemy ?

1. **strumień danych,** brak struktury komunikatów
2. **połączenie wirtualne**, dwukierunkowe, ident przez 4 liczby: saddr/daddr/sport/dport
3. **bufory** nadawczy i odbiorczy na obu końcach poł. TCP (flush/psh)
4. zachowuje się jak łącza unix-owe (close(), prod/kons)
5. **przesuwające się okno** (ang. sliding window), segm TCP z danymi i ack...



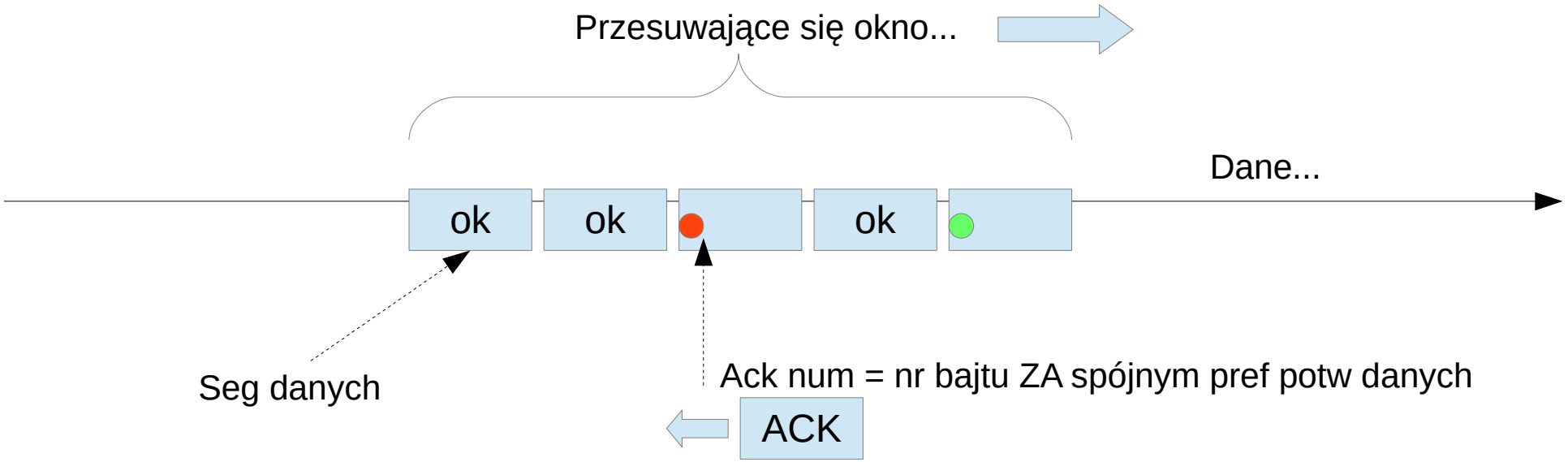
format segmentu TCP



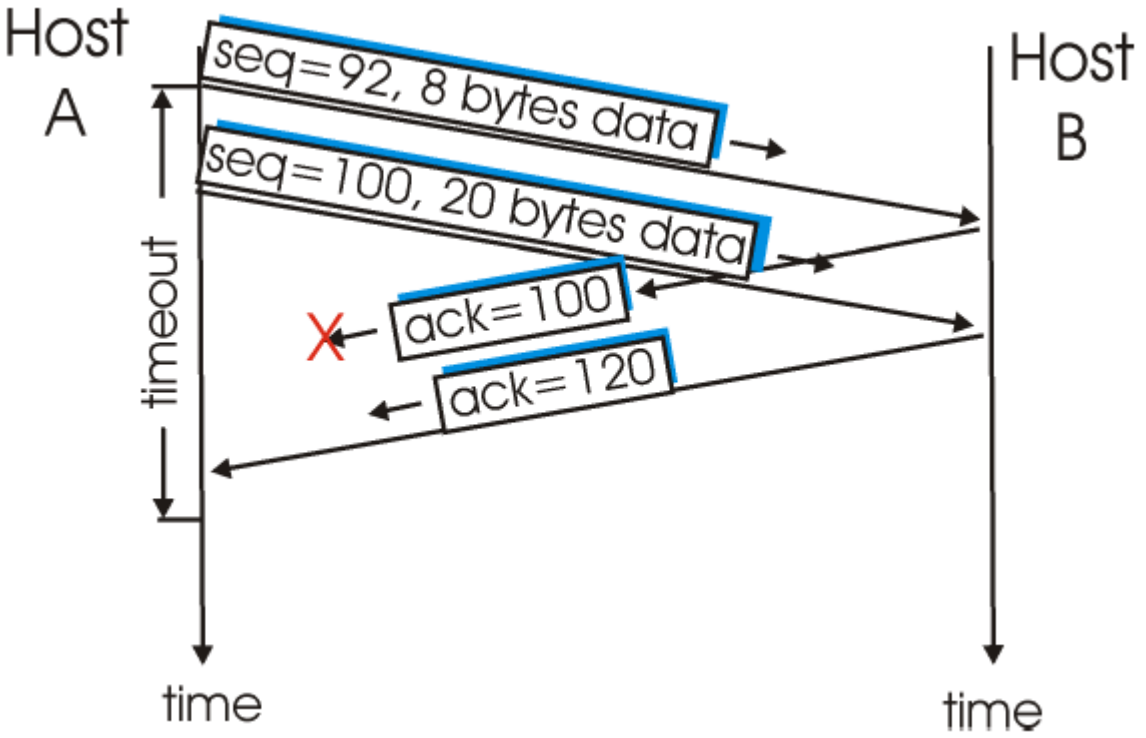
Najważniejsze pola:

1. sport, dport
2. seq num (pol. nr porządkowy), A->B
   nr pierwszego bajtu w seg danych z pkt widzenia nadawcy
3. ack num (pol. nr potwierdzenia), B->A
   nr bajtu ZA potwierdzonym spójnym ciągiem danych u odbiorcy,
   1 seg ACK może potwierdzić wiele seg danych !!
4. flagi: ACK, PSH, RST, SYN, FIN, URG, …
   określają znaczenie seg, może być kilka naraz,
   ACK seg potwierdzający dostarczenie danych,
   PSH ma związek z flush(), dotyczy buf odbiorczego
   SYN/FIN nawiązywanie i zamykanie poł TCP
5. win size, proponowany przez odbiorcę rozmiar „sliding window” (wolna pamięć odbiorcy)

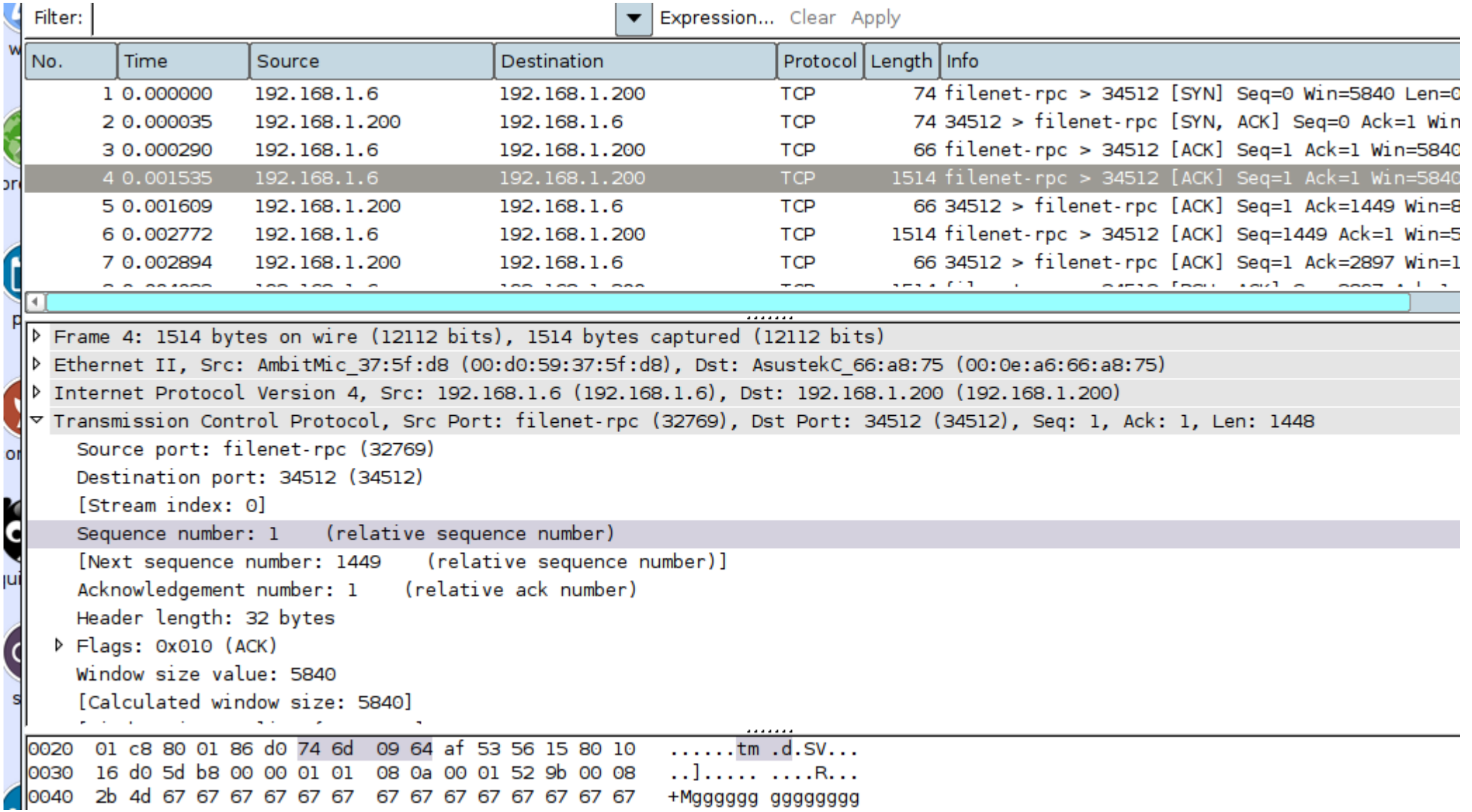
okno, seg z danymi, seg ack...



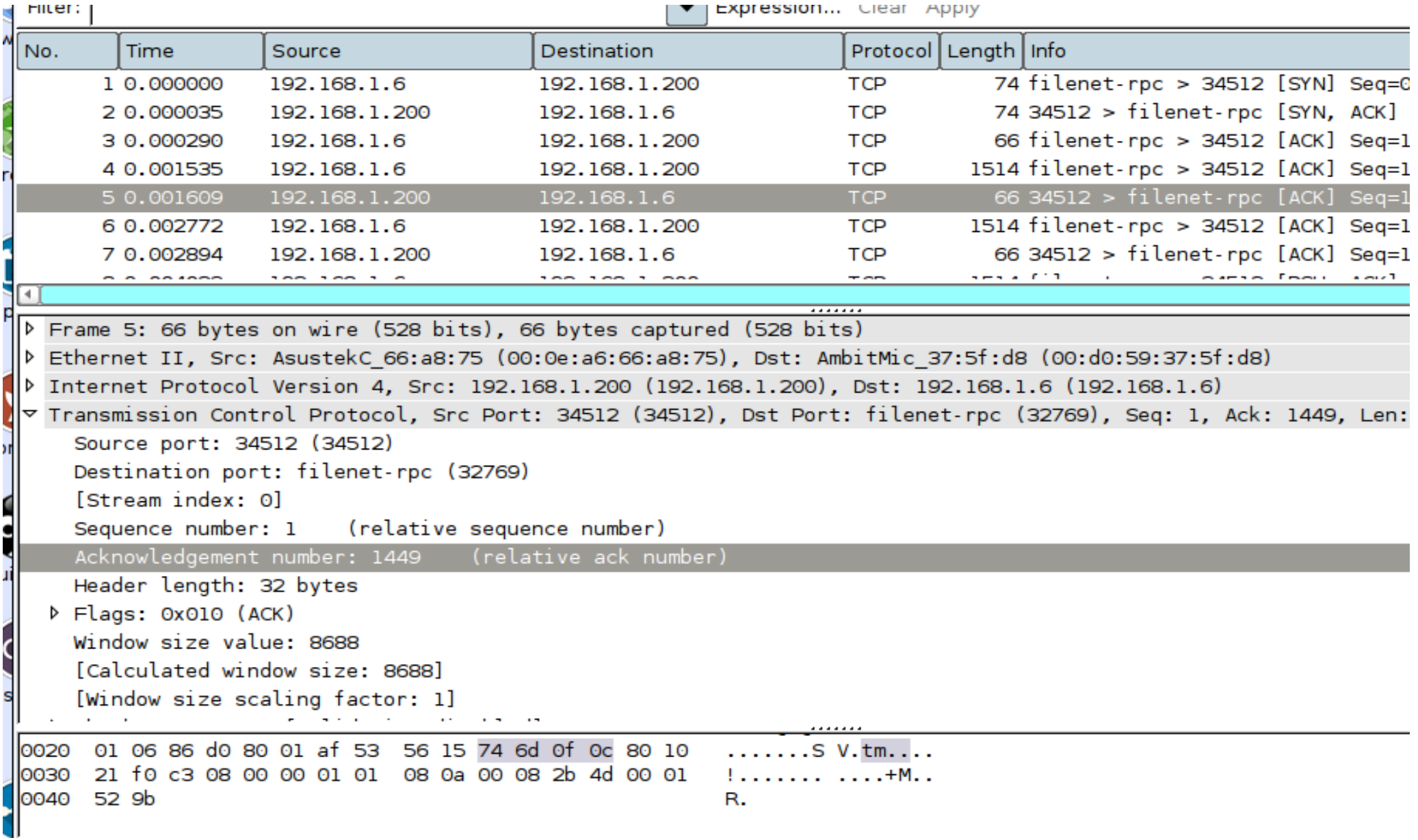
okno, seg z danymi, seg ack...



widać, kiedy 1 seg ack może potwierdzać dane z 2 seg danych...



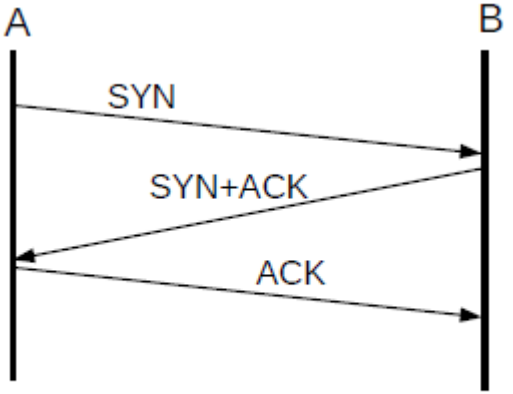
Seg danych nr 1 w wireshark...
seg ten ma ok 1.5kb oraz seq num NIE jest w rzeczywistosci =1 (!)



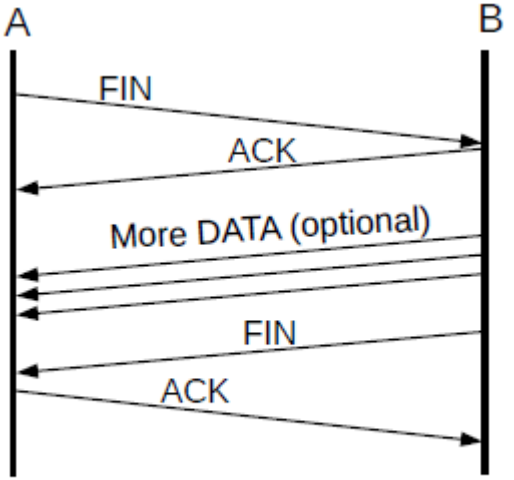
Seg ack dla poprzedniego seg danych...

ack num wskazuje na następny bajt którego spodziewa się odbiorca

Tworzenie/ niszczenie poł TCP



TCP three-way handshake



	A, ISN=**1000**	B, ISN=7000
1	SYN, **seq=1000**	
2		SYN+ACK, seq=7000, **ack=1001**
3	ACK, **seq=1001**, ack=7001	
4	“abc”, **seq=1001**, ack=7001	
5		ACK, seq=7001, **ack=1004**
6	“defg”, **seq=1004**, ack=7001	
7		seq=7001, **ack=1008**
8	“foobar”, **seq=1008**, ack=7001	
9		seq=7001, **ack=1014**, “hello”
10	**seq=1014**, ack=7006, “goodbye”	

ISN=inital seq num, potw z wartością „isn+1”, isn to nie 1 !!,
obie strony poznaja numercję bajtów z obu stron...

Automat skonczony dla TCP, klient, (czynności: niebieski/ kli, czerwony/ ser)

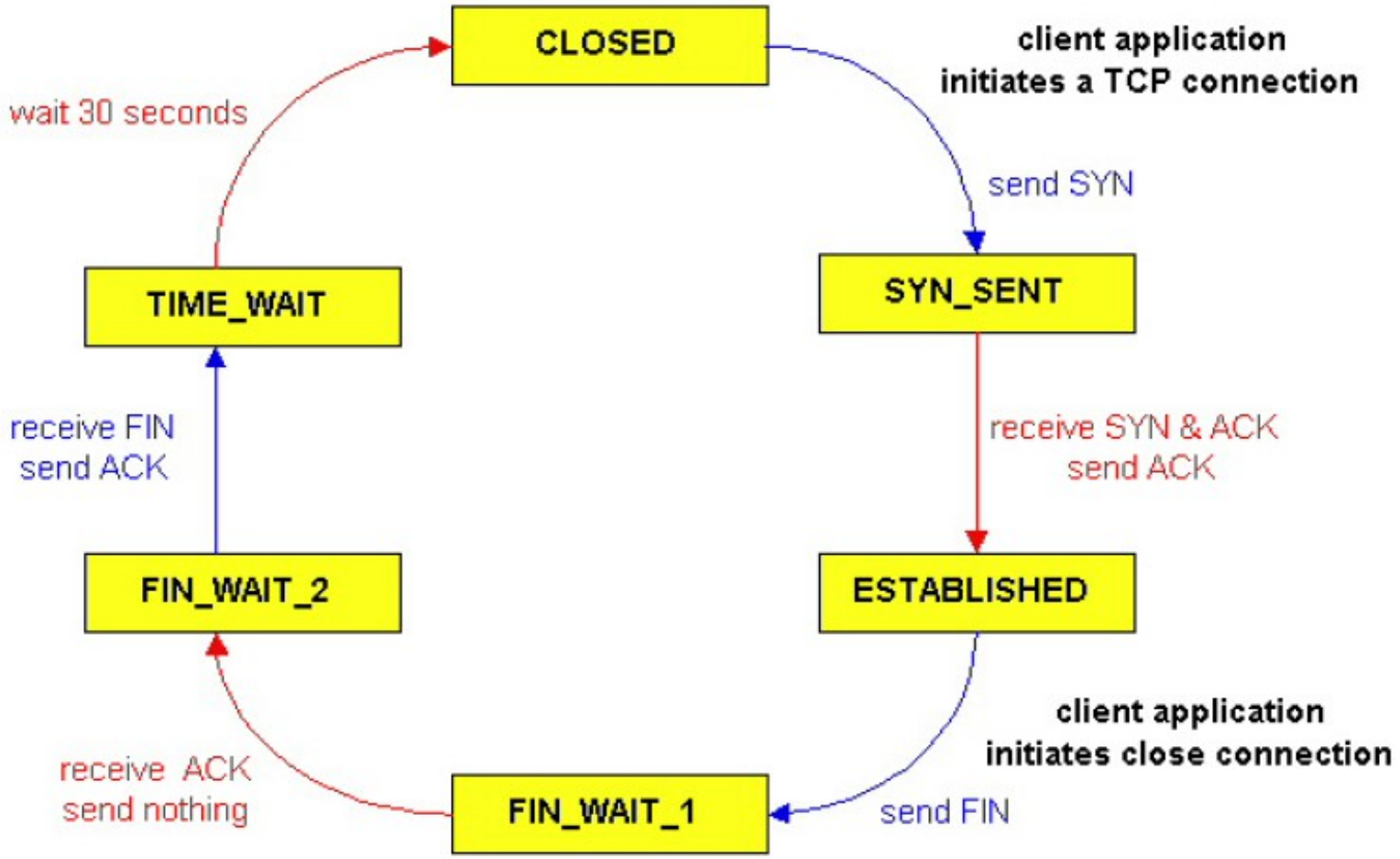


Figure 3.5-11: A typical sequence of TCP states visited by a client TCP

Automat skonczony dla TCP, serwer, (czynności: niebieski/ kli, czerwony/ ser)

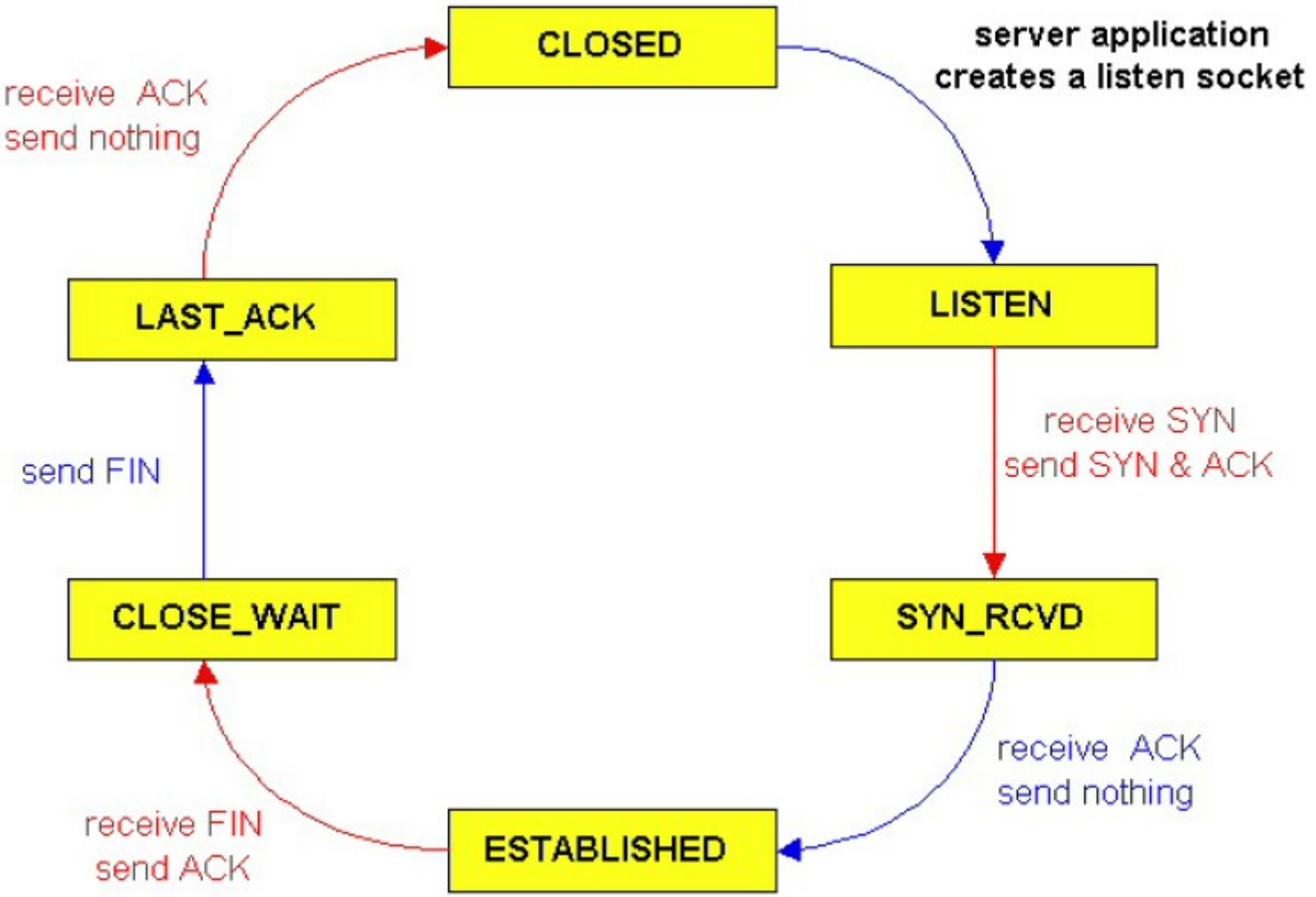


Figure 3.5-12: A typical sequence of TCP states visited by a server-side TCP

Pytania

Jak prot TCP zapewnia „niezawodnośc połączenia” ?
- Potwierdzanie i retransmisja seg danych...
  - Po jakim czasie retransmisja?? tzw. timeout

Jak prot TCP walczy z „przeciążeniem routerów” ?
- Zmniejszanie okna, zwiększanie timeout
  - Potem trzeba wrócić do normalnej przepustowości...

Problemy z którymi zmaga się prot TCP:
- Długość seg danych: na ogół 1.5kb, wg Comera trudne...
- Timeout retransmisji: na podstawie szacowanego RTT (ang. round trip time)
- Rozmiar okna: powoli rośnie (tzw „powolny start”), gwałtownie maleje przy przeciążeniu

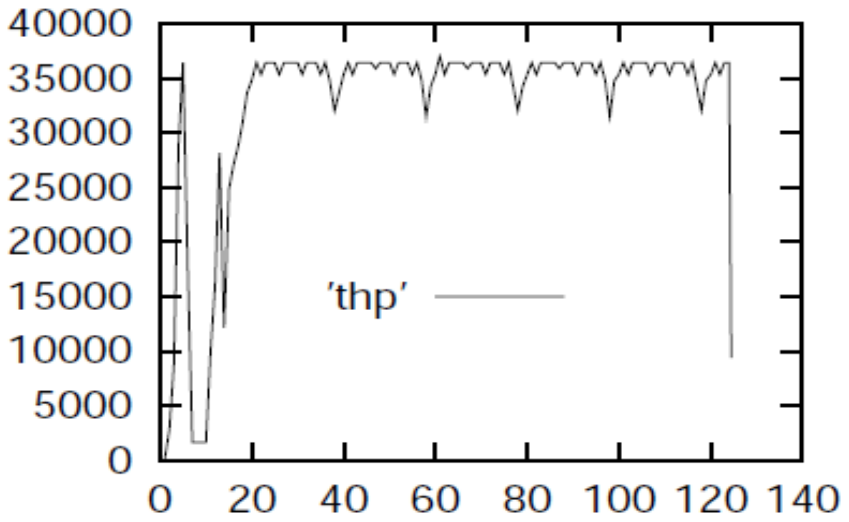


Figure 4.1: Throughput of TCP connection

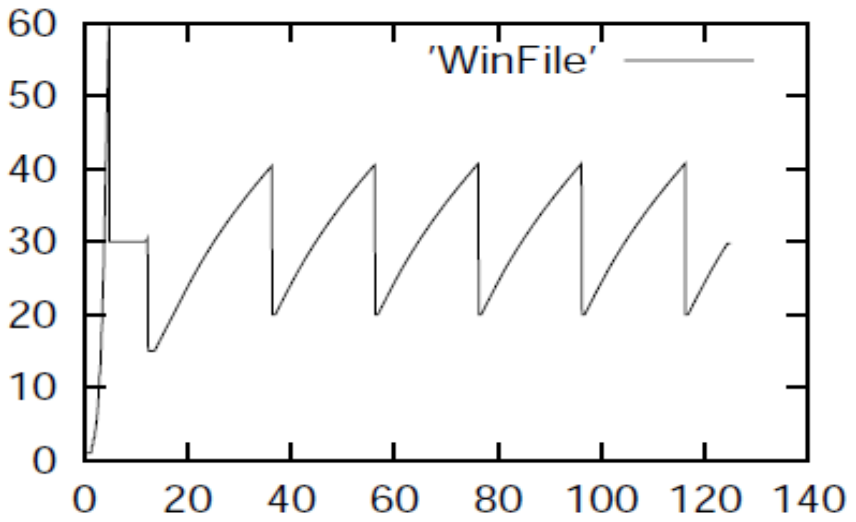


Figure 4.2: Window size of TCP connection

Szacowanie param dla TCP/ szczegóły

Obliczanie „timeout” retransmisji: $x \in (0,1)$, np. x=0.1

EstimatedRTT = (1-x) EstimatedRTT + x SampleRTT

RTT = round trip time

Deviation = (1-x) Deviation + x |SampleRTT – EstimatedRTT|

odchylenie

Timeout = EstimatedRTT + 4*Deviation

po tym czasie retransmitujemy seg danych (jeśli nie ma „ack”)

Obliczanie rozmiaru okna:

Dozwolone_okno= min(propozycja_odbiorcy, okno_przeciążeniowe)

Okno przeciążeniowe: normalnie jest ono równe prop_odbiorcy

1. jeśli zgubi się seg danych => zmniejsz okno_przeciążeniowe o połowę
2. rozpocznij od okna= 1 seg, po przyjściu ack podwajaj dla każdego seg danych, „powolny start” (wcale nie taki powolny bo wykładniczy!), istnieje próg „win_th”, po jego przekroczeniu okno rośnie wolniej (to faza „unikania przeciążenia”)

„alg Nagle-a”: *mechanizm?* unikanie krótkich seg danych o ile to możliwe, dzięki temu prot tcp jest przystosowany do usługi „telnet” jak i „ftp”
wynika z opóźnienia ack...

„alg Karna”: *heurystyka?* wydłużanie timeout
nie brać pod uwagę RTT przy retransmisji...