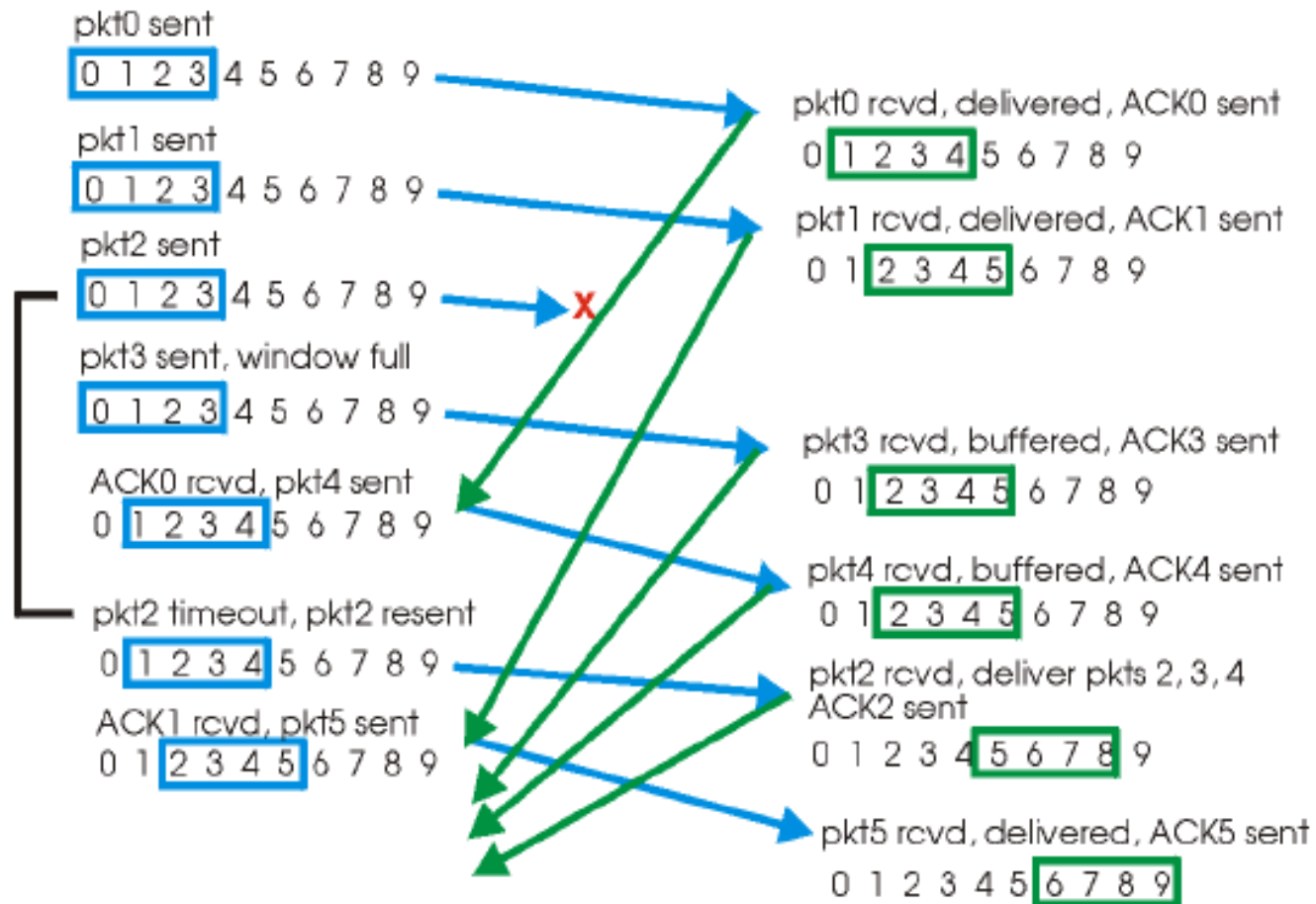


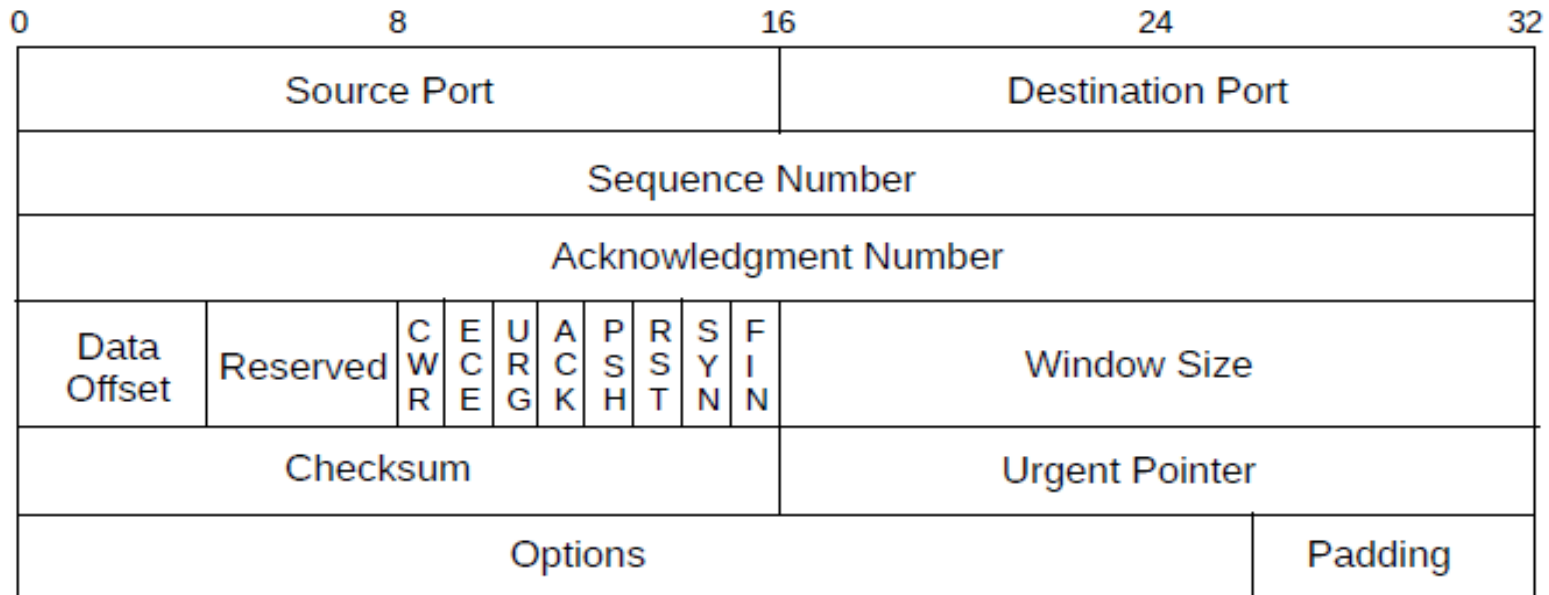
TCP

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...



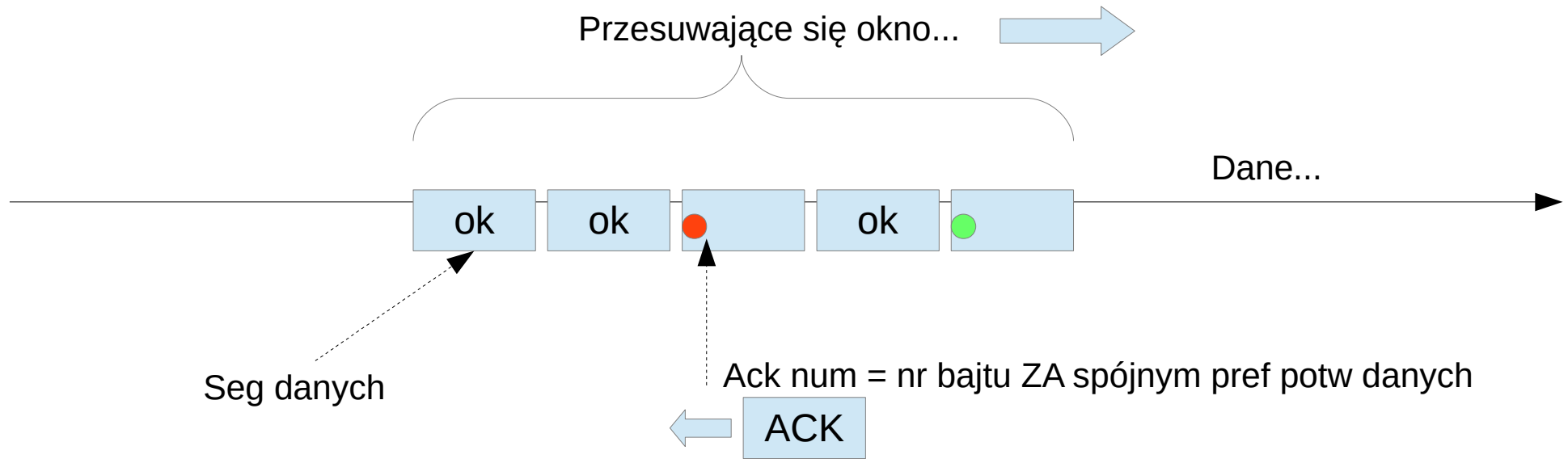
format segmentu TCP



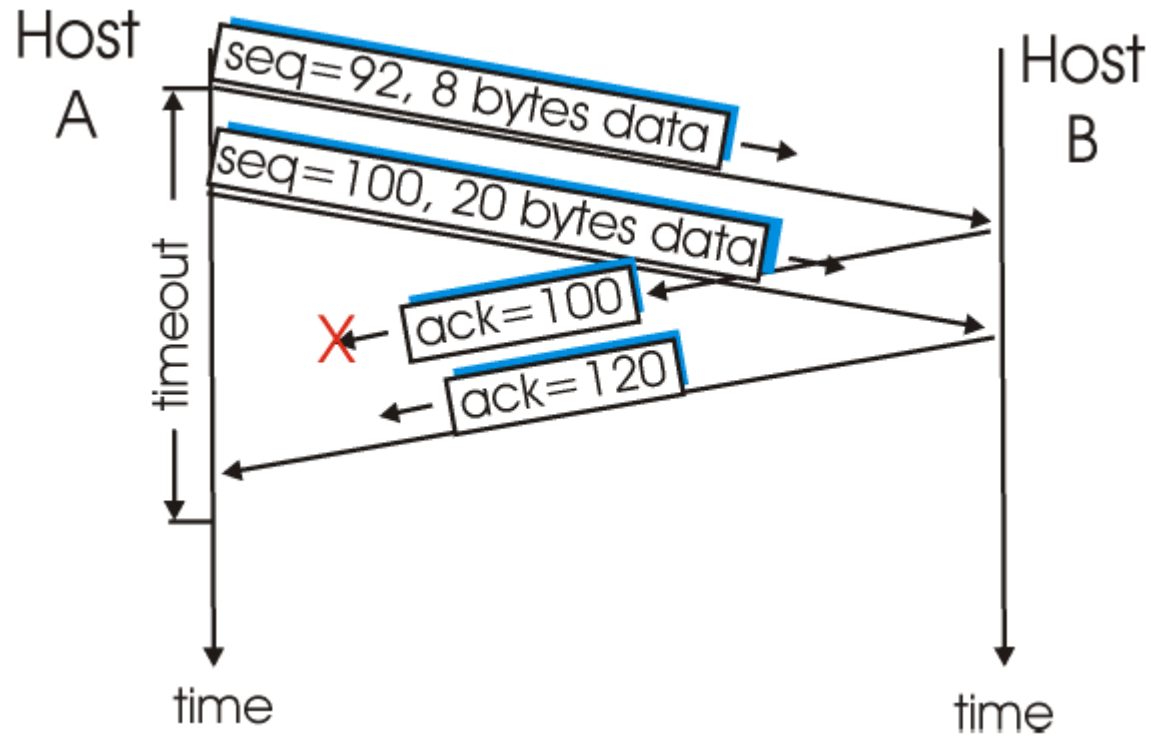
Najważniejsze pola:

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...

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.6	192.168.1.200	TCP	74	filenet-rpc > 34512 [SYN] Seq=0 Win=5840 Len=0
2	0.000035	192.168.1.200	192.168.1.6	TCP	74	34512 > filenet-rpc [SYN, ACK] Seq=0 Ack=1 Win=
3	0.000290	192.168.1.6	192.168.1.200	TCP	66	filenet-rpc > 34512 [ACK] Seq=1 Ack=1 Win=5840
4	0.001535	192.168.1.6	192.168.1.200	TCP	1514	filenet-rpc > 34512 [ACK] Seq=1 Ack=1 Win=5840
5	0.001609	192.168.1.200	192.168.1.6	TCP	66	34512 > filenet-rpc [ACK] Seq=1 Ack=1449 Win=8
6	0.002772	192.168.1.6	192.168.1.200	TCP	1514	filenet-rpc > 34512 [ACK] Seq=1449 Ack=1 Win=5
7	0.002894	192.168.1.200	192.168.1.6	TCP	66	34512 > filenet-rpc [ACK] Seq=1 Ack=2897 Win=1

Frame 4: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)

Ethernet II, Src: AmbitMic_37:5f:d8 (00:d0:59:37:5f:d8), Dst: AsustekC_66:a8:75 (00:0e:a6:66:a8:75)

Internet Protocol Version 4, Src: 192.168.1.6 (192.168.1.6), Dst: 192.168.1.200 (192.168.1.200)

Transmission Control Protocol, Src Port: filenet-rpc (32769), Dst Port: 34512 (34512), Seq: 1, Ack: 1, Len: 1448

Source port: filenet-rpc (32769)

Destination port: 34512 (34512)

[Stream index: 0]

Sequence number: 1 (relative sequence number)

[Next sequence number: 1449 (relative sequence number)]

Acknowledgement number: 1 (relative ack number)

Header length: 32 bytes

Flags: 0x010 (ACK)

Window size value: 5840

[Calculated window size: 5840]

```

0020  01 c8 80 01 86 d0 74 6d 09 64 af 53 56 15 80 10  .....tm .d.SV...
0030  16 d0 5d b8 00 00 01 01 08 0a 00 01 52 9b 00 08  ..]..... ....R...
0040  2b 4d 67 67 67 67 67 67 67 67 67 67 67 67 67 67  +Mggggggg ggggggggg

```

Seg danych nr 1 w wireshark...

seg ten ma ok 1.5kb oraz seq num NIE jest w rzeczywistosci =1 (!)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.6	192.168.1.200	TCP	74	filenet-rpc > 34512 [SYN] Seq=0
2	0.000035	192.168.1.200	192.168.1.6	TCP	74	34512 > filenet-rpc [SYN, ACK]
3	0.000290	192.168.1.6	192.168.1.200	TCP	66	filenet-rpc > 34512 [ACK] Seq=1
4	0.001535	192.168.1.6	192.168.1.200	TCP	1514	filenet-rpc > 34512 [ACK] Seq=1
5	0.001609	192.168.1.200	192.168.1.6	TCP	66	34512 > filenet-rpc [ACK] Seq=1
6	0.002772	192.168.1.6	192.168.1.200	TCP	1514	filenet-rpc > 34512 [ACK] Seq=1
7	0.002894	192.168.1.200	192.168.1.6	TCP	66	34512 > filenet-rpc [ACK] Seq=1

▶ Frame 5: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
 ▶ Ethernet II, Src: AsustekC_66:a8:75 (00:0e:a6:66:a8:75), Dst: AmbitMic_37:5f:d8 (00:d0:59:37:5f:d8)
 ▶ Internet Protocol Version 4, Src: 192.168.1.200 (192.168.1.200), Dst: 192.168.1.6 (192.168.1.6)
 ▼ Transmission Control Protocol, Src Port: 34512 (34512), Dst Port: filenet-rpc (32769), Seq: 1, Ack: 1449, Len:

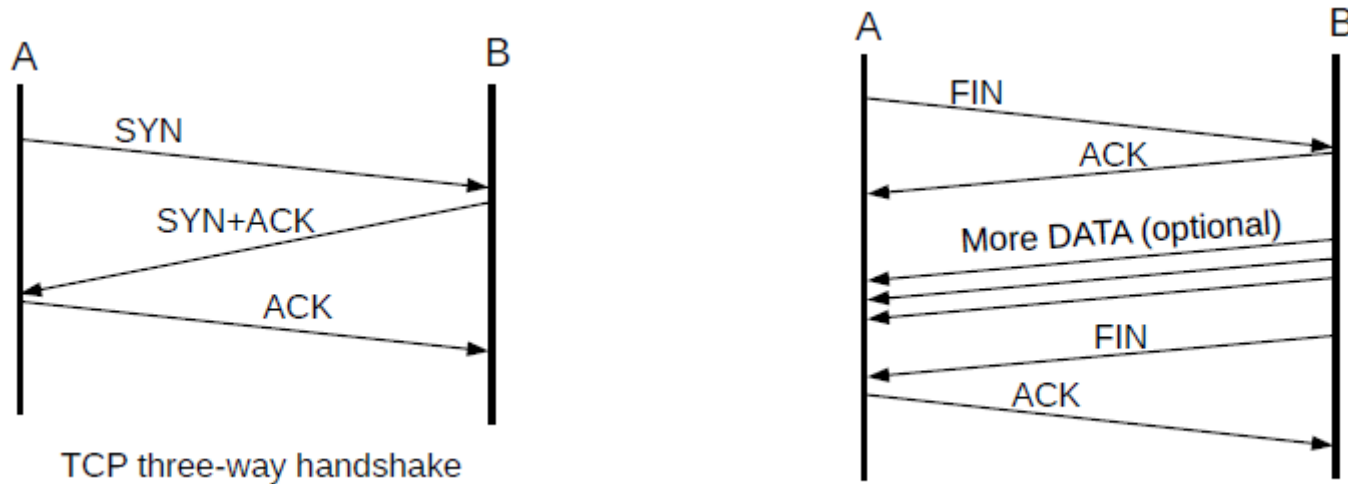
- Source port: 34512 (34512)
- Destination port: filenet-rpc (32769)
- [Stream index: 0]
- Sequence number: 1 (relative sequence number)
- Acknowledgement number: 1449 (relative ack number)
- Header length: 32 bytes
- ▶ Flags: 0x010 (ACK)
- Window size value: 8688
- [Calculated window size: 8688]
- [Window size scaling factor: 1]

```

0020  01 06 86 d0 80 01 af 53 56 15 74 6d 0f 0c 80 10  .....S V.tm....
0030  21 f0 c3 08 00 00 01 01 08 0a 00 08 2b 4d 00 01  !.....+M..
0040  52 9b                                         R.
  
```

Seg ack dla poprzedniego seg danych...
 ack num wskazuje na następny bajt którego spodziewa się odbiorca

Tworzenie/ niszczenie połą 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=initial seq num, potw z wartością „isn+1”, isn to nie 1 !!,
 obie strony poznają numercję bajtów z obu stron...

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

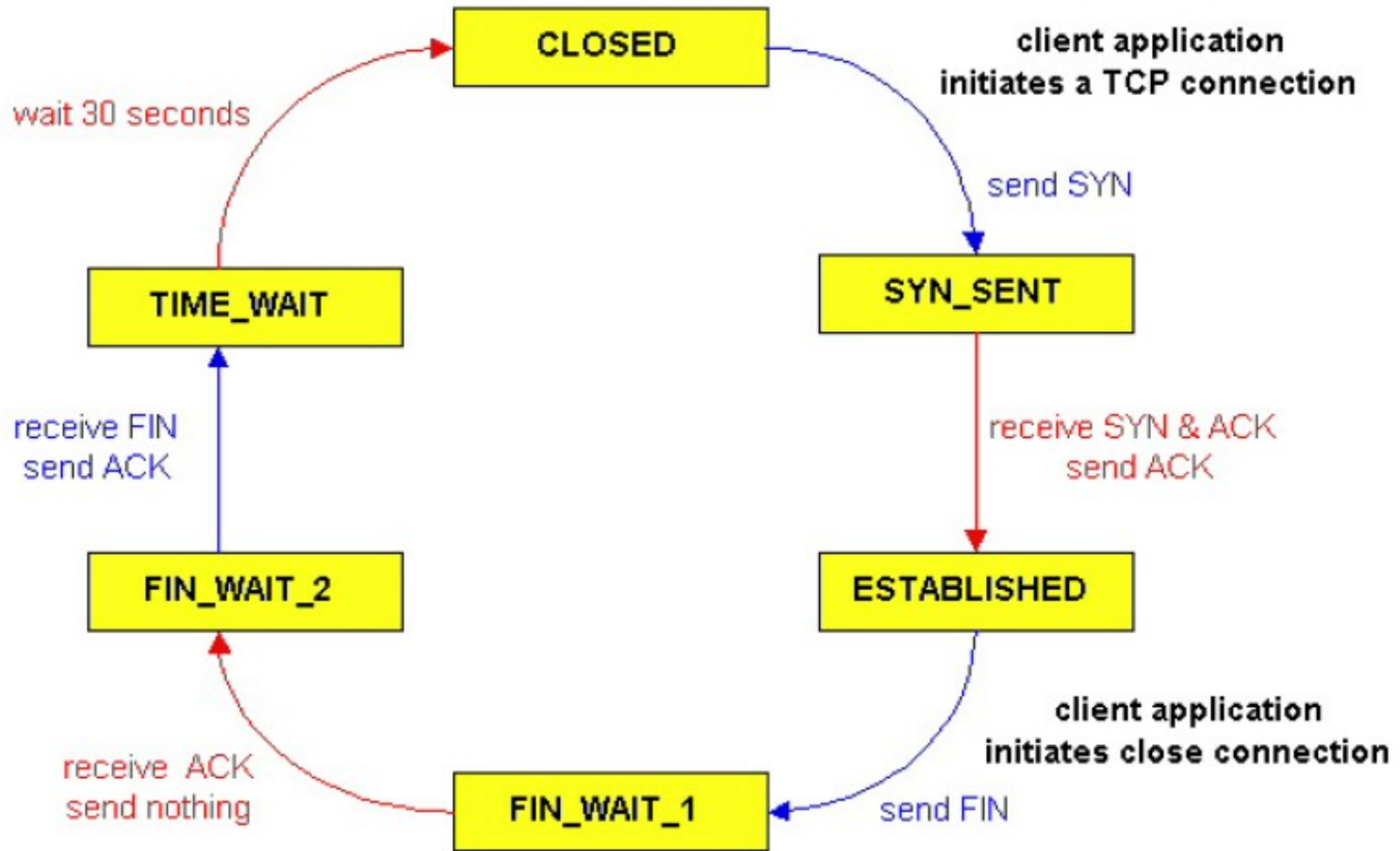


Figure 3.5-11: A typical sequence of TCP states visited by a client TCP

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

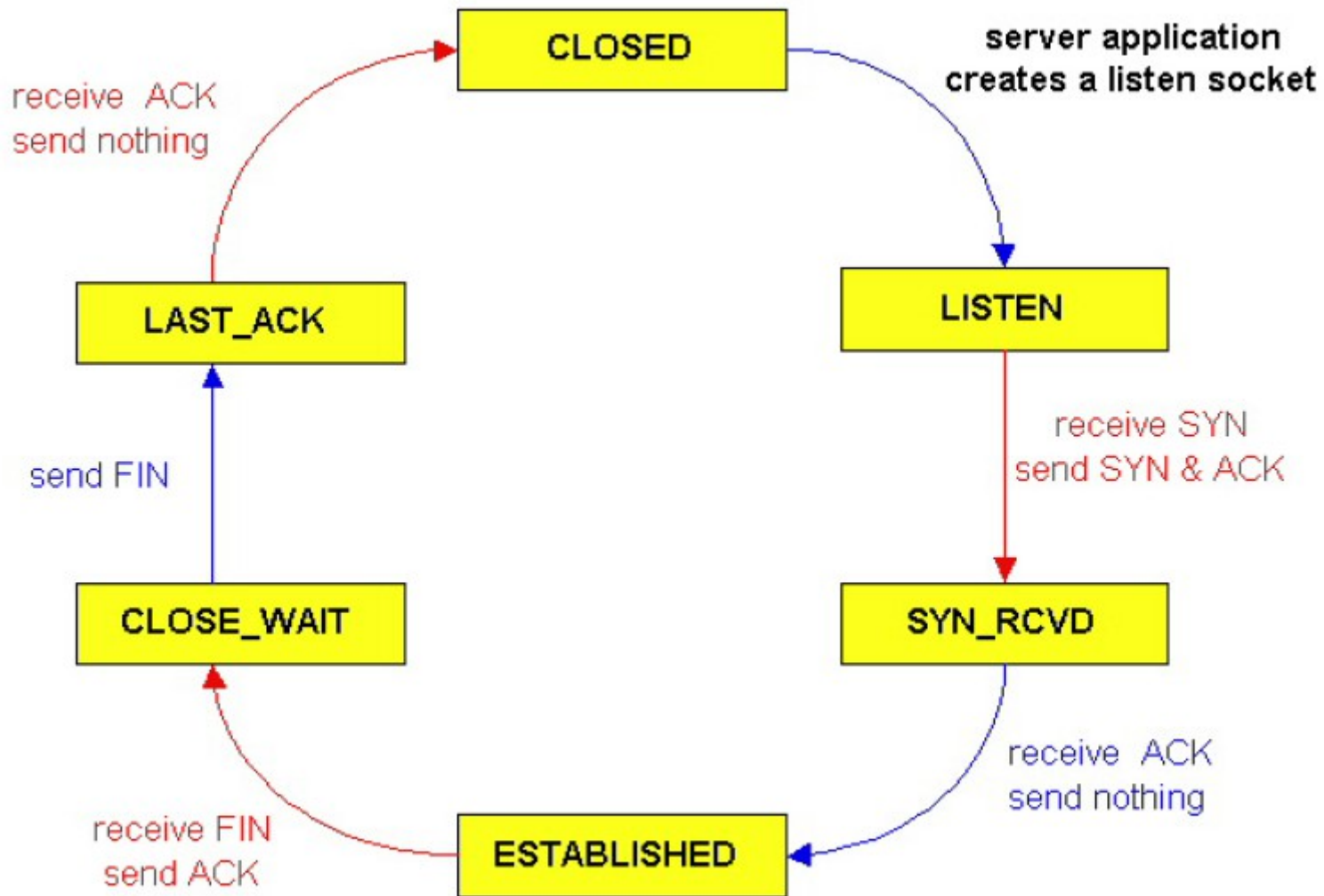


Figure 3.5-12: A typical sequence of TCP states visited by a server-side TCP

Pytania

Jak prot TCP zapewnia „niezawodność 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 zмага 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

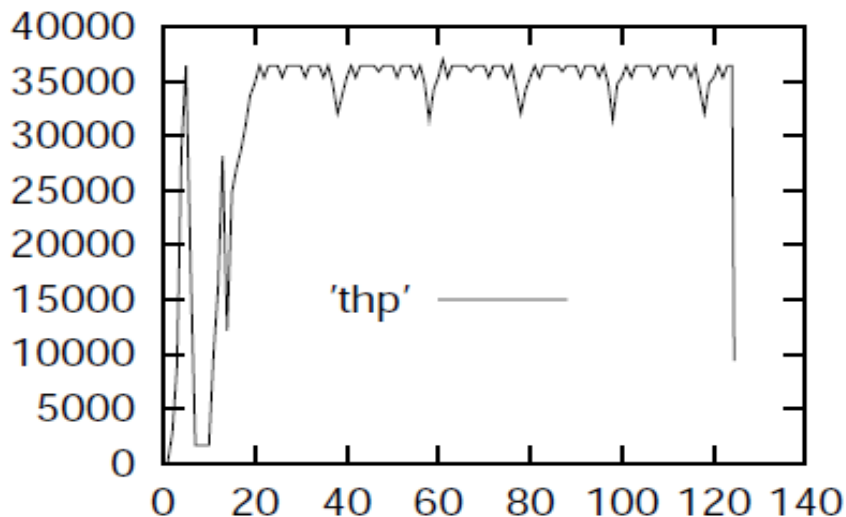


Figure 4.1: Throughput of TCP connection

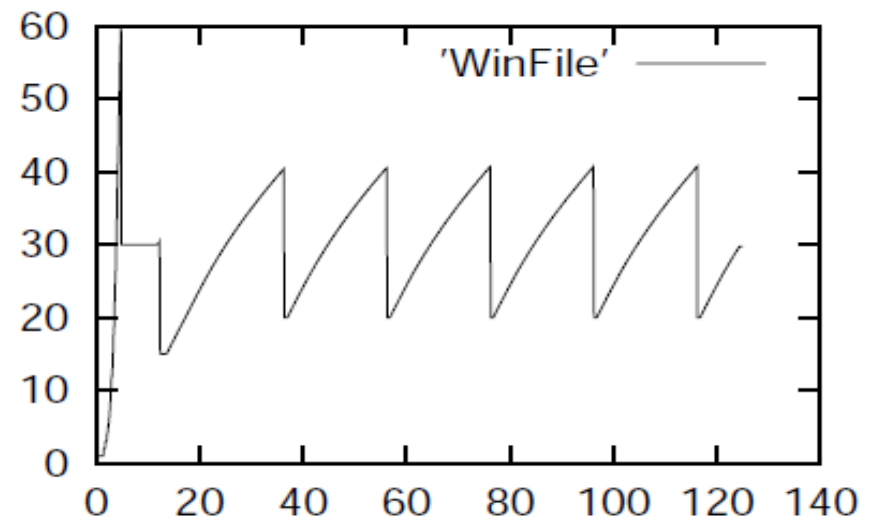


Figure 4.2: Window size of TCP connection

Szacowanie param dla TCP/ szczegóły

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

$$\text{EstimatedRTT} = (1-x) \text{ EstimatedRTT} + x \text{ SampleRTT}$$

RTT = round trip time

$$\text{Deviation} = (1-x) \text{ Deviation} + x |\text{SampleRTT} - \text{EstimatedRTT}|$$

odchylenie

$$\text{Timeout} = \text{EstimatedRTT} + 4 * \text{Deviation}$$

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

Obliczanie rozmiaru okna:

$$\text{Dozwolone_okno} = \min(\text{propozycja_odbiorcy}, \text{okno_przeciążeniowe})$$

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 przyjsciu 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 Karny”: *heurystyka?* wydłużanie timeout nie brać pod uwagę RTT przy retransmisji...

