

# Sieci Komputerowe

Grzegorz Gutowski

Uniwersytet Jagielloński

2024/25



# API - TCP - serwer

```
#!/usr/bin/env python3

import socket
import time
import threading

HOST = ''
PORT = 4567

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind((HOST, PORT))
sock.listen(1)

while True:
    conn, addr = sock.accept()
    print(addr)
    class handler(threading.Thread):
        def __init__(self, conn, addr):
            super().__init__()
            self.conn = conn
            self.addr = addr
        def run(self):
            while True:
                msg = self.conn.recv(4096)
                print(self.addr, msg)
```

# API - TCP - klient

```
#!/usr/bin/env python3

import socket

HOST = b'localhost'
PORT = 4567
NUM = 1000

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))
sock.send(b'Hello World!')
sock.send(b'Hello World!')
sock.send(b'Hello World!')
msg = sock.recv(4096)
print(msg)
sock.close()
```

# Przykład: klient HTTP

```
#!/usr/bin/env python3

import socket

HOST = b'www.tcs.uj.edu.pl'
PORT = 80

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))
req = b'GET / HTTP/1.1\r\nHost: '+HOST+b'\r\n\r\n'
print(req)
sock.send(req)
msg = sock.recv(65536)
sock.close()
print(msg)
```

## Przykład: klient HTTP 2

```
#!/usr/bin/env python3

import re
import socket

HOST = b'satori.tcs.uj.edu.pl'
PORT = 80

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))
sock.send(b'GET / HTTP/1.1\r\nHost: '+HOST+b'\r\n\r\n')
flo = sock.makefile()
response = flo.readline().rstrip()
headers = dict()
while True:
    line = flo.readline().rstrip()
    if not line:
        break
    m = re.match(r'([^\s:]+) *: *(.*)', line)
    if m:
        headers[m.group(1).lower()] = m.group(2)
body = flo.read(int(headers.get('content-length', '-1')))
sock.close()

print(response)
print(headers)
```

# Jak uzyskać niezawodny transport

- ▶ Niezawodne łącze

# Jak uzyskać niezawodny transport

- ▶ Niezawodne łącze
- ▶ Łącze z błędami, ale bez traconych pakietów

# Jak uzyskać niezawodny transport

- ▶ Niezawodne łącze
- ▶ Łącze z błędami, ale bez traconych pakietów
  - ▶ Potwierdzenia transmisji
  - ▶ Retransmisje



# Jak uzyskać niezawodny transport

- ▶ Niezawodne łącze
- ▶ Łącze z błędami, ale bez traconych pakietów
  - ▶ Potwierdzenia transmisji
  - ▶ Retransmisje
  - ▶ Błędy w potwierdzeniach

# Jak uzyskać niezawodny transport

- ▶ Niezawodne łącze
- ▶ Łącze z błędami, ale bez traconych pakietów
  - ▶ Potwierdzenia transmisji
  - ▶ Retransmisje
  - ▶ Błędy w potwierdzeniach
- ▶ Łącze z błędami i traconymi pakietami

# TCP/UDP

- ▶ różnice w API

# TCP/UDP

- ▶ różnice w API
- ▶ negocjacja

# TCP/UDP

- ▶ różnice w API
- ▶ negocjacja
- ▶ narzut

# TCP/UDP

- ▶ różnice w API
- ▶ negocjacja
- ▶ narzut
- ▶ broadcasty

# TCP/UDP

- ▶ różnice w API
- ▶ negocjacja
- ▶ narzut
- ▶ broadcasty
- ▶ przeciążenie sieci

# Jak działa TCP?

- ▶ Czego nie da się zrobić?



# Jak działa TCP?

- ▶ Czego nie da się zrobić?
- ▶ Co da się zrobić?

# Jak działa TCP?

- ▶ Czego nie da się zrobić?
- ▶ Co da się zrobić?
  - ▶ Nawiązywanie połączenia
  - ▶ Utrzymywanie połączenia
  - ▶ Potwierdzenia transmisji
  - ▶ Retransmisja
  - ▶ Prędkość
  - ▶ Kontrola czasu transmisji
  - ▶ Kontrola rozmiaru buforów
  - ▶ Unikanie przeciążeń sieci

# Packet Network Intercommunication

- ▶ Vinton G. Cerf, Robert E. Kahn, *A Protocol for Packet Network Intercommunication*, 1974

# Packet Network Intercommunication

- ▶ Vinton G. Cerf, Robert E. Kahn, *A Protocol for Packet Network Intercommunication*, 1974
- ▶ Jon Postel, *Transmission Control Protocol*, 1981, RFC 793 + RFC 7805 + ...

# TFTP

- ▶ TFTP RFC 1350

# TFTP

- ▶ TFTP RFC 1350
- ▶ Rozmiar bloku, czas transmisji RFC 2348, RFC 2349

# TFTP

- ▶ TFTP RFC 1350
- ▶ Rozmiar bloku, czas transmisji RFC 2348, RFC 2349
- ▶ Okno bloków RFC 7440

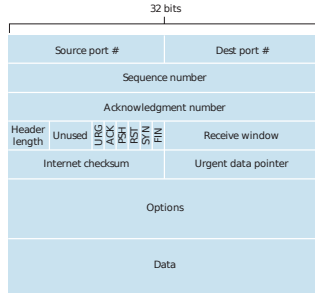
```

[ DRCV ]      <---traffic--->      [ DSND ]
  ACK#        ->                    <-  Data Block#  window block#
                ...
                <-          |DB n+01|          1
                <-          |DB n+02|          2
                <-          |DB n+03|          3
                <-          |DB n+04|          4
|ACK n+04|     ->
                <-          |DB n+05|          1
Error |<-      |DB n+06|          2
                <-          |DB n+07|          3
|ACK n+05|     ->
                <-          |DB n+06|          1
                <-          |DB n+07|          2
                <-          |DB n+08|          3
                <-          |DB n+09|          4
|ACK n+09|     ->
                <-          |DB n+10|         1
Error |<-      |DB n+11|          2
                <-          |DB n+12|          3
|ACK n+10|     ->| Error
                <-          |DB n+13|          4
                - timeout -
                <-          |DB n+10|          1
                <-          |DB n+11|          2
                <-          |DB n+12|          3
                <-          |DB n+13|          4
|ACK n+13|     ->
                ...

```



# TCP



źródło: Kurose, Ross, *Computer Networking: A Top-Down Approach*

# Prędkość

- ▶ Oczekiwanie na potwierdzenie spowalnia łącze

# Prędkość

- ▶ Oczekiwanie na potwierdzenie spowalnia łącze
  - ▶ Go-Back-N (Sliding Window)

# Prędkość

- ▶ Oczekiwanie na potwierdzenie spowalnia łącze
  - ▶ Go-Back-N (Sliding Window)
  - ▶ Selective Repeat

# Prędkość

- ▶ Oczekiwanie na potwierdzenie spowalnia łącze
  - ▶ Go-Back-N (Sliding Window)
  - ▶ Selective Repeat
- ▶ Jak szybko uznać pakiet za zagubiony?

# Prędkość

- ▶ Oczekiwanie na potwierdzenie spowalnia łączy
  - ▶ Go-Back-N (Sliding Window)
  - ▶ Selective Repeat
- ▶ Jak szybko uznać pakiet za zagubiony?
  - ▶  $EstimatedRTT = 0.875 \cdot EstimatedRTT + 0.125 \cdot SampleRTT$
  - ▶  $DevRTT = 0.75 \cdot DevRTT + 0.25 \cdot |SampleRTT - EstimatedRTT|$
  - ▶  $TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$

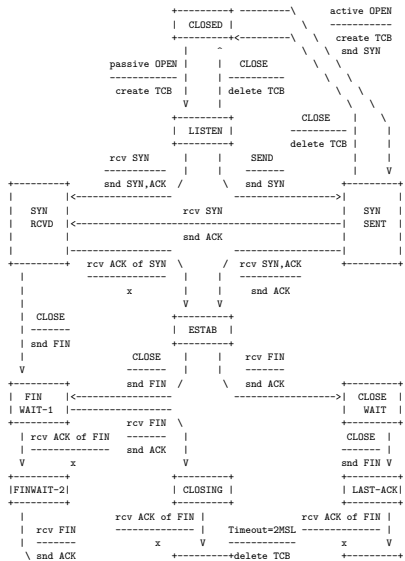
# Prędkość

- ▶ Oczekiwanie na potwierdzenie spowalnia łączy
  - ▶ Go-Back-N (Sliding Window)
  - ▶ Selective Repeat
- ▶ Jak szybko uznać pakiet za zagubiony?
  - ▶  $EstimatedRTT = 0.875 \cdot EstimatedRTT + 0.125 \cdot SampleRTT$
  - ▶  $DevRTT = 0.75 \cdot DevRTT + 0.25 \cdot |SampleRTT - EstimatedRTT|$
  - ▶  $TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$
- ▶ Jak zgłosić uszkodzony pakiet?

# Prędkość

- ▶ Oczekiwanie na potwierdzenie spowalnia łącze
  - ▶ Go-Back-N (Sliding Window)
  - ▶ Selective Repeat
- ▶ Jak szybko uznać pakiet za zagubiony?
  - ▶  $EstimatedRTT = 0.875 \cdot EstimatedRTT + 0.125 \cdot SampleRTT$
  - ▶  $DevRTT = 0.75 \cdot DevRTT + 0.25 \cdot |SampleRTT - EstimatedRTT|$
  - ▶  $TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$
- ▶ Jak zgłosić uszkodzony pakiet?
- ▶ Kontrola rozmiaru buforów
- ▶ Unikanie przeciążeń sieci





TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
3. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED
5. ESTABLISHED	--> <SEQ=101><ACK=301><CTL=ACK><DATA>	--> ESTABLISHED

# Nawiązywanie i zrywanie połączenia

- ▶ SYN, SYNACK, ...

# Nawiązywanie i zrywanie połączenia

- ▶ SYN, SYNACK, ...
- ▶ SYNFLOOD, SYN cookies

# Nawiązywanie i zrywanie połączenia

- ▶ SYN, SYNACK, ...
- ▶ SYNFLOOD, SYN cookies
- ▶ FIN, FINACK

# Kontrola rozmiaru buforów

- ▶ Lepiej wysyłać duże segmenty. Nagle

# Kontrola rozmiaru buforów

- ▶ Lepiej wysyłać duże segmenty. Nagle
- ▶ Czasem trzeba wysyłać małe segmenty. Push

## Kontrola rozmiaru buforów

- ▶ Lepiej wysyłać duże segmenty. Nagle
- ▶ Czasem trzeba wysyłać małe segmenty. Push
- ▶ Można wysyłać priorytetowe segmenty. URG



# Kontrola rozmiaru buforów

- ▶ Lepiej wysyłać duże segmenty. Nagle
- ▶ Czasem trzeba wysyłać małe segmenty. Push
- ▶ Można wysyłać priorytetowe segmenty. URG
- ▶ Kiedy wysyłać ACK?

# Przyspieszanie

- ▶ Większe okno

# Przyspieszanie

- ▶ Większe okno
- ▶ Selective Repeat

# Przyspieszanie

- ▶ Większe okno
- ▶ Selective Repeat
- ▶ RTTM, Echo + Reply

# Przyspieszanie

- ▶ Większe okno
- ▶ Selective Repeat
- ▶ RTTM, Echo + Reply
- ▶ 32 bity offsetu to mało

Network	B*8	B bits/sec	Twrap bytes/sec	secs
-----	-----	-----	-----	-----
ARPANET		56kbps	7KBps	3*10**5 (~3.6 days)
DS1		1.5Mbps	190KBps	10**4 (~3 hours)
Ethernet		10Mbps	1.25MBps	1700 (~30 mins)
DS3		45Mbps	5.6MBps	380
FDDI		100Mbps	12.5MBps	170
Gigabit		1Gbps	125MBps	17

# Unikanie przeciążeń sieci

- ▶ (Szybki) slow start, congestion avoidance, fast retransmission, fast recovery

# Unikanie przeciążeń sieci

- ▶ (Szybki) slow start, congestion avoidance, fast retransmission, fast recovery
- ▶ TCP Tahoe
- ▶ TCP Reno
- ▶ TCP Vegas
- ▶ ..., CUBIC, Compound



# Unikanie przeciążeń sieci

- ▶ (Szybki) slow start, congestion avoidance, fast retransmission, fast recovery
- ▶ TCP Tahoe
- ▶ TCP Reno
- ▶ TCP Vegas
- ▶ ..., CUBIC, Compound
- ▶ Wiele równoczesnych połączeń

# Unikanie przeciążeń sieci

- ▶ (Szybki) slow start, congestion avoidance, fast retransmission, fast recovery
- ▶ TCP Tahoe
- ▶ TCP Reno
- ▶ TCP Vegas
- ▶ ..., CUBIC, Compound
- ▶ Wiele równoczesnych połączeń
- ▶ Konkurencja z UDP

# Unikanie przeciążeń sieci

- ▶ (Szybki) slow start, congestion avoidance, fast retransmission, fast recovery
- ▶ TCP Tahoe
- ▶ TCP Reno
- ▶ TCP Vegas
- ▶ ..., CUBIC, Compound
- ▶ Wiele równoczesnych połączeń
- ▶ Konkurencja z UDP
- ▶ Oszukujące TCP

Gdzie to wszystko jest zaimplementowane?