Mapa wykładu

- 2.1 Zasady budowy protokołów w. aplikacji
- 2.2 WWW i HTTP
- 2.3 DNS
- 2.4 Programowanie przy użyciu gniazd TCP
- 2.5 Programowanie przy użyciu gniazd UDP
- 2.6 Poczta elektroniczna
  - SMTP, POP3, IMAP
- 2.7 FTP
- 2.8 Dystrybucja zawartości
  - Schowki Internetowe
  - Sieci dystrybucji zawartości
- 2.9 Dzielenie plików P2P

Programowanie z gniazdami

Cel: nauczyć się budować aplikacje klient/serwer komunikujące się przy pomocy gniazd

API gniazd

- wprowadzone w UNIX BSD4.1, 1981
- tworzone, używane, i zwalniane przez aplikacje
- model klient/serwer
- dwa rodzaje transportu przez API gniazd:
  - zawodne datagramy
  - niezawodne strumienie bajtów

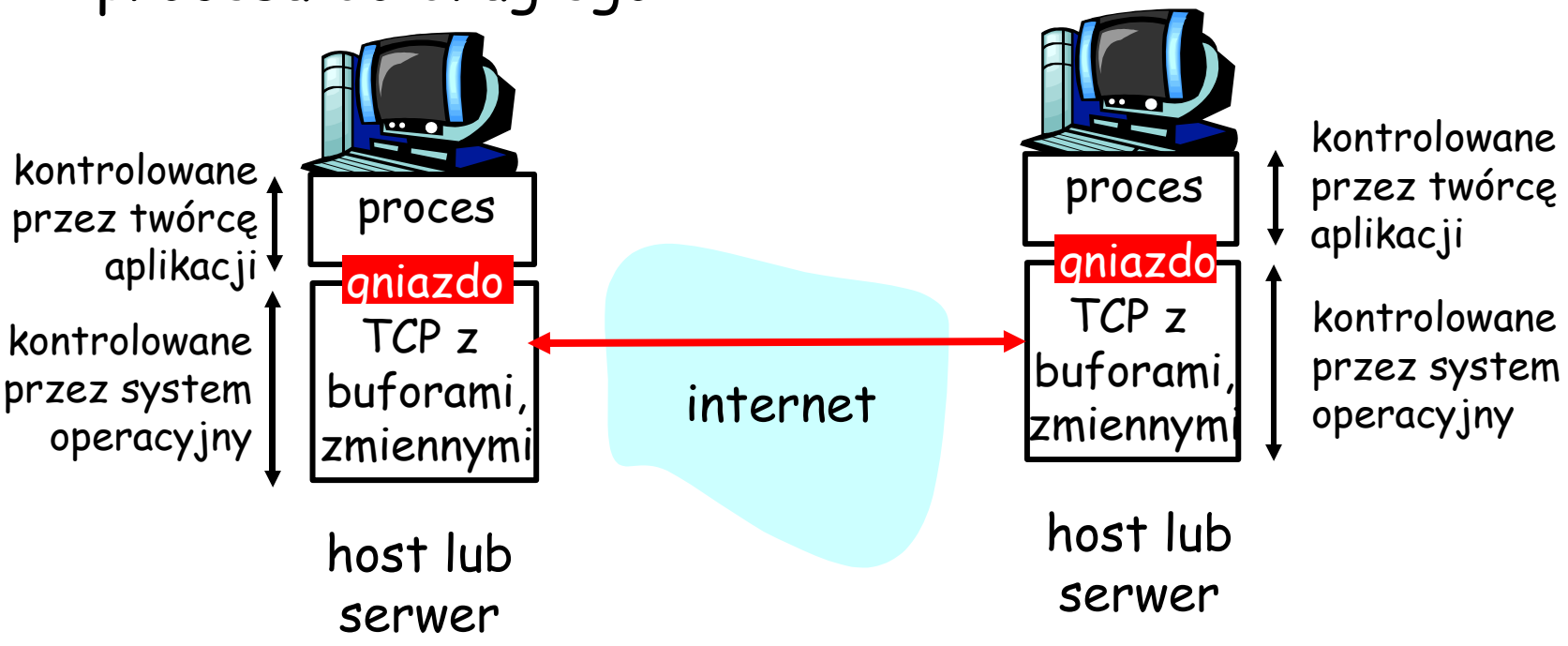
gniazdo

lokalny na hoście, *tworzony przez aplikację*, *kontrolowany przez SO* interfejs, przez który proces aplikacji może zarówno wysyłać jak i odbierać komunikaty od/do innego procesu aplikacji

Programowanie gniazd TCP

Gniazdo: interfejs pomiędzy procesem aplikacji i protokołem transportowym koniec-koniec (UCP lub TCP)

Usługa TCP: niezawodny transfer **bajtów** z jednego procesu do drugiego



Programowanie gniazd *TCP*

Klient musi połączyć się z serwerem

- proces serwera musi przedtem zostać uruchomiony
- serwer musi utworzyć gniazdo-centralę, które przyjmie rozmowę z klientem

Klient łączy się z serwerem poprzez:

- utworzenie lokalnego gniazda
- określenie adresu IP, numeru portu procesu serwera
- Następnie łączymy gniazda: TCP klienta tworzy połączenie do TCP serwera

- Po otrzymaniu połączenia, TCP serwera tworzy nowe gniazdo do komunikacji pomiędzy procesem klienta i serwera
  - pozwala serwerowi rozmawiać z wieloma klientami
  - numery portów źródła są używane do odróżnienia klientów

punkt widzenia programisty

TCP umożliwia niezawodną, uporządkowaną komunikację ("strumień") bajtów pomiędzy klientem i serwerem

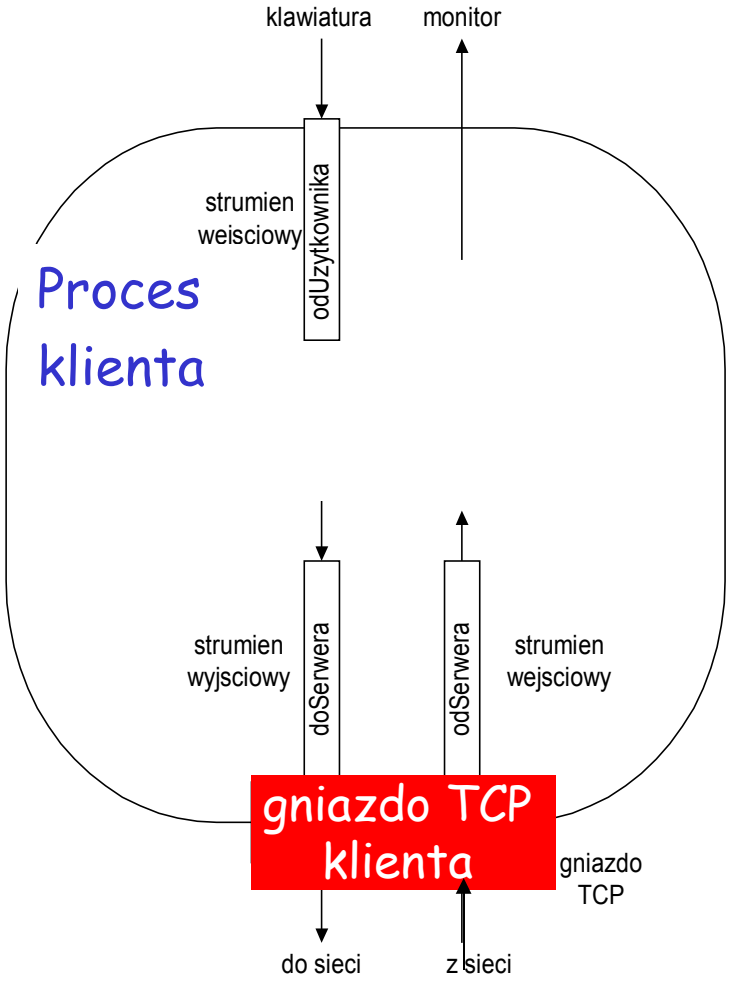
Terminologia strumieni

- Strumień jest ciągiem bajtów, które wpływają/wypływają z procesu.
- Strumień wejściowy jest podłączony do źródła danych wejściowych procesu, np, klawiatury lub gniazda.
- Strumień wyjściowy jest podłączony do odbiorcy danych z procesu, np, monitora lub gniazda.

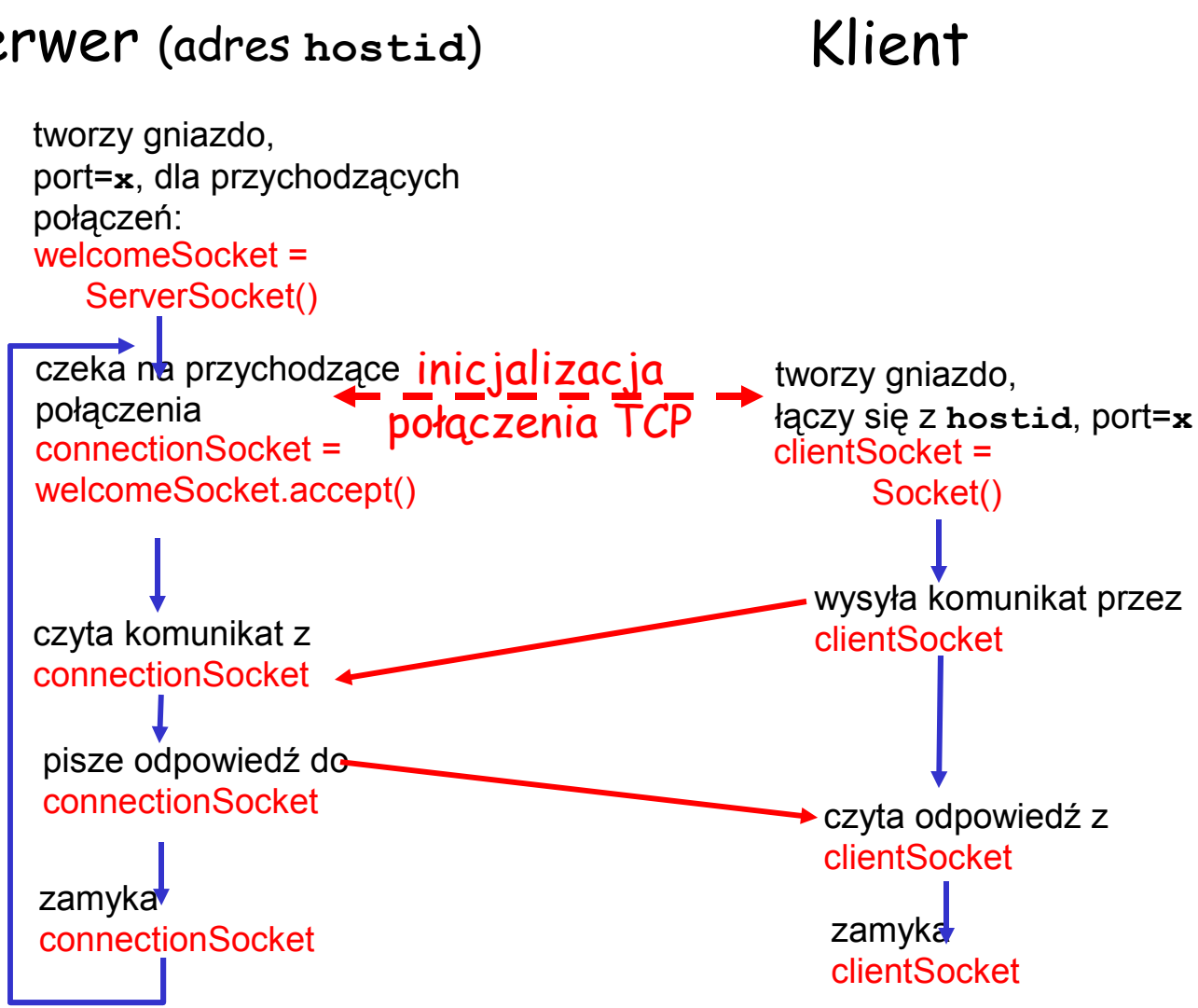
Programowanie gniazd TCP

Przykład aplikacji klient/serwer:

1) klient czyta linię ze standardowego wejścia (strumień **odUżytkownika**), wysyła do serwera przez gniazdo (strumień **doSerwera**)
2) serwer czyta linię z gniazda
3) serwer zmienia linię na wielke litery, odsyła do klienta
4) klient czyta, drukuje zmienioną linię z gniazda (strumień **odSerwera**)



Interakcja klient/serwer: TCP



Przykład: Klient w Javie (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

Tworzy strumień wejściowy

Tworzy gniazdo klienta, łączy się z serwerem

Tworzy strumień wyjściowy podłączony do gniazda

Przykład: Klient w Javie (TCP), c.d.

Tworzy strumień wejściowy podłączony do gniazda

Wysyła linię do serwera

Czyta linię z serwera

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Przykład: serwer (iteracyjny) w Javie (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Tworzy gniazdo odbierające na porcie 6789

Oczekuje na gnieździe na połączenie od klienta (blokujące)

Tworzy strumień wejściowy, podłączony do gniazda

Przykład: serwer w Javie(TCP), c.d.

Tworzy strumień wyjściowy, podłączony do gniazda

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Czyta linię z gniazda od klienta

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Wysyła linię przez gniazdo do klienta

```
outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

Koniec pętli "while", powrót i oczekiwanie na następne połączenie klienta

Mapa wykładu

- 2.1 Zasady budowy protokołów w. aplikacji
- 2.2 WWW i HTTP
- 2.3 DNS
- 2.4 Programowanie przy użyciu gniazd TCP
- 2.5 Programowanie przy użyciu gniazd UDP
- 2.6 Poczta elektroniczna
  - SMTP, POP3, IMAP
- 2.7 FTP
- 2.8 Dystrybucja zawartości
  - Schowki Internetowe
  - Sieci dystrybucji zawartości
- 2.9 Dzielenie plików P2P

Programowanie gniazd *UDP*

UDP: brak "połączenia" pomiędzy klientem i serwerem

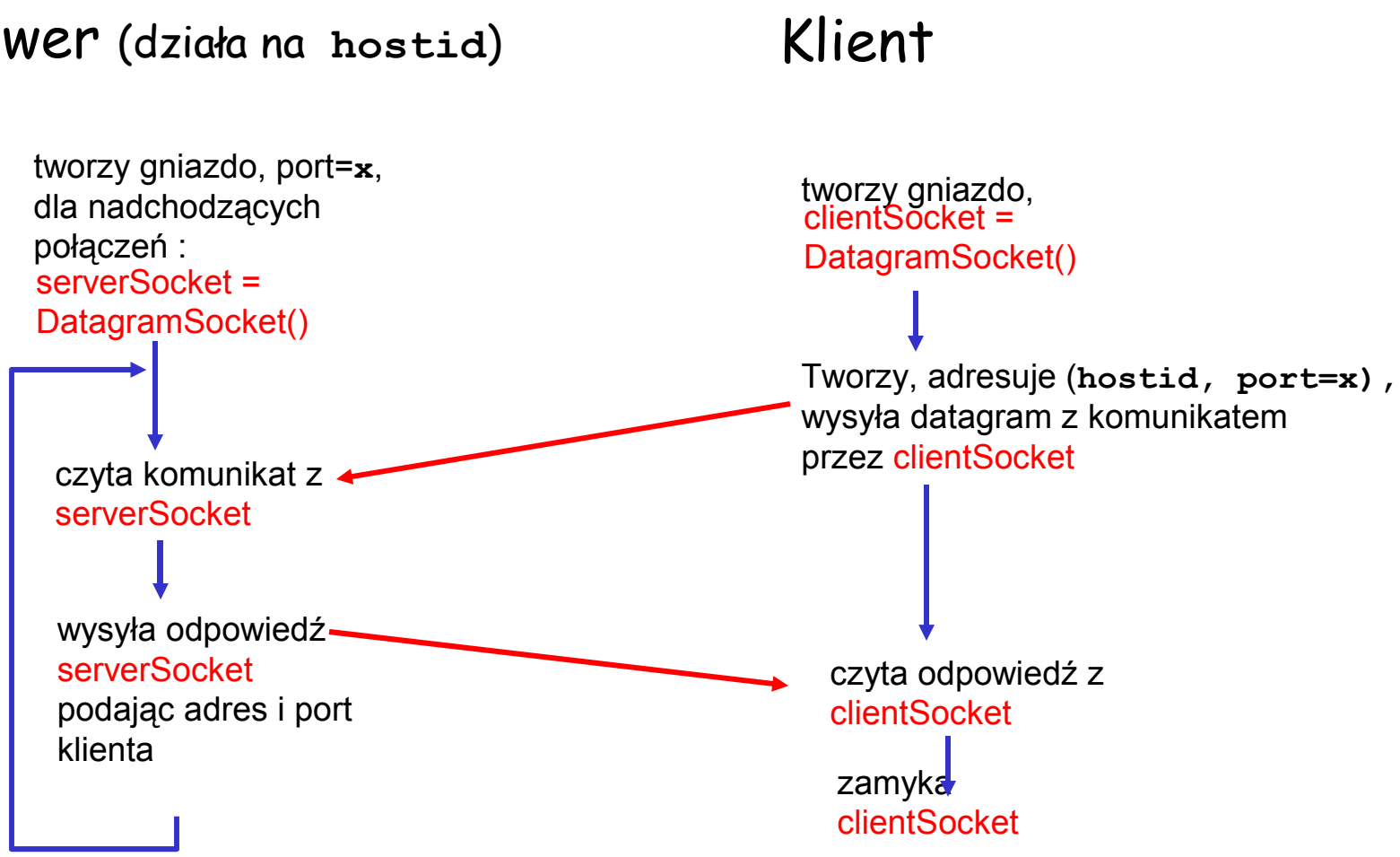
- brak inicjalizacji połączenia
- nadawca nadaje każdemu pakietowi adres IP i port odbiorcy
- serwer musi pobrać adres IP, port nadawcy z otrzymanego pakietu

UDP: wysyłane informacje mogą być gubione lub otrzymywane w innym porządku

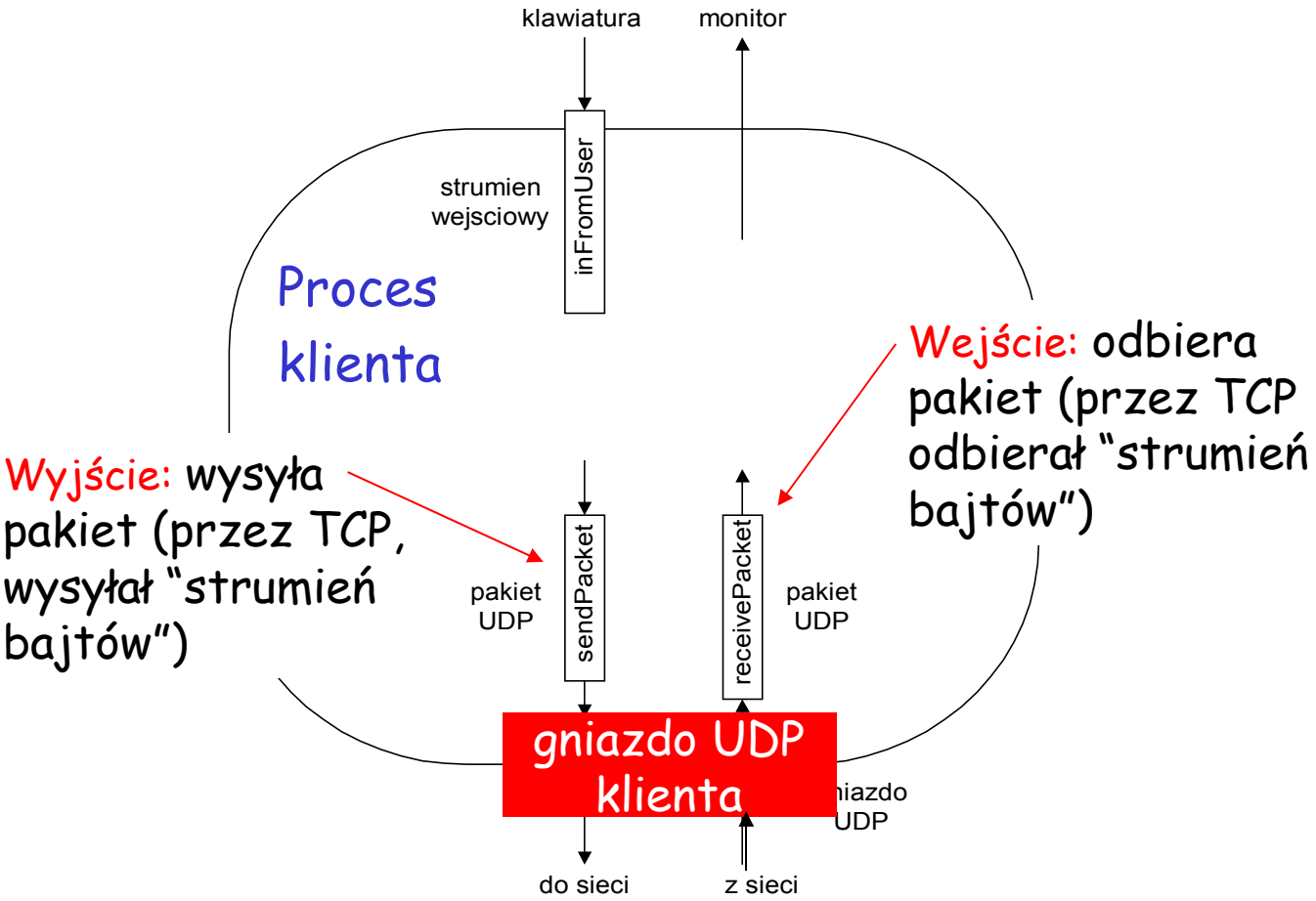
punkt widzenia programisty

UDP udostępnia zawodną komunikację ciągów bajtów ("datagramów") pomiędzy klientem i serwerem

Interakcja klient/serwer: UDP



Przykład: Klient w Javie (UDP)



Przykład: klient w Javie (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String args[]) throws Exception
    {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

Tworzy strumień wejściowy

Tworzy gniazdo klienta

Tłumaczy nazwę na adres IP używając DNS

Przykład: klient w Javie (UDP), c.d.

Tworzy datagram z danymi do wysłania, długością, adresem IP, portem

Wysyła datagram do serwera

Czyta datagram z serwera

```
DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
     new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
      new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}

}
```

Przykład: serwer w Javie (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
    {

      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
        {

        DatagramPacket receivePacket =
           new DatagramPacket(receiveData, receiveData.length);

        serverSocket.receive(receivePacket);
```

Tworzy gniazdo UDP na porcie 9876

Tworzy miejsce na otrzymany datagram

Odbiera datagram

Przykład: serwer w Javie (UDP), c.d.

```
String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket =
   new DatagramPacket(sendData, sendData.length, IPAddress,
                      port);

serverSocket.send(sendPacket);
        }
    }
}
```

Pobiera adres IP, numer portu, nadawcy pakietu

Tworzy datagram do wysłania do klienta

Wysyła datagram przez gniazdo

Koniec pętli "while", powrót i oczekiwanie na następny datagram

Budowa prostego serwera WWW

- obsługuje jedno żądanie HTTP
- przyjmuje żądanie
- parsuje nagłówek
- pobiera żądany plik z systemu plików serwera
- tworzy odpowiedź HTTP:
  - linie nagłówków + plik
- wysyła odpowiedź do klienta

- po napisaniu serwera, można żądać plików używając przeglądarki WWW
- Będzie to jedno z zadań programistycznych

Programowanie gniazd: bibliografia

W języku C (gniazda BSD):

- "Programowanie zastosowań sieciowych w systemie Unix" (R. Stevens),

http://manic.cs.umass.edu/~amldemo/courseware/intro.

Ćwiczenia w Javie:

- "All About Sockets" (tutorial Sun), http://java.sun.com/docs/books/tutorial/networking/sockets/index.htm
- "Socket Programming in Java: a tutorial," http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html

Mapa wykładu

- 2.1 Zasady budowy protokołów w. aplikacji
- 2.2 WWW i HTTP
- 2.3 DNS
- 2.4 Programowanie przy użyciu gniazd TCP
- 2.5 Programowanie przy użyciu gniazd UDP
- 2.6 Poczta elektroniczna
  - SMTP, POP3, IMAP
- 2.7 FTP
- 2.8 Dystrybucja zawartości
  - Schowki Internetowe
  - Sieci dystrybucji zawartości
- 2.9 Dzielenie plików P2P

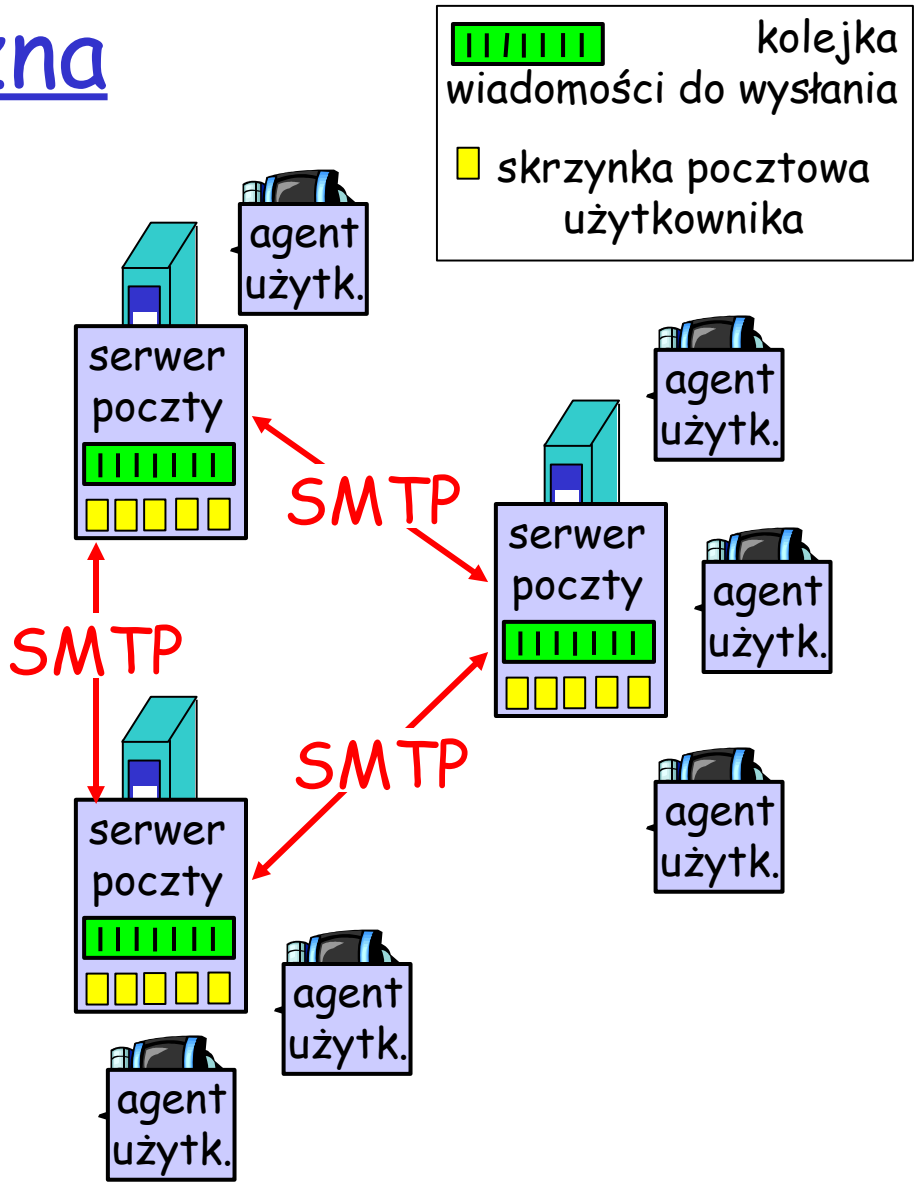
Poczta elektroniczna

Trzy główne składniki:

- agenci użytkownika
- serwery poczty
- simple mail transfer protocol: SMTP

Agent użytkownika (AU)

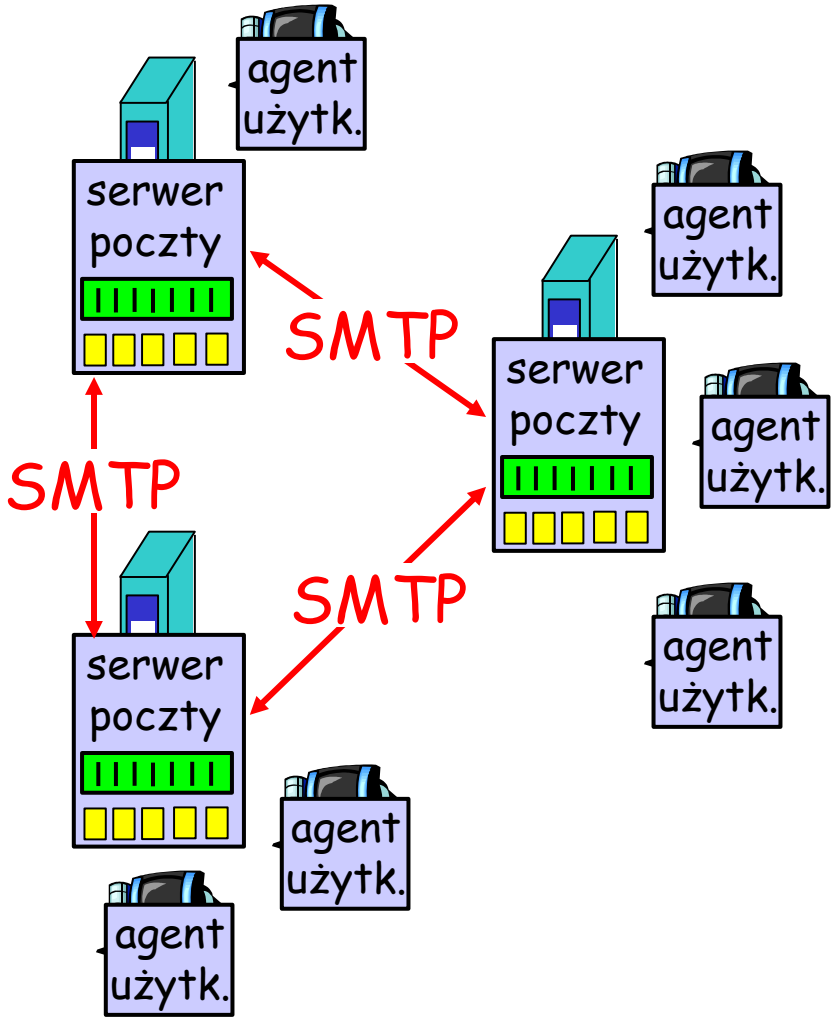
- czyli "przeglądarka poczty"
- kompozycja, edycja, czytanie poczty elektronicznej
- n.p., Eudora, Outlook, elm, Netscape Messenger
- wychodzące i przychodzące wiadomości zachowywane są na serwerze



Poczta elektroniczna: serwery poczty

Serwery poczty

- skrzynka zawiera wiadomości przychodzące od użytkowników
- kolejka wiadomości zawiera wiadomości do wysłania
- protokół SMTP wysyła pocztę pomiędzy serwerami poczty
  - tak naprawdę, jest to protokół w modelu partnerskim (ang. *peer-to-peer*)

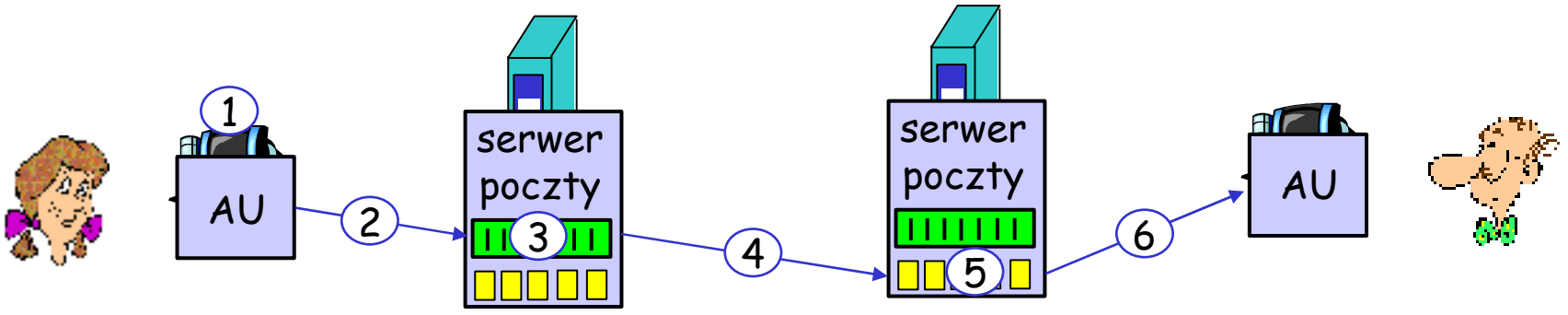


Poczta elektroniczna: SMTP [RFC 2821]

- używa TCP do niezawodnej komunikacji poczty pomiędzy serwerami, port 25
- bezpośrednia komunikacja: serwer nadawcy do serwera odbiorcy
- trzy etapy komunikacji:
  - inicjalizacja (powitanie)
  - wymiana komunikatów
  - zakończenie
- interakcja typu "polecenie/odpowiedź"
  - polecenia: tekst ASCII
  - odpowiedź: kod i fraza statusu
- komunikaty muszą być kodowane 7-bitowym ASCII

Scenariusz: Alicja wysyła pocztę do Boba

1) Alicja używa AU do skomponowania listu i wysyła go do: `bob@szkola.edu.pl`
2) AU alicji wysyła komunikat do jej serwera poczty; komunikat jest umieszczany w kolejce
3) Serwer SMTP otwiera połączenie TCP z serwerem poczty Boba
4) Serwer SMTP Alicji wysyła komunikat przez połączenie TCP
5) Serwer SMTP Boba umieszcza list w skrzynce Boba
6) Bob używa AU do przeczytania wiadomości



Przykładowa interakcja SMTP

```
S: 220 hamburger.edu
C: HELO nalesnik.pl
S: 250  Hello nalesnik.pl, pleased to meet you
C: MAIL FROM: <alice@nalesnik.pl>
S: 250 alice@nalesnik.pl... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Czy lubisz ketchup?
C:   A moze ogoreczka?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Spróbuj sam porozmawiać w SMTP:

- **`telnet nazwaserwera 25`**
- obejrzyj odpowiedź 220 serwera
- wpisz polecenia HELO, MAIL FROM, RCPT TO, DATA, QUIT

W ten sposób można wysyłać pocztę bez przeglądarki poczty

Podsumowanie o SMTP

- SMTP używa trwałych połączeń
- SMTP wymaga, żeby komunikat (nagłówek i dane) były kodowane w 7-bitowym ASCII
- Serwer SMTP używa `CRLF.CRLF` do rozpoznania końca danych

Porównania z HTTP:

- HTTP: pull (pobieranie)
- SMTP: push (wypychanie)
- Oba mają komunikaty żądań/odpowiedzi w ASCII, kody wynikowe
- HTTP: każdy obiekt zawarty w swoim własnym komunikacie odpowiedzi
- SMTP: wiele obiektów może być wysłane w wieloczęściowym komunikacie

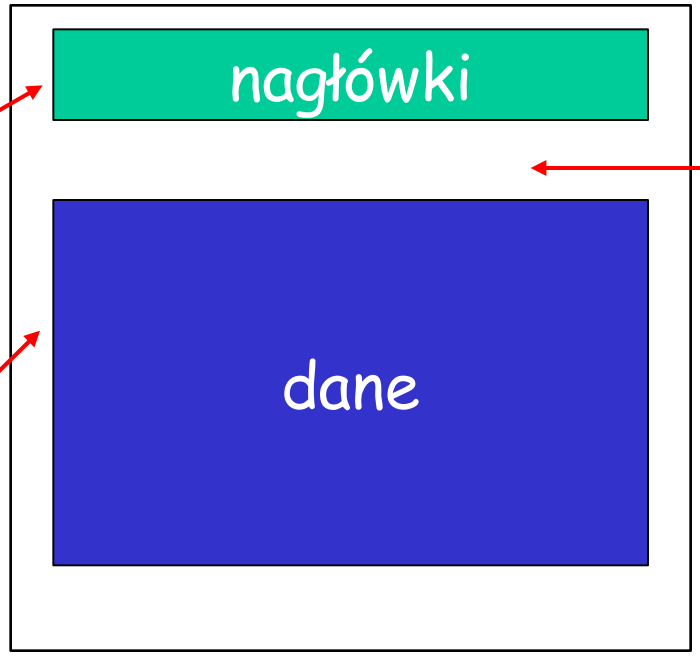
Format komunikatu poczty

SMTP: protokół dla poczty elektronicznej

RFC 822: standard opisujący format komunikatów tekstowych:

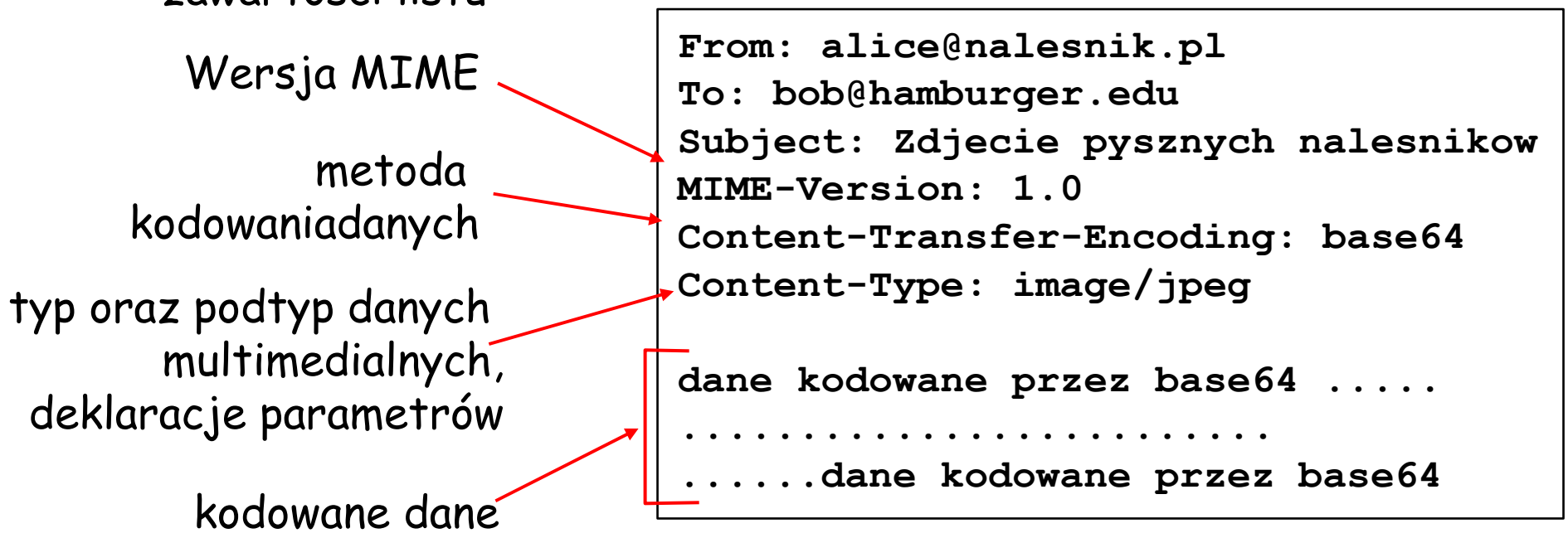
- linie nagłówków, n.p.,
  - To:
  - From:
  - Subject:

  różne od poleceń SMTP!
- dane
  - "list", tylko znaki ASCII



Format komunikatu poczty: rozszerzenia dla multimediów

- MIME: multimedia mail extension, RFC 2045, 2056
- dodatkowe linie nagłówka określają typ MIME dla zawartości listu



Typy MIME

Content-Type: typ/podtyp; parametery

Tekst

- przykładowe podtypy: **plain, html**

Obraz

- przykładowe podtypy: **jpeg, gif**

Dźwięk

- przykładowe podtypy: **basic** (kodowanie 8-bit mu-law), **32kadpcm** (kodowanie 32 kbps)

Wideo

- przykładowe podtypy: **mpeg, quicktime**

Dane aplikacji

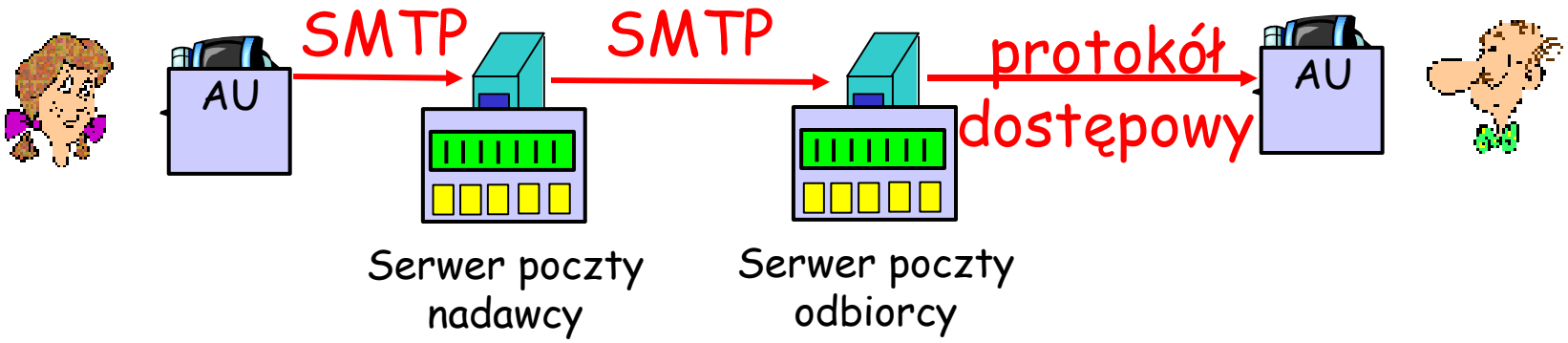
- dane, które muszą zostać przetworzone przez aplikacje, zanim można je pokazać
- przykładowe podtypy: **msword, octet-stream**

Typ Multipart (załączniki poczty)

```
From: alice@nalesnik.pl
To: bob@hamburger.edu
Subject: Zdjecie pysznych nalesnikow
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=Zalacznik

--Zalacznik
Kochany Bobie, oto zdjecie nalesnika.
--Zalacznik
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.........................
......base64 encoded data
--Zalacznik
Czy chcesz przepis?
```

Protokoły dostępu do poczty



- SMTP: dostarczanie poczty do serwera odbiorcy
- Protokół dostępowy: odbieranie poczty z serwera i zarządzanie skrzynką pocztową
  - POP: Post Office Protocol [RFC 1939]
    - uwierzytelnienie (agent <--> serwer) o pobranie poczty
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - więcej funkcji (bardziej złożony)
    - synchronizacja lokalnej skrzynki oraz skrzynki na serwerze
  - HTTP: Hotmail , Yahoo! Mail, itd.

Protokół POP3

etap uwierzytelnienia

- polecenia klienta:
  - `user:` podaję login
  - `pass:` podaję hasło
- odpowiedzi serwera
  - `+OK`
  - `-ERR`

etap transakcji, klient:

- `list:` podaj numery listów
- `retr:` pobierz list o numerze
- `dele:` usuń
- `quit:` zakończ

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass glodny
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

Protokół POP3 (cd) oraz IMAP

Więcej o POP3

- Poprzedni przykład używał tryby "pobierz i usuń".
- Bob nie może przeczytać listu ponownie, jeśli zmieni przeglądarkę
- "Pobierz i zostaw": kopie listów w wielu przeglądarkach
- POP3 jest bezstanowy pomiędzy sesjami

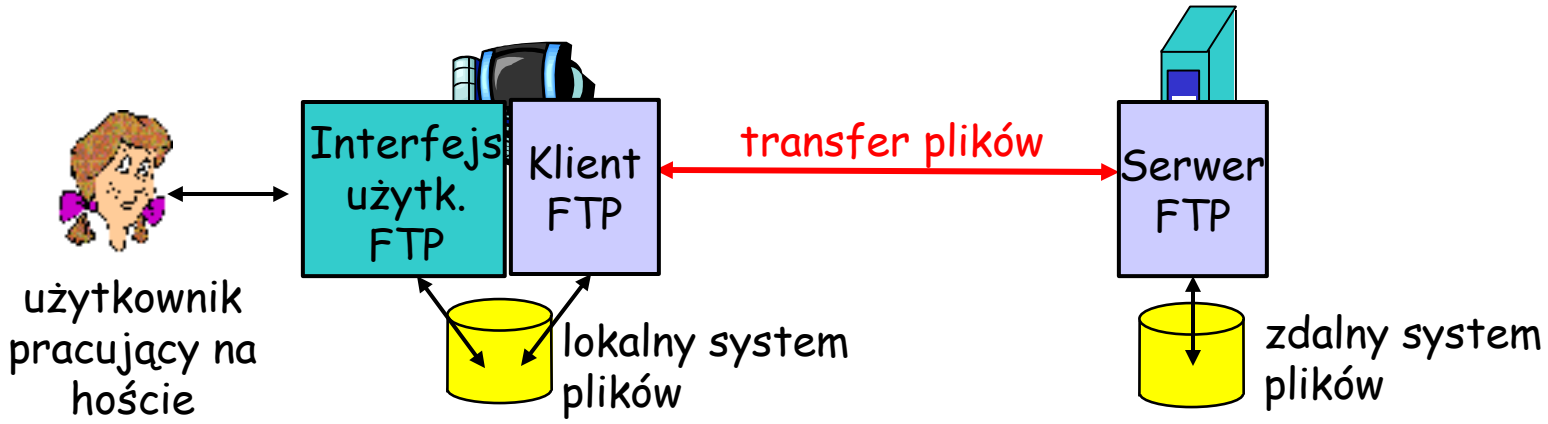
IMAP

- Wszystkie listy są w jednym miejscu: na serwerze
- Użytkownik może organizować pocztę w foldery
- IMAP zachowuje stan użytkownika pomiędzy sesjami:
  - nazwy folderów oraz przyporządkowanie listów do folderów

Mapa wykładu

- 2.1 Zasady budowy protokołów w. aplikacji
- 2.2 WWW i HTTP
- 2.3 DNS
- 2.4 Programowanie przy użyciu gniazd TCP
- 2.5 Programowanie przy użyciu gniazd UDP
- 2.6 Poczta elektroniczna
  - SMTP, POP3, IMAP
- 2.7 FTP
- 2.8 Dystrybucja zawartości
  - Schowki Internetowe
  - Sieci dystrybucji zawartości
- 2.9 Dzielenie plików P2P

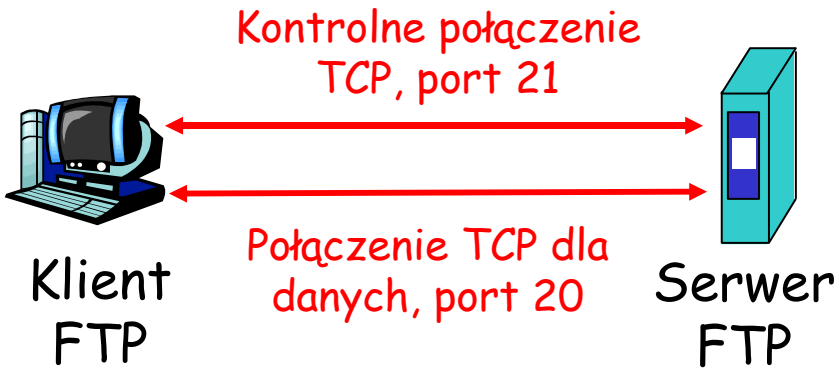
FTP: file transfer protocol



- transfer pliku z/na zdalny host
- model klient/serwer
  - *klient:* strona, która rozpoczyna transmisję
  - *serwer:* zdalny host
- ftp: RFC 959
- serwer ftp: port 21

FTP: oddzielne połączenie kontrolne i transferu plików

- Klient FTP kontaktuje się z serwerem na porcie 21 (TCP)
- Przez połączenie kontrolne, klient uzyskuje autoryzację
- Klient przegląda zdalny system plików przesyłając polecenia FTP przez połączenie kontrolne.
- Gdy serwer otrzymuje polecenie transferu pliku, otwiera połączenie TCP do klienta
- Po przesłaniu pliku, serwer zamyka nowe połączenie.



- Dla przesłania drugiego pliku, serwer otwiera drugie połączenie.
- Połączenie kontrolne: "poza pasmem"
- Serwer FTP utrzymuje "stan": aktualny katalog, wcześniejszą autoryzację

Polecenia i odpowiedzi FTP

Przykładowe polecenia:

- posyłane jako tekst ASCII przez połączenie kontrolne
- **USER *login***
- **PASS *password***
- **LIST** zwraca listę plików w aktualnym katalogu
- **RETR nazwaPliku** pobiera plik
- **STOR nazwaPliku** zapisuje plik na zdalnym hoście

Przykładowe odpowiedzi FTP

- kod i opis wyniku (jak w HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

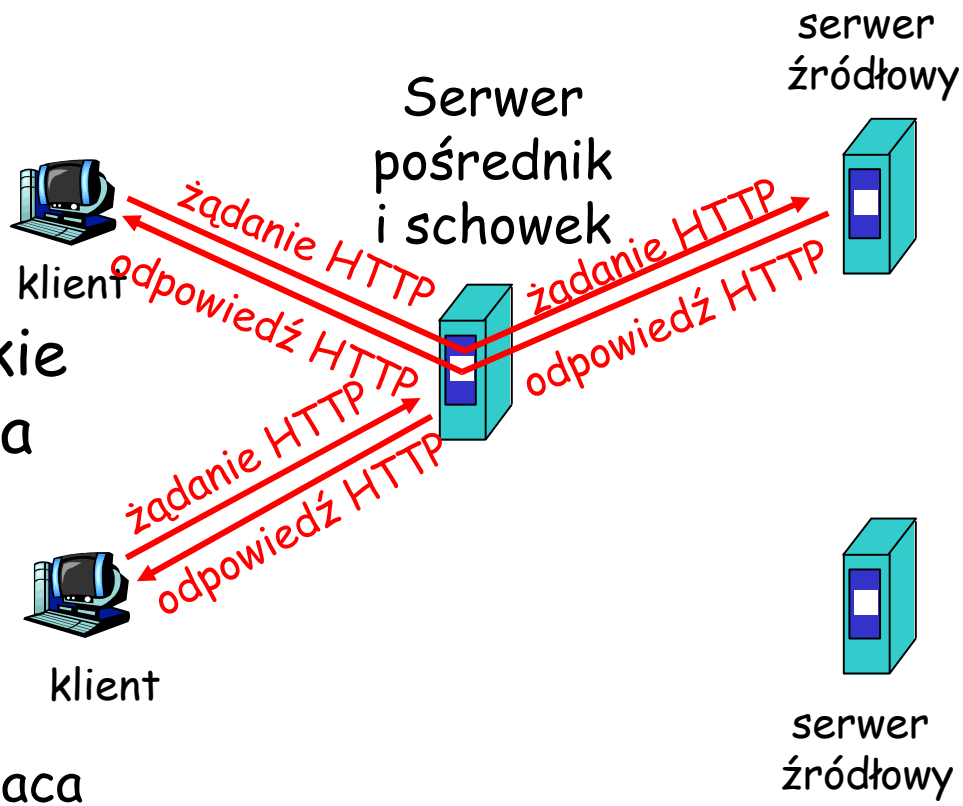
Mapa wykładu

- 2.1 Zasady budowy protokołów w. aplikacji
- 2.2 WWW i HTTP
- 2.3 DNS
- 2.4 Programowanie przy użyciu gniazd TCP
- 2.5 Programowanie przy użyciu gniazd UDP
- 2.6 Poczta elektroniczna
  - SMTP, POP3, IMAP
- 2.7 FTP
- 2.8 Dystrybucja zawartości
  - Schowki Internetowe
  - Sieci dystrybucji zawartości
- 2.9 Dzielenie plików P2P

Schowki Internetowe (serwery pośredniczące) (ang. Web caches, proxy servers)

Cel: Obsłużyć żądanie klient bez komunikacji z serwerem źródłowym

- użytkownik konfiguruje użycie serwera pośredniczącego w przeglądarce
- przeglądarka śle wszystkie żądania HTTP do schowka
  - jeśli obiekt w schowku: schowek zwraca obiekt
  - jeśli nie, schowek żąda obiektu z serwera źródłowego, następnie zwraca obiekt do klienta



Więcej o schowkach Internetowych

- Serwer-pośrednik jest zarówno klientem, jak i serwerem
- Serwer może sprawdzić zgodność obiektu przy pomocy nagłówka HTTP `If-modified-since`
  - Czy schowki powinny ryzykować i zwracać obiekty bez sprawdzania?
  - Używa się heurystyk.
- Typowo schowki są instalowane prez DI (uniwersytet, firmę, osiedlowego DI)

Po co używa się schowków?

- Zmniejszają czas oczekiwania na obsługę żądania.
- Zmniejszają ruch na łączach dostępowych instytucji.
- Duża ilość schowków w Internecie pozwala "ubogim" dostawcom zawartości na wydajne działanie

Przykład działania schowka (1)

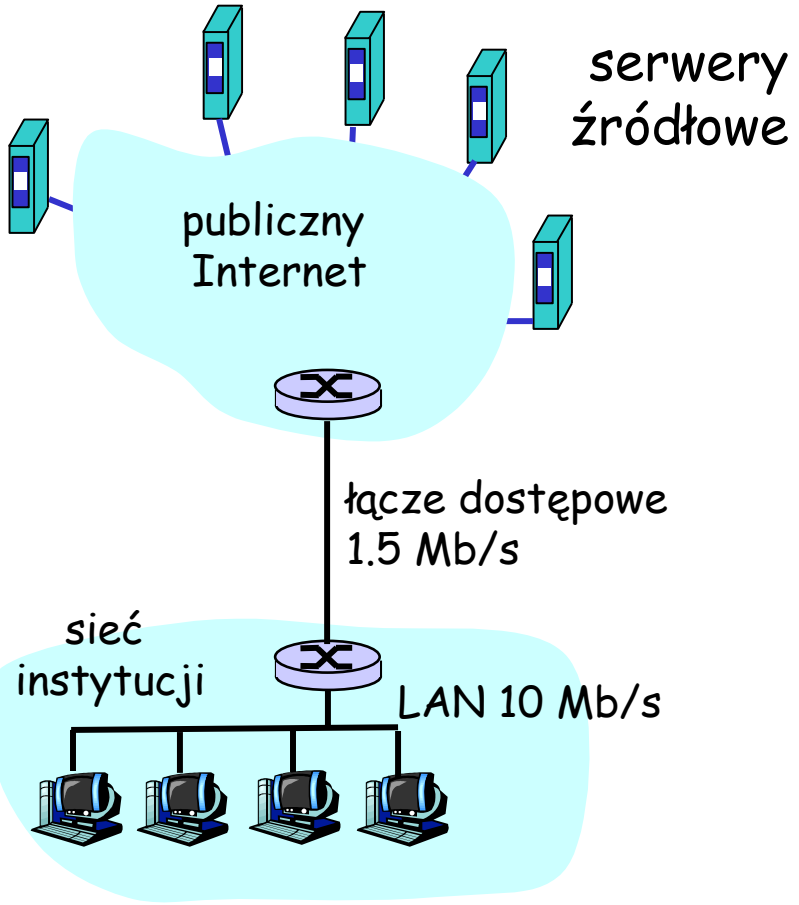
Założenia

- średni rozmiar obiektu = 100,000 bitów
- średnia ilość żądań z sieci instytucji do serwerów źródłowych = 15/sec
- opóźnienie z sieci instytucji do dowolnego serwera źródłowego i z powrotem = 2 sec

Konsekwencje

- wykorzystanie sieci LAN = 15%
- wykorzystanie łącza dostępowego = 100%
- całkowite opóźnienie = opóźnienie w Internecie + opóźnienie na łączu dostępowym + opóźnienie LAN

  = 2 s. + minuty + milisekundy



Przykład działania schowka (2)

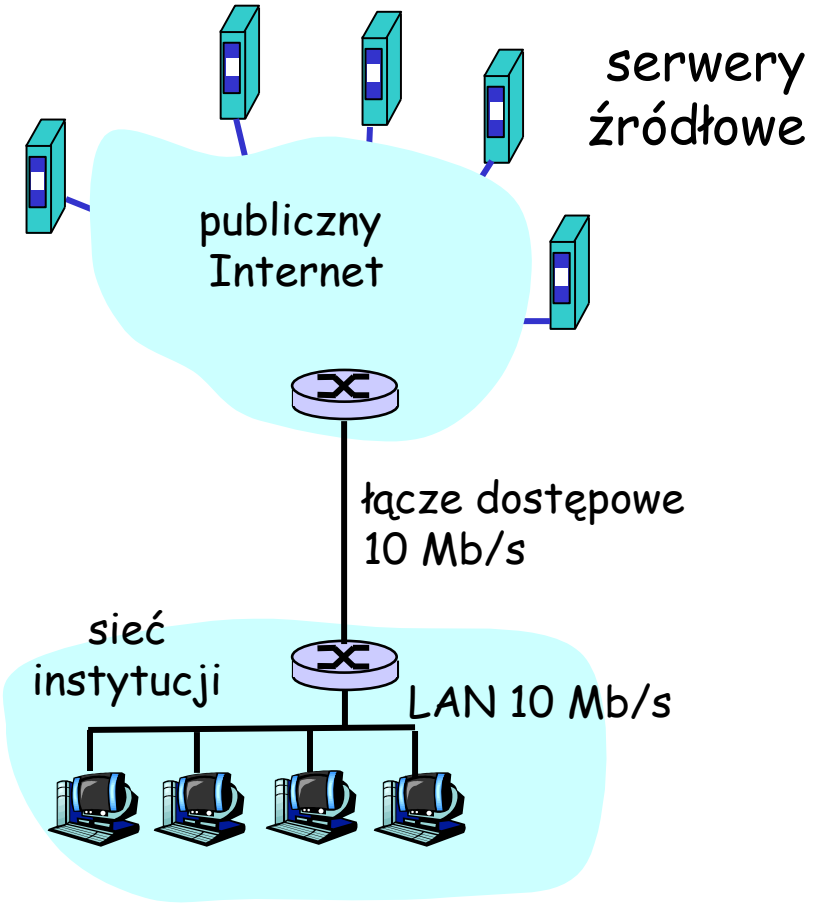
Możliwe rozwiązanie

- zwiększyć przepustowość łącza dostępowego, do, n.p., 10 Mb/s

Konsekwencje

- wykorzystanie sieci LAN = 15%
- wykorzystanie łącza dostępowego = 15%
- Całkowite opóźnienie = opóźnienie w Internecie + opóźnienie na łączu dostępowym + opóźnienie LAN

  = 2 s + ms + ms
- zwiększenie przepustowości jest często bardzo drogie



Przykład działania schowka (3)

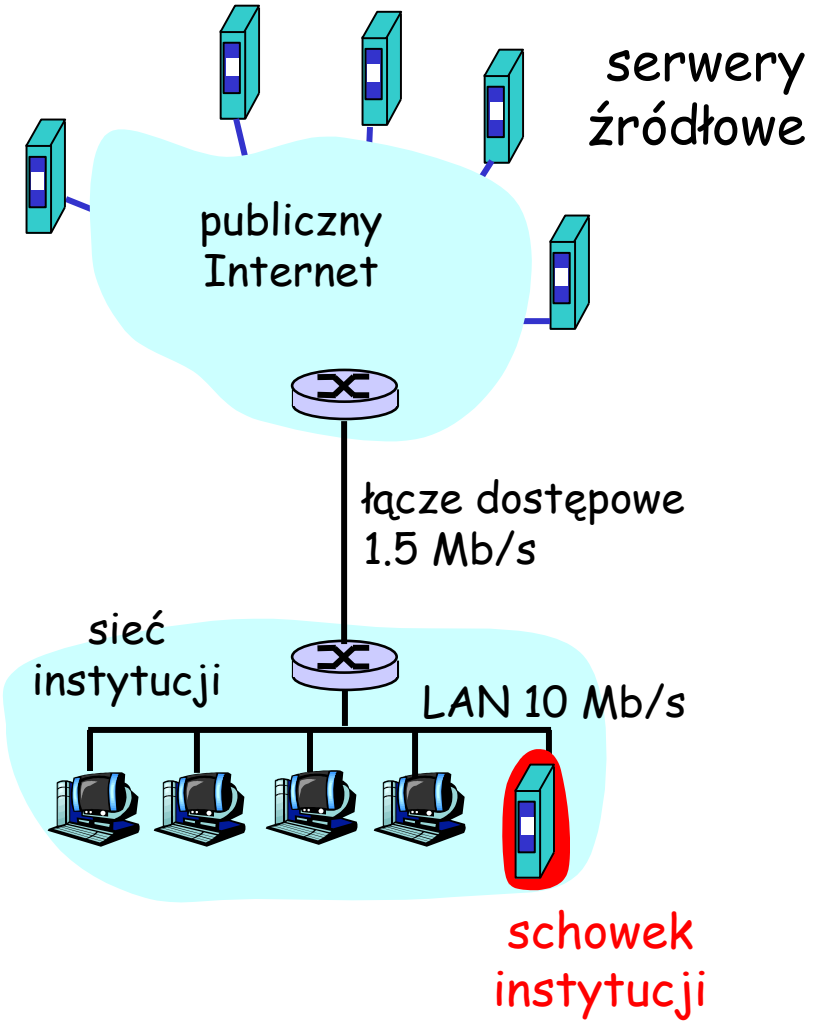
Instalacja schowka

- załóżmy częstość trafień 40%

Konsekwencje

- 40% żądań będzie obsłużona prawie natychmiast
- 60% żądań nadal obsługiwanych przez serwery źródłowe
- wykorzystanie łącza dostępowego spada do 60%, co zmniejsza opóźnienia do ok. 10 ms
- Całkowite opóźnienie = opóźnienie w Internecie + opóźnienie na łączu dostępowym + opóźnienie LAN

= 0.6*2 s + 0.6*.01 s + ms < 1.3 s

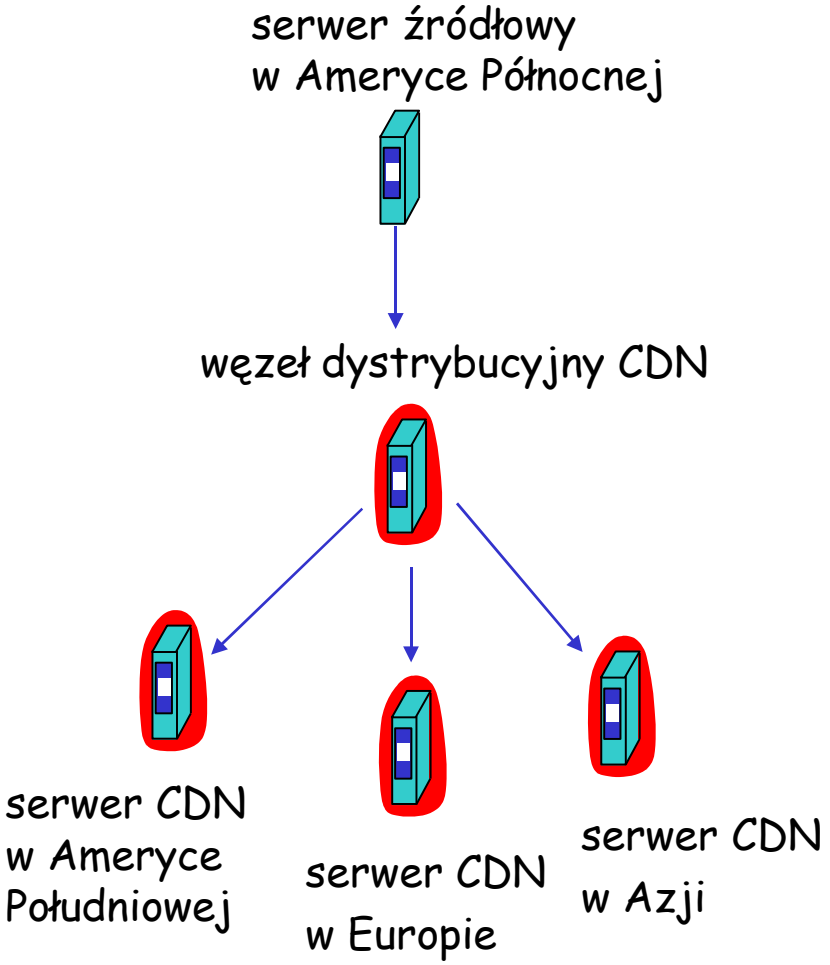


Sieci dystrybucji zawartości ang. Content Distribution Networks, (CDNs)

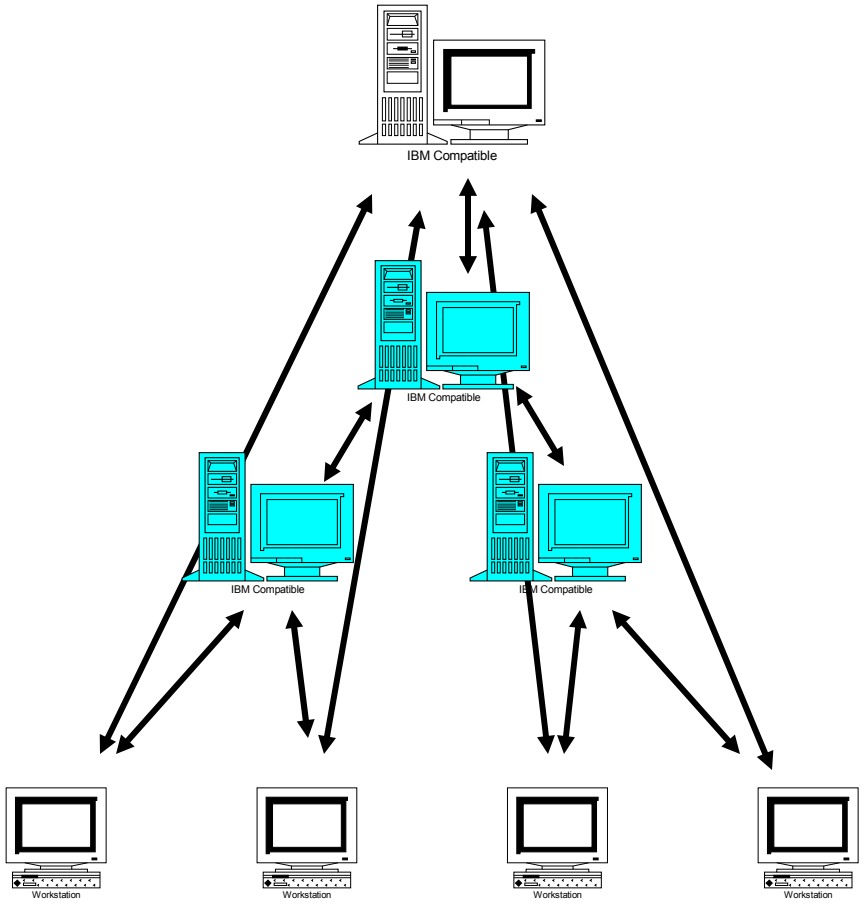
- Dostawcy zawartości są klientami sieci CDN.

Replikacja zawartości

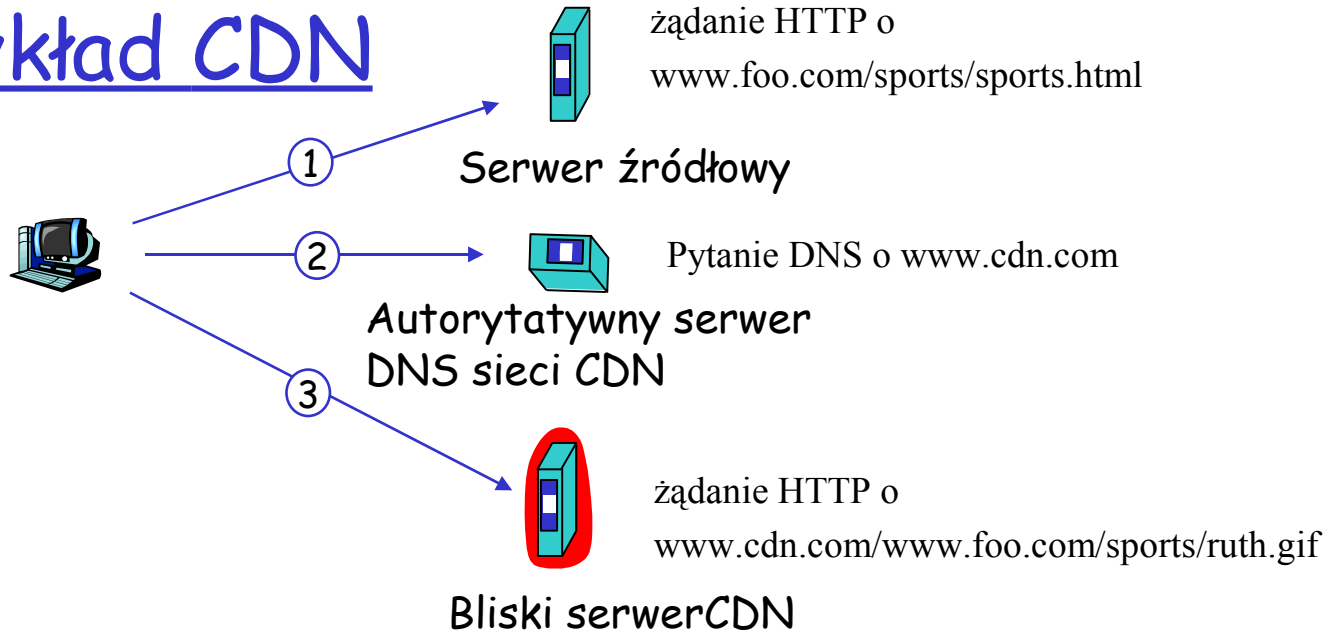
- Firma CDN instaluje setki serwerów CDN w Internecie
  - w DI niższego poziomu, blisko użytkowników
- CDN replikuje zawartość klientów w swoich serwerach. Gdy dostawca aktualizuje zawartość, CDN aktualizuje serwery



Dystrybucja zawartości informacyjnej



Przykład CDN



serwer źródłowy

- www.foo.com
- dostarcza HTML
- Zamienia:

  http://www.foo.com/sports.ruth.gif

  na

  http://www.cdn.com/www.foo.com/sports/ruth.gif

Firma CDN

- cdn.com
- dostarcza pliki gif
- używa autorytatywnego serwera DNS do kierowania żądań

Więcej o sieciach CDN

kierowanie żądań: ruting w warstwie aplikacji!

- CDN tworzy "mapę", wskazującą odległości od sieci DI do węzłów CDN
- gdy żądanie trafia do autorytatywnego serwera DNS:
  - serwer sprawdza, z jakiego DI pochodzi żądanie
  - używa "mapy" do znalezienia najlepszego serwera CDN

nie tylko strony WWW

- komunikacja strumieniowa nagranego audio/wideo
- komunikacja strumieniowa audio/wideo w czasie rzeczywistym
  - Węzły CDN tworzą sieć do dystrybucji multicast (rozsiewczej) w warstwie aplikacji

Mapa wykładu

- 2.1 Zasady budowy protokołów w. aplikacji
- 2.2 WWW i HTTP
- 2.3 DNS
- 2.4 Programowanie przy użyciu gniazd TCP
- 2.5 Programowanie przy użyciu gniazd UDP
- 2.6 Poczta elektroniczna
  - SMTP, POP3, IMAP
- 2.7 FTP
- 2.8 Dystrybucja zawartości
  - Schowki Internetowe
  - Sieci dystrybucji zawartości
- 2.9 Dzielenie plików P2P

Dzielenie plików P2P (model partnerski)

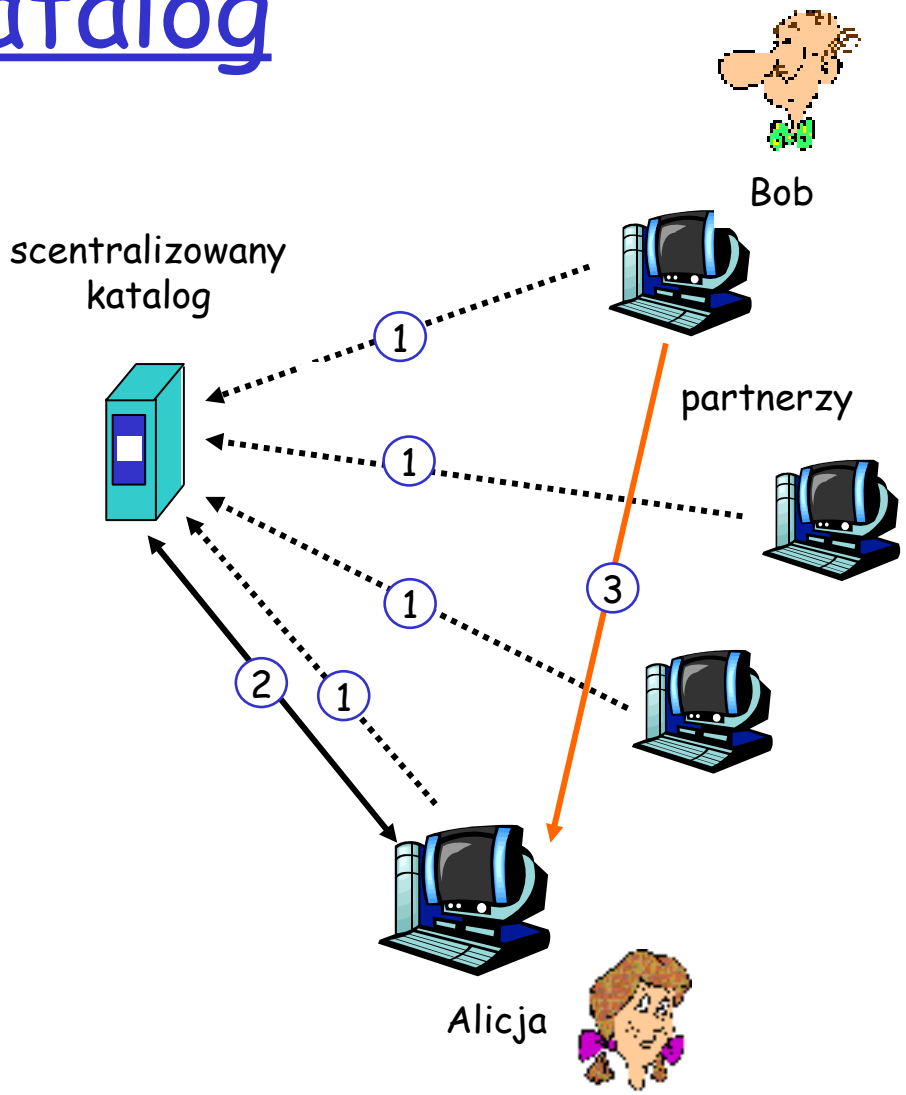
Przykład

- Alicja używa aplikacji P2P na swoim komputerze przenośnym
- Alicja łączy się z Internetem z przerwami; za każdym razem ma nowy adres IP
- Szuka piosenki "Charlie, Charlie"
- Aplikacja wyświetla partnerów, które mają piosenkę.
- Alicja wybiera jednego z partnerów, Boba.
- Plik jest kopiowany z komputera Boba na komputer Alicji: HTTP
- Dopóki Alicja jest w sieci, inni partnerzy mogą kopiować pliki od niej.

P2P: centralny katalog

projekt typu "Napster"

1) gdy partner łączy się z siecią, informuje centralny serwerer o:
   - adresie IP
   - zawartości
2) Alicja pyta o "Charlie, Charlie"
3) Alicja żąda pliku od Boba



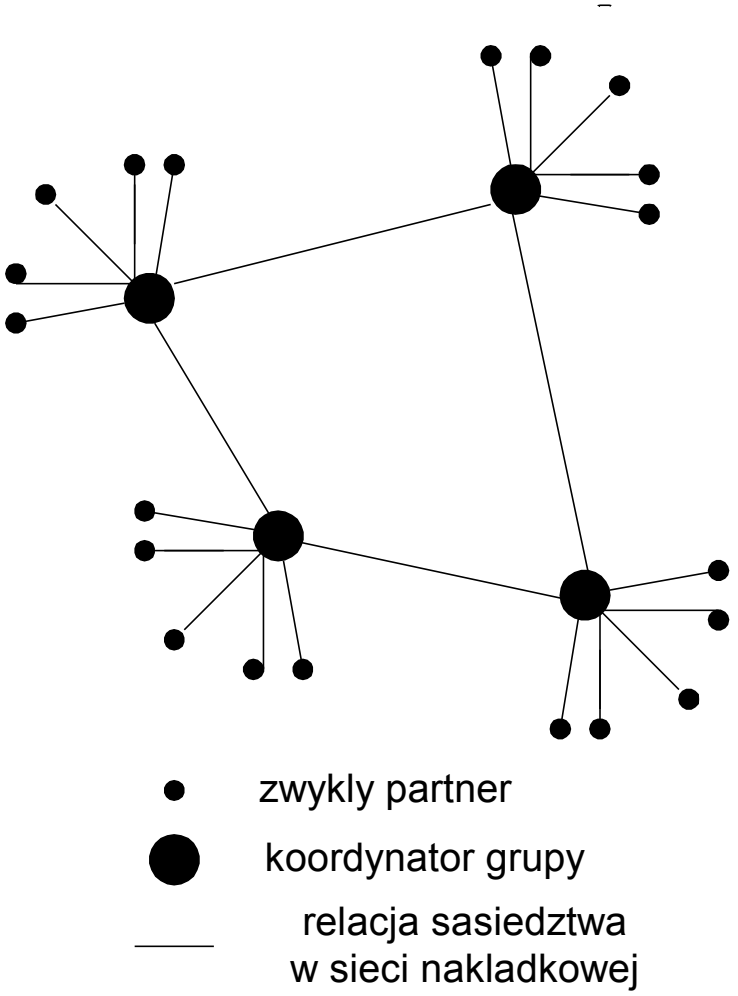
P2P: problemy z centralnym katalogiem

- Pojedynczy punkt awarii
- Wąskie gardło
- Pod kontrolą jednej organizacji

transfer plików jest rozproszony, lecz wyszukiwanie zawartości jest wysoce scentralizowane

P2P: rozproszony katalog

- Każdy partner jest koordynatorem grupy lub jest przypisany do koordynatora.
- Koordynatorzy znają zawartość wszystkich swoich dzieci.
- Partner pyta koordynatora swojej grupy; koordynatorzy mogą pytać innych koordynatorów.



Więcej o rozproszonym katalogu

sieć nakładkowa

- węzły to partnerzy
- łącza pomiędzy partnerami i ich koordynatorami
- łącza pomiędzy niektórymi koordynatorami

węzeł startowy

- łączący się partner jest przypisywany do grupy lub zostaje koordynatorem

zalety podejścia

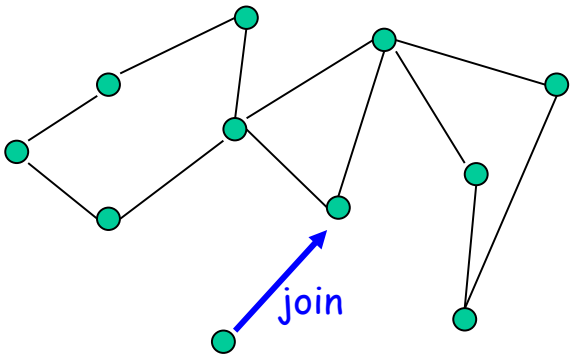
- brak centralnego katalogu
  - wyszukiwanie jest rozproszony wśród partnerów
  - Większa odporność

wady podejścia

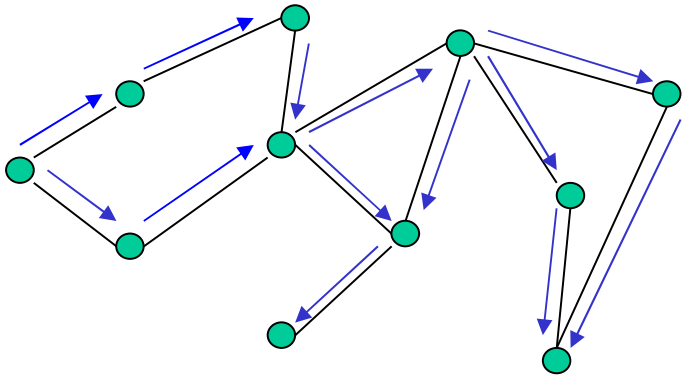
- potrzebny węzeł startowy
- koordynatorzy mogą zostać przeciążeni

P2P: zalew pytaniami

- Gnutella
- sieć nakładkowa nie ma hierarchii ani struktury
- partnerzy poznają innych przez węzeł startowy
- partnerzy wysyłają komunikat "dołącz"



- Partner wysyła pytanie do sąsiadów
- Sąsiedzi przekazują pytanie dalej
- Jeśli pytany partner ma obiekt, wysyła komunikat do pytającego partnera



P2P: więcej o zalewie pytaniami

Zalety

- partnerzy mają te same obowiązki: nie ma koordynatorów
- wysoce rozproszone
- żaden partner nie utrzymuje katalogu

Wady

- zbyt wiele ruchu w sieci z powodu pytań
- zasięg pytania: może nie znaleźć istniejącej zawartości
- węzeł startowy
- utrzymywanie sieci nakładkowej

Podsumowanie wykładów o warstwie aplikacji

Skończyliśmy wykład o aplikacjach sieciowych!

- wymagania aplikacji dotyczące usług:
  - niezawodność, przepustowość, opóźnienie
- model klient/serwer
- Usługi transportowe Internetu
  - połączeniowe, niezawodne: TCP
  - zawodne, datagramowe: UDP
- konkretne protokoły:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
- programowanie gniazd
- dystrybucja zawartości
  - schowki, sieci CDN
- sieci partnerskie (P2P)

Podsumowanie wykładów o warstwie aplikacji

Najważniejsze: uczyliśmy się o *protokołach*

- typowa wymiana komunikatów żądanie/odpowiedź:
  - klient żąda informacji lub usługi
  - serwer odpowiada informacją, kodem wynikowym
- formaty komunikatów:
  - nagłówki: zawierają informacje o danych
  - dane: komunikowana informacja
- komunikacja kontrolna i z danymi
  - w paśmie, poza pasmem
- centralne albo rozproszone
- stanowe lub bezstanowe
- komunikacja zawodna albo niezawodna
- "złożoność na brzegu sieci"
- bezpieczeństwo: uwierzytelnienie