POLITECHNIKA POZNAŃSKA

Poznan University of Technololgy

Distributed Systems Group

Sieci komputerowe - programowanie

Wykład 10: arch. programowanie

Andrzej Stroiński

andrzej.stroinski@cs.put.edu.pl

http://www.cs.put.poznan.pl/astroinski/

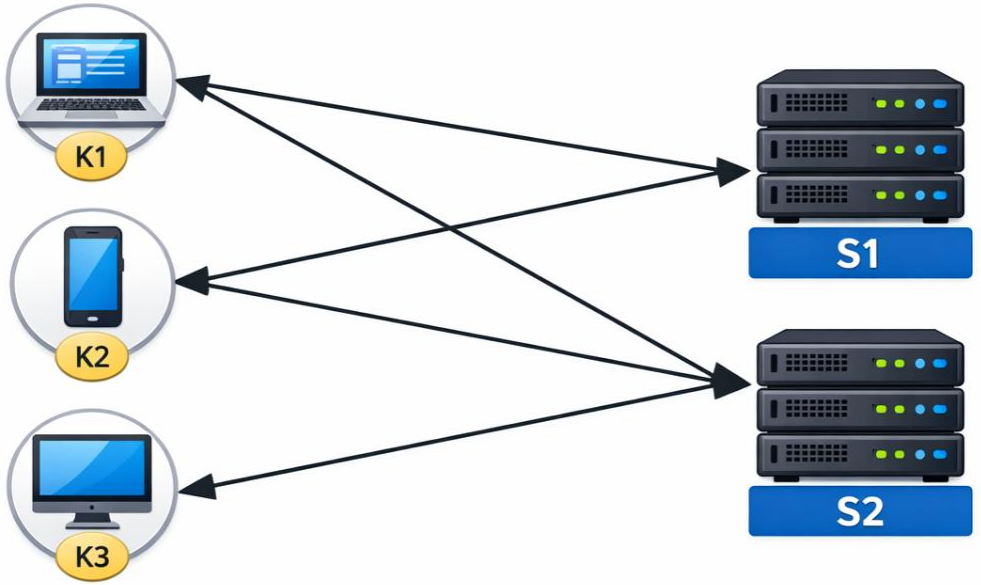
17.01.2026

Wykład przygotowany na podstawie książki i slajdów: TCP/IP Protocol suite 4ed, Behrouz A. Forouzan

Aplikacje sieciowe

Klient-Serwer

- Asymetryczna architektura
- Podział na dwie role
  - Klient – zleceniodawca, potrzebuje usługi
  - Serwer – dostarcza usługę, funkcjonalność



Aplikacje sieciowe

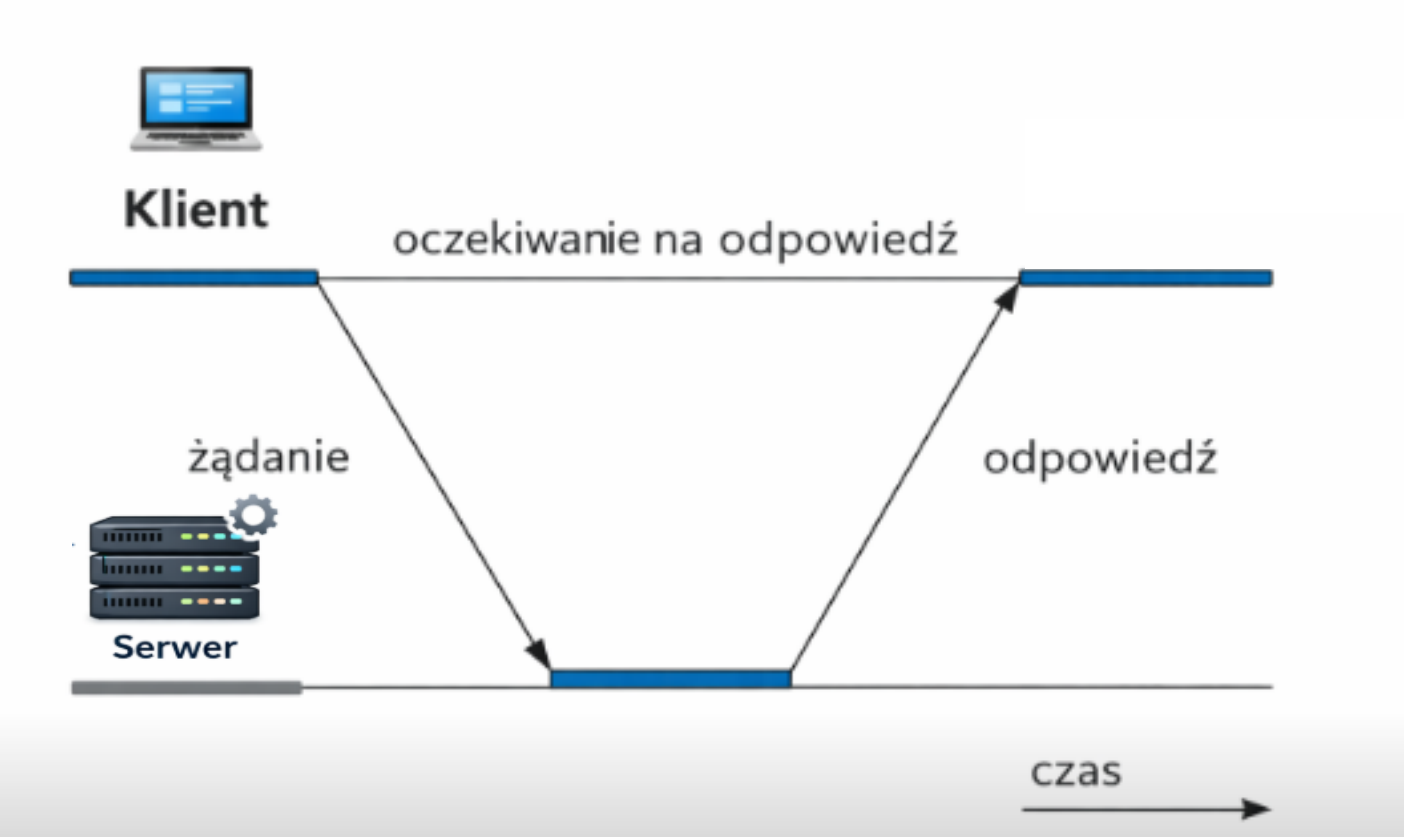
Klient-Serwer – aplikacja serwera

- Pasywny
- Czeka na żądanie (sam nie wykonuje akcji)
- Po wywołaniu obsługuje je i zwraca odpowiedź

Klient-Serwer – aplikacja klienta

- Aktywny
- Inicjuje komunikację, wysyła żądania do serwera
- Po wywołaniu serwera, oczekuje na odpowiedź

Aplikacje sieciowe



Sposób obsługi żądań

Serwer iteracyjny (sekwencyjny) – iterative server

Serwer obsługuje **jedno żądanie naraz**. W praktyce nie stosowany.

- **Jak działa:**
  - Czeka na połączenie
  - Obsługuje klienta
  - Kończy obsługę
  - Dopiero potem przyjmuje kolejnego
- **Cechy:**
  - brak równoległości
  - każdy klient blokuje serwer
- **Zalety:**
  - bardzo prosty w implementacji
  - łatwy do debugowania
- **Wady:**
  - fatalna skalowalność
  - jeden wolny klient blokuje innych

Sposób obsługi żądań

Serwer współbieżny – concurrent server

Serwer obsługuje wielu klientów jednocześnie. Przykłady: serwery HTTP (Nginx, Node.js, Apache, Kestrel), serwery gier, mikroserwisy.

- **Jak działa:**
  - Każde połączenie w:
    - osobnym wątku
    - osobnym procesie
    - puli wątków
- **Cechy:**
  - równoległa obsługa klientów
  - lepsze wykorzystanie CPU
- **Zalety:**
  - skalowalność
  - brak blokowania
- **Wady:**
  - wyścig pomiędzy klientami
  - większa złożoność

Stan serwera

Serwer bezstanowy – stateless server

Serwer **nie zapamiętuje stanu klienta** pomiędzy żądaniami. Przykłady: REST API, HTTP bez sesji, etc.

- **Jak działa:**
  - każde żądanie jest kompletne (wszystkie dane do obsługi dostępne w żądaniu)
  - brak sesji po stronie serwera
- **Cechy:**
  - każde żądanie niezależne
  - Łatwo skalowalne
- **Zalety:**
  - bardzo dobra skalowalność
  - łatwy load balancing
  - odporność na awarie
- **Wady:**
  - klient musi przesyłać kontekst
  - trudniejsze logiki biznesowe

Stan serwera

Serwer stanowy – stateful server

Serwer przechowuje informacje o kliencie. Przykłady: serwery gier, WebSocket servers, FTP, Telnet, systemy transakcyjne

- **Jak działa:**
  - Sesje
  - kontekst połączenia
  - dane użytkownika w pamięci
- **Cechy:**
  - kolejne żądania zależne od poprzednich
- **Zalety:**
  - prostsze protokoły
  - naturalne dla gier, czatów
- **Wady:**
  - trudny load-balancing
  - problemy z replikacją
  - migracja sesji

Rodzaj klienta

Cienki klient – thin client

Logika i dane są głównie po stronie serwera. Przykład: aplikacje webowe, systemy terminalowe, cloud gaming

- **Cechy klienta:**
  - UI
  - wysyłanie żądań
  - wyświetlanie wyników
- **Zalety:**
  - łatwe aktualizacje
  - większe bezpieczeństwo
- **Wady:**
  - większe obciążenie serwera
  - zależność od sieci

Rodzaj klienta

„Bogaty” klient – ang. rich client

Duża część logiki po stronie klienta. Przykład: plikacje webowe, systemy terminalowe, cloud gaming

- **Cechy klienta:**
  - Walidacja
  - Symulacje
  - Cache
  - logika biznesowa
- **Zalety:**
  - mniejsze opóźnienia
  - mniejsze obciążenie serwera
- **Wady:**
  - trudniejsze aktualizacje
  - ryzyko cheatów i manipulacji

Gniazda BSD

- Począwszy od wersji 4.1 systemu UNIX BSD, wprowadzono do niego obsługę protokołów TCP/IP wraz z interfejsem dostępu dla programów o nazwie **gniazda** (ang. *sockets*). Interfejs ten jest także dostępny w systemach Linux, a jego zadaniem jest pośredniczenie pomiędzy programami użytkowników, a implementacją stosu TCP/IP.
- Do dyspozycji programistów oddano **zestaw funkcji bibliotecznych**, które służą do obsługi komunikacji pomiędzy procesami działającymi na różnych węzłach w sieci (możliwa jest także komunikacji pomiędzy procesami działającymi lokalnie), które składają się na interfejs sieci. Funkcje te ukrywają przed programistą warstwę transportową oraz wymagają utworzenia specjalnego punktu końcowego kanału komunikacji, które nazywany jest ***gniazdem***. W procesie systemowym gniazdo jest traktowane jak plik specjalny, po jego utworzeniu programista **otrzymuje deskryptor**, co pozwala na realizację funkcji odczytu i zapisu, w sposób analogiczny jak na pliku (np. **read()** i **write()**).

Model oraz trybu komunikacji

- Nawiązanie połączenia i rozpoczęcie komunikacji wymaga aby jeden z uczestniczących procesów oczekiwał na zgłoszenia; wówczas inny proces może nawiązać z nim komunikację, w dowolnym, obsługi przez siebie momencie.
- Proces, który oczekuje na połączenie nazywany jest serwerem (ang. server). Serwer może działać według dwóch schematów:
  - po odebraniu połączenia od innego procesu następuje przetwarzanie przyjętego zgłoszenia oraz ewentualna odpowiedź, po tym serwer przechodzi w stan dalszego oczekiwania na nowe zgłoszenia, schemat taki nazywany jest iteracyjnym,
  - odbieranie połączenia od klientów obsługiwane są przez serwer jednocześnie, np. poprzez wykorzystanie procesów potomnych lub wątków, główny proces serwera w tym czasie może dalej nasłuchiwać nowych połączeń; model ten nazywany jest współbieżnym.
- Proces, który nawiązuje połączenie z serwerem nazywany jest klientem. Proces ten musi posiadać informacje o adresie IP serwera oraz porcie TCP lub UDP, na którym serwer nasłuchuje. Proces serwera może uzyskać informacje o kliencie, dopiero po nawiązaniu przez tego połączenia.
- Interfejs gniazd obsługuje dwa tryby komunikacji: połączeniowy, za pomocą protokołu transportowego TCP (RFC 793), oraz bezpołączeniowy, z wykorzystaniem protokołu transportowego UDP (RFC 768).

Gniazda BSD

- pdf

Sun RPC

- pdf

Sun RPC

- pdf

Peer-to-Peer

- Peer-to-Peer to architektura sieciowa, w której:
  - nie ma centralnego serwera,
  - każdy węzeł (peer) jest jednocześnie klientem i serwerem,
  - węzły komunikują się bezpośrednio między sobą.

- Każdy peer może:
  - inicjować połączenia,
  - udostępniać zasoby,
  - przetwarzać dane.

„wszyscy są równi – brak centrum systemu”

Typy sieci Peer-to-Peer

- P2P (pure P2P)
  - brak serwerów centralnych
  - pełna decentralizacja
  - przykłady: wczesny Gnutella, blockchain

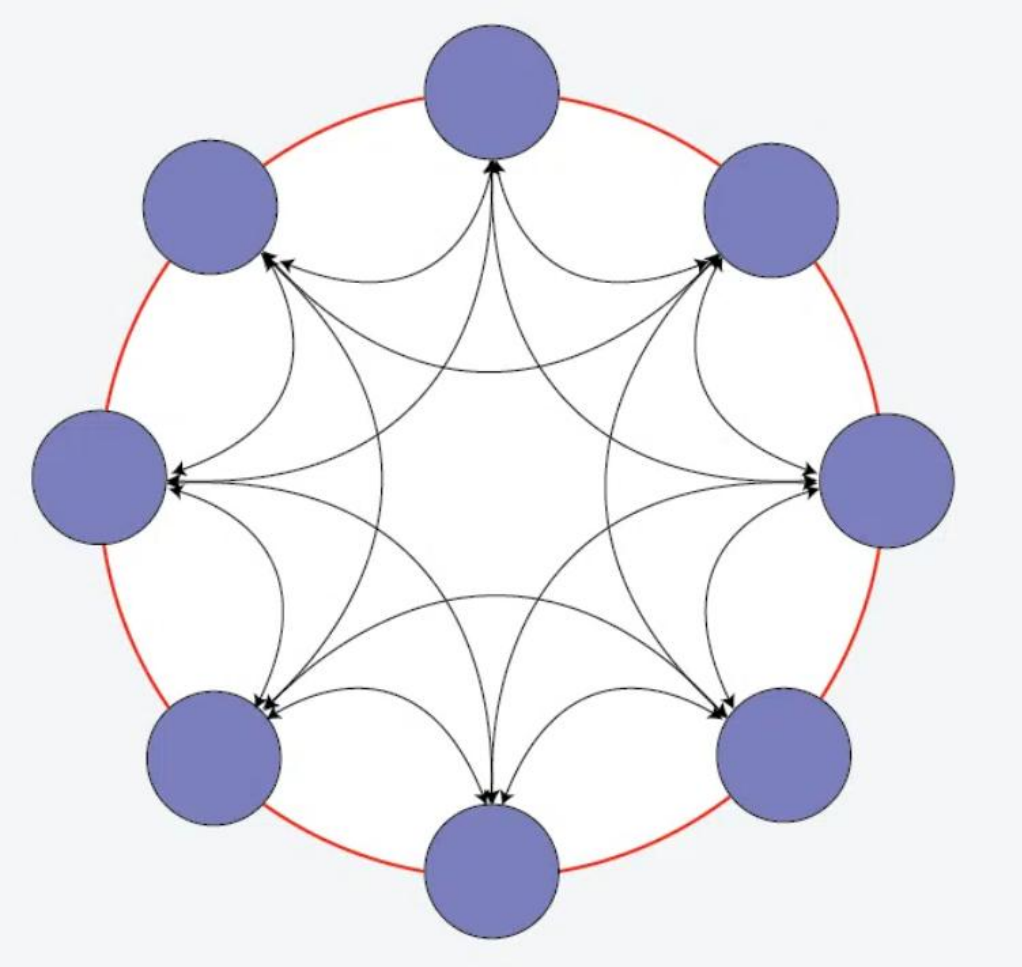
- Hybrydowe P2P
  - serwer centralny do wyszukiwania/koordynacji
  - dane przesyłane bezpośrednio między peerami
  - przykłady: BitTorrent, gry multiplayer, WebRTC

Typy sieci Peer-to-Peer

- Structured Peer-to-Peer Systems (P2P)
  - Zorganizowane systemy rozproszone bez centralnego serwera
  - Structured P2P to architektura sieci, w której węzły i dane są rozmieszczane według ściśle określonych algorytmów, zapewniających szybkie wyszukiwanie, skalowalność i odporność na awarie.

- Kluczowe cechy
  - Overlay network – logiczna, wirtualna topologia nad siecią fizyczną
  - Distributed Hash Table (DHT) – przechowywanie danych jako klucz → wartość
  - Przewidywalne rozmieszczenie danych (consistent hashing)
  - Logarytmiczne wyszukiwanie (O(log N))
  - Replikacja danych → fault tolerance
  - Automatyczna skalowalność i równoważenie obciążenia

Typy sieci Peer-to-Peer



Typy sieci Peer-to-Peer

- Działanie
  - Każdy węzeł ma unikalne ID
  - Dane trafiają do węzłów o najbliższych ID
  - Routing odbywa się po ściśle zdefiniowanych trasach
  - Sieć sama się organizuje przy dołączaniu/odłączaniu węzłów

- Przykładowe algorytmy
  - Chord – pierścień, finger tables
  - Kademlia – metryka XOR, k-buckets
  - Pastry / Tapestry – routing prefiksowy

Typy sieci Peer-to-Peer

- Zastosowania
  - BitTorrent, IPFS (dystrybucja plików)
  - Blockchain (Bitcoin, Ethereum)
  - Rozproszone bazy danych (Cassandra)
  - CDN, IoT, systemy kolaboracyjne (Git)

Typy sieci Peer-to-Peer

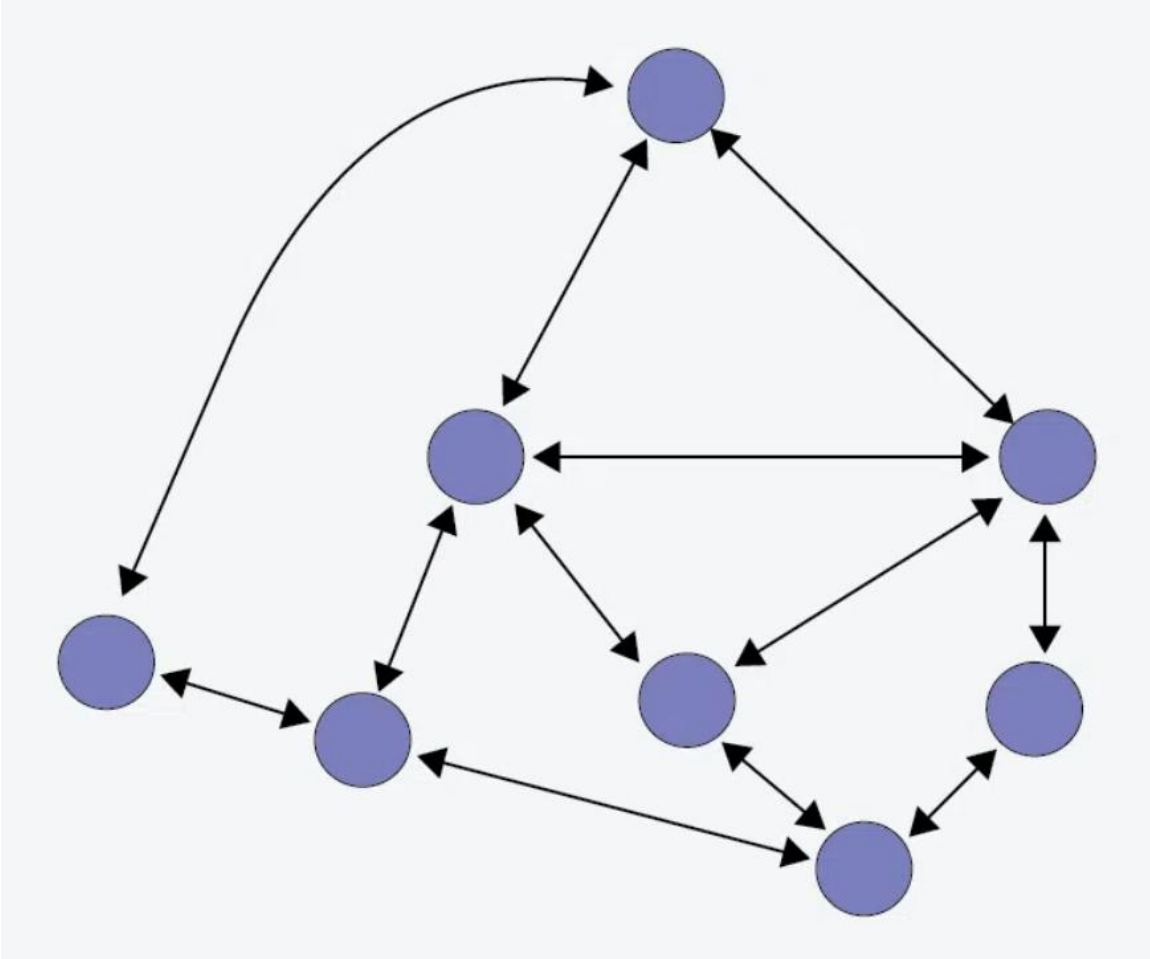
- Unstructured Peer-to-Peer Systems (P2P)
  - węzły łączą się **losowo**,
  - nie istnieje globalna struktura ani mapa sieci,
  - dane **nie mają przewidywalnego miejsca** przechowywania.
  - Sieć powstaje spontanicznie i dynamicznie.

- Kluczowe cechy
  - dynamiczna topologia (wysoki churn)
  - brak centralnego zarządzania
  - prosta implementacja
  - wysoka elastyczność
  - brak deterministycznego wyszukiwania

Typy sieci Peer-to-Peer

- Podstawowe mechanizmy
  - losowe połączenia między węzłami
  - rozproszona lista sąsiadów
  - bezpośrednie udostępnianie zasobów
  - wyszukiwanie przez rozgłaszanie zapytań
- Typowy scenariusz
  - Węzeł dołącza do dowolnych peerów
  - Buduje lokalną listę sąsiadów
  - Rozsyła zapytania do sąsiadów
  - Odbiera odpowiedzi
  - Pobiera dane bezpośrednio
  - Sieć nic „nie wie” globalnie — wiedza jest lokalna.

Typy sieci Peer-to-Peer



Typy sieci Peer-to-Peer

- Query Flooding (rozgłaszanie zapytań)
  - zapytanie wysyłane do wszystkich sąsiadów
  - sąsiedzi przekazują je dalej
  - mechanizm TTL ogranicza liczbę przeskoków

- Superpeers
  - węzły o większych zasobach
  - obsługują wyszukiwanie i routing
  - agregują informacje o peerach

Typy sieci Peer-to-Peer

- Query Flooding (rozgłaszanie zapytań)
  - zapytanie wysyłane do wszystkich sąsiadów
  - sąsiedzi przekazują je dalej
  - mechanizm TTL ogranicza liczbę przeskoków

- Superpeers
  - węzły o większych zasobach
  - obsługują wyszukiwanie i routing
  - agregują informacje o peerach

Typy sieci Peer-to-Peer

- Zalety
  - bardzo elastyczne
  - odporne na losowe awarie
  - proste w budowie
  - dobre do sieci ad-hoc
  - szybki start sieci
- Wady
  - flooding → wysoki ruch
  - brak gwarancji znalezienia danych
  - rosnące opóźnienia
  - trudna lokalność danych
  - bottlenecki superpeerów
  - wrażliwość na ataki

Typy sieci Peer-to-Peer

- Przykłady
  - wymiana plików
  - zdecentralizowane komunikatory
  - systemy kryzysowe
  - sieci IoT i mobilne

Dziękuje za uwagę. Pytania?