

# Основе веб програмирања

Борисав Живановић (borisavz)

18. децембар 2022.

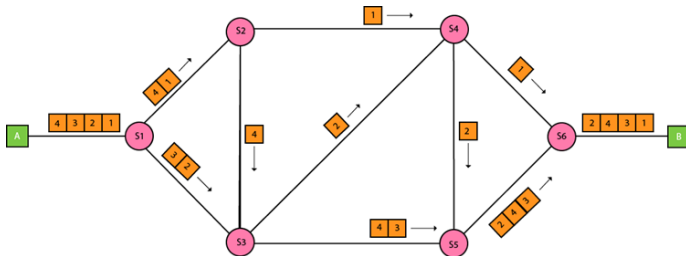
# Садржај

- 1 Основни појмови мрежног програмирања
- 2 Клијент-сервер архитектура
- 3 Еволуција веб апликација
- 4 HTTP протокол
- 5 Рад са базом података
- 6 Архитектура веб апликације
- 7 Аутентификација и ауторизација

# Packet switching I

- Потребно је да поруку пошаљемо примаоцу
- Директна веза са сваким примаоцем није остварива
- Идеја: повезивање пошиљаоца/примаоца у мрежу, дељење комуникационог канала
- Решење: **комутација пакета (packet switching)**
  - Поруку изделимо на пакете
  - Пакетима додамо заглавље (header) са адресом пошиљаоца и примаоца
  - Систем зна путање до примаоца
  - Поруку шаљемо пакет по пакет
  - Само један пакет заузима комуникациони канал
  - Пакети могу да путују различитим путањама кроз мрежу, да дођу у различитом редоследу до примаоца, или да нестану

# Packet switching II

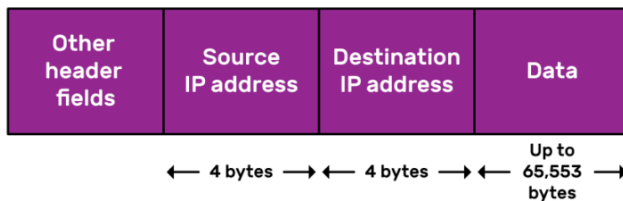


Слика: комутација пакета (packet switching)

# Internet Protocol I

- Како би комуницирали у мрежи, потребно је да сваки учесник у комуникацији има додељену **јединствену** адресу
- Поруци придружујемо **заглавље (header)** које садржи:
  - Адресу пошиљаоца (source address)
  - Адресу примаоца (destination address)
  - Додатна поља (верзија IP протокола, flags, TTL, checksum, ...)
- Захваљујући овом заглављу систем зна коме да проследи поруку
- У одговори су адресе пошиљаоца и примаоца **замене**не!

# Internet Protocol II

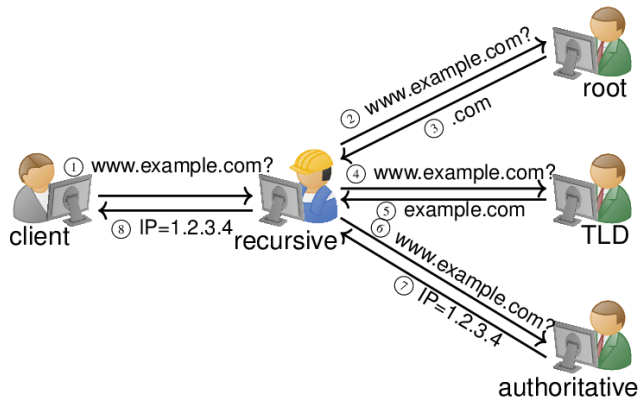


Слика: упрошћена структура IP пакета

# DNS I

- Проблем: све више сервера на мрежи
- Није практично памтити сваку адресу у бројчаном облику
- Идеја: систем за придруживање имена, сличан телефонском именику
- Решење: **DNS (Domain Name System)**
  - IP адреси додељујемо симболичко име (домен)
  - Домени су хијерархијски (структура стабла)
  - DNS је одговоран за одређени део хијерархије
  - Као одговор враћа IP адресу или адресу одговорног DNS сервера
  - Морамо знати IP адресу DNS сервера!

## DNS II



Слика: DNS упит



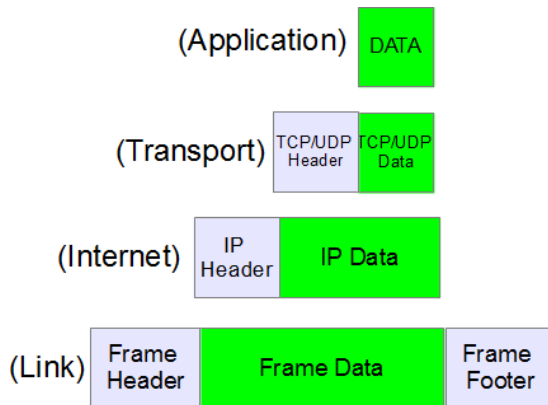
# Transmission Control Protocol I

- Решили смо проблем адресирања уређаја на мрежи...
- ...али нисмо проблеме редоследа пристиглих пакета и нестајања пакета
- Додатни проблем: шта ако имамо више мрежних апликација на истом рачунару, како да проследимо поруку одговарајућој апликацији?

# Transmission Control Protocol II

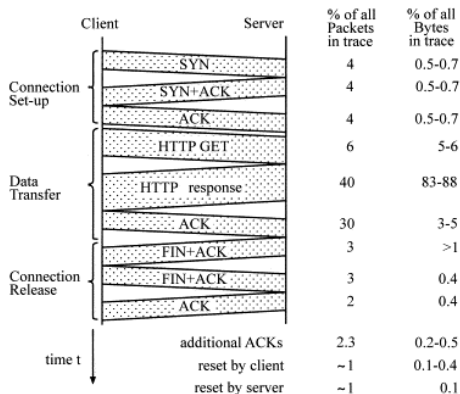
- Решење: **TCP (Transmission Control Protocol)**
  - Додајемо додатно заглавље на нашу поруку
  - Заглавље садржи source и destination port (слично адреси пошilhaоца и примаоца, али се односи на апликацију), sequence number (редослед поруке)
  - Уколико пакет нестане, шаље се поново
  - Оперативни систем осигурава да само једна апликација користи одређени порт

# Transmission Control Protocol III



Слика: енкапсулација пакета

# Transmission Control Protocol IV

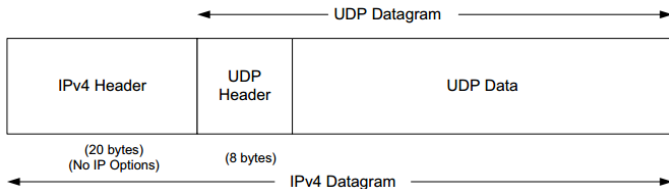


Слика: Ток TCP комуникације

# User Datagram Protocol I

- Успостављање конекције траје одређено време
- За поруке које стају у један пакет, можемо користити једноставнији **UDP (User Datagram Protocol)**
- Задржавамо адресирање апликација, али губимо гаранцију испоруке
- DNS користи UDP
- Омогућава изградњу протокола који имају гаранције испоруке
  - пример: HTTP3/QUIC

# User Datagram Protocol II



Слика: енкапсулација пакета

# User Datagram Protocol III



Слика: Садржај заглавља

# Однос између чворова

- До сада смо говорили искључиво о чворовима који учествују у комуникацији
- Видели смо да један чвор започиње комуникацију, а други даје одговор
- У раду уочавамо две врсте односа између чворова:
  - **peer-to-peer**: обе стране су подједнако важне у комуникацији
  - **client-server**: клијент се обраћа серверу за податке или обављање акције



# Клијент-сервер архитектура I

- Модел настао још раних дана рачунарства
- Рачунари су били велики и скупи
- Било је потребно омогућити дељење ресурса између више корисника
- Клијенти су били далеко једноставнији, главна намена је била слање команде и испис резултата
- Данас је овај приступ познат као **thin-client**

# Клијент-сервер архитектура II



Слика: PDP-7 (рачунар)

# Клијент-сервер архитектура III



Слика: DEC VT100 (терминал)

# Клијент-сервер архитектура IV

- Кроз године, рачунарска моћ је расла
- Ово је довело до појаве **PC (Personal Computer)**
  - користи се непосредно
  - без конукурентних корисничких сесија
- Потреба за централним сервером и даље није потпуно избачена, али је могућа далеко већа интерактивност
- Данас је овај приступ познат као **thick-client**
  - пример: Google Docs

# Еволуција веб апликација I

- **World Wide Web (WWW)** је изумео Тим Бернерс-Ли у CERN-у
- Оригинална замисао је била систем за дељење докумената
- Језик докумената: **HTML (HyperText Markup Language)**
- Протокол за комуникацију: **HTTP (HyperText Transfer Protocol)**
- Иницијално садржај је био статички (могуће је прегледање искључиво предефинисаних докумената)
- Убрзо су уочени недостаци и настала је потреба за динамичким садржајем

# Еволуција веб апликација II

- Идеја: чувати садржај у бази података и на основу њега динамички генерисати HTML документе
- Постоје два решења:
  - **server-side render**: HTML документ генеришемо користећи шаблон и вредности из базе података
  - **client-side render**: са сервера учитавамо основни HTML и JavaScript код, након тога размењујемо JSON објекте

# Еволуција веб апликација III

```
<!DOCTYPE html>
<html>

  <head>
    <title>Page Title</title>
  </head>

  <body>
    <h2>Heading Content</h2>
    <p>Paragraph Content</p>
  </body>

</html>
```

Слика: HTML документ

# Еволуција веб апликација IV

```
1 {  
2   "string": "Hi",  
3   "number": 2.5,  
4   "boolean": true,  
5   "null": null,  
6   "object": { "name": "Kylie", "age": 24 },  
7   "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],  
8   "arrayOfObjects": [  
9     { "name": "Jerry", "age": 28 },  
10    { "name": "Sally", "age": 26 }  
11  ]  
12 }  
13
```

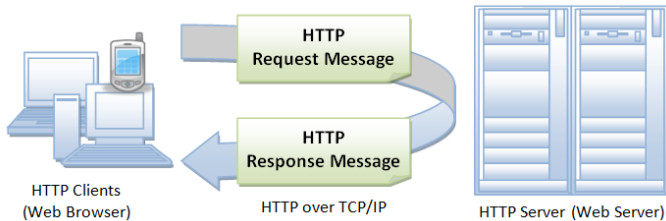
Слика: JSON објект



# HTTP протокол

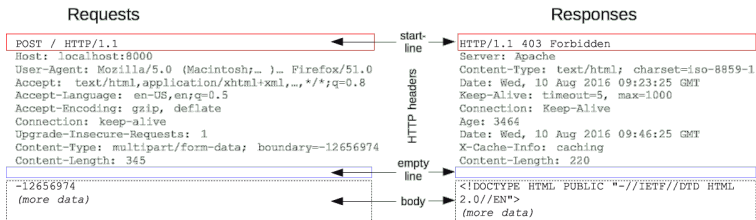
- Текстуални протокол (поруке једноставно могу да читају и људи)
- Користи TCP (гаранција испоруке је неопходна како би протокол успешно функционисао!)
- Подразумевани порт: 80 (HTTP), 443 (HTTPS)
- Stateless протокол
  - неопходно је придружити додатне информације уз сваки захтев како би пратили корисничку сесију
  - обично преко header-a
- Путања означава **ресурс** у систему
  - додатни атрибути кроз **query params**
- Метода означава **акцију** коју желимо да извршимо над ресурсом
- Статусни код означава да ли је акција успешно извршена, и ако није, разлог

# Request-response |



Слика: Request-response модел

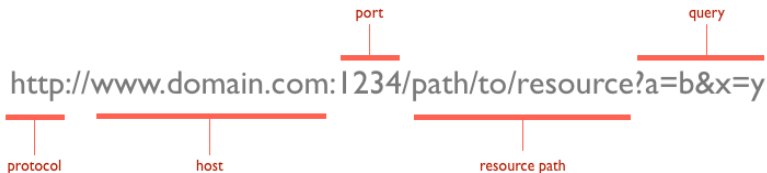
## Request-response II

Слика: Садржај **request** и **response** порука

# Request-response III

- Формирамо HTTP захтев (string)
- Извршавамо DNS упит како би добили IP адресу сервера
  - могуће је и кеширање DNS одговора на клијентској страни
- Успостављамо TCP конекцију са сервером (подразумевани или наведени порт)
- Захтев шаљемо издељен у пакете
- Чекамо одговор
  - клијенти обично постављају timeout
- Затварамо TCP конекцију
  - потенцијално уско грло уколико у кратком временском периоду шаљемо више захтева
  - исправљено у наредним верзијама протокола

# Request-response IV



Слика: URL

# HTTP методе

- **GET**: добављање ресурса из система
- **PUT**: измена постојећег ресурса у целости
- **POST**: додавање новог ресурса у систем
- **PATCH**: измена дела постојећег ресурса
- **DELETE**: брисање ресурса из система

# Status codes

- **1xx:** информациони одговор
  - 100 Continue, 101 Switching Protocols, 103 Early Hints, ...
- **2xx:** успешан одговор
  - 200 OK, 201 Created, 202 Accepted, ...
- **3xx:** редирекција
  - 301 Moved Permanently, 302 Found, ...
- **4xx:** грешка клијента
  - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 405 Method Not Allowed, 415 Unsupported Media Type, 422 Unprocessable Entity, ...
- **5xx:** грешка сервера
  - 500 Internal Server Error, 501 Not Implemented, 502 Bad Gateway, 503 Service Unavailable, 505 HTTP Version Not Supported, ...