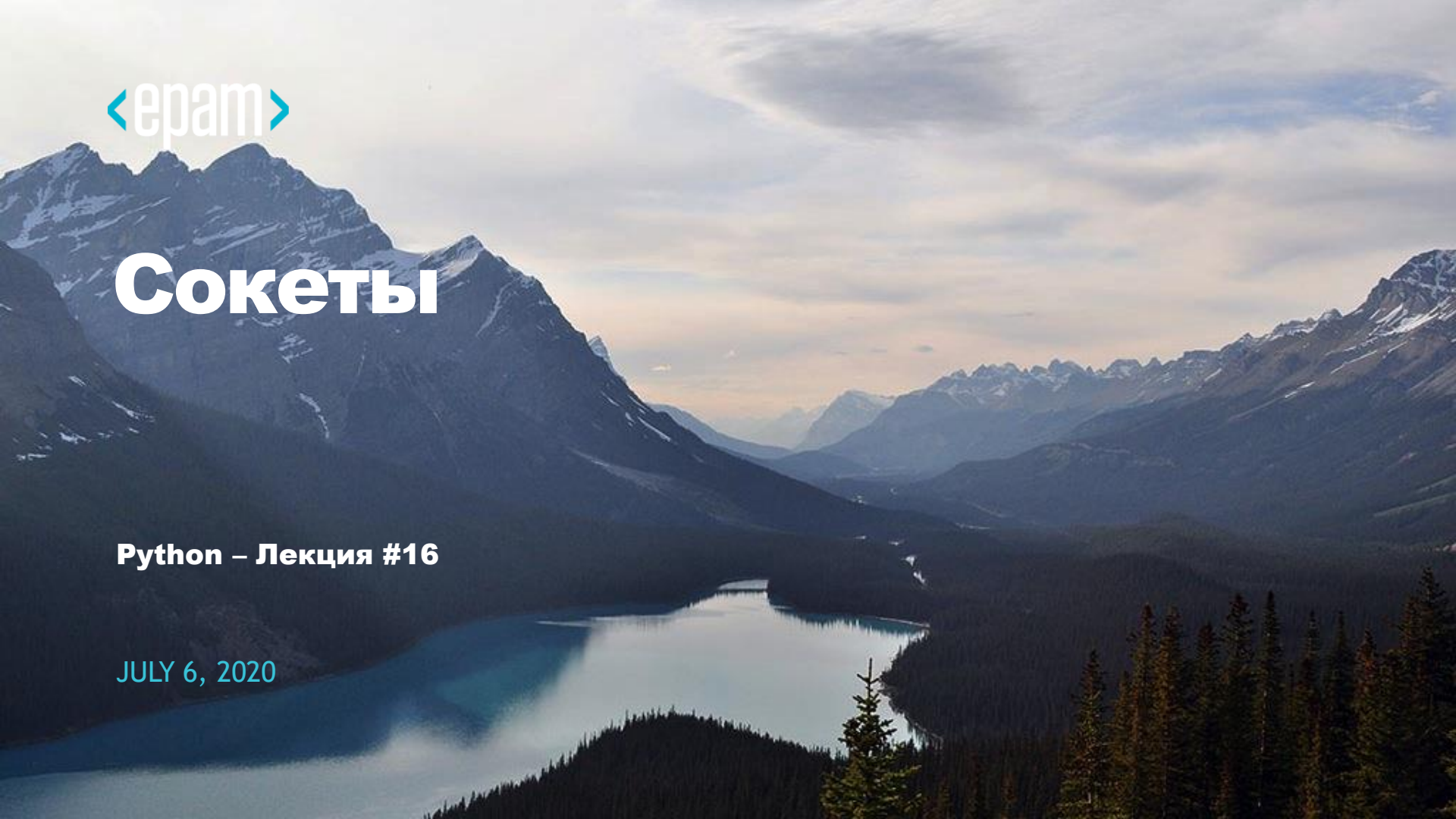


Сокеты

Python – Лекция #16

JULY 6, 2020

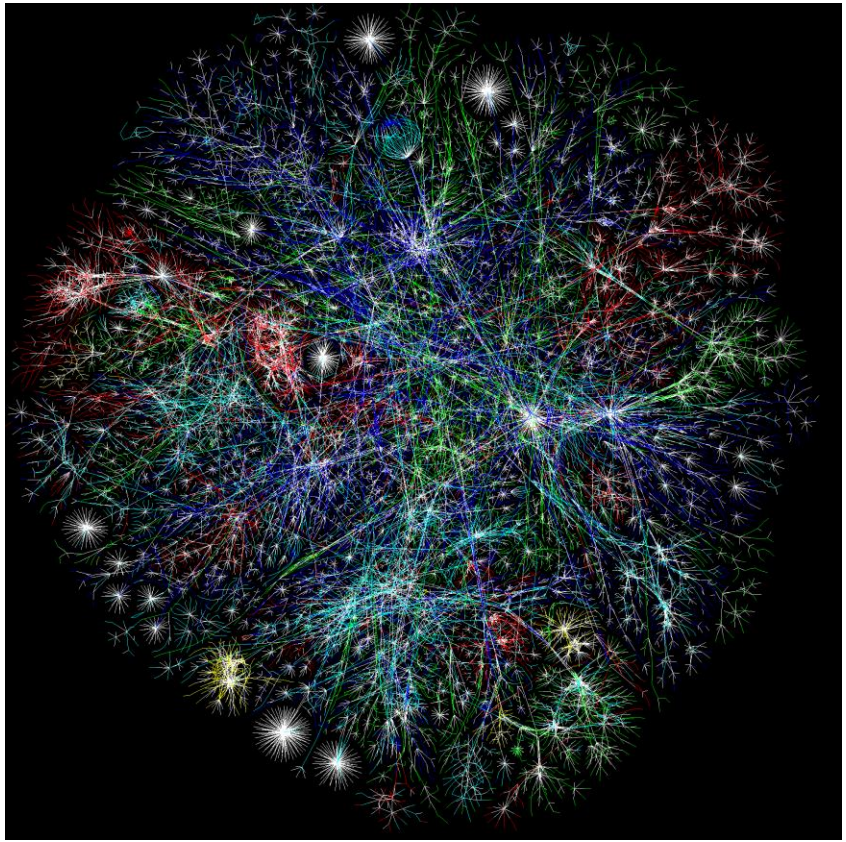
ТЕМЫ

1. Основные понятия, протоколы TCP/UDP
2. Сокеты и использование сокетов
3. Асинхронность и параллелизм

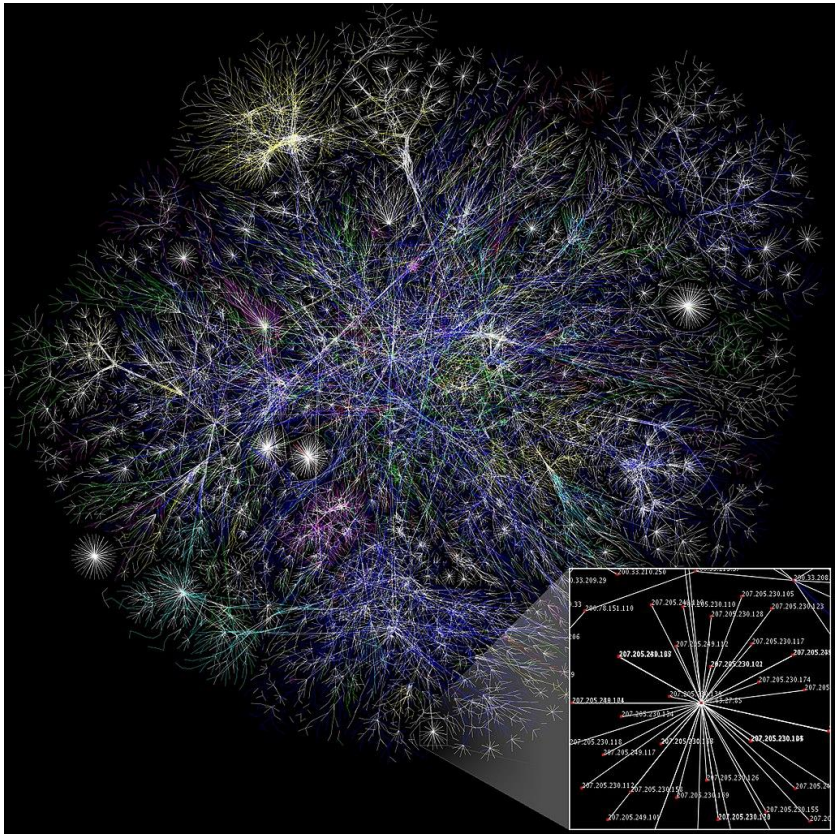
Часть 1

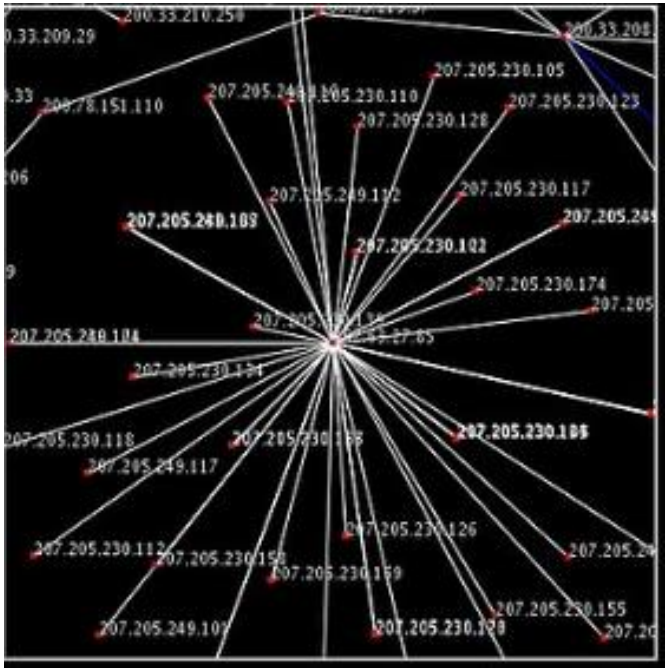
ОСНОВНЫЕ ПОНЯТИЯ

Что это такое?



Частичная карта Интернета от 15 января 2005 года





Протокол IP

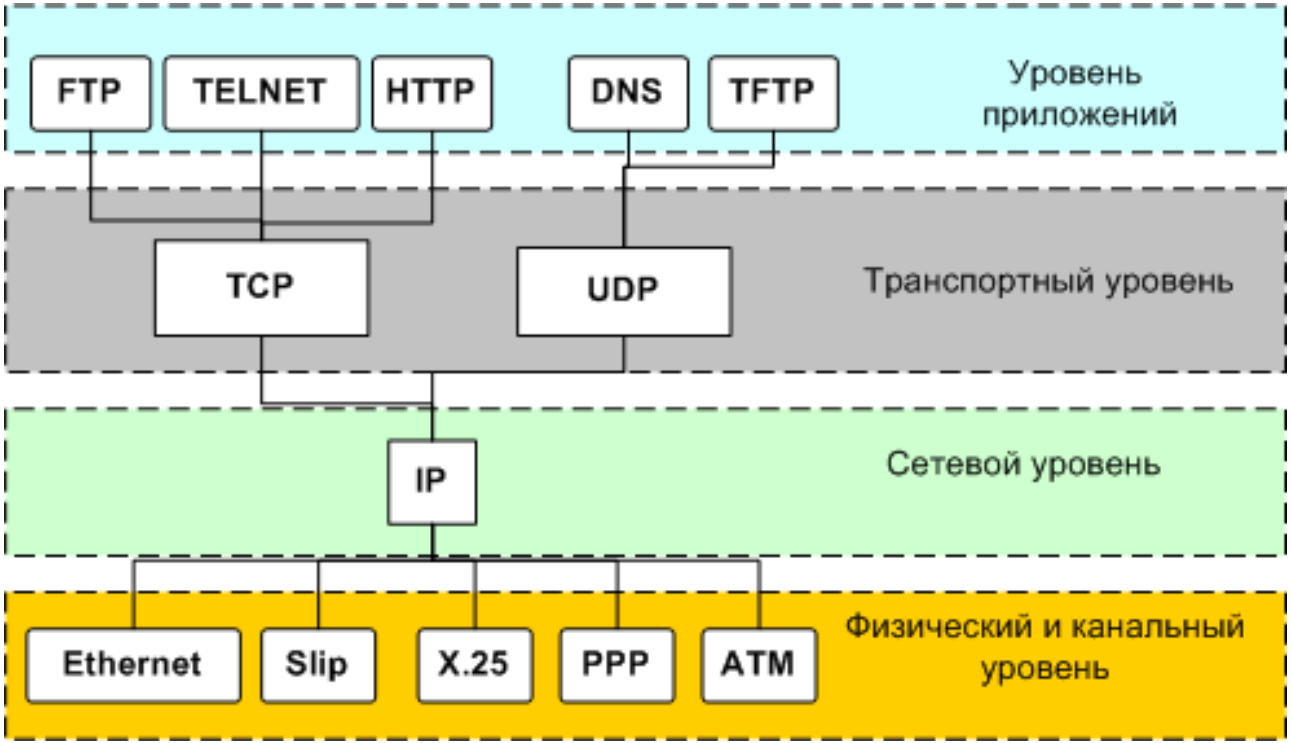
Internet Protocol **(IP,** межсетевой протокол) — протокол сетевого уровня стека TCP/IP, объединяющий отдельные компьютерные сети в глобальную сеть и осуществляющий маршрутизацию (доставку пакета данных через произвольное количество узлов-маршрутизаторов).

Именно IP стал тем протоколом, который объединил отдельные компьютерные сети во всемирную сеть Интернет.

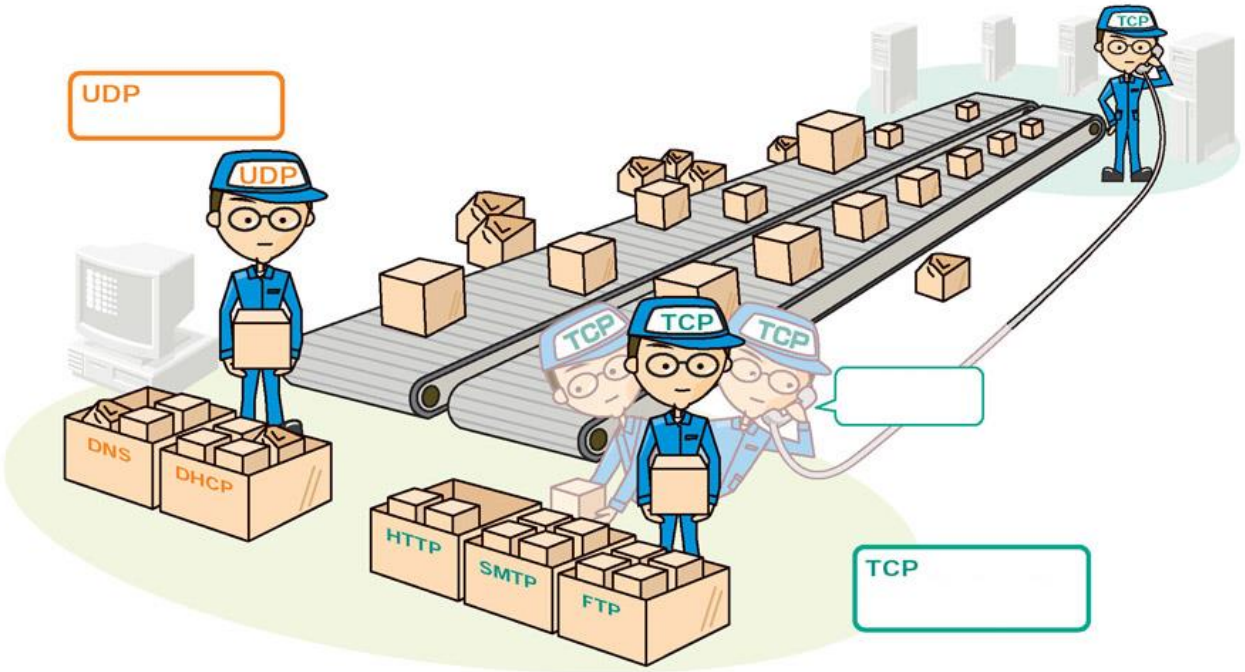
Неотъемлемой частью протокола является *адресация* сети (каждый узел сети имеет уникальный *IP-адрес*).

- IPv4 (*172.16.254.1*)
- IPv6 (*2001:db8:0:1234:0:567:8:1*)
- IP не гарантирует надёжной доставки пакета до адресата, это могут гарантировать следующие уровни

СЕМЕЙСТВО ПРОТОКОЛОВ TCP/IP



Протоколы TCP И UDP



Протоколы TCP И UDP

TCP – Transmission Control Protocol
UDP – User Datagram Protocol

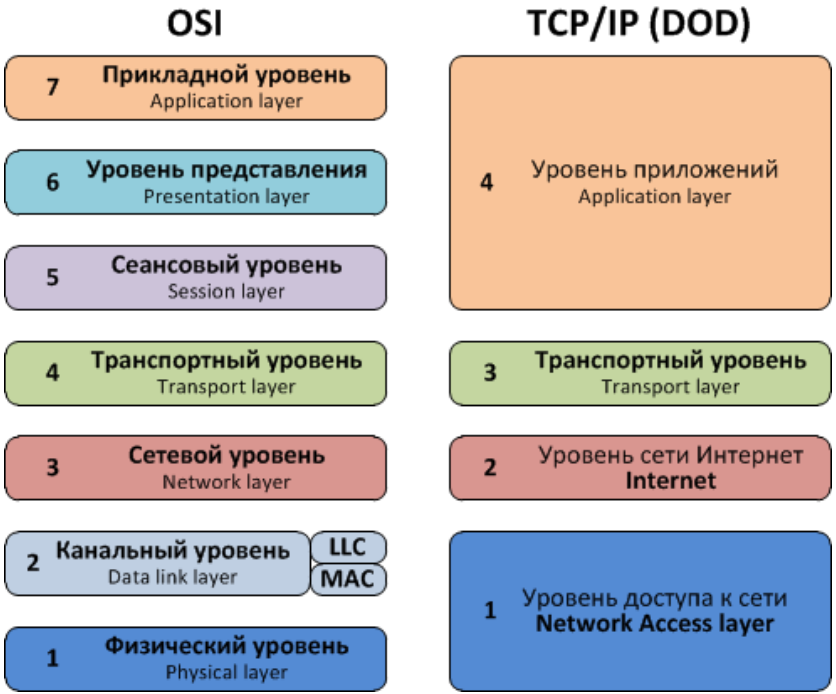
Добавляют в IP-пакеты дополнительные данные, в частности номер порта (1... 65535).

UDP обеспечивает более высокую скорость передачи данных

С UDP возможна передача широковещательных сообщений (broadcast)

	TCP	UDP
Размер заголовка, байт	20-60	8
Форма передачи данных	Поток	Датаграмма
Надежность	Да	Нет
Упорядоченность	Да	Нет
Контроль перегрузок	TCP Congestion Avoidance Algorithm	Нет
Тяжеловесность	Дополнительные 3 пакета для установки соединения	Никаких дополнительных пакетов не нужно
Применение	Там, где нужна надежность и упорядоченность **WWW, e-mail, FTP, SSH**	Там, где высокая нагрузка на сервер и потеря некоторых пакетов некритична **DNS, DHCP, SNMP, голосовой и видео трафик, игры**

Семейство протоколов TCP/IP



- Прикладной уровень - HTTP, FTP, SSH, DNS
- Транспортный уровень - TCP, UDP
- Межсетевой уровень - IP, ICMP
- Канальный уровень – Ethernet, WLAN

Сетевая модель OSI

OSI (англ. *open systems interconnection basic reference model)* — семиуровневая теоретическая модель стека сетевых протоколов.

Уровень (layer)		Тип данных	Функции	Примеры
Host layers	7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, SMTP, RDP, SNMP, DHCP
	6. Представительский (presentation)		Представление и шифрование данных	ASCII, EBCDIC
	5. Сеансовый (session)		Управление сеансом связи	RPC, PAP
	4. Транспортный (transport)	Сегменты (segment)	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SCTP, PORTS
Media layers	3. Сетевой (network)	Пакеты (packet) Дейтаграммы (datagram)	Определение маршрута и логическая адресация	IPv4, IPv6, Ipsec
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация	PPP, IEEE 802.22, Ethernet, DSL, ARP, L2TP, сетевая карта.
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными	USB, кабель ("витая пара", коаксиальный, оптоволоконный), радиоканал

ВОПРОСЫ?

Часть 2

ИСПОЛЬЗОВАНИЕ СОКЕТОВ

Понятие сокета

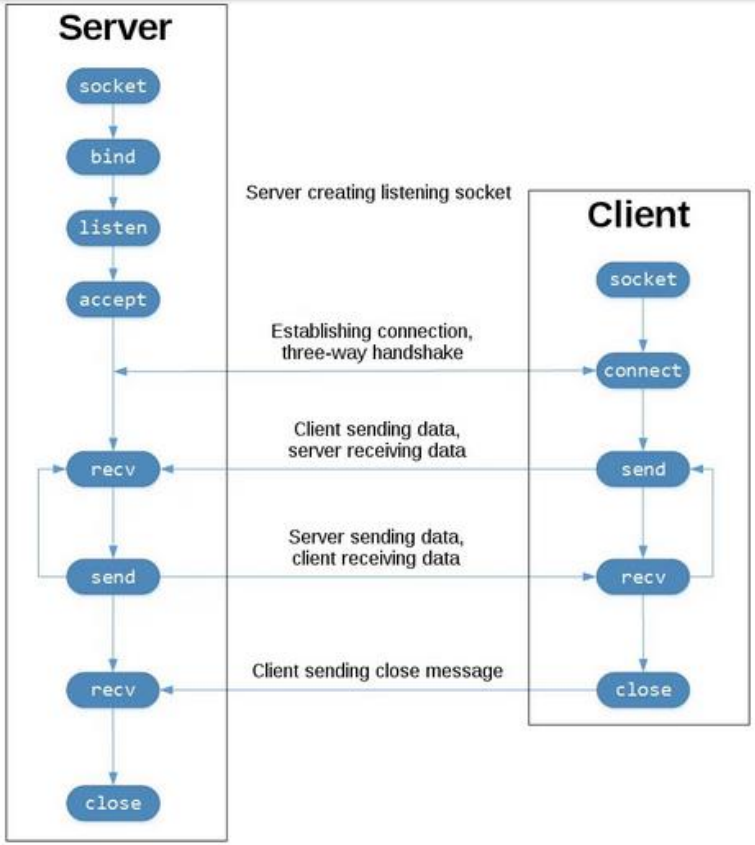
Сокет (англ. *socket* — разъём) — название программного интерфейса для обеспечения двунаправленного обмена данными между процессами.

Процессы при таком обмене могут исполняться как на одном хосте (**доменные сокеты** Unix или IPC-сокеты), так и на различных, связанных между собой сетью (**клиентские** и **серверные сокеты**).

Зачем нужно понимание работы сокетов?

- Для написания сетевых приложений транспортного уровня
- Для понимания работы протоколов прикладного уровня (особенно по части механизмов обработки параллельных соединений)
- Для организации межпроцессного взаимодействия на одной машине

TCP Sockets



Python для работы с сокетами используется модуль socket

```
import socket
```

Создание сокета: `socket()`

Связка с хостом и портом: `bind()`

Начинаем слушать: `listen()`

Принять подключение: `accept()`

Подключение к сокету: `connect()`

Отправка и получение пакетов: `send(), recv()`

Закрываем сокет `close()`

Создание сокета

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Принимаемые аргументы:

- *domain*, указывающий семейство протоколов создаваемого сокета
  AF_INET для сетевого протокола IPv4
  AF_INET6 для IPv6
  AF_UNIX для локальных сокетов в Linux (используя файл)

- *type*
  SOCK_STREAM (TCP)
  SOCK_DGRAM (UDP)
  SOCK_RAW (протокол поверх сетевого уровня).

TCP сервер

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

sock.bind(('localhost', 9999))
sock.listen(5) # максимум подключений

while True:
    client, addr = sock.accept()   # ожидание подключения, блокировка
    print('Connected: ', addr)

    while True:
        data = client.recv(1024)         # чтение, блокировка
        if not data:
            break
        udata = data.decode('utf-8')
        client.send(udata.upper().encode('utf-8'))  # запись, блокировка

    client.close()                    # Закрыть сокет
```

TCP клиент

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('localhost', 9999))

while True:
    data = input()
    if data == 'q':
        break
    sock.send(data.encode('utf-8'))
    resp = sock.recv(1024)
    print(resp.decode('utf-8'))

sock.close()
```

На Linux также можно использовать консольный клиент *netcat*:

```
$ nc localhost 9999
```

Подробнее про метод bind

```
# ожидание подключений с localhost
sock.bind(('127.0.0.1', 9999))
# почти аналогично
sock.bind(('localhost', 9999))

# ожидание подключений из внешней сети
sock.bind(('192.168.1.12', 9999))          # IP-адрес сервера (именно сервера!)
# аналогично, но с автоматическим распознаванием текущего имени хоста
sock.bind((socket.gethostname(), 9999))

# ожидание подключений со всех интерфейсов
sock.bind(('', 9999))
# то же самое, что и выше
sock.bind(('0.0.0.0', 9999))
```

Подробнее про порты

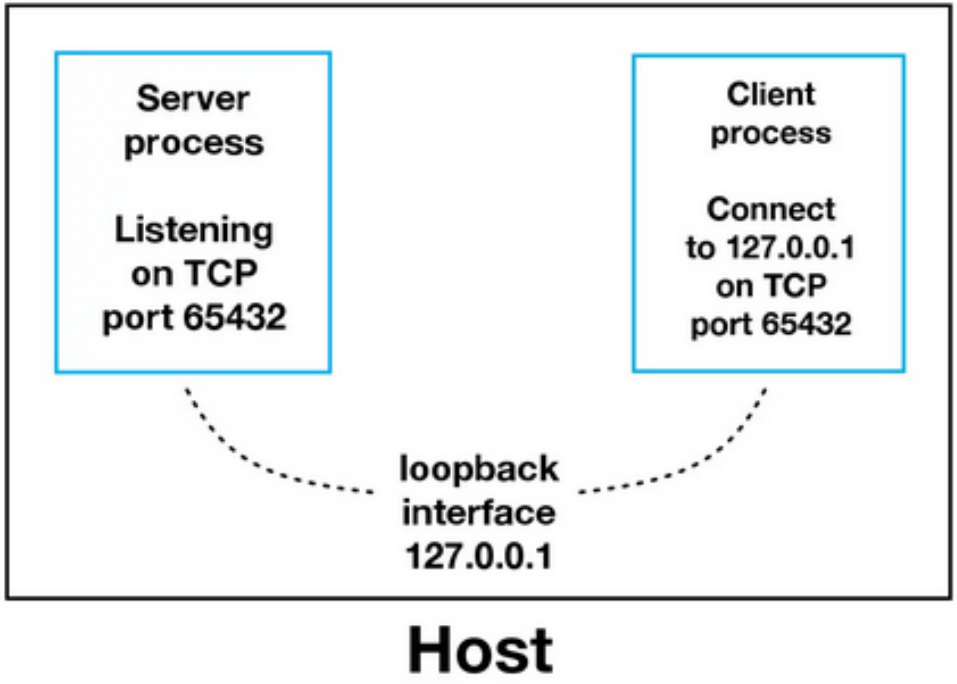
Если на хосте какой-либо процесс постоянно использует один номер порта (например, web-сервер может использовать порт 80 для приёма и передачи данных), говорят, что порт является «открытым».

Если процесс получил номер порта у ОС («открыл порт») и «держит его открытым» для приёма и передачи данных, говорят, что процесс «слушает» (англ. *listen*) порт. Обычно, «слушает» порт процесс программы, реализующей сервер для какого-либо протокола.

Термины «открытый порт» и «закрытый порт» (заблокированный) также используются, когда речь идёт о фильтрации сетевого трафика.

Отдельное понятие - порт процесса-отправителя (клиента). Он может быть постоянным (статическим) или назначаться динамически операционной системой.

Практика TCP



Использование UDP

Отправка:

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

data = 'a' * 4096
sock.sendto(data.encode('ascii'), ('127.0.0.1', 5005))
```

Получение:

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(('127.0.0.1', 5005))

while True:
    data, addr = sock.recvfrom(4096)
    print("Received %d symbols from %s." % (len(data.decode('ascii')), addr))
```

ВОПРОСЫ?
И ПЕРЕРЫВ

Часть 3

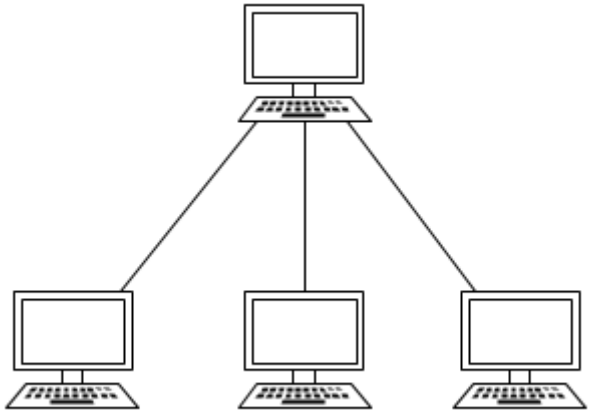
АСИНХРОННОСТЬ И ПАРАЛЛЕЛИЗМ

ОБЗОР

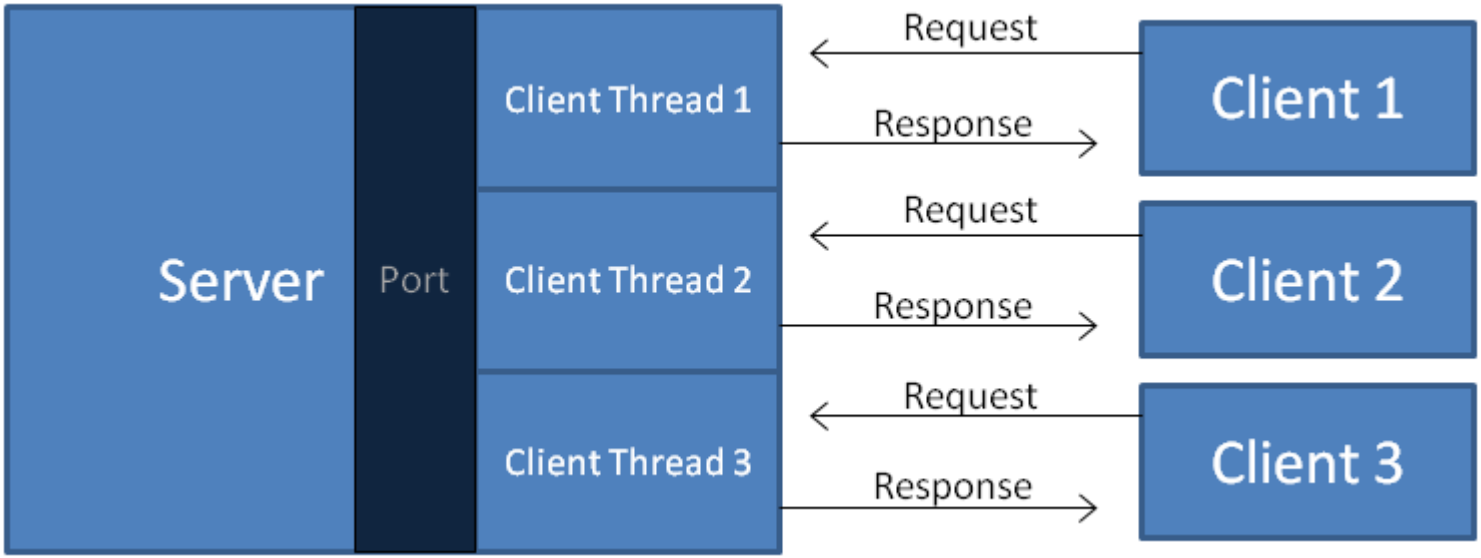
- Многопоточный сервер
- Неблокирующий режим для сокета
- Опрос состояния сокета с помощью *select*

Проблема

Как обрабатывать несколько соединений одновременно?



Многопоточный сервер



Многопоточный сервер

```
import socket, threading

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

sock.bind(('localhost', 9999))
sock.listen(10)                         # длина очереди

def handler(client):
    while True:
        data = client.recv(1024)
        if not data:
            break
        udata = data.decode('utf-8')
        client.send(udata.upper().encode('utf-8'))
    client.close()

while True:
    client, addr = sock.accept(); print('Connected: ', addr)
    threading.Thread(target=handler, args=(client,)).start()
```

Многопоточный сервер с пулом потоков

```
import socket
from multiprocessing.pool import ThreadPool

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('localhost', 9999))
sock.listen(10)                         # длина очереди

def handler(client):
    while True:
        data = client.recv(1024)
        if not data:
            break
        udata = data.decode('utf-8')
        client.send(udata.upper().encode('utf-8'))
    client.close()

pool = ThreadPool(4)
while True:
    client, addr = sock.accept(); print('Connected:', addr)
    pool.apply_async(handler, (client,))
```

Сокеты, три режима

1. *blocking mode (default) – операции блокируются пока не выполнятся или вернётся ошибка*
2. *non-blocking mode – операции возвращают ошибку, если не могут выполниться немедленно*
3. *timeout mode – операции возвращают ошибку, если в течении заданного времени, операция не выполнилась*

Неблокирующий режим

```
sock.setblocking(True)   # sock.settimeout(None)
```

```
sock.setblocking(False)  # sock.settimeout(0.0)
```

Неблокирующий режим

```
import socket, time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

sock.bind(('localhost', 9999))
sock.listen(1)
sock.setblocking(False)

while True:
    try:
        client, addr = sock.accept()      # неблокирующий вызов
        break
    except socket.error:          # отсутствие подключения
        print('Waiting for connection...')
        time.sleep(1)

print('Connected: ', addr)

# То же самое применимо для операций чтения и записи сокета...
```

Опрос состояния сокета, модуль select

Модуль select - Waiting for I/O completion

```
ready_to_read, ready_to_write, in_error = \
            select.select(
               potential_readers, # []
               potential_writers, # []
               potential_errs, # []
               timeout) # float in seconds
```

Опрос состояния сокета

```
import select, socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(('localhost', 9999))
sock.listen(1)

while True:
    print('Waiting for connection...')

    readable, writable, errable, timeout = [sock], [], [], 1

    to_read, to_write, to_err = select.select(
        readable, writable, errable, timeout
    )
    if to_read:
        break

client, addr = sock.accept(); print('Connected: ', addr)

client.close()  # Закрыть сокет
```

Материалы

1. Документация socket
2. Python sockets
3. Сокеты для начинающих

Homework, задача “минимум”

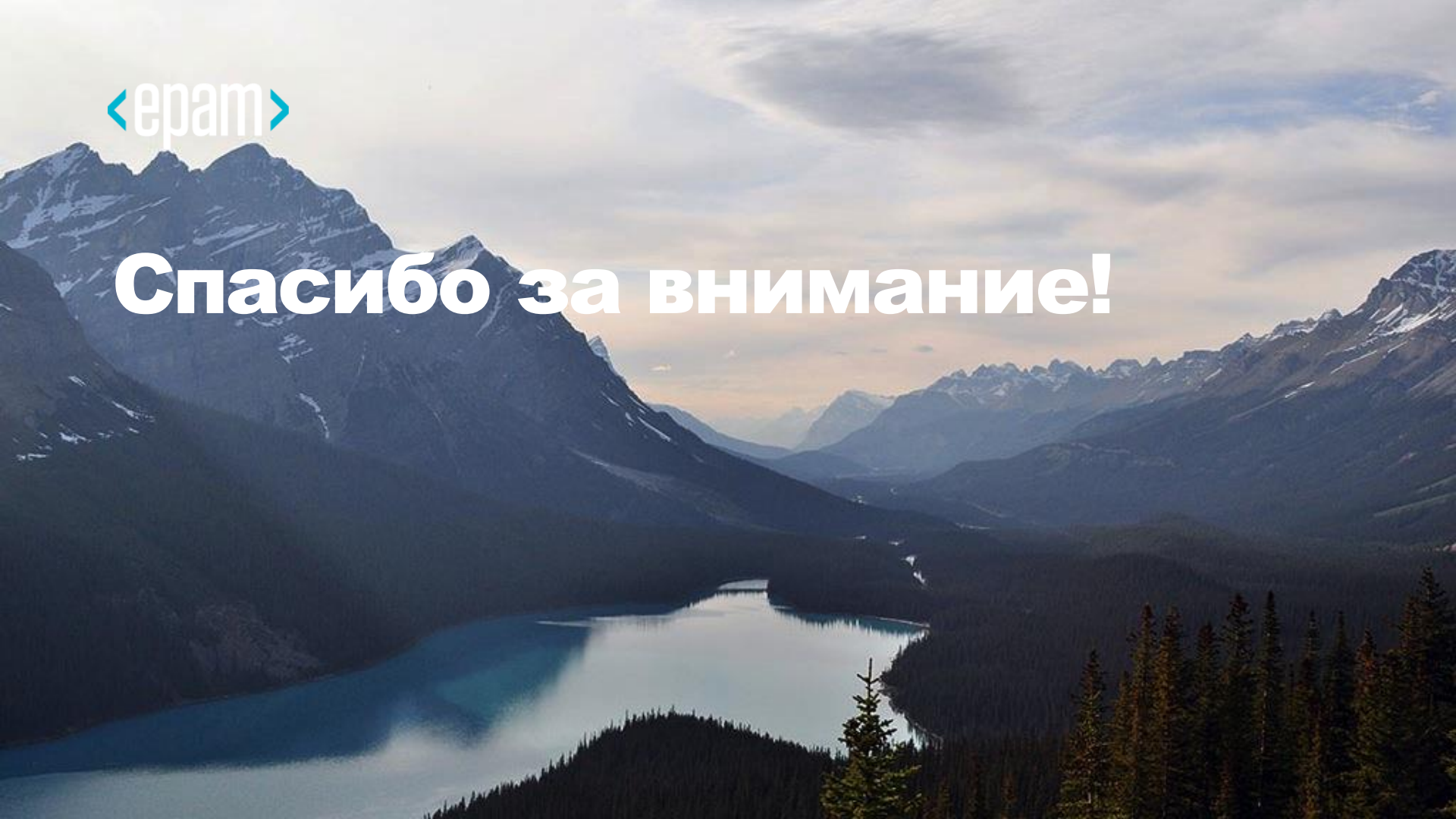
1. Реализовать TCP-server (или UDP-server)
   - Принимает данные от клиента
   - Сохраняет в файл
2. Реализовать TCP-client (или UDP-client):
   - Раз в минуту отправляет данные (эмуляция датчика - random) на сервер
   - Формат данных: текущее время, значение

Homework, задача “хорошо”

1. Реализовать TCP-server (или UDP-server)
   - Принимает данные от клиента
   - Сохраняет в файл
2. Реализовать TCP-client (или UDP-client):
   - Раз в минуту отправляет данные на сервер, данные на выбор:
     - Данные CPU, температуры
     - Количество секунд использования мышки
     - Количество нажатий hot keys (ctrl+c, code inspect, ...)
     - Свой вариант
   - Формат данных: текущее время, название метрики, значение

Homework, задача “максимум”

1. Реализовать TCP-server (или UDP-server)
   - Принимает данные от клиента
   - Сохраняет в файл
2. Реализовать класс для сборки данных,
   - Данные на выбор из пункта (“хорошо”) или свой вариант
   - Реализовать методы для метрики:
     - start_collect() – начинает сборку метрики
     - get_current_state() – получить название метрики, значение
     - cleanup() – сброс значения
     - stop_collect() – остановка сборщика
   - Использование thread для запуска сборщика метрики
3. Реализовать TCP-client (или UDP-client):
   - Отправка данных на сервер раз в минуту, с результатами работы сборщика
   - Формат данных: текущее время, название метрики, значение


epam
Спасибо за внимание!