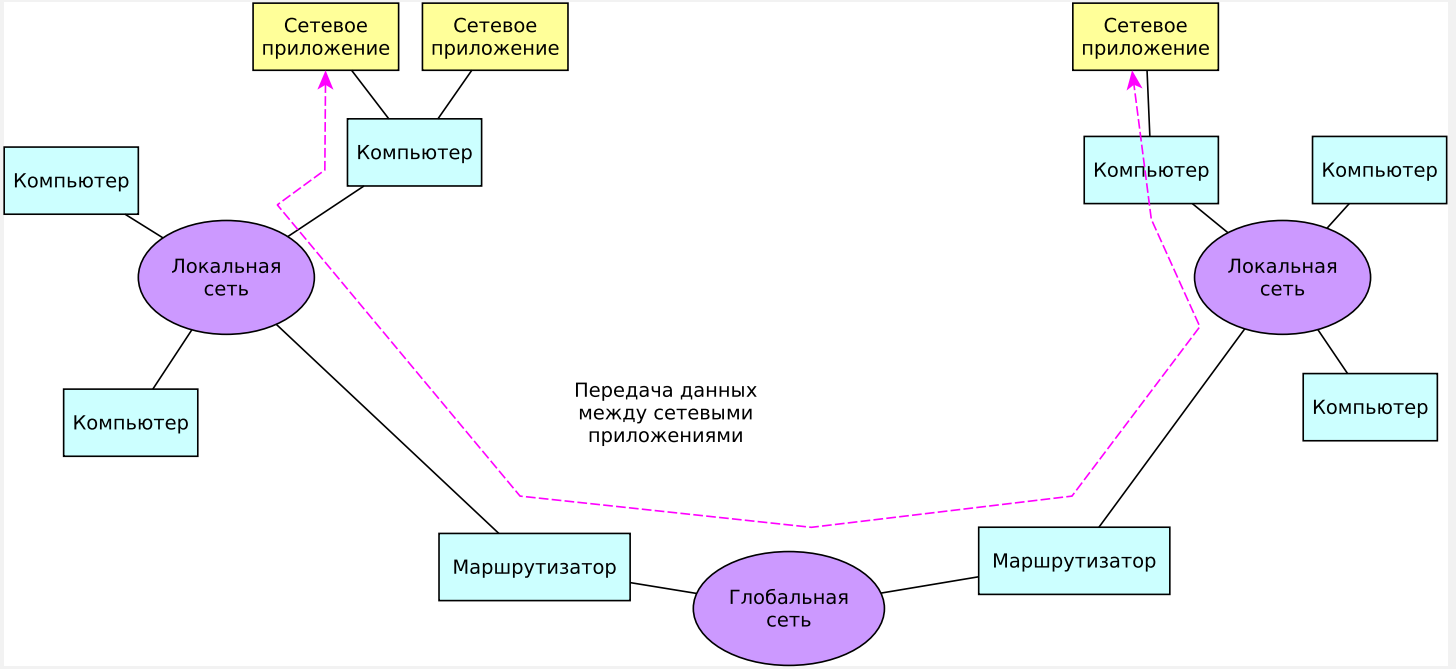
ПО сетевых устройств

Трещановский Павел Александрович, к.т.н.

21.04.20

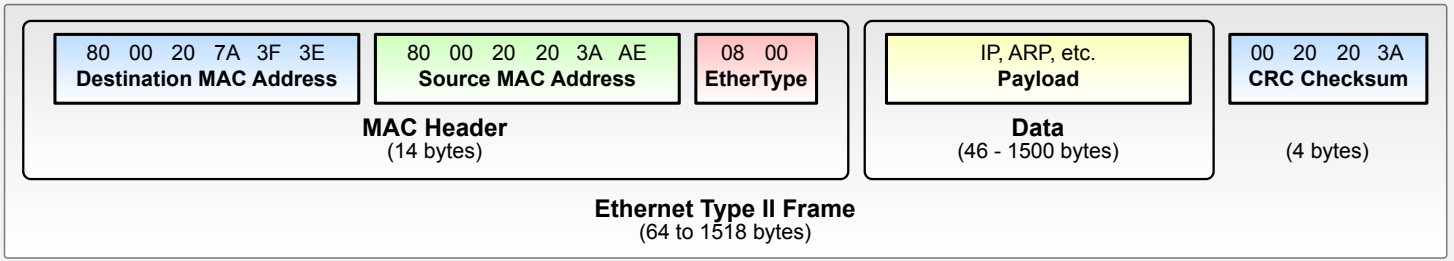
Компьютерные сети



Локальная сеть Ethernet

- Локальная сеть Ethernet состоит из конечных узлов, соединенных коммутаторами.
- Сеть передает Ethernet-кадры.
- Каждый конечный узел имеет уникальный MAC-адрес. Пример: 68:eb:c5:00:01:82.
- Простота - не требуются протоколы маршрутизации для доставки кадров.
- Адресат кадра обнаруживается с помощью широковещательных рассылок.
- Множество сред передачи: медь, оптика, радио.

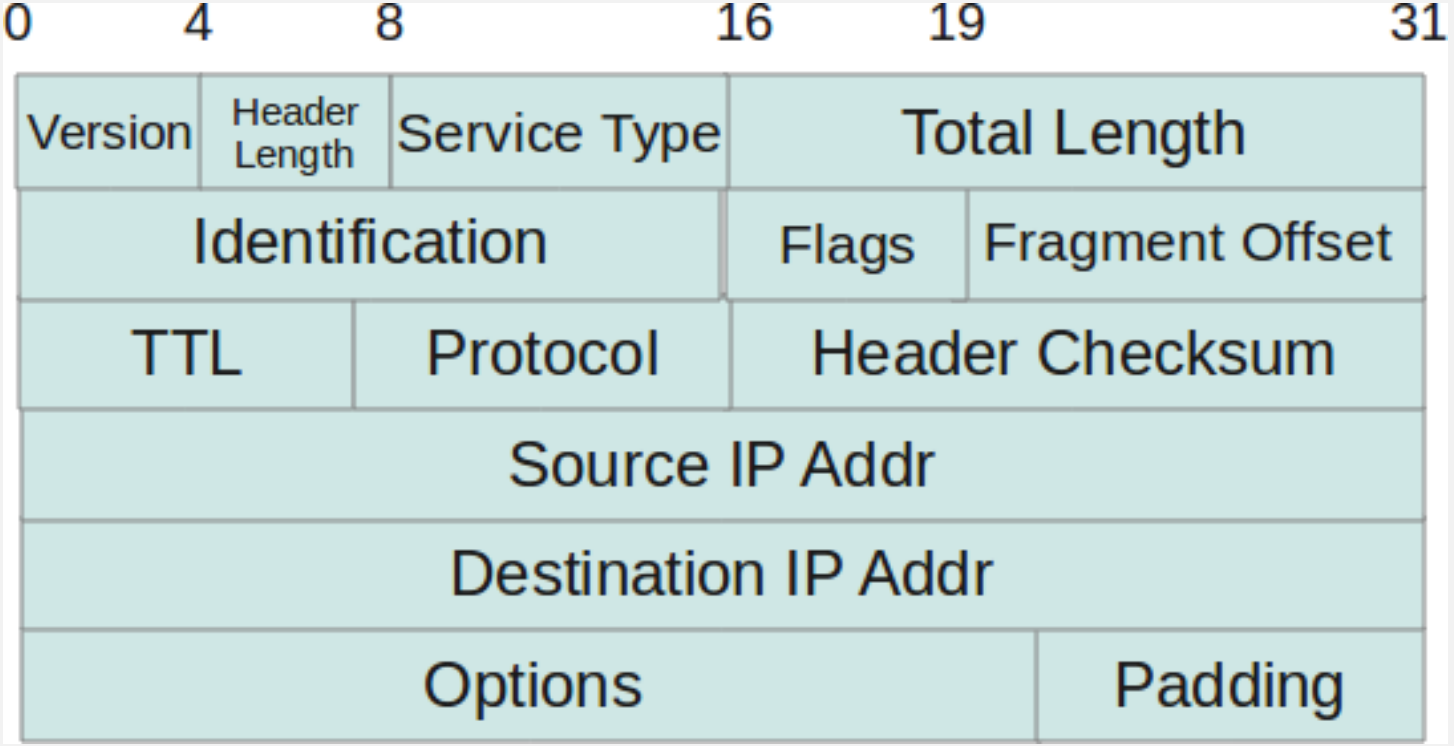
Заголовок Ethernet-кадра



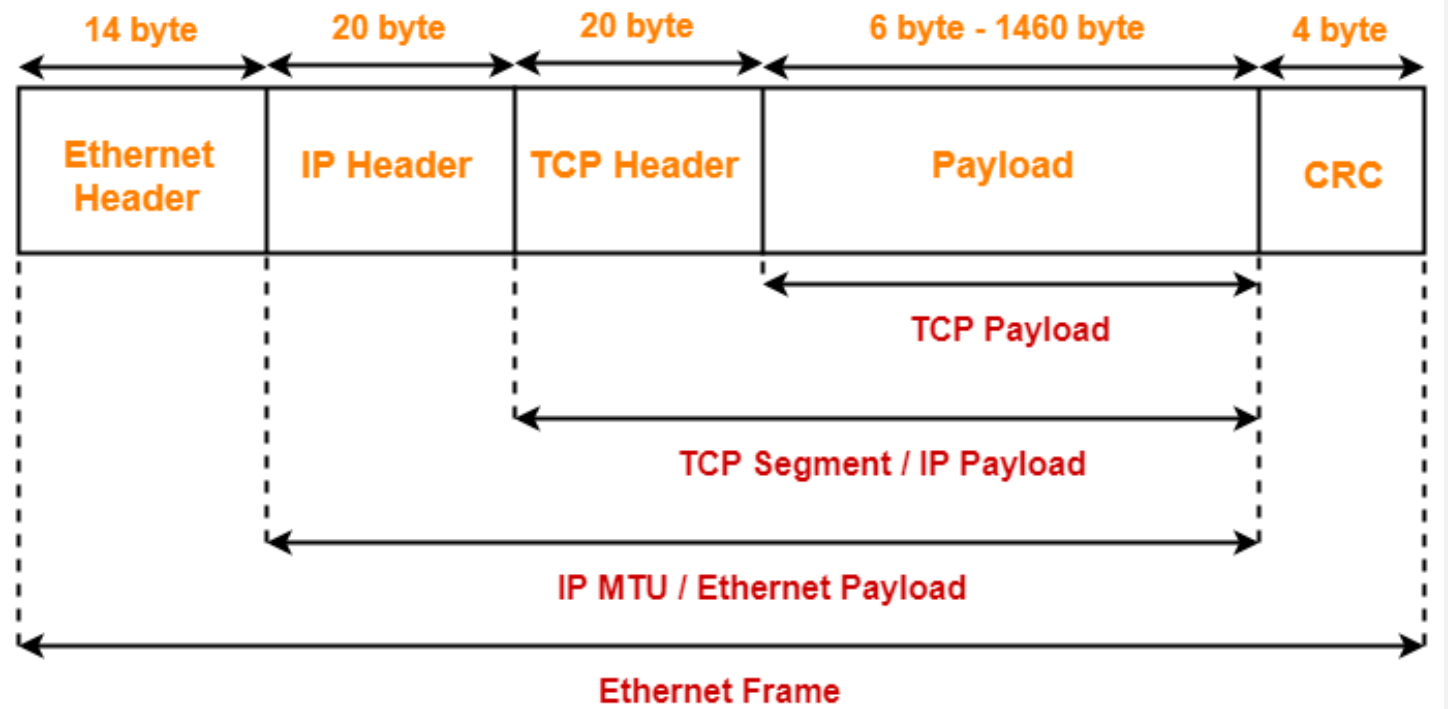
Глобальная сеть IP

- Глобальная сеть IP состоит из локальных сетей, соединенных маршрутизаторами.
- Сеть передает IP-дейтаграммы.
- Каждый конечный узел имеет уникальный IP-адрес. Пример: 64.233.162.113.
- Глобальные широковещательные рассылки запрещены.
- Требуются протоколы маршрутизации для нахождения адресата.
- Для передачи по локальной сети используется Ethernet или аналогичная технология.

Заголовок IP-дейтаграммы



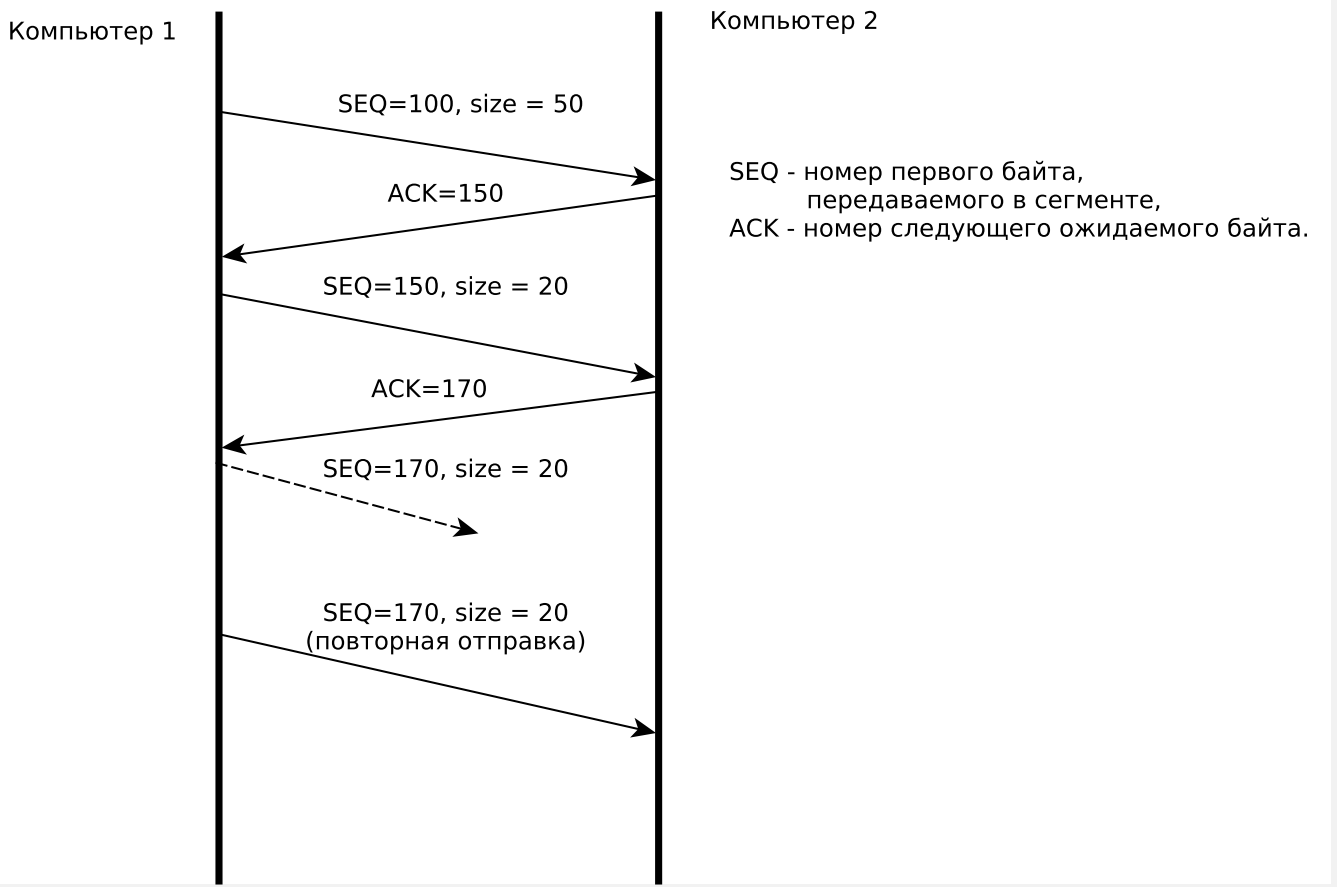
Инкапсуляция



Проблемы сетей Ethernet/IP

- Возможна потеря пакетов из-за помех в среде передачи или переполнения буферов коммутаторов/маршрутизаторов.
- Возможно изменение порядка пакетов при передаче из-за переходных процессов при изменении таблиц маршрутизации.
- Отсутствие гарантий по максимально возможной задержке доставки.

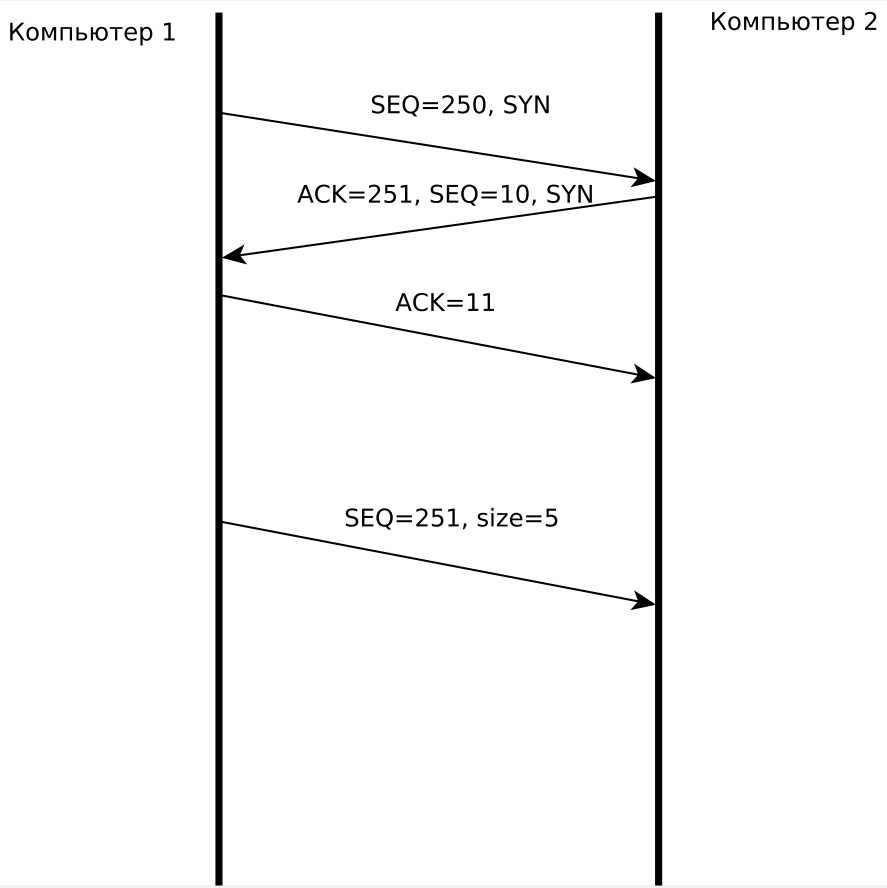
Повторная отправка пакета



Протокол TCP

- Протокол соединяет удаленные приложения.
- Приложение идентифицируется номером порта (числа от 1 до 65535).
- Сообщения протокола называются *сегментами*.
- Каждый передаваемый байт нумеруется. Принимающая сторона должна подтвердить прием отправкой ACK. При отсутствии подтверждения выполняется повторная отправка.
- Передача данных требует установки соединения для синхронизации ожидаемых номеров байтов.

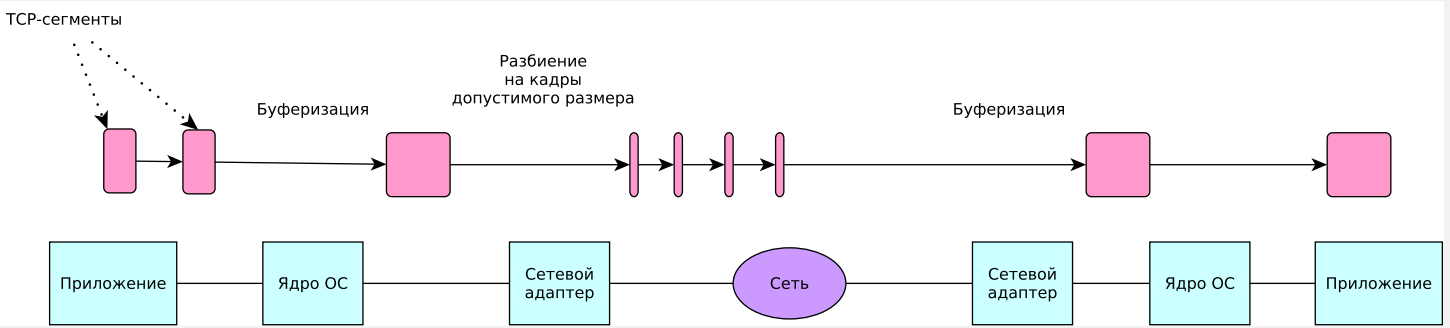
Установка соединения



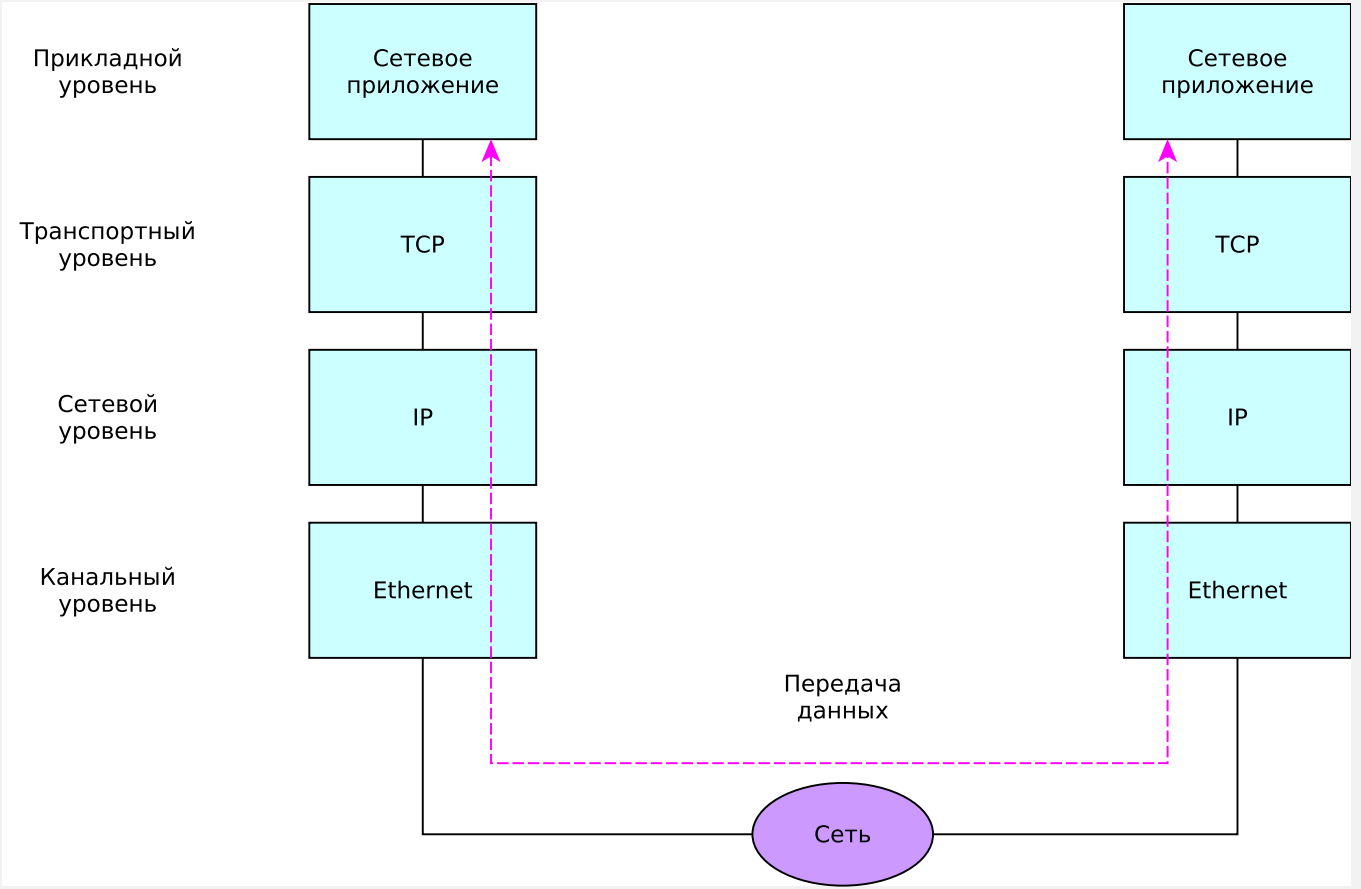
Особенности TCP

- Протокол передает поток данных, разбитый на *сегменты*. Сохранение границ сегментов не гарантируется.
- Типичные приложения: передача файлов (FTP), удаленный терминал (SSH), передача почтовых сообщений (SMTP).
- Повторная отправка может значительно увеличивать время доставки.
- Повторная отправка съедает пропускную способность сети. При наличии множества потоков требуется ограничивать скорость передачи сообщений.

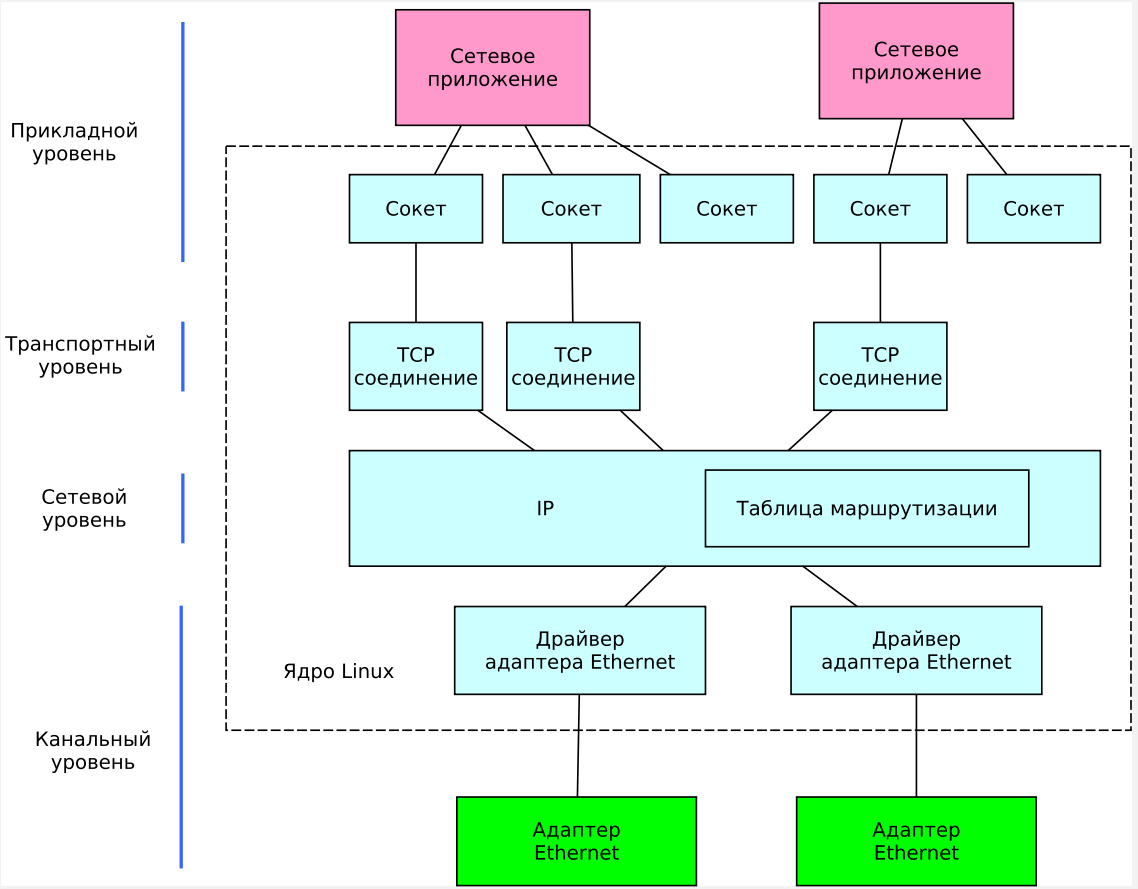
Границы сегмента



4-уровневая модель сетевого взаимодействия



4-уровневая модель, реализация



Сокеты

```
int sock_fd;

/* int socket(int domain, int type, int protocol); */
sock_fd = socket(AF_INET, SOCK_STREAM, 0);
if (sock_fd < 0)
       goto on_error;
```

- `domain` - семейство адресов. `AF_INET` - IPv4, `AF_INET6` - IPv6 и др.
- `type` - тип сокета. Обычно либо потоковый сокет (`SOCK_STREAM`) или дейтаграммный сокет (`SOCK_DGRAM`).
- `protocol` - протокол передачи данных. 0 - протокол по умолчанию. Для потоковых IP-сокетов протокол по умолчанию - TCP.

Установка соединения, API

```
int sock_fd;
struct sockaddr_in addr;
int ret;

sock_fd = socket(AF_INET, SOCK_STREAM, 0);

addr.sin_family = AF_INET;
/* Для номера порта и адреса используется сетевой порядок байтов! */
add.sin_port = htons(21);
inet_aton("127.0.0.1", &addr.sin_addr);

/* int connect(int sockfd, struct sockaddr *serv_addr, socklen_t addrlen); */
ret = connect(sock_fd, (struct sockaddr *)&addr, sizeof(addr));
if (ret < 0)
       goto on_error;
```

Передача данных, API

```
int sock_fd;
char out_buf[] = {2, 0, 2, 0};
char in_buf[100];
int sz;

/* int send(int s, const void *buf, size_t len, int flags); */
sz = send(sock_fd, out_buf, sizeof(out_buf), 0);
if (sz < 0)
       goto on_error;

sz = recv(sock_fd, in_buf, sizeof(in_buf), 0);

printf("Received %d bytes from socket\n", sz);
```

Передача данных через поток ввода-вывода

```
int sock_fd;
FILE *sock_stream;
char response[100];

sock_stream = fdopen(sock_fd, "r+");
if (!sock_stream)
       goto on_error;

ret = fprintf(sock_stream, "request %s %d\n", "abc", 555);
if (ret < 0)
       goto on_error;

fflush(sock_stream);

fgets(response, sizeof(response), sock_stream);
```

Протокол UDP

Создание сокета:

```
sock_fd = socket(AF_INET, SOCK_DGRAM, 0);
```

Особенности.

- Отсутствуют гарантии доставки.
- Сохраняются границы дейтаграмм при передаче через сеть.
- Не вносится дополнительная задержка.
- Адресация аналогична TCP - IP-адрес + порт.
- Общий API с TCP, но смысл функций (`connect`, `send`, `recv` и др.) отличается.

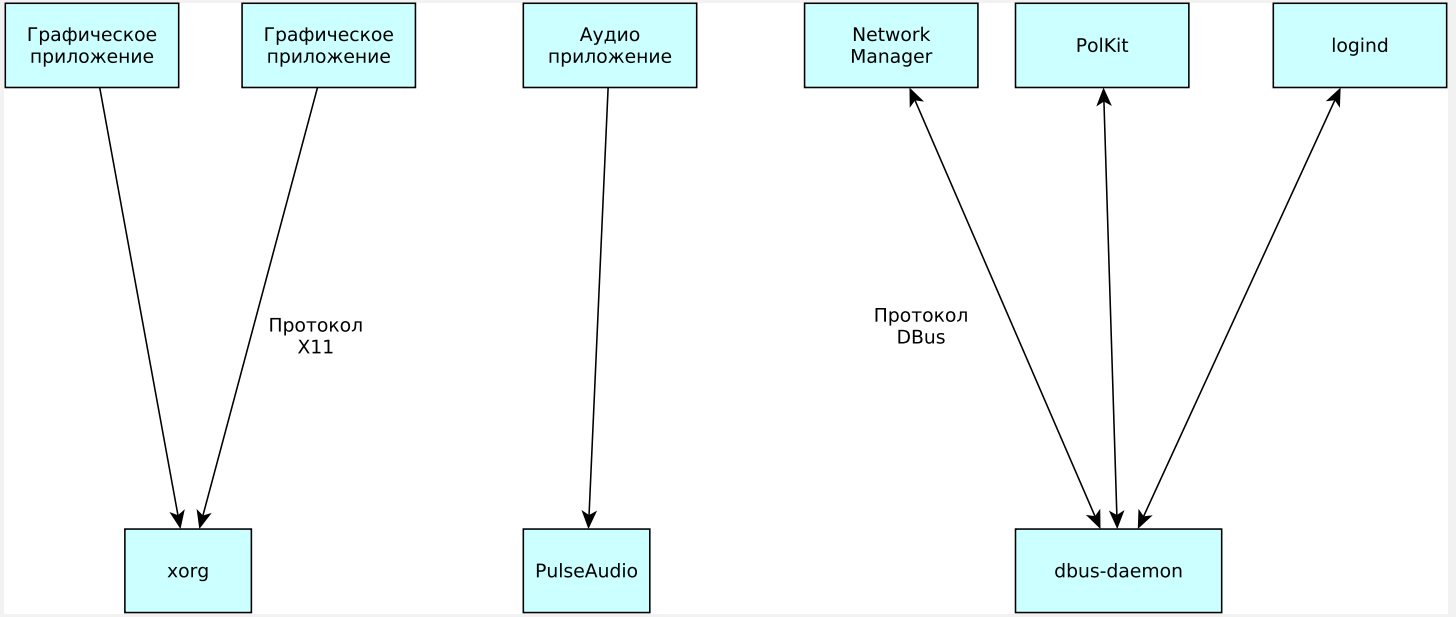
Unix-сокеты

Создание сокета:

```
sock_fd1 = socket(AF_UNIX, SOCK_STREAM, 0);
sock_fd2 = socket(AF_UNIX, SOCK_DGRAM, 0);
```

- Передача только внутри между приложениями одной ОС.
- Адрес приложения - строка. IP-адреса и порты не используются.
- Нет передачи через сеть - нет потерь, повторных отправок и т.д.
- Поддерживается как потоковый, так и дейтаграммный режим.

Использования Unix-сокетов в Linux



Клиентские сокеты в сетевых устройствах

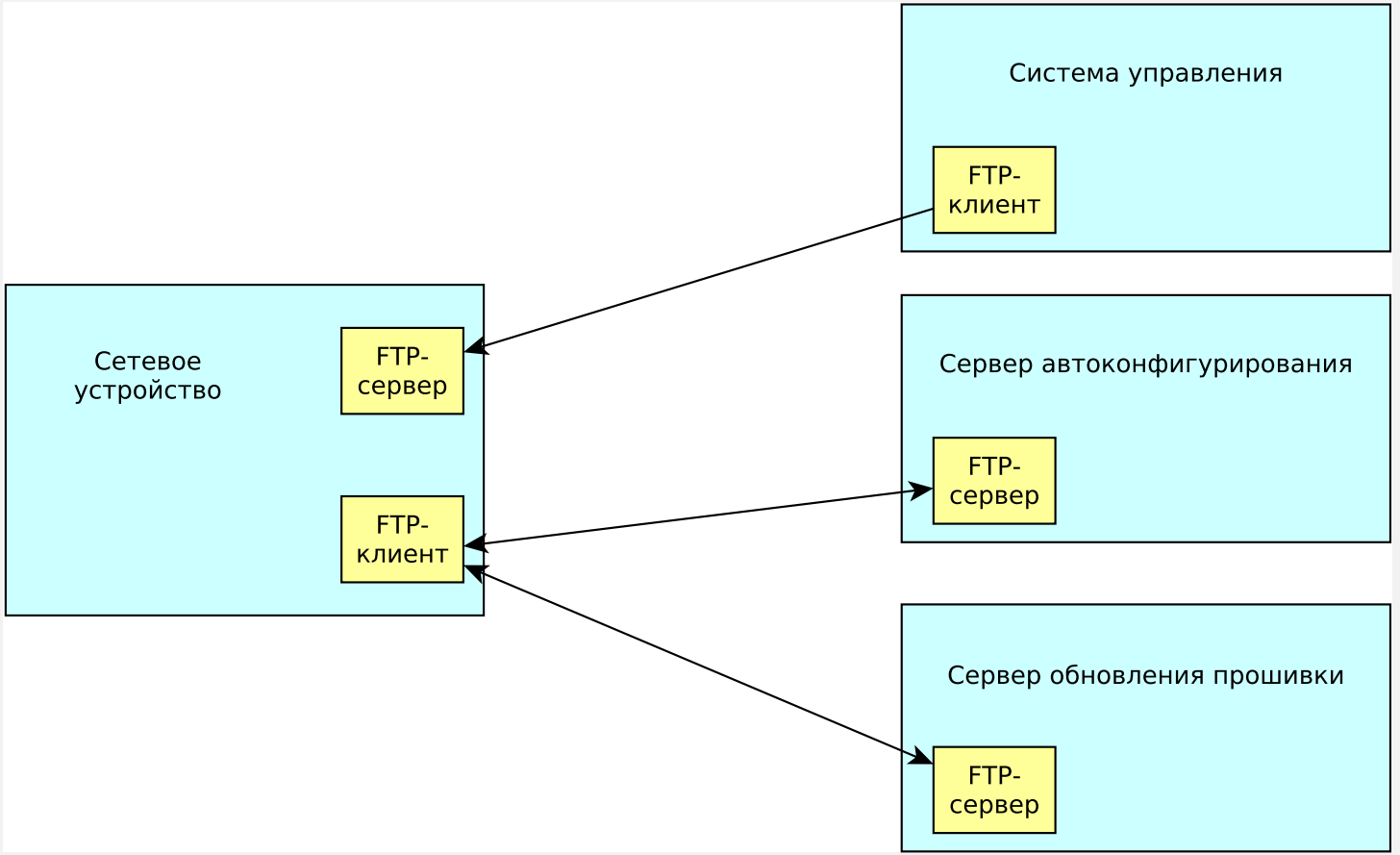
UDP-сокеты:

- DNS - протокол получения IP-адреса из доменного имени (например, www.google.com или www.angtel.ru).
- DHCP - протокол автоматической настройки IP-адреса.
- NTP - протокол автоматической настройки системного времени.
- SNMP - отправка сообщений об изменении состояния системы.

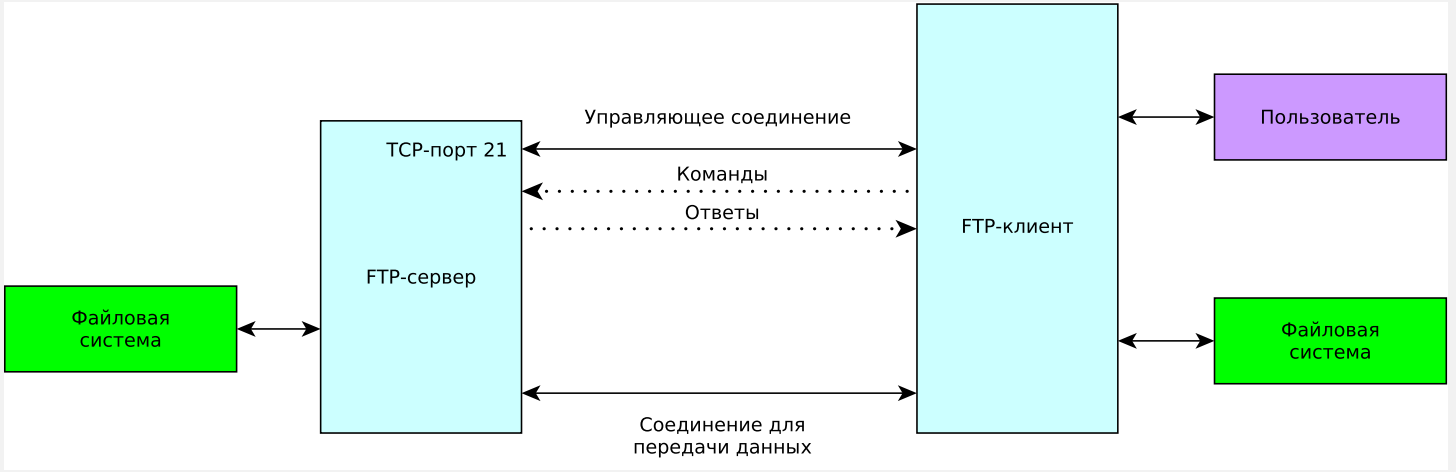
TCP-сокеты:

- OSPF, BGP - протоколы маршрутизации.
- FTP - передача файлов.

Протокол FTP в сетевых устройствах



Протокол FTP, архитектура



Команды и ответы

Синтаксис команды (аргумент нужен не для всех команд):

```
<имя команды> <аргумент>\r\n
```

Команда скачивания файла:

```
RETR file1\r\n
```

Синтаксис ответа:

```
<код> <сообщение>\r\n
```

Пример ответа при успешном выполнении:

```
200 Command okay\r\n
```

Коды 2xx - успешное выполнение, 4xx и 5xx - ошибка, 3xx - необходимо дополнительное действие.

Аутентификация в FTP

Отправка имени пользователя:

```
USER root\r\n
```

Если имя правильное, получаем ответ:

```
331 User name okay, need password\r\n
```

Необходим пароль. Передача пароля:

```
PASS 123\r\n
```

Если пароль правильный, получаем ответ:

```
230 User logged in, proceed\r\n
```

Передача данных

Для передачи данных необходимо дополнительное TCP-соединение. Запрашиваем открытие сокета на стороне сервера:

```
PASV\r\n
```

Ответ содержит адрес и номер порта серверного сокета:

```
227 PASV ok (192,168,0,8,159,107)
```

Каждое число в скобках - 1 байт. Первые 4 - IP-адрес, последние 2 - номер порта.
Устанавливаем соединение для передачи данных. Далее даем команду на скачивание:

```
RETR file1\r\n
```

Ответ:

```
226 Operation successful\r\n
```

После этого данные можно принимать через ранее открытое соединение.

FTP-команда, пример

```
FILE *sock_stream;
char *user_name = "root;
char response[100];
int ftp_code;
char *first_word;

ret = fprintf(sock_stream, "USER %sr\n", user_name);
if (ret < 0)
       goto on_error;
fflush(sock_stream);

fgets(response, sizeof(response), sock_stream);
/* Формат %ms означает чтение строки в динамически выделенный буфер. */
ret = sscanf(response, "%d %ms", &code, &first_word);
if (ret != 2)
       goto on_error;
```

Прочие команды

- PWD - вывод текущего каталога FTP-сервера,
- CWD - смена текущего каталога,
- LIST - вывод содержимого текущего каталога,
- MKD - создание нового каталога,
- QUIT - завершение сессии.

Стандартный клиент

```
$ ftp 192.168.0.8
Connected to 192.168.0.8 (192.168.0.8).
220 Operation successful
Name (192.168.0.8:pavel): root
530 Login with USER+PASS
331 Specify password
Password:
230 Operation successful
ftp> passive
Passive mode on.
ftp> get init.sh
local: init.sh remote: init.sh
227 PASV ok (192,168,0,8,141,187)
150 Opening BINARY connection for init.sh (654 bytes)
226 Operation successful
654 bytes received in 0,000176 secs (3,6e+03 Kbytes/sec)
ftp>
```

C-строка vs строка текста

- C-строка - последовательность символом, ограниченная нулем: `'s', 't', 'r', 'i', 'n', 'g', '\0'`. Используется только внутри приложения.
- Строка текста - последовательность символов, ограниченная символом перехода на новую строку: `'s', 't', 'r', 'i', 'n', 'g', '\n'`. Используется в файлах и пакетах.
- В данных, которые находятся в файле или пакете, никогда нет нулевого окончания. Его должна добавлять программа.
- Некоторые стандартные функции (`read()`, `fread()`) используют массив байтов. Они не используют и не добавляют нулевое окончание.
- Некоторые стандартные функции автоматически добавляют нулевое окончание. Например, `fgets()` читает строку текста в буфер и записывает в конец нулевое окончание.

Порядок байтов

Как представить в памяти многобайтовое число 2864434397 (0xaabbccdd)?

Порядок little-endian:

Адрес	0	1	2	3
Значение	0xdd	0xcc	0xbb	0xaa

Порядок big-endian:

Адрес	0	1	2	3
Значение	0xaa	0xbb	0xcc	0xdd

- Каждый процессор ожидает определённый порядок байтов в памяти для корректного выполнения математических операций.
- x86 использует порядок little-endian.
- В сетевых протоколах используется порядок big-endian (сетевой порядок байтов).

Порядок байтов, API

Перевод из процессорного порядка в сетевой (host to network):

```
int processor_int, network_int;
short processor_short, network_short;

network_int = htonl(processor_int);
network_short = htons(processor_short);
```

Перевод из сетевого порядка в процессорный (network to host):

```
processor_int = htonl(network_int);
processor_short = htons(network_short);
```