# ВВЕДЕНИЕ

**Модель ISO/OSI** (International Standard Organization / Open System Interconnection) была разработана стандартом ISO 7498 в 1980-х годах.

**Официальное название модели ISO/OSI** – сетевая эталонная модель взаимодействия открытых систем международной организации по стандартизации.

Распределенное приложение имеет архитектуру клиент-сервер, если все процессы распределенного приложения можно условно разбить на две группы. Одна группа называется **серверами**, другая – **клиентами.**

Обмен данными осуществляется только между процессами-клиентами и процессами-серверами. Инициатором соединения всегда является процесс-клиент, т.е. процесс-клиент обращается за услугой к процессу-серверу.

По методу обслуживания сервера подразделяются на **интерактивные** (один клиент, очередь) и **параллельные.**

Принципиальная разница заключается в том, что параллельный сервер предназначен для обслуживания нескольких клиентов одновременно (поэтому использует специальные средства ОС, позволяющие распараллеливать обработку нескольких клиентских запросов). Интерактивный сервер обслуживает запросы клиентов поочередно, заставляя клиентов ожидать своей очереди на обслуживание, или просто отказывает клиенту в обслуживании.

**Интерфейс –** набор инструментов (программных/аппаратных) для работы с ПК.

**Протокол –** набор правил и соглашений, описывающих процедуры взаимодействия каждого уровня модели с соседними уровнями.

**Функции,** обеспечивающие взаимодействие открытых систем в модели ISO/OSI распределены по следующим уровням:

1. Физический – определяет свойства среды передачи д-х и способы ее соединения с сетевыми адаптерами.
2. Канальный – подготовить блок данных для сетевого уровня

\*управление доступом к среде передачи данных

(опред.методы совместного исп-ния сетевыми адаптерами среды перед д-х)

\*управление логическим каналом

(опред.понятие канала между 2 сетевыми адаптерами + способы обнаруж.и исправления ошибок передачи данных)

1. Сетевой – опред методы адресации и маршрутизации ПК в сети.
2. Транспортный – подготовка и доставка пакетов данных между конечными точками без ошибок и в правильной последовательности.
3. Сеансовый – опред способы установка и разрыва соединений 2 приложений, работающих в сети.
4. Представительский – опред формат д-х, используемых прилож
5. Прикладной – опред способы взаимодействия пользователей с системой (определить интерфейс)

**Задача каждого уровня** – предоставление услуг вышестоящему уровню, таким образом, чтобы детали реализации этих услуг были скрыты.

Данные проходят от прикладного уровня одной системы до прикладного уровня другой системы через все нижние уровни системы. При чем по мере своего движения от отправителя к получателю данные подвергаются необходимому преобразованию: при движении от прикладного к физическому данные преобразовываются в формат, позволяющий передать данные по физическому каналу; при движении от физического уровня до прикладного происходит обратное преобразование данных.

Передача данных в модели ISO/OSI осуществляется на физическом уровне.



# СТЕК ПРОТОКОЛОВ TCP/IP

**TCP** (Transmission Control Protocol) обеспечивает надежную доставку данных в сети.

**IP** (Internet Protocol) организует маршрутизацию сетевых передач от отправителя к получателю и отвечает за адресацию сетей и компьютеров.

В настоящее время существует **6 групп стандартизации** координирующих TCP/IP:

\*ISOC (Internet Society)

\*IAB (Internet Architecture Board)

\*IETF (Internet Engineering Task Force)

\*IRTF (Internet Research Task Force)

\*ISTF (Internet Society Task Force)

\*ICANN (Internet Corporation for Assigned Names and Numbers)

**IETF** – наиболее проблемная группа по проектированию в Internet, поскольку именно она занимается поддержкой документов именуемых **RFC** (Request for comments), в которых описаны все правила и форматы всех протоколов и служб TCP/IP в сети Internet.

**Сетевой протокол –** набор правил и соглашений взаимодействия двух соседних уровней модели ISO/OSI.

Структура TCP/IP отличается от эталонной модели ISO/OSI.



**Протоколы Уровня доступа к сети**

На уровне доступа к сети задействованы протоколы для создания локальных сетей (**LAN**, Local-Area Networks) и для соединения с глобальными сетями (**WAN**, Wide-Area Networks).

Работа протоколов этого уровня регулируется семейством стандартов IEEE 802.

1. Кто может назначать узлам IP адреса и прочие параметры

Администратор

1. С помощью чего клиент сможет получить от сервера все необходимые параметры

Запросы, DHCP

1. Как узел, не имеющий IP адреса, может общаться с сервером? От какого IP адреса узел, пока не имеющий Ip адреса сможет отправлять пакеты?

Широковещательный запрос.

От кого – пишем все 0.

1. Откуда клиент будет знать IP адрес сервера перед тем, как начинать взаимодействие?

Со стороны клиента рассылаем широковещательный запрос, сервер получает и отсылает клиенту свой IP

1. Как клиет сможет получить от сервера ответ, если он еще не имеет IP адреса?

Сервер отправляет широковещательный запрос всем клиентам.

Клиент взаимодействует с первым сервером, приславшим запрос.

Предполагается, что тот кто прислал первым, находится ближе, а значит лучше обмен.

1. Откуда DHCP сервер узнает, какие адреса и дополнительные конфигурационные параметры предлагать клиенту

Клиент при отправке запроса на получение IP адреса может попросить и другие параметры (маску, время аренды IP-адреса и т.д.).

# DHCP

Служба DHCP занимает порт UDP 67.

Отметим, что DHCP клиент пользуется портом UDP 68.

Служба находится на прикладном уровне, а реализация протоколов на сетевом уровне. (ОТЛИЧИЕ)

## Взаимодействие клиента и сервера

* **DHCPDISCOVER** – отправляется от клиента к серверу. в качестве отправителя все 0000. в качестве получателя 255.255.255.255. клиент, не зная что за сеть, может выдвинуть требования серверу для установки IP адреса с параметрами (маска, время и т.д.). этот пакет рассылается по сети. Содержит в себе ключ, который генерируется. Сервера обязаны отреагировать ответом DCHPOFFER.
* **DHCPOFFER** – ответ сервера; посылается клиенту с ip адресом; отсылается ответ «я сервер, со мной можно общаться по такому-то адресу. я могу выдать вам такой-то ip». этот пакет также отсылается всем в сети. Вставляет также ключ, который был сформирован клиентом. Клиент смотрит его ключ или нет, и в зависимости от этого или принимает, или не принимает этот IP.
* **DHCPREQUEST** – возможность дополнительно установить какие-то параметры. Отсылается от клиента серверам широковещательно. IP адрес сервера уже известен. Указываем параметры какие хотим получить.
* **DHCPPACK** – получает DHCPREQUEST и смотрит был ли какой-то клиент, который быстрее запросил что-то, чем мы. Если такого не было, то записываем клиента (нас) в таблицу с выданными параметрами. И выдаем это подтверждение клиенту. Клиент выставляет параметры и пользуется выделенным IP адресом.
* **DHCPNACK** – если кто-то успел забрать IP-адрес быстрее нас, то он отправляет сообщение клиенту, что данный IP адрес уже занят. Тогда клиент заново посылает широковещательный запрос на получение IP адреса и делает это до тех пор, пока не получит его.
* **DHCPDECLINE** – клиент отправляет запрос серверу, если в сети уже есть хост с таким же IP адресом и параметрами. Т.е. если клиент получил и пришел хост с уже установленными параметрами. Клиент в таком случае просит новый IPадрес и параметры.

Стационарная часть заголовка DHCP составляет целых 59(!) четырехбайтовых слов или 236 байт.

## Структура сообщений

### Поле op

показывает тип операции:

\* значение 01 – данный пакет является запросом от клиента серверу (BOOTREQUEST)

\* значение 02 – данный пакет является ответом сервера (BOOTREPLY)

Очень важно подчеркнуть, что данное поле принимает только ДВА значения и не предназначено для идентификации типа пакета (DHCPDISCOVER, DHCPOFFER, DHCPREQUEST и т.д.)

### Поле htype

Показывает какая технология канального уровня используется сервером и клиентом и полность аналогично соответствующему полю заголовка ARP.

### Поле hlen

Показывает длину аппаратного адреса, используемого DHCP клиентом. Так как формально говоря длина аппаратного адреса в разных канальных технологиях может быть отличной, то DHCP серверу придется считывать поле MAC адреса станции клиента переменной длины, и один из самых простых способов обеспечить нормальное чтение такого поля – просто заранее оговорить его длину.

### Поле hops

Используется при маршрутизации DHCP сообщений. Это поле клиент всегда обязан установить в 0, в маршрутизаторы, перенаправляющие DHCP сообщения могут это поле использовать.

### Поле xid

Служит для идентификации транзакции между клиентом и сервером. Чтобы клиенты и сервера могли отличать пакеты посланные им от чужих пакетов используется данное поле «идентификатор транзакции».

**Работа** данного поля:

Клиент в своем первом пакете серверу устанавливает случайное число в это поле, сервер во всех своих пакетах данному клиенту в рамках данной процедуры взаимодействия будет цитировать значение этого поля, а клиент соответственно в рамках данного обмена пакетами с сервером все свои пакеты будет снабжать тем же значением поля xid.

Таким образом, участники DHCP транзакции могут четко отличать пакеты, поступившие в рамках данной транзакции от пакетов, принадлежащих другим транзакциям. Без подобного поля сервера и клиенты легко могли бы запутаться при осуществлении нескольких взаимодействий одновременно.

## Опции

Если в заголовке DHCP будут присутствовать опции (по стандарту есть опции, которые обязательно должны присутствовать в заголовке), то первые четыре байта поля опций должны принимать фиксированное значение, так называемое **«магическое число»**, равное 63 82 53 63 в шестнадцатеричной записи.

После этого поля следуют непосредственно опции.

Опции DHCP начинаются с однобайтового поля code, показывающего, по сути, тип опции.

Существует **2 типа опций:**

# РАЗРАБОТКА ПАРАЛЛЕЛЬНОГО СЕРВЕРА

**Ожидающий таймер** – объект синхронизации, который переходит в **сигнальное состояние** при наступлении заданного момента времени.

Если ожидающий таймер ждет момента перехода в сигнальное состояние, то говорят, что он находится в **активном состоянии.**

Другое состояние ожидающего таймер **пассивное** – из этого состояния он не может перейти в сигнальное состояние.

По способу перехода из сигнального состояния в несигнальное, ожидающие таймеры разделяются на **таймеры с ручным способом** и таймеры с **автоматическим сбросом**, иначе называемые **таймерами синхронизации.**

По способу перехода из несигнального состояния в сигнальное, ожидающие таймеры бывают **периодические** и **непериодические.**

Периодические таймеры работают по циклу: активное состояние – сигнальное состояние – активное состояние.

Непериодические таймеры могут только один раз перейти из активного состояния в сигнальное.

|  |  |
| --- | --- |
| **Название функции** | **Назначение** |
| CancelWaitableTimer | Отменить ожидающий таймер |
| CreateWaitableTimer | Создать ожидающий таймер |
| OpenWaitableTimer | Открыть существующий ожидающий таймер |
| SetWaitableTimer | Установить ожидающий таймер |
| WaitForSingleObject | Ждать сигнального состояния ожидающего таймера |

Описание функций в лекциях

Блокирующие функции выполняющие несколько элементарных операций, которые объединяются в одну неделимую операцию, называемую **атомарной операцией.**

**Зачем нужны атомарные операции, если есть критические секции?**

Применяется

|  |  |
| --- | --- |
| Название функции | Назначение |
| InterlockedCompareExchange | Сравнить и заменить значение |
| InterlockedDecrement | Уменьшить значение на единицу |
| InterlockedExchange | Заменить значение |
| InterlockedExchangeAdd | Изменить значение |
| InterlockedIncrement | Увеличить значение на единицу |

Функции в методичке(описание)

Переключение в **режим без блокировки** позволяет избежать приостановки программы.

Для того, чтобы отличить ошибку от отсутствия запроса, используется уже рассмотренная функция WSAGetLastError, которая в последнем случае возвращает значение WSAEWOULDBLOCK.

**Основные функции необходимые для работы с приоритетами**

|  |  |
| --- | --- |
| Наименование функции | Назначение |
| GetPriorityClass | Получить приоритет процесса |
| GetThreadPriority | Получить приоритет потока |
| getProcessPriorityBoost | Определить состояние режима процесса |
| GetThreadPriorityBoost | Определить состояние режима потока |
| SetProirityClass | Изменить приоритет процесса |
| SetThreadPriority | Изменить приоритет потока |
| SetProcessPriorityBoost | Установить или отменить динамический режим для всех потоков процесса |
| SetThreadPriorityBoost | Установить или отменить динамический режим потока |
| GetCurrentProcess | Получить дескриптор процесса |

**Таблица приоритетов процесса и потока**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Приоритет потока** | **Приоритет процесса** | | | | | |
| **Реал.врем** | **высокий** | **Выше норм** | **норм** | **Ниже норм** | **фонов** |
| **Реал.времени** | 31 | 15 | 15 | 15 | 15 | 15 |
| **Высший** | 26 | 15 | 12 | 10 | 8 | 6 |
| **Выше норм** | 25 | 14 | 10 | 9 | 7 | 5 |
| **Норм** | 24 | 13 | 10 | 8 | 6 | 4 |
| **Ниже норм** | 23 | 12 | 9 | 7 | 5 | 3 |
| **Низший** | 22 | 11 | 8 | 6 | 4 | 2 |
| **фоновый** | 16 | 1 | 1 | 1 | 1 | 1 |

Структура параллельного сервера с диспетчером запросов

**Функции для работы с событиями**

CreateEvent – создать событие + установить параметры

OpenEvent – открыть событие (создание дескриптора)

PulseEvent – освободить ожидающие потоки

ResetEvent – перевести событие в несигнальное состояние

SetEvent – перевод события в сигнальное состояние

Структура параллельного сервера с динамической библиотекой

BOOL WINAPI DllMain – обозначает точку входа в прогр.модуль диманически подключаемой библоиотеки, функции получает управление от ОС в момент загрузки.

**Функции, необходимые для работы с динамически подключаемыми библиотеками**

FreeLibrary – отключить dll-библиотеку от процессора

GetProcAddress – импортировать функцию

LoadLibrary – загрузить dll-библиотеку

//прототип экспортируемой dll-библиотеки функции

Extern “C” \_\_\_declspec (dllexport) HANDLE SSS (char\* id, LPVOID prm)

**ВЫВОДЫ**

* По принципу работы различают 2 типа серверов: итеративные и параллельные
* Итеративные серверы, как правило, используют для связи с клиентами протокол UDP и используются в тех случаях, когда предполагается, что запросы клиентов являются редкими, а исполнение их не требует много времени. В каждый момент времени, итеративным сервером всегда обслуживается только один клиент.
* В параллельных серверах с каждым клиентом устанавливается TCP-соединение. Одновременно к одному параллельному серверу мб подключено несколько клиентов
* Для организации параллельной работы с несколькими соединениями, применяются механизмы потоков и(или) механизмы процессов. Механизм процессов является наиболее затратным, но и более надежным, т.к. предполагает полное разделение параллельно работающих компонент сервера. С точки зрения программирования проще является, конечно, применение потоков, т.к. в этом случае потоки разделяют общую память процесса, что существенно упрощает разработку.
* Наличие ресурсов, допускающих только последовательное использование делает необходимым синхронизировать поток и(или) процессы. В арсенале windows имеется много различных механизмов синхронизации: критические секции, мьютексы, семафоры, события, асинхронные процедуры, ожидающие таймеры и т.д.
* В тех случаях, когда требуется совместная работа нескольких потоков с одной общей переменной, может быть применен специальный механизм упрощенной синхронизации – атомарные операции
* Для подключения клиентов к параллельному серверу применяется не блокирующий режим работы сокета, позволяющий сделать этот процесс управляемым
* Одним из способов распределения ресурса процессорного времени между параллельными потоками сервера является назначение потоками приоритетов.
* Параллельный сервер, обрабатывающий разнотипные запросы, должен распределять эти запросы с помощью специальной таблицы, связывающей код запроса и функцию обрабатывающего потока. Для того, чтобы сделать процедуры управления сервером независимыми от функций обслуживания, последние вместе с таблицей кодов часто размещают в отдельной динамически подключаемой библиотеке
* При разработке системы безопасности сервера, целесообразно использовать систему безопасности ОС. Это значительно снизит затраты на ее разработку.

# NAT. Межсетевой экран. Proxy server. Ремейлер

NAT (Network Address Translation) –

Существует 3 базовых концепции трансляции адресов:

\*статическая (Static NAT)

\*динамическая (Dynamic NAT)

\*маскарадная (NAPT, NAT Overload, PAT)

**Статический NAT –** отображение незарегистрированного IP адреса на зарегистрированный IP адрес на основании один к одному. Особенно полезно, когда устройство должно быть доступным снаружи сети.

**Динамический NAT** – отображает незарегистрированный IP адрес на зарегистрированный адрес от группы зарегистрированных IP адресов. Динамический NAT также устанавливает непосредственное отображение между незарегистрированным и зарегистрированным адресом, но отображение может меняться в зависимости от зарегистрированного адреса, доступно в пуле адресов во время коммуникации.

# Формат сообщений Internet

RFC 5322

Сообщения, соответствующие данной спецификации, включают символы с десятичными кодами от 1 до 127, интерпретируемые в соответствии и кодировкой ASCII.

Данная спецификация вносит **два ограничения** на число символов в строке:

* строка должна содержать не более 998 символов (без учета CRLF)
* следует использовать строки размером не более 78 (без учета CRLF)
  + CR – возврат каретки (перемещает курсор в начало строки, не переводя на следующую строку)
  + LF – перевод на новую строку.

Поля заголовков представляют собой строки, начинающиеся с имени поля, за которыми следует двоеточие («:»), содержимое поля и знак завершения строки CLRF. Имя поля должно состоять только из печатаемых символов US\_ASCII (т.е., символов с кодами от 33 до 126, включительно), исключая двоеточие. Значение поля может включать печатаемые символы US-ASCII, символы пробела (SP, код ASCII - 32) и горизонтальной табуляции.

Для удобства и с учетом ограничения размеров строки (998/78 символов), значения поля мб разбито на несколько строк, это называется **фальцовкой.**

*Subject: This is a test* (поле: значение поля)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Subject: This*

*is a test*

Процесс преобразования фальцованного многострочного представления поля в обычное однострочное называется **расфальцовкой** и выполняется путем простого удаления всех последовательностей CRLF, непосредственно за которыми следуют пробельные символы (WSP)

Если строка превышает 78 символов, на ее желательно разбить на несколько строк. Перед пробелом ставим сочетание CRLF и значение поля будет разбито на несколько строк. В некоторых реализациях значение превышающее 80 могут не восприниматься.

Некоторые символы имею специальное значение (например, используются в качестве границ лексем). Для использования таких символов в общепринятом смысле служит **механизм квотирования** (добавления «кавычек»).

quoted-pair = (“\” (VCHAR / WSP) ) / obs-qp

**Пробелы для фальцовки и комментарии**

FWS = ([\*WSP CRLF] 1\*WSP) / obs-FWS ; пробельные символы для фальцовки.

ctext = %d33-39 / ; печатаемые символы US-ASCII

%d42-91 / ; не включая

%d93-126 / ; “(“, “)” и “\”

obs-ctext

ccontent = ctext / quoted –pair / comment

comment = “(“ \* ([FWS] ccontent) [FWS] “)”

CFWS = (1 \* ([FWS] comment) [FWS] ) / FWS

Некоторые конструкции в теле структурированных полей заголовков представляют собой просто строки некоторых базовых символов. Такие символ называют **атомами.**

**Задание даты и времени**(вк)

**Задание адреса**

Поле addr-spec представляет собой специфический для интернет идентификатор, содержащий локально интерпретируемую строку, за которой следует символ @ (ASCII - 64) и доменное имя Internet. Это локально интерпретируемая строка представляет собой строку в кавычках или атом с точкой.

**Общий синтаксис сообщений**

**Определение полей**

**Поле даты**

**Поле источника**

**Поля адресов получателей**

From – nash pochtovyj address

Sender – tot kto otpravliajet soobch (sekretarsha)

Reply-to – kuda otpravliajem

**Поле идентификации**

**Информационные поля**

**Поля пересылки**

**Дополнительные поля**

# Протокол HTTP

**HTTP** – протокол передачи гипертакста – протокол прикладного уровня передачи данных (изначально – в виде гипертекстовых документов).

Основой является технология «клиент-сервер», то есть предполагается существование потребителей(клиентов), которые инициируют соединение и посылают запрос, и поставщиков (Серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

HTTP в настоящее время повсеместно используется во всемирной паутине для получения информации с веб-сайтов.

**Спецификация HTTP**

* Название: HyperText Transfer Protocol
* Уровень (по модели OSI): прикладной
* Семейство: TCP/IP
* Создан: 1990г
* Порт/ID: 80/TCP
* Назначение протокола: доступ к гипертексту, ныне стал универсальным
* Спецификация: RFC 1945, RFC 2616
* Основные реализации (клиенты): веб-браузеры (IE, Opera, Mozilla, GCh)
* Основные реализации (серверы): Apache, IIS

**HTTP** используется также в качестве «транспорта» для других протоколов прикладного уровня, таких как SOAP, XML-RPC, WevDAV.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими мб логические объекты или что-то абстрактное.

Особенностью протокола HTTP является возможность указать в запросе и ответе способ представления одного и того же ресурса по различным параметрам: формату, кодировке, языку и т.д. Именно благодаря возможности указания способа кодирования сообщения клиент и сервер могут обмениваться двоичными данными, хотя данный протокол является текстовым.

**HTTP** – протокол прикладного уровня, аналогичными ему являются FTP и SMTP.

Обмен сообщениями идет по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между параметрами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществляет сохранение информации о состоянии, связанной с последними запросами и ответами. Браузер, посылающий запросы, может отслеживать задержки ответов. Сервер может хранить IP-адреса и заголовки запросов последних клиентов. Однако сам протокол не осведомлен о предыдущих запросах и ответах, в нем не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

**Достоинства HTTP**

* **Простота**

Настольно прост в реализации, что позволяет с легкостью создавать клиентские приложения

* **Расширяемость**

Возможности протокола легко расширяются благодаря внедрению своих собственных заголовков, сохраняя совместимость с другими клиентами и серверами. Они будут игнорировать неизвестные заголовки, но при этом можно получить необходимую функциональность при решении специфической задачи.

* **Распространенность**

При выборе протокола для решения некоторых задач немаловажным фактором является его распространённость. Как следствие, это обилие различной документации по протоколу на многих языках мира, включение удобных в использовании средств разработки в популярные IDE, поддержка протокола в качестве клиента многими программами и обширный выбор среди хостинговых компаний с серверами HTTP

**Недостатки HTTP**

* **Большой размер сообщений**

Использование текстового формата в протоколе порождает соответствующий недостаток: большой размер сообщений по сравнению с передачей двоичных данных. Из-за этого возрастает нагрузка на оборудование при формировании, обработке и передаче сообщений. Для решения данной проблемы в протокол встроены средства для обеспечения кэширования на стороне клиента, а также средства компрессии передаваемого контента.

* **Отсутствие «навигации»**

Хотя протокол разрабатывался как средства работы с ресурсами сервера, у него отсутствуют в явном виде средства навигации этих ресурсов. Например, клиент не может явным образом запросить список доступных файлов, как в протоколе FTP. Предполагалось, что конечный пользователь уже знает URI необходимого ему документа, закачав который, он будет производить навигацию благодаря гиперссылкам. Это вполне нормально и удобно для человека, но затруднительно, когда стоят задачи автоматической обработки и анализа всех ресурсов сервера без участия человека. Решение этой проблемы лежит полностью на плечах разработчиков приложений, использующих данный протокол.

* **Нет поддержки распределённости**

Протокол разрабатывался для решения типичных бытовых задач, где само по себе время обработки запроса должно занимать незначительное время или вообще не приниматься в расчет. Но в промышленном использовании с применением распределенных вычислений при высоких нагрузках на сервер протокол HTTP оказывается беспомощным.

**Программное обеспечение**

Все ПО для работы с протоколом делится на **3 большие категории**:

\***серверы** как основным поставщики услуг хранения и обработки информации (обработка запросов-

\***клиенты** – конечные потребители услуг сервера (отправка запроса)

\***прокси** для выполнения транспортных служб.

Для отличия конечных серверов от прокси в официальной документации используется термин origin server. Разумеется, один и тот же программный продукт может одновременно выполнять функции клиента, сервера или посредника в зависимости от поставленных задач. В спецификациях протокола подробно описывается поведение для каждой из этих ролей.

**Структура протокола**

Каждое HTTP-сообщение состоит из 3 частей, которые передается в указанном порядке:

1. Стартовая строка – определяет тип сообщения
2. Заголовки – характеризуют тело сообщения, параметры передеачи и прочие сведения
3. Тело сообщения – непосредственно данные сообщения. Обязательно должно отдельться от заголовков пустой строкой.

Как с помощью гет записать данные на сервер? - смелов

Заголовки и тело сообщения могут отсутствовать, но стартовая строка