Тестирование ПО. Уровень 1

Клиент-серверные приложения

Что такое клиент-серверная архитектура. Тонкий и толстый клиент. Что такое сервер? Что такое веб сервер и сервер приложений? N-звенная архитектура клиент-серверных приложений.

Оглавление

[Клиент-серверные приложения](#_tyjcwt)

[Что такое клиент-серверное приложение?](#_3dy6vkm)

[Тонкий и толстый клиент](#_1t3h5sf)

[Тонкий клиент](#_tf7746nvtct9)

[Толстый клиент](#_iiaveiu0s7jz)

[Сервер](#_uhelaxm2qt8u)

[Производительность](#_q7pdy042juod)

[Надёжность](#_uusdehmtpfjd)

[Масштабируемость](#_4a8659ql5hb9)

[Управляемость](#_uef9dqb0r3wp)

[Пример комплектации сервера Hyperion RS230 G4](#_yzevhsimgq7u)

[Виртуализация и виртуальные серверы](#_kt8cxfhegssk)

[Типы серверов](#_przm79w7glgr)

[Пример работы веб-сервера](#_9jjf2abmscw6)

[Сервер приложений (application server)](#_rkya1acrwjri)

[N-звенная архитектура клиент-серверных приложений](#_i930q3mwsjxf)

[Практика работы с Wireshark](#_86plfe3v6ie7)

[Операторы при построении фильтров](#_cqgjoiuqi0a5)

[Фильтры по запросам к DNS](#_zduwh8vrf9fu)

[Фильтры по IP-адресам](#_cphccbr4pex)

[Фильтры по адресам (только IPv4)](#_8xms0ts9kvfm)

[Фильтры для протоколов](#_731k2p7nm335)

[Фильтры для портов](#_blg9u4z3vzdu)

[Глоссарий](#_jt8s5xnna35v)

[Теория урока](#_3dy6vkm)

[Сложности в теме](#_4d34og8)

[Практическое задание](#_clx8y3gby0rl)

[Дополнительные материалы](#_3rdcrjn)

[Используемые источники](#_lnxbz9)

# Клиент-серверные приложения

## Что такое клиент-серверное приложение?

Клиент-серверное приложение — приложение, которое базируется на технологии «клиент-сервер», где общение между клиентом и сервером происходит по схеме «запрос-ответ». Запрос формируется на клиенте (например, браузер), а ответ — на стороне сервера (например, веб-сервер). Таким образом происходит распределение сетевой нагрузки. Сервер спроектирован, чтобы можно было поддерживать множественную обработку большого количества запросов от множества клиентов. Программа-сервер, отвечающая за обработку клиентских запросов, как правило, размещается на отдельно выделенной машине (физической или виртуальной), и её производительность должна быть гораздо выше, чем производительность машины, на которой установлена программа-клиент.

**Преимущества клиент-серверной архитектуры:**

* многопользовательский режим работы с данными;
* удобство централизованного управления доступом;
* низкая стоимость разработки;
* высокая скорость разработки.

**Недостатки клиент-серверной архитектуры:**

* зависимость производительности от сети (и инфраструктуры в целом), сервера и клиента;
* неработоспособность сервера может уронить всю сеть;
* проблема сервера затрагивает всех пользователей приложения.

Когда говорят про клиент, как правило, выделяют понятия «тонкий клиент» и «толстый клиент».

## Тонкий и толстый клиент

### Тонкий клиент

**Тонкий клиент** — компьютер или программа-клиент, которая запущена (или работает) на стороне клиента и формирует запросы для обращения к удалённым серверам для получения определённых данных или выполнения нужных функций.

Тонкий клиент только формирует запросы. Вся логика обработки запроса и формирование ответа ложится на серверное приложение (сервер).

Наиболее частный пример тонкого клиента — компьютер с установленным веб-браузером.

Также примерами тонкого клиента являются:

1. Thinstation — дистрибутив Linux, разработанный специально для создания тонких клиентов. Представляет собой урезанный Linux с предустановленными программами, необходимыми для работы сети.
2. Linux Terminal Server Project (LTSP) — свободно распространяемый дополнительный пакет для Linux с открытым исходным кодом, который позволяет нескольким людям с маломощными компьютерами (терминалами) использовать вычислительные мощности одного более производительного компьютера (сервера).
3. Терминальный доступ — доступ к информационной системе (ИС), организованный так, что локальная машина-терминал не выполняет вычислительной работы, а лишь перенаправляет ввод информации (от мыши и клавиатуры) на центральную машину (терминальный сервер) и отображает графическую информацию на монитор.
4. Пример: терминалы для оплаты покупок банковскими картами. Терминал считывает с карты данные для аутентификации покупателя. Далее сама аутентификация и транзакция производятся на сервере банка. Результат операции — списание средств или отказ — передаются обратно на терминал.
5. Virtual Network Computing (VNC) — система удалённого доступа к рабочему столу компьютера. Управление осуществляется путём передачи нажатий клавиш на клавиатуре и движений мыши с одного компьютера на другой и ретрансляции содержимого экрана через компьютерную сеть.

**Преимущества тонкого клиента:**

1. Нет необходимости устанавливать на клиентскую машину дистрибутивов большого размера и локальных баз данных.
2. Клиентское ПО не нуждается в администрировании.

**Недостатки тонкого клиента:**

1. Требуется стабильный и постоянный доступ к сети.
2. В случае отказа сервера работа тонкого клиента может быть прекращена.

### Толстый клиент

**Толстый клиент** — приложение, обеспечивающее расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае лишь хранилище данных, а вся работа по их обработке и представлению переносится на машину клиента.

Примером толстого клиента может служить 1С, когда операции проводятся непосредственно на рабочем месте пользователя, а сами данные хранятся на сервере, куда и обращается за ними клиентское приложение.

**Преимущества толстого клиента:**

1. Толстый клиент обладает широким функционалом, в отличие от тонкого.
2. Режим многопользовательской работы.
3. Предоставляет возможность работы даже при обрывах связи с сервером.
4. Высокое быстродействие.

**Недостатки толстого клиента:**

1. Большой размер дистрибутива.
2. Многое в работе клиента зависит от того, для какой платформы он разрабатывался.
3. При работе с ним возникают проблемы с удалённым доступом к данным.
4. Довольно сложный процесс установки и настройки.
5. Сложность обновления и связанная с ней неактуальность данных.
6. Очень важен вопрос синхронизации толстого клиента и базы данных.

# Сервер

Понятие сервера можно рассматривать с двух точек зрения — программного обеспечения и специализированного оборудования. Есть физические серверы, как отдельные машины или специализированное оборудование, а есть серверы, как программное обеспечение, то есть ПО, которое обеспечивает функциональные возможности для других программ или устройств (для клиентов).

**Сервер** — специализированный компьютер или оборудование для выполнения на нём сервисного программного обеспечения.

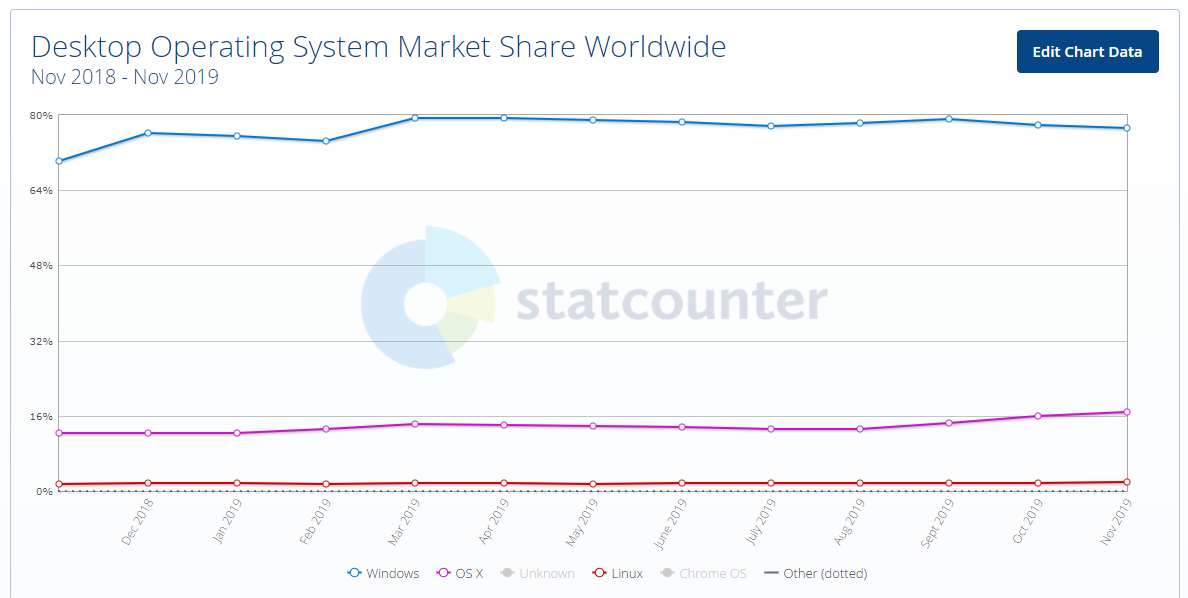
Если говорить проще, сервер — компьютер, который, как и привычный нам домашний ПК, включает в себя процессор, оперативную память, жесткий диск и системную плату. Однако функции у них немного отличаются, и большинство серверов гораздо мощнее стандартных устройств или домашних ПК.

Для серверов — физических машин — выделяют специальные серверные операционные системы, такие как Windows или Linux.

Рассмотрим обзор наиболее популярных ОС для настольных ПК и ноутбуков и отдельно для серверов или серверных машин.

Для настольных ПК распределение на ноябрь 2019 такое:

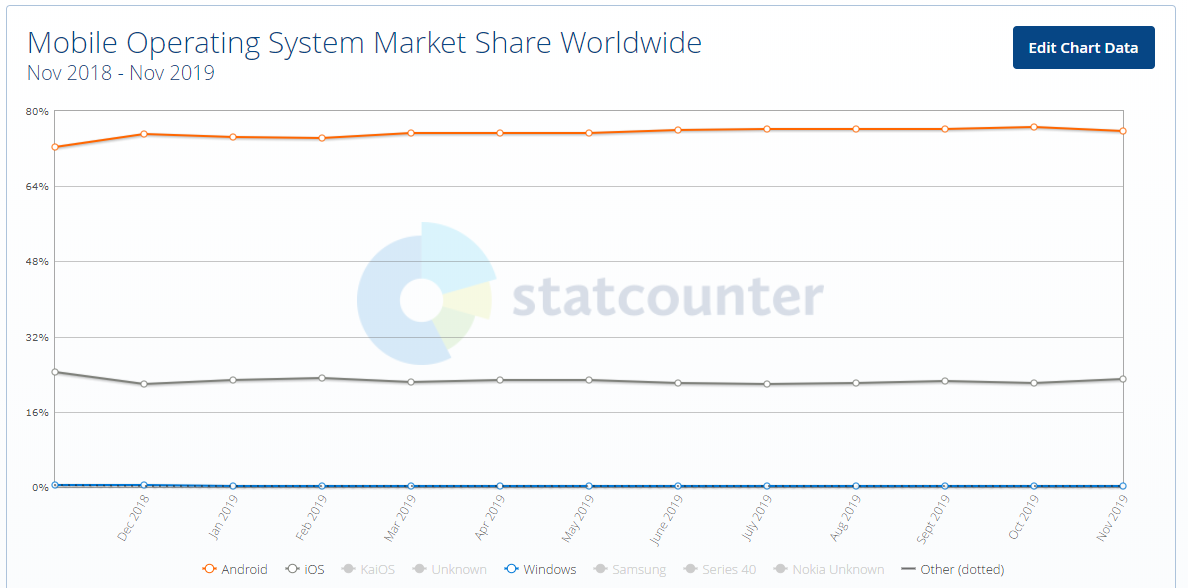
* 1 место: ОС Windows — 77,25%,
* 2 место: ОС MacOS —16,73%,
* 3 место: Linux — 1.82%.



То есть для настольных ПК лидер — ОС семейства Windows.

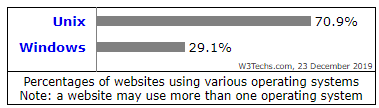
Небольшая статистика для лидера среди мобильных ОС (ноябрь 2019):

* 1 место: ОС Android — 75,82%,
* 2 место: ОС iOS — 22,9%,
* 3 место: Windows — 0.15%.



Среди серверов, используемых для веб-сайтов, другая картина. Среди серверных ОС распределение такое:

* 70,9% — Unix-подобные ОС,
* 29,1% — Windows ОС.



Среди десктопных ОС лидирует Windows, среди серверных — Unix.

Приведенная статистика — для сравнения настольных ПК и серверных машин.

Вернемся к серверам, а именно к машинам физическим.

Изначально, при проектировании, перед серверами стоят совершенно другие цели и задачи, нежели перед обычными ПК. Серверы, как правило, работают в режиме 24\7, нагрузка на них гораздо больше, чем на обычные ПК, требования к их надёжности — выше.

Серверы должны отвечать некоторым требованиям или иметь определенные свойства, такие как:

* Производительность.
* Надёжность.
* Масштабируемость.
* Управляемость.

## **Производительность**

Как правило, в начинке сервера — более мощные комплектующие: многоядерные процессоры, гораздо больший объём оперативной памяти, который может исчисляться десятками гигабайт, а также скорость и объём жестких дисков. Скорость и объём, конечно, гораздо выше, чем у обычных ПК.

## Надёжность

Надёжность — показатель сервера, отличающий его от обычного компьютера. Почему, к примеру, многие говорят «зачем покупать дорогой сервер, когда его можно собрать из простых комплектующих для домашнего компьютера?»

Безусловно можно назначить роль сервера обычному десктопному компьютеру. Но есть «но» — сервер, в отличие от простого компьютера, дороже стоит, оттого, что рассчитан он на более длительное время службы. И изначально в серверную платформу (материнская плата + процессор + память + дисковый массив) закладываются разработки, увеличивающие долговечность и надёжность машины.

Надёжность не только в отношении того что он будет работать, а не зависать и тормозить, что часто бывает с домашним компьютером. В него заложено два вида надёжности:

1. Физическая надёжность — стабильность работы, достигаемая надёжными комплектующими и качественной сборкой в целом.
2. Аппаратная надёжность — отсутствие программных сбоев за счёт стабильности работы аппаратной части.

## Масштабируемость

Масштабируемость — возможность увеличить вычислительную мощность сервера за счёт установки большего числа процессоров, оперативной памяти и т. д., или их замены на более производительные. Это — масштабируемость аппаратная.

Изначально серверы в продаже идут в базовой комплектации, но с заложенным потенциалом к апгрейду — аппаратная масштабируемость.

К примеру, в базовом наборе сервера — один процессор, два модуля памяти (в серверах всегда используются только парные модули памяти), например, 2х2 Гб. и дисковый массив из двух жестких дисков, допустим, 146 Гб. Далее (или сразу) по мере потребности можно доустановить еще один процессор, память или добавить диски в массив.

## Управляемость

Возможность удалённого мониторинга состояния, удалённое включение и перезагрузка сервера. Возможность удалённой диагностики сервера даже в выключенном состоянии, системы прогнозирования неполадок. В серверах уже давно внедряются модули, позволяющие делать всю диагностическую работу удалённо, в любом состоянии, главное условие — чтобы присутствовало электрическое питание.

## Пример комплектации сервера Hyperion RS230 G4

Рассмотрим пример комплектации сервера.



Конфигурация серверной машины на примере Hyperion RS230 G4. Сервер предназначен для производительных вычислений.

**Системные требование сервера:**

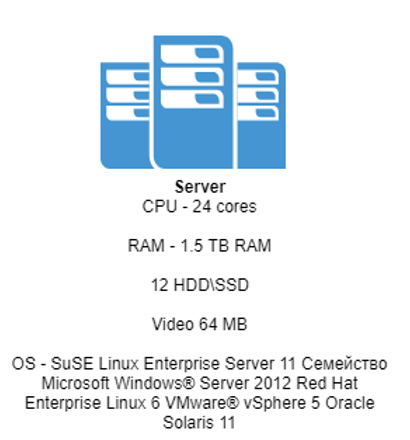
1. Применение: виртуализация, большие БД, cloud computing, data mining, ERP, CRM, сервер Citrix, производительные вычисления.
2. Набор микросхем: Intel C602.
3. Процессоры: 1 или 2 Intel Xeon E5-26xx v2 (до 24 ядер).
4. Системная шина: Intel DMI 4GB/s.
5. Максимальный объём памяти: 1.5TB восьмиканальной DDR3, 24 слота.
6. Встроенные контроллеры: технология дочерних плат расширения.
7. Intel SAS RAID: 8 портов (активация через дополнительный ROM).
8. Опциональные контроллеры: дочерняя плата SAS HBA (не занимает слот PCI), дочерняя плата двухпортового 10 Gb Ethernet SFP+ SAS RAID контроллеры.
9. Адаптеры FibreChannel и InfiniBand HCA.
10. Максимальное количество дисков: 12x 3.5” SAS/SATA/SSD с горячей заменой, 6G экспандер с двойным входом или 24x 2.5” SAS/SATA/SSD с горячей заменой, 6G экспандер с двойным входом, 1 внутренний SATA DOM 2x 2.5” SATA/SSD с горячей заменой.
11. Видеоконтроллер: ASPEED 2300, 64Mb.
12. Слоты расширения: 3x PCI-E 3.0 16x 2x PCI-E 3.0 16x LP 2x PCI-E 3.0 4x mezz.
13. Сетевые интерфейсы: 2x Intel Powerville I350 1GbE RJ45 или 2x Intel Twinville X540 10GbE RJ45 2x Intel® 82599ES 10Gb SFP+ дочерняя плата (опция), поддержка технологии FCoE.
14. Интерфейсы: задняя панель VGA RS232 3 x RJ45 2 x USB.
15. Управление системой: IPMI 2.0 интегрировано KVM over IP, Virtual Media, выделенный порт Ethernet DCMI.
16. Поддерживаемые ОС: SuSE Linux Enterprise Server 11, семейство Microsoft Windows® Server 2012, Red Hat Enterprise, Linux 6, VMware®, vSphere 5, Oracle Solaris 11.
17. Блок питания: 1100Вт (опционально 460Вт) дублированный с горячей заменой, поддержка Intel Node Management.
18. Охлаждение: 4 управляемых вентилятора охлаждения системы с легкой заменой.
19. Рабочие условия: относительная влажность: 10-75%, диапазон температур: 10-35 °C.

## Виртуализация и виртуальные серверы

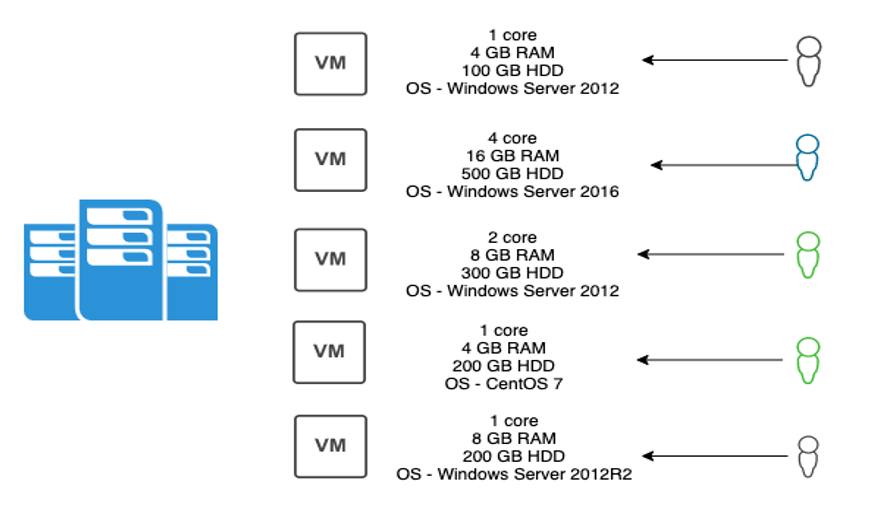
Если у нас в компании или на проекте есть хотя бы один такой мощный сервер, его, скорее всего, будут использовать для создания набора виртуальных машин для проектных необходимостей. На базе одного такого сервера можно создать несколько тестовых для команды тестирования с разным набором конфигураций и операционных систем.

К примеру, у нас есть сервер:

* 24 ядра,
* 1,5 TB (терабайт) оперативной памяти RAM,
* 12 слотов (разъемов) для подключения HDD или SSD,
* Поддерживаемые ОС — семейства Linux или семейства Windows.



Имея такой сервер, можно создать несколько виртуальных машин для проектных необходимостей. Несколько машин для команды разработки и несколько машин (тестовых виртуальных машин) для команды тестирования.



## Типы серверов

Рассмотрим, что такое сервер с программной точки зрения. Есть несколько типов программных серверов. Перечислим несколько их видов:

* Веб-сервер.
* Вычислительный сервер.
* Игровой сервер.
* Медиасервер.
* Почтовый сервер.
* Прокси-сервер.
* Сервер базы данных.
* Сервер приложений.
* Файловый сервер.
* DNS Server.

Скорее всего, вы будете сталкиваться с веб-серверами при тестировании веб-приложений. А также серверами баз данных, в которых вас будут интересовать те или иные базы, и, возможно, серверы приложений.

**Веб-сервер (web server)**

Веб-сервер — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиапотоком или другими данными.

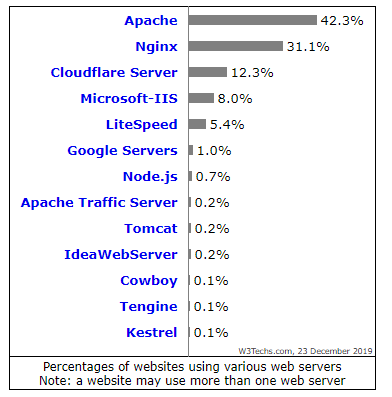
Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором оно работает.

Клиент, обычно — веб-браузер, передаёт веб-серверу запросы на получение ресурсов, обозначенных URL-адресами. Ресурсы — HTML-страницы, изображения, файлы, медиапотоки или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт клиенту запрошенные данные. Этот обмен происходит по протоколу HTTP или HTTPS.

Веб-серверы могут иметь различные дополнительные функции, например,

* автоматизация работы веб-страниц;
* ведение журнала обращений пользователей к ресурсам;
* аутентификация и авторизация пользователей;
* поддержка динамически генерируемых страниц;
* поддержка [HTTPS](https://ru.wikipedia.org/wiki/HTTPS) для защищённых соединений с клиентами.

Ниже представлена статистика наиболее популярных веб-серверов.



Один из наиболее распространённых веб-серверов — Apache. Это свободный веб-сервер, наиболее часто используемый в UNIX-подобных операционных системах.

Некоторые другие используемые веб-серверы:

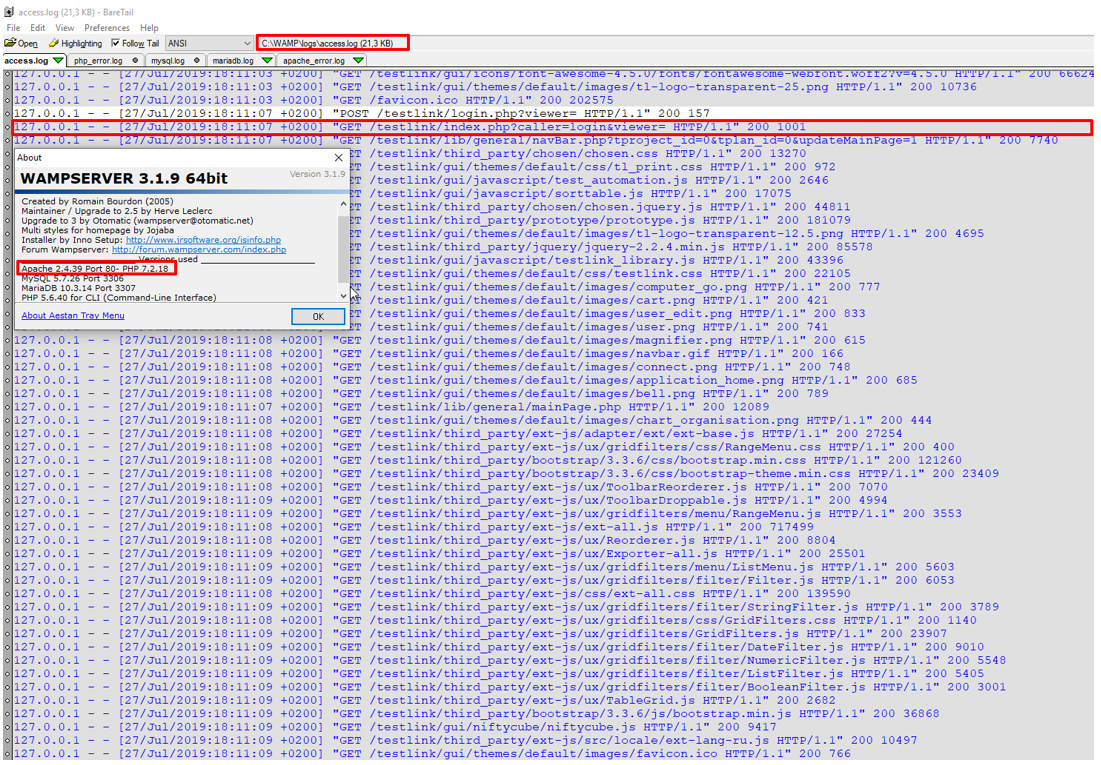
1. nginx — свободный веб-сервер,пользующийся большой популярностью на крупных сайтах;
2. IIS от компании Microsoft, распространяемый с ОС семейства Windows;
3. lighttpd — свободный веб-сервер;
4. Google Web Server — веб-сервер, основанный на Apache и доработанный компанией Google;
5. Resin — свободный веб-сервер приложений;
6. Cherokee — свободный веб-сервер, управляемый только через web-интерфейс;
7. Rootage — веб-сервер, написанный на Java;
8. THTTPD — простой, маленький, быстрый и безопасный веб-сервер;
9. Open Server — бесплатная программа с графическим интерфейсом, использует множество исключительно свободных программных комплексов;
10. H2O — свободный быстрый веб-сервер, написанный на C.

В качестве клиентов для обращения к веб-серверам могут использоваться различные программы и устройства:

* веб-браузер, работающий на настольном компьютере или переносном устройстве (например, карманном ПК);
* разнообразные программы, самостоятельно обращающиеся к веб-серверам для получения обновлений или другой информации (например, антивирус может периодически запрашивать у определённого веб-сервера обновления своих баз данных);
* мобильный телефон, получающий доступ к ресурсам веб-сервера при помощи протокола WAP;
* другие цифровые устройства или бытовая техника.

## Пример работы веб-сервера

Может быть, некоторые из вас устанавливали WAMP для Testlink. В комплект WAMP входит веб-сервер Apache, одна из задач которого — логирование запросов или обращений к приложению.



## Сервер приложений (application server)

Помимо веб-сервера, существует и сервер приложений.

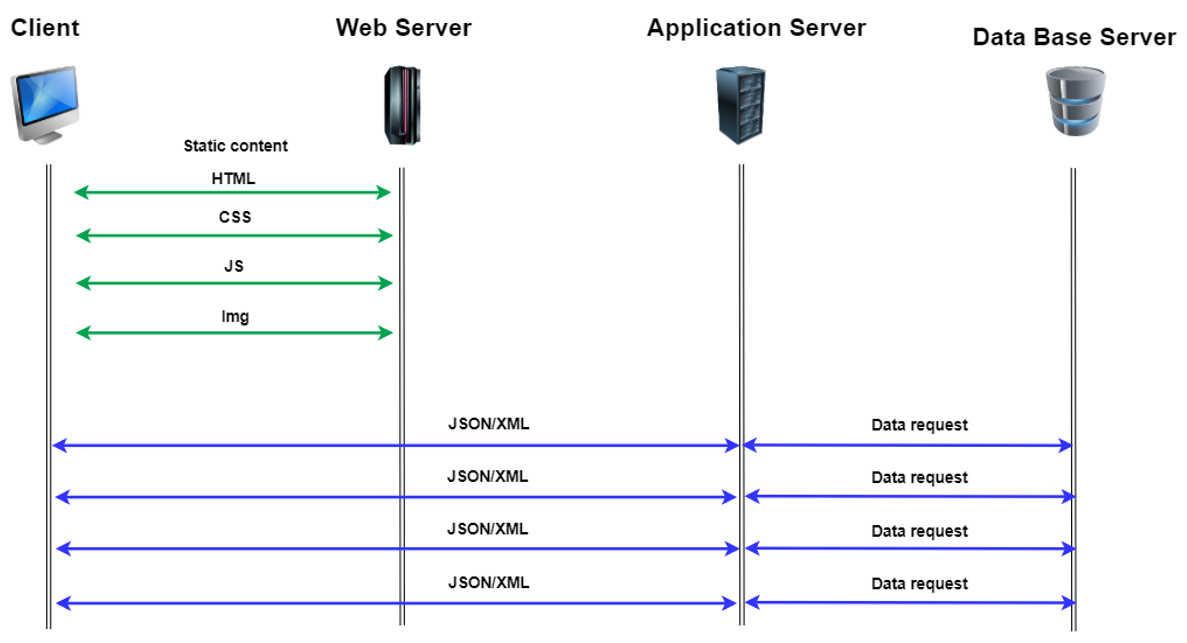
Сервер приложений (англ. application server) — программная платформа (фреймворк), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API (интерфейс прикладного программирования), определённый самой платформой. Сервер приложений отвечает за выполнение бизнес-процессов приложения.

Для веб-приложений основная задача компонентов сервера — обеспечивать создание динамических страниц. В случае Java-сервера приложений, сервер приложений ведёт себя как расширенная виртуальная машина для запуска приложений, прозрачно управляя соединениями с базой данных с одной стороны, и соединениями с веб-клиентом с другой.

Примеры реализации серверов приложений:

1. К классу серверов приложений относятся такие продукты, как Sun GlassFish, IBM WebSphere, RedHat JBoss Application Server, Apple WebObjects (англ.), Oracle Weblogic Server и др.
2. Zope, развитый сервер веб-приложений.
3. Терминальные серверы, например, поставляемые компанией Citrix.

Пример работы клиента с веб-сервером и сервером приложений.



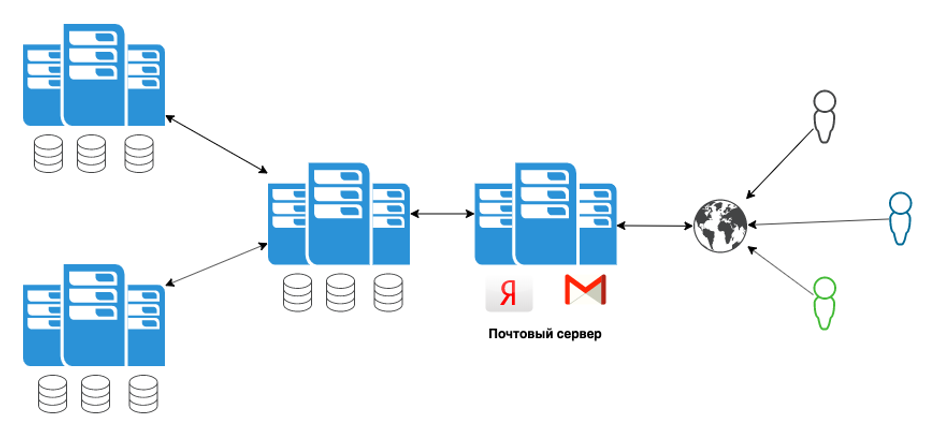
Более подробно о том, как это происходит.

1. Клиент отправляет запрос в сторону веб-сервера для получения статического контента — самой HTML-страницы и её контента. Как мы уже с вами смотрели, при первоначальном запросе к веб-серверу, на котором хостится сайт или веб-приложение, клиент от веб-сервера получает документ HTML. Дальше происходит запрос дополнительных статических файлов, — файлы со стилями, файлы с JS, картинки. Страница полностью загрузилась и отображается на UI-клиенте, то есть это и есть интерфейс пользователя.
2. Далее, работая с приложением, клиент отправляет запрос для реализации определённой бизнес-логики. Этот запрос уже адресуется в сторону сервера приложений. Когда JS-код загружен, и, к примеру, пользователь заполняет форму, нажимает на кнопку отправить/загрузить или ещё какое-то действие, которое требует выполнения бизнес-логики.
3. На стороне сервера приложений происходит обработка запроса и создание запроса в сторону сервера баз данных.
4. Далее происходит отправка запроса в сторону сервера базы данных для получения данных.
5. После получения ответа от сервера базы данных, сервер приложений формирует ответ клиенту и отправляет его. Ответ клиенту формируется, как правило, в виде сырых данных, то есть возвращается документ в формате JSON или XML.
6. Клиент, получив эти данные, используя загруженный javascript-код, обрабатывает полученные данные и отображает их пользователю.

Надо сказать, что сейчас очень популярны форматы данных JSON, поскольку их отображение проще, и размер таких файлов меньше, чем файлы XML.

# N-звенная архитектура клиент-серверных приложений

На примере рассматриваем трёхуровневую архитектуру **физического** построения клиент-серверных приложений.



Первый уровень — клиент, браузер, к примеру.

Второй уровень — сервер приложений.

Третий уровень — сервер базы данных.

На первом уровне формируется запрос от клиента, на втором он обрабатывается и формируется запрос к базе данных, на третьем — запрос к базе обрабатывается и возвращает результат на второй уровень. Второй уровень формирует ответ на первоначальный запрос от клиента и возвращает его.

Сейчас это очень популярная архитектура разработки клиент-серверных приложений. Но часто в литературе у разных авторов можно услышать и про N-звенную архитектуру (N-tier или multi-tier). N-звенная архитектура — **логическое представление** приложения с разбиением на несколько логических уровней:

* Уровень представления (Presentation layer).
* Уровень приложения (Application layer).
* Уровень бизнес логики (Business layer).
* Уровень доступа к данным (Data access layer).

# Практика работы с Wireshark

Во время тестирования клиент-серверных приложений часто необходимо проанализировать, какие данные отправляются со стороны клиента в сторону сервера. Для перехвата и анализа пакетов данных, отправляемых клиентом (или сервером) используется программа WireShark.

Здесь требуется напомнить, что WireShark нужно использовать только в легальных целях.

Для чего может использоваться WireShark:

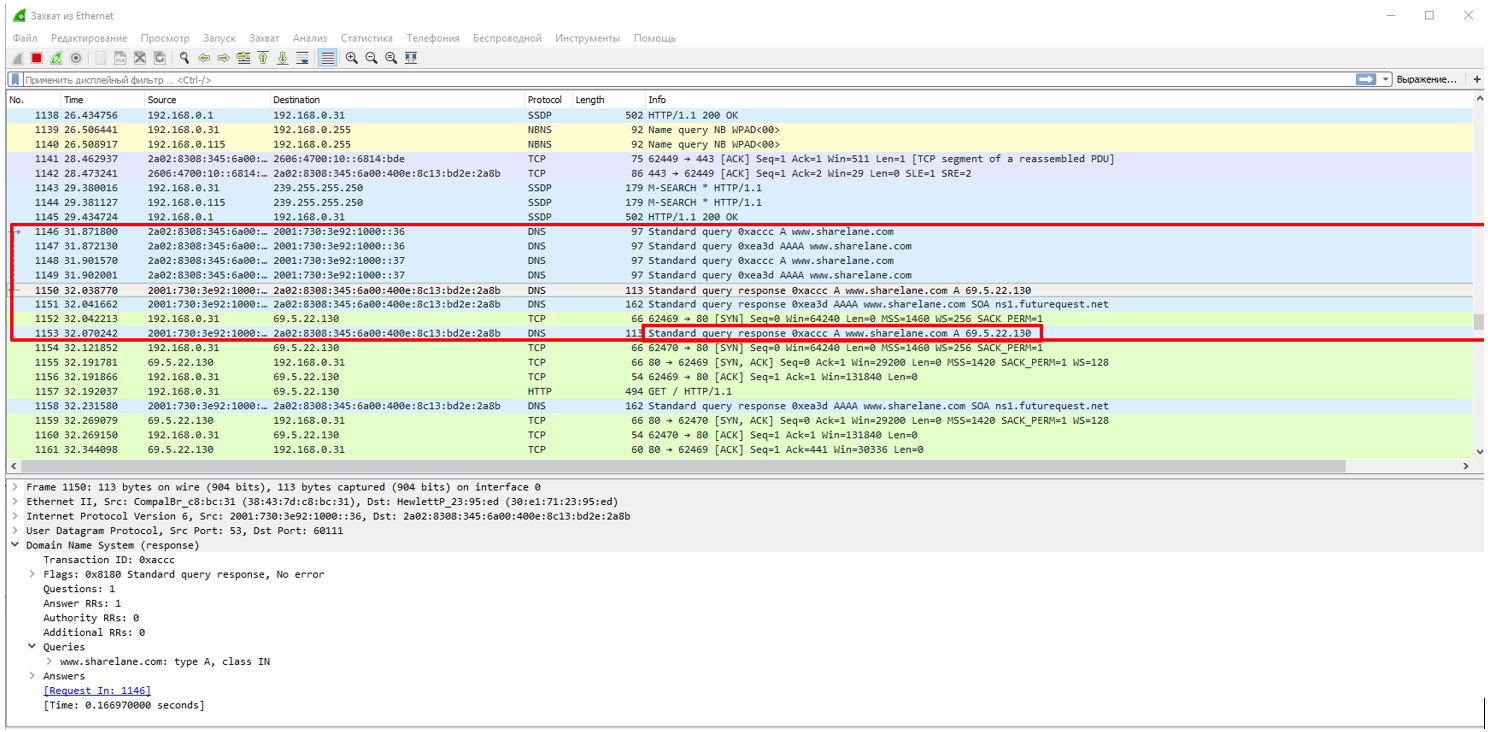
* Глубокая проверка множества протоколов.
* Захват трафика и его анализ в автономном режиме.
* Использование на многих платформах: Windows, Linux, macOS, Solaris, FreeBSD, NetBSD и других.
* Просмотр захваченных сетевых данных с помощью графического интерфейса.
* Использование множества фильтров для трафика.
* Возможность анализа VoIP.
* Чтение и запись множества различных форматов файлов захвата: tcpdump (libpcap), Pcap NG, Catapult DCT2000, iplog Cisco Secure IDS, Microsoft Network Monitor, Network General Sniffer® (сжатый и несжатый), Sniffer® Pro и NetXray®, Network Instruments Observer, NetScreen snoop, Novell LANalyzer, RADCOM WAN / LAN Analyzer, Shomiti / Finisar Surveyor, Tektronix K12xx, Visual Networks Visual UpTime, WildPackets EtherPeek / TokenPeek / AiroPeek и другие.
* Поддержка расшифровки для многих протоколов, включая IPsec, ISAKMP, Kerberos, SNMPv3, SSL / TLS, WEP и WPA / WPA2.
* Экспорт данных в XML, PostScript®, CSV или обычный текст.

Чтобы понять, как происходит установка соединения и обмен данными между клиентом и сервером, посмотрим на примере открытия сайта sharelane.com.

Делаем запрос на получение какого-либо ресурса, в нашем случае — получение HTML-страницы сайта sharelane.com.

В первую очередь компьютер отправляет запрос на DNS-сервер, чтобы разрешить набранный адрес сайта в IP-адрес сервера, куда надо подключаться.

Как видно, подключение происходит по протоколу DNS:



Мы видим, что DNS-сервер в ответ на запрос вернул нам IP 69.5.22.130.

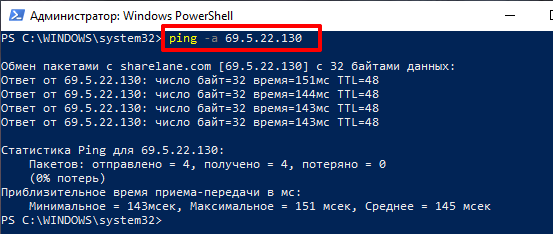
Пропингуем этот IP-адрес и посмотрим, что нам вернется.

|  |
| --- |
| $ ping 69.5.22.130 |

Мы видим, что нам приходит ответ. Но не знаем, откуда. Чтобы проверить какой IP мы пингуем, нужно добавлять ключ –a для команды ping.

Итак, команда:

|  |
| --- |
| $ ping -a 69.5.22.130 |



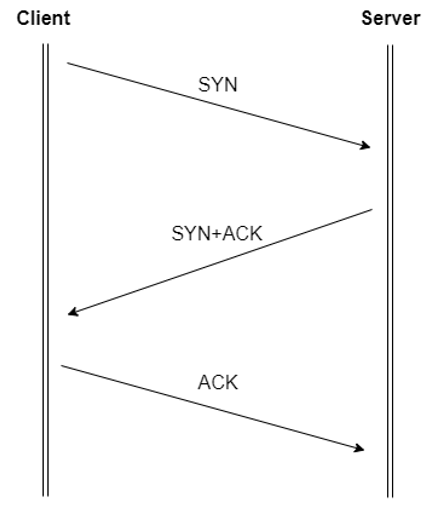
Когда клиент знает, куда отправлять запросы (на какой IP), дальнейшее подключение (установка соединения), между клиентом и сервером происходит через так называемое трёхстороннее рукопожатие.

В нашем случае, инициатор подключения — клиент (наш браузер) — отправляет первый запрос (с флагом SYN) в сторону сервера на подключение.

Сервер отправляет пакет ответа (с флагами SYN+ACK) обратно клиенту.

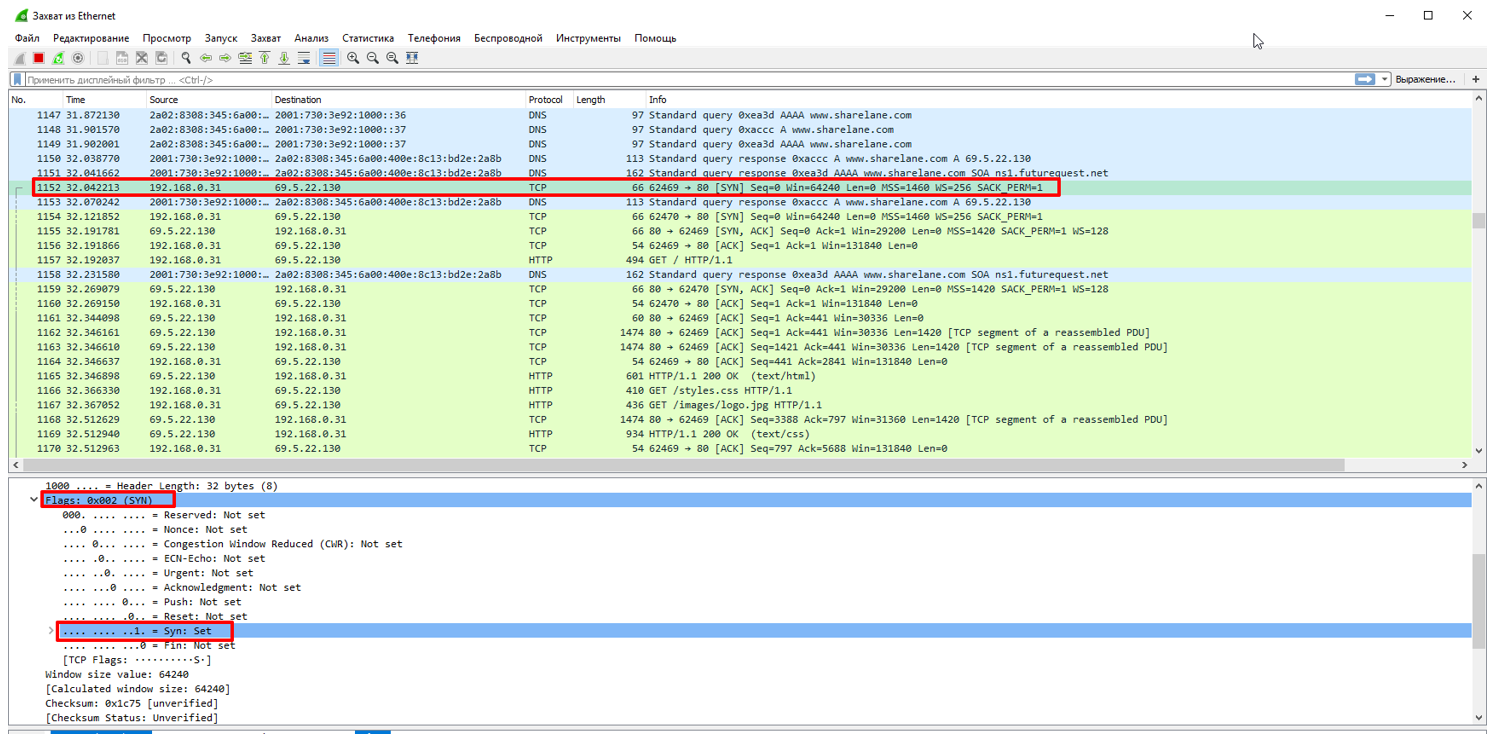
Для завершения установки соединения клиент отправляет сообщение с флагом ACK.

Это можно представить в таком виде:

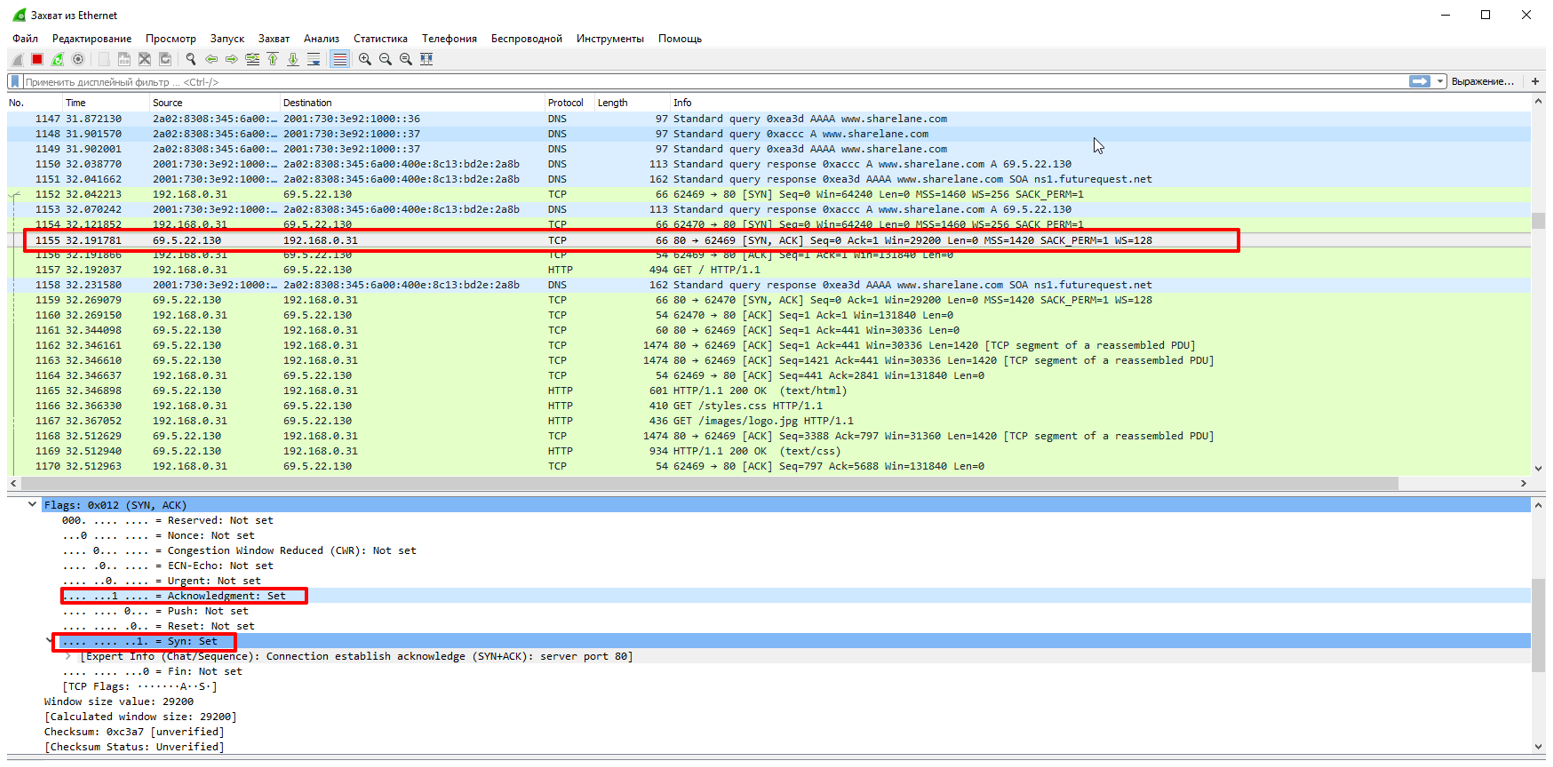


Как это выглядит в WireShark:

1. Отправляется первый пакет с флагом SYN.

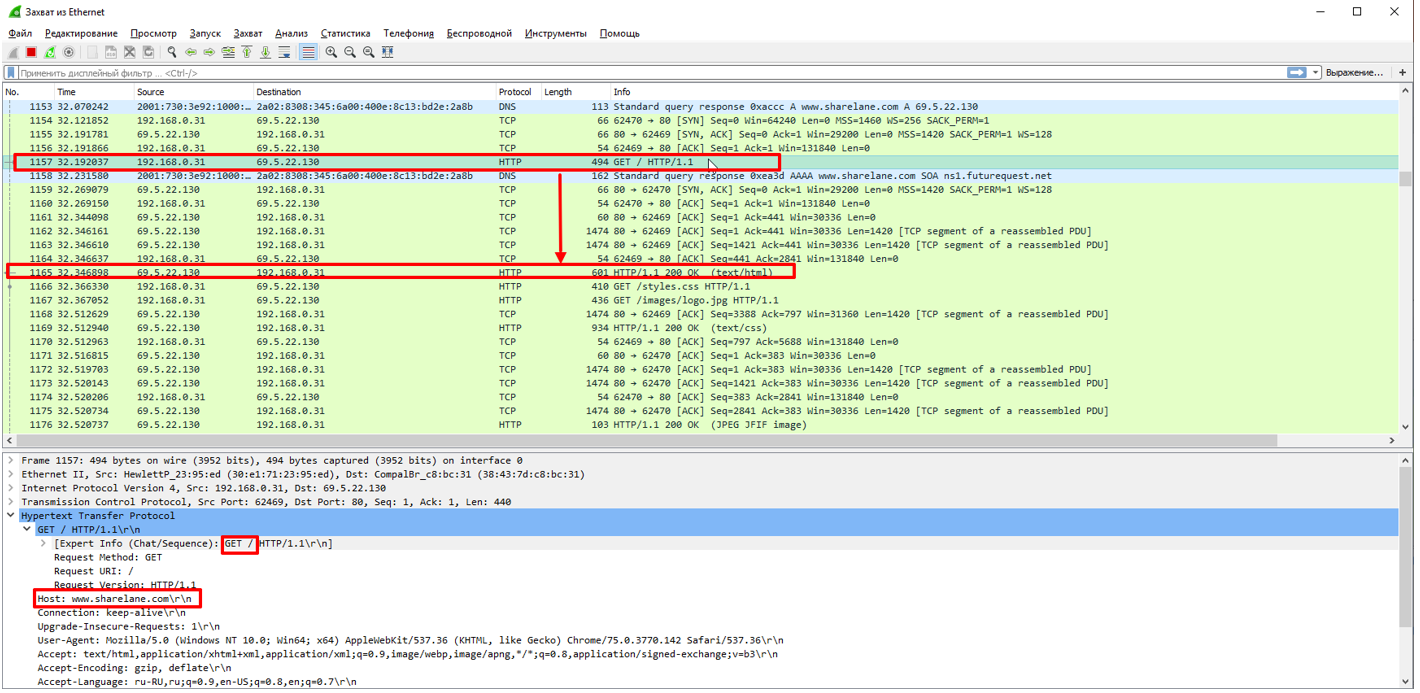


1. Далее сервер отправляет пакет с включенными флагами SYN и ACK.



1. Последний этап — клиент отправляет пакет с флагом ACK (предыдущий скриншот — пакет с флагом ACK). Теперь можно считать, что соединение между клиентом и сервером установлено.

Дальше происходит запрос и передача запрашиваемой страницы от сервера клиенту — сначала отправка документа в формате HTML, а затем — документов со стилями, картинок и так далее.

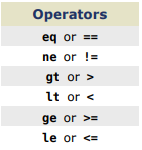


## Операторы при построении фильтров

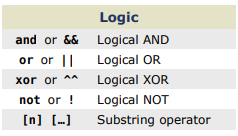
Посмотрим, какие операторы используются при построении фильтров.

|  |  |  |
| --- | --- | --- |
| **Команда** | **Значение** | **Пример использования** |
| = = | Равенство | ip.dst == 193.168.3.10 |
| != | Не равно | udp.dst != 53 |
| < | Меньше чем | ip.ttl < 24 |
| > | Больше чем | frame.len > 10 |
| <= | Меньше или равно | frame.len <= 0x20 |
| >= | Больше или равно | tcp.analysis.bytes\_in\_flight >= 1000 |
| matches | Регулярные выражения | frame matches "[Pp][Aa][Ss][Ss]" |
| contains | Содержит | dns.resp.name contains google |

Можем использовать обычные операторы:



Или логические операторы:



Мы с вами посмотрели, как происходит первоначальное подключение к серверу, обмен пакетами после подключения и получение требуемых документов с сервера. Теперь поговорим про фильтры, которые используются для исследования полученного трафика.

На случай, если вы постоянно не пользуетесь Wireshark, чтобы не вспоминать постоянно какие-то фильтры, добавьте в менеджер фильтров подсказку, какой фильтр для чего. Это находится здесь — Manage Display Filters. Нажимаете + и добавляете правило.

### Фильтры по запросам к DNS

* $ dns.resp.name contains sharelane,
* $ dns contains sharelane.

Ещё мы будем фильтровать IP-адреса назначения, откуда исходил трафик или протоколы и порты.

### Фильтры по IP-адресам

|  |  |
| --- | --- |
| **Фильтруем по IP назначения** | |
| $ ip.dst == 69.5.22.130 | фильтруем по получателю |
| **Фильтруем по IP отправления** | |
| $ ip.src == 192.168.0.115 | фильтруем по отправителю |

### Фильтры по адресам (только IPv4)

|  |  |
| --- | --- |
| $ ip.addr == 192.168.0.115 | Присутствует в отправителе или получателе |
| $ ip.addr == 69.5.22.130 | Присутствует в отправителе или получателе |
| **Только IPv4** | |
| $ ip | |
| **Только IPv6** | |
| $ ipv6 | |

### Фильтры для протоколов

|  |  |
| --- | --- |
| **Фильтруем по протоколу HTTP** | |
| $ http |  |
| **Фильтруем по протоколу DNS** | |
| $ dns |  |
| **Фильтруем по протоколу ICMP** | |
| $ icmp | Если мы пингуем какой-либо ресурс или IP |

### Фильтры для портов

|  |  |
| --- | --- |
| $ tcp.port == 80 | HTTP |
| $ tcp.port == 443 | HTTPS |

# Глоссарий

**Клиент-серверное приложение** — приложение, которое базируется на технологии «клиент-сервер», где общение между клиентом и сервером происходит по схеме «запрос-ответ». Запрос формируется на клиенте (например, браузер), а ответ — на стороне сервера (например, веб-сервер).

**Тонкий клиент** — компьютер или программа-клиент, которая запущена (или работает) на стороне клиента и формирует запросы для обращения к удаленным серверам, чтобы получить определённые данные или выполнить свои функции

**Толстый клиент** — приложение, обеспечивающее расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае — хранилище данных, а вся работа по их обработке и представлению переносится на машину клиента.

**Сервер** — специализированный компьютер или оборудование для выполнения на нём сервисного программного обеспечения.

# Практическое задание

**Используя приложенную таблицу, выполните следующие задания:**

1. Ответьте на вопросы в приложенной форме. Выберите только один вариант ответа.
2. Установите на свою машину (host машину) Postman. Используя Postman, отправьте указанные в приложенной форме GET-запросы. В колонке ответа укажите, какой статус код был вами получен, например, 200, 410 или 502.

## Требования к выполненной работе

1. Практическое задание должно быть выполнено в приложенной форме.
2. Расширение файла должно быть формата Excel (XLS, XLSX).
3. Название файла не менять, и вместо ФИО укажите свои фамилию и имя.

# Дополнительные материалы

1. [Многоуровневые системы клиент-сервер](http://www.osp.ru/nets/1997/06/142618/).
2. [Трёхуровневая архитектура](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D1%91%D1%85%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0).
3. [Multitier architecture](https://en.wikipedia.org/wiki/Multitier_architecture).
4. <https://q-seo.com.ua/blog/gid-po-3xx-redirektam/>
5. [About Wireshark](https://www.wireshark.org/).
6. [Wireshark (русская Википедия)](https://ru.wikipedia.org/wiki/Wireshark).
7. [Wireshark — приручение акулы](https://habr.com/ru/post/204274/).
8. [DisplayFilters](https://wiki.wireshark.org/DisplayFilters).
9. [Cheat sheets: tcpdump and Wireshark](https://packetlife.net/blog/2008/oct/18/cheat-sheets-tcpdump-and-wireshark/).
10. [Видео SF18EU - 01: Back to the Basics (Hansang Bae)](https://www.youtube.com/watch?v=y13zH-8OPE8&t=9s).

# Используемые источники

Для подготовки методического пособия мы использовали эти ресурсы:

1. [Тонкий клиент](https://dic.academic.ru/dic.nsf/ruwiki/83095).
2. [Толстый клиент](https://dic.academic.ru/dic.nsf/ruwiki/690886).
3. [Statcounter](https://gs.statcounter.com/).
4. [Comparison of the usage statistics of Linux vs. Windows for websites](https://w3techs.com/technologies/comparison/os-linux,os-windows).
5. [Multitier architecture](https://en.wikipedia.org/wiki/Multitier_architecture).
6. [Архитектура веба: основы для начинающих разработчиков](https://tproger.ru/translations/web-architecture-101/).