# DHCP (Dynamic Host Configuration Protocol)

DHCP (англ. Dynamic Host Configuration Protocol — протокол динамической конфигурации узла) — это сетевой протокол, позволяющий компьютерам автоматически получать IP-адрес и другие параметры, необходимые для работы в сети TCP/IP. Данный протокол работает по модели «клиент-сервер». Для автоматической конфигурации компьютер-клиент на этапе конфигурации сетевого устройства обращается к так называемому серверу DHCP, и получает от него нужные параметры. Сетевой администратор может задать диапазон адресов, распределяемых сервером среди компьютеров. Это позволяет избежать ручной настройки компьютеров сети и уменьшает количество ошибок. Протокол DHCP используется в большинстве крупных (и не очень) сетей TCP/IP.

Работа протокола DHCP базируется на классической схеме клиент-сервер. В роли клиентов выступают компьютеры сети, стремящиеся получить IP-адреса в так называемую аренду (lease), а DHCP-серверы выполняют функции диспетчеров, которые выдают адреса, контролируют их использование и сообщают клиентам требуемые параметры конфигурации. Сервер поддерживает пул свободных адресов и, кроме того, ведет собственную регистрационную базу данных. Взаимодействие DHCP-серверов со станциями-клиентами осуществляется путем обмена сообщениями.

Стандарт протокола DHCP был принят в октябре 1993 года. Действующая версия протокола (март 1997 года) описана в RFC 2131. Новая версия DHCP, предназначенная для использования в среде IPv6, носит название DHCPv6 и определена в RFC 3315 (июль 2003 года).

DHCP появился не на пустом месте - различные схемы управления IP-адресами в сетевой среде предлагались и раньше (см. врезку <Распределение IP-адресов...>). Однако эти схемы имеют по крайней мере один из двух недостатков - не допускают динамического назначения IP-адресов либо позволяют передавать от сервера на станцию-клиент лишь небольшое число параметров конфигурации.

При разработке протокола DHCP преследовалась цель устранить оба ограничения. Требовался механизм, который позволил бы ликвидировать стадию ручного конфигурирования компьютеров, поддерживал многосегментные сети, не требуя наличия DHCP-сервера в каждой подсети, не конфликтовал с существующими сетевыми протоколами и компьютерами, имеющими статичную конфигурацию, был способен взаимодействовать с ретранслирующими агентами протокола BOOTP и обслуживать BOOTP-клиентов, наконец, допускал управление передаваемыми параметрами конфигурации. BOOTP протокол, позволяющий бездисковым клиентам во время старта конфигурировать установки TCP/IP. Что касается более узких задач, то DHCP должен был обеспечивать уникальность сетевых адресов, используемых разными компьютерами сети в данный момент, сохранение прежней конфигурации клиентской станции после перезагрузки клиента или сервера, автоматическое присвоение параметров конфигурации вновь подключенным машинам.

Упомянутое выше требование поддержки базовых элементов протокола BOOTP возникло не случайно. DHCP разрабатывался как непосредственное расширение BOOTP и именно в таком качестве воспринимается BOOTP-клиентами. Этому обстоятельству в первую очередь способствует формат сообщений DHCP, во многом совпадающий с форматом, который применяется протоколом-предшественником и определен в документе RFC 951.

Сравнивая протоколы BOOTP и DHCP, нельзя не отметить появления в DHCP новых услуг. Во-первых, в этом протоколе предусмотрен механизм автоматической выдачи IP-адресов во временное пользование с возможностью их последующего присвоения новым клиентам. Во-вторых, клиент может получить от сервера все параметры конфигурации, которые ему необходимы для успешного функционирования в IP-сети.

Указанные отличия потребовали частичного расширения формата сообщений. Так, в нем появилось отдельное поле идентификатора клиента, сделана более прозрачной интерпретация адреса сервера (поле siaddr), переменный размер получило поле options, используемое, в частности, для передачи параметров конфигурации (его длина обычно находится в диапазоне 312-576 байт, хотя возможно и дополнительное расширение этого поля за счет полей sname и file).

В роли транспортного протокола для обмена DHCP-сообщениями выступает UDP. При отправке сообщения с клиента на сервер используется 67-й порт DHCP-сервера, при передаче в обратном направлении - 68-й. Эти номера портов, как и схожая структура сообщений, обеспечивают обратную совместимость DHCP с BOOTP. Конкретные процедуры взаимодействия клиентов и серверов BOOTP и DHCP регламентирует документ RFC 1542.

DHCP является расширением протокола BOOTP, использовавшегося ранее для обеспечения бездисковых рабочих станций IP-адресами при их загрузке. DHCP сохраняет обратную совместимость с BOOTP.

**Устройство протокола**

Протокол DHCP является клиент-серверным, то есть в его работе участвуют клиент DHCP и сервер DHCP. Передача данных производится при помощи протокола UDP, при этом сервер принимает сообщения от клиентов на порт 67 и отправляет сообщения клиентам на порт 68.

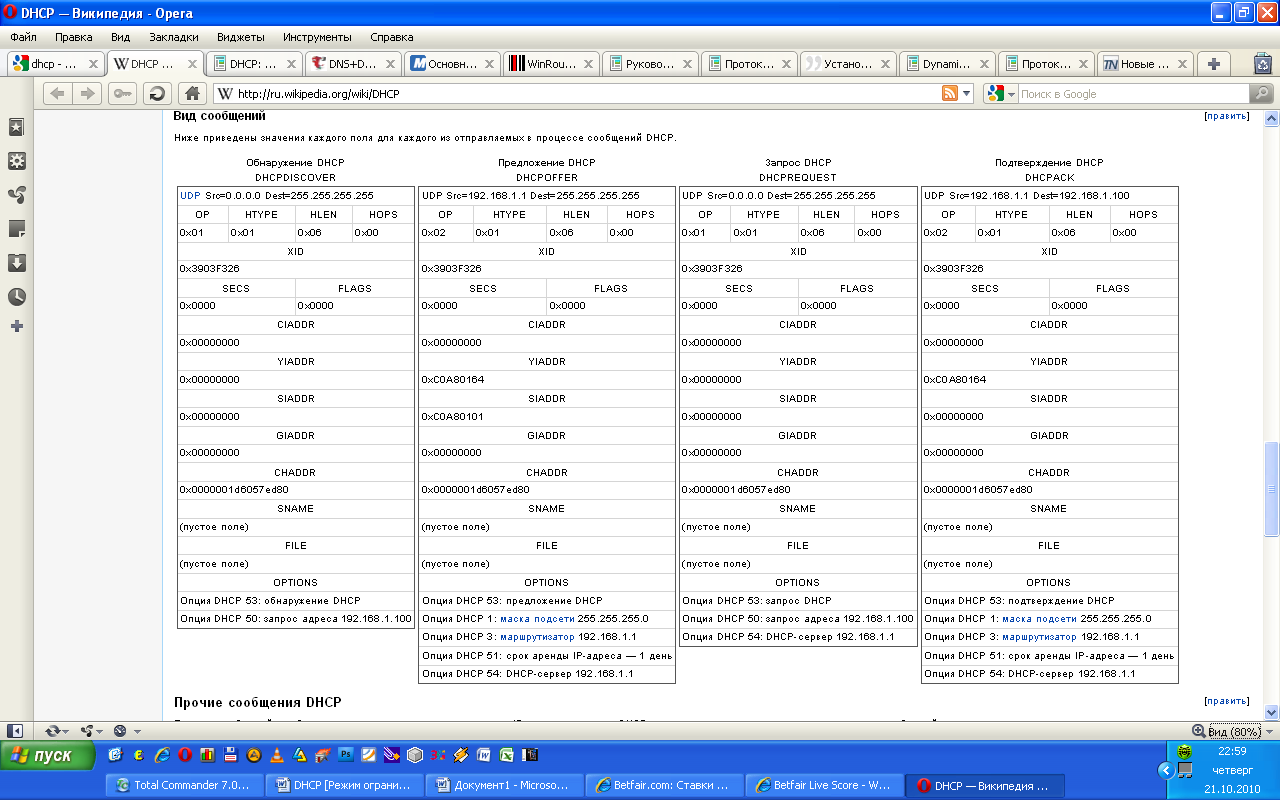
**Структура сообщений DHCP**

Все сообщения протокола DHCP разбиваются на поля, каждое из которых содержит определённую информацию. Все поля, кроме последнего (поля опций DHCP), имеют фиксированную длину.

|  |  |  |
| --- | --- | --- |
| Поле | Описание | Длина (в байтах) |
| op | Тип сообщения. Может принимать два значения: BOOTREQUEST (1, запрос от клиента к серверу) и BOOTREPLY (2, ответ от сервера к клиенту). | 1 |
| htype | Тип аппаратного адреса. Допустимые значения этого поля определены в RFC «Assigned Numbers». Например, для MAC-адреса Ethernet 10 Мбит/с это поле принимает значение 1. | 1 |
| hlen | Длина аппаратного адреса в байтах. Для MAC-адреса Ethernet — 6. | 1 |
| hops | Количество промежуточных маршрутизаторов (так называемых агентов ретрансляции DHCP), через которые прошло сообщение. Клиент устанавливает это поле в 0. | 1 |
| xid | Уникальный идентификатор транзакции, генерируемый клиентом в начале процесса получения адреса. | 4 |
| secs | Время в секундах с момента начала процесса получения адреса. Может не использоваться (в этом случае оно устанавливается в 0). | 2 |
| flags | Поле для флагов — специальных параметров протокола DHCP. | 2 |
| ciaddr | IP-адрес клиента. Заполняется только в том случае, если клиент уже имеет собственный IP-адрес и способен отвечать на запросы ARP (это возможно, если клиент выполняет процедуру обновления адреса по истечении срока аренды). | 4 |
| yiaddr | 'your' (client) IP address (RFC 2131) | 4 |
| siaddr | IP-адрес сервера. Возвращается в предложении DHCP (см. ниже). | 4 |
| giaddr | IP-адрес агента ретрансляции, если таковой участвовал в процессе доставки сообщения DHCP до сервера. | 4 |
| chaddr | Аппаратный адрес (обычно MAC-адрес) клиента. | 16 |
| sname | Необязательное имя сервера в виде нуль-терминированной строки. | 64 |
| file | Необязательное имя файла на сервере, используемое бездисковыми рабочими станциями при удалённой загрузке. Как и sname, представлено в виде нуль-терминированной строки. | 128 |
| options | Поле опций DHCP. Здесь указываются различные дополнительные параметры конфигурации. В начале этого поля указываются четыре особых байта со значениями 99, 130, 83, 99 («волшебные числа»), позволяющие серверу определить наличие этого поля. | переменная |

**Вид сообщений**

Отправляемых в процессе сообщений DHCP.Обнаружение DHCP



**Распределение IP-адресов**

Протокол DHCP поддерживает три механизма выделения адресов: автоматический, динамический и ручной.

Ручное распределение. При этом способе сетевой администратор сопоставляет аппаратному адресу (обычно MAC-адресу) каждого клиентского компьютера определённый IP-адрес. Фактически, данный способ распределения адресов отличается от ручной настройки каждого компьютера лишь тем, что сведения об адресах хранятся централизованно (на сервере DHCP), и потому их проще изменять при необходимости.

Автоматическое распределение. При данном способе каждому компьютеру на постоянное использование выделяется произвольный свободный IP-адрес из определённого администратором диапазона.

Динамическое распределение. Этот способ аналогичен автоматическому распределению, за исключением того, что адрес выдаётся компьютеру не на постоянное пользование, а на определённый срок. Это называется арендой адреса. По истечении срока аренды IP-адрес вновь считается свободным, и клиент обязан запросить новый (он, впрочем, может оказаться тем же самым).

Некоторые реализации службы DHCP способны автоматически обновлять записи DNS, соответствующие клиентским компьютерам, при выделении им новых адресов. Это производится при помощи протокола обновления DNS, описанного в RFC 2136.

Выдача адреса в аренду производится по запросу клиента. DHCP-сервер (или группа серверов) гарантирует, что выделенный адрес до истечения срока его аренды не будет выдан другому клиенту; при повторных обращениях сервер старается предложить клиенту адрес, которым тот пользовался ранее. Со своей стороны, клиент может запросить пролонгацию срока аренды адреса либо, наоборот, досрочно отказаться от него. Протоколом предусмотрена также выдача IP-адреса в неограниченное пользование. При острой нехватке адресов сервер может сократить срок аренды адреса по сравнению с запрошенным.

Выдача нового адреса. Последовательность событий в этом случае такова (рис. 2).

Рассмотрим пример процесса получения IP-адреса клиентом от сервера DHCP. Предположим, клиент ещё не имеет собственного IP-адреса, но ему известен его предыдущий адрес — 192.168.1.100.

1. Клиент посылает в собственную физическую подсеть широковещательное сообщение DHCPDISCOVER, в котором могут указываться устраивающие клиента IP-адрес и срок его аренды.

при этом в качестве IP-адреса источника указывается 0.0.0.0 (так как компьютер ещё не имеет собственного IP-адреса), а в качестве адреса назначения — широковещательный адрес 255.255.255.255.

Если в данной подсети DHCP-сервер отсутствует, сообщение будет передано в другие подсети ретранслирующими агентами протокола BOOTP (они же вернут клиенту ответные сообщения сервера).

Клиент заполняет несколько полей сообщения начальными значениями:

В поле xid помещается уникальный идентификатор транзакции, который позволяет отличать данный процесс получения IP-адреса от других, протекающих в то же время.

В поле chaddr помещается аппаратный адрес (MAC-адрес) клиента.

В поле опций указывается последний известный клиенту IP-адрес. В данном примере это 192.168.1.100. Это необязательно и может быть проигнорировано сервером.

2. Любой из DHCP-серверов может ответить на поступившее сообщение DHCPDISCOVER сообщением DHCPOFFER, включив в него доступный IP-адрес (yiaddr) и, если требуется, параметры конфигурации клиента. На этой стадии сервер не обязан резервировать указанный адрес. В принципе, он имеет право предложить его другому клиенту, также отправившему запрос DHCPDISCOVER. Тем не менее спецификации RFC 2131 рекомендуют серверу без необходимости не применять подобную тактику, а кроме того, убедиться (например, выдав эхо-запрос ICMP) в том, что предложенный адрес в текущий момент не используется каким-либо из компьютеров сети.

3. Клиент не обязан реагировать на первое же поступившее предложение. Допускается, чтобы он дождался откликов от нескольких серверов и, остановившись на одном из предложений, отправил в сеть широковещательное сообщение DHCPREQUEST. В нем содержатся идентификатор выбранного сервера и, возможно, желательные значения запрашиваемых параметров конфигурации.

Не исключено, что клиента не устроит ни одно из серверных предложений. Тогда вместо DHCPREQUEST он снова выдаст в сеть запрос DHCPDISCOVER, а серверы так и не узнают, что их предложения отклонены. Именно по этой причине сервер не обязан резервировать помещенный в DHCPOFFER адрес.

Если в процессе ожидания серверных откликов на DHCPDISCOVER достигнут тайм-аут, клиент выдает данное сообщение повторно.

4. Присутствующий в сообщении DHCPREQUEST идентификатор позволяет соответствующему DHCP-серверу убедиться в том, что клиент принял именно его предложение. В ответ сервер отправляет подтверждение DHCPACK, содержащее значения требуемых параметров конфигурации, и производит соответствующую запись в базу данных.

Если к моменту поступления сообщения DHCPREQUEST предложенный адрес уже <ушел> к другому клиенту (например, первая станция слишком долго <размышляла> над поступившими предложениями), сервер отвечает сообщением DHCPNACK.

5. Получив сообщение DHCPACK, клиент обязан убедиться в уникальности IP-адреса (средствами протокола ARP) и зафиксировать суммарный срок его аренды. Последний рассчитывается как время, прошедшее между отправкой сообщения DHCPREQUEST и приемом ответного сообщения DHCPACK, плюс срок аренды, указанный в DHCPACK.

Обнаружив, что адрес уже используется другой станцией, клиент обязан отправить серверу сообщение DHCPDECLINE и не ранее чем через 10 с начать всю процедуру снова. Процесс конфигурирования возобновляется и при получении серверного сообщения DHCPNACK.

При достижении тайм-аута в процессе ожидания серверных откликов на сообщение DHCPREQUEST клиент выдает его повторно.

6. Для досрочного прекращения аренды адреса клиент отправляет серверу сообщение DHCPRELEASE.

Приведенная последовательность действий заметно упрощается, если станция-клиент желает повторно работать с IP-адресом, который когда-то уже был ей выделен. В этом случае первым отправляемым сообщением является DHCPREQUEST, в котором клиент указывает прежде использовавшийся адрес. В ответ он может получить сообщение DHCPACK или DHCPNACK (если адрес занят либо клиентский запрос является некорректным, например из-за перемещения клиента в другую подсеть). Обязанность проверить уникальность IP-адреса опять-таки возлагается на клиента.

Выбор адреса DHCP-сервером. Если на момент получения запроса DHCPDISCOVER сервер не располагает свободными IP-адресами, он может направить уведомление о возникшей проблеме администратору. В противном случае при выборе адреса обычно применяется следующий алгоритм. Клиенту выделяется адрес, записанный за ним в данный момент. Если это невозможно, сервер предложит адрес, которым пользовался клиент до окончания срока последней аренды (при условии, что данный адрес свободен), либо адрес, запрошенный самим клиентом при помощи соответствующей опции (опять же, если адрес не занят). Наконец, в том случае, когда все предыдущие варианты не проходят, новый адрес выбирается из пула доступных адресов с учетом подсети, из которой поступил клиентский запрос.

Заметим, что исходя из определенной сетевым администратором политики сервер может выдать клиенту адрес, отличающийся от запрошенного (даже при доступности последнего), вообще отказать в предоставлении адреса или предложить адрес, относящийся к другой подсети. Более того, DHCP-сервер вообще не обязан реагировать на каждый поступивший запрос DHCPDISCOVER. Это предоставляет администратору возможность контролировать доступ к сети, например разрешив серверу отвечать только тем клиентам, которые предварительно зарегистрировались с помощью специальной процедуры.

**Истечение срока аренды.** По мере того как срок аренды подходит к концу, клиент может завершить работу с данным адресом, отправив на DHCP-сервер сообщение DHCPRELEASE, либо заблаговременно запросить продление срока аренды. В первом случае возвращение в сеть потребует выполнения всей процедуры инициализации заново. Во втором - станция продолжит функционировать в сети без видимого замедления работы пользовательских приложений.

При пролонгировании аренды клиент проходит два состояния - обновления адреса (RENEWING) и обновления конфигурации (REBINDING). Первое наступает примерно на половине срока аренды адреса (так называемый момент T1), второе - по истечении приблизительно 7/8 полного времени аренды (момент T2); для рассинхронизации процессов реконфигурирования разных клиентов значения этих временных меток рандомизируются с помощью случайной добавки.

В момент T1 клиент оправляет DHCP-серверу, выдавшему адрес, сообщение DHCPREQUEST с просьбой продлить срок аренды. Получив положительный ответ (DHCPACK), клиент пересчитывает срок аренды и продолжает работу в обычном режиме. Клиент ожидает прихода ответа от сервера в течение (T2 - t)/2 с (при условии, что это значение не меньше 60 с), где t - время отсылки последнего сообщения DHCPREQUEST, после чего отправляет данное сообщение повторно.

Если ответ от сервера не поступил к моменту T2, клиент переходит в состояние REBINDING и передает уже широковещательное сообщение DHCPREQUEST со своим текущим сетевым адресом. В этом случае моменты повторных выдач запросов DHCPREQUEST рассчитываются аналогично предыдущему случаю, только вместо T2 фигурирует время окончания срока аренды.

Не исключено, однако, что ответ DHCPACK не придет до окончания срока аренды. Тогда клиент обязан немедленно прекратить выполнение любых сетевых операций и заново начать процесс инициализации. Если запоздавший ответ DHCPACK все-таки поступит, клиенту рекомендуется сразу же возобновить работу под прежним адресом.

**Параметры конфигурации**

Хранение параметров сетевой конфигурации станций-клиентов является второй услугой, предоставляемой DHCP-сервером. В создаваемой базе данных на каждого клиента заводится отдельная запись с уникальным ключом-идентификатором и строкой конфигурационных параметров.

Роль идентификатора может играть пара <номер подсети IP, аппаратный адрес>, которая позволит использовать аппаратный адрес сразу в нескольких подсетях, либо пара <номер подсети IP, имя хост-компьютера>, позволяющая серверу взаимодействовать с клиентом, перемещенным в другую подсеть.

Что касается собственно параметров конфигурации, то их набор, поддерживаемый протоколом DHCP, определен в спецификациях RFC 1122, 1123, 1196 и 1256. В него входят выданный адрес, срок его аренды, назначавшиеся ранее адреса, а также максимальный размер реассемблируемого пакета, перечень фильтров для нелокальной маршрутизации от источника, адрес, используемый в широковещательных пакетах, параметры статических маршрутов и т.д. Впрочем, из всей совокупности допустимых параметров (а их более 30) в процессе инициализации могут передаваться только те, которые действительно необходимы для работы клиента либо определяются спецификой конкретной подсети.

Редукция объема передаваемых сведений о конфигурации достигается двумя способами. Во-первых, для большей части параметров в упомянутых выше документах RFC определены значения, принимаемые по умолчанию. Клиент будет использовать их, если в сообщении, поступившем от сервера, какие-то параметры опущены. Во-вторых, отправляя сообщение DHCPDISCOVER или DHCPREQUEST, клиентская станция может явно указать в нем параметры, значения которых она хотела бы получить.

Очевидно, что в обоих случаях передача параметров конфигурации осуществляется в ходе основной процедуры выделения IP-адреса. Возможен, однако, случай, когда клиент уже имеет IP-адрес (например, он был задан вручную). Тогда он может выдать сообщение DHCPINFORM\*, содержащее уже имеющийся адрес и запрос об отдельных параметрах конфигурации. Получив это сообщение, DHCP-сервер проверяет правильность адреса клиента (но не наличие аренды) и направляет ему сообщение DHCPACK с требуемыми параметрами конфигурации.

Отметим одно логическое противоречие, с которым связано применение протокола DHCP. Алгоритм выделения IP-адреса компьютеру сети предполагает, что установленное на нем программное обеспечение TCP/IP в состоянии воспринимать адресованные ему посредством <аппаратного> адреса IP-пакеты и транслировать их на IP-уровень еще до того, как станция получит свой IP-адрес, а сами средства TCP/IP будут полностью сконфигурированы. Такая возможность, очевидно, существует не всегда. Для работы с клиентами, не способными корректно обрабатывать одноадресные IP-дейтаграммы, используется поле flags. Такие клиенты должны установить первый бит данного поля в единичное значение, тем самым указав серверу на необходимость отправки в соответствующую подсеть только широковещательных сообщений.

**Недостатки DHCP**

Освобождая сетевых администраторов от множества рутинных операций, DHCP оставляет нерешенными ряд проблем, которые рано или поздно могут возникнуть в реальной сетевой среде.

К недостаткам этого протокола прежде всего следует отнести крайне низкий уровень информационной безопасности, что обусловлено непосредственным использованием протоколов UDP и IP. В настоящее время не существует практически никакой защиты от появления в сети несанкционированных DHCP-серверов, способных рассылать клиентам ошибочную или потенциально опасную информацию - некорректные или уже задействованные IP-адреса, неверные сведения о маршрутизации и т.д. И наоборот, клиенты, запущенные с неблаговидными целями, могут извлекать конфигурационные сведения, предназначенные для <законных> компьютеров сети, и тем самым оттягивать на себя значительную часть имеющихся ресурсов. Понятно, что возможности административного ограничения доступа, о которых говорилось выше, не способны закрыть эту брешь в системе информационной безопасности.

По мнению некоторых экспертов, в настоящее время DHCP недостаточно отказоустойчив. Протоколу явно недостает механизма активного уведомления клиентов об экстремальных ситуациях (например, о систематической нехватке адресов) и серверного подтверждения об освобождении адреса, иногда в сети наблюдаются всплески числа запросов на повторное использование адресов и т.д. Впрочем, работа над протоколом еще не завершена, и не исключено, что некоторые недостатки будут устранены в последующих редакциях.

**Опции DHCP**

Помимо IP-адреса, DHCP также может сообщать клиенту дополнительные параметры, необходимые для нормальной работы в сети. Эти параметры называются опциями DHCP. Список стандартных опций можно найти в RFC 2132.

Некоторыми из наиболее часто используемых опций являются:

IP-адрес маршрутизатора по умолчанию;

маска подсети;

адреса серверов DNS;

имя домена DNS.

Некоторые поставщики программного обеспечения могут определять собственные, дополнительные опции DHCP.

**Информация DHCP**

Сообщение информации DHCP (DHCPINFORM) предназначено для определения дополнительных параметров TCP/IP (например, адреса маршрутизатора по умолчанию, DNS-серверов и т. п.) теми клиентами, которым не нужен динамический IP-адрес (то есть адрес которых настроен вручную). Серверы отвечают на такой запрос сообщением подтверждения (DHCPACK) без выделения IP-адреса.

**Реализации**

Компания Microsoft впервые включила сервер DHCP в поставку серверной версии Windows NT 3.5, выпущенной в 1994 году. Начиная с Windows 2000 Server реализация DHCP-сервера от Microsoft позволяет динамически обновлять записи DNS, что используется в Active Directory.

Internet Systems Consortium выпустил первую версию ISC DHCP Server (для Unix-подобных систем) 6 декабря 1997 года. 22 июня 1999 года вышла версия 2.0, более точно соответствующая стандарту.

Компания Cisco включила сервер DHCP в Cisco IOS 12.0 в феврале 1999 года. Sun добавила DHCP-сервер в Solaris 8 в июле 2001 года.

Тем не менее, как частенько бывает в сетевой индустрии, механизмы DHCP уже реализованы в продуктах ряда производителей. К счастью, любые изменения в алгоритмах его работы легко учесть на программном уровне, так что, приобретая серверное или клиентское программное обеспечение определенной компании, можно не опасаться заточения в неприступную крепость патентованных решений. Скорее, следует обратить внимание на то, насколько удачно конкретная реализация DHCP вписывается в имеющуюся вычислительную среду и взаимодействует с другими сетевыми службами, в частности с DNS. Публикуемые в этом номере результаты сравнительных испытаний DNS- и DHCP-серверов (см. статью <Игра в имена>) способны стать неплохим подспорьем для пользователя.

**Распределение IP-адресов: возможны варианты**

BOOTP, DRARP, TFTP, ICMP, NIP - вероятно, это еще не полный перечень протоколов, в той или иной мере претендующих на решение задачи управления IP-адресами и конфигурацией в сетевой среде. Не исключено, что после стандартизации DHCP довольно быстро вытеснит их со сцены, однако на сегодняшний день многие из названных протоколов нередко упоминаются в литературе и используются в готовых продуктах.

Подобно DHCP, протокол Bootstrap Protocol (BOOTP) сегодня также имеет статус предварительного стандарта и рекомендован к применению консорциумом IETF. Именно его следует считать непосредственным предшественником DHCP. Появившиеся в 1993 г. дополнения расширили перечень конфигурационных параметров, охватываемых протоколом BOOTP. Более того, к настоящему времени даже предложена модификация BOOTP, поддерживающая динамическое назначение IP-адресов.

Протокол Reverse Address Resolution Protocol (RARP), впервые описанный в июне 1984 г. (RFC 903), используется компанией Sun Microsystems и рядом других производителей, в частности, для запуска бездисковых рабочих станций. Основной принцип его работы сводится к тому, что клиент должен самостоятельно отыскать свой IP-адрес, а не принять адрес, выделенный сервером. Сравнительно недавно появился динамический вариант этого протокола (Dynamic RARP, DRARP), реализующий алгоритм автоматического присвоения IP-адресов. Стоит отметить, что изначальный вариант RARP не поддерживает передачу клиенту каких-либо параметров конфигурации (кроме IP-адреса), в том числе применяемых при маршрутизации. В результате один сервер RARP способен обслуживать только одну локальную сеть.

Упрощенный вариант FTP - протокол Trivial File Transfer Protocol (TFTP) - увидел свет около 20 лет назад, его вторая версия описана в документе RFC 783. Фактически TFTP предоставляет транспортный механизм для передачи с сервера загрузочного образа клиентской системы.

Протокол Internet Control Message Protocol (ICMP) сегодня также можно отнести к поколению ветеранов Всемирной сети. Он позволяет информировать компьютеры о наличии в сети дополнительных маршрутизаторов (имеется специальный механизм для обнаружения этих устройств), предусматривает специальные типы сообщений для передачи маски подсети и других служебных сведений.

Наконец, протокол Network Information Protocol (NIP) был разработан в конце 80-х гг. сотрудниками Массачусетского технологического института в качестве распределенного механизма для динамического присвоения IP-адресов в сетях Ethernet.

Отметим еще спецификации RFC 1122 и 1123, которые используются рядом протоколов (в том числе DHCP). Они содержат требования к процедуре изменения конфигурации компьютеров сети и, кроме того, предлагают сценарий первоначального конфигурирования бездисковых станций.

# DNS (Domain Name System)

1. **Протокол DNS** относится к протоколам стека TCP/IP (прикладной уровень).

Описан в RFC 1034 (Concepts and Facilities), 1035(implementation and specification) или STD 13.

**Служба DNS (домены, зоны; зоны прямого и обратного просмотра; основные и дополнительные зоны; рекурсивный и итеративный запросы на разрешение имен).**

***Историческая справка:*** *Систему доменных имен разработал в 1983 году Пол Мокапетрис. Тогда же было проведено первое успешное тестирование DNS, ставшей позже одним из базовых компонентов сети Internet. С помощью DNS стало возможным реализовать масштабируемый распределенный механизм, устанавливающий соответствие между иерархическими именами сайтов и числовыми IP-адресами.*

*В 1983 году Пол Мокапетрис работал научным сотрудником института информатики (Information Sciences Institute, ISI), входящего в состав инженерной школы университета Южной Калифорнии (USC). Его руководитель, Джон Постел, предложил Полу придумать новый механизм, устанавливающий связи между именами компьютеров и адресами Internet, — взамен использовавшемуся тогда централизованному каталогу имен и адресов хостов, который поддерживала калифорнийская компания SRI International.*

*«Все понимали, что старая схема не сможет работать вечно, — вспоминает Мокапетрис. — Рост Internet становился лавинообразным. К сети, возникшей на основе проекта ARPANET, инициированного Пентагоном, присоединялись все новые и новые компании и исследовательские институты».*

*Предложенное Мокапетрисом решение — DNS — представляло собой распределенную базу данных, которая позволяла организациям, присоединившимся к Internet, получить свой домен.*

*«Как только организация подключалась к сети, она могла использовать сколь угодно много компьютеров и сама назначать им имена», — подчеркнул Мокапетрис. Названия доменов компаний получили суффикс .com, университетов — .edu и так далее.*

*Первоначально DNS была рассчитана на поддержку 50 млн. записей и допускала безопасное расширение до нескольких сотен миллионов записей. По оценкам Мокапетриса, сейчас насчитывается около 1 млрд. имен DNS, в том числе почти 20 млн. общедоступных имен. Остальные принадлежат системам, расположенным за межсетевыми экранами. Их имена неизвестны обычным Internet-пользователям.*

*Новая система внедрялась постепенно, в течение нескольких лет. В это время ряд исследователей экспериментировали с ее возможностями, а Мокапетрис занимался в ISI обслуживанием и поддержанием стабильной работы «корневого сервера», построенного на мэйнфреймах компании Digital Equipment. Копии таблиц хостов хранились на каждом компьютере, подключенном к Internet, еще примерно до 1986 года. Затем начался массовый переход на использование DNS.*

*Необходимость отображения имен сетевых узлов в IP-адреса*

Компьютеры и другие сетевые устройства, отправляя друг другу пакеты по сети, используют IP-адреса.. Однако пользователю (человеку) гораздо проще и удобнее запомнить некоторое символические имена сетевых узлов, чем четыре бессодержательных для него числа. Однако, если люди в своих операциях с сетевыми ресурсами будут использовать имена узлов, а не IP-адреса, тогда должен существовать механизм, сопоставляющий именам узлов их IP-адреса.

Есть два таких механизма — локальный для каждого компьютера файл *hosts* и централизованная иерархическая служба имен *DNS*.

1. **ICANN** (Internet Corporation for Assign Names and Numbers) – организация, обеспечивающая управление всем адресным пространством Internet.

*Использование локального файла hosts и системы доменных имен DNS для разрешения имен сетевых узлов*

На начальном этапе развития сетей, когда количество узлов в каждой сети было небольшое, достаточно было на каждом компьютере хранить и поддерживать актуальное состояние простого текстового файла, в котором содержался список сетевых узлов данной сети. Список устроен очень просто — в каждой строке текстового файла содержится пара «IP-адрес — имя сетевого узла». В системах семейства Windows данный файл расположен в папке “*%system root%\system32\drivers\etc*” (где *%system root%* обозначает папку, в которой установлена операционная система). Сразу после установки системы Windows создается файл hosts с одной записью «127.0.0.1 localhost».

С ростом сетей поддерживать актуальность и точность информации в файл hosts становится все труднее. Для этого надо постоянно обновляя содержимое этого файла на всех узлах сети. Кроме того, такая простая технология не позволяет организовать пространство имен в какую-либо структуру. Поэтому появилась необходимость в централизованной базе данных имен, позволяющей производить преобразование имен в IP-адреса без хранения списка соответствия на каждом компьютере. Такой базой стала DNS (Domain Name System) — система именования доменов, которая начала массовую работу в 1987 году.

Заметим, что с появлением службы DNS актуальность использования файла host совсем не исчезла, в ряде случаев использование этого файла оказывается очень эффективным.

*Служба DNS: пространство имен, домены*

DNS — это иерархическая база данных, сопоставляющая имена сетевых узлов и их сетевых служб IP-адресам узлов. Содержимое этой базы, с одной стороны, распределено по большому количеству серверов службы DNS, а с другой стороны, является централизованно управляемым. В основе иерархической структуры базы данных DNS лежит *доменное пространство имен* (*domain namespace*), основной структурной единицей которого является домен, объединяющий сетевые узлы (хосты), а также поддомены. Процесс поиска в БД службы DNS имени некого сетевого узла и сопоставления этому имени IP-адреса называется «разрешением имени узла в пространстве имен DNS».

Служба DNS состоит из трех основных компонент:

* **Пространство имен DNS и соответствующие ресурсные записи (RR, resource record)** — это сама распределенная база данных DNS;
* **Серверы имен DNS** — компьютеры, хранящие базу данных DNS и отвечающие на запросы DNS-клиентов;
* **DNS-клиенты (DNS-clients, DNS-resolvers)** —компьютеры, посылающие запросы серверам DNS для получения ресурсных записей.

Пространство имен.

Пространство имен DNS — иерархическая древовидная структура, начинающаяся с корня, не имеющего имени и обозначаемого точкой “**.**”. Схему построения пространства имен DNS лучше всего проиллюстрировать на примере сети Интернет (Рис. 3.8).

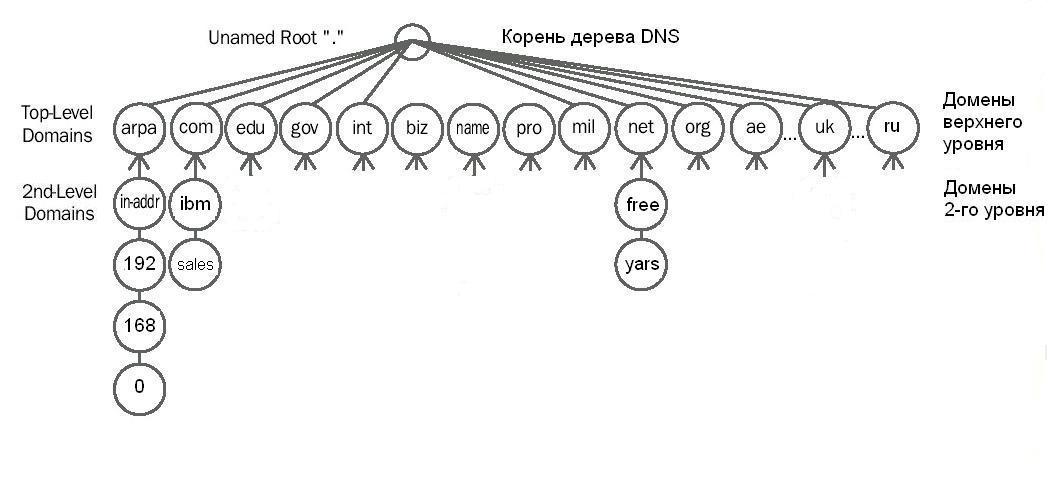


Рис 3.8

Для доменов 1-го уровня различают 3 категории имен:

* **ARPA** — специальное имя, используемое для обратного разрешения DNS (из IP-адреса в полное имя узла);
* **Общие (generic) имена 1-го уровня** — 16 (на данный момент) имен, назначение которых приведено в таблице 3.4;
* **Двухбуквенные имена для стран** — имена для доменов, зарегистрированных в соответствующих странах (например, *ru* — для России, *ua* — для Украины, *uk* — для Великобритании и т.д.).

|  |  |
| --- | --- |
| **Имя домена** | **Назначение** |
| aero | Сообщества авиаторов |
| biz | Компании (без привязки к стране) |
| com | Коммерческие организации, преимущественно в США (например, домен *microsoft.com* для корпорации Microsoft) |
| coop | Кооперативы |
| edu | Образовательные учреждения в США |
| gov | Правительственные учреждения США |
| info | Домен для организаций, предоставляющих любую информацию для потребителей |
| int | международные организации (например, домен *nato.int* для НАТО) |
| mil | Военные ведомства США |
| museum | Музеи |
| name | Глобальный домен для частных лиц |
| net | Домен для Интернет-провайдеров и других организаций, управляющих структурой сети Интернет |
| org | Некоммерческие и неправительственные организации, преимущественно в США |
| pro | Домен для профессиональных объединений (врачей, юристов, бухгалтеров и др.) |
| job | Кадровые агентства |
| travel | Туроператоры |

Корневые серверы DNS — DNS-серверы, содержащие информацию о доменах верхнего уровня, указывающую на DNS-серверы, поддерживающие работу каждого из этих доменов. Основные корневые серверы DNS размещены в домене root-servers.net обозначаются латинскими буквами от A до М. Они управляются различными организациями, действующими по согласованию с ICANN. Количество серверов ограничено в связи с максимальным объёмом UDP-пакета (большее количество серверов потребовало бы перехода на TCP-протокол для получения ответа, что существенно бы увеличило нагрузку).

Хотя из-за ограничения на размеры DNS-пакета (512 байт) в DNS-ответ может быть помещено всего 13 серверов, на самом деле сейчас за этими 13 «виртуальными» серверами стоят 190 серверов (зеркала).

По разным оценкам только от 18 до 32 % разрешений доменных имен приводит к обращению непосредственно к одному из корневых серверов, остальные запросы используют кэшированные DNS-записи о TLD NS.

Домен верхнего (первого) уровня (англ. top-level domain — TLD) — в иерархии системы доменнных имён (DNS) самый высокий уровень после корневого домена (англ. root domain). Является начальной точкой отсчёта (справа налево), с которой начинается доменное имя в Интернете.

Полное имя узла (FQDN, fully qualified domain name) состоит из нескольких имен, называемых метками (label) и разделенных точкой. Самая левая метка относится непосредственно к узлу, остальные метки — список доменов от домена первого уровня до того домена, в котором находится узел (данный список просматривается справа налево).

FQDN (сокр. от англ. Fully Qualified Domain Name, полностью определённое имя домена, иногда сокращается до «полное имя») — имя домена, не имеющее неоднозначностей в определении. Включает в себя имена всех родительских доменов иерархии DNS.

В DNS и, что особенно существенно, в файлах зоны(англ.), FQDN завершаются точкой (например, example.com.), то есть включают корневое доменное имя, которое является безымянным.

Максимальный размер FQDN — 255 байт, с ограничением в 63 байта на каждое имя домена.

Серверы имен DNS.

Серверы имен DNS (или DNS-серверы) — это компьютеры, на которых хранятся те части БД пространства имен DNS, за которые данные серверы отвечают, и функционирует программное обеспечение, которое обрабатывает запросы DNS-клиентов на разрешение имен и выдает ответы на полученные запросы.

DNS-клиенты.

DNS-клиент — это любой сетевой узел, который обратился к DNS-серверу для разрешения имени узла в IP-адрес или, обратно, IP-адреса в имя узла.

*Служба DNS: домены и зоны*

Как уже говорилось выше, каждый DNS-сервер отвечает за обслуживание определенной части пространства имен DNS. Информация о доменах, хранящаяся в БД сервера DNS, организуется в особые единицы, называемые *зонами* (*zones*). Зона — основная единица репликации данных между серверами DNS. Каждая зона содержит определенное количество ресурсных записей для соответствующего домена и, быть может, его поддоменов.

Системы семейства Windows Server поддерживают следующие типы зон:

* **Стандартная основная** (**standard primary**) — главная копия стандартной зоны; только в данном экземпляре зоны допускается производить какие-либо изменения, которые затем реплицируются на серверы, хранящие дополнительные зоны;
* **Стандартная дополнительная** (**standard secondary**) — копия основной зоны, доступная в режиме «только-чтение», предназначена для повышения отказоустойчивости и распределения нагрузки между серверами, отвечающими за определенную зону; процесс репликации изменений в записях зон называется «*передачей зоны*» (*zone transfer*)  
  (информация в стандартных зонах хранится в тестовых файлах, файлы создаются в папке *«%system root%\system32\dns*», имя файла, как правило, образуется из имени зоны с добавлением расширения файла «*.dns*»; термин «стандартная» используется только в системах семейства Windows);
* **Интегрированная в Active Directory** (**Active Directory–integrated**) — вся информация о зоне хранится в виде одной записи в базе данных Active Directory (такие типы зон могут существовать только на серверах Windows, являющихся контроллерами доменов Active Directory; в интегрированных зонах можно более жестко управлять правами доступа к записям зоны; изменения в записях зоны между разными экземплярами интегрированной зоны производятся не по технологии передачи зоны службой DNS, а механизмами репликации службы Active Directory);
* **Зона-заглушка** (**stub**; только в Windows 2003) — особый тип зоны, которая для данной части пространства имен DNS содержит самый минимальный набор ресурсных записей (начальная запись зоны SOA, список серверов имен, отвечающих за данную зону, и несколько записей типа A для ссылок на серверы имен для данной зоны).

Рассмотрим на примере соотношение между понятиями *домена* и *зоны*. Проанализируем информацию, представленную на рис. 3.9.

**microsoft.com**

**sales.microsoft.com**

**it.microsoft.com**

**edu.microsoft.com**

**Файл:**

**“sales.microsoft.com.dns”**

**Файл:**

**“it.microsoft.com.dns”**

**Файл:**

**“microsoft.com.dns”**

Рис. 3.9

В данном примере пространство имен DNS начинается с домена *microsoft.com*, который содержит 3 поддомена: *sales.microsoft.com*, *it.microsoft.com* и *edu.microsoft.com* (домены на рисунке обозначены маленькими горизонтальными овалами). Домен — понятие чисто логическое, относящееся только к распределению имен — древовидная структура доменов и поддоменов и для каждого домена свой список узлов. Понятие домена никак не связано с технологией хранения информации о домене. Зона — это способ представления информации о домене и его поддоменах в хранилище тех серверов DNS, которые отвечают за данный домен и поддомены. В данной ситуации, если для хранения выбрана технология стандартных зон, то размещение информации о доменах может быть реализовано следующим образом:

* записи, относящиеся к доменам microsoft.com и edu.microsoft.com, хранятся в одной зоне в файле *«microsoft.com.dns»* (на рисунке зона обозначена большим наклонным овалом);
* управление доменами sales.microsoft.com и it.microsoft.com *делегировано* другим серверам DNS, для этих доменов на других серверах созданы соответствующие файлы *«sales.microsoft.com*.*dns*» и *«it.microsoft.com.dns»* (данные зоны обозначены большими вертикальными овалами).

Делегирование управления — передача ответственности за часть пространства имен другим серверам DNS.

**Первичный DNS-сервер**: сервер, который всегда может прочитать свой зонный файл. сервер, имеющий право на внесение изменений в данные зоны. Для каждой зоны может быть только один первичный сервер.

1. **Вторичный DNS**-**сервер:** сервер, получающий свои данные о первичного сервера. Он поддерживает свой зонный файл, но постоянно его обновляет. Для обновления использует запись SOA. Для каждой зоны, должен существовать вторичный сервер. Большинство реализаций обязывают вторичный сервер обновлять только изменения зоны первичного сервера. Вторичные сервера поддерживаю актуальную копию зонного файла.
2. **Кэширующий сервер:**  используется редко, в больших разветвленных организациях.

*Зоны прямого и обратного просмотра*

Зоны, рассмотренные в предыдущем примере, являются *зонами прямого просмотра* (*forward lookup zones*). Данные зоны служат для разрешения имен узлов в IP-адреса. Наиболее часто используемые для этого типы записей: A, CNAME, SRV.

Для определения имени узла по его IP-адресу служат *зоны обратного просмотра* (*reverse lookup zones*), основной тип записи в «обратных» зонах — PTR. Для решения данной задачи создан специальный домен. с именем «*in-addr.arpa*». Для каждой IP-сети в таком домене создаются соответствующие поддомены, образованные из идентификатора сети, записанного в обратном порядке. Записи в такой зоне будут сопоставлять идентификатору узла полное FQDN-имя данного узла. Например, для IP-сети *192.168.0.0/24* необходимо создать зону с именем «*0.168.192.in-addr.arpa*». Для узла с IP-адресом 192.168.0.10 и именем *host.company.ru* в данной зоне должна быть создана запись «10 PTR host.company.ru».

*Алгоритмы работы итеративных и рекурсивных запросов DNS*

Все запросы, отправляемые DNS-клиентом DNS-серверу для разрешения имен, делятся на два типа:

* итеративные запросы (клиент посылает серверу DNS запрос, в котором требует дать наилучший ответ без обращений к другим DNS-серверам);
* рекурсивные запросы (клиент посылает серверу DNS запрос, в котором требует дать окончательный ответ даже если DNS-серверу придется отправить запросы другим DNS-серверам; посылаемые в этом случае другим DNS-серверам запросы будут итеративными).

В основном DNS-клиентами используются рекурсивные запросы. На рис. 5.3 проиллюстрирован процесс разрешения доменного имени с помощью рекурсивного запроса.

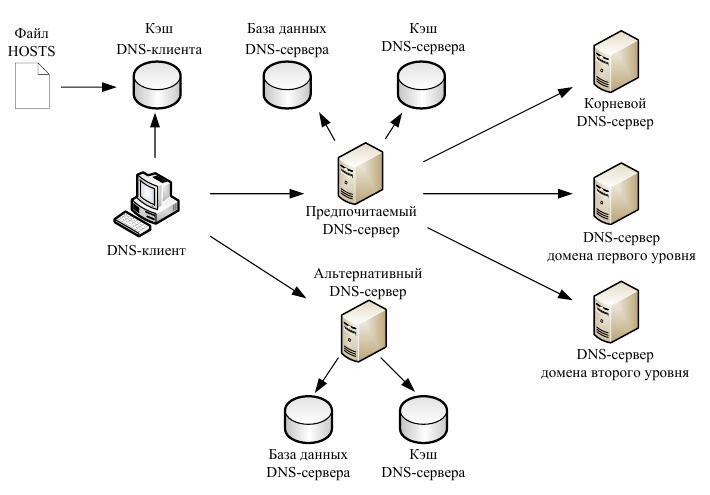


Рис. 5.3. Процесс обработки рекурсивного DNS-запроса

Сначала DNS-клиент осуществляет поиск в собственном локальном кэше DNS-имен. Это память для временного хранения ранее разрешенных запросов. В эту же память переносится содержимое файла HOSTS (каталог windows/system32/drivers/etc). Утилита IPconfig с ключом /displaydns отображает содержимое DNS-кэша. Если кэш не содержит требуемой информации, DNS-клиент обращается с рекурсивным запросом к предпочитаемому DNS-серверу (Preferred DNS server), адрес которого указывается при настройке стека TCP/IP. DNS-сервер просматривает собственную базу данных, а также кэш-память, в которой хранятся ответы на предыдущие запросы, отсутствующие в базе данных. В том случае, если запрашиваемое доменное имя не найдено, DNS-сервер осуществляет итеративные запросы к DNS-серверам верхних уровней, начиная с корневого DNS-сервера.

Рассмотрим на примерах, как происходит взаимодействие DNS-клиента и DNS-сервера при обработке итеративных и рекурсивных запросов.

Допустим, что пользователь запустил программу Обозреватель Интернета и ввел в адресной строке адрес «*http://www.microsoft.com*». Прежде чем Обозреватель установит сеанс связи с веб-сайтом по протоколу HTTP, клиентский компьютер должен определить IP-адрес веб-сервера. Для этого клиентская часть протокола TCP/IP рабочей станции пользователя (так называемый *resolver*) сначала просматривает свой локальный кэш разрешенных ранее имен в попытке найти там имя «*www.microsoft.com*». Если имя не найдено, то клиент посылает запрос DNS-серверу, указанному в конфигурации TCP/IP данного компьютера (назовем данный DNS-сервер «*локальным DNS-сервером*»), на разрешение имени «*www.microsoft.com*» в IP-адрес данного узла. Далее DNS-сервер обрабатывает запрос в зависимости от типа запроса.

Вариант 1 (итеративный запрос).

Если клиент отправил серверу итеративный запрос (напомним, что обычно клиенты посылают рекурсивные запросы), то обработка запроса происходит по следующей схеме:

* сначала локальный DNS-сервер ищет среди зон, за которые он отвечает, зону «*microsoft.com*»; если такая зона найдена, то в ней ищется запись для узла *www*; если запись найдена, то результат поиска сразу же возвращается клиенту;  
  в противном случае локальный DNS-сервер ищет запрошенное имя «*www.microsoft.com*» в своем кэше разрешенных ранее DNS-запросов; если искомое имя есть в кэше, то результат поиска возвращается клиенту;  
  если локальный DNS-сервер не нашел в своей базе данных искомую запись, то клиенту посылается IP-адрес одного из корневых серверов DNS;
* клиент получает IP-адрес корневого сервера и повторяет ему запрос на разрешение имени «*www.microsoft.com*»;  
  корневой сервер не содержит в своей БД зоны «*microsoft.com*», но ему известны DNS-серверы, отвечающие за зону «*com*», и корневой сервер посылает клиенту IP-адрес одного из серверов, отвечающих за эту зону;
* клиент получает IP-адрес сервера, отвечающего за зону «*com*», и посылает ему запрос на разрешение имени «*www.microsoft.com*»;  
  сервер, отвечающий за зону «*com*», не содержит в своей БД зоны «*microsoft.com*», но ему известны DNS-серверы, отвечающие за зону «*microsoft.com*», и данный DNS-сервер посылает клиенту IP-адрес одного из серверов, отвечающих уже за зону «*microsoft.com*»;
* клиент получает IP-адрес сервера, отвечающего за зону «*microsoft.com*», и посылает ему запрос на разрешение имени «*www.microsoft.com*»;  
  сервер, отвечающий за зону «*microsoft.com*», получает данный запрос, находит в своей базе данных IP-адрес узла *www*, расположенного в зоне *«microsoft.com»*, и посылает результат клиенту;  
  клиент получает искомый IP-адрес, сохраняет разрешенный запрос в своем локальном кэше и передает IP-адрес веб-сайта программе Обозреватель Интернета (после чего Обзреватель устанавливает связь с веб-сайтом по протоколу HTTP).

Вариант 2 (рекурсивный запрос).

Если клиент отправил серверу рекурсивный запрос, то обработка запроса происходит по такой схеме:

* сначала локальный DNS-сервер ищет среди зон, за которые он отвечает, зону «*microsoft.com*»; если такая зона найдена, то в ней ищется запись для узла *www*; если запись найдена, то результат поиска сразу же возвращается клиенту;  
  в противном случае локальный DNS-сервер ищет запрошенное имя «*www.microsoft.com*» в своем кэше разрешенных ранее DNS-запросов; если искомое имя есть в кэше, то результат поиска возвращается клиенту;
* если локальный DNS-сервер не нашел в своей базе данных искомую запись, то сам локальный DNS-сервер выполняет серию итеративных запросов на разрешение имени «*www.microsoft.com*», и клиенту посылается либо найденный IP-адрес, либо сообщение об ошибке.

*Реализация службы DNS в системах семейства Windows Server*

Главная особенность службы DNS в системах семейства Windows Server заключается в том, что служба DNS разрабатывалась для поддержки службы каталогов Active Directory. Для выполнения этой функции требуются обеспечение двух условий:

* поддержка службой DNS динамической регистрации (dynamic updates);
* поддержка службой DNS записей типа SRV.

Служба DNS систем Windows Server удовлетворяет обоим условиям, и реализация служб каталогов Active Directory может быть обеспечена только серверами на базе систем Windows Server.

Все рассматриваемые далее в пособии примеры были выполнены в следующей конфигурации:

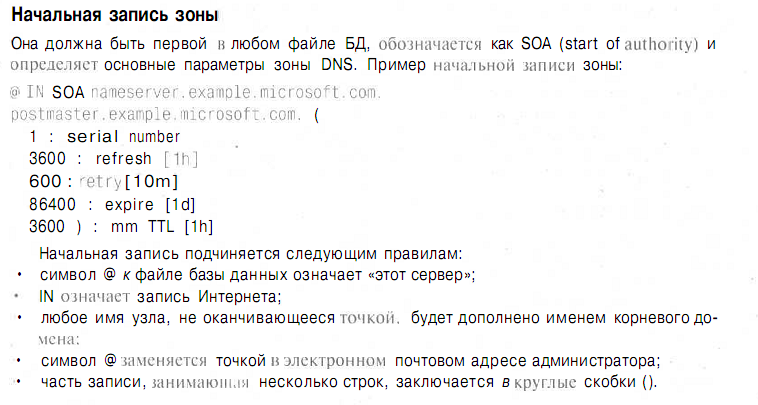
* сеть состоит из двух серверов Windows 2003 Server;
* операционная система — ограниченная по времени 120-дневная русская версия Windows 2003 Server Enterprise Edition;
* первый сервер установлен на ПК с процессором Intel Pentium-4 3Ггц и оперативной памятью 512 МБ, имя сервера — DC1, IP-адрес — 192.168.0.1/24;
* второй сервер работает в качестве виртуальной системы с помощью Microsoft VirtualPC 2004 (данный продукт в настоящее время доступен для свободной загрузки на сайте корпорации Microsoft), имя сервера — DC2, IP-адрес — 192.168.0.2/24;
* имя домена в пространстве DNS и соответствующее имя в службе каталогов Active Directory — world.ru (сеть полностью изолирована от других сетей, поэтому в данном примере авторы были свободны в выборе имени домена; в реальной обстановке конкретного учебного заведения преподавателю нужно скорректировать данную информацию).

Для непосредственного отображения пространства имен в пространство IP-адресов служат т.н. ресурсные записи (RR, resource record). Каждый сервер DNS содержит ресурсные записи для той части пространства имен, за которую он несет ответственность (authoritative). Таблица 3.5 содержит описание наиболее часто используемых типов ресурсных записей.

|  |  |  |
| --- | --- | --- |
| **Тип ресурсной записи** | **Функция записи** | **Описание испоьзования** |
| A | Host Address  Адрес хоста, или узла | Отображает имя узла на IP-адрес  (например, для домена *microsoft.com* узлу с именем *www.microsoft.com* сопоставляется IP-адрес с помощью такой записи:  www A 207.46.199.60) |
| CNAME | Canonical Name (alias)  Какноническое имя (псевдоним) | Отображает одно имя на другое |
| MX | Mail Exchanger  Обмен почтой | Управляет маршрутизация почтовых сообщений для протокола SMTP |
| NS | Name Server  Сервер имен | Указывает на серверы DNS, ответственные за конкретный домен и его поддомены |
| PTR | Pointer  Указатель | Используется для обратного разрешения IP-адресов в имена узлов в домене *in-addr.arpa* |
| SOA | Start of Authority  Начальная запись зоны | Используется для указания основного сервера для данной зоны и описания свойств зоны |
| SRV | Service Locator  Указатель на службу | Используется для поиска серверов, на которых функционируют определенные службы (например, контроллеры доменов Active Directory или серверы глобального каталога) |

Таблица 3.5

1. **Запись SOA:** ***имя домена зоны***, ***почтовый адрес администратора***, ***порядок*** – число, увеличивается при изменении зоны; ***период обновления***  (для вторичных серверов); ***задержка перед следующей попыткой обновления*** (после неудачной попытки); ***интервал времени***, через который сервер утратит статус ответственного после обновления; ***минимальное время жизни*** показывает другим (кэширующим) серверам, как долго они могут пользоваться данными данного сервера.



1. **Запись A:**  связывает доменное имя и ip-адрес. Если создать несколько записей с одним и тем же доменным именем но разными ip-адресами, то это позволяет ***равномерно*** распределить нагрузку между несколькими хостами (***карусель***). Сервер выбирает ***ближайший*** ip-адрес к клиенту.

A-запись

Запись типа A позволяет установить соответствие между именем хоста в домене и его IP-адресом.

Запись типа A имеет следующий формат:

имя\_хоста [TTL] A IP-адрес

Имя\_хоста: доменное имя хоста (устройства), подключенного к Интернету, для которого данная запись определяет соответствие с его IP-адресом.

А: тип записи

IP-адрес: IP-адрес хоста

Примеры A-записи для хоста info.test.ru в файле зоны test.ru:

info A 194.85.61.44

или

info.test.ru. 86400 A 194.85.61.44

1. **Запись CNAME**: позволяет одному ip-адресу сопоставить несколько доменных имен.

CNAME-запись

Запись типа CNAME (Canonical Name - Каноническое имя) позволяют присваивать хосту мнемонические имена. Мнемонические имена, или псевдонимы, широко применяются для связывания с хостом какой-либо функции, либо просто для сокращения имени.

Реальное имя иногда называют каноническим.

Если для хоста есть запись типа CNAME, которая содержит его мнемонические имена, другие записи для данного хоста должны ссылаться на его реальное (каноническое) имя, а не на мнемоническое. Когда программы DNS встречают запись CNAME, они прекращают свои запросы по мнемоническому имени и переключаются на реальное имя.

Кроме того, если данное имя использовано в качестве псевдонима, то на него нельзя занести записи любого другого типа.

Т.е. недопустима конструкция вида:

domain CNAME имя\_хоста

domain MX 10 почтовый сервер

Мнемоимена полезны, например, в случае, когда имя хоста изменилось и вы хотите разрешить пользователям, знающим старое имя, получить доступ к хосту.

Запись типа CNAME имеет следующий формат:

Мнемоимя [TTL] CNAME имя\_хоста

TTL: см. описание параметра TTL

CNAME: тип записи

имя\_хоста: каноническое имя хоста

Примеры CNAME-записей для хоста arhive.test.ru в домене test.ru.:

ftp.test.ru. CNAME arhive.test.ru.

gopher.test.ru. CNAME arhive.test.ru.

или

ftp.test.ru. 86400 CNAME arhive.test.ru.

gopher.test.ru. 172800 CNAME arhive.test.ru.

1. **Запись MX**: связывает имя почтового домена с именем хоста, содержащего почтовый сервер. Можно указать несколько MX-записей с указанием приоритета.

MX-запись

Запись типа MX (Mail Exchange - почтовый сервер) определяет почтовый сервер - машину, которая обрабатывает почту для вашего домена.

Запись типа MX имеет следующий формат:

ваш\_домен [TTL] MX приоритет почтовый сервер

TTL: см. описание параметра TTL

MX: тип записи

Приоритет: определяет значение приоритетности почтового сервера. Чем меньше число, тем выше приоритет почтового сервера (0 означает самый высокий приоритет, 65535 - самый низкий). Таким образом, почтовый сервер с более высоким приоритетом является основным, а почтовые серверы с более низкими приоритетами будут второстепенным и вступят в работу в том случае, если все более приоритетные серверы по каким-либо причинам недоступны или неработоспособны.

почтовый сервер: имя почтового сервера

Примеры MX-записей для домена test.ru:

test.ru. MX 10 relay2.test.ru.

test.ru. MX 20 relay3.test.ru.

или

test.ru. 86400 MX 10 relay2.test.ru.

test.ru. 86400 MX 20 relay3.test.ru.

Обращаем ваше внимание на то, что у всех записей типа MX, относящихся к одному домену, значение TTL должно быть одинаковым.

Таким образом, почтовый сервер relay2.test.ru является основным, а relay3.test.ru является второстепенным почтовым сервером и вступает в работу в том случае, если relay2.test.ru по каким-либо причинам недоступен или неработоспособен.

1. **Запись PTR**: запись обратная записи A (зона обратного просмотра).

PTR-запись

Записи типа PTR (Pointer - указатель) служат для выполнения обратного преобразования IP-адресов в имена хостов. Для каждого сетевого интерфейса хоста рекомендуется создать запись PTR.

Записи типа PTR, как правило, имеет смысл вносить только в обратные зоны.

Если провайдер выделил вам несколько IP-адресов из своей сети, то по поводу записей в обратной зоне вам следует обращаться к нему.

Запись типа PTR имеет следующий формат:

адрес [TTL] PTR имя\_хоста

адрес: преобразованный IP-адрес хоста

TTL: см. описание параметра TTL

PTR: тип записи

Примеры PTR-записей

Если провайдер выделил вам IP-адрес 194.85.61.42 в сети 194.85.61.0/24, то запись о вашем хосте (например, www.mydomain.ru) будет сделана провайдером в обратной зоне 61.85.194.in-addr.arpa. Запись будет иметь вид:

42 PTR www.mydomain.ru.

или

42 86400 PTR [www.mydomain.ru](http://www.mydomain.ru).

1. **Запись NS:**  определяет полномочный сервер имен субдомена. Связывает имя субдомена с именем полномочного dns-сервера.

NS-запись

Записи типа NS (Name Server - cервер имен) описывают DNS-серверы для данного домена. Количество записей типа NS в файле зоны должно точно соответствовать количеству DNS-серверов, обслуживающих домен и включать все DNS-серверы, указанные в домене. Для доменов второго уровня это DNS-серверы, указанные в полях "nserver" в информации по домену, которую можно получить, используя сервис Whois (https://www.nic.ru/whois/)

Запись типа NS имеет следующий формат:

доменное\_имя [TTL] NS имя\_хоста

TTL: см. описание параметра TTL

NS: тип записи

имя\_хоста: доменное имя DNS-сервера

Примеры NS-записей в домене test.ru. Указаны DNS-серверы, обслуживающие домен test.ru и домен третьего уровня mf в домене test.ru (mf.test.ru)

Для домена test.ru:

test.ru. NS ns3.test.ru.

test.ru. NS ns4.test.ru.

или

test.ru. 86400 NS ns3.test.ru.

test.ru. 86400 NS ns4.test.ru.

Для домена mf.test.ru:

mf.test.ru. NS ns3.test.ru.

mf.test.ru. NS ns4.test.ru.

или

mf.test.ru. 86400 NS ns3.test.ru.

mf.test.ru. 86400 NS ns4.test.ru.

Обращаем ваше внимание на то, что у всех записей типа NS, относящихся к одному доменному имени, значение TTL должно быть одинаковым.

1. **Запись SRV:** отражают порядок и доступность служб в конкретной зоне. Определяет сетевой протокол (\_kerberos, \_ldap, \_gc, \_kpasswd) , порт, доменное имя хоста.

SRV-запись

Записи типа SRV используются для поиска серверов, обеспечивающих работу тех или иных служб в данном домене.

С подробным описанием этого типа записей вы можете ознакомиться в RFC-2782.

Запись типа SRV имеет следующий формат:

\_Service.\_Proto.Name [TTL] SRV Priority Weight Port Target

Service: название службы (пример: ldap, kerberos, gc и другие)

Proto: протокол, при помощи которого клиенты могут подключиться к данной службе (пример: tcp, udp)

Name: имя домена, в котором размещена данная служба

TTL: см. описание параметра TTL

SRV: тип записи

Priority: приоритет данного сервера. Чем меньше число, тем выше приоритет (0 означает самый высокий приоритет, 65535 - самый низкий)

Weight: относительный вес для серверов с одинаковым приоритетом. Предназначен для распределения нагрузки между серверами, для которых указан равный приоритет

Port: порт, на котором размещена указанная служба на данном сервере

Target: доменное имя сервера, предоставляющего данную службу

Примеры SRV-записей

\_foobar.\_tcp SRV 0 1 9 old-slow-box.example.com.

или

\_foobar.\_tcp 86400 SRV 0 3 9 new-fast-box.example.com.

1. **Типы запросов в DNS-серверу: *рекурсивный***, ***итеративный***.
2. **Реализации DNS-серверов**: BIND (Berkeley Internet Name Domain), Microsoft DNS Server, OpenDNS.

BIND (Berkeley Internet Name Domain, до этого: Berkeley Internet Name Daemon) — открытая и наиболее распространённая реализация DNS-сервера, обеспечивающая выполнение преобразования DNS-имени в IP-адрес и наоборот.

BIND поддерживается организацией Internet Systems Consortium. BIND был создан студентами и впервые был выпущен в BSD 4.3.

В Unix этот сервер является стандартом де-факто, но имеются и альтернативы:

PowerDNS — www.powerdns.com;

MyDNS — DNS-сервер, использующий в качестве БД MySQL;

Microsoft DNS Server — входит в состав серверных версий Windows.

***Утилита NSLOOKUP***

Утилита nslookup используется для проверки способности DNS-серверов выполнять разрешение имен. Утилита может работать в двух режимах:

• режим командной строки – обычный режим запуска утилит командной строки. Утилита nslookup выполняется в этом режиме, если указан какой-либо ключ;

• интерактивный режим – в этом режиме возможен ввод команд и ключей утилиты без повторения ввода имени утилиты.

Команды утилиты nslookup:

• help или ? – вывод справки о командах и параметрах утилиты;

• set – установка параметров работы утилиты;

• server <имя> – установка сервера по умолчанию (Default Server), используемого утилитой, с помощью текущего сервера по умолчанию;

• lserver <имя> – установка сервера по умолчанию утилиты с помощью первоначального; • root – установка сервера по умолчанию утилиты на корневой сервер;

• ls <домен> – вывод информации о соответствии доменных имен IP-адресам для заданного домена;

• exit – выход из интерактивного режима.

# NAT (Network Address Translation)

NAT (от англ. Network Address Translation — «преобразование сетевых адресов») — это механизм в сетях TCP/IP, позволяющий преобразовывать IP-адреса транзитных пакетов.

Механизм NAT определён в RFC 1631, RFC 3022.

Функционирование

Преобразование адресов методом NAT может производиться почти любым маршрутизирующим устройством — маршрутизатором, сервером доступа, межсетевым экраном. Наиболее популярным является SNAT, суть механизма которого состоит в замене адреса источника (англ. source) при прохождении пакета в одну сторону и обратной замене адреса назначения (англ. destination) в ответном пакете. Наряду с адресами источник/назначение могут также заменяться номера портов источника и назначения.

Ваш компьютер может быть подключен к интернету напрямую. Тогда говорят, что у него внешний IP адрес. Обычно это значит, что компьютер подключен сразу к модему (DSL, кабельному или обычному аналоговому).

За NAT означает, что ваш компьютер подключен не к интернету, а к локальной сети. Тогда у него внутренний IP адрес, из интернета сам по себе недоступный.

Доступ к интернету ваш компьютер получает через NAT - процесс трансляции внутренних адресов во внешние и обратно. NAT-устройство обычно называют раутером.

NAT позволяет хостам из intranet прозрачным образом обращаться к хостам в общедоступном пространстве, при этом для внутренних хостов не требуется наличия зарегистрированных (и наиболее дефицитных) сетевых Internet-адресов.

Internet Engineering Task Force вот уже почти десять лет предупреждает о грядущем дефиците адресов в существующем сейчас адресном пространстве, формируемом IPv4. И хотя, по-видимому, новая версия протокола IPv6 будет способна достаточно долгое время удовлетворять требованиям постоянно растущего Internet, предложенные за последние несколько лет меры уже получили законный статус.

Одно из подобных предложений описывает опубликованный в 1994 году стандарт RFC 1631, "Транслятор сетевых IP-адресов" (The IP Network Address Translator). В период становления Internet потребители были вынуждены использовать уникальные в глобальном масштабе сетевые адреса вне зависимости от того, планировали они подключаться к глобальному пространству Internet или нет. Идея состояла в том, чтобы избежать потенциальных конфликтов в том случае, если бывшая ранее частной сеть в конце концов присоединялась к общедоступной Internet.

В условиях продолжающего экспоненциально расширяться Internet присваивание "хороших", то есть уникальных в глобальном смысле сетевых адресов частным сетям стало считаться непроизводительной тратой ценных виртуальных ресурсов. В соответствии со стандартом Network Address Translation определенные IP-адреса выделены для параллельного использования в частных сетях и в общедоступной сети. Как указано в спецификации RFC 1597 "Резервирование адресов для частных Internet" (Address Allocation for Private Internets), разрешено использовать адреса в следующих диапазонах: от 10.0.0.0 до 10.255.255.255; от 172.16.0.0 до 172.31.255.255; от 192.168.0.0 до 192.168.255.255. В соответствии с условиями этого соглашения не предполагается, что по этим адресам маршрутизаторы передают какие-либо пакеты в Internet.

Простейшее устройство NAT имеет два сетевых соединения: одно с Internet и одно с частной сетью. Хосты в частной сети, использующие частные IP-адреса (иногда также называемые адресами Network 10, которые начинаются с 10.0.0.0 в диапазоне, выделенном для частного использования), связываются с Internet, передавая пакеты непосредственно на устройство NAT. В отличие от обычных маршрутизаторов, которые просто считывают адреса отправителя и получателя каждого пакета, прежде чем передать их по назначению, устройства NAT модифицируют заголовки пакетов, заменяя адрес отправителя в частной сети на свой собственный Internet-адрес.

Можно проверить под каким IP адресом вас видят другие компьютеры в интернете. Например на whatsmyip.org ("Your IP Address is x.x.x.x" вверху страницы) или на myipaddress.com.

Если IP адрес вашего компьютера совпадает с показанным одним из этих сайтов, то вы точно подключены к интернету напрямую.

Если, к примеру, один сетевой компьютер обращается к FTP-серверу (порт 21), то при открытии сокета операционная система присваивает сессии любой порт выше 1023. Например, это может быть порт 2153. Тогда IP-пакет, отправляемый со стороны ПК к FTP-серверу, будет содержать IP-адрес отправителя, порт отправителя (2153), IP-адрес получателя и порт назначения (21). IP-адрес и порт отправителя будут использоваться для ответа сервера клиенту. Использование разных портов для различных сетевых сессий позволяет клиентам сети одновременно устанавливать несколько сессий с различными серверами или с сервисами одного сервера.

Теперь рассмотрим процесс установления сессии при использовании NAT-маршрутизатора на границе внутренней сети и сети Интернет.

Когда клиент внутренней сети устанавливает связь с сервером внешней сети, то, как и в случае установки соединения между двумя ПК, открывается сокет, определяемый IP-адресом источника, портом источника, IP-адресом назначения, портом назначения и сетевым протоколом. Когда приложение передает данные через этот сокет, IP-адрес источника и порт источника вставляются в пакет в поля параметров источника. Поля параметров пункта назначения будут содержать IP-адрес сервера и порт сервера. К примеру, компьютер внутренней сети с IP-адресом 192.168.0.1 может обратиться к Web-серверу Глобальной сети с IP-адресом 64.233.188.104. В этом случае операционная система клиента может назначить установленной сессии порт 1251 (порт источника), а порт назначения — это порт Web-сервиса, то есть 80. Тогда в заголовке отправляемого пакета будут указаны следующие атрибуты (рис. 2):

IP-адрес источника: 192.168.0.1;

порт источника: 1251;

IP-адрес получателя: 64.233.183.104;

порт получателя: 80;

протокол: TCP.

Устройство NAT (маршрутизатор) перехватывает исходящий из внутренней сети пакет и заносит в свою внутреннюю таблицу сопоставление портов источника и получателя пакета, используя IP-адрес назначения, порт назначения, внешний IP-адрес устройства NAT, внешний порт, сетевой протокол, а также внутренние IP-адрес и порт клиента.

Предположим, что в рассмотренном выше примере NAT-маршрутизатор имеет внешний IP-адрес 195.2.91.103 (адрес WAN-порта), а для установленной сессии внешний порт NAT-устройства — 3210. В этом случае внутренняя таблица сопоставления портов источника и получателя пакета содержит следующую информацию:

IP-адрес источника: 192.168.0.1;

порт источника: 1251;

внешний IP-адрес

NAT-устройства: 195.2.91.103;

внешний порт NAT-устройства: 3210;

IP-адрес получателя: 64.233.183.104;

порт получателя: 80;

протокол: TCP.

Затем устройство NAT «транслирует» пакет, преобразуя в пакете поля источника: внутренние IP-адрес и порт клиента заменяются внешними IP-адресом и портом устройства NAT. В рассмотренном примере преобразованный пакет будет содержать следующую информацию:

IP-адрес источника: 195.2.91.103;

порт источника: 3210;

IP-адрес получателя: 64.233.183.104;

порт получателя: 80;

протокол: TCP.

Преобразованный пакет пересылается по внешней сети и в итоге попадает на заданный сервер.

Получив пакет, сервер будет направлять ответные пакеты на внешний IP-адрес и порт устройства NAT (маршрутизатора), указывая в полях источника свои собственные IP-адрес и порт (рис. 3). В рассмотренном примере ответный пакет от сервера будет содержать в заголовке следующую информацию:

IP-адрес источника: 64.233.183.104;

порт источника: 80;

IP-адрес получателя: 195.2.91.103;

порт получателя: 3210;

протокол: TCP.

Устройство NAT принимает эти пакеты от сервера и анализирует их содержимое на основе своей таблицы сопоставления портов. Если в таблице будет найдено сопоставление порта, для которого IP-адрес источника, порт источника, порт назначения и сетевой протокол из входящего пакета совпадают с IP-адресом удаленного узла, с удаленным портом и с сетевым протоколом, указанным в сопоставлении портов, то NAT выполнит обратное преобразование: заменит внешний IP-адрес и внешний порт в полях назначения пакета на IP-адрес и внутренний порт клиента внутренней сети. Таким образом, пакет, передаваемый во внутреннюю сеть, для рассмотренного выше примера будет иметь следующие атрибуты:

IP-адрес источника: 64.233.183.104;

порт источника: 80;

IP-адрес получателя: 192.168.0.1;

порт получателя: 1251;

протокол: TCP.

Однако если в таблице сопоставления портов не находится соответствия, то входящий пакет отвергается и соединение разрывается.

Благодаря NAT-маршрутизатору любой ПК внутренней сети получает возможность передавать данные в Глобальную сеть с использованием внешнего IP-адреса и порта маршрутизатора. При этом IP-адреса внутренней сети, как присвоенные сессиям порты, остаются невидимыми со стороны внешней сети.

Помимо source NAT (предоставления пользователям локальной сети с внутренними адресами доступа к сети Интернет) часто применяется также destination NAT, когда обращения извне транслируются межсетевым экраном на сервер в локальной сети, имеющий внутренний адрес и потому недоступный извне сети непосредственно (без NAT).

Существует 3 базовых концепции трансляции адресов: статическая (Static Network Address Translation), динамическая (Dynamic Address Translation ), 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, доступного в пуле адресов, во время коммуникации.

Перегруженный NAT (NAPT, NAT Overload, PAT, маскарадинг) — форма динамического NAT, который отображает несколько незарегистрированных адресов в единственный зарегистрированный IP адрес, используя различные порты. Известен также как PAT (Port Address Translation)

При перегрузке, каждый компьютер в частной сети транслируется в тот же самый адрес, но с различным номером порта.

Маскарадинг (англ. Masquerading) — тип трансляции сетевого адреса, при которой адрес отправителя подставляется динамически, в зависимости от назначенного интерфейсу адреса. Разделение SNAT (source NAT) и маскарадинга характерно для iptables, в маршрутизаторах cisco операция трансляции (ip nat source) умеет работать как с заданными адресами, так и с интерфейсами (используя автоматически назначенный интерфейсу адрес).

Ещё одной особенностью маскарадинга (в iptables) является «забывание» про установленные трансляции при остановке (down) интерфейса. Это связано с тем, что после поднятия интерфейса его адрес, вероятнее всего (в случае DHCP/Dialup) будет другим, и записи о ранее выполненных трансляциях не будут иметь смысла.

Преимущества

NAT выполняет три важных функции.

Позволяет сэкономить IP-адреса (только в случае использования NAT в режиме PAT), транслируя несколько внутренних IP-адресов в один внешний публичный IP-адрес (или в несколько, но меньшим количеством, чем внутренних). По такому принципу построено большинство сетей в мире: на небольшой район домашней сети местного провайдера или на офис выделяется 1 «белый» (то есть внешний) IP-адрес, за которым работают и получают доступ вовне все «серые» (то есть внутренние) IP-адреса.

Позволяет предотвратить или ограничить обращение снаружи ко внутренним хостам, оставляя возможность обращения изнутри наружу. При инициации соединения изнутри сети создаётся трансляция. Ответные пакеты, поступающие снаружи, соответствуют созданной трансляции и поэтому пропускаются. Если для пакетов, поступающих снаружи, соответствующей трансляции не существует (а она может быть созданной при инициации соединения или статической), они не пропускаются.

Позволяет скрыть определённые внутренние сервисы внутренних хостов/серверов. По сути, выполняется та же указанная выше трансляция на определённый порт, но возможно подменить внутренний порт официально зарегистрированной службы (например, 80-й порт TCP (HTTP-сервер) на внешний 54055-й). Тем самым, снаружи, на внешнем IP-адресе после трансляции адресов на сайт (или форум) для осведомлённых посетителей можно будет попасть по адресу http://example.org:54055, но на внутреннем сервере, находящимся за NAT, он будет работать на обычном 80-м порту. Повышение безопасности и скрытие «непубличных» ресурсов.

Недостатки

При использовании NAT хосты Internet взаимодействуют напрямую с NAT-устройством, а не с реальным хостом в частной сети. Входящие пакеты передаются по IP-адресу устройства NAT, а устройство меняет адрес назначения в заголовке пакета со своего собственного Internet-адреса на адрес частной сети реального хоста назначения.

В результате - теоретически - единый уникальный в глобальном масштабе IP-адрес может скрывать за собой сотни, тысячи или даже миллионы хостов, чьи адреса определены в частной сети. На практике, однако, такой подход имеет ряд недостатков. Например, многие Internet-протоколы и приложения существенно опираются на тот факт, что сеть в действительности поддерживает соединение из конца в конец, когда пакеты передаются без каких-либо изменений от отправителя получателю. Архитектура IP-защиты, к примеру, не может поддерживать работу через устройство NAT, поскольку оригинальные заголовки, в которых указаны исходные IP-адреса отправителей, защищены с помощью цифровой подписи. Измените исходный адрес - и цифровая подпись потеряет свою целостность.

Использование NAT к тому же усложняет работу администраторов. Конечно, NAT - превосходное решение для организации, филиала или даже подразделения, которые не могут получить достаточное количество уникальных Internet-адресов. Однако при реорганизации, объединении или покупке другой компанией, когда требуется объединение двух или нескольких частных сетей, возникают большие сложности. Даже если структура организации остается достаточно стабильной, системы NAT могут случайно оказаться вложенными структурами, превращая маршрутизацию в настоящий кошмар.

Не все протоколы могут «преодолеть» NAT. Некоторые не в состоянии работать, если на пути между взаимодействующими хостами есть трансляция адресов. Некоторые межсетевые экраны, осуществляющие трансляцию IP-адресов, могут исправить этот недостаток, соответствующим образом заменяя IP-адреса не только в заголовках IP, но и на более высоких уровнях (например, в командах протокола FTP)..

Из-за трансляции адресов «много в один» появляются дополнительные сложности с идентификацией пользователей и необходимость хранить полные логи трансляций.

DoS со стороны узла, осуществляющего NAT — если NAT используется для подключения многих пользователей к одному и тому же сервису, это может вызвать иллюзию DoS-атаки на сервис (множество успешных и неуспешных попыток). Например, избыточное количество пользователей мессенджера за NAT приводит к проблеме с подключением к серверу некоторых пользователей из-за превышения допустимой скорости подключений. Частичным решением проблемы является использование пула адресов (группы адресов), для которых осуществляется трансляция.

В некоторых случаях, необходимость в дополнительной настройке при работе с пиринговыми сетями и некоторыми другими программами (Battle.net-игры), в которых необходимо не только инициировать исходящие соединения, но также принимать входящие. Однако, если NAT устройство и ПО требующее дополнительной настройки поддерживает технологию Universal Plug and Play, то в этом случае настройка произойдет полностью автоматически и прозрачно для пользователя.

За границей частной сети

Хосты внутри частной сети обычно без труда связываются с серверами за пределами организации, однако хосты в Internet не всегда могут легко получать доступ к серверам частной сети. Если внешним хостам требуется такой доступ, они взаимодействуют напрямую с одним хостом - самим устройством NAT. Частная сеть, по существу, не видна внешнему миру, и считается, что весь трафик из этой сети - это трафик, отправляемый и присылаемый на устройство NAT.

Network Address Port Translation (NAPT) помогает до некоторой степени решить эту проблему за счет преобразования не только самого IP-адреса, но и порта транспортного уровня. Таким образом, входящий пакет, адресованный на порт Port 80 (как правило, используемый для HTTP-пакетов) на устройстве NAPT, может быть преобразован и передан на Web-сервер частной сети. Без преобразования порта устройство NAT не может "узнать", на какой хост частной сети передавать эти пакеты.

NAT часто рассматривают как решение проблемы защиты. В конце концов, частная сеть действительно как бы скрыта от посторонних глаз. Однако, если злоумышленник получит управление устройством NAT, уязвимой становится вся сеть целиком. Устройство не должно рассматриваться как замена межсетевому экрану, хотя простые устройства, реализующие преобразование адресов, могут оказаться полезны для защиты сетей дома или в небольшом офисе.

Хотя сторонники NAT пропагандируют его как долговременное решение проблемы дефицита адресов IPv4, на самом деле оно способно лишь временно снять остроту ситуации. Если игнорировать архитектурные проблемы и сложности, связанные с развертыванием сетей, само по себе адресное пространство IPv4 тем не менее остается ограниченным и в скором времени может быть исчерпано.

NAT Traversal

NAT Traversal (прохождение или автонастройка NAT) — это набор возможностей, позволяющих сетевым приложениям определять, что они находятся за устройством, обеспечивающим NAT, узнавать внешний IP-адрес этого устройства и выполнять сопоставление портов для пересылки пакетов из внешнего порта NAT на внутренний порт, используемый приложением; все это выполняется автоматически, пользователю нет необходимости вручную настраивать сопоставления портов или вносить изменения в какие-либо другие параметры. Однако существуют меры предосторожности в доверии к таким приложениям — они получают обширный контроль над устройством, появляются потенциальные уязвимости.

Протокол NAT Traversal (NAT-T) инкапсулирует трафик IPSec и одновременно создает пакеты UDP, которые NAT корректно пересылает. Для этого NAT-T помещает дополнительный заголовок UDP перед пакетом IPSec, чтобы он во всей сети обрабатывался как обычный пакет UDP и хост получателя не проводил никаких проверок целостности. После поступления пакета по месту назначения заголовок UDP удаляется, и пакет данных продолжает свой дальнейший путь как инкапсулированный пакет IPSec. Итак, с помощью техники NAT-T возможно установление связи между клиентами IPSec в защищённых сетях и общедоступными хостами IPSec через межсетевые экраны.

Операционные системы с поддержкой NAT

При недостаточном финансировании, либо при наличии уже существующего сервера под управлением серверной ОС возможно организовать трансляцию адресов без необходимости закупки дополнительных устройств. В таком случае, оптимальным будет наличие по крайней мере двух сетевых адаптеров в сервере (возможны варианты с одним, но при наличии trunk-VLAN).

Все существующие и использующиеся серверные ОС поддерживают простейшую трансляцию адресов.

С точки зрения же отказоустойчивости, гибкости и производительности, используются операционные системы UNIX (большинство GNU/Linux, \*BSD-системы, а также OpenSolaris и др.). Во многих из них NAT доступен «из коробки», в других возможна реализация за счёт добавления модулей в сочетании с межсетевыми экранами с функциями трансляции адресов (IPFW, IPtables и др.).

Межсетевой экран или сетевой экран — комплекс аппаратных или программных средств, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами.

Также сетевые экраны часто называют фильтрами, так как их основная задача — не пропускать (фильтровать) пакеты, не подходящие под критерии, определённые в конфигурации.

Некоторые сетевые экраны также позволяют осуществлять трансляцию адресов — динамическую замену внутрисетевых (серых) адресов или портов на внешние, используемые за пределами ЛВС.

Другие названия

Брандма́уэр (нем. Brandmauer) — заимствованный из немецкого языка термин, являющийся аналогом английского firewall в его оригинальном значении (стена, которая разделяет смежные здания, предохраняя от распространения пожара). Интересно, что в области компьютерных технологий в немецком языке употребляется слово «firewall».

Файрво́лл, файрво́л, файерво́л, фаерво́л — образовано транслитерацией английского термина firewall, эквивалентного термину межсетевой экран.

Основная задача любого брандмауэра в конечном счете сводится к обеспечению безопасности внутренней сети. Для решения этой задачи брандмауэры должны уметь маскировать защищаемую сеть, блокировать все известные типы хакерских атак, блокировать утечку информации из внутренней сети, контролировать приложения, получающие доступ во внешнюю сеть.

Для того чтобы реализовать указанные функции, брандмауэры анализируют весь трафик между внешней и внутренней сетями на предмет его соответствия тем или иным установленным критериям или правилам, определяющим условия прохождения трафика из одной сети в другую. Если трафик отвечает заданным критериям, то брандмауэр пропускает его через себя. В противном случае, то есть если установленные критерии не соблюдены, трафик блокируется брандмауэром. Брандмауэры фильтруют как входящий, так и исходящий трафики, а также позволяют управлять доступом к определенным сетевым ресурсам или приложениям. Они могут фиксировать все попытки несанкционированного доступа к ресурсам локальной сети и выдавать предупреждения о попытках проникновения.

По своему назначению брандмауэры больше всего напоминают контрольно-пропускной пункт (КПП) охраняемого объекта, где производится проверка документов всех входящих на территорию объекта и всех покидающих ее. Если пропуск в порядке — доступ на территорию разрешен. Совершенно аналогично действуют и брандмауэры, только в роли людей, проходящих через КПП, выступают сетевые пакеты, а пропуском является соответствие заголовков этих пакетов предопределенному набору правил.

Можно ли утверждать, что брандмауэр обеспечивает 100-процентную безопасность пользовательской сети или персонального ПК? Безусловно, нет. Хотя бы потому, что вообще ни одна система не дает 100-процентной гарантии безопасности. К брандмауэру стоит относиться как к средству, которое, при правильной его настройке, способно в значительной мере осложнить задачу злоумышленника по проникновению в персональный компьютер пользователя. Подчеркнем: лишь осложнить, но вовсе не гарантировать абсолютной безопасности. Кстати, если речь заходит не о защите локальной сети, а о защите отдельного ПК, имеющего доступ в Интернет, то с обеспечением его персональной безопасности успешно справляется и брандмауэр ICF (Internet Connection Firewall), встроенный в операционную систему начиная с Windows XP. Поэтому в дальнейшем мы будем говорить лишь о корпоративных аппаратных брандмауэрах, ориентированных на защиту небольших сетей.

Если брандмауэр, устанавливаемый на входе в локальную сеть, активирован по полной программе (как правило, это соответствует настройкам по умолчанию), то защищаемая им сеть полностью непроницаема и недоступна извне. Однако у столь полной непроницаемости внутренней сети есть и своя оборотная сторона. Дело в том, что в этом случае становится невозможно пользоваться Интернет-сервисами (например, ICQ и тому подобными программами), установленными на ПК. Таким образом, задача настройки брандмауэра заключается в том, чтобы в первоначально глухой стене, которую представляет собой брандмауэр для злоумышленника, проделать окошки, предоставив возможность пользовательским программам отвечать на запросы извне и в конечном счете реализовать подконтрольное взаимодействие внутренней сети с внешним миром. Однако чем больше подобных окон появляется в такой стене, тем более уязвимой становится и сама сеть. Так что еще раз подчеркнем: ни один брандмауэр не может гарантировать абсолютной безопасности защищаемой им локальной сети.

Возможности брандмауэров и степень их интеллектуальности зависят от того, на каком уровне эталонной модели OSI они функционируют. Чем выше уровень OSI, на основе которого построен брандмауэр, тем выше обеспечиваемый им уровень защиты.

Помним, что модель OSI (Open System Interconnection) включает семь уровней сетевой архитектуры. Первый, самый нижний, — это физический уровень. За ним следуют канальный, сетевой, транспортный, сеансовый уровни, уровень представления и прикладной уровень, или уровень приложений. Для того чтобы обеспечивать фильтрацию трафика, брандмауэр должен работать как минимум на третьем уровне модели OSI, то есть на сетевом уровне, где происходит маршрутизация пакетов на основе преобразования MAC-адресов в сетевые адреса. С точки зрения протокола TCP/IP этот уровень соответствует уровню IP (Internet Protocol). Получая информацию сетевого уровня, брандмауэры способны определить адрес источника и получателя пакета и проверить, допустима ли передача трафика между данными адресатами. Однако информации сетевого уровня для анализа содержимого пакета недостаточно. Брандмауэры, функционирующие на транспортном уровне модели OSI, получают несколько больше информации о пакетах и в этом смысле могут предоставлять более интеллектуальные схемы защиты сетей. Что же касается брандмауэров, работающих на уровне приложений, то им доступна полная информация о сетевых пакетах, а значит, такие брандмауэры обеспечивают наиболее надежную сетевую защиту.

В зависимости от уровня модели OSI, на которых функционируют брандмауэры, исторически сложилась следующая классификация этих устройств:

пакетный фильтр (packet filter);

шлюз сеансового уровня (circuit-level gateway);

шлюз прикладного уровня (application-level gateway);

Stateful Packet Inspection (SPI).

Отметим, что данная классификация имеет лишь исторический интерес, поскольку все современные брандмауэры относятся к категории наиболее совершенных (в плане защиты сети) SPI-брандмауэров.

Пакетные фильтры

Брандмауэры типа пакетных фильтров являются наиболее простыми (наименее интеллектуальными). Эти брандмауэры работают на сетевом уровне модели OSI или на IP-уровне стека протоколов TCP/IP. Такие брандмауэры в обязательном порядке присутствуют в каждом маршрутизаторе, поскольку любой маршрутизатор работает как минимум на третьем уровне модели OSI.

Задача пакетных фильтров заключается в фильтрации пакетов на основе информации об IP-адресе источника или получателя, а также о номерах портов.

В брандмауэрах типа пакетных фильтров каждый пакет, до того как он будет передан, анализируется на предмет соответствия критериям передачи или блокировки передачи. В зависимости от пакета и от сформированных критериев передачи брандмауэр может передать пакет, отвергнуть его или послать уведомление инициатору передачи.

Пакетные фильтры просты в реализации и практически не влияют на скорость маршрутизации.

Шлюзы сеансового уровня

Шлюзы сеансового уровня — это брандмауэры, которые работают на сеансовом уровне модели OSI или на TCP (Transport Control Protocol) уровне стека протоколов TCP/IP. Данные брандмауэры отслеживают процесс установления TCP-соединения (организацию сеансов обмена данными между оконечными машинами) и позволяют определить, является ли данный сеанс связи легитимным. Данные, передаваемые к удаленному компьютеру во внешней сети через шлюз на сеансовом уровне, не содержат информации об источнике передачи, то есть все выглядит таким образом, как будто данные отправляются самим брандмауэром, а не компьютером во внутренней (защищаемой) сети. Все брандмауэры на основе NAT-протокола являются шлюзами сеансового уровня (протокол NAT будет описан ниже).

Шлюзы сеансового уровня также не оказывают существенного влияния на скорость маршрутизации. В то же время эти шлюзы не способны осуществлять фильтрацию отдельных пакетов.

Шлюзы прикладного уровня

Шлюзы прикладного уровня, или proxy-серверы, функционируют на прикладном уровне модели OSI. Прикладной уровень отвечает за доступ приложений в сеть. К задачам этого уровня относятся перенос файлов, обмен почтовыми сообщениями и управление сетью. Получая информацию о пакетах на прикладном уровне, шлюзы прикладного уровня могут реализовывать блокировку доступа к определенным сервисам. К примеру, если шлюз прикладного уровня сконфигурирован как Web-proxy, то любой трафик, относящийся к протоколам Telnet, FTP, Gopher, будет заблокирован. Поскольку эти брандмауэры анализируют пакеты на прикладном уровне, они способны осуществлять фильтрацию специфических команд, например, http:post, get и т.д. Данная функция недоступна ни пакетным фильтрам, ни шлюзам сеансового уровня. Шлюзы прикладного уровня могут также использоваться для регистрации активности отдельных пользователей и для установления ими сеансов связи. Эти брандмауэры предлагают более надежный способ защиты сетей по сравнению со шлюзами сеансового уровня и пакетными фильтрами.

SPI-брандмауэры

Последний тип брандмауэров — Stateful Packet Inspection (SPI) — объединяет в себе преимущества одновременно и пакетных фильтров, и шлюзов сеансового уровня, и шлюзов прикладного уровня. То есть фактически речь идет о многоуровневых брандмауэрах, которые работают одновременно на сетевом, сеансовом и прикладном уровнях.

SPI-брандмауэры осуществляют фильтрацию пакетов на сетевом уровне, определяют легитимность установления сеанса связи на основании данных сеансового уровня и анализируют содержимое пакетов, основываясь на данных прикладного уровня.

Эти брандмауэры реализуют наиболее надежный способ защиты сетей и на данный момент являются стандартом де-факто.

Методология и возможности по настройке брандмауэров зависят от конкретной модели. К сожалению, не существует единых правил настройки и тем более единообразного интерфейса. Можно говорить лишь о неких общих правилах, которых стоит придерживаться. Собственно, основное правило довольно простое — необходимо запрещать все, что не требуется для нормального функционирования сети.

Чаще всего возможности по настройке брандмауэров сводятся к активированию некоторых предопределенных правил и к созданию статических правил в виде таблицы.

Другой способ настройки брандмауэра сводится к созданию статических правил, которые позволяют не только защитить сеть снаружи, но и ограничить пользователям локальной сети доступ во внешнюю сеть. Возможности по созданию правил достаточно гибкие и позволяют реализовать практически любую ситуацию. Для создания правила задаются IP-адрес (или диапазон адресов) источника, порты источника, IP-адреса и порты получателя, тип протокола, направление передачи пакета (из внутренней сети во внешнюю или наоборот), а также действие, которое должно предприниматься при обнаружении пакета с означенными свойствами (отбросить или пропустить пакет). К примеру, если требуется запретить пользователям внутренней сети (диапазон IP-адресов: 192.168.1.1-192.168.1.100) обращение к FTP-серверу (порт 21), расположенному по внешнему IP-адресу 64.233.183.104, то правило может быть сформулировано следующим образом:

направление передачи пакетов: LAN-to-WAN;

IP-адреса источника: 192.168.1.1-192.168.1.100;

порт источника: 1-65535;

IP-адрес получателя: 64.233.183.104;

порт получателя: 21;

протокол: TCP;

действие: drop.

все современные маршрутизаторы со встроенными брандмауэрами поддерживают протокол трансляции сетевых адресов NAT (Network Address Translation).

Разновидности сетевых экранов

Сетевые экраны подразделяются на различные типы в зависимости от следующих характеристик:

обеспечивает ли экран соединение между одним узлом и сетью или между двумя или более различными сетями;

происходит ли контроль потока данных на сетевом уровне или более высоких уровнях модели OSI;

отслеживаются ли состояния активных соединений или нет.

В зависимости от охвата контролируемых потоков данных сетевые экраны делятся на:

традиционный сетевой (или межсетевой) экран — программа (или неотъемлемая часть операционной системы) на шлюзе (сервере передающем трафик между сетями) или аппаратное решение, контролирующие входящие и исходящие потоки данных между подключенными сетями.

персональный сетевой экран — программа, установленная на пользовательском компьютере и предназначенная для защиты от несанкционированного доступа только этого компьютера.

Вырожденный случай — использование традиционного сетевого экрана сервером, для ограничения доступа к собственным ресурсам.

В зависимости от уровня, на котором происходит контроль доступа, существует разделение на сетевые экраны, работающие на:

сетевом уровне, когда фильтрация происходит на основе адресов отправителя и получателя пакетов, номеров портов транспортного уровня модели OSI и статических правил, заданных администратором;

сеансовом уровне (также известные как stateful) — отслеживающие сеансы между приложениями, не пропускающие пакеты нарушающих спецификации TCP/IP, часто используемых в злонамеренных операциях — сканировании ресурсов, взломах через неправильные реализации TCP/IP, обрыв/замедление соединений, инъекция данных.

уровне приложений, фильтрация на основании анализа данных приложения, передаваемых внутри пакета. Такие типы экранов позволяют блокировать передачу нежелательной и потенциально опасной информации, на основании политик и настроек.

Некоторые решения, относимые к сетевым экранам уровня приложения, представляют собой прокси-серверы с некоторыми возможностями сетевого экрана, реализуя прозрачные прокси-серверы, со специализацией по протоколам. Возможности прокси-сервера и многопротокольная специализация делают фильтрацию значительно более гибкой, чем на классических сетевых экранах, но такие приложения имеют все недостатки прокси-серверов (например, анонимизация трафика).

.

Прокси-сервер (от англ. proxy — «представитель, уполномоченный») — служба в компьютерных сетях, позволяющая клиентам выполнять косвенные запросы к другим сетевым службам. Сначала клиент подключается к прокси-серверу и запрашивает какой-либо ресурс (например, e-mail), расположенный на другом сервере. Затем прокси-сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного кэша (в случаях, если прокси имеет свой кэш). В некоторых случаях запрос клиента или ответ сервера может быть изменён прокси-сервером в определённых целях. Также прокси-сервер позволяет защищать клиентский компьютер от некоторых сетевых атак и помогает сохранять анонимность клиента.

Использование

Чаще всего прокси-серверы применяются для следующих целей:

Обеспечение доступа с компьютеров локальной сети в Интернет.

Кэширование данных: если часто происходят обращения к одним и тем же внешним ресурсам, то можно держать их копию на прокси-сервере и выдавать по запросу, снижая тем самым нагрузку на канал во внешнюю сеть и ускоряя получение клиентом запрошенной информации.

Сжатие данных: прокси-сервер загружает информацию из Интернета и передаёт информацию конечному пользователю в сжатом виде. Такие прокси-серверы используются в основном с целью экономии внешнего трафика.

Защита локальной сети от внешнего доступа: например, можно настроить прокси-сервер так, что локальные компьютеры будут обращаться к внешним ресурсам только через него, а внешние компьютеры не смогут обращаться к локальным вообще (они «видят» только прокси-сервер).

Ограничение доступа из локальной сети к внешней: например, можно запретить доступ к определённым веб-сайтам, ограничить использование интернета каким-то локальным пользователям, устанавливать квоты на трафик или полосу пропускания, фильтровать рекламу и вирусы.

Анонимизация доступа к различным ресурсам. Прокси-сервер может скрывать сведения об источнике запроса или пользователе. В таком случае целевой сервер видит лишь информацию о прокси-сервере, например, IP-адрес, но не имеет возможности определить истинный источник запроса. Существуют также искажающие прокси-серверы, которые передают целевому серверу ложную информацию об истинном пользователе.

Прокси-сервер, к которому может получить доступ любой пользователь сети интернет, называется открытым.

Виды прокси-серверов

Прозрачный прокси — схема связи, при которой трафик, или его часть, перенаправляется на прокси-сервер неявно (средствами маршрутизатора). При этом клиент может использовать все преимущества прокси-сервера без дополнительных настроек, но с другой стороны, не имеет выбора.

Обратный прокси — прокси-сервер, который в отличие от прямого, ретранслирует запросы клиентов из внешней сети на один или несколько серверов, логически расположенных во внутренней сети. Часто используется для балансировки сетевой нагрузки между несколькими веб-серверами и повышения их безопасности, играя при этом роль межсетевого экрана на прикладном уровне.

Веб-прокси (англ. «web-based proxy») — это прокси-сервер и анонимайзер особого вида, представляющий собой веб-приложение (чаще всего PHP или Perl скрипт) установленное на веб-сервере, выступающее в роли посредника для загрузки контента различных веб-сайтов.

Веб-прокси могут быть использованы для:

ускорения загрузки веб-сайтов;

тестирования онлайн сервисов;

обхода ограничений Администратора локальной сети на доступ к определенным адресам веб-сайтов;

сокрытия реального IP-адреса и анонимного доступа к веб-сайтам;

получения доступа к веб-сайтам закрытым для просмотра пользователей определенных стран;

и многих других целей.

Использование веб-прокси

Веб-прокси устанавливается на веб-сервере и обеспечивает возможность его пользователям загружать произвольные веб-ресурсы от IP адреса этого сервера, что и обеспечивает анонимность веб-сёрфинга. Для использования веб-прокси, необходимо ввести адрес интересующего ресурса и, при желании указав дополнительные настройки, активировать переход. К дополнительным настройкам относятся:

запрет/разрешение на обработку JavaScript;

использование Cookie;

кодирование веб-адреса и страницы;

замена или очистка заголовка;

и ряд других, зависящий от конкретного приложения.

При кодирование веб-адреса загружаемого ресурса применяются алгоритмы Base64 и Rot13. Для кодирования веб-страниц используется преобразование в HEX код всего текстового содержания веб-страницы. Применение этих методов позволяет получить доступ к страницам с запрещенным контентом и перейти по веб-адресу заблокированному Администратором. Административными методами противодействия использованию в локальных сетях веб-прокси могут стать: усиление фильтрации контента и блокировка адресов публичных веб-прокси.

Стоит отметить, что большинство хостеров запрещают устанавливать на дешевых тарифах виртуального хостинга веб-прокси по причине большой нагрузки на сервер и активного потребления входящего трафика. Несмотря на это ограничение, в Интернет широко распространено создание целых сетей веб-прокси, они представляют собой большое количество различных доменых имен, размещенных на одном или нескольких IP-адресах выделенных серверов. Такие сети используются их владельцами для получения прибыли от рекламы, размещаемой на этих ресурсах.

Ремейлер (англ. remailer) — это сервер, получающий сообщение электронной почты и переправляющий его по адресу, указанному отправителем. В процессе переадресации вся информация об отправителе уничтожается, поэтому конечный получатель лишён возможности выяснить, кто является автором сообщения.[1] Некоторые из ремейлеров позволяют также шифровать письма и указывать фиктивный адрес отправителя, но большинство из них прямо указывают в заголовке, что электронное сообщение анонимно. В качестве ремейлеров могут выступать специализированные веб-сайты, открытые SMTP-серверы и анонимные сети вроде Mixminion.

Виды ремейлеров

Ремейлеры делятся на анонимные и псевдо-анонимные.

При использовании псевдо-анонимного ремейлера, его оператор знает адрес электронной почты, который необходим для получения ответа на письмо. Тайна связи полностью зависит от оператора, который может стать жертвой угроз, шантажа или социальной инженерии. Преимуществом псевдо-анонимных ремейлеров является их юзабилити, за которое пользователь расплачивается меньшей защищённостью.

Анонимные ремейлеры обеспечивают гораздо более высокую секретность, но при этом они и сложнее в использовании. Их операторы не могут знать, какие данные пересылаются через них, а поэтому нет гарантии своевременной доставки сообщения, которое может и вовсе затеряться.[2] В обмен на высокое время ожидания анонимные ремейлеры достаточно надёжно скрывают от посторонних глаз реальный адрес и содержимое сообщения.

Mixminion — стандарт реализации третьего типа протокола анонимной пересылки электронной почты. Mixminion может отсылать и принимать анонимные сообщения электронной почты.

Mixminion использует архитектуру «перемешанных сетей» (англ. Mix network) для предоставления очень высокой степени анонимности, а также для предотвращения прослушивания и других видов атак при пересылке сообщений. Серверы, названные «миксерами», которые запускают волонтёры, принимают сообщения, расшифровывают их, реформируют и передают в истинный пункт назначения. Каждое письмо электронной почты проходит через несколько серверов так, что ни один из серверов не может быть связан как отправитель для получателя.

Отсылая анонимные сообщения, Mixminion разбивает его на одинаковые по размеру участки, приводя пакеты в одинаковый вид и выбирая путь для отправления через смешанную сеть для каждого пакета. Программа шифрует каждый пакет публичным ключом для каждого сервера на его пути, один за другим. При отправлении пакета, Mixminion посылает его в первый «микс-сервер» на пути. Первый сервер расшифровывает пакет, читая, какой следующий сервер должен принять пакет, и передаёт его. В итоге, пакет прибывает к конечному «микс-серверу», который посылает его выбранному получателю. Так как ни один сервер не знает более чем одного смежного сервера, они не могут связать получателя с отправителем.

Mixminion поддерживает технологию «одноразовых блоков ответа» (англ. «Single-Use Reply Blocks», сокр. «SURB») для опознания анонимных адресатов. SURB кодирует половину пути к адресату, далее, каждый «микс-сервер» в очереди «разворачивает» один слой в пути, а после этого зашифровывает сообщение для получателя-адресата. Когда сообщение достигает адресата, он может расшифровать сообщение и прочитать, какой SURB был использован для его отправки, но отправитель не знает, какой получатель принял анонимное сообщение.

# Webservice (веб-сервис)

**Веб-сервисы**

Веб-сервисы меняют все! Точно так же, как железнодорожные перевозки влияют на экономику страны, так и веб-сервисы коренным образом изменяют правила электронной коммерции. Передавая большой объем данных более эффективно и дешевле, чем ранее, они обеспечивают связь между приложениями, выполняемыми в любых удаленных точках земного шара. В результате между предпринимателями и потребителями устанавливается быстрая, удобная и продуктивная связь.

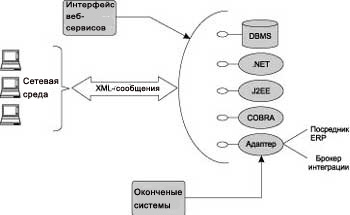
Сеть в ее современном состоянии плохо поддерживает программно-ориентированное взаимодействие. Частным лицам и компаниям, занимающимся коммерцией с использованием сети Интернет, необходим способ публикации ссылок на их приложения и данные, практически такой же, как при публикации ссылок на их веб-страницы. Интернет-приложения должны иметь возможность находить другие интернет-приложения, обращаться к ним и автоматически взаимодействовать с ними. Обеспечивая "общение" между программами, веб-сервисы расширяют возможности Интернета. За счет поддержки веб-сервисов размещенные в различных узлах Интернета приложения могут взаимодействовать непосредственно, как будто они являются частью одной крупной информационной системы.

**Основы веб-сервисов**

Веб-сервисы преобразуют XML-документы (Extensible Markup Language, XML) в ИТ-системах. Веб-сервисы - это XML-приложения, осуществляющие связывание данных с программами, объектами, базами данных либо с деловыми операциями целиком. Между веб-сервисом и программой осуществляется обмен XML-документами, оформленными в виде сообщений. Стандарты веб-сервисов определяют формат таких сообщений, интерфейс, которому передается сообщение, правила привязки содержания сообщения к реализующему сервис приложению и обратно, а также механизмы публикации и поиска интерфейсов.

Веб-сервисы могут использоваться во многих приложениях. Независимо от того, откуда запускаются веб-сервисы, с настольных компьютеров клиентов или с переносных, они могут использоваться для обращения к таким интернет-приложениям, как система предварительных заказов или контроля выполнения заказов. Веб-сервисы пригодны для В2В-интеграции (business-to-business), замыкая приложения, выполняемые различными организациями, в один производственный процесс. Веб-сервисы также могут решать более широкую проблему интеграции приложений предприятия (Enterprise Application Integration, EAI), осуществляя связь нескольких приложений одного предприятия с несколькими другими приложениями, размещенными как "до", так и "после" брандмауэра. Во всех перечисленных случаях технологии веб-сервисов являются "связующим звеном", объединяющим различные части программного обеспечения.

Как видно из рис. 1.1, веб-сервисы представляют собой оболочку, обеспечивающую стандартный способ взаимодействия с прикладными программными средами, такими как системы управления базами данных (СУБД), .NET, J2EE (Java2 Platform, Enterprise Edition), CORBA (Common Object Request Broker Architecture), посредники пакетов планирования ресурсов предприятия (Enterprise Resource Planning, ERP), брокеров интеграции и пр.



*Рис. 1.1. Веб-сервисы взаимодействуют с прикладными системами*

Интерфейсы веб-сервисов получают из сетевой среды стандартные XML-сообщения, преобразуют XML-данные в формат, "понимаемый" конкретной прикладной программной системой, и отправляют ответное сообщение (последнее - не обязательно). Программная реализация веб-сервисов (базовое программное обеспечение, нижний уровень) может быть создана на любом языке программирования с использованием любой операционной системы и любого связующего программного обеспечения (middleware).

Веб-сервисы объединяют программирование и концепции Сети. Веб-сервисы сочетают параметры программных приложений и абстрактные характеристики Сети. Современные интернет-технологии частично достигают своих целей, поскольку они определены на очень высоком отвлеченном уровне, что обеспечивает совместимость с любой операционной системой, любым программным и аппаратным обеспечением. Инфраструктура, основанная на применении веб-сервисов, пользуется этим уровнем абстракции и включает в себя связанную с данными семантическую информацию, то есть веб-сервисы определяют не только данные, но и порядок обработки и преобразования этих данных в базовые программные приложения и обратно.

**Простой пример: поиск информации**

В настоящее время большинство сервисов вызываются по Сети посредством ввода данных в HTML-формы и отправки этих данных сервису путем добавления их в строку унифицированного указателя информационного ресурса (Uniform Resource Locator, URL):

<http://www.google.com/search?q=Skate+boots&btnG=Google+Search>

Этот пример иллюстрирует простоту веб-взаимодействия (например, поиска, покупки акций или запроса маршрута движения), где параметры и ключевые слова внедряются непосредственно в URL. В данном случае представлен простой запрос поиска skate boots (ботинки с коньками) в строке обращения к поисковой машине Google. Ключевое слово search (искать) представляет сервис, к которому будет осуществлено обращение, а параметр Skate+boots является строкой поиска, которая была введена в HTML-форме на странице веб-сайта Google. Сервис поиска Google передаст этот запрос к различным поисковым машинам, которые вернут список URL для страниц, на которых имеется соответствие параметру поиска Skate+boots. Данный малоэффективный способ поиска в Сети полностью основан на установлении соответствия указанной текстовой строки и индексированных HTML-страниц.

XML - лучший способ отправки данных. XML предоставляет значительные преимущества при передаче данных через Интернет. Теперь предыдущий запрос можно представить в виде XML-документа:

<SOAP-ENV:Body>

<s:SearchRequest

xmlns:s="www.xmlbus.com/SearchService">

<pl>Skate</pl>

<p2>boots</p2>

<p3>size 7.5</p3>

</s:SearchRequest>

</SOAP-ENV:Body>

Отправка запроса в виде XML-документа имеет следующие преимущества: возможность определения типов данных и структур, большую гибкость и расширяемость. XML может представлять структурированные данные или данные определенного типа (например, допустимо указывать значение поля size (размер) как в виде строки цифр, так и в форме числа с плавающей точкой) и содержать больший объем информации, чем это допускает URL.

Данный пример представлен в форме SOAP-сообщения (Simple Object Access Protocol) - стандартной формы обмена XML-сообщениями - одной из технологий, лежащих в основе веб-сервисов. В SOAP-сообщении имя запрашиваемого сервиса и входные параметры представлены в виде отдельных XML-элементов. Рассматриваемый пример также иллюстрирует использование пространства имен XML (xmlns:), еще одного важного элемента веб-сервисов. Благодаря тому, что XML-документы поддерживают разные типы данных, сложные структуры и объединение схем, современные технологии веб-сервисов обеспечивают значительное преимущество над существующими возможностями обращения к программным приложениям посредством HTML и URL.

**Следующее поколение Сети**

Следующее поколение Сети будет основано на программно-ориентированных взаимодействиях. Веб-сервисы предполагают использовать созданные для взаимодействия людей глобальные сети совершенно в иных целях. Программно-ориентированные взаимодействия будут автоматически выполнять операции, которые ранее обязательно требовали "ручного" вмешательства:

 поиск и покупка товаров и услуг по самой выгодной цене;

 согласование заказов авиабилетов и мест в ресторане на определенную дату (планирование путешествий);

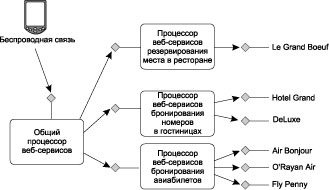
 оптимизация коммерческих операций закупки товаров, выписки счетов и доставки.

Следующее поколение Сети будет использовать программно-ориентированные сервисы для непосредственного взаимодействия с приложениями, построенными на основе любой комбинации объектов, программ и баз данных.

Веб-сервисы являются не только интерфейсом объектов, программ, связующего программного обеспечения и баз данных для доступа по Сети. Объединение ряда веб-сервисов позволяет осуществлять новые типы взаимодействия.

Веб-сервисы предоставляют новые варианты взаимодействия. Представим, например, что вы живете в Сан-Франциско и желаете зарезервировать столик в любимом парижском ресторане, а затем составить план путешествия, позволяющий оказаться там в необходимое время. В настоящее время для осуществления заказа необходимо непосредственно позвонить в ресторан, учитывая 9-часовую разницу во времени и различия в языке, а затем связаться с агентом бюро путешествий и подобрать подходящий рейс и гостиницу. Однако, используя веб-сервисы, можно спланировать обед с помощью календаря карманного компьютера (Personal Digital Assistant, PDA), а для автоматического резервирования столика на определенное время просто щелкнуть на кнопке. После выполнения резервирования веб-сервис запустит другие сервисы, которые забронируют наиболее дешевые авиабилеты и закажут номер в самом ближайшем четырехзвездочном отеле.

На рис. 1.2 представлен порядок осуществления взаимодействия веб-сервисов с карманным компьютером, подключенным по беспроводному каналу к процессору веб-сервисов, который для заказа столика в любимом ресторане использует веб-сервис этого ресторана. Процессор веб-сервисов получает запросы от функции "календарь" карманного компьютера и находит веб-сервисы, связанные с расширенной функцией календаря, такой как заказ столика. Для завершения планирования путешествия после успешного резервирования места в ресторане процессор веб-сервисов контактирует с веб-сервисами бронирования авиабилетов и номеров в гостиницах.

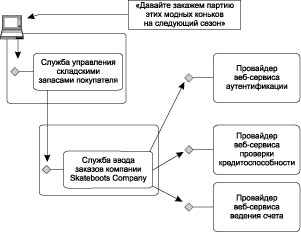


*Рис. 1.2. Приложения могут использовать веб-сервисы для резервирования места в ресторане   
и бронирования авиабилетов и номеров в гостиницах*

Веб-сервисы способны находить другие сервисы и взаимодействовать с ними. Веб-сервисы также полезны при поиске и осуществлении взаимодействия с сайтами, являющимися интерактивными системами заказов, например с системой продаж модных коньковых ботинок с выдвигающимися лезвиями.

Розничные продавцы спортивных товаров, заинтересованные в создании запасов таких ботинок, могут использовать веб-сервисы для размещения предварительных заказов, для проверки состояния выполнения заказа или размещения заказов сезонного возобновления запасов. Они также получат немедленный ответ при отсутствии коньков на складе. "Строительные" блоки веб-сервиса предоставляют стандартные компоненты конкретного приложения для компании Skateboots Company, которая не настолько велика, чтобы содержать свою собственную инфраструктуру приложений. Компания-владелец веб-сервиса также обеспечивает безопасность, гарантируя получение заказов компанией Skateboots Company только от уполномоченных оптовых покупателей, и предлагает сервисы проверки полномочий для крупных предварительных заказов. Другие компании помогут Skateboots Company создать электронный каталог и выполнить расчеты.

Целиком реализованная как веб-сервис система ввода заказов компании Skateboots зависит от Интернета, но необходимая функциональность обеспечивается за счет совместной работы ряда других веб-сервисов. На рис. 1.3 продемонстрировано, как веб-сервисы могут изменить способ построения и использования коммерческого приложения. Заинтересованный в закупке коньков оптовый покупатель вводит запрос с помощью собственной локальной системы управления складскими запасами, доступной компьютерам магазинов как веб-сервис. Локальная система управления складскими запасами контактирует через Интернет с веб-сервисом производителя и посылает ему заказ на определенное число коньков, основываясь на объеме имеющихся складских помещений и учитывая наиболее ходовые размеры ботинок.



*Рис. 1.3. Служба ввода заказов Skateboots Company содержит   
несколько других веб-сервисов*

Система ввода заказов компании Skateboots Company состоит из нескольких сервисов, включая специальную часть, которая отвечает уникальным аспектам ее продукции, и несколько торговых компонентов, выполняющих стандартные функции, например проверку полномочий пользователя, проверку кредитоспособности и отслеживание счета. Все эти сервисы предоставляются другими компаниями, специализирующимися на оказании таких услуг через Интернет.

Создание использующих веб-сервисы коммерческих приложений влечет за собой необходимость установления между этими сервисами соответствующих взаимоотношений, что реализуется с помощью любой комбинации языков программирования, операционных систем, пакетов программ, размещенных "до" или "после" брандмауэра. (Это также является способом решения сложных EAI-проблем.) Установление необходимых отношений (или потока управления) связанных веб-сервисов также приводит к автоматизации соответствующих бизнес-процессов и процедур.

Использование веб-сервисов очень выгодно с коммерческой точки зрения. За счет повсеместного распространения веб-сервисов Интернет становится более эффективным, особенно при осуществлении коммерческих сделок. Сочетая прямой доступ к программным приложениям и коммерческим документам, веб-сервисы следующего поколения Сети обеспечат полностью автоматическое взаимодействие, что позволит обращаться непосредственно к данным программ, игнорируя знакомые веб-страницы. Более того, основные компоненты веб-сервисов, скорее всего, будут предоставляться и публиковаться множеством различных компаний, специализирующихся на отдельных функциональных элементах (проверка полномочий, координация сделок, ведение счетов). Это обеспечит непосредственное взаимодействие "приложение-приложение" - принцип, лежащий в основе веб-сервисов и определяющий их суть и реализацию.

**ОБЩЕЕ ВЗАИМОПОНИМАНИЕ**

Технология веб-сервисов существует на очень высоком уровне абстракции, позволяя поддерживать множество одновременных определений, которые иногда бывают противоречивы. На простейшем уровне веб-сервисы могут восприниматься как интернет-ориентированные текстовые брокеры интеграции. Любые данные могут преобразовываться в ASCII-текст и обратно, и этот подход в течение долгого времени был общим знаменателем для систем графического вывода и систем управления базами данных. Как говорится, "если ничего не получается - преобразуй данные в текстовую форму". Ориентированные на использование текста системы также лежат в основе успешного развития Интернета, на котором базируется дополнительная абстракция веб-сервисов. Любой компьютер или операционная система может поддерживать HTML, браузеры и веб-сервисы; и при получении по сети файлов им совершенно безразлично и даже неизвестно, с каким типом прикладной системы они взаимодействуют.

То же самое можно сказать и о веб-сервисах, которые зачастую вызывают замешательство, когда разработчики традиционного, устоявшегося компьютерного окружения пытаются воспринимать веб-сервисы как отдельный тип распределенной программной среды, такой как CORBA, J2EE или .NET. Поскольку веб-сервисы гораздо более абстрактны - как посредники (преобразователи), а не как интерфейсы - так и будет продолжаться, пока они не будут приведены в соответствие с общими описаниями и соглашениями.

**Взаимодействие с веб-сервисами**

Веб-сервисы поддерживают несколько парадигм обмена сообщениями. Уровень абстракции, на котором оперируют веб-сервисы, подразумевает такие стили взаимодействия, как эмуляцию удаленного вызова процедуры (Remote Procedure Call, RPC), асинхронный обмен сообщениями, однонаправленную передачу сообщений, широковещание и публикацию/подписку. Основные СУБД, такие как Oracle, SQL Server и DB2, поддерживают анализ XML и службы преобразования, обеспечивая непосредственное взаимодействие между веб-сервисами и СУБД. Производители связующего программного обеспечения обычно также предоставляют возможность привязки веб-сервисов к своим программным системам (серверам приложений и брокерам интеграции). Следовательно, для пользователя взаимодействие с веб-сервисами может проявляться в интерактивной или пакетной форме, поддерживающей синхронную и асинхронную модели связи; а также как пользовательский интерфейс, написанный с использованием Java, VB (Visual Basic), офисных приложений, браузеров или "толстых" клиентов СУБД. Такое взаимодействие может привязываться к любому типу базовой (более низкого уровня) программной системы.

Веб-сервисы выполняют RPC- и документно-ориентированное взаимодействия. Стандарты и технологии веб-сервисов обычно подразумевают два основных типа моделей взаимодействия приложений:

 удаленный вызов процедуры (онлайновая);

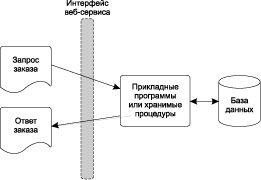
 документно-ориентированный (пакетная).

Эти два типа взаимодействия мы и рассмотрим в последующих разделах.

**RPC-ориентированные взаимодействия**

RPC-ориентированные взаимодействия удобны для краткого обмена данными. В RPC-ориентированном взаимодействии запросы веб-сервисов приобретают форму вызова метода или процедуры с соответствующими входными или выходными параметрами. В отличие от документно-ориентированного взаимодействия, RPC-ориентированное взаимодействие производит отправку документа, специально отформатированного для передачи в отдельную логическую программу или базу данных (рис. 1.4). Поскольку, например, заказ коньков в режиме реального времени зависит от их наличия на складе, программа обращается к базе данных с проверкой наличия заказываемого товара. Если получено подтверждение, то программа вернет заказчику XML-документ в формате "запрос/ответ", сообщающий о принятии заказа с его последующим исполнением.

Если поставка невозможна, будет получено сообщение об отсрочке выполнения заказа или о полном отказе от его выполнения. В отличие от документно-ориентированного стиля взаимодействия, запрос и ответ моделируются как синхронные сообщения, то есть приложение, посылающее сообщение, ждет реакции на него.

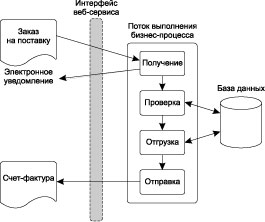


*Рис. 1.4. Веб-сервисы поддерживают интерактивный   
заказ в форме запроса/ответа*

**Документно-ориентированные взаимодействия**

Документно-ориентированные взаимодействия удобны для обмена большими объемами данных. При документно-ориентированном взаимодействии запросы веб-сервиса имеют форму завершенного XML-документа, предназначенного для обработки целиком. Например, веб-сервис, который представляет заказ на поставку (оптовый предварительный заказ на поставку коньков), должен сразу предъявлять производителю полную форму заказа (рис. 1.5). Это напоминает помещение сообщения в очередь для асинхронной обработки. Производитель обычно по электронной почте либо в какой-либо иной форме высылает заказчику подтверждение, свидетельствующее о том, что заказ принят и будет выполнен в соответствии с предопределенной последовательностью выполнения бизнес-процесса. Последовательность выполнения может содержать такие этапы, как проверка в базе данных предыдущих заказов данного оптового покупателя на предмет исчерпания кредитного лимита, согласованных объемов или графика поставок по данному заказу. В реальном потоке выполнения заказа и отправки счета, конечно же, существуют и другие этапы, но в данном примере рассматривается только заключительная стадия: отправка XML-счета для оплаты после доставки и получения заказа.

Необходимые взаимодействия определяются в договоре торговых партнеров. Документно-ориентированные взаимодействия зачастую предполагают, что использующие веб-сервисы стороны заранее согласовали порядок оформления общих документов, таких как заказ на приобретение, счет за доставку или общий счет. Эти стороны обычно идентифицируются как "торговые партнеры" или "сотрудничающие партнеры". Торговые партнеры также обыкновенно согласовывают общий поток выполнения процесса или модель взаимодействия при обмене документами, например, оговаривают необходимость подтверждения квитанции заказа на приобретение, передачу специальной информации о состоянии в ответ на запрос заказа или отправку сигнала оповещения по электронной почте после отгрузки заказа. В ходе реализации бизнес-процесса необходим обмен полными документами. Если до этого документ содержал общую, фрагментированную информацию, то теперь требуется согласованное заполнение специальных разделов, таких как цена покупки или обязательная дата доставки.



*Рис. 1.5. Веб-сервис обрабатывает полный заказ на поставку*

Для синхронных/асинхронных парадигм обмена сообщений удобны два стиля привязки. В нашем примере предварительный оптовый заказ выполняется в компании Skateboots Company с использованием заказов на покупку, представляемых в виде пакетов в соответствии с предварительными договоренностями, что помогает производителю спланировать объемы. В период активного спроса заказы на немедленное пополнение обрабатываются более интерактивными сервисами, которые зависят от заказов на пополнение с имеющихся складов и позволяют тотчас идентифицировать отложенные заказы. Таким образом, сайт Skateboots.com предоставляет веб-сервисы, поддерживающие оба основных типа взаимодействия.

**Технология веб-сервисов**

Порядок описания, поиска и взаимодействия веб-сервисов друг с другом определяют стандарты. Взаимодействующие через Интернет программы должны уметь обнаруживать друг друга, находить информацию, позволяющую им осуществить связь, понимать, какая модель контактирования должна быть применена (простая, типа "запрос/ответ", или более сложная последовательность), и договариваться об использовании таких услуг, как защита информации, подтверждение передачи сообщений и составление сделок. Некоторые из этих сервисов реализуются существующими технологиями и предлагаемыми стандартами, а другие - нет. Использующее веб-сервисы сообщество стремится удовлетворить все эти требования, но это - эволюционный процесс, как и сам Интернет. С самого начала инфраструктура и стандарты веб-сервисов подразумевали возможность расширения (так же как до них XML и HTML), что позволяет использовать их сразу же после появления новых стандартов и технологий.

Веб-сервисы требуют использования нескольких смежных XML-технологий. Для транспортировки и преобразования данных в программы и обратно веб-сервисы требуют использования нескольких смежных XML-технологий.

 Язык XML (Extensible Markup Language) - фундамент, на котором строятся веб-сервисы. Он предоставляет язык определения данных и порядок их обработки. XML представляет семейство связанных спецификаций, публикуемых и поддерживаемых интернет-консорциумом (World Wide Web Consortium, W3C) и другими организациями.

 WSDL (Web Services Description Language) - технология, основанная на XML, определяющая интерфейсы веб-сервисов, типы данных и сообщений, а также модели взаимодействия и протоколы связывания.

**НОВАЯ "СЕРЕБРЯНАЯ ПУЛЯ"?**

С точки зрения современных проблем вычислительной техники веб-сервисы иногда представляются как решение типа "серебряная пуля", исполняя роль, которая ранее отводилась Сети, реляционным базам данных, языкам программирования 4-го поколения и искусственному интеллекту. К сожалению, сами по себе веб-сервисы не в состоянии решить многие проблемы. Они представляют новый уровень (или другой способ выполнения), а не фундаментальное изменение, которое приходит на смену существующей компьютерной инфраструктуре. Этот новый технологический уровень выполняет новую функцию, но, что более важно, обеспечивает механизм интеграции, определенный на более высоком уровне абстракции.

Важность веб-сервисов заключается в том, что они могут служить "связующим мостом", а не заменой существующих технологий. Можно сказать, что новые языки программирования, такие как Visual Basic, C#, C/C++ и Java, пришли на смену старым языкам, таким как COBOL и FORTRAN, хотя множество программ на этих языках все еще используются, как и привязки веб-сервисов для них. В качестве веб-серверов веб-сервисы являются дополнением и не конфликтуют с существующими приложениями, программами и базами данных. Разработка приложений по-прежнему требует знаний Java, VB и C#. Все, что является новым, - это способ преобразования данных для передачи в приложение и обратно на основе стандартных форматов XML-данных и протоколов, что позволяет обеспечить новый уровень интеграции и взаимодействия.

При проектировании и разработке новых программ и баз данных разработчики могут учитывать веб-сервисы, но эти программы и базы данных все равно потребуют использования упаковщиков веб-сервисов. Веб-сервисы сами по себе не являются исполняемыми; они полагаются на исполняемые программы, написанные с помощью языков программирования или сценариев. Веб-сервисы определяют мощный уровень абстракции, который может применяться для межпрограммного взаимодействия на основе существующей сетевой инфраструктуры, но без этой инфраструктуры они ничего не значат.

 Протокол SOAP (Simple Object Access Protocol) - совокупность XML-технологий, определяющих "конверт" для связи веб-сервисов (привязанный к HTTP и другим транспортам), - специализированный формат для передачи XML-документов по Сети, а также соглашения RPC-взаимодействий.

 Технология UDDI (Universal Description, Discovery and Integration) - реестр веб-сервисов и механизм поиска. Он используется для хранения и упорядочения деловой информации, а также для нахождения указателей на интерфейсы веб-сервисов.

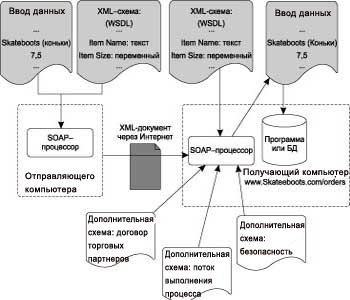
**Пример использования**

Стандарты веб-сервисов обычно используются совместно. Основные стандарты веб-сервисов используются согласованно. После обнаружения WSDL в UDDI или другом месте генерируется SOAP-сообщение для отправки на удаленный сайт.

Как видно из рис. 1.6, при предоставлении документа по адресу веб-сервиса программа использует XML-схему определенного типа (такую как WSDL), позволяющую преобразовать данные из ее входного источника (в этом примере структурированный файл) и на основе того же WSDL-файла создать экземпляр XML-документа в формате, согласованном с целевым веб-сервисом. WSDL-файл используется для определения как входного, так и выходного преобразования данных.

SOAP-процессор отправляющего компьютера преобразует данные из собственного ("родного") формата в тип данных, предопределенный в соответствии с содержащейся в WSDL-файле XML-схемой на основе таблиц преобразования для текстов, значений с плавающей точкой и других данных. Таблицы преобразования "связывают" собственные типы данных с соответствующими конкретной XML-схеме. (Стандартное преобразование типов широко используется в Java, Visual Basic, CORBA и других известных системах. Многие средства XML позволяют настраивать специальные преобразования типов.) SOAP-процессор получающего компьютера выполняет обратное преобразование данных из типов XML-схемы в собственные типы данных.

Файлы описаний веб-сервисов обычно регистрируются с помощью URL. URL, повсеместно используемый в Сети, указывает на IP-адрес, соответствующий веб-ресурсу. Схемы веб-сервисов являются одной из форм веб-ресурса, они содержатся в доступных через Интернет файлах и к ним применим тот же механизм, что используется при загрузке HTML-файлов. Главное отличие между загрузкой HTML-файла и обращением к ресурсу веб-сервиса заключается в том, что веб-сервис оперирует XML-документами, а не HTML-документами и опирается на соответствующие технологии, такие как использование схем, преобразование, проверка подлинности, что и обеспечивает поддержку удаленного соединения приложений. Но способ, согласно которому схемы веб- сервисов публикуются и загружаются, одинаков: HTTP-операция по указанному URL.



*Рис. 1.6. Веб-сервисы используют XML-документы и   
осуществляют преобразование данных*

Для проверки достоверности сообщений веб-сервисы используют XML-схемы. После получения документа реализация веб-сервиса сначала должна проанализировать XML-сообщение и удостовериться в корректности данных, выполнить проверку качества услуг (Quality-of-Service), такую как проверку политики безопасности или соглашений торговых партнеров, а затем произвести последовательность связанных с данным документом коммерческих операций. Веб-сервис на вымышленном нами сайте skateboots.com размещен в папке skateboots.com/ order, на которую и указывает URL.

Веб-сервис, доступный по данному интернет-адресу, идентифицируется с помощью публичного WSDL-файла, который может быть загружен на отправляющий компьютер и использоваться при генерации сообщения. Компания Skateboots Company также осуществляет отправку в общедоступный каталог UDDI-листинга, позволяющего клиентам находить компанию с помощью технологии UDDI. В общем случае, любой, кто хочет взаимодействовать с веб-сервисом, размещающим или контролирующим по Сети заказы для Skateboots Company, для генерации сообщения должен найти способ получения и использования WSDL-файла.

Размещенные по адресу skateboots.com программы представляют собой прослушивающий HTTP-процесс, связанный с соответствующими веб-сервисами для распознавания XML-сообщений, определенных в данном формате. Эти программы включают в себя XML-анализаторы и преобразователи. Кроме того, они осуществляют конвертацию данных SOAP-сообщения в форматы, необходимые для системы ввода заказов компании Skateboots Company.

Технологии веб-сервисов сформировались из основных структур. Этих технологий достаточно для построения, развертывания и публикации базовых веб-сервисов. На самом деле, необходим лишь базовый протокол SOAP. С момента появления веб-сервисов к ним все время добавляются другие технологии. Хотя для деловой связи, а также для создания "моста" с несовместимыми технологиями вполне достаточно базовых принципов, данная форма веб-взаимодействия тем не менее была одобрена очень быстро.

.

**WSDL: описание веб-сервисов**

Язык описания веб-сервисов (Web Services Description Language, WSDL) - это формат XML-схем, определяющий расширенную структуру описания интерфейсов веб-сервисов. WSDL первоначально был разработан компаниями Microsoft и IBM. А затем его поддержали консорциум W3C и 25 компаний. WSDL - это сердце структуры веб-сервиса. Это общий способ представления передаваемых в сообщениях типов данных, указывающий действия, которые должны быть выполнены с данным сообщением и согласно которому сообщения привязываются к сетевым транспортам.

WSDL - это XML-формат, описывающий состав веб-сервиса. WSDL предназначен для использования как в процедурно-ориентированных, так и в документно-ориентированных приложениях. Так же как и другие XML-технологии, WSDL является расширяемым языком и имеет такое количество параметров, что обеспечение совместимости при организации взаимодействия между различными реализациями может вызвать сложности. Полное взаимопонимание возможно лишь в том случае, если отправитель и получатель сообщения могут совместно использовать и одинаково интерпретировать один и тот же WSDL-файл.

WSDL в соответствии с уровнем абстрагирования состоит из трех элементов. WSDL можно разделить на три основные составляющие:

 определение типов данных;

 абстрактные операции;

 связывание сервисов.

Каждая составляющая может быть указана в различных XML-документах и импортирована в виде различных комбинаций, что позволяет создавать окончательное описание веб-сервиса. Кроме того, все составляющие могут быть представлены и в одном XML-документе. "Определение типов данных" задает структуру и содержание сообщений. "Абстрактные операции" определяют операции, которые должны быть выполнены с содержанием сообщения, а "связывание сервисов" подразумевает сетевой транспорт, который доставит сообщение по месту назначения.

На рис. 1.7 представлены составляющие WSDL, упорядоченные в соответствии с их уровнем абстракции, который определяется независимо от транспорта, особенно когда для одного сервиса используется множество транспортов. Например, к одному и тому же сервису можно обращаться посредством "SOAP поверх HTTP" или "SOAP поверх JMS". Аналогично, определения типов данных размещены в разных разделах, что позволяет использовать их различными сервисами. Основные WSDL-составляющие разделены на компоненты.



*Рис. 1.7. WSDL состоит из трех основных составляющих  
и семи компонентов*

WSDL-интерфейсы подобны интерфейсам CORBA и DCOM. Части описания содержат описания типов данных, сообщения и абстрактные операции, которые схожи с интерфейсами CORBA и DCOM. Сообщения могут состоять из нескольких секций и могут быть предназначены для использования в случае процедурно-ориентированного стиля взаимодействия, документно-ориентированного стиля или при применении обоих стилей. В зависимости от уровня абстрагирования одни и те же сообщения могут определяться и использоваться для нескольких типов портов. Как и другие части WSDL, сообщения также поддерживают компоненты расширения, в частности, для включения в себя атрибутов других сообщений.

Типы данных веб-сервисов основаны на XML-схемах, но могут быть распространены на любые другие механизмы. WSDL-описания типов данных базируются на XML-схемах, но здесь годятся для использования и другие эквивалентные или аналогичные системы определения типов данных. Например, вместо типов данных XML-схемы может применяться язык описания интерфейса (Interface Definition Language, IDL) CORBA. (Если используется иная модель определения типов, ее должны "понимать" обе стороны.)

Абстрактные операции и сообщения отображаются на специальный транспорт. Связывание сервисов отображает абстрактные сообщения и операции на специальный транспорт, такой как SOAP. Для включения сведений, характерных для SOAP и других типов отображения, используется связывание расширяемых компонентов. Абстрактные определения могут привязываться к различным физическим транспортам. Спецификация WSDL v1.1 включает примеры однонаправленного SOAP-отображения для SMTP (Simple Mail Transfer Protocol), SOAP RPC-привязки к HTTP, SOAP-привязки к методам get и post протокола HTTP, а также пример отображения для MIME (Multipurpose Internet Mail Extensions) - многокомпонентного связывания SOAP.

Пространство имен гарантирует уникальность имен WSDL-элементов. Пространство имен XML предназначено для обеспечения уникальности имен, используемых в любой из трех основных WSDL-составляющих. Конечно же, при раздельной разработке WSDL-элементов и импортировании их в один файл описанные в отдельных файлах пространства имен не должны перекрываться. Связанные схемы используются для проверки достоверности как WSDL-файла и сообщений, так и для операций, определенных в WSDL-файле.

Можно смело сказать, что по мере развития веб-сервисов WSDL, по всей видимости, будет включать множество расширений, изменений и добавлений. Как и SOAP, WSDL разработан в виде расширяемой XML-структуры, которая может быть легко модифицирована для привязки множества типов данных, определений типов сообщений, операций и транспортов. Например, рабочие группы IETF (Internet Engineering Task Force) предлагают новый стандарт протоколов - Blocks Extensible Exchange Protocol (BEEP), - позволяющий определять транспорт с установлением соединений. (В отличие от HTTP, ориентированного на связь без установления соединения, что усложняет решение проблем качества обслуживания на транспортном уровне.) Компании, интересующиеся использованием веб-сервисов во внутренних приложениях или для осуществления интеграции, могут расширить WSDL, обеспечив его привязку к более традиционным протоколам, таким как DCOM или IIOP (Internet Inter-Orb Protocol).

**SOAP: доступ к веб-сервисам**

До сего момента вы определяли данные (XML) и абстрактно представили службу, необходимую для обеспечения соединения и обработки сообщения (WSDL). Теперь же необходимо уточнить, каким способом сообщение будет пересылаться с одного узла на другой, что позволит провести его обработку на конечном компьютере.

SOAP обеспечивает механизм подключения к веб-сервисам. Спецификация протокола SOAP определяет структуру сообщения, используемого для обмена форматированными XML-данными посредством сети Интернет. Структура сообщения достаточно проста, удобна при разработке и полностью нейтральна по отношению к операционной системе, языку программирования и компьютерной платформе. Протокол SOAP предназначен для обеспечения транспорта базового уровня, на основе которого могут возводиться более сложные протоколы и способы взаимодействия.

SOAP - это XML-способ определения: какая информация должна пересылаться и как. SOAP - это базовая однонаправленная модель соединения, обеспечивающая согласованную передачу сообщения от отправителя к получателю, потенциально допускающая наличие посредников, которые могут обрабатывать часть сообщения или добавлять к нему дополнительные элементы. Спецификация SOAP содержит соглашения по преобразованию однонаправленного обмена сообщениями в соответствии с принципом "запрос/ответ", популярным при организации соединения типа RPC, а также определяет, как осуществлять передачу всего XML-документа.

SOAP включает дополнительные правила кодирования для различных типов данных, но конечные узлы SOAP-соединения могут определить собственные правила посредством конфиденциального соглашения сторон. В ходе связи зачастую используется буквенное или встроенное в XML кодирование.

Как видно из рис. 1.8, SOAP предназначена для поддержания независимого абстрактного протокола связи, обеспечивающего коммуникацию двух и более предприятий или двух и более удаленных корпоративных сайтов. Связанные системы можно построить на основе любой комбинации аппаратного и программного обеспечения, поддерживающей доступ через Интернет к таким средам, как .NET и J2EE. Существующие системы обычно также представлены множественными инфраструктурами и пакетами программ. SOAP и остальные структуры XML предоставляют возможность двум и более предприятиям, торговым площадкам или партнерам выработать общий подход по вопросу предоставления услуг в Сети.

SOAP-сообщения содержат конверт, заголовок и тело сообщения. SOAP-сообщения состоят из нескольких основных частей.

Envelope (конверт) - определяет начало и конец сообщения.

Header (заголовок) - содержит любые дополнительные атрибуты сообщения, используемые в ходе обработки сообщения как посредником, так и конечным получателем.

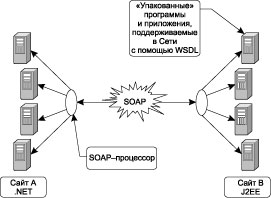
Body (тело сообщения) - содержит XML-данные, передаваемые данным сообщением.

Attachment (вложение) - состоит из одного и более документов, "прикрепленных" к основному сообщению. (Относится только к SOAP with Attachments ("SOAP с вложениями").)

RPC interaction (SOAP:RPC-взаимодействие) - определяет, как моделировать взаимодействия RPC-типа.

Encoding (кодировка) - определяет, как будут представлены простые и сложные данные, передаваемые в сообщении.

Обязательными являются только конверт и тело сообщения.



*Рис. 1.8. SOAP-сообщения осуществляют соединение удаленных узлов*

**UDDI: публикация и поиск веб-сервисов**

После того как вы определили данные в сообщении (XML), описали сервисы, которые будут принимать и обрабатывать сообщения (WSDL), и предъявили требования к средству их передачи и приема (SOAP), вам необходимы способы публикации предлагаемой вами службы, а также инструменты поиска нужных вам, но предоставляемых другими участниками услуг. Именно эти функции и выполняет реестр UDDI (universal distribution, discovery, and integration).

UDDI регистрирует и публикует определения веб-сервисов. Структура UDDI определяет модель данных в программных интерфейсах (API) XML и SOAP для регистрации и обнаружения коммерческой информации, включая веб-сервисы. Реестр UDDI был создан независимым консорциумом производителей (основанным компаниями Microsoft, IBM и Ariba), занимающимся разработкой стандарта для описания, регистрации и локализации веб-сервисов. Microsoft, IBM, Hewlett- Packard и SAP на начальном этапе предоставляют общедоступный UDDI-сервис, который концептуально копирует DNS (систему имен доменов Интернета, осуществляющую "перевод" имен Интернета в IP-адреса). На практике UDDI более похож на сервис реплицируемой базы данных, доступной через Интернет.

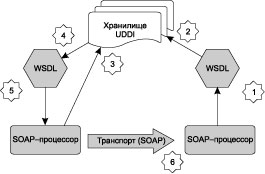
UDDI - это каталог веб-сервисов. UDDI идейно подобен каталогу "Желтые страницы". Предприниматели регистрируют свою контактную информацию, в том числе номер телефона, факса, почтовый адрес и адрес веб-сайта. Регистрация включает различные необходимые для поиска категории информации, такие как географическое положение, код отрасли промышленности, тип бизнеса и т. д. Другие предприниматели при поиске поставщиков комплектующих, служб доставки, а также аукционов и торгов могут найти нужные данные, сохраненные в UDDI. Можно также обнаружить информацию об определенном веб-сервисе обычным путем, пытаясь найти URL WSDL-файла, указывающего на веб-сервис поставщика.

Для регистрации и поиска информации UDDI использует SOAP. Для регистрации предпринимателей в UDDI служит SOAP; далее клиенты реестра для поиска информации необходимого им торгового партнера используют программный интерфейс SOAP-запроса. Первоначальный запрос может вернуть несколько соответствий, из которых может быть выбрано только одно. После выбора необходимого элемента для получения необходимой для осуществления сделки конкретной контактной информации производится заключительный вызов API.

На рис. 1.9 показано, как предприятие будет осуществлять регистрацию в UDDI сведений о веб-сервисе.

Сначала предприятие генерирует WSDL-файл, который описывает веб-сервисы, поддерживаемые SOAP-процессором (1), и с помощью API UDDI регистрирует информацию в хранилище (2). После отправки предприятием этих сведений (совместно с другой контактной информацией) элемент реестра будет содержать URL, указывающий на WSDL-файл сайта SOAP-сервера или на другой файл XML-схемы, описывающий данный веб-сервис.

После этого SOAP-процессор другого предприятия запрашивает реестр (3) для получения WSDL или другой схемы (4). Для отправки конкретной операции по определенному протоколу (6) клиент может сгенерировать соответствующее сообщение (5). Конечно же, клиент и сервер должны договориться о том, чтобы протокол был одинаковым (в данном примере "SOAP поверх HTTP"), и совместно использовать одинаковые понятия или семантические описания сервиса, который в данном примере представлен WSDL. При повсеместном распространении основных стандартов, скорее всего, общее понимание будет гарантироваться посредством WSDL.



*Рис. 1.9. Для обнаружения веб-сервиса может   
использоваться репозиторий UDDI*

# Безопасность в сетях

**Безопасность в сетях**

1. **Первые сети** использовались для обмена файлами или для совместного использования оборудования (принтеров, сканеров и т.п.) и поэтому вопросы безопасности **на этом этапе не привлекали особого внимания**.
2. Большинство проблем безопасности возникает из-за **злоумышленников (злостных или случайных)**, пытающихся извлечь пользу для себя и причинить вред другим. Причем криминальная статистика говорит, что большинство атак на информационные системы возникают изнутри системы (специалистами эксплуатирующими систему).
3. На сегодняшний день **специалист по информационной безопасности – наиболее востребованная профессия**.
4. В первом приближении вопросы безопасности могут быть разделены на 4 пересекающиеся области:

* **секретность (конфиденциальность)**;
* **аутентификация**;
* **обеспечение строгого выполнения обязательств (не возможность отказа от авторства**;
* **обеспечение целостности**.

1. **Конфиденциальность** –предотвращение попадания информации неавторизованным пользователям.
2. **Аутентификация** - проверка принадлежности субъекту предъявленного им идентификатора (подтверждение личности). Процесс аутентификации может осуществляться различным способом:

- логин и пароль;

- электронный сертификат;

- старт-карта;

- идентификация личности по биометрическим данным.

1. **Идентификация** -процесс присвоение субъектам идентификатора и сравнение идентификатора с перечнем идентификаторов.
2. **Авторизация** -процесспроверки прав субъекта на выполнение некоторых действий (говорят: пользователь авторизовался в системе). Иногдапроцесс предоставление прав доступа (говорят: администратор системы авторизует пользователя – наделяет его перечнем прав и полномочий).
3. **Обеспечение строгого выполнения обязательств** - электронная подпись документа, гарантирующая подлинность отправителя.
4. **Обеспечение целостности** – методы защиты электронных документов от модификации в процессе передаче по сети.
5. Безопасность в сетях охватывает **все уровни протоколов**: 1) на **физическом** уровне можно поместить сетевой кабель в специальные герметические трубы наполненные специальным газом (если просверлить, то утечка газа вызывает сигнал тревоги); 2)на **логическом** уровне – аппаратное сжатие, шифрование, перемешивание и пр. данных; 3) на **сетевом** уровне – firewall и brandmauer (отвергаются подозрительные пакеты); 4)на **транспортном** уровне – можно поддерживать зашифрованное соединение между процессами; 5)на **сеансовом** уровне – продолжительность действия ключей для шифров; 6) на **представительском** уровне – методы шифрования; 4) на **прикладном** уровне – процессы аутентификации.
6. На всех уровнях за исключением физического защита информации осуществляется на основе **криптографии**.
7. **Криптография (греч. скрытое письмо)** – наука о методах обеспечения **конфиденциальности (секретности)** и **аутентичности (целостности и невозможность отказа от авторства)**.
8. Родственные понятия для криптографии: **криптосистема** – семейство обратимых преобразований открытого текста; **криптоанализ** – наука, изучающая методы разрушения конфиденциальности и целостности; **криптология** = криптография + криптоанализ; **криптографическая стойкость** - способность криптографического алгоритма противостоять криптоанализу.
9. **Шифр** – побитовое или посимвольное преобразование независимое от лингвистической особенности сообщения. **Ключ** – параметр шифра, определяющий преобразование открытого текста. **Код** – заменяет слово языка другим словом или символом (американцы использовали язык индейского племени навахо (использовали носителей языка) в ходе переговоров на войне с Японией). Электронная цифровая подпись – реквизит электронного документа, предназначенный для защиты документа от подделки и позволяющий установить подлинность отправителя.

Назначение и применение ЭЦП

Цифровая подпись предназначена для аутентификации лица, подписавшего электронный документ[1]. Кроме этого, использование цифровой подписи позволяет осуществить:

Контроль целостности передаваемого документа: при любом случайном или преднамеренном изменении документа подпись станет недействительной, потому что вычислена она на основании исходного состояния документа и соответствует лишь ему.

Защиту от изменений (подделки) документа: гарантия выявления подделки при контроле целостности делает подделывание нецелесообразным в большинстве случаев.

Невозможность отказа от авторства. Так как создать корректную подпись можно, лишь зная закрытый ключ, а он должен быть известен только владельцу, то владелец не может отказаться от своей подписи под документом.

Доказательное подтверждение авторства документа: Так как создать корректную подпись можно, лишь зная закрытый ключ, а он должен быть известен только владельцу, то владелец пары ключей может доказать своё авторство подписи под документом. В зависимости от деталей определения документа могут быть подписаны такие поля, как «автор», «внесённые изменения», «метка времени» и т. д.

Все эти свойства ЭЦП позволяют использовать её для следующих целей[2]:

Декларирование товаров и услуг (таможенные декларации)

Регистрация сделок по объектам недвижимости

Использование в банковских системах

Электронная торговля и госзаказы

Контроль исполнения государственного бюджета

В системах обращения к органам власти

Для обязательной отчетности перед государственными учреждениями

Организация юридически значимого электронного документооборота

В расчетных и трейдинговых системах

1. Два фундаментальных принципа криптографии**: избыточность** (все пересылаемые данные должны содержать дополнительную информацию не используемую для понимания сообщения); **ограниченный срок действия** - сообщения не должны повторятся (фильм «Игры разума», Джон Нэш (Рассел Кроу), расшифровали предаваемые координаты).
2. Стандарты шифрования: **DES (Data Encryption Standard)** (IBM 56 бит, тройной 168 бит), **AES (Advanced DES)** (NIST,128, 192, 256 бит), **Rijndael**(128-256 бит).

Американский стандарт симметричного шифрования DES (Data Encryption Standard — стандарт шифрования данных) разработан и принят в 1977 г. При разработке считалось, что стандарт выбирается на 10—15 лет, однако решение о смене стандарта было принято только в 1997 г., т.е. через 20 лет. Причем серьезных недостатков в архитектуре алгоритма найдено не было, за исключением одного — чересчур короткого ключа.

Ключ шифрования DES имеет размер 64 бита, из которых 56 бит являются значащими, а остальные используются для контроля четности. DES - симметричная криптосистема, определенная как 16-раундовый [шифр Фейстеля](http://www.racal.ru/rsp/f.htm#Feistel), была первоначально предназначена для аппаратной реализации. Когда DES используется для передачи информации, то чтобы зашифровать и расшифровать сообщение или чтобы создать и проверить код подлинности сообщения, отправитель и получатель должны знать секретный ключ. DES может также использоваться одним пользователем, например, для шифрования файлов на жестком диске. В многопользовательской среде организовать защищенное распределение ключа сложно; идеальное решение этой проблемы предлагает криптография общего ключа.

Национальный институт стандартизации и технологий США (NIST) в 2004г. официально [признал](http://www.infoworld.com/article/04/07/29/HNdesinadequate_1.html) популярный стандарт шифрования DES недействительным и объявил, что государственный патент на его использование будет аннулирован. Причина отмены патента - недостаточная стойкость шифра DES. В 1997 г. такого размера хватало с запасом, но с тех пор компьютерная техника развивалась настолько быстро, что 56-битового ключа стало недостаточно.

Поэтому в 1997 г. было принято решение обновить стандарт, и с этой целью объявили о проведении конкурса AES (Advanced Encryption Standard — «продвинутый» стандарт шифрования) по выбору нового стандарта.

На конкурс могли быть присланы алгоритмы шифрования, разработанные как организациями, так и частными лицами в любой стране. Алгоритм-победитель этого конкурса должен был стать новым стандартом блочного симметричного шифрования США.

Для участия в конкурсе алгоритм шифрования должен был соответствовать всего двум обязательным требованиям:

* 128-битный размер блока шифруемых данных,
* не менее трех поддерживаемых алгоритмом размеров ключей шифрования: 128, 192 и 256 бит.

Кроме того, NIST предъявил большое число требований, носивших рекомендательный характер:

1.     Алгоритм должен быть стойким против криптоаналитических атак, известных на время проведения конкурса.

2.     Структура алгоритма должна быть ясной, простой и обоснованной.

3.     Должны отсутствовать слабые и эквивалентные ключи (т.е. ключи, являющиеся различными, но приводящие к одному и тому же результату шифрования).

4.     Скорость шифрования данных должна быть высокой на всех потенциальных аппаратных платформах – от 8-битных до 64-битных.

5.     Структура алгоритма должна позволять распараллеливание операций в многопроцессорных системах и аппаратных реализациях.

6.     Алгоритм должен предъявлять минимальные требования к оперативной и энергонезависимой памяти.

7.     Не должно быть ограничений для использования алгоритма.

Список участников конкурса оказался весьма разнообразен, но среди них в ходе длительного анализа был выбран победитель - алгоритм Rijndael, который по большинству критериев оказался лучшим и стал называться AES.

1. **Алгоритмы с открытым ключом**: Диффи и Хелман D(E(P))=P, E = f(D) – легко, D = ff(E) – трудно вычислить. Одностороння функция. Открытым ключом можно только зашифровать, а расшифровать сложно, для расшифровки специальный скрытый ключ. Алгоритм RSA. **Ассиметричное шифрование**.
2. **IPsec (RFC 2401-2412) –** набор протоколов длязащиты данных передаваемых с помощью IP.IPsec встроен в **IPv6** и является его неотъемлемой частью. **IPsec** состоит: 1)протоколы для защиты потока - **AH** (**Authentication Header,** гарантируется целостность, но данные не шифруются) и **ESP (Encapsulation Security Payload**, обеспечивает целостность и шифрование**);** 2) протокол обмена криптографическими ключами **IKE** **(Internet Key Exchange)**.

**1.1. Краткая историческая справка появления протокола**

В 1994 году Совет по архитектуре Интернет IAB (<http://www.iab.org/>) выпустил отчет **«Безопасность архитектуры Интернет»**. В этом документе описывались основные области применения дополнительных средств безопасности в сети Интернет, а именно защита от несанкционированного мониторинга, подмены пакетов и управления потоками данных. В числе первоочередных и наиболее важных защитных мер указывалась необходимость разработки концепции и основных механизмов обеспечения целостности и конфиденциальности потоков данных. Поскольку изменение базовых протоколов семейства TCP/IP вызвало бы полную перестройку сети Интернет, была поставлена задача обеспечения безопасности информационного обмена в открытых телекоммуникационных сетях на базе существующих протоколов. Таким образом, начала создаваться спецификация IPSec.

**1.2. Архитектура, функции, режимы работы и стандарты IPSec**

**IPSec** — это комплекс протоколов, касающихся вопросов шифрования, аутентификации и обеспечения защиты при транспортировке IP-пакетов; в его состав на данный момент входят около 20ти предложений по стандартам и 18ти RFC.

Первоначально протоколы IPSec были определены в RFC с номерами от **1825** до **1827**, принятых в **1995 году**. В **1998 году** были приняты новые редакции стандартов (**RFC с 2401** по **2412**), несовместимые с **RFC 1825—1827**. В **2005 году** была принята третья редакция, незначительно отличающаяся от предыдущей.

**Общая архитектура IPsec** описана в **RFC 4301**, **аутентифицирующий заголовок** — в **RFC 4302**, **инкапсуляция зашифрованных данных** — в **RFC 4303**. Ряд других RFC описывает другие детали IPsec, такие как применение различных алгоритмов шифрования, протоколы обмена ключами и т. п.

Большинство современных (**2007 год**) реализаций IPsec основано на **RFC 2401-2412**.

Вот наиболее использующие в настоящее время:

|  |  |  |
| --- | --- | --- |
| **RFC**  **Количество** | **Имя** | **Описание** |
| **2401** | Архитектура безопасности для Интернет-протокола | Основной документ, описывающий архитектуру IPSec. |
| **2402** | Аутентификация IP заголовка | Определяет AH протокол, используется для обеспечения целостности данных. |
| **2403** | Использование HMAC-MD5-96 в пределах ESP и AH | Описывает особенности алгоритма шифрования для использования AH и ESP называемой сообщения Сборник 5 (MD5). |
| **2404** | Использование HMAC-SHA-1-96 в пределах ESP и AH | Описывает особенности алгоритма шифрования для использования AH и ESP призвал Безопасный алгоритм хеширования 1 (SHA-1). |
| **2406** | IP Encapsulating безопасности полезной нагрузки (ESP) | Описывает IPSec Инкапсуляция безопасности полезной нагрузки (ESP) протокол, который обеспечивает шифрование данных для сохранения конфиденциальности. |
| **2408** | Интернет Ассоциация безопасности и управления ключами протокола (ISAKMP) | Определяет методы для обмена ключами и переговорных безопасности ассоциаций. |
| **2409** | IKE | Описывает IKE протокол используется для ведения переговоров безопасности ассоциациями и обмен ключами между устройствами для безопасной связи. Основываясь на ISAKMP и Oakley. |
| **2412** | Oakley ключевые определения протокола | Описывает общий протокол для обмена ключом. |

Средства безопасности для IP описываются семейством спецификаций IPsec. Следовательно, IPSec работает на третьем уровне модели OSI – сетевом, в результате чего защита передаваемых IP-пакетов становится прозрачной для сетевых приложений т.о. IPSec призван обеспечить низкоуровневую защиту.

**Спецификация IPSec** разрабатывается Рабочей группой IP Security Protocol IETF (http://www.ietf.org/).

**Основными функциями IPSec являются:**

* Обеспечение конфиденциальности.

Отправитель должен иметь возможность шифровать пакеты перед тем, как передавать их по сети.

* Обеспечение целостности.

Получатель должен иметь возможность аутентифицировать стороны, учавствующие в процессе обмена информацией, и пакеты IPSec, посылаемые этими сторонами, дабы быть уверенным в том, что передаваемые данные не были изменены в пути.

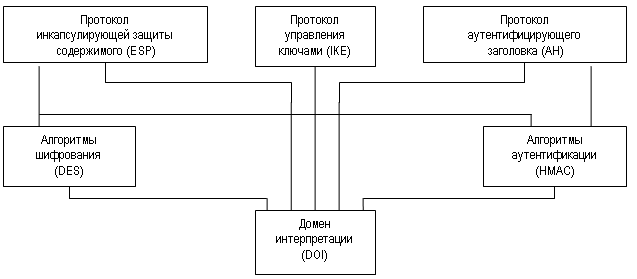
* Обеспечение защиты от воспроизведения пакетов.

Получатель должен иметь возможность обнаруживать и отбрасывать воспроизведенные пакеты, исключая таким образом проведение атак внедрения посредника.

**В комплекс спецификации IPSec входят следующие протоколы и стандарты:**

* **IKE**, обеспечивающий аутентификацию сторон, согласование параметов ассоциаций защиты (SA), а так же выбор ключей шифрования.
* **AH**, обеспечивающий аутентификацию пакетов и выявление их воспроизведения.
* **ESP** - обеспечивает конфиденциальность, аутентификацию источника и целостность данных, а также сервис защиты от воспроизведения пакетов.
* **HMAC** - механизм аутентификации сообщений с использованием хэш- функций.
* **DES**, 3DES – стандарты шифрования данных.

Взаимосвязи между протоколами представлены на рисунке 1:



**Рис 1**. Архитектура IPSec

Архитектура IPSec специфицирована в документе Security Architecture for the Internet Protocol (2401). Ее основные составляющие представлены в соответствии с рисунком 1. Это, прежде всего протоколы обеспечения аутентичности (протокол аутентифицирующего заголовка - AH) и конфиденциальности (протокол инкапсулирующей защиты содержимого - ESP), а также механизмы управления ключами (IKE). На более низком архитектурном уровне располагаются конкретные алгоритмы шифрования и аутентификации. Наконец, роль фундамента выполняет так называемый домен интерпретации (DOI), являющийся, по сути, базой данных, хранящей сведения об алгоритмах, их параметрах, протокольных идентификаторах.

**Режимы работы IPSec**

Существует два режима работы IPsec: **транспортный режим** и **туннельный режим**.

В транспортном режиме шифруется только информативная часть IP-пакета. Заголовок IP-пакета не изменяется (не шифруется). Транспортный режим как правило используется для установления соединения между хостами. Он может также использоваться между шлюзами, для защиты туннелей, организованных каким-нибудь другим способом (IP tunnel и др.).

В туннельном режиме IP-пакет шифруется целиком. Для того чтобы его можно было передать по сети, он помещается в другой IP-пакет. По существу, это защищённый IP-туннель. Туннельный режим может использоваться для подключения удалённых компьютеров к виртуальной частной сети или для организации безопасной передачи данных через открытые каналы связи (например, Интернет).

Режимы IPSec не являются взаимоисключающими. На одном и том же узле некоторые SA могут использовать транспортный режим, а другие — туннельный.

## 1.3. Принципы работы

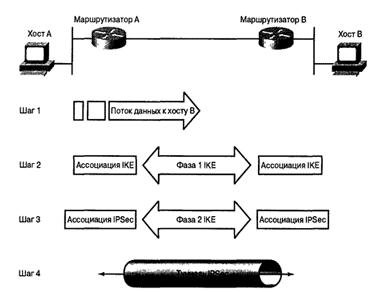
Целостность и конфиденциальность данных посредством IPSec обеспечивается за счет реализации механизмов аутентификации сторон и пакетов, а также шифрования, которые, в свою очередь, требуют предварительного согласования сторонами параметров информационного обмена – так называемого «контекста безопасности»- используемых криптографических алгоритмов, протоколов управления ключами и их параметров.

**Спецификация IPSec** является алгоритмонезависимой, то есть предусматривает возможность использования сторонами нескольких протоколов и параметров аутентификации и шифрования, различных схем распределения ключей. Результатом согласования контекста безопасности является индекс параметров безопасности (SPI), который представляет собой указатель на определенный элемент базы данных политики безопасности (SPD). На основании информации, содержащейся в SPD, для пакета данных может быть выбрано одно из трёх действий: отбросить пакет, обработать пакет без вмешательства IPSec или обработать пакет с помощью IPSec. В последнем случае в SPD также содержится указатель на SA, который необходимо использовать в процессе обмена данными.

В свою очередь SA – это согласованная политика или способ обработки данных, которыми обмениваются стороны. Два устройства с каждой стороны одной ассоциации защиты содержат данные о протоколах, алгоритмах и ключах, используемых в SA. Отдельно взятая ассоциация защиты используется для связи только в одном направлении, для двунаправленной связи их требуется две. Каждый SA реализует один режим и протокол, поэтому в случае, когда для анализа некоторого пакета требуется применить два протокола (например, AH и ESP, хотя этот случай и является очень редким), требуется создание двух различных SA.

**Процесс функционирования IPSec в общем случае можно разбить на следующие шаги:**

1. Инициация IPSec. Приложение, трафику которого требуется защита IPSec, начинает процесс обмена данными IКЕ-протокола.
2. Первая фаза IKE, в которой выполняется аутентификация сторон, и ведутся переговоры о параметрах ассоциаций защиты IKE, в результате чего создается защищенный канал для обмена параметрами защиты IPSec в ходе второй фазы IKE.
3. Вторая фаза IKE, по ходу которой ведутся переговоры о параметрах ассоциации защиты IPSec, создаются соответствующие SA для устройств обеих сторон.
4. Собственно передача данных, т.е. процесс обмена данными, основывающийся на параметрах IPSec и ключах, хранимых в SPD.
5. Завершение работы IPSec. SA IPSec завершают свою работу в результате либо их удаления, либо превышения лимита времени их существования.



**1.4. Оценка IPSec**

Спецификация IPSec получила неоднозначную оценку со стороны специалистов**.**

**С одной стороны**, отмечается, что протокол IPSec является лучшим среди всех других протоколов защиты передаваемых по сети данных, разработанных ранее (включая разработанный Microsoft PPTP).

**По мнению** **другой стороны**, присутствует чрезмерная сложность и избыточность протокола. Так, Niels Ferguson и Bruce Schneier в своей работе "A Cryptographic Evaluation of IPsec" отмечают, что они обнаружили серьезные проблемы безопасности практически во всех главных компонентах IPSec. Эти авторы также отмечают, что набор протоколов требует серьезной доработки для того, чтобы он обеспечивал хороший уровень безопасности.

**ГЛАВА 2. IKE**

**IKE**— протокол обмена Интернет-ключами. Этот протокол предусматривает метод аутентификации для защиты данных и каналов связи и позволяет кодировать заголовки и содержимое пакетов с помощью ключа, обеспечивая практически абсолютную безопасность линии связи. В соответствии с протоколом IKE пакеты шифруются с помощью секретного ключа, заранее известного обеим сторонам, или с помощью стандартного открытого ключа.

**Протокол IKE** является гибридным протоколом, обеспечивающим специальный сервис для IPSec, а именно аутентификацию сторон IPSec, согласование параметров ассоциаций защиты IKE и IPSec, а также выбор ключей для алгоритмов шифрования, используемых в рамках IPSec.

Протокол IKE опирается на протоколы **ISAKMP** (-- протокол управления ассоциациями и ключами защиты в сети Интернет) и Oakley, которые применяются для управления процессом создания и обработки ключей шифрования, используемых в преобразованиях IPSec.

IKE поддерживает набор различных примитивных функций для использования в протоколах. Среди них можно выделить хэш-функцию.

**Хэш-функция** - это функция, устойчивая к коллизиям. Под устойчивостью к коллизиям понимается тот факт, что невозможно найти два разных сообщения m1 и m2, таких, что

H(m1)=H(m2), где H – хэш-функция.

Технология IKE реализована большинством крупнейших производителей (Cisco Systems, Check Point, IBM, Microsoft) и прошла апробацию/внедрение в большом количестве системных проектов.

Электронная Беларусь

С 2003 года действует в Республике Беларусь государственная программа поэтапного перехода к информационному обществу «Электронная Беларусь», которая предполагает создание в республике единой инфраструктуры с широким использованием информационных технологий в государственном управлении, народном хозяйстве, образовании, медицине, торговле и других сторонах жизни общества. В настоящий момент программы состоит из девяти направлений и включает порядка 100 проектов.

Обратной стороной информатизации общества является проблема защиты информации. Утечка, искажение или разрушение информации может негативно отразиться на деятельности государственных органов управления, предприятий и организаций, нарушить права физических лиц и т.п. В условиях информационного общества одной из основных задач стоящих перед государством является построение инфраструктуры национальной информационной безопасности.

**Инфраструктура национальной информационной безопасности.** Инфраструктура национальной информационной безопасности (ИНИБ)представляет собой комплекс взаимосвязанных обслуживающих систем различной природы (организационных, правовых, информационных и т.п.) обеспечивающих основу для решения задачи информационной безопасности (ИБ) в масштабах страны. ИНИБ является составной частью общей системы национальной безопасности страны. На рис. 1 представлена обобщенная схема ИНИБ, исторически сложившаяся в большинстве развитых стран. На рисунке изображены основные элементы ИНИБ (замкнутые фигуры, различной геометрии) и подписанные стрелки, указывающие тип взаимоотношения между парами элементов.

Основой любой ИНИБ является законодательная и нормативно-правовая база обеспечения информационной безопасности (ЗБИБ), являющаяся выражением государственной политики в этой сфере. Государство исходит из того, что информационные ресурсы являются объектами собственности, участвующими в хозяйственном обороте. Законы и правовые акты ЗБИБ наделяют определенные органы государственной власти полномочиями осуществлять государственное регулирование, контроль (мониторинг) и управление в сфере информационной защиты, а также определяют взаимоотношение субъектов информационной деятельности. Технические нормативные акты представляют собой стандарты и технические регламенты, которые могут (или обязаны) использовать субъекты информационной деятельности. Ответственность субъектов информационной деятельности за нарушение законов в сфере информационной безопасности определяется Уголовным и Административным кодексами.

Условно субъекты в сфере информационной безопасности можно разбить на три группы: государственные учреждения, осуществляющие государственное регулирование в этой сфере (группа Г); разработчики технических средств защиты информации (группа Р) и пользователи (группа П). Очевидно, что все эти группы пересекаются.

Полномочия и деятельность субъектов группы Г, определена законом или специальными положениями. Группа Г координирует и лицензирует всю деятельность по технической защите информации, разрабатывает (или участвует в подготовке) нормативных правовых актов, осуществляет экспертизу (сертификацию) технических средств информационной безопасности. Кроме того, субъекты этой группы осуществляют управление государственными информационными системами, связанными с информационной безопасностью (например, удостоверяющие и регистрационные центры).

Основной деятельностью субъекты группы P является разработка технических средств защиты информации. Свою деятельность они осуществляют на основе лицензии, а все созданные ими средства защиты должны пройти сертификацию, прежде, чем они будут применяться пользователями. Сертификация – это процедура подтверждения соответствия продукции определенным техническим нормативным документам (стандартам).

Пользователи информационных систем (группа П) представляют самую многочисленную группу субъектов в сфере информационной безопасности. Субъекты этой группы применяют средства технической защиты информации для защиты собственных информационных ресурсов и взаимодействуют с системами защиты других субъектов. В своей деятельности они могут использовать государственные автоматизированные системы, обеспечивающие сервисные услуги в области технической защиты информации.

Исторически первой ИНИБ сложилась в США. Первый закон о защите информации датируется 1906 годом, а на сегодняшний день ЗБИБ США насчитывает их более 500 [5, 6].

Появление глобальных информационных систем привело к необходимости выработки единого подхода к проблемам информационной безопасности на международном уровне.

Первый важный шаг, который сделан в этом направлении, является разработка международных стандартов по информационной безопасности.

По понятным причинам основой международных стандартов по информационной безопасности стали стандарты, разработанные стандартизирующими организациями ведущих индустриальных стран (прежде всего США и Великобритании). Нормативная документация национальной ЗБИБ, как правило, являются гармонизированными международными стандартами.

**Международные стандарты по информационной безопасности.** Основой для большинства международных стандартов информационной безопасности (таблица 1, столбец «ISO/IEC») служат нормативные документы, разработанные национальными стандартизирующими организациями ведущих индустриальных стран, профессиональными техническими организациями, международными консорциумами или ведущими в области информационных технологий компаниями.

Таблица 1. Основные международные стандарты информационной безопасности

|  |  |  |  |
| --- | --- | --- | --- |
|  | Группы стандартов | ISO/IEC | СТБ |
| 1 | Терминология, общие понятия | 2383-8; 10164-7,8; 10181-1, 2,3,4,5,6,7; 10745; 24767-1; 15443-1, 2,3; 9798-1; 15292; 15816; 18014-1, 2, 3 | 34.101.27, 30 |
| 2 | Требования безопасности,  критерии и методологии оценки, методики испытаний | 15408-1,2.3; 19791; 15446; 18045; 19790; 18045; 24759; 15446 | 34.101.1, 2, 3, 8, 9, 10, 12, 13, 15 |
| 3 | Методы и алгоритмы шифрования, цифровая подпись | 9797-1,2; 10116; 10118-1,2,3,4; 13888-2, 3; 15946-1; 18031, 18032, 18033-1,2,3,4; 9796-2, 3; 9798-2, 3, 4, 5; 14888-1,2,3; 15945; 9594-8 | П 34.101.27, 31; П ISO/IEC 10118-3; П ISO/IEC 18033-1, 3; 34.101.24. 25, 26, 31;  1176-1, 2;  ГОСТ 28147-89 |
| 4 | Управление ключами | 11770-1,2,3, 4 |  |
| 5 | Защита сетевых технологий | 18028-1,2,3,4,5 |  |
| 6 | Организация информационной безопасностью, управление безопасностью, рисками и защитой информации | 13335-1, 27001, 27002, 27005, 27006, 18043,18044, 21827; 24762, 14516; 15447; | П ISO/IEC 27001 |
| 7 | Специализированные стандарты | 9579(SQL), 14762, 29341-13-10,11(UPnP); 15067-4(HES) |  |

Исторически первым (1983 г.) стандартом, сформулировавшим критерии безопасности и получившим широкое распространение, стал стандарт Министерства обороны и Национального комитета компьютерной безопасности США «Критерии оценки доверенных компьютерных систем» (Trusted Computer System Evaluation, TCSEC), чаще всего называемый по цвету обложки «Оранжевой книгой». Стандарт определяет четыре «уровня доверия» к компьютерным системам: D, C , B, A ( перечислены в порядке усиления требований). Уровни С и B подразделяются на классы: C1, C2 и B1, B2, B3. Европейской переработкой этого стандарта является документ «Гармонизированные критерии Европейских стран» (Information Technology Security Evaluation Criteria, ITSEC), опубликованный в 1991 г. от имени соответствующих органов четырех стран: Франции, Германии, Нидерландов и Великобритании.

Часто в литературе упоминается федеральный стандарт США FIPS 140-2 «Требования для безопасности криптографических модулей» (Security requirements for cryptographic modules), опубликованный в 2001 г., заменивший действующий c 1994 г. аналогичный стандарт FIPS 140-1. Для оценки уровня безопасности стандарт использует специальную модель – криптографический модуль, представляющий собой набор программных и (или) аппаратных средств, заключенных в пределах явно определенного и непрерывного периметра.

Наиболее известным международным проектом в области оценки безопасности, результаты которого послужили основой международного стандарта ISO/IEC 15408, является проект «Общие критерии оценки безопасности информационных технологий» (Common Criteria IT Security Evaluation, CC), более известный под коротким наименованием «Общие критерии» [5]. Проект стартовал в 1993 году по инициативе правительственных организаций шести стран: Канады, США, Великобритании, Германии, Нидерландов и Франции. Первоисточниками для проекта авторы называют документы перечисленные выше. Стандарт предусматривает семь уровней безопасности. Все требования к информационной безопасности разбиты на два вида: функциональные и требования доверия. Модель требований представляется в виде иерархии: класс-семейство-компонент элемент. Требования безопасности оформляются в виде специальных документов «Профиля безопасности» (для семейства информационных систем) или «Задания по безопасности» для конкретной системы.

Международный стандарт ISO/IEC 15408: 1999 фактически совпадает с версией 2.1 отчета «Общие критерии».

Стандарты менеджмента, аудита и сертификации информационной безопасности излагаются в международных стандартах ISO/IEC 27001 ISO/IEC 27002, ISO/IEC 27005 и ISO/IEC 27006. Все они являются развитием британского национального стандарта BS 7799. Важным является то, что эти стандарты совместимы с известными стандартами ISO 9000, ISO 9001 и ISO 14001, регламентирующими менеджмент качества на предприятии.

Большое влияние на стандарты безопасности оказывают рекомендации одной старейших стандартизирующих организаций в области телекоммуникаций и радио – Международного союза электросвязи (International Telecommunication Union, ITU). Рекомендации X.800 («Архитектура безопасности для взаимодействия открытых систем»), серия X.500 («Служба директориев») лежат в основе стандартов безопасности распределенных систем.

Стандарты безопасности Интернет разрабатываются группой IETF (Internet Engineering Task Force) сообщества Интернет (ISOC). Наиболее применяемыми являются следующие спецификации:

- IKE/IPSec (RFC 2401-2412, 2451) для протокола IP версий 4 и 6;

-TLS (RFC 2246) для протокола транспортного уровня;

- GSS-API (RFC 2744) – обобщенный прикладной интерфейс службы безопасности;

- протокол Kerberos (RFC 1510, 1964) для аутентификации в разнородной распределенной среде.

Международные стандарты представляют собой исчерпывающую основу для построения систем информационной безопасности. Национальные стандартизирующие организации в основном идут по пути гармонизации (адаптации) в международных стандартов. Процесс гармонизации определяется руководством ISO/IEC 2 и, как правило, сводится к переводу, изменению шифра и (или) формы.

**Инфраструктура национальной информационной безопасности республики Беларусь.** Инфраструктура национальной информационной безопасности республики Беларусь собственное наполнение.

**Законы и правовые акты о защите информации в республике Беларусь**. ЗБИБ Беларуси основывается на двух законах: «Об информатизации» и «Об электронном документе».

Закон «Об информатизации» принят 6 сентября 1995 года, а 9 октября 2008 года Палатой представителей Национального собрания Республики Беларусь был принят проект Закона Республики Беларусь «Об информации, информатизации и защите информации», внесенный Советом Министров Республики Беларусь и предполагающий отмену закона 1995 года [3]. Закон определяет процедуры управления и регулирования в сфере защиты информации.

Закон «Об электронном документе» был принят 10 января 2000 года, а 10 сентября 2008 года в Совет Министров Республики Беларусь представлен проект закона «Об электронном документе и электронной цифровой подписи» [2]. Этот закон «устанавливает правовые основы применения электронных документов, определяет основные требования, предъявляемые к электронным документам, а также права, обязанности и ответственность участников правоотношений, возникающих в сфере обращения электронных документов». Кроме того, закон определяет понятие электронной подписи, как неотъемлемую часть электронного документа.

**Государственное регулирование и управление в сфере защиты информации**. В соответствии со статьей 8 нового закона «Государственное регулирование и управление в сфере информации, информатизации и защиты информации осуществляется Президентом Республики Беларусь, Советом Министров Республики Беларусь, Оперативно-аналитическим центром при Президенте Республики Беларусь, Министерством связи и информатизации Республики Беларусь, Национальной академией наук Беларуси и иными государственными органами в соответствии с компетенцией определенной настоящим Законом и иными актами законодательства Республики Беларусь».

Статьи 9-14 разграничивают полномочия перечисленных государственных органов управления.

Координация и лицензирование деятельности по защите информации, а также организация и проведение работ по технической защите информации в национальном сегменте (BY) возложена на Оперативно-аналитический центр при Президенте Республики Беларусь (ОАЦ).

В соответствии с Указом Президента от 28.11.2000, № 639 головной научно-исследовательской организацией Республики Беларусь в области защиты информации является Научно-производственное республиканское предприятие «Научно-исследовательский институт технической защиты информации» (НИИ ТЗИ) [4]. Эта организация обеспечивает координацию в области научных, исследовательских, методических и практических работ по технической защите информационных систем в интересах министерств, ведомств и организаций Республики Беларусь.

Государственное научное учреждение «Объединенный институт проблем информатики» Национальной Академии Наук Беларуси привлекается ОАЦ и НИИ ТЗИ для выполнения работ по разработке проектов нормативных актов в сфере защиты информации, а также исследовательских работ в области методологии и оценке эффективности защиты информации.

**Лицензирование в сфере технической защиты информации в Республике Беларусь**. Порядок лицензирования и виды лицензируемой деятельности в сфере защиты информации определяется «Положением о лицензировании деятельности по технической защите информации, в том числе криптографическим методам, включая применение электронной подписи» утвержденным постановлением Совета Министров Республики Беларусь № 1374 от 20.10.2003 [1].

В соответствии с этим Положением выдачу и учет, лицензий на осуществление деятельности по технической защите информации осуществляет ОАЦ. Лицензируемой является практически любая деятельность. связанная с проектированием, разработкой, установкой и оказанием услуг в области технической защиты информации.

**Административная ответственность в сфере защиты информации в Республике Беларусь.** Административный кодекс Республики Беларусь предусматривает ответственность за следующие правонарушения в сфере защиты информации (глава 22 Кодекса):

* самовольное использование сетей электросвязи;
* несанкционированный доступ к компьютерной информации;
* нарушение правил защиты информации;
* незаконная деятельность в области защиты информации.

**Уголовная ответственность в сфере защиты информации в Республике Беларусь**. Уголовный кодекс Республики Беларусь предусматривает уголовную ответственность за следующие преступления против информационной безопасности (Раздел XII Уголовного кодека Республики Беларусь):

* несанкционированный доступ к компьютерной информации;
* модификация компьютерной информации;
* компьютерный саботаж;
* неправомерное завладение компьютерной информацией;
* изготовление либо сбыт специальных средств для получения неправомерного доступа к компьютерной системе или сети;
* разработка, использование либо распространение вредоносных программ;
* нарушение правил эксплуатации компьютерной системы или сети.

**Технические нормативные акты в сфере защиты информации в Республике Беларусь**. Основные стандарты в сфере защиты информации являются гармонизированными международными стандартами (столбец «СТБ» таблицы 1) и, судя по всему, эта тенденция будет продолжаться. Исключением является ГОСТ 28147-89, «Система обработки информации. Защита криптографическая. Алгоритм криптографический» успешно применяемый с 1989 года.

**Заключение.** В республике Беларусь планомерно ведется работа, направленная на создание условий информатизации общественной жизни [7].

Важнейшим этапом на этом пути является создание инфраструктуры национальной информационной безопасности. В основном сформирована законодательная основа, определены органы управления и регулирования, созданы механизмы контроля. Ясно, что существующая на сегодняшний день инфраструктура не является совершенной, но предпосылки для ее развития есть.

Существенным тормозом развития информационных технологий является отсутствие национальных стандартов. Но и здесь в последнее время наметился существенный прогресс. Гармонизация международных стандартов в области защиты информации будет способствовать интеграции Республики Беларусь в мировое информационное пространство.

Построение инфраструктуры национальной информационной безопасности формулирует ряд новых требований для автоматизированных систем и, по сути, создает новую среду для их функционирования. Все это потребует переосмысление методологии построения новых автоматизированных систем и реижиниринга уже работающих.

1. **TLS = TLS Record Protocol + TLS Handshake Protocol.** Три фазы: 1)выбор алгоритма шифрования; 2) обмен ключами или аутентификация на основе сертификатов; 3) передача данных на основе симметричного шифрования (AES, DES, ...)
2. **TLS Handshake Protocol** - протокол установки соединения: 1) клиент подключается к серверу и просит защищенное соединение и предоставляет список поддерживаемых им алгоритмов шифрование и хеш-функций; 2)сервер выбирает алгоритм шифрования и сообщает о своем выборе клиенту; 3)сервер отправляет клиенту сертификат (имя сервера, имя доверенного центра сертификации и открытый ключ сервера); 3)клиент связывается (не обязательно) с сервером доверенного центра сертификации и подтверждает аутентичность переданного сертификата (RFC 2459, X.509); 4)клиент шифрует случайную последовательность (ключ для симметричного шифрования) открытым ключом; 5) сервер расшифровывает сообщение скрытым ключом; 6) ключ для симметричного шифрования используется для обмена данными.
3. **TLS Record Protocol –** обмен зашифрованными данными
4. **HTTPS –** расширение HTTP, поддерживающее шифрование; **HTTPS** – это HTTP над SSL/TSL, 443 порт
5. **DNSsec**
6. **Kerberos (RFC 1510, 4210) - протокол аутентификации MIT, DES. GSS-API**

**Kerberos** — [сетевой протокол](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [аутентификации](http://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F), позволяющий безопасно передавать данные через незащищённые [сети](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C) для безопасной [идентификации](http://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F). Также является набором бесплатного [ПО](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) от [Массачусетского технологического института](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B0%D1%87%D1%83%D1%81%D0%B5%D1%82%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82) (Massachusetts Institute of Technology (MIT)), разработавшего этот протокол. Её организация направлена в первую очередь на [клиент-серверную](http://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_%C2%AB%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%C2%BB) модель и обеспечивает взаимную [аутентификацию](http://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) — оба [пользователя](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C) через [сервер](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80) подтверждают [личности](http://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C) друг друга. [Сообщения](http://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B5), отправляемые через протокол Kerberos, защищены от прослушивания и атак повторного воспроизведения.

Kerberos является одним из вариантов [протокола Нидхема-Шрёдера](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%9D%D0%B8%D0%B4%D1%85%D0%B5%D0%BC%D0%B0%E2%80%94%D0%A8%D1%80%D1%91%D0%B4%D0%B5%D1%80%D0%B0)[[1]](http://ru.wikipedia.org/wiki/Kerberos#cite_note-0), основан на [симметричной криптосистеме](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) и требует третье доверенное лицо ([сервер](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80)). Расширение Kerberos позволяет использовать [открытые ключи](http://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC) в процессе [аутентификации](http://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F).

# Вызов удаленных процедур (RPC)

**Концепция удаленного вызова процедур**

Идея вызова удаленных процедур *(Remote Procedure Call - RPC)* состоит в расширении хорошо известного и понятного механизма передачи управления и данных внутри программы, выполняющейся на одной машине, на передачу управления и данных через сеть. Средства удаленного вызова процедур предназначены для облегчения организации распределенных вычислений. Наибольшая эффективность использования RPC достигается в тех приложениях, в которых существует интерактивная связь между удаленными компонентами с небольшим временем ответов и относительно малым количеством передаваемых данных. Такие приложения называются RPC-ориентированными.

Характерными чертами вызова локальных процедур являются:

* Асимметричность, то есть одна из взаимодействующих сторон является инициатором;
* Синхронность, то есть выполнение вызывающей процедуры при останавливается с момента выдачи запроса и возобновляется только после возврата из вызываемой процедуры.

Реализация удаленных вызовов существенно сложнее реализации вызовов локальных процедур. Начнем с того, что поскольку вызывающая и вызываемая процедуры выполняются на разных машинах, то они имеют разные адресные пространства, и это создает проблемы при передаче параметров и результатов, особенно если машины не идентичны. Так как RPC не может рассчитывать на разделяемую память, то это означает, что параметры RPC не должны содержать указателей на ячейки нестековой памяти и что значения параметров должны копироваться с одного компьютера на другой. Следующим отличием RPC от локального вызова является то, что он обязательно использует нижележащую систему связи, однако это не должно быть явно видно ни в определении процедур, ни в самих процедурах. Удаленность вносит дополнительные проблемы. Выполнение вызывающей программы и вызываемой локальной процедуры в одной машине реализуется в рамках единого процесса. Но в реализации RPC участвуют как минимум два процесса - по одному в каждой машине. В случае, если один из них аварийно завершится, могут возникнуть следующие ситуации: при аварии вызывающей процедуры удаленно вызванные процедуры станут "осиротевшими", а при аварийном завершении удаленных процедур станут "обездоленными родителями" вызывающие процедуры, которые будут безрезультатно ожидать ответа от удаленных процедур.

Кроме того, существует ряд проблем, связанных с неоднородностью языков программирования и операционных сред: структуры данных и структуры вызова процедур, поддерживаемые в каком-либо одном языке программирования, не поддерживаются точно так же во всех других языках.

Эти и некоторые другие проблемы решает широко распространенная технология RPC, лежащая в основе многих распределенных операционных систем.

**Базовые операции RPC**

Чтобы понять работу RPC, рассмотрим вначале выполнение вызова локальной процедуры в обычной машине, работающей автономно. Пусть это, например, будет системный вызов

count=read (fd,buf,nbytes);

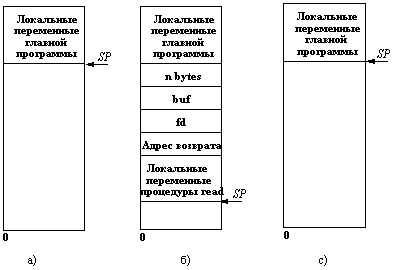
где fd - целое число,   
buf - массив символов,   
nbytes - целое число.

Чтобы осуществить вызов, вызывающая процедура заталкивает параметры в стек в обратном порядке (рисунок 3.1). После того, как вызов read выполнен, он помещает возвращаемое значение в регистр, перемещает адрес возврата и возвращает управление вызывающей процедуре, которая выбирает параметры из стека, возвращая его в исходное состояние. Заметим, что в языке С параметры могут вызываться или по ссылке (by name), или по значению (by value). По отношению к вызываемой процедуре параметры-значения являются инициализируемыми локальными переменными. Вызываемая процедура может изменить их, и это не повлияет на значение оригиналов этих переменных в вызывающей процедуре.

Если в вызываемую процедуру передается указатель на переменную, то изменение значения этой переменной вызываемой процедурой влечет изменение значения этой переменной и для вызывающей процедуры. Этот факт весьма существенен для RPC.

Существует также другой механизм передачи параметров, который не используется в языке С. Он называется call-by-copy/restore и состоит в необходимости копирования вызывающей программой переменных в стек в виде значений, а затем копирования назад после выполнения вызова поверх оригинальных значений вызывающей процедуры.

Решение о том, какой механизм передачи параметров использовать, принимается разработчиками языка. Иногда это зависит от типа передаваемых данных. В языке С, например, целые и другие скалярные данные всегда передаются по значению, а массивы - по ссылке.



*Рис. 3.1. а) Стек до выполнения вызова read;   
б) Стек во время выполнения процедуры;   
в) Стек после возврата в вызывающую программу*

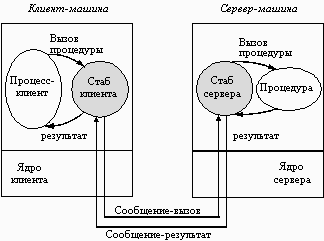
Идея, положенная в основу RPC, состоит в том, чтобы сделать вызов удаленной процедуры выглядящим по возможности также, как и вызов локальной процедуры. Другими словами - сделать RPC прозрачным: вызывающей процедуре не требуется знать, что вызываемая процедура находится на другой машине, и наоборот.

RPC достигает прозрачности следующим путем. Когда вызываемая процедура действительно является удаленной, в библиотеку помещается вместо локальной процедуры другая версия процедуры, называемая клиентским стабом (stub - заглушка). Подобно оригинальной процедуре, стаб вызывается с использованием вызывающей последовательности (как на рисунке 3.1), так же происходит прерывание при обращении к ядру. Только в отличие от оригинальной процедуры он не помещает параметры в регистры и не запрашивает у ядра данные, вместо этого он формирует сообщение для отправки ядру удаленной машины.

**Этапы выполнения RPC**

Взаимодействие программных компонентов при выполнении удаленного вызова процедуры иллюстрируется рисунком 3.2. После того, как клиентский стаб был вызван программой-клиентом, его первой задачей является заполнение буфера отправляемым сообщением. В некоторых системах клиентский стаб имеет единственный буфер фиксированной длины, заполняемый каждый раз с самого начала при поступлении каждого нового запроса. В других системах буфер сообщения представляет собой пул буферов для отдельных полей сообщения, причем некоторые из этих буферов уже заполнены. Этот метод особенно подходит для тех случаев, когда пакет имеет формат, состоящий из большого числа полей, но значения многих из этих полей не меняются от вызова к вызову.

Затем параметры должны быть преобразованы в соответствующий формат и вставлены в буфер сообщения. К этому моменту сообщение готово к передаче, поэтому выполняется прерывание по вызову ядра.



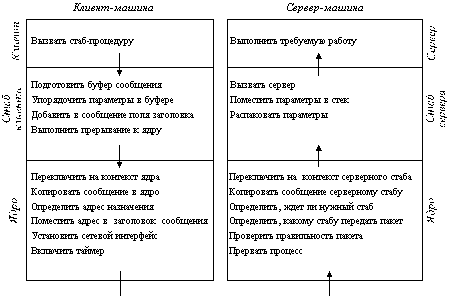
*Рис. 3.2. Remote Procedure Call*

Когда ядро получает управление, оно переключает контексты, сохраняет регистры процессора и карту памяти (дескрипторы страниц), устанавливает новую карту памяти, которая будет использоваться для работы в режиме ядра. Поскольку контексты ядра и пользователя различаются, ядро должно точно скопировать сообщение в свое собственное адресное пространство, так, чтобы иметь к нему доступ, запомнить адрес назначения (а, возможно, и другие поля заголовка), а также оно должно передать его сетевому интерфейсу. На этом завершается работа на клиентской стороне. Включается таймер передачи, и ядро может либо выполнять циклический опрос наличия ответа, либо передать управление планировщику, который выберет какой-либо другой процесс на выполнение. В первом случае ускоряется выполнение запроса, но отсутствует мультипрограммирование.

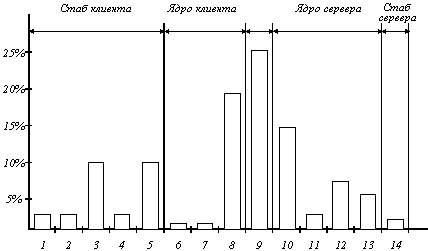
На стороне сервера поступающие биты помещаются принимающей аппаратурой либо во встроенный буфер, либо в оперативную память. Когда вся информация будет получена, генерируется прерывание. Обработчик прерывания проверяет правильность данных пакета и определяет, какому стабу следует их передать. Если ни один из стабов не ожидает этот пакет, обработчик должен либо поместить его в буфер, либо вообще отказаться от него. Если имеется ожидающий стаб, то сообщение копируется ему. Наконец, выполняется переключение контекстов, в результате чего восстанавливаются регистры и карта памяти, принимая те значения, которые они имели в момент, когда стаб сделал вызов receive.

Теперь начинает работу серверный стаб. Он распаковывает параметры и помещает их соответствующим образом в стек. Когда все готово, выполняется вызов сервера. После выполнения процедуры сервер передает результаты клиенту. Для этого выполняются все описанные выше этапы, только в обратном порядке.

Рисунок 3.3 показывает последовательность команд, которую необходимо выполнить для каждого RPC-вызова, а рисунок 3.4 - какая доля общего времени выполнения RPC тратится на выполнение каждого их описанных 14 этапов. Исследования были проведены на мультипроцессорной рабочей станции DEC Firefly, и, хотя наличие пяти процессоров обязательно повлияло на результаты измерений, приведенная на рисунке гистограмма дает общее представление о процессе выполнения RPC.



*Рис. 3.3. Этапы выполнения процедуры RPC*



*Рис. 3.4. Распределение времени между 14 этапами выполнения RPC*

***1. Вызов стаба***

***2. Подготовить буфер***

***3. Упаковать параметры***

***4. Заполнить поле заголовка***

***5. Вычислить контрольную сумму в сообщении***

***6. Прерывание к ядру***

***7. Очередь пакета на выполнение***

***8. Передача сообщения контроллеру по шине QBUS***

***9. Время передачи по сети Ethernet***

***10. Получить пакет от контроллера***

***11. Процедура обработки прерывания***

***12. Вычисление контрольной суммы***

***13. Переключение контекста в пространство пользователя***

***14. Выполнение серверного стаба***

**Динамическое связывание**

Рассмотрим вопрос о том, как клиент задает месторасположение сервера. Одним из методов решения этой проблемы является непосредственное использование сетевого адреса сервера в клиентской программе. Недостаток такого подхода - его чрезвычайная негибкость: при перемещении сервера, или при увеличении числа серверов, или при изменении интерфейса во всех этих и многих других случаях необходимо перекомпилировать все программы, которые использовали жесткое задание адреса сервера. Для того, чтобы избежать всех этих проблем, в некоторых распределенных системах используется так называемое динамическое связывание.

Начальным моментом для динамического связывания является формальное определение (спецификация) сервера. Спецификация содержит имя файл-сервера, номер версии и список процедур-услуг, предоставляемых данным сервером для клиентов. Для каждой процедуры дается описание ее параметров с указанием того, является ли данный параметр входным или выходным относительно сервера. Некоторые параметры могут быть одновременно входными и выходными - например, некоторый массив, который посылается клиентом на сервер, модифицируется там, а затем возвращается обратно клиенту (операция copy/ restore).

Формальная спецификация сервера используется в качестве исходных данных для программы-генератора стабов, которая создает как клиентские, так и серверные стабы. Затем они помещаются в соответствующие библиотеки. Когда пользовательская (клиентская) программа вызывает любую процедуру, определенную в спецификации сервера, соответствующая стаб-процедура связывается с двоичным кодом программы. Аналогично, когда компилируется сервер, с ним связываются серверные стабы.

При запуске сервера самым первым его действием является передача своего серверного интерфейса специальной программе, называемой binder'ом. Этот процесс, известный как процесс регистрации сервера, включает передачу сервером своего имени, номера версии, уникального идентификатора и описателя местонахождения сервера. Описатель системно независим и может представлять собой IP, Ethernet, X.500 или еще какой-либо адрес. Кроме того, он может содержать и другую информацию, например, относящуюся к аутентификации.

Когда клиент вызывает одну из удаленных процедур первый раз, например, read, клиентский стаб видит, что он еще не подсоединен к серверу, и посылает сообщение binder-программе с просьбой об импорте интерфейса нужной версии нужного сервера. Если такой сервер существует, то binder передает описатель и уникальный идентификатор клиентскому стабу.

Клиентский стаб при посылке сообщения с запросом использует в качестве адреса описатель. В сообщении содержатся параметры и уникальный идентификатор, который ядро сервера использует для того, чтобы направить поступившее сообщение в нужный сервер в случае, если их несколько на этой машине.

Этот метод, заключающийся в импорте/экспорте интерфейсов, обладает высокой гибкостью. Например, может быть несколько серверов, поддерживающих один и тот же интерфейс, и клиенты распределяются по серверам случайным образом. В рамках этого метода становится возможным периодический опрос серверов, анализ их работоспособности и, в случае отказа, автоматическое отключение, что повышает общую отказоустойчивость системы. Этот метод может также поддерживать аутентификацию клиента. Например, сервер может определить, что он может быть использован только клиентами из определенного списка.

Однако у динамического связывания имеются недостатки, например, дополнительные накладные расходы (временные затраты) на экспорт и импорт интерфейсов. Величина этих затрат может быть значительна, так как многие клиентские процессы существуют короткое время, а при каждом старте процесса процедура импорта интерфейса должна быть снова выполнена. Кроме того, в больших распределенных системах может стать узким местом программа binder, а создание нескольких программ аналогичного назначения также увеличивает накладные расходы на создание и синхронизацию процессов.

**Семантика RPC в случае отказов**

В идеале RPC должен функционировать правильно и в случае отказов. Рассмотрим следующие классы отказов:

1. Клиент не может определить местонахождения сервера, например, в случае отказа нужного сервера, или из-за того, что программа клиента была скомпилирована давно и использовала старую версию интерфейса сервера. В этом случае в ответ на запрос клиента поступает сообщение, содержащее код ошибки.
2. Потерян запрос от клиента к серверу. Самое простое решение - через определенное время повторить запрос.
3. Потеряно ответное сообщение от сервера клиенту. Этот вариант сложнее предыдущего, так как некоторые процедуры не являются идемпотентными. Идемпотентной называется процедура, запрос на выполнение которой можно повторить несколько раз, и результат при этом не изменится. Примером такой процедуры может служить чтение файла. Но вот процедура снятия некоторой суммы с банковского счета не является идемпотентной, и в случае потери ответа повторный запрос может существенно изменить состояние счета клиента. Одним из возможных решений является приведение всех процедур к идемпотентному виду. Однако на практике это не всегда удается, поэтому может быть использован другой метод - последовательная нумерация всех запросов клиентским ядром. Ядро сервера запоминает номер самого последнего запроса от каждого из клиентов, и при получении каждого запроса выполняет анализ - является ли этот запрос первичным или повторным.
4. Сервер потерпел аварию после получения запроса. Здесь также важно свойство идемпотентности, но к сожалению не может быть применен подход с нумерацией запросов. В данном случае имеет значение, когда произошел отказ - до или после выполнения операции. Но клиентское ядро не может распознать эти ситуации, для него известно только то, что время ответа истекло. Существует три подхода к этой проблеме:

* Ждать до тех пор, пока сервер не перезагрузится и пытаться выполнить операцию снова. Этот подход гарантирует, что RPC был выполнен до конца по крайней мере один раз, а возможно и более.
* Сразу сообщить приложению об ошибке. Этот подход гарантирует, что RPC был выполнен не более одного раза.
* Третий подход не гарантирует ничего. Когда сервер отказывает, клиенту не оказывается никакой поддержки. RPC может быть или не выполнен вообще, или выполнен много раз. Во всяком случае этот способ очень легко реализовать.

Ни один из этих подходов не является очень привлекательным. А идеальный вариант, который бы гарантировал ровно одно выполнение RPC, в общем случае не может быть реализован по принципиальным соображениям. Пусть, например, удаленной операцией является печать некоторого текста, которая включает загрузку буфера принтера и установку одного бита в некотором управляющем регистре принтера, в результате которой принтер стартует. Авария сервера может произойти как за микросекунду до, так и за микросекунду после установки управляющего бита. Момент сбоя целиком определяет процедуру восстановления, но клиент о моменте сбоя узнать не может. Короче говоря, возможность аварии сервера радикально меняет природу RPC и ясно отражает разницу между централизованной и распределенной системой. В первом случае крах сервера ведет к краху клиента, и восстановление невозможно. Во втором случае действия по восстановлению системы выполнить и возможно, и необходимо.

1. Клиент потерпел аварию после отсылки запроса. В этом случае выполняются вычисления результатов, которых никто не ожидает. Такие вычисления называют "сиротами". Наличие сирот может вызвать различные проблемы: непроизводительные затраты процессорного времени, блокирование ресурсов, подмена ответа на текущий запрос ответом на запрос, который был выдан клиентской машиной еще до перезапуска системы.

Как поступать с сиротами? Рассмотрим 4 возможных решения.

* *Уничтожение.* До того, как клиентский стаб посылает RPC-сообщение, он делает отметку в журнале, оповещая о том, что он будет сейчас делать. Журнал хранится на диске или в другой памяти, устойчивой к сбоям. После аварии система перезагружается, журнал анализируется и сироты ликвидируются. К недостаткам такого подхода относятся, во-первых, повышенные затраты, связанные с записью о каждом RPC на диск, а, во-вторых, возможная неэффективность из-за появления сирот второго поколения, порожденных RPC-вызовами, выданными сиротами первого поколения.
* *Перевоплощение*. В этом случае все проблемы решаются без использования записи на диск. Метод состоит в делении времени на последовательно пронумерованные периоды. Когда клиент перезагружается, он передает широковещательное сообщение всем машинам о начале нового периода. После приема этого сообщения все удаленные вычисления ликвидируются. Конечно, если сеть сегментированная, то некоторые сироты могут и уцелеть.
* *Мягкое перевоплощение* аналогично предыдущему случаю, за исключением того, что отыскиваются и уничтожаются не все удаленные вычисления, а только вычисления перезагружающегося клиента.
* *Истечение срока*. Каждому запросу отводится стандартный отрезок времени Т, в течение которого он должен быть выполнен. Если запрос не выполняется за отведенное время, то выделяется дополнительный квант. Хотя это и требует дополнительной работы, но если после аварии клиента сервер ждет в течение интервала Т до перезагрузки клиента, то все сироты обязательно уничтожаются.

На практике ни один из этих подходов не желателен, более того, уничтожение сирот может усугубить ситуацию. Например, пусть сирота заблокировал один или более файлов базы данных. Если сирота будет вдруг уничтожен, то эти блокировки останутся, кроме того уничтоженные сироты могут остаться стоять в различных системных очередях, в будущем они могут вызвать выполнение новых процессов и т.п.

# HTTP

Request for Comments, **RFC** — [документ](http://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82) из серии пронумерованных информационных документов [Интернета](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), содержащих технические спецификации и Стандарты, широко применяемые во Всемирной сети. Название «Request for Comments» ещё можно перевести как «заявка на обсуждение» или «тема для обсуждения». В настоящее время первичной публикацией документов RFC занимается IETF под эгидой открытой организации Общество Интернета (Internet Society, ISOC). Правами на RFC обладает именно Общество Интернета.

(cl 2) Internet Engineering Task Force — открытое международное сообщество проектировщиков, учёных, сетевых операторов и провайдеров, созданное IAB в 1986 году, которое занимается развитием протоколов и архитектуры Интернета.

Вся техническая работа осуществляется в рабочих группах IETF, занимающихся конкретной тематикой (например, вопросами маршрутизации, транспорта данных, безопасности и т. д.). Работа в основном ведётся через почтовые списки, но трижды в году проводятся собрания IETF.

Результаты деятельности рабочих групп оформляются в виде рабочих проектов (Internet drafts), которые затем используются ISOC для кодификации новых стандартов.

Задачи IETF (в соответствии с RFC 4677):

* Идентификации проблем и предложение решений в технических аспектах организации Интернете
* Разработка спецификаций, стандартов и соглашений по общим архитектурным принципам протоколов Интернет
* Вынесение рекомендаций на относительно стандартизации протоколов на рассмотрение Internet Engineering Steering Group (IESG)
* Содействие широкому распространению технологий и стандартов, разрабатываемых в Internet Research Task Force (IRTF)
* Организация дискуссии для обмена информации в сообществе Интернет между учеными, разработчиками, пользователями, производителями оборудования и услуг, сетевыми администраторами и т. д.

Internet Society, **ISOC** — международная профессиональная организация, занимающаяся развитием и обеспечением доступности сети [Интернет](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). Организация насчитывает более 20 тысяч индивидуальных членов и более 100 организаций-членов в 180 странах мира. Общество Интернета предоставляет организационную основу для множества других консультативных и исследовательских групп, занимающихся развитием Интернета, включая IETF и IAB.

Общество Интернета было основано в 1992 году, чтобы обеспечить корпоративную структуру для организаций, занимающихся развитием Интернета, вроде IETF. Дело в том, что IETF и подобные ей организации были и остаются довольно неформальными с юридической точки зрения, но они нуждаются в финансовой поддержке и определённом правовом статусе. Для этих целей и было создано Общество Интернета.

(The Internet Engineering Steering Group) — группа по выработке инженерного регламента Интернета, которая отвечает за техническое руководство деятельностью IETF и процессом стандартизации Интернета. Как подразделение ISOC, она отвечает за принятие новых спецификаций в качестве стандартов Интернета с соблюдением всех установленных процедур.

Общество Интернета ставит своей задачей способствовать развитию Интернета, разработке новых интернет-технологий и обеспечению доступности Всемирной сети в мировом масштабе. Официально ISOC является некоммерческой образовательной организацией, она имеет офисы в [Вирджинии](http://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D0%B4%D0%B6%D0%B8%D0%BD%D0%B8%D1%8F) (США) и Женеве (Швейцария). ISOC официально владеет правами на все документы RFC и прикладывает много усилий для практического внедрения Стандартов Интернета, описанных в RFC. Официально миссия ISOC сформулирована так:

**«Обеспечить открытое развитие, эволюцию и использование Интернета на благо всех людей во всём мире.»**

Но Общество Интернета занимается не только информационной и образовательной деятельностью. ISOC также занимается финансированием и координацией общественных инициатив, связанных с Интернетом. Она спонсирует множетсво мероприятий по всему миру (в основном, в развивающихся странах), направленных на популяризацию Интернета и освоение навыков работы в Сети широкими слоями населения. Общество Интернета также занимается подсчётом сетевой статистики и проведением маркетинговых исследований.

ISOC ежегодно проводит крупные конференции «International Networking» (INET), на которых собираются представители интернет-сообщества и обсуждают вопросы дальнейшего развития и стандартизации Глобальной сети.

Формат RFC появился в 1969 году при обсуждении проекта ARPANET. RFC 1 был опубликован 7 апреля 1969 г. и назывался «Host Software». Первые RFC распространялись в печатном виде на бумаге в виде обычных писем, но уже с декабря 1969 г., когда заработали первые сегменты ARPANET, документы начали распространяться в электронном виде.

Большинство ранних RFC были созданы в Калифорнийском университете Лос-Анджелеса и Стэнфордском исследовательском институте.

С 1969 по 1989 гг. бессменным и единственным редактором RFC был Джон Постел. После его смерти Общество Интернета (ISOC) поручило редактирование и публикацию RFC Институту информационных наук Университета Южной Калифорнии.

## **Содержимое RFC**

Несмотря на название, запросы комментариев RFC сейчас рассматриваются как стандарты Интернета (а рабочие версии стандартов обычно называют драфтами, от англ. draft здесь — черновик). Согласно RFC 2026, жизненный цикл стандарта выглядит следующим образом:

1. Выносится на всеобщее рассмотрение Интернетовский черновик (Internet Draft). Черновики не имеют официального статуса, и удаляются из базы через шесть месяцев после последнего изменения.
2. Если черновик стандарта оказывается достаточно удачным и непротиворечивым, он получает статус Предложенного стандарта (Proposed Standard), и свой номер RFC. Наличие программной реализации стандарта желательно, но не обязательно.
3. Следующая стадия — Черновой стандарт (Draft Standard) означает, что предложенный стандарт принят сообществом, в частности, существуют две независимые по коду совместимые реализации разных команд разработчиков. В черновые стандарты ещё могут вноситься мелкие правки, но они считаются достаточно стабильными и рекомендуются для реализации.
4. Высший уровень — Стандарт Интернета (Internet Standard). Это спецификации с большим успешным опытом применения и зрелой формулировкой. Параллельно с нумерацией RFC они имеют свою собственную нумерацию STD. Список стандартов имеется в документе STD 1 (сейчас это RFC 3700, но нумерация может измениться). Из более чем трёх тысяч RFC этого уровня достигли только несколько десятков.
5. Многие старые RFC замещены более новыми версиями под новыми номерами, или вышли из употребления. Такие документы получают статус Исторических (Historic)

Практически все стандарты Глобальной сети существуют в виде опубликованных заявок RFC. Но в виде документов RFC выходят не только стандарты, но также концепции, введения в новые направления в исследованиях, исторические справки, результаты экспериментов, руководства по внедрению технологий, предложения и рекомендации по развитию существующих Стандартов и другие новые идеи в информационных технологиях:

1. Экспериментальные (Experimental) спецификации содержат информацию об экспериментальных исследованиях, интересных для интернет-сообщества. Это могут быть, например, прототипы, реализующие новые концепции.
2. Информационные (Informational) RFC предназначены для ознакомления общественности не являются стандартами и не являются результатом консенсуса или рекомендациями. Некоторые черновики, не получившие статуса Предложенного стандарта, но представляющие интерес, могут быть опубликованы как Информационные RFC.
3. Лучший современный опыт (Best Current Practice). Эта серия RFC содержит рекомендации по реализации стандартов, в том числе, от сторонних организаций, а также внутренние документы о структуре и процедурах стандартизации.

Почти все стандарты разрабатываются под эгидой каких-либо научных или интернет-организаций (например W3C, IETF, консорциум Юникода, Интернет2).

Запросы комментариев официально существуют только на английском языке. Строгих требований к оформлению нет.

Network Working Group  
Request for Comments: 2068  
Category: Standards Track

**Протокол Http:/**

## RFC 2068. Протокол передачи гипертекста -- HTTP/1.1

(cl 3)

Протокол передачи Гипертекста (HTTP) - протокол прикладного уровня для распределенных, совместных, многосредных информационных систем (изначально — в виде гипертекстовых документов).

Основой HTTP 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.

(cl 4)

HTTP используется в World Wide Web (WWW) начиная с 1990 года. Первой версией HTTP, известной как HTTP/0.9, был простой протокол для передачи необработанных данных через Интернет. HTTP/1.0, как определено в RFC 1945, был улучшением этого протокола, позволяя сообщениям иметь MIME-подобный формат, содержащий метаинформацию о передаваемых данных и имел модифицированную семантику запросов/ответов. Однако, HTTP/1.0 недостаточно хорошо учитывал особенности работы с иерархическими прокси-серверами (hierarchical proxies), кэшированием, постоянными соединениями, и виртуальными хостами (virtual hosts). Кроме того, быстрое увеличение не полностью совместимых приложений, называющих тот протокол, который они использовали "HTTP/1.0", потребовало введения версии протокола, в которой были бы заложены возможности, позволяющие приложениям определять истинные возможности друг друга.

**HTTP/1.1**

Текущая версия протокола, принята в июне 1999 года. Новым в этой версии был режим «постоянного соединения»: TCP-соединение может оставаться открытым после отправки ответа на запрос, что позволяет посылать несколько запросов за одно соединение. Клиент теперь обязан посылать информацию об имени хоста, к которому он обращается, что сделало возможной более простую организацию виртуального хостинга.

Эта спецификация определяет протокол "HTTP/1.1". Этот протокол содержит более строгие требования, чем HTTP/1.0, гарантирующие надежную реализацию возможностей. Практически информационные системы требуют большей функциональности, чем просто загрузку информации, включая поиск, модификацию при помощи внешнего интерфейса, и аннотацию (annotation). Http предоставляет открытый набор методов, которые указывают цель запроса. Они основаны на дисциплине ссылки, обеспеченной Универсальным Идентификатором Ресурса (URI), как расположение (URL) или имя (URN), для идентификации ресурса, к которому этот метод применяется. Сообщения передаются в формате, подобном используемому электронной почтой, как определено Многоцелевыми Расширениями Электронной Почты (MIME).

(cl 5)

(cl 6)

* **HTTP** используется также в качестве «транспорта» для других протоколов прикладного уровня, таких как **SOAP**, **XML-RPC**, **WebDAV**.
* Основным объектом манипуляции в **HTTP** является ресурс, на который указывает **URI** (англ. Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть 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.

(cl 7)

HTTP также используется как обобщенный протокол связи между агентами пользователей и прокси-серверами/шлюзами (proxies/gateways) или другими сервисами Интернета, включая такие, как SMTP, NNTP, FTP, Gopher, и WAIS. Таким образом, http закладывает основы многосредного (hypermedia) доступа к ресурсам для разнообразных приложений.

Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов **HTTP** использует глобальные **URI**. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять 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 осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

Терминология:

Эта спецификация использует ряд терминов для описания роли участников, некоторых объектов, и HTTP связи.

Соединение (connection)

Виртуальный канал транспортого уровня, установленный между двумя программами с целью связи.

До постоянных соединений для запроса каждого URL устанавливалось отдельное TCP соединение, что увеличивало нагрузку на HTTP сервера и вызывало загрузку Интернета. Использование встроенных изображений и других связанных данных часто требует от клиента делать несколько запросов к одному серверу за короткий промежуток времени.

Постоянные HTTP соединения имеют ряд преимуществ:

Открытие и закрытие меньшего количества TCP соединений экономит время центрального процессора и память, используемую для управляющих блоков протокола TCP.

HTTP запросы и ответы может быть конвейеризованы в соединении. Конвейерная обработка позволяет клиенту делать множество запросов не ожидая ответа на каждый, следовательно, одиночное TCP соединение, использование которого намного более эффективно, теряет меньше времени.

Загрузка сети уменьшается с уменьшением числа пакетов, вызванных открытием TCP соединений, и, следовательно, дает протоколу TCP достаточное время для определения состояния загрузки сети.

HTTP может развиваться более элегантно; так как ошибки могут сообщаться без закрытия TCP соединения в качестве штрафа. Клиенты, использующие будущие версии HTTP могли бы оптимистично пробовать новые возможности, но при связи со старым сервером, повторять запрос, используя старую семантику после сообщения об ошибке.

HTTP реализациям СЛЕДУЕТ реализовывать постоянные соединения.

Значительное отличие HTTP/1.1 от более ранних версий HTTP состоит в том, что постоянные соединения являются заданным по умолчанию поведением любого HTTP соединения. То есть если не обозначено иного, клиент может считать, что сервер поддержит постоянное соединение.

Постоянные соединения обеспечивают механизм, согласно которому клиент и сервер могут сообщить о разрыве TCP соединения. Это сигнализируется при помощи использования поля заголовка Connection. При получении сообщения о разрыве соединения клиент НЕ ДОЛЖЕН посылать больше запросов по этому соединению.

Сообщение (message)

Основной модуль HTTP связи, состоящей из структурной последовательности октетов, соответствующих синтаксису, определенному в разделе 4 и передаваемых по соединению.

Запрос (request)

Любое HTTP сообщение, содержащее запрос.

Сообщение запроса от клиента к серверу содержит в первой строке: метод, который нужно применить к ресурсу, идентификатор ресурса и используемую версию протокола.

Ответ (response)

Любое HTTP сообщение, содержащее ответ.

После получения и интерпретации сообщения запроса, сервер отвечает сообщением HTTP ответа.

Ресурс (resource)

Сетевой объект данных или сервис, который может быть идентифицирован URI, определеляемым в разделе 3.2. Ресурсы могут быть доступны в нескольких представлениях (например на нескольких языках, в разных форматах данных, иметь различный размер, иметь различную разрешающую способность) или различаться по другим параметрам.

Объект (entity)

Информация, передаваемая в качестве полезной нагрузки запроса или ответа. Объект состоит из метаинформации в форме полей заголовка объекта и содержания в форме тела объекта.

Этот раздел относится как к отправителю, так и к получателю, то есть к клиенту или серверу, в зависимости от того, кто посылает, а кто получает объект.

Представление (representation)

Объект включенный в ответ, и подчиняющийся обсуждению содержимого Content Negotiation). Может существовать несколько представлений, связанных со специфическими состояниями ответа.

Обсуждение содержимого (content negotiation)

Механизм для выбора соответствующего представления во время обслуживания запроса. Представление объектов в любом ответе может быть обсуждено (включая ошибочные ответы).

HTTP/1.1 сервер МОЖЕТ считать, что HTTP/1.1 клиент не предполагает поддерживать постоянное соединение, если посланный в запросе заголовок Connection содержит лексему соединения (connection-token) "close". Если сервер решает закрыть соединение немедленно после посылки ответа, то ему СЛЕДУЕТ послать заголовок Connection, который содержит лексему соединения (connection-token) "close".

Вариант (variant)

Ресурс может иметь одно, или несколько представлений, связанных с ним в данный момент. Каждое из этих представлений называется "вариант". Использование термина "вариант" не обязательно подразумевает, что ресурс подчинен обсуждению содержимого.

Клиент (client)

Программа, которая устанавливает соединения с целью посылки запросов.

Агент пользователя (user agent)

Клиент, который инициирует запрос. Как правило браузеры, редакторы, роботы (spiders), или другие инструментальные средства пользователя.

Сервер (server)

Приложение, которое слушает соединения, принимает запросы на обслуживание и посылает ответы. Любая такая программа способна быть как клиентом, так и сервером; наше использование данного термина относится скорее к роли, которую программа выполняет, создавая специфические соединения, нежели к возможностям программы вообще. Аналогично, любой сервер может действовать как первоначальный сервер, прокси-сервер, шлюз, или туннель (tunnel), изменяя поведение, основываясь на характере каждого запроса.

Первоначальный сервер (origin server)

Сервер, на котором данный ресурс постоянно находится или должен быть создан.

Прокси-сервер (proxy)

Программа-посредник, которая действует и как сервер, и как клиент с целью создания запросов от имени других клиентов. Запросы обслуживаются прокси-сервером, или передаются им, возможно с изменениями. Прокси-сервер должен удовлетворять требованиям клиента и сервера, согласно этой спецификации.

Очень важно, чтобы прокси-сервера правильно выполняли свойства полей заголовка Connection, как определено в 14.2.1.

Прокси-сервер ДОЛЖЕН сообщать о постоянных соединениях отдельно своим клиентам и отдельно первоначальным серверам (или другим прокси-серверам), которые с ним соединены. Каждое постоянное соединение применяется только к одной транспортной связи.

Прокси-сервер НЕ ДОЛЖЕН устанавливать постоянное соединение с HTTP/1.0 клиентом.

Шлюз (gateway)

Сервер, который действует как посредник для некоторого другого сервера. В отличие от прокси-сервера, шлюз получает запросы в качестве первоначального сервера для запрошенного ресурса; клиент запроса может не знать, что он соединяется со шлюзом.

Туннель (tunnel)

Программа-посредник, которая поддерживает соединение. Один раз созданный, туннель не рассматривается как часть HTTP связи, хотя туннель, возможно, был инициализирован запросом HTTP. Туннель прекращает существовать, когда оба конца соединения закрываются.

Кэш (cache)

Локальная память, в которой программа хранит сообщения ответов, и в которой располагается подсистема, управляющая хранением, поиском и стиранием сообщений. Кэш сохраняет ответы, которые могут быть сохранены, чтобы уменьшить время ответа и загрузку сети (траффик) при будущих эквивалентных запросах. Любой клиент или сервер может иметь кэш, но кэш не может использоваться сервером, который действует как туннель.

(cl 9)

Кэшируемый (cachable)

Ответ является кэшируемым, если кэшу разрешено сохранить копию ответного сообщения для использования при ответе на последующие запросы. Правила для определения кэшируемости HTTP ответов определены в разделе 13. Даже если ресурс кэшируем, могут существовать дополнительные ограничения на использование КЭШем сохраненной копии для сходного запроса.

Непосредственный (first-hand)

Ответ считается непосредственным, если он приходит непосредственно от первоначального сервера без ненужной задержки, возможно через один или несколько прокси-серверов. Ответ также является непосредственным, если его правильность только что была проверена непосредственно первоначальным сервером.

Точное время устаревания (explicit expiration time)

Время, определенное первоначальным сервером и показывающее кэшу, когда объект больше не может быть возвращен кэшем клиенту без дополнительной проверки правильности.

Эвристическое время устаревания (heuristic expiration time)

Время устаревания, назначенное кэшем, если не указано точное время устаревания.

Возраст (age)

Возраст ответа - время, прошедшее с момента отсылки, или успешной проверки ответа первоначальным сервером.

Время жизни (freshness lifetime)

Отрезок времени между порождением ответа и временем устаревания.

Свежий (fresh)

Ответ считается свежим, если его возраст еще не превысил время жизни.

Просроченнный (stale)

Ответ считается просроченным, если его возраст превысил время жизни.

Семантически прозрачный (semantically transparent)

Говорят, что кэш ведет себя "семантически прозрачным" образом в отношении специфического ответа, когда использование кэша не влияет ни на клиента запроса, ни на первоначальный сервер, но повышает эффективность. Когда кэш семантически прозрачен, клиент получает точно такой же ответ (за исключением промежуточных (hop-by-hop) заголовков), который получил бы, запрашивая непосредственно первоначальный сервер, а не кэш.

Указатель правильности (validator)

Элемент протокола (например, метка объекта или время последней модификации (Last-Modified time)), который используется, чтобы выяснить, является ли находящаяся в кэше копия эквивалентом объекта.

Общее описание:

Протокол HTTP - это протокол запросов/ответов. Клиент посылает серверу запрос, содержащий метод запроса, URI, версию протокола, MIME-подобное сообщение, содержащее модификаторы запроса, клиентскую информацию, и, возможно, тело запроса, по соединению. Сервер отвечает строкой состояния, включающей версию протокола сообщения, код успешного выполнения или код ошибки, MIME-подобное сообщение, содержащее информацию о сервере, метаинформацию объекта, и, возможно, тело объекта. Большинство HTTP соединений инициализируется агентом пользователя и состоит из запроса, который нужно применить к ресурсу на некотором первоначальном сервере. В самом простом случае, он может быть выполнен посредством одиночного соединения (v) между агентом пользователя (UA) и первоначальным сервером (O).

(cl 10)

цепочка запросов --------------------->

UA -------------------v------------------- O

<----------------------- цепочка ответов

Более сложная ситуация возникает, когда в цепочке запросов/ответов присутствует один или несколько посредников. Существуют три основных разновидности посредников: прокси-сервера, шлюзы, и туннели. Прокси-сервер является агентом-посредником, который получает запросы на некоторый URI в абсолютной форме, изменяет все сообщение или его часть, и отсылает измененный запрос серверу, идентифицированному URI. Шлюз - это принимающий агент, действующий как бы уровень выше некоторого другого сервера(ов) и, в случае необходимости, транслирующий запросы в протокол основного сервера. Туннель действует как реле между двумя соединениями не изменяя сообщения; туннели используются, когда связь нужно производить через посредника (например Firewall), который не понимает содержание сообщений.

цепочка запросов ----------------------------------->

UA -----v----- A -----v----- B -----v----- C -----v----- O

<------------------------------------ цепочка ответов

На последнем рисунке показаны три посредника (A, B, и C) между агентом пользователя и первоначальным сервером. Запросы и ответы передаются через четыре отдельных соединения. Это различие важно, так как некоторые опции HTTP соединения применимы только к соединению с ближайшим не туннельным соседом, некоторые только к конечным точкам цепочки, а некоторые ко всем соединениям в цепочке. Хотя диаграмма линейна, каждый участник может быть задействован в нескольких соединениях одновременно. Например, B может получать запросы от других клиентов, а не только от A, и/или пересылать запросы к серверам, отличным от C, в то же время, когда он обрабатывает запрос от А.

Любая сторона соединения, которая не действует как туннель, может использовать внутренний кэш для обработки запросов. Эффект кэша заключается в том, что цепочка запросов/ответов сокращается, если один из участников в цепочке имеет кэшированный ответ, применимый к данному запросу. Далее иллюстрируется цепочка, возникающая в результате того, что B имеет кэшированую копию раннего ответа O (полеченного через C) для запроса, и который не кэшировался ни UA, ни A.

цепочка запросов ------->

UA -----v----- A -----v----- B - - - - - - C - - - - - - O

<-------- цепочка ответов

Не все ответы полезно кэшировать, а некоторые запросы могут содержать модификаторы, которые включают специальные требования, управляющие поведением кэша.

Фактически, имеется широкое разнообразие архитектур и конфигураций кэшей и прокси-серверов, в настоящее время разрабатываемых или развернутых в World Wide Web; эти системы включают национальные иерархии прокси-кэшей, которые сохраняют пропускную способность межокеанских каналов, системы, которые распространяют во много мест содержимое кэша, организации, которые распространяют подмножества кэшируемых данных на CD-ROM, и так далее. HTTP системы используются в корпоративных интранет-сетях с высокоскоростными линиями связи, и для доступа через PDA с маломощными линиями и неустойчивой связи. Цель HTTP/1.1 состоит в поддержании широкого многообразия конфигураций, уже построенных при введении ранних версий протокола, а также в удовлетворении потребностей разработчиков web приложений, требующих высокой надежности, по крайней мере надежных относительно индикации отказа.

HTTP соединение обычно происходит посредством TCP/IP соединений. Заданный по умолчанию порт TCP - 80, но могут использоваться и другие порты. HTTP также может быть реализован посредством любого другого протокола Интернета, или других сетей. HTTP необходима только надежная передача данных, следовательно может использоваться любой протокол, который гарантирует надежную передачу данных; отображение структуры запроса и ответа HTTP/1.1 на транспортные модули данных рассматриваемого протокола - вопрос, не решаемый этой спецификацией.

Большинство реализаций HTTP/1.0 использовало новое соединение для каждого обмена запросом/ответом. В HTTP/1.1, установленное соединение может использоваться для одного или нескольких обменов запросом/ответом, хотя соединение может быть закрыто по ряду причин. Все механизмы, определенные этим документом, описаны как в обычной, так и в увеличенной нормальной записи Бекуса-Наура (BNF), подобной используемой в RFC 822. Разработчик должен быть знаком с такой формой записи, чтобы понять данную спецификацию. Увеличенная нормальная запись Бекуса-Наура включает следующие конструкции:

имя = определение

name = definition

Имя правила - это просто его название (не включающее символов "<" и ">"), и отделяемое от определения символом равенства "=". Пробел важен только при выравнивании продолжающихся строк, используемых для указания определений правил, которые занимают более одной строки. Некоторые основные правила, такие как SP, LWS, HT, CRLF, DIGIT, ALPHA и т.д, представлены в верхнем регистре. Угловые скобки используются в определении всякий раз, когда их присутствие облегчает использование имен правил.

"литерал"

"literal"

Кавычки окружают литеральный текст. Если не установлено иного, этот текст регистро-независим.

правило1 | правило2

rule1 | rule2

Элементы, отделяемые полосой ("|") являются вариантами. Например, "да | нет" принимает значение либо да, либо нет.

(правило1 правило2)

(rule1 rule2)

Элементы, включенные в круглые скобки обрабатываются как один элемент. Таким образом, "(elem (foo | bar) elem)" допускает последовательности лексем "elem foo elem" и "elem bar elem".

\*правило

\*rule

Символ "\*", предшествующий элементу, указывает повторение. Полная форма - "<n>\*<m>element" означает минимум <n>, максимумm<m> вхождений элемента. Значения по умолчанию - 0 и бесконечность. Таким образом запись "\*(element)" допускает любое число повторений (в том числе ноль); запись "1\*element" требует по крайней мере одно повторение; а "1\*2element" допускает либо один, либо два повторения.

[правило]

[rule]

В квадратные скобки заключают опциональные элементы; "[foo bar]" эквивалентно "\*1(foo bar)".

N правило

N rule

Точное количество повторений: "<n>(element)" эквивалентно "<n>\*<n>(element)"; то есть присутствует точно <n> повторов элемента. Таким образом 2DIGIT - номер из 2 цифр, а 3ALPHA - строка из трех алфавитных символов.

**(cl 11)**

**Достоинства HTTP**

* **Простота**

Протокол настолько прост в реализации, что позволяет с лёгкостью создавать клиентские приложения.

* **Расширяемость**

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

* **Распространённость**

При выборе протокола HTTP для решения конкретных 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.

(cl 12)

**Недостатки и проблемы**

* **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.

(cl 13)

* **Отсутствие «навигации»**

Хотя протокол разрабатывался как средство работы с ресурсами сервера, у него отсутствуют в явном виде средства навигации среди 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.

(cl 14)

* **Нет поддержки распределённости**

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

В 1998 году W3C предложил альтернативный протокол HTTP-NG (англ. HTTP Next Generation) для полной замены устаревшего с акцентированием внимания именно на этой области. Идею его необходимости поддержали крупные специалисты по распределённым вычислениям, но данный протокол до сих пор находится на стадии разработки.

(cl 15)

**Программное обеспечение**

Всё программное обеспечение для работы с протоколом HTTP разделяется на три больших категории:

\* **Серверы** как основные 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 (рус. исходный сервер). Разумеется, один и тот же программный продукт может одновременно выполнять функции клиента, сервера или посредника в зависимости от поставленных задач. В спецификациях протокола HTTP подробно описывается поведение для каждой из этих ролей.

(cl 16)

* Большинство протоколов предусматривают установление TCP-сессии, в ходе которой один раз происходит авторизация, и дальнейшие действия выполняются в контексте этой авторизации. HTTP же устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP было разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включённые в неё объекты (картинки, каскадные стили и т. п.), а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (неанонимного) доступа в HTTP используются cookies; причём такой способ авторизации позволяет сохранить сессию даже после перезагрузки клиента и сервера.
* При доступе к данным по FTP или по файловым протоколам тип файла (точнее, тип содержащихся в нём данных) определяется по расширению имени файла, что не всегда удобно. HTTP перед тем, как передать сами данные, передаёт в заголовке строчку «Content-Type: тип/подтип», позволяющую клиенту однозначно определить, каким образом обрабатывать присланные данные. Это особенно важно при работе с CGI-скриптами, когда расширение имени файла указывает не на тип присылаемых клиенту данных, а на необходимость запуска данного файла на сервере и отправки клиенту результатов работы программы, записанной в этом файле (при этом один и тот же файл в зависимости от аргументов запроса и своих собственных соображений может порождать ответы разных типов — в простейшем случае картинки в разных форматах).
* Кроме того, HTTP позволяет клиенту прислать на сервер параметры, которые будут переданы запускаемому CGI-скрипту. Для этого же в HTML были введены формы.

Перечисленные особенности HTTP позволили создавать поисковые машины (первой из которых стала AltaVista, созданная фирмой DEC), форумы и Internet-магазины. Это превратило Internet из «академической игрушки» в «коммерческий сервис»: появились компании, основным полем деятельности которых стало предоставление доступа в Internet (компании-провайдеры) и создание сайтов

* **HTTP** (англ. *HyperText Transfer Protocol* — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных (изначально — в виде гипертекстовых документов).

Основой HTTP 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.

* **Название:**Hypertext Transfer Protocol
* **Уровень (по моделиOSI):**Прикладной
* **Семейство:**TCP/IP
* **Создан в:**1990 г.
* **Порт/ID:**80/TCP
* **Назначение протокола:** Доступ к гипертексту, ныне стал универсальным
* **Спецификация:**RFC 1945, RFC 2616
* **Основные реализации (клиенты):**Веб-браузеры, например Internet Explorer, Mozilla Firefox, Opera, Google Chrome и др.

**Основные реализации (серверы):**Apache, IIS

* **HTTP** используется также в качестве «транспорта» для других протоколов прикладного уровня, таких как **SOAP**, **XML-RPC**, **WebDAV**.
* Основным объектом манипуляции в **HTTP** является ресурс, на который указывает **URI** (англ. Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть 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 идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов **HTTP** использует глобальные **URI**. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ». Компоненты, использующие HTTP, могут самостоятельно осуществлять 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 осведомлён о предыдущих запросах и ответах, в нём не предусмотрена внутренняя поддержка состояния, к нему не предъявляются такие требования.

Достоинства http

* **Простота**

Протокол настолько прост в реализации, что позволяет с лёгкостью создавать клиентские приложения.

* **Расширяемость**

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

* **Распространённость**

При выборе протокола HTTP для решения конкретных 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

* **Отсутствие «навигации»**

Хотя протокол разрабатывался как средство работы с ресурсами сервера, у него отсутствуют в явном виде средства навигации среди 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 HTTP разрабатывался для решения типичных бытовых задач, где само по себе время обработки запроса должно занимать незначительное время или вообще не приниматься в расчёт. Но в промышленном использовании с применением распределённых вычислений при высоких нагрузках на сервер протокол HTTP оказывается беспомощен.

В 1998 году W3C предложил альтернативный протокол HTTP-NG (англ. HTTP Next Generation) для полной замены устаревшего с акцентированием внимания именно на этой области. Идею его необходимости поддержали крупные специалисты по распределённым вычислениям, но данный протокол до сих пор находится на стадии разработки.

Всё программное обеспечение для работы с протоколом HTTP разделяется на три больших категории:

\* **Серверы** как основные 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 (рус. исходный сервер). Разумеется, один и тот же программный продукт может одновременно выполнять функции клиента, сервера или посредника в зависимости от поставленных задач. В спецификациях протокола HTTP подробно описывается поведение для каждой из этих ролей.

История развития

* **HTTP/0.9**

HTTP был предложен в марте 1991 года **Тимом Бернерсом-Ли**, работавшим тогда в CERN, как механизм для доступа к документам в Интернете и облегчения навигации посредством использования гипертекста. Самая ранняя версия протокола HTTP/0.9 была впервые опубликована в январе 1992 г. (хотя реализация датируется 1990 годом). Спецификация протокола привела к упорядочению правил взаимодействия между клиентами и серверами HTTP, а также чёткому разделению функций между этими двумя компонентами. Были задокументированы основные синтаксические и семантические положения.

* **HTTP/1.0**

В мае 1996 года для практической реализации HTTP был выпущен информационный документ **RFC 1945**, что послужило основой для реализации большинства компонентов HTTP/1.0.

* **HTTP/1.1**

«Текущая» версия протокола, принята в июне 1999 года. Новым в этой версии был режим «постоянного соединения»: TCP-соединение может оставаться открытым после отправки ответа на запрос, что позволяет посылать несколько запросов за одно соединение. Клиент теперь обязан посылать информацию об имени хоста, к которому он обращается, что сделало возможной более простую организацию виртуального хостинга.

Структура http

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

1. **Стартовая строка** (англ. Starting line) — определяет тип сообщения;

2. **Заголовки** (англ. Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;

3. **Тело сообщения** (англ. Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа. Исключением является версия 0.9 протокола, у которой сообщение запроса содержит только стартовую строку, а сообщения ответа только тело сообщения.

Стартовые строки различаются для запроса и ответа. Строка запроса выглядит так:

**GET URI** — для версии протокола 0.9.

**Метод URI HTTP/Версия** — для остальных версий.

Здесь:

\* **Метод (англ. Method)** — название запроса, одно слово заглавными буквами. В версии HTTP 0.9 использовался только метод GET, список запросов для версии 1.1 представлен ниже.

\* **URI** определяет путь к запрашиваемому документу.

\* **Версия (англ. Version)** — пара разделённых точкой арабских цифр. Например: 1.0.

Для запроса страницы, клиент должен передать строку:

**GET /net/index.html HTTP/1.0**

**Метод HTTP (англ. HTTP Method)** — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Обратите внимание, что название метода чувствительно к регистру.

Каждый сервер обязан поддерживать как минимум методы **GET** и **HEAD**. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус **501 (Not Implemented).** Если серверу метод известен, но он не применим к конкретному ресурсу, то возвращается сообщение с кодом **405 (Method Not Allowed).** В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

Кроме методов **GET** и **HEAD**, часто применяется метод **POST**.

Методы http

OPTIONS

Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. В ответ серверу следует включить заголовок Allow со списком поддерживаемых методов. Также в заголовки ответа может включаться информация о поддерживаемых расширениях.

Предполагается, что запрос клиента может содержать тело сообщения для указания интересующих его сведений. Формат тела и порядок работы с ним в настоящий момент не определён. Сервер пока должен его игнорировать. Аналогичная ситуация и с телом в ответе сервера.

Для того чтобы узнать возможности всего сервера, клиент должен указать в URI звёздочку — «\*». Запросы «OPTIONS \* HTTP/1.1» могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1.

Результат выполнения этого метода не кэшируется.

GET

Используется для запроса содержимого указанного ресурса. С помощью **метода GET** можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»:

**GET /path/resource?param1=value1&param2=value2 HTTP/1.1**

Согласно стандарту HTTP, запросы типа GET считаются идемпотентными — многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам (при условии, что сам ресурс не изменился за время между запросами). Это позволяет кэшировать ответы на запросы GET.

Кроме обычного метода GET, различают ещё **условный GET** и **частичный GET**. Условные запросы GET содержат заголовки **If-Modified-Since, If-Match, If-Range** и подобные. Частичные GET содержат в запросе **Range**. Порядок выполнения подобных запросов определён стандартами отдельно.

HEAD

Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.

Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая.

POST

Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в **HTML-форму**, после чего они передаются серверу методом **POST** и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы на сервер.

В отличие от метода **GET**, метод **POST** не считается идемпотентным, то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться одна копия этого комментария).

При результатах выполнения **200 (Ok)** и **204 (No Content)** в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ **201 (Created)** с указанием URI нового ресурса в заголовке Location.

Сообщение ответа сервера на выполнение метода POST не кэшируется.

PUT

Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус **201 (Created).** Если же был изменён ресурс, то сервер возвращает **200 (Ok)** или **204 (No Content).** Сервер не должен игнорировать некорректные заголовки Content-\* передаваемые клиентом вместе с сообщением. Если какой-то из этих заголовков не может быть распознан или не допустим при текущих условиях, то необходимо вернуть код ошибки **501 (Not Implemented).**

Фундаментальное различие методов **POST** и **PUT** заключается в понимании предназначений **URI ресурсов**. Метод POST предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого. Используя PUT, клиент предполагает, что загружаемое содержимое соответствует находящемуся по данному URI ресурсу.

Сообщения ответов сервера на метод PUT не кэшируются.

Дополнительные методы

* **PATCH**

Аналогично PUT, но применяется только к фрагменту ресурса.

* **DELETE**

Удаляет указанный ресурс.

* **TRACE**

Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные сервера добавляют или изменяют в запросе.

* **LINK**

Устанавливает связь указанного ресурса с другими.

* **UNLINK**

Убирает связь указанного ресурса с другими.

Коды состояния

**Код состояния** является частью первой строки ответа сервера. Он представляет собой целое число из трех арабских цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа.

Примеры:

200 Ok

201 Webpage Created

403 Access allowed only for registered users

507 Insufficient Storage

Классы кодов состояния:

* **1xx Informational (русск. Информационный)**

В этот класс выделены коды, информирующие о процессе передачи. В HTTP/1.0 сообщения с такими кодами должны игнорироваться. В HTTP/1.1 клиент должен быть готов принять этот класс сообщений как обычный ответ, но ничего отправлять серверу не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. Прокси-сервера подобные сообщения должны отправлять дальше от сервера к клиенту.

* **2xx Success (русск. Успешно)**

Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.

* **3xx Redirection (русск. Перенаправление)**

Коды класса 3xx сообщают клиенту что для успешного выполнения операции необходимо сделать другой запрос (как правило по другому URI). Из данного класса пять кодов 301, 302, 303, 305 и 307 относятся непосредственно к перенаправлениям (жарг. редирект). Адрес, по которому клиенту следует произвести запрос, сервер указывает в заголовке Location. При этом допускается использование фрагментов в целевом URI.

* **4xx Client Error (русск. Ошибка клиента)**

Класс кодов 4xx предназначен для указания ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя.

Для запоминания значений кодов с 400 по 417 существуют приёмы иллюстративной мнемотехники.

* **5xx Server Error (русск. Ошибка сервера)**

Коды 5xx выделены под случаи неудачного выполнения операции по вине сервера. Для всех ситуаций, кроме использования метода HEAD, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.

Заголовки

**Заголовки HTTP** (англ. HTTP Headers) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых сообщений ARPA (см. RFC 822). Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

**Примеры заголовков:**

Server: Apache/2.2.11 (Win32) PHP/5.3.0

Last-Modified: Sat, 16 Jan 2017 21:16:42 GMT

Content-Type: text/plain; charset=windows-1251

Content-Language: ru

В примере выше каждая строка представляет собой один заголовок. При этом то, что находится до первого двоеточия, называется **именем** (англ. name), а что после неё — **значением** (англ. value).

Основные группы заголовков

1. **General Headers** (русск. Основные заголовки) — должны включаться в любое сообщение клиента и сервера.

2. **Request Headers** (русск. Заголовки запроса) — используются только в запросах клиента.

3. **Response Headers** (русск. Заголовки ответа) — только для ответов от сервера.

4. **Entity Headers** (русск. Заголовки сущности) — сопровождают каждую сущность сообщения.

Именно в таком порядке рекомендуется посылать заголовки получателю.

Примеры диалогов http:

**Обычный GET-запрос**

**Запрос клиента:**

GET /wiki/*страница* HTTP/1.1 Host: ru.wikipedia.org User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5 Accept: text/html Connection: close

**Ответ сервера:**

HTTP/1.0 200 OK

Date: Wed, 11 Feb 2016 11:20:59 GMT

Server: Apache

X-Powered-By: PHP/5.2.4-2ubuntu5wm1

Last-Modified: Wed, 11 Feb 2016 11:20:59 GMT

Content-Language: ru

Content-Type: text/html; charset=utf-8

Content-Length: 1234

Connection: close

(далее следует запрошенная страница в HTML)

Перенаправления

**Запрос:**

GET /about.html HTTP/1.1 Host: www.example-corp.com User-Agent: MyLonelyBrowser/5.0

**Ответ:**

HTTP/1.x 301 Moved Permanently

Location: http://www.example.com/about.html#contacts

Date: Thu, 19 Feb 2016 11:08:01 GMT

Server: Apache/2.2.4

Content-Type: text/html; charset=windows-1251

Content-Length: 110

(пустая строка)

<html><body><a href="http://www.example.com/about.html#contacts">Click here</a></body></html>

* Допустим, вымышленная организация предлагает скачать с сайта видео прошедшей конференции по адресу http://example.org/conf-2016.avi объёмом примерно 160 МБ. Рассмотрим, как происходит докачивание этого файла в случае сбоя и как менеджер закачек организовал бы многопоточную загрузку нескольких фрагментов.
* В обоих случаях клиенты произведут свой первый запрос наподобие этого:

GET /conf-2016.avi HTTP/1.0

Host: example.org

Accept: \*/\*

User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 7)

Referer: http://example.org/

* Заголовок Referer указывает, что файл был запрошен с главной страницы сайта. Менеджеры закачек обычно тоже его указывают, чтобы эмулировать переход со страницы сайта. Без него сервер может ответить 403 (Access Forbidden), если не допускаются запросы с других сайтов. В нашем случае сервер вернул успешный ответ:

HTTP/1.1 200 OK

Date: Thu, 19 Feb 2016 12:27:04 GMT

Server: Apache/2.2.3

Last-Modified: Wed, 18 Jun 2016 16:05:58 GMT

ETag: "56d-9989200-1132c580"

Content-Type: video/x-msvideo

Content-Length: 160993792

Accept-Ranges: bytes

Connection: close

(пустая строка)

(двоичное содержимое всего файла)

Заголовок Accept-Ranges информирует клиента о том, что он может запрашивать у сервера фрагменты, указывая их смещения от начала файла в байтах. Если этот заголовок отсутствует, то клиент может предупредить пользователя, что докачать файл, скорее всего, не удастся. Исходя из значения заголовка Content-Length, менеджер закачек поделит весь объём на равные фрагменты и запросит их по отдельности, организовав несколько потоков. Если сервер не укажет размер, то клиенту параллельное скачивание реализовать не удастся, но при этом он сможет докачивать файл, пока сервер не ответит 416 (Requested Range Not Satisfiable).

Допустим, на 84-ом мегабайте соединение с Интернетом прервалось и процесс загрузки приостановился. Когда соединение с Интернетом было восстановлено, браузер автоматически послал новый запрос на сервер, но с указанием выдать содержимое с 84-ого мегабайта:

GET /conf-2016.avi HTTP/1.0

Host: example.org

Accept: \*/\*

User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows 7)

Range: bytes=88080384-

Referer: http://example.org/

Сервер не обязан помнить, какие и от кого запросы были до этого, и поэтому клиент снова вставил заголовок Referer, как будто это его самый первый запрос. Указанное значение заголовка Range говорит серверу — «выдай содержимое от 88080384-ого байта до самого конца». В связи с этим сервер вернёт ответ:

HTTP/1.1 206 Partial Content

Date: Thu, 19 Feb 2016 12:27:08 GMT

Server: Apache/2.2.3

Last-Modified: Wed, 18 Jun 2016 16:05:58 GMT

ETag: "56d-9989200-1132c580"

Accept-Ranges: bytes

Content-Range: bytes 88080384-160993791/160993792

Content-Length: 72913408

Connection: close

Content-Type: video/x-msvideo

(пустая строка)

(двоичное содержимое от 84-ого мегабайта)

Большинство протоколов предусматривают установление TCP-сессии, в ходе которой один раз происходит авторизация, и дальнейшие действия выполняются в контексте этой авторизации. HTTP же устанавливает отдельную TCP-сессию на каждый запрос; в более поздних версиях HTTP было разрешено делать несколько запросов в ходе одной TCP-сессии, но браузеры обычно запрашивают только страницу и включённые в неё объекты (картинки, каскадные стили и т. п.), а затем сразу разрывают TCP-сессию. Для поддержки авторизованного (неанонимного) доступа в HTTP используются cookies; причём такой способ авторизации позволяет сохранить сессию даже после перезагрузки клиента и сервера.

При доступе к данным по FTP или по файловым протоколам тип файла (точнее, тип содержащихся в нём данных) определяется по расширению имени файла, что не всегда удобно. HTTP перед тем, как передать сами данные, передаёт в заголовке строчку «Content-Type: тип/подтип», позволяющую клиенту однозначно определить, каким образом обрабатывать присланные данные. Это особенно важно при работе с CGI-скриптами, когда расширение имени файла указывает не на тип присылаемых клиенту данных, а на необходимость запуска данного файла на сервере и отправки клиенту результатов работы программы, записанной в этом файле (при этом один и тот же файл в зависимости от аргументов запроса и своих собственных соображений может порождать ответы разных типов — в простейшем случае картинки в разных форматах).

Кроме того, HTTP позволяет клиенту прислать на сервер параметры, которые будут переданы запускаемому CGI-скрипту. Для этого же в HTML были введены формы.

Перечисленные особенности HTTP позволили создавать поисковые машины (первой из которых стала AltaVista, созданная фирмой DEC), форумы и Internet-магазины. Это превратило Internet из «академической игрушки» в «коммерческий сервис»: появились компании, основным полем деятельности которых стало предоставление доступа в Internet (компании-провайдеры) и создание сайтов.

# Форматы сообщений

RFC 5322 — Формат сообщений Internet

1. Введение

1.1. Рамки документа

Этот документ задает формат сообщений Internet (IMF) — синтаксис текстовых сообщений, передаваемых между пользователями компьютеров через систему электронной почты. Данная спецификация является пересмотром RFC 2822 [RFC2822], который, в свою очередь, пересматривает RFC 822, "Стандарт для формата текстовых сообщений ARPA Internet" [RFC0822], с учетом опыта использования и накопленных изменений, отраженных в других RFC (таких, как [RFC1123]).

В этом документе задан синтаксис только для текстовых сообщений.

В контексте электронной почты сообщения представляются состоящими из конверта и тела (содержимого) письма. Конверт содержит информацию, требуемую для [передачи и доставки сообщения](http://rfc2.ru/5321.rfc). Тело письма является объектом, который доставляется адресату. Данная спецификация применима только к формату и части семантики содержимого писем. Документ не включает спецификации данных, включаемых в конверт сообщения.

Однако некоторые системы обработки сообщений могут использовать информацию из тела письма для создания конверта. Одной из задач настоящей спецификации является упрощение извлечения такой информации программными средствами.

Назначением этой спецификации является задание формата содержимого писем, передаваемых между системами.

2. Лексический анализ сообщений

2.1. Общее описание

На базовом уровне сообщение представляет собой последовательность символов. Сообщения, соответствующие данной спецификации, включают символы с десятичными кодами от 1 до 127, интерпретируемые в соответствии с кодировкой US-ASCII [ANSI.X3-4.1986]. Для краткости в этом документе такие символы будут называться просто «символами US-ASCII».

Символы сообщения делятся на строки. Строка представляет собой последовательность символов, завершающуюся кодами возврата каретки и перевода строки — в конце строки помещается символ возврата каретки (CR, десятичный код ASCII — 13), непосредственно за которым следует символ перевода строки (LF, десятичный код ASCII - 10). В данном документе такая последовательность обозначается «CRLF».

Сообщение состоит из полей заголовков (совокупность этих полей называют разделом заголовков сообщения), за которыми может следовать тело сообщения. Раздел заголовков представляет собой последовательность символьных строк, синтаксис которых описан в данной спецификации. Тело сообщения представляет собой последовательность символов, которая следует после раздела заголовков и отделена от него пустой строкой (строкой, содержащей только CRLF).

**2.1.1. Предельные размеры строк**

Данная спецификация вносит два ограничения на число символов в строке. Строка должна содержать не более 988 символов; следует использовать строки размером не более 78, без учета CRLF.

Ограничение в 998 обусловлено возможностями множества реализаций в части передачи, приема и хранения, которые не позволяют работать с сообщениями IMF, содержащими строки размером более 998 символов. Системы приема могут из соображений отказоустойчивости отказываться от ограничения на размер строк. Однако существует множество реализаций, которые не принимают сообщения со строками размером более 1000 (включая CR и LF в конце строки), — разработчикам важно принимать этот факт во внимание.

Рекомендация использовать строки размером не более 78 обусловлена параметрами пользовательского интерфейса многих реализаций, которые могут отсекать лишние символы или неаккуратно переносить слова при наличии в строке более 78 символов, несмотря на то, что такие реализации не соответствуют требованиям данной спецификации и приводят к потере символов. Однако наличие такого ограничения не запрещает реализациям корректно отображать сообщения со строками произвольной длины для повышения уровня отказоустойчивости.

2.2. Поля заголовков

Поля заголовков представляют собой строки, начинающиеся с имени поля, за которым следует двоеточие (":"), содержимое поля и знак завершения строки CRLF. 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 (HTAB, код ASCII — 9), которые вместе называют также пробельными символами. В значение поля недопустимо включать символы CR и LF, за исключением их использования в «фальцованных» и «нефальцованных» полях, как описанов в параграфе 2.2.3. Значения полей должны соответствовать синтаксису, описанному в разделах 3 и 4 настоящей спецификации.

**2.2.1. Бесструктурные поля заголовков**

Некоторые поля заголовков в этой спецификации определены просто как неструктурированные (эти поля содержат произвольный набор печатаемых символов US-ASCII и пробельных символов) без дополнительных ограничений. Такие поля будем называть бесструктурными. Семантически бесструктурное поле трактуется просто как строка символов без дополнительной обработки.

**2.2.2. Структурированные поля заголовков**

Синтаксис некоторых полей в данной спецификации вносит дополнительные ограничения по сравнению с описанными выше бесструктурными полями. Такие поля называются структурированными. Структурированное поле представляет собой последовательность лексем. Многие из таких лексем (в соответствии с их синтаксисом) допускают включение комментариев в начале или в конце лексемы, а также пробельных символов, которые могут использоваться для фальцовки.

**2.2.3. Длинные поля заголовков**

Каждое поле заголовка логически представляет собой строку символов, состоящую из имени поля, двоеточия и тела (значения) поля. Однако для удобства и с учетом ограничения размеров строки (998/78 символов), значение поля может быть разбито на несколько строк; это называется «фальцовкой» (folding). Общим правило заключается в том, что данная спецификация разрешает включение последовательности CRLF (новая строка) перед любыми пробельными символами.

Например, поле заголовка

Subject: This is a test

можно записать в форме

Subject: This

is a test

Процесс преобразования фальцованного многострочного представления поля в обычное однострочное называется расфальцовкой (unfolding) и выполняется путем простого удаления всех последовательностей CRLF, непосредственно за которыми следуют пробельные символы (WSP). Каждое поле заголовка для дальнейшего синтаксического и семантического анализа следует трактовать в его нефальцованном представлении. На нефальцованные поля заголовков не накладываются ограничения по размеру и они, следовательно, могут иметь любую длину.

2.3. Тело письма

Тело сообщения представляет собой простые строки символов US-ASCII. Для содержимого сообщения существует только два типа ограничений:

* символы CR и LF должны использоваться только совместно, как CRLF; недопустимо использование этих символов в теле сообщения по-отдельности;
* строки символов должны быть не длиннее 998 символов, следует ограничивать размер строк 78 символами (без учета CRLF).

3. Синтаксис

3.1. Введение

Описанный в этом разделе синтаксис задает корректный формат почтовых сообщений Internet. Сообщения, соответствующие данной спецификации.

Для определяемых выражений дается краткое описание синтаксиса и применения, после чего приводится синтаксис в формате ABNF и семантический анализ. Часть примитивов, используемых в документе, не определена здесь и заимствована из основных правил, приведенных в документа [RFC5234]. К таким примитивам относятся: CR, LF, CRLF, HTAB, SP, WSP, DQUOTE, DIGIT, ALPHA и VCHAR.

В некоторых определениях будут встречаться элементы, имена которых начинаются с префикса "obs-". Такие элементы обозначают лексемы, относящиеся к устаревшему синтаксису. Во всех случаях такие конструкции при генерации корректных почтовых сообщений Internet игнорируются и их недопустимо использовать в качестве компонент сообщения. Однако при интерпретации сообщений такие лексемы должны рассматриваться как синтаксически корректные.

3.2. Лексемы

Разберем правила, используемые для определения базового лексического анализатора, который представляет лексемы анализаторам верхнего уровня. Разберем лексемы, используемые в структурированных полях заголовков.

**3.2.1. Квотирование символов**

Некоторые символы имеют 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

При появлении любой пары с квотированием (quoted-pair) она интерпретируется как отдельный символ. Т. е., символ \, являющийся частью пары с квотированием, становится семантически «невидимым».

**3.2.2. Пробелы для фальцовки и комментарии**

Пробельные символы, включая и те, которые служат для фальцовки, могут появляться между разными элементами в теле полей заголовков. Строки символов, трактуемые, как комментарии, также могут включаться в тело структурированных полей заголовков с заключением их в скобки.

Строки символов, заключенные в скобки, рассматриваются, как комментарии, если они не являются частью строки с квотированием. Комментарии могут быть вложенными.

В этой спецификации есть несколько мест, где комментарии и FWS могут вставляться свободно. Для согласования с таким синтаксисом определена дополнительная лексема CFWS, показывающая возможность включения комментариев и/или FWS. Однако в тех случаях, где в данной спецификации разрешается CFWS, недопустимо использовать их так, чтобы фальцованное поле заголовка включало только символы WSP и ничего другого.

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

В этой спецификации появление FWS (пробельные символы для фальцовки) означает указание места, где возможно выполнение фальцовки. Всякий раз при использовании фальцовки в сообщении должна выполняться расфальцовка до любого семантического анализа, выполняемого по отношению к заголовку в соответствии с данной спецификацией. Т. е., любые последовательности CRLF, включенные в FWS, являются семантически невидимыми.

Комментарии обычно используются в теле структурированных полей для обеспечения некой дополнительной информации для человека. Поскольку в комментариях могут содержаться символы FWS, это позволяет выполнять фальцовку внутри комментария. Отметим также, что возможность использования в комментариях пар с квотированием, позволяет включать в комментарии скобки и символы обратной дробной черты (\), если они задаются в форме пары с квотированием. Семантически внешние скобки не являются частью комментария — комментарием является то, что заключено в эти скобки. Как было отмечено выше, символы «\» в парах с квотированием и последовательности CRLF внутри FWS внутри комментариев являются семантически невидимыми и, следовательно, не являются частью комментария.

FWS в качестве комментария (CFWS) между лексемами в структурированном поле заголовка семантически интерпретируется как один символ пробела.

3.2.3. Атом

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

В некоторых структурированных полях заголовков допускается включение точки («.», код ASCII - 46) в atext. Для таких конструкций определена дополнительная лексема «атом с точкой» (dot-atom).

atext = ALPHA / DIGIT / ; Печатаемые символы US-ASCII,

"!" / "#" / ; не включая специальных символов.

"$" / "%" / ; Используются для атомов.

"&" / "'" /

"\*" / "+" /

"-" / "/" /

"=" / "?" /

"^" / "\_" /

"`" / "{" /

"|" / "}" /

"~"

atom = [CFWS] 1\*atext [CFWS]

dot-atom-text = 1\*atext \*("." 1\*atext)

dot-atom = [CFWS] dot-atom-text [CFWS]

specials = "(" / ")" / ; Специальные символы, которые не

"<" / ">" / ; появляются в atext

"[" / "]" /

":" / ";" /

"@" / "\" /

"," / "." /

DQUOTE

Лексемы atom и dot-atom интерпретируются, как единый элемент, включающий строку символов. Семантически дополнительные комментарии и FWS, окружающие остальные символы, не являются частью атома — атом представляет собой только символы atext (или atext и «.» для dot-atom).

**3.2.4. Строки в кавычках**

Строки, включающие символы, недопустимые для использования в атомах, могут быть представлены с использованием двойных кавычек (DQUOTE, код ASCII - 34), окружающих такие символы.

qtext = %d33 / ; Печатаемые символы US-ASCII,

%d35-91 / ; не включая "\"

%d93-126 / ; и символа кавычек

obs-qtext

qcontent = qtext / quoted-pair

quoted-string = [CFWS]

DQUOTE \*([FWS] qcontent) [FWS] DQUOTE

[CFWS]

Строка в кавычках трактуется как единый элемент. Т. е., семантически строка в кавычках эквивалентна атому. Поскольку строки в кавычках могут содержать FWS, в них возможна фальцовка. Отметим также, что благодаря возможности использования внутри кавычек пар с квотированием, строки в кавычках могут содержать также символы кавычек и обратной дробной черты, если они представлены в форме пар с квотированием.

Семантически, ни опциональные CFWS за пределами кавычек, ни сами символы кавычек не являются частью строки в кавычках — к такой строке относятся только символы, расположенные между кавычками. Как было отмечено выше, символ «\» в паре с квотированием или CRLF в FWS/CFWS, включенные в строку в кавычках, семантически невидимы и, следовательно, не являются частью строки в кавычках.

**3.2.5. Прочие лексемы**

Определены и дополнительных лексемы: слово (word) и фраза (phrase) для комбинаций атомов и/или строк в кавычках и неструктурированный текст (unstructured) для использования в бесструктурных полях заголовков и в некоторых местах структурированных полей.

word = atom / quoted-string

phrase = 1\*word / obs-phrase

unstructured = (\*([FWS] VCHAR) \*WSP) / obs-unstruct

3.3. Задание даты и времен

Значения даты и времени появляются в нескольких полях заголовка. В этом параграфе определяет синтаксис для задания даты и времени. Хотя спецификация даты/времени допускает включение пробельных символов фальцовки, рекомендуется использовать один пробел в каждом случае включения FWS (так, где это требуется или допускается); некоторые старые реализации неспособны корректно интерпретировать пробельные символы фальцовки.

date-time = [ day-of-week "," ] date time [CFWS]

day-of-week = ([FWS] day-name) / obs-day-of-week

day-name = "Mon" / "Tue" / "Wed" / "Thu" /

"Fri" / "Sat" / "Sun"

date = day month year

day = ([FWS] 1\*2DIGIT FWS) / obs-day

month = "Jan" / "Feb" / "Mar" / "Apr" /

"May" / "Jun" / "Jul" / "Aug" /

"Sep" / "Oct" / "Nov" / "Dec"

year = (FWS 4\*DIGIT FWS) / obs-year

time = time-of-day zone

time-of-day = hour ":" minute [ ":" second ]

hour = 2DIGIT / obs-hour

minute = 2DIGIT / obs-minute

second = 2DIGIT / obs-second

zone = (FWS ( "+" / "-" ) 4DIGIT) / obs-zone

Дата (day) показывает порядковый номер дня в месяце. Год (year) может принимать любое значение, не ранее 1900.

Время суток (time-of-day) показывает число часов, минут и (опционально) секунд, прошедщих с полуночи указанной даты.

Значения date и time-of-day следует указывать по местному времени.

Часовой пояс (zone) показывает смещение относительно универсального времени (UTC, ранее использовался термин GMT) значений местного времени, указанного в date и time-of-day. Знаки «+» и «-» показывают направление смещения — вперед (т. е., к востоку) или назад (к западу) от универсального времени. Первые две цифры показывают разницу с универсальным временем в часах, а две последних — дополнителюную разницу в минутах. Следовательно, +hhmm означает смещение на +(hh \* 60 + mm) минут от универсального времени, а -hhmm — на -(hh \* 60 + mm) минут. Для индикации часового пояса, время которого совпадает с универсальным, следует использовать форму +0000. Хотя вариант -0000 указывает на тот же часовой пояс, этот вариант используется для индикации того, что время было указано системой, которая может находиться в часовом поясе, отличном от UT, а значение date-time не содержит информации о часовом поясе.

Значение date-time должно быть семантически корректным. Т. е., значение day-of-week (при его наличии) должно содержать день недели, числовое значение day-of-month должно находиться в диапазоне от 1 до числа дней в соответствующем месяце (указанного года), значение time-of-day должно находиться в диапазоне от 00:00:00 до 23:59:60 (число секунд, допустимое для смены суток; см. [RFC1305]), а две последних цифры поля zone должны иметь значение от 00 до 59.

3.4. Задание адреса

Адреса появляются в нескольких полях заголовка для индикации отправителей и получателей сообщения. Адрес может задавать персональный почтовый ящик или группу почтовых ящиков.

address = mailbox / group

mailbox = name-addr / addr-spec

name-addr = [display-name] angle-addr

angle-addr = [CFWS] "<" addr-spec ">" [CFWS] /

obs-angle-addr

group = display-name ":" [group-list] ";" [CFWS]

display-name = phrase

mailbox-list = (mailbox \*("," mailbox)) / obs-mbox-list

address-list = (address \*("," address)) / obs-addr-list

group-list = mailbox-list / CFWS / obs-group-list

Почтовый ящик принимает почту. Это концептуальный объект, который не обязательно связан с файловым хранилищем. Например, некоторые сайты могут выводить почтовые сообщения на принтер или передавать их на специальные адресуемые устройства вывода.

Обычно почтовый ящик состоит из двух частей: (1) необязательное отображаемое имя, которое идентифицирует имя получателя (человека или системы) и может выводиться пользователю в почтовых программах и (2) поле addr-spec, заключенное в угловые скобки («<» и «>»). Существует дополнительная форма указания почтового ящика, в которой имя получателя отсутствует, а addr-spec указывается без угловых скобок.

Примечание. Некоторое унаследованные реализации используют простую форму, где addr-spec указывается без угловых скобок, а имя получателя указывается в скобках, как комментарий вслед за addr-spec. Поскольку трактовка комментариев не задается спецификацией, реализациям для задания связанного с почтовым ящиком отображаемого имения следует использовать полную форму name-addr взамен такой унаследованной формы. Кроме того, поскольку некоторые унаследованные реализации интерпретируют комментарии, в общем случае не следует использовать комментарии в поле адреса во избежание возможной путаницы.

Когда желательно трактовать несколько почтовых ящиков, как один объект (например, список рассылки), может использоваться групповая конструкция. Такая конструкция позволяет отправителю указать именованную группу получателей. Это обеспечивается путем создания для группы отображаемого имени, за которым следует список разделенных запятыми почтовых ящиков произвольного (включая 0) размера, завершающийся точкой с запятой (;). Поскольку список почтовых ящиков может быть пустым, использование групповой конструкции также обеспечивает простой способ взаимодействия с адресатами, когда сообщение передается одной или множеству именованных групп адресатов без указания иднивидуальных почтовых ящиков этих адресатов.

**3.4.1. Задание addr-spec**

Поле addr-spec представляет собой специфический для Internet идентификатор, содержащий локально интерпретируемую строку, за которой следует символ @ (код ASCII - 64) и доменное имя Internet. Эта локально интерпретируемая строка представляет собой строку в кавычках или атом с точкой. Если строка может быть представлена атомом с точкной (т. е., не содержит символов, кроме atext и завершающей точки или, за которой следуют символы atext), следует использовать форму dot-atom и не следует применять форму quoted-string. Комментарии и пробельные символы фальцовки не следует помещать рядом с @ в поле addr-spec.

addr-spec = local-part "@" domain

local-part = dot-atom / quoted-string / obs-local-part

domain = dot-atom / domain-literal / obs-domain

domain-literal = [CFWS] "[" \*([FWS] dtext) [FWS] "]" [CFWS]

dtext = %d33-90 / ; Печатаемые символы US-ASCII,

%d94-126 / ; не включая

obs-dtext ; "[", "]", or "\"

Доменная часть идентифицирует точку, в которую доставляется почта. В форме dot-atom она интерпретируется, как доменное имя Internet. В форме domain-literal доменная часть интерпретируется, как «дословный» Internet-адрес конкретного хоста. В обоих случаях использование адресации и транспортировка почты на конкретный хост определяются отдельными документами.

Локальная часть (local-part) зависит от домена. В адресах она просто интерпретируется на конкретном хосте, как имя почтового ящика.

3.5. Общий синтаксис сообщений

Сообщение состоит из полей заголовка, за которыми может следовать тело сообщения. Строки сообщения должны иметь размер не более 998 символов, не считая CRLF, но рекомендуется использовать строки не длиннее 78 без учета CRLF. Хотя в теле сообщения могут появляться любые символы, перечисленные в правиле text, использование управляющих символов US-ASCII (коды 1-8, 11, 12, 14-31) не является хорошим тоном, поскольку их интерпретация при отображении на приемной стороне не гарантируется.

message = (fields / obs-fields)

[CRLF body]

body = (\*(\*998text CRLF) \*998text) / obs-body

text = %d1-9 / ; Символы, за исключением

%d11 / ; CR и LF

%d12 /

%d14-127

Тело сообщения представляет собой просто набор строк текста, интерпретируемых в соответствии с настоящей спецификацией.

3.6. Определения полей

Все поля заголовка имеют одинаковую синтаксическую структуру: имя поля, за которым следует двоеточие (:) и тело (значение) поля.

Важно подчеркнуть, что порядок следования полей заголовка не гарантируется. Поля заголовков могут появляться в произвольном порядке. Более того, известно, что порядок полей заголовков может изменяться при передаче сообщений через Internet. Однако в соответствиии с данной спецификацией порядок полей заголовка не следует менять при передаче или преобразовании сообщений. Более важно отметить, что порядок трассировочных полей и полей resent изменять недопустимо и следует сохранять эти поля в блоках, добавляемых в начало сообщения (prepend).

Обязательными полями заголовка являются только поле даты и поле адреса отправителя сообщения. Все остальные поля являются синтаксически опциональными.

fields = \*(trace

\*optional-field /

\*(resent-date /

resent-from /

resent-sender /

resent-to /

resent-cc /

resent-bcc /

resent-msg-id))

\*(orig-date /

from /

sender /

reply-to /

to /

cc /

bcc /

message-id /

in-reply-to /

references /

subject /

comments /

keywords /

optional-field)

Приведенная ниже таблица показывает минимальное и максимальное число полей каждого типа в разделе заголовков сообщения, а также ограничения на использование полей. Звездочка (\*) после в колонке минимального или максимального числа полей говорит о наличии дополнительных ограничений, указанных в колонке «Примечания».

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Минимум | Максимум | Примечания |
| trace | 0 | Не ограничен | Блок добавляется в начало |
| resent-date | 0\* | Не ограничен\* | Одно на блок; требуется при наличии других полей resent |
| resent-from | 0 | Не ограничен\* | Одно на блок |
| resent-sender | 0\* | Не ограничен\* | Одно на блок; должно присутствовать при наличии множества адресов |
| resent-to | 0 | Не ограничен\* | Одно на блок |
| resent-cc | 0 | Не ограничен\* | Одно на блок |
| resent-bcc | 0 | Не ограничен\* | Одно на блок |
| resent-msg-id | 0 | Не ограничен\* | Одно на блок |
| orig-date | 1 | 1 |  |
| from | 1 | 1 |  |
| sender | 0\* | 1 | Должно присутствовать при наличии множества адресов |
| reply-to | 0 | 1 |  |
| to | 0 | 1 |  |
| cc | 0 | 1 |  |
| bcc | 0 | 1 |  |
| message-id | 0\* | 1 | Следует включать |
| in-reply-to | 0\* | 1 | Следует включать |
| references | 0\* | 1 | Следует включать |
| subject | 0 | 1 |  |
| comments | 0 | Не ограничен |  |
| keywords | 0 | Не ограничен |  |
| optional-field | 0 | Не ограничен |  |

Точная интерпретация каждого поля рассмотрена в последующих параграфах.

**3.6.1. Поле даты создания**

Поле даты создания состоит из имени поля «Date», за которым следуют значения даты и времени.

orig-date = "Date:" date-time CRLF

Поле указывает дату и время, когда отправитель, указанный в сообщении, завершил подготовку сообщения к передаче системе доставки. Например, это может быть момент нажатия пользователем кнопки «send» или «submit» в почтовой программе. В любом случае, это поле не предназначено для указания времени реальной передачи сообщения, а содержит значение времени, когда человек или другой создатель сообщения завершил подготовку сообщения к передаче (например, пользователь портативного компьютера, не подключенного к сети, мог просто поместить сообщение в очередь для доставки; поле даты создания предназначено для хранения даты и времени в момент постановки письма в очередь, а не момента подключения пользователя к сети).

**3.6.2. Поля источника**

В число полей источника сообщения входят from и sender (когда применим), а также опционально — reply-to. Поле from содержит имя «From» и разделенный запятыми список из одного или нескольких имен почтовых ящиков. Эсли это поле содержит более одного адреса почтового ящика в списке, в сообщение должно включаться поле «Sender» с указанием одного почтового ящика. Дополнительно может также включаться поле «Reply-To», содержащее список разделенных запятыми почтовых ящиков (не менее одного).

from = "From:" mailbox-list CRLF

sender = "Sender:" mailbox CRLF

reply-to = "Reply-To:" address-list CRLF

Поле источника показывает почтовый ящик(и) источника сообщения. Поле «From:» задает автора (авторов) сообщения, т. е., почтовый ящик(и) человека или системы, ответственных за создание данного сообщения. Поле «Sender:» указывает почтовый ящик агента, ответственного за реальную передачу сообщения. Например, если секретарь отправил письмо от имени своего руководителя, почтовый ящик секретаря будет указан в поле «Sender:», а адрес действительного автора сообщения — в поле «From:». Если источник сообщения может быть задан единственным почтовым ящиком (автор и отправитель совпадают), поле «Sender:» использовать не следует. В остальных случаях это поле следует включать в заголовок.

Поля источника также обеспечивают информацию, требуемую для ответа на сообщение. При наличии поля «Reply-To:» оно указывает адрес(а), по которому отправитель сообщения предлагает направлять ответ. В отсутствие поля «Reply-To:» ответ следует по умолчанию отправлять по адресу (адресам), указанному в поле «From:», если автором ответа явно не указан иной адрес.

В любом случае в поле «From:» не следует указывать адрес, не имеющий отношения к автору письма. Формирование адресов для отправки ответов на сообщения дополнительно рассматривается в параграфе 3.6.3.

**3.6.3. Поля адресов получателей**

К полям получателей относятся три однотипных поля, , каждое из которых содержит имя («To», «Cc» или «Bcc») и список из одного или множества разделенных запятыми адресов (почтовых ящиков или групп).

to = "To:" address-list CRLF

cc = "Cc:" address-list CRLF

bcc = "Bcc:" [address-list / CFWS] CRLF

Эти поля задают получателей сообщения. Каждое из полей может содержать один или множество адресов, идентифицирующих получателей сообщения. Различие между полями заключается только в способе их использования.

Поле «To:» содержит адрес(а) основного получателя (получателей) сообщения.

Поле «Cc:» (от «Carbon Copy» — копия по аналогии с печатью под копирку на пишущей машинке) содержит адреса других лиц, которым направляется это сообщение, хотя содержимое сообщения может быть не адресовано им напрямую.

Поле «Bcc:» (от «Blind Carbon Copy» — «слепая копия» по аналогии с последним экземпляром при печати под копирку) содержит адреса получателей, которые не будут показаны другим получателям этого сообщения. Существует три варианта использования поля «Bcc:». В первом случае, когда сообщение с полем «Bcc:» готовится к отправке, строка «Bcc:» удаляется из него, хотя все адресаты (включая указанных в поле «Bcc:») получают копию этого письма. Во втором варианте получатели, указанные в полях «To:» и «Cc:», получат сообщение с удаленной строкой «Bcc:», но адресаты, указанные в поле «Bcc:» получат копию письма с сохраненной строкой «Bcc:» (при указании в поле «Bcc:» множества адресатов некоторые реализации на самом деле шлют каждому адресату из поля «Bcc:» отдельную копию письма, содержащую в этом поле только адрес данного получателя). И, наконец, когда поле «Bcc:» не содержит адресов, это поле может включаться во все копии сообщения для адресатов, заданных другими полями. Выбор конкретного варианта использования поля «Bcc:» определяется реализацией, но при этом следует принимать во внимание информацию, приведенную ниже в разделе «Вопросы безопасности».

Когда сообщение является ответом на другое письмо, почтовые ящики авторов исходного письма (значение поля «From:») или почтовые ящики, указанные в поле «Reply-To:» (если оно есть), могут появляться в поле «To:» ответного письма, поскольку эти адресаты являются явными отправителями исходного письма. Если сообщение передается в ответ на письмо, имеющее поля получателей, зачастую бывает полезно отправить копию ответа всем получателям сообщения в дополнение к отправке письма автору. При создании такого отклика адреса из полей «To:» и «Cc:» исходного сообщения могут появляться в поле «Cc:» ответного письма, поскольку эти адресаты были открытоо указаны в числе получателей копии исходного письма. Если в исходном письме присутствует поле «Bcc:», адреса из этого поля могут появляться в поле «Bcc:» ответа на письмо, но их не следует включать в поле «To:» или «Cc:».

**3.6.4. Поля идентификации**

Хотя в таблице поле «Message-ID:» указано, как необязательное, это поле следует включать в каждое сообщение. Более того, в ответные сообщения следует включать поля «In-Reply-To:» и «References:» в соответствии с приведенным ниже описанием.

Поле «Message-ID:» содержит один уникальный идентификатор сообщения. Каждое из полей «References:» и «In-Reply-To:» содержит один или множество уникальных идентификаторов сообщений, опционально разделенных символами CFWS.

Синтаксис идентификатора сообщения (msg-id) является ограниченной версией конструкции addr-spec, заключенной в угловые скобки «<» и «>». В отличие от addr-spec, этот синтаксис разрешает форму dot-atom-text слева от символа @ и не имеет сиволов CFWS где-либо внутри идентификатора.

message-id = "Message-ID:" msg-id CRLF

in-reply-to = "In-Reply-To:" 1\*msg-id CRLF

references = "References:" 1\*msg-id CRLF

msg-id = [CFWS] "<" id-left "@" id-right ">" [CFWS]

id-left = dot-atom-text / obs-id-left

id-right = dot-atom-text / no-fold-literal / obs-id-right

no-fold-literal = "[" \*dtext "]"

Поле «Message-ID:» обеспечивает уникальный идентификатор сообщения, который указывает на конкретный вариант конкретного письма. Уникальность идентификатора гарантируется хостом, создающим сообщение. Этот идентификатор предназначен для машинной обработки и не имеет большого смысла для человека. Идентификатор сообщения относится только к одному варианту конкретного письма — для следующего экземпляра сообщения будет создан другой идентификатор.

Поля «In-Reply-To:» и «References:» используются при создании ответных сообщений. Они содержат идентификатор исходного сообщения и идентификаторы других сообщений (например, в случае ответа на письмо, которое само является ответом на другое письмо). Поле «In-Reply-To:» может использоваться для идентификации сообщения (сообщений), на которое отвечает данное сообщение, тогда как поле «References:» может служить для идентификации «ветви» разговора.

При создании ответа на сообщение поля «In-Reply-To:» и «References:» в ответном письме строятся в соответствии с приведенным ниже описанием.

Поле «In-Reply-To:» ответного письма будет содержать информацию из поля «Message-ID:» исходного («родительского») сообщения. Если ответ дается на несколько писем сразу, поле «In-Reply-To:» будет содержать информацию из полей «Message-ID:» всех родительских сообщений. Если в родительских сообщениях нет полей «Message-ID:», в ответе не будет содержаться поля «In-Reply-To:».

Поле «References:» будет включать содержимое поля «References:» из родительского письма (если там это поле присутствует), вслед за которым будет включено родительское поле «Message-ID:» (при его наличии). Если в родительском сообщении нет поля «References:», но имеется поле «In-Reply-To:» с единственным идентификатором, в ответе поле «References:» будет включать содержимое родительского поля «In-Reply-To:», за которым будет следовать содержимое родительского поля «Message-ID:» (при его наличии). Если в родительском сообщении нет полей «References:», «In-Reply-To:» и «Message-ID:», в ответе не будет поля «References:».

Идентификатор сообщения (msg-id) должен быть уникальным в глобальном масштабе. Эту уникальность должен обеспечивать генератор идентификаторов сообщений. Существует несколько алгоритмов, обеспечивающих решение этой задачи. Поскольку синтаксис msg-id подобен сиснтаксису addr-spec, хорошим методом является включение в идентификатор доменного имени (или полного адреса IP) хоста, на котором создается идентификатор сообщения, справа от знака @ (доменные имена и адреса IP обычно уникальны) и включение текущего абсолютного значения даты и времени в комбинации с неким другим уникальным в данный момент (возможно последовательным) идентификатором, доступным в системе (например, идентификатором процесса), слева от @. Хотя будут работать и другие алгоритмы, рекомендуется использовать в правой части некий идентификатор домена (имя хоста или нечто иное), чтобы генератор идентификатора сообщения мог гарантировать уникальность левой части идентификатора в масштабе данного домена.

Семантически угловые скобки не являются частью msg-id; идентификатором является строка символов между скобками.

**3.6.5. Информационные поля**

Информационные поля являются необязательными. Поля «Subject:» и «Comments:» являются бесструктурными и, следовательно, могут содержать текст или пробельные символы для фальцовки. Поле «Keywords:» содержит список из одного или более разделенных запятыми слов или строк в кавычках.

subject = "Subject:" unstructured CRLF

comments = "Comments:" unstructured CRLF

keywords = "Keywords:" phrase \*("," phrase) CRLF

Эти три поля предназначены только для включения понятной человеку информации о сообщении. Поле «Subject:» используется наиболее широко и содержит короткую строку, описывающую тему сообщения. При использовании в ответах это поле может начинаться с префикса «Re: » (сокращение от латинского «in re», означающего «по вопросу ..»), за которым следует содержимое поля «Subject:» исходного письма. В таких случаях следует использовать префикс «Re: » только один раз, поскольку использование другого текста или включение нескольких префиксов может приводить к нежелательным последствиям. Поле «Comments:» содержит произвольную информацию, дополняющую текст в теле письма. Поле «Keywords:» содержит список разделенных запятыми слов и фраз " field contains a comma-separated list of important words and phrases that might be useful for the recipient.

**3.6.6. Поля пересылки**

Поля пересылки следует добавлять во все сообщения, которые повторно вводятся пользователем в транспортную систему. Каждый раз, когда это происходит, следует добавлять отдельный набор полей пересылки. Все поля пересылки, соответствующие отдельному факту пересылки, следует группировать. Каждый набор полей пересылки помещается в начало сообщения; т. е., самый свежий набор таких полей появляется в сообщении первым. При добавлении полей пересылки никакие другие поля сообщения не меняются.

Каждое поле пересылки синтаксически соответствует определенному полю при обычной передаче. Например, поле «Resent-Date:» соответствует полю «Date:», а «Resent-To:» — полю «To:».

При использовании полей пересылки должны передаваться поля «Resent-From:» и «Resent-Date:». Следует передавать также поле «Resent-Message-ID:». Поле «Resent-Sender:» не следует использовать, если это поле будет идентично полю «Resent-From:».

resent-date = "Resent-Date:" date-time CRLF

resent-from = "Resent-From:" mailbox-list CRLF

resent-sender = "Resent-Sender:" mailbox CRLF

resent-to = "Resent-To:" address-list CRLF

resent-cc = "Resent-Cc:" address-list CRLF

resent-bcc = "Resent-Bcc:" [address-list / CFWS] CRLF

resent-msg-id = "Resent-Message-ID:" msg-id CRLF

Поля пересылки служат для идентификации сообщений, как повторно вводимых пользователем в транспортную систему. Цель использования этих полей заключается в предоставлении конечному адресату сообщения в таком виде, как будто оно было получено непосредственно от исходного отправителя с сохранением всех исходных полей. Каждый набор полей пересылки соответствует одному факту пересылки. Т. е., при многократной пересылке сообщения каждый из таких наборов описывает отдельный факт пересылки. Поля пересылки являются информационными. Эти поля недопустимо использовать в процессах автоматической генерации ответов и других операциях автоматизированной обработки сообщений.

Поля инициатора пересылки указывают почтовый ящик лиц(а) или систем(ы), переславших это сообщение. Как и для обычных полей инициатора существует две формы — простой вариант «Resent-From:», содержащий почтовый ящик выполняющего пересылку лица, и более сложный вариант, когда одно лицо (идентифицируется полем «Resent-Sender:») пересылает сообщение от имени одного или множества других лиц (идентифицируются полем «Resent-From:»).

Примечание. При ответе на пересланное сообщение исходные поля «From:», «Reply-To:», «Message-ID:» и др. используются в обычном порядке. Поля пересылки являются только информационными и недопустимо использовать их при ответе на сообщение.

Поле «Resent-Date:» указывает дату и время диспетчеризации сообщения при его пересылке. Подобно полю «Date:», оно содержит не дату и время реальной отправки сообщения, а дату и время постановки в очередь.

Функционально поля «Resent-To:», «Resent-Cc:» и «Resent-Bcc:» идентичны полям «To:», «Cc:» и «Bcc:», соответственно, за исключением того, что они указывают получателей пересланного сообщения, а не исходного.

Поле «Resent-Message-ID:» содержит уникальный идентификатор пересланного сообщения.

**3.6.7. Поля трассировки**

Поля трассировки представляют собой группу полей заголовка, включающую опциональное поле «Return-Path:» и одно или множество полей «Received:». Поле заголовка «Return-Path:содержит пару угловых скобок, в которые заключено необязательное значение addr-spec. Поле «Received:» содержит (возможно пустой) список маркеров, за которым следует точка с запятой (;) и значение даты и времени. Каждый маркер должен представлять собой элемент типа word, angle-addr, addr-spec или domain. Спецификации, описывающие использование полей трассировки (такие, как [RFC5321]), могут вносить дополнительные ограничения.

trace = [return]

1\*received

return = "Return-Path:" path CRLF

path = angle-addr / ([CFWS] "<" [CFWS] ">" [CFWS])

received = "Received:" \*received-token ";" date-time CRLF

received-token = word / angle-addr / addr-spec / domain

Полное описание использования полей трассировки в почте Internet содержится в [RFC5321]. В данной спецификации трассировочные поля рассматриваются исключительно в качестве информационных и любая формальная интерпретация этих полей выходит за рамки документа.

**3.6.8. Дополнительные поля**

В сообщениях могут появляться поля, не рассмотренные в этом документе. Такие поля должны соответствовать синтаксису optional-field. Этот синтаксис включает имя поля, состоящее из печатаемых символов US-ASCII без двоеточий и пробелов (SP), за которым следует двоеточие и произвольный текст, соответствующий синтаксису бесструктурных полей.

Недопустимо совпадение имен дополнительных полей с именами полей, указанными в данном документе.

optional-field = field-name ":" unstructured CRLF

field-name = 1\*ftext

ftext = %d33-57 / ; Печатаемые символы US-ASCII,

%d59-126 ; за исключением «:».

В настоящей спецификации дополнительные поля считаются неинтерпретируемыми.

4. Устаревший синтаксис

В ранних версиях спецификации допускался иной (обычно, более либеральный) синтаксис по сравнению с дозволенным в этой версии. Кроме того, некоторые синтаксические элементы, используемые в почтовых сообщениях Internet, никогда не были документированы. Хотя генерация таких синтаксических форм недопустима в соответствии, они должны восприниматься и разбираться соответствующим спецификации получателем. В этом разделе документированы многие элементы такого типа. Использование грамматики с добавлением содержащихся в данном разделе определений, создает грамматику для использования при интерпретации сообщений.

Важное различие между устаревшим (интерпретация) и современным (генерация) синтаксисом заключается в том, что структурированные поля заголовков (т. е., текст между двоеточием и завершающей строку последовательностью CRLF) могут содержать пробелы (включая фальцовочные) и комментарии между любыми лексемами. Это позволяет создавать множество сложных форм, разбор которых может оказаться затруднительным для некоторых реализаций.

Другим ключевым различием между устаревшим и современным синтаксисом является правило запрещающее включение фальцовочных символов в строки, содержащие только пробелы в комментарии.

также рассматриваются некоторые символы, которые раньше разрешалось использовать в сообщениях. Символ NUL (код ASCII - 0) считался разрешенным, но сейчас к таковым не относится. Аналогично, управляющие символы US-ASCII, отличные от CR, LF, SP и HTAB (коды ASCII от 1 до 8, 11, 12, 14 - 31 и 127) разрешалось использовать в полях заголовков. Символы CR и LF разрешалось использовать в сообщениях не только в форме последовательности CRLF; такое использование также рассматривается здесь.

4.1. Прочие устаревшие маркеры

Описанные здесь синтаксические элементы используются в устаревшем или основном синтаксисе. Отдельные символы CR, LF и NUL добавлены в obs-qp, obs-body и obs-unstruct. Управляющие символы US-ASCII добавлены в obs-qp, obs-unstruct, obs-ctext и obs-qtext. Символ точки (.) добавлен в obs-phrase. Поддерживается лексема obs-phrase-list для (возможно пустых) списков разделенных запятыми фраз, которые могут включать «пустые» элементы. Т. е., в таком списке могут быть две и более запятых, между которыми не содержится ничего; возможны также запятые в начале и в конце списка.

obs-NO-WS-CTL = %d1-8 / ; Управляющие символы US-ASCII,

%d11 / ; не включая символов

%d12 / ; возврата картеки,

%d14-31 / ; перевода строки и

%d127 ; пробельных символов

obs-ctext = obs-NO-WS-CTL

obs-qtext = obs-NO-WS-CTL

obs-utext = %d0 / obs-NO-WS-CTL / VCHAR

obs-qp = "\" (%d0 / obs-NO-WS-CTL / LF / CR)

obs-body = \*((\*LF \*CR \*((%d0 / text) \*LF \*CR)) / CRLF)

obs-unstruct = \*((\*LF \*CR \*(obs-utext \*LF \*CR)) / FWS)

obs-phrase = word \*(word / "." / CFWS)

obs-phrase-list = [phrase / CFWS] \*("," [phrase / CFWS])

Отдельные символы CR и LF, появляющиеся в сообщениях, могут иметь двоякий смысл. Во многих случаях одиночные символы CR или LF некорректно используются вместо CRLF для индикации завершения строк. В остальных случаях одиночные символы CR и LF просто используются в качестве управляющих символов US-ASCII в традиционном их смысле.

4.2. Устаревшие пробелы для фальцовки

В устаревшем синтаксисе можно включать любое количество фальцовочных пробелов в тех случаях, когда разрешено правило obs-FWS. Это позволяет включить в строку две фальцовки подряд и, следовательно, делает возможными строки в заголовках полей, содержащие только пробельные символы.

obs-FWS = 1\*WSP \*(CRLF 1\*WSP)

4.3. Устаревшие форматы даты и времени

Устаревший сиснтаксис для полей даты допускает использование 2 цифр для указания года, а также разрешает буквенные обозначения часовых поясов, которые использовались в ранних версиях спецификации. Этот синтаксис также позволяет включение комментариев и фальцовочных пробелов между большинством лексем.

obs-day-of-week = [CFWS] day-name [CFWS]

obs-day = [CFWS] 1\*2DIGIT [CFWS]

obs-year = [CFWS] 2\*DIGIT [CFWS]

obs-hour = [CFWS] 2DIGIT [CFWS]

obs-minute = [CFWS] 2DIGIT [CFWS]

obs-second = [CFWS] 2DIGIT [CFWS]

obs-zone = "UT" / "GMT" / ; Смещение от UT для Северной Америки

"EST" / "EDT" / ; восток: - 5/ - 4

"CST" / "CDT" / ; центр: - 6/ - 5

"MST" / "MDT" / ; горы: - 7/ - 6

"PST" / "PDT" / ; тихоокеанское побережье: - 8/ - 7

;

%d65-73 / ; Военные часовые пояса — от A

%d75-90 / ; до I и от K

%d97-105 / ; до Z,

%d107-122 ; в верхнем и нижнем регистре

При использовании 2 или трех цифр в поле года интерпретация выполняется следующим образом — если 2-значное обозначение года лежит в диапазоне от 00 до 49, к значению прибавляется 2000, т. е., год принимает значение от 2000 до 2049; если 2-значное обозначение года лежит в диапазоне от 50 до 99, к значению прибавляется 1900.

У устаревших часовых поясах идентификаторы «UT» и «GMT» указывают на «универсальное время» и «время по Гринвичу», соответственно; оба значения семантически эквивалентны «+0000».

Оставшиеся три символа часового пояса являются обозначениями часовых поясов США. Первая буква («E», «C», «M» или «P») означает «Eastern» (восточный), «Central» (центральный), «Mountain» (горный) и «Pacific» (тихоокеанский). Вторая буква может быть «S» (Standard — стандартное время) или «D» (Daylight Savings — летнее время).

Ниже приводится интерпретация используемых значений.

* EDT семантически эквивалентно -0400
* EST семантически эквивалентно -0500
* CDT семантически эквивалентно -0500
* CST семантически эквивалентно -0600
* MDT семантически эквивалентно -0600
* MST семантически эквивалентно -0700
* PDT семантически эквивалентно -0700
* PST семантически эквивалентно -0800

Один символ военного часового пояса был определен нестандартным способом в [RFC0822] и, следовательно, его значение непредсказуемо. Исходные определения военных часовых поясов от «A» до «I» эквивалентны поясам от «+0100» до «+0900», соответственно; «K», «L» и «M» эквивалентны «+1000», «+1100» и «+1200», соответственно; «N» — «Y» эквивалентны поясам от «-0100» до «-1200», соответственно; «Z» эквивалентно «+0000». Однако в результате ошибки в [RFC0822] все эти обозначения следует трактовать, как эквивалентные «-0000» если явно не задано иное их толкование.

В почтовых сообщениях Internet применяются и другие многосимвольные (обычно от 3 до 5 букв) обозначения часовых поясов. Все обозначения, смысл которых непонятен, следует трактовать, как эквивелент «-0000» если явно не указано иное их толкование.

4.4. Устаревшая адресация

В адресации имеется четыре основных различия. Во-первых, в адресе почтового ящика разрешалось использовать перед addr-spec маршрутную часть, заключенную в угловые скобки. Маршрут является просто списком разделенных запятыми доменных имен, каждое из которых имеет префикс «@»; для завершения списка используется двоеточие (:). Во-вторых, разрешалось включать символы CFWS между разделенными точками элементами локальной части и домена (т. е., лексема dot-atom не использовалась). Кроме того, в local-part можно было в дополнение к атомам включать строки в кавычках. В-третьих, разрешалось включать в списки почтовых ящиков (mailbox-list) и списки адресов (address-list) пустые (null) элементы. Т. е., в списке могли следовать две или более запятых подряд, а также разрешалось присутствие запятых в начале и в конце списков. В-четвертых, разрешалось использовать управляющие символы US-ASCII и пары с квотированием в доменных литералах.

obs-angle-addr = [CFWS] "<" obs-route addr-spec ">" [CFWS]

obs-route = obs-domain-list ":"

obs-domain-list = \*(CFWS / ",") "@" domain

\*("," [CFWS] ["@" domain])

obs-mbox-list = \*([CFWS] ",") mailbox \*("," [mailbox / CFWS])

obs-addr-list = \*([CFWS] ",") address \*("," [address / CFWS])

obs-group-list = 1\*([CFWS] ",") [CFWS]

obs-local-part = word \*("." word)

obs-domain = atom \*("." atom)

obs-dtext = obs-NO-WS-CTL / quoted-pair

При интерпретации адресов маршрутную часть следует игнорировать.

4.5. Устаревшие поля заголовков

Синтаксически, основным различием является то, что устаревший синтаксис полей разрешал многократное включение любых полей в произвольном порядке. Кроме того, устаревший синтаксис разрешает включать любое количество пробелов перед двоеточием в конце имени поля.

obs-fields = \*(obs-return /

obs-received /

obs-orig-date /

obs-from /

obs-sender /

obs-reply-to /

obs-to /

obs-cc /

obs-bcc /

obs-message-id /

obs-in-reply-to /

obs-references /

obs-subject /

obs-comments /

obs-keywords /

obs-resent-date /

obs-resent-from /

obs-resent-send /

obs-resent-rply /

obs-resent-to /

obs-resent-cc /

obs-resent-bcc /

obs-resent-mid /

obs-optional)

За исключением поля адреса получателя (см. параграф 4.5.3) интерпретация множественного вхождения полей не специфицирована. Также нет спецификации для интерпретации полей трассировки и пересылки, которые не располагаются в блоке, предшествующем сообщению. Если в следующих подпараграфах явно не указано иное, интерпретация таких полей происходит идентично интерпретации подобных полей современного синтаксиса, описанных в разделе 3.

**4.5.1. Устаревшее поле даты создания**

obs-orig-date = "Date" \*WSP ":" date-time CRLF

**4.5.2. Устаревшие поля отправителя**

obs-from = "From" \*WSP ":" mailbox-list CRLF

obs-sender = "Sender" \*WSP ":" mailbox CRLF

obs-reply-to = "Reply-To" \*WSP ":" address-list CRLF

**4.5.3. Устаревшие поля адресов получателей**

obs-to = "To" \*WSP ":" address-list CRLF

obs-cc = "Cc" \*WSP ":" address-list CRLF

obs-bcc = "Bcc" \*WSP ":"

(address-list / (\*([CFWS] ",") [CFWS])) CRLF

При наличии в сообщении множества полей с адресом получателя их следует трактовать, как объединение адресного списка из первого поля с адресными списками из последующих полей путем добавления запятых и конкатенации строк.

**4.5.4. Устаревшие поля идентификации**

Устаревшие поля «In-Reply-To:» и «References:» отличаются от современного синтаксиса тем, что в них допускается включение фраз (слова или строки в кавычках). Устаревшие формы ревой и правой сторон msg-id разрешают промежуточные CFWS, что делает эти части синтаксически эквивалентными local-part и domain, соответственно.

obs-message-id = "Message-ID" \*WSP ":" msg-id CRLF

obs-in-reply-to = "In-Reply-To" \*WSP ":" \*(phrase / msg-id) CRLF

obs-references = "References" \*WSP ":" \*(phrase / msg-id) CRLF

obs-id-left = local-part

obs-id-right = domain

При интерпретации фразы в полях «In-Reply-To:» и «References:» игнорируются.

Семантически, ни один из дополнительных символов CFWS в local-part и domain не является частью obs-id-left и obs-id-right, соответственно.

**4.5.5. Устаревшие информационные поля**

obs-subject = "Subject" \*WSP ":" unstructured CRLF

obs-comments = "Comments" \*WSP ":" unstructured CRLF

obs-keywords = "Keywords" \*WSP ":" obs-phrase-list CRLF

**4.5.6. Устаревшие поля пересылки**

В устаревшем синтаксисе имеется поле «Resent-Reply-To:», которое состоит из имени поля, необязательных комментариев и фальцовочных пробелов, двоеточия и списка разделенных запятыми адресов.

obs-resent-from = "Resent-From" \*WSP ":" mailbox-list CRLF

obs-resent-send = "Resent-Sender" \*WSP ":" mailbox CRLF

obs-resent-date = "Resent-Date" \*WSP ":" date-time CRLF

obs-resent-to = "Resent-To" \*WSP ":" address-list CRLF

obs-resent-cc = "Resent-Cc" \*WSP ":" address-list CRLF

obs-resent-bcc = "Resent-Bcc" \*WSP ":"

(address-list / (\*([CFWS] ",") [CFWS])) CRLF

obs-resent-mid = "Resent-Message-ID" \*WSP ":" msg-id CRLF

obs-resent-rply = "Resent-Reply-To" \*WSP ":" address-list CRLF

Как и остальные поля пересылки, поле «Resent-Reply-To:» трактуется исключительно, как информационное.

**4.5.7. Устаревшие поля трассировки**

Поля obs-return и obs-received приведены здесь, как шаблоны определений, идентичные return и received в разделе 3. Полный синтаксис описан в [RFC5321].

obs-return = "Return-Path" \*WSP ":" path CRLF

obs-received = "Received" \*WSP ":" \*received-token CRLF

**4.5.8. Устаревшие дополнительные поля**

obs-optional = field-name \*WSP ":" unstructured CRLF

5. Вопросы безопасности

Требуются определенные меры предосторожности при отображении сообщений на терминале или в программе эмуляции терминала. Многофункциональные терминалы могут реагировать на escape-последовательности и другие комбинации управляющих символов US-ASCII, что может вызывать неожиданные эффекты. В число таких эффектов может входить изменение клавиатурной раскладки и другие эффекты, способные приводить к нарушению работы и даже к повреждению данных. Управляющие символы могут вызывать (иногда программируемо) генерацию ответов на сообщения, что позволяет вводить команды от имени пользователя. Возможно также воздействие на подключенные к терминалу устройства (например, принтеры). Программы просмотра сообщений могут по своему усмотрению удалить опасные escape-последовательности из сообщения перед его выводом. Однако некоторые escape-последовательности могут быть нужны в сообщениях и поэтому не следует удалять escape-последовательности без разбора.

Передача отличных от текста объектов в сообщениях может приводить к возникновению дополнительных опасностей.

Многие реализации используют поле «Bcc:» (последний экземпляр), для доставки сообщений некоторым получателям без ведома других адресатов этого сообщения. Некорректная обработка полей «Bcc:» может привоодить к утечке конфиденциальной информации, что, в свою очередь, может снижать уровень безопасности за счет распространения информации о существовании определенных почтовых адресов. Например, при использовании первого метода, когда строка «Bcc:» удаляется из сообщения, скрытые получатели не имеют явного указания на то, что они были скрыты от других адресатов (за исключением того факта, что их адреса отсутствуют в заголовке сообщения). По этой причине кто-либо из скрытых получателей сообщения может отправить свой ответ всем показанным в заголовке получателям и непреднамеренно показать, что часть адресатов сообщения была скрыта. При использовании второго метода скрытые адресаты указываются в поле «Bcc:» отдельной копии сообщения. Если поле «Bcc:» содержит все скрытые адреса, получатели узнают о других скрытых адресатах. Даже при создании отдельного поля «Bcc:» для каждого скрытого адресата, реализациям следует с осторожностью относиться к обработке ответов на такие сообщения, во избежание непреднамеренного распространения информации о скрытых получателях сообщения другим адресатам.

Приложение A. Примеры сообщений

представлены различные примеры сообщений. Примеры предназначены для оказания помощи разработчикам при реализации данной спецификации, но их не следует воспринимать как нормативы. Хотя примеры в этом разделе подобраны аккуратно, при возникновении противоречий между примером и текстом предпочтение должно отдаваться спецификации, а не примеру.

Примеры сообщений отделены от текста строками «----», которые не являются частью сообщений.

Приложение A.1. Примеры адресации

В этом параграфе приведены примеры сообщений, которыми могут обмениваться два индивидуальных адресата.

**Приложение A.1.1. Сообщение от одного лица другому с простой адресацией**

Этот пример можно назвать каноническим сообщением. Письмо имеет одного автора (John Doe), одного получателя (Mary Smith), тему, дату, идентификатор сообщения и текст в теле письма.

----

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

Если у John есть секретарь Michael, который на самом деле отправляет сообщение, автором которого является по-прежнему John и ответы на письмо должны направляться секретарю, следует использовать поле sender:

----

From: John Doe <jdoe@machine.example>

Sender: Michael Jones <mjones@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

**Приложение A.1.2. Различные типы почтовых ящиков**

Это сообщение содержит множество адресов в полях получателей и использует иной формат адресов.

----

From: "Joe Q. Public" <john.q.public@example.com>

To: Mary Smith <mary@x.test>, jdoe@example.org, Who? <one@y.test>

Cc: <boss@nil.test>, "Giant; \"Big\" Box" <sysservices@example.net>

Date: Tue, 1 Jul 2003 10:52:37 +0200

Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.

----

Отметим, что отображаемые имена Joe Q. Public и Giant; "Big" Box требуется заключать в двойные кавычки, поскольку первое имя включает точку, а во втором содержится точка с запятой и символы двойных кавычек (в форме пар с квотированием). Отображаемое имя Who?, напротив, может использоваться без кавычек, поскольку в атомах допускается использование вопросительного знака. Отметим также, что в адресах jdoe@example.org и boss@nil.test не указаны связанные с ними отображаемые имена, а для адреса jdoe@example.org используется упрощенная форма без угловых скобок.

**Приложение A.1.3. Групповые адреса**

----

From: Pete <pete@silly.example>

To: A Group:Ed Jones <c@a.test>,joe@where.test,John <jdoe@one.test>;

Cc: Undisclosed recipients:;

Date: Thu, 13 Feb 1969 23:32:54 -0330

Message-ID: <testabcd.1234@silly.example>

Testing.

----

В этом примере поле «To:» включает группу «A Group», в состав которой входят 3 адреса, а в поле «Cc:» указана пустая группы получателей Undisclosed recipients (нераскрытые получатели).

Приложение A.2. Ответные сообщения

Следующая группа из трех примеров показывает поток обмена сообщениями между John и Mary. Сначала John отправляет сообщение Mary, на которое Mary отвечает и John в ответ отправляет письмо Mary.

Отметим специально поля «Message-ID:», «References:» и «In-Reply-To:» в каждом сообщении.

----

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

При отправке ответов поле Subject зачастую сохраняется с добавлением префикса «Re: », как описано в параграфе 3.6.5.

----

From: Mary Smith <mary@example.net>

To: John Doe <jdoe@machine.example>

Reply-To: "Mary Smith: Personal Account" <smith@home.example>

Subject: Re: Saying Hello

Date: Fri, 21 Nov 1997 10:01:10 -0600

Message-ID: <3456@example.net>

In-Reply-To: <1234@local.machine.example>

References: <1234@local.machine.example>

This is a reply to your hello.

----

Отметим поле «Reply-To:» в приведенном выше сообщении. Когда John отвечает на приведенное выше письмо Mary, ответ следует направлять по адресу из поля «Reply-To:», а не по адресу из поля «From:».

----

To: "Mary Smith: Personal Account" <smith@home.example>

From: John Doe <jdoe@machine.example>

Subject: Re: Saying Hello

Date: Fri, 21 Nov 1997 11:00:00 -0600

Message-ID: <abcd.1234@local.machine.test>

In-Reply-To: <3456@example.net>

References: <1234@local.machine.example> <3456@example.net>

This is a reply to your reply.

----

Приложение A.3. Пересылка сообщений

Начнем с сообщения, которое будет использоваться в качестве примера несколько раз:

----

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

Mary, получив это сообщение, хочет отправить копию Jane так, что (a) сообщение выглядит, как отправленное John; (b) если Jane ответит на это сообщение, ответ должен быть отправлен John; (c) вся исходная информация, включая дату письма, изначално посланного Mary, идентификатор сообщения и исходный адрес сохраняется. В этом случае в начало сообщения добавляются поля пересылки:

----

Resent-From: Mary Smith <mary@example.net>

Resent-To: Jane Brown <j-brown@other.example>

Resent-Date: Mon, 24 Nov 1997 14:22:01 -0800

Resent-Message-ID: <78910@example.net>

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

Если Jane, в свою очередь, решит переслать сообщение еще кому-либо, она будет добавлять свои поля пересылки перед сообщением, показанным выше (для краткости этот вариант примера опущен).

Приложение A.4. Сообщения с полями трассировки

При пересылке сообщения через транспортную систему, как описано в [RFC5321], в начало сообщения добавляются трассировочные поля. Ниже приведен пример, показывающий, как могут выглядеть эти поля. Отметим, что первая строка «Received:» оказалась слишком длинной и в нее включены фальцовочные пробелы.

----

Received: from x.y.test

by example.net

via TCP

with ESMTP

id ABC12345

for <mary@example.net>; 21 Nov 1997 10:05:43 -0600

Received: from node.example by x.y.test; 21 Nov 1997 10:01:22 -0600

From: John Doe <jdoe@node.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: Fri, 21 Nov 1997 09:55:06 -0600

Message-ID: <1234@local.node.example>

This is a message just to say hello.

So, "Hello".

----

Приложение A.5. Пробелы, комментарии и другие странности

Пробеля, включая фальцовочные, и комментарии могут вставляться между разными элементами поля. Возьмем пример из параграфа A.1.3 и добавим в него пробелы и комментарии в полях заголовка.

----

From: Pete(A nice \) chap) <pete(his account)@silly.test(his host)>

To:A Group(Some people)

:Chris Jones <c@(Chris's host.)public.example>,

joe@example.org,

John <jdoe@one.test> (my dear friend); (the end of the group)

Cc:(Empty list)(start)Hidden recipients :(nobody(that I know)) ;

Date: Thu,

13

Feb

1969

23:32

-0330 (Newfoundland Time)

Message-ID: <testabcd.1234@silly.test>

Testing.

----

Приведенный пример выглядит странновато, но является совершенно допустимым. Отметим специально (1) комментарий в поле «From:» (включая скобку в паре с квотированием); (2) отсутствие пробела после двоеточия в поле «To:», а также комментарий и фальцовочные пробелы после имени группы, специальный символ (.) в комментарии поля с адресом Chris Jones и фальцовочные пробелы перед и после «joe@example.org,»; (3) множество вложенных комментариев в поле «Cc:», а также комментарий, следующий непосредственно за двоеточием после «Cc»; (4) фальцовочный пробел (но не комментарии, за исключением комментария в конце поля даты), а также отсутствие значения секунд; (5) пробелы после (но не внутри) идентификатора в поле «Message-ID:».

Приложение A.6. Устаревшие формы

Здесь приведены примеры устаревших синтаксических элементов (которые недопустимо создавать), описанных в разделе 4 данного документа.

**Приложение A.6.1. Устаревшая адресация**

Отметим в приведенном ниже примере отсутствие кавычек вокруг Joe Q. Public, маршрут в адресе Mary Smith, две запятых в поле «To:» и пробелы рябом с точкой в адресе jdoe.

----

From: Joe Q. Public <john.q.public@example.com>

To: Mary Smith <@node.test:mary@example.net>, , jdoe@test . example

Date: Tue, 1 Jul 2003 10:52:37 +0200

Message-ID: <5678.21-Nov-1997@example.com>

Hi everyone.

----

**Приложение A.6.2. Устаревшие даты**

В приведенном ниже сообщении используется устаревший формат даты, включая символьное представление часового пояса и 2-значное обозначение года. Отметим, что отсутствие дня недели не является спецификой устаревшего синтаксиса — это поле является опциональным и в современном синтаксисе.

----

From: John Doe <jdoe@machine.example>

To: Mary Smith <mary@example.net>

Subject: Saying Hello

Date: 21 Nov 97 09:55:06 GMT

Message-ID: <1234@local.machine.example>

This is a message just to say hello.

So, "Hello".

----

**Приложение A.6.3. Устаревшие пробелы и комментарии**

Наличие пробелов и комментариев в устаревшем синтаксисе разрешается в большем числе мест, нежели позволяет современный синтаксис. Однако фальцовочные строки, содержащие только пробелы, по прежнему допускаются.

----

From : John Doe <jdoe@machine(comment). example>

To : Mary Smith

\_\_

<mary@example.net>

Subject : Saying Hello

Date : Fri, 21 Nov 1997 09(comment): 55 : 06 -0600

Message-ID : <1234 @ local(blah) .machine .example>

This is a message just to say hello.

So, "Hello".

----

Особо отметим вторую строку поля «To:», начинающуюся с двух пробельных символов. (отметим, «\_\_» представляет пробелы). Следовательно, это является рассматриваемой частью фальцовки, как описано в параграфе 4.2. Комментарии и пробелы в адресах, датах и идентификаторах сообщений являются частью устаревшего синтаксиса.

Приложение B. Отличия от ранних спецификаций

В этом приложении приведен список отличий формата сообщений Internet (IMF) от более ранних спецификаций, в частности [RFC0822], [RFC1123] и [RFC2822]. Элементы, отмеченные в списке звездочкой (\*), относятся к описанным в разделе 4 данного документа устаревшим элементам, и в настоящее время они не должны создаваться.

Ниже перечислены изменения [RFC0822] и [RFC1123], внесенные в [RFC2822] и сохраненные здесь:

1. Допускается использование точки в устаревшей форме фраз.
2. Описание ABNF исключено из документа (в настоящее время оно содержится в [RFC5234]).
3. В поле года разрешается использовать 4 и более цифр.
4. Порядок полей заголовка (и отсутствие такового) указан явно.
5. Удалено шифрованное поле заголовка.
6. Явно разрешено обозначение часового пояса «-0000» и описано его значение.
7. Не разрешается использовать фальцовочные пробелы между любыми лексемами.
8. Снято требование относительно получателей.
9. Заново определены понятия пересылки (forwarding и resending).
10. Расширенные поля заголовка больше не вызываются специфически.
11. Отменено использование символа ASCII 0 (null).\*
12. Строки продолжения фальцовки не могут содержать только пробелы.\*
13. Не разрешается произвольная вставка комментариев в даты.\*
14. Не разрешаются символьные обозначения часовых поясов.\*
15. Не разрешается двухзначное представление года.\*
16. Трехзначное представление года интерпретируется, но не должно использоваться.\*
17. Не разрешается включать маршруты в адреса.\*
18. Не разрешается использовать CFWS в локальной и доменной части адреса.\*
19. Не разрешается использование пустых элементов в списках адресов.\*
20. Не разрешается использование фальцовочных пробелов между именем поля и двоеточием.\*
21. Не разрешается использование комментариев между именем поля и двоеточием.
22. Более жестко задан синтаксис in-reply-to и references.\*
23. Не разрешается использование CFWS в msg-id.\*
24. Семантика полей resent отнесена исключительно к информационной.
25. Не разрешается использовать Resent-Reply-To.\*
26. Не разрешается повторение полей (за исключением resent и received).\*
27. Не разрешается использование символов CR и LF по-отдельности.\*
28. Задано ограничение на размер строк.
29. Разъяснено назначение поля Bcc.

Далее перечислены отличия настоящего документа от [RFC2822].

1. Исправлены найденные опечатки и приведены разъяснения.
2. Термин «стандарт» применительно к данному документу заменен на «документ» или «спецификация».
3. Разделены понятия «поле заголовка» (header field) и «раздел заголовков» (header section).
4. Удалено NO-WS-CTL из ctext, qtext, dtext и бесструктурных полей.\*
5. Исправлено обсуждение специальных символов (specials) в параграфе «Atom». Текст перенесен в параграф «3.5. Общий синтаксис сообщений».
6. Упрощен синтаксис CFWS.
7. Исправлен синтаксис бесструктурных полей.
8. Изменен синтаксис полей даты и времени в части пробелов в устаревшем синтаксисе дат.
9. Удалены пары с квотированием из доменных литералов и идентификаторов сообщений.\*
10. Разъяснены ограничения других спецификаций для синтаксиса доменных имен.
11. Упрощен синтаксис «Bcc:» и «Resent-Bcc:».
12. Разрешено включение дополнительного поля в трассировачную информацию.
13. Удалено no-fold-quote из msg-id. Разъяснены синтаксические ограничения.
14. Обобщен синтаксис «Received:» для исправления ошибок и удаления определения из данного документа.
15. Упрощено obs-qp. Исправлено и обобщено obs-utext (появляется не только в устаревшем синтаксисе). Удалены obs-text и obs-char, добавлено obs-body.
16. Исправлен устаревший синтаксис дат, чтобы разрешить больше (или меньше) комментариев и пробелов.
17. Исправлен синтаксис всех устаревших списков (obs-domain-list, obs-mbox-list, obs-addr-list, obs-phrase-list и вновь добавленный obs-group-list).
18. Исправлен синтаксис obs-reply-to.
19. Исправлены obs-bcc и obs-resent-bcc, чтобы разрешить пустые списки.
20. Удалено obs-path.
21. ***Назначение интерфейса Named Pipe***

Все функции ***Named Pipe API*** можно разбить на три группы:

- функции управления каналом (создать канал, соединить сервер с каналом, открыть канал, получить информацию об именованном канале, получить состояние канала, изменить характеристики канала);

- функции обмена данными (писать в канал, читать из канала, копировать данные канала);

- функции для работы с транзакциями.

Именованный канал представляет собой объект операционной системы Windows, позволяющий создавать между распределенными в локальной TCP/IP-сети процессами дуплексные и полудуплексные каналы, по которым может осуществляться передача данных в синхронном или асинхронном режимах.

1. ***Назначение и основные принципы работы Proxy-сервера***

**Прокси-сервер** (от англ. proxy — «представитель, уполномоченный») — служба в компьютерных сетях, позволяющая клиентам выполнять косвенные запросы к другим сетевым службам.

***Использование***

1. ***Обеспечение доступа с компьютеров локальной сети в Интернет***
2. ***Кэширование данных***
3. ***Сжатие данных***
4. ***Защита локальной сети от внешнего доступа***
5. ***Ограничение доступа из локальной сети к внешней***
6. ***Анонимизация доступа к различным ресурсам.***
7. ***Назначение интерфейса Mailslot***
8. Механизм Mailslots (почтовый ящик) является одним из IPC-механизмов операционной системы Windows, позволяющий создавать распределенные приложения архитектуры клиент-сервер в локальной сети TCP/IP.
9. Почтовый ящик представляет собой объект операционной системы, предоставляющий возможность пересылать данные в одном направлении: от клиента к серверу.
10. Почтовый ящик идентифицируется своим именем. Сервером называется процесс создающий почтовый ящик. Клиентом – процесс, который подключается к почтовому ящику и записывает в него данные.
11. Обмен данными осуществляется сообщениями и может происходить в синхронном и асинхронном режимах. Если клиент и сервер находятся на разных компьютерах, доставка сообщений не гарантируется.
12. Допускается создание нескольких ящиков с одним и тем же именем. Если пересылаемые сообщения не превышают 425 байт, то возможна передача данных одновременно нескольким почтовым ящикам.
13. В состав Mailslots API входят функции создания почтового ящика, подсоединения клиента к почтовому ящику, функции записи и чтения сообщений, а также функции для получения и установки характеристик почтового ящика.
14. ***Поясните понятие «Фальцовка » и «Расфольцовка»***

Для удобства и с учетом ограничения размеров строки (998/78 символов), значение поля может быть разбито на несколько строк; это называется «фальцовкой» (folding).

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

1. ***Назначение и основные принципы работы (с указанием уровней модели ISO/OSI) межсетевого экрана***

***Межсетевой экран или сетевой экран*** — комплекс аппаратных или программных средств, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами

Основной задачей сетевого экрана является защита компьютерных сетей или отдельных узлов от несанкционированного доступа. Также сетевые экраны часто называют фильтрами, так как их основная задача — не пропускать (фильтровать) пакеты, не подходящие под критерии, определённые в конфигурации.

В зависимости от уровня, на котором происходит контроль доступа, существует разделение на сетевые экраны, работающие на:

1. **сетевом** уровне, когда фильтрация происходит на основе адресов отправителя и получателя пакетов, номеров портов транспортного уровня модели OSI и статических правил, заданных администратором;
2. **сеансовом уровне** (также известные как stateful) — отслеживающие сеансы между приложениями, не пропускающие пакеты нарушающих спецификации TCP/IP, часто используемых в злонамеренных операциях — сканировании ресурсов, взломах через неправильные реализации TCP/IP, обрыв/замедление соединений, инъекция данных.
3. **уровне приложений**, фильтрация на основании анализа данных приложения, передаваемых внутри пакета. Такие типы экранов позволяют блокировать передачу нежелательной и потенциально опасной информации на основании политик и настроек.

Некоторые решения, относимые к сетевым экранам уровня приложения, представляют собой прокси-серверы с некоторыми возможностями сетевого экрана, реализуя прозрачные прокси-серверы, со специализацией по протоколам. Возможности прокси-сервера и многопротокольная специализация делают фильтрацию значительно более гибкой, чем на классических сетевых экранах, но такие приложения имеют все недостатки прокси-серверов (например, анонимизация трафика)

1. ***Последовательное выполнение при вызове удаленной процедуры***
2. Процедура клиента вызывает клиентскую заглушку.
3. Клиентская заглушка создает сообщение и вызывает службу RPC локальной операционной системы.
4. Служба RPC пересылает сообщение службе RPC удаленной операционной системе.
5. Служба RPC удаленной операционной системы вызывает серверную заглушку и передает ей сообщение.
6. Серверная заглушка извлекает из сообщения параметры и вызывает удаленную процедуру.
7. Удаленная процедура исполняет код и возвращает параметры и значение возврата серверной заглушке.
8. Серверная заглушка формирует сообщение и вызывает службу RPC своей локальной операционной системе.
9. Служба RPC своей локальной операционной системы сервера пересылает сообщение службе RPC локальной операционной системы клиента.
10. Служба RPC локальной операционной системы клиента возвращает сообщение клиентской заглушке.
11. Заглушка извлекает результаты и возвращает их.
12. ***Какие ограничения вносит спецификация REC5322(формат сообщения Internet) на размерность сообщений***

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

Данная спецификация вносит два ограничения на число символов в строке. Строка должна содержать не более 998 символов; следует использовать строки размером не более 78, без учета CRLF.

Поля заголовков представляют собой строки, начинающиеся с имени поля, за которым следует двоеточие (":"), содержимое поля и знак завершения строки CRLF. 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.

1. ***Назначение и основные принципы работы NAT-устройства***

Существует 3 базовых концепции трансляции адресов: статическая (Static Network Address Translation), динамическая (Dynamic Address Translation), 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, доступного в пуле адресов, во время коммуникации.

*Перегруженный NAT* (NAPT, NAT Overload, PAT, маскарадинг) — форма динамического NAT, который отображает несколько незарегистрированных адресов в единственный зарегистрированный IP-адрес, используя различные порты. Известен также как PAT (Port Address Translation). При перегрузке каждый компьютер в частной сети транслируется в тот же самый адрес, но с различным номером порта.

*NAT выполняет три важных функции.*

1. Позволяет сэкономить IP-адреса (только в случае использования NAT в режиме PAT), транслируя несколько внутренних IP-адресов в один внешний публичный IP-адрес (или в несколько, но меньшим количеством, чем внутренних). По такому принципу построено большинство сетей в мире: на небольшой район домашней сети местного провайдера или на офис выделяется 1 публичный (внешний) IP-адрес, за которым работают и получают доступ интерфейсы с приватными (внутренними) IP-адресами.
2. Позволяет предотвратить или ограничить обращение снаружи ко внутренним хостам, оставляя возможность обращения изнутри наружу. При инициации соединения изнутри сети создаётся трансляция. Ответные пакеты, поступающие снаружи, соответствуют созданной трансляции и поэтому пропускаются. Если для пакетов, поступающих снаружи, соответствующей трансляции не существует (а она может быть созданной при инициации соединения или статической), они не пропускаются.
3. Позволяет скрыть определённые внутренние сервисы внутренних хостов/серверов. По сути, выполняется та же указанная выше трансляция на определённый порт, но возможно подменить внутренний порт официально зарегистрированной службы (например, 80-й порт TCP (HTTP-сервер) на внешний 54055-й). Тем самым, снаружи, на внешнем IP-адресе после трансляции адресов на сайт (или форум) для осведомлённых посетителей можно будет попасть по адресу http://example.org:54055, но на внутреннем сервере, находящемся за NAT, он будет работать на обычном 80-м порту. Повышение безопасности и скрытие «непубличных» ресурсов.
4. ***Опишите процесс получения элементов IP-адресов DHCP-сервера с указанием протоколов, адресов и получателей***

DHCP. Протокол, предназначенный для автоматизации конфигурирования стека TCP/IP на узлах. С его помощью можно назначить узлам IP адрес, маску подсети, адрес шлюза по умолчанию, сконфигурировать многие другие настраиваемые параметры стека TCP/IP, сделав соответствующие настройки на сервере, а не на каждом узле.

DHCP использует принцип «аренды» или определенного времени, в течение которого IP-адрес будет действителен для компьютера. Это время может различаться, в зависимости от того, как долго пользователю требуется сетевое подключение в определенной точке.

Протокол DHCP предоставляет три способа распределения IP-адресов:

1. Ручное распределение. При этом способе сетевой администратор сопоставляет аппаратному адресу (для Ethernet сетей это MAC-адрес) каждого клиентского компьютера определённый IP-адрес. Фактически, данный способ распределения адресов отличается от ручной настройки каждого компьютера лишь тем, что сведения об адресах хранятся централизованно (на сервере DHCP), и потому их проще изменять при необходимости.
2. Автоматическое распределение. При данном способе каждому компьютеру на постоянное использование выделяется произвольный свободный IP-адрес из определённого администратором диапазона.
3. Динамическое распределение. Этот способ аналогичен автоматическому распределению, за исключением того, что адрес выдаётся компьютеру не на постоянное пользование, а на определённый срок. Это называется арендой адреса. По истечении срока аренды IP-адрес вновь считается свободным, и клиент обязан запросить новый (он, впрочем, может оказаться тем же самым). Кроме того, клиент сам может отказаться от полученного адреса.

Некоторые реализации службы DHCP способны автоматически обновлять записи DNS, соответствующие клиентским компьютерам, при выделении им новых адресов. Это производится при помощи протокола обновления DNS, описанного в RFC 2136.

1. ***Назначение записей «А» и «Мх» в таблице DNS***

|  |  |  |
| --- | --- | --- |
| **Тип ресурсной записи** | **Функция записи** | **Описание испоьзования** |
| A | Host Address  Адрес хоста, или узла | Отображает имя узла на IP-адрес  (например, для домена *microsoft.com* узлу с именем *www.microsoft.com* сопоставляется IP-адрес с помощью такой записи:  www A 207.46.199.60) |
| CNAME | Canonical Name (alias)  Какноническое имя (псевдоним) | Отображает одно имя на другое |
| MX | Mail Exchanger  Обмен почтой | Управляет маршрутизация почтовых сообщений для протокола SMTP |
| NS | Name Server  Сервер имен | Указывает на серверы DNS, ответственные за конкретный домен и его поддомены |
| PTR | Pointer  Указатель | Используется для обратного разрешения IP-адресов в имена узлов в домене *in-addr.arpa* |
| SOA | Start of Authority  Начальная запись зоны | Используется для указания основного сервера для данной зоны и описания свойств зоны |
| SRV | Service Locator  Указатель на службу | Используется для поиска серверов, на которых функционируют определенные службы (например, контроллеры доменов Active Directory или серверы глобального каталога) |

1. ***Назначение и основные принципы работы ремейтеров***

**Ремейлер** (англ. remailer) — это сервер, получающий сообщение электронной почты и переправляющий его по адресу, указанному отправителем. В процессе переадресации вся информация об отправителе уничтожается, поэтому конечный получатель лишён возможности выяснить, кто является автором сообщения

Ремейлеры делятся на анонимные и псевдо-анонимные.

При использовании псевдо-анонимного ремейлера, его оператор знает адрес электронной почты, который необходим для получения ответа на письмо. Тайна связи полностью зависит от оператора, который может стать жертвой угроз, шантажа или социальной инженерии. Преимуществом псевдо-анонимных ремейлеров является их юзабилити, за которое пользователь расплачивается меньшей защищённостью.

Анонимные ремейлеры обеспечивают гораздо более высокую секретность, но при этом они и сложнее в использовании. Их операторы не могут знать, какие данные пересылаются через них, а поэтому нет гарантии своевременной доставки сообщения, которое может и вовсе затеряться. В обмен на высокое время ожидания анонимные ремейлеры достаточно надёжно скрывают от посторонних глаз реальный адрес и содержимое сообщения.

1. ***Поясните понятие «Маршалинг параметров»***

Маршалинг – процесс передачи (копирования) параметров из адресного пространства клиента в адресное пространство компонента (ЕХЕ-модуля).

Маршалинг (marshaling) — это упаковка всех параметров и возвращаемых значений и передача их через границы процессов. СОМ поддерживает так называемый стандартный маршалинг, который надежно работает для большинства объектов, значительно снижая требования к программной реализации и делая процесс мар-шалинга полностью прозрачным.

Маршалинг (от англ. marshal — упорядочивать), по смыслу похож на сериализацию, в информатике — процесс преобразования представления объекта в памяти в формат данных, пригодный для хранения или передачи. Обычно применяется, когда данные необходимо передавать между различными частями одной программы или от одной программы к другой.

Маршалинг задействуется при использовании различных механизмов RPC, где есть необходимость в передаче данных между процессами и потоками. Переход от неуправляемого типа в тип CLR, как, например, в процессах P/Invoke, используемых в платформе .NET Framework, является подходящим примером, демонстрирующим применение маршалинга.

Кроме того, маршалинг широко используется в скриптах и приложениях, применяющих технологии XPCOM, поставляемой в составе Mozilla application framework. Браузер Mozilla Firefox — одно из самых известных приложений, созданных с применением этой платформы, что позволило скриптовым языкам использовать технологию XPCOM через Cross-Platform Connect (XPConnect).