Компьютерные сети: основные понятия

Теория построения сетей

Прежде чем перейти к подробному описанию организации компьютерных сетей, следует рассмотреть некоторые общие положения. Выясним, какие задачи необходимо решить человеку, перед которым встала задача организации сети передачи данных между компьютерами.

1. Определение типа носителя данных, то есть среды передачи данных. Передача может вестись по медным проводам различных типов, по оптоволокну, по радиоволнам, или, например, при помощи голубиной почты --- как предлагает один юмористический стандарт. Чем больше размер сети, тем больше вероятность того, что в ней используется различные типы носителей данных.

Предположим для определенности, что данные в сети передаются по кабелю с четырьмя проводам. Тогда необходимо также выяснить, как выглядит этот носитель, какие его физические характеристики, какие провода используются для передачи, а какие --- для приема данных, и как выглядит представление двоичных данных в носителе. Кроме того, нужно отличать факт передачи от отсутствия передачи: допустим, наличие напряжения --- есть данные (+5 вольт --- единица, -5 вольт --- ноль), отсутствие напряжения --- нет данных.

Но до тех пор, пока носитель не подключен к компьютеру, мы все равно не можем организовать передачу данных. Поэтому следующая проблема --- организация интерфейса.

2. Интерфейс между компьютером и средой передачи данных обеспечивается сетевой картой, часто интегрированной в материнскую плату компьютера. После создания адаптера, преобразующего электрический сигнал в среде передач данных в передаваемые на системную шину данные и наоборот, уже можно обмениваться данными между двумя компьютерами. Но изначально планировалось обеспечить доступ к среде передачи данных для множества машин, а не только двух. Для решения этой проблемы надо разработать дисциплину доступа нескольких компьютеров к общей среде передачи данных. Даже если к среде подключены только два компьютера, то всё равно надо договориться, например, о том, как выглядит отсутствие данных (как абонент поймет, что сейчас передача не идет), и как определяется начало передачи. Например, для начала передачи можно использовать некие последовательности из нулей и единиц. Когда же компьютеров несколько, то надо точно указывать адресата передаваемых данных. Также дисциплина должна обеспечивать одновременную (или псевдо-одновременную) передачу данных от нескольких компьютеров друг другу.

Однако и после решения этих проблем задача объединения большого количества компьютеров в одну вычислительную сеть еще не решена. Мы решили задачу объединения лишь небольшого числа компьютеров в рамках одной среды, внутри которой каждый компьютер может непосредственно связаться с каждым, или, другими словами, каждый виден другому. Как только появится несколько таких сред передачи данных, то мы вернемся к первой из описываемых нами проблем, а именно: придется снова налаживать передачу между абонентами этих сред. Для этого необходимо всех подключенных абонентов проидентифицировать.

3. Класс задач уровня организации сети содержит две задачи.

- Идентификация всех компьютеров сети, например, присвоение им уникального номера.
- Объединение разных сред передачи данных. Когда сетей много, в каждой все

подключенные устройства пронумерованы, и возникает задача передача информации от компьютера одной сети компьютеру другой, то, очевидно, что данные будут переброшены через несколько сред передачи. В частности, на пути будут несколько устройств, которые и занимаются передачей из одной среды в другую. Такие устройства в ряде случаев называются маршрутизаторами, а данная задача называется задачей маршрутизации.

Фактически, после решения указанных проблем возможен обмен информацией в больших сетях и между ними, но целостность передаваемой информации и безопасность обмена все еще не обеспечена. У нас есть способ связаться между любыми двумя машинами, мы это делаем, но мы не проверяем, насколько качественно происходит передача. Можно сказать, что речь идет о работе с каналом передачи данных. То есть фактически мы забываем, что у нас есть какой-то маршрут и компьютеры перекидывают друг другу массивы данных, и представляем себе канал, открытый между отправителем и получателем. Соответственно, следующей проблемой является обеспечение гарантии доставки.

4. Контроль за доставкой информации. Способов нарушить сохранность данных немного: данные можно либо изменить при передаче, т.е. испортить, либо вообще потерять, т.е. адресат их не получит. Есть и такой изощренный способ, когда вы отправляете один массив данных, а приходит два. Такое случается, когда какое-либо устройство послало данные, а ему вдруг пришло сообщение об ошибке и оно отправило их еще раз, а на самом деле ошибки не случилось. Потому в задачу контроля качества доставки (QoS --- *Quality of Service*) входят следующие компоненты.

- Доставка в целости и сохранности (*security & integrity*). Данные должны быть защищены от случайных изменений при передаче, а так же, желательно, от преднамеренных изменений и от просмотра третьими лицами. Сюда также входят такие компоненты, как обязательное оповещение отправителя об успешной доставке или наоборот, запрос на повторную посылку в случае неудачи.
- Отслеживание состояния канала передачи данных. Приведем пример ситуации, когда это жизненно важно: пусть отправитель подключен в сеть через очень быстрый интерфейс и извергает гигабитный поток данных, а на стороне получателя находится обычный модем. Было бы неплохо не только сообщать отправителю, его данные никуда не пришли и произошла ошибка по дороге, но еще и отслеживать параметры качества этого виртуального канала, организованного между двумя компьютерами, и регулировать скорость передачи данных соответствующим образом.
- Управление потоками данных (идентификация разных потоков и предотвращение их смешивания). Между двумя компьютерами могут передаваться несколько независимых потоков данных: например, программа-броузер загружает параллельно несколько изображений на одной странице. Эта задача решается при пакетной передаче данных. Если в потоке есть несколько элементов (пакетов), которые передаются от одного компьютера другому, то каждый пакет содержит информацию о том, к какому потоку он сам относится.

5. Интерпретация полученных при передаче данных является последней задачей. Обмен данными не является самоцелью, а принятые данные обычно имеют свою собственную семантику: это может быть электронное письмо или веб-страница. За эту задачу отвечает соответствующая прикладная программа.

Введение в TCP/IP

Схема TCP/IP

Общепринятым стандартом на организацию сетей передачи данных в настоящий момент является набор протоколов, называемых "схемой TCP/IP". IP и TCP --- названия двух ключевых протоколов схемы. Сетевой протокол определяется:

- своим назначением и предоставляемым возможностями;
- типами сообщений и правилами обмена ими;
- используемым нижестоящим протоколом или протоколами.

Перечисленные выше пять задач соответствуют четырехуровневой схеме TCP/IP. Уровень физического носителя иногда так же относят к интерфейсному уровню TCP/IP.

Решаемые задачи	**Уровни схемы TCP/IP**
Спецификация носителя данных	Не охватывается схемой TCP/IP
Интерфейс к носителю	Интерфейсный (канальный) уровень (*Link Layer*)
Организация межсетевого взаимодействия	Сетевой уровень (*IP Layer* или *Internet Layer*)
Котроль доставки данных	Транспортный уровень (*Transport Layer*)
Интерпретация данных	Прикладной уровень (*Application layer*)

Схему TCP/IP иногда также называют стеком TCP/IP или моделью TCP/IP.

Независимость уровней схемы TCP/IP

Одна из важных характеристик схемы TCP/IP --- высокая независимость уровней друг от друга.

- Если обеспечен интерфейс между компьютером и средой передачи данных (например, установлена сетевая карта Gigabit-Ethernet), то при решении задачи маршрутизации пакетов на сетевом уровне можно не беспокоиться о свойствах кабеля или радиосигнала.
- Транспортный уровень, организуя надёжный канал передачи данных, может не заботиться о том, по каким маршрутам передаются отдельные пакеты на сетевом уровне.
- Когда программа прикладного уровня, например, веб-браузер, получает некоторую страничку, она может не беспокоиться, сколько раз на транспортном уровне пришлось запрашивать повторные пакеты для обеспечения надёжной доставки.

Понятия пакета

Необходимо отметить, что существуют две фундаментально разных дисциплины передачи данных.

- **Сети с коммутацией каналов**. В качестве примера можно рассмотреть аналоговую телефонную сеть. Два абонента арендуют канал для звонка. Этот канал в некотором смысле виртуальный --- он состоит из цепочки физических каналов, арендованных на

каждом участке сети, однако абоненты этого не замечают. Во время звонка каждый из абонентов занят и не может осуществить другой звонок. Достоинством подобных сетей является то, что на время соединения качество связи более или менее гарантировано. Основной их недостаток --- возможные перегрузки сети, при которых из-за нехватки физических каналов на том или ином участке сети, не обязательно ближайшем к абоненту, абонент может быть "занят", хотя сам и не совершает звонка.

- **Сети с коммутацией пакетов** в настоящий момент. Единицей передачи данных является пакет. Один абонент может отправить другому абоненту пакет в любой момент. Время между отправками отдельных пакетов определяется дисциплиной передачи пакетов, например, нужно ли дождаться ответного пакета от получателя или серию пакетов можно послать без подтверждения. Возможна ситуация, когда качество существующего соединения между абонентами ухудшится, например из-за перегрузки пакетами на каком-то из промежуточных узлов.

Уже на канальном уровне в свое время было принято решение использовать сеть с коммутацией пакетов. Понятие пакета встречается на каждом уровне. Для физического уровня это может быть кадр Ethernet, он же Ethernet-фрейм (от англ. *frame*), для сетевого --- IP-пакет, для траспортного --- TCP-пакет, для прикладного --- например, DNS-пакет.

Уровни схемы TCP/IP	Название пакета
Канальный	кадр Ethernet (для сети Ethernet)
Сетевой	IP-пакет
Транспортный	TCP-пакет и датаграмма UDP

Для сложных протоколов верхнего уровня вместо понятия пакета обычно используются понятия сообщения, запроса и ответа (например, HTTP-запрос и HTTP-ответ) или команды (например, команда FTP).

Понятие адреса

На каждом из уровней схемы TCP/IP используются своя система адресов отправителя и получателя.

Уровни схемы TCP/IP	**Тип адреса**	**Пример**
Канальный	MAC-адрес сетевой карты	00:16:e6:4a:3b:60
Сетевой	IP-адрес компьютера	192.168.0.1
Транспортный	IP-адрес компьютера + TCP- или UDP-порт сетевой службы	192.168.0.1:80

На прикладном уровне используются так же символьные DNS-адреса (например, edu.ru) и адреса ресурсов (например, http://edu.ru/new.php).

Инкапсуляция протоколов TCP/IP

Пакеты протоколов более высокого уровня прозрачно передаются в качестве данных в пакетах протокола более низкого уровня. Именно поэтому используется понятие "стек

TCP/IP".

Например, пусть у нас есть HTTP-клиент (веб-броузер) и HTTP-сервер, протокол HTTP относится к прикладному уровню. Допустим, сервер приготовил HTTP-специфический массив данных --- веб-страницу, и собирается его передать, воспользовавшись услугами транспортного уровня. Для передачи по протоколу TCP этот файл должен быть разбит на фрагменты определенного размера. Каждый такой фрагмент инкапсулируется в TCP-пакет и снабжается дополнительной служебной информацией, характерной для протокола TCP. На этом дело не останавливается. Для передачи каждого TCP-пакета средствами IP он должен быть, в свою очередь разбит на фрагменты определенного размера, характерного для протокола IP. Каждый из этих фрагментов инкапсулируется в IP-пакет и снабжается своей служебной информацией, на этот раз характерной для протокола IP. Допустим, по правилам маршрутизации выяснилось, что нам нужно передать этот пакет нашему ближайшему соседу по сети. Операция повторяется снова --- IP-пакет разбивается и инкапсулируется в несколько пакетов физического уровня, например, ethernet-фреймов.

После передачи фреймов по физическому каналу происходит обратное преобразование: протокол физического уровня Ethernet составляет из нескольких своих фреймов некоторый набор данных, который передает на уровень выше. Протокол IP в этом наборе данных узнаёт IP-пакет и интерпретирует служебную информацию, в частности IP-адрес. Предположим, по правилам маршрутизации IP-пакет надо отправить следующему соседу. Тогда пакет снова передаётся на физический уровень, где инкапсулируется в несколько фреймах, и т.д. Если IP-протокол решит, что именно эта машина является получателем, то он из нескольких IP-пакетов соберёт некоторый набор данных, и передает его на уровень TCP, который узнает в этом наборе TCP-пакет и обработает его подобающим образом.

В некоторых случаях, в частности, когда это связано с ретрансляцией через сетевые экраны, решение о судьбе пакета может приниматься на TCP-уровне.

Инкапсуляция протколов приводит к накладным расходам: фактически по сети передаётся существенно больше данных, чем было запланировано прикладном уровне.

Сетевые сервисы

На прикладном уровне стека TCP/IP существует множество разнообразных протоколов, например протокол обмена почтой SMTP или протокол обмена файлами FTP. Общая схема работы сетевой программы-сервера, реализующего один из этих протоколов, следующая: она принимает данные, передаваемые программой-клиентом на определенный порт TCP или UDP, обрабатывает их, и отправляет ответ клиенту. Запросы и ответы протокола прикладного уровня часто являются командами в виде обычного текста.

Операционная система предоставляет специальный инструмент для создания серверов такого типа. Этот инструмент --- механизм сокетов. Сокеты предоставляют интерфейс синхронного и асинхронного обмена данными. В случае синхронного, или блокирующего, обмена в случае отсутствия данных для чтения при чтении из сокета программа будет ждать, пока эти данные появятся; в случае асинхронного, или неблокирующего, взаимодействия, программа продолжит работу. Сокеты в UNIX-системах могут иметь различное физическое представление. Кроме сетевых сокетов, которые могут передавать данные с одного компьютера на другой, существуют и локальные сокеты Unix, находящиеся в файловой системе. Интерфейс работы с обоими видами сокетов одинаков, за исключением создания сокета. Сетевые сокеты являются низкоуровневым программным интерфейсом к транспортному уровню сетевого взаимодействия в лице протокола TCP или UDP.

Работающие на машине с ПСПО программы, передающие данные по сети, используют для доступа к сети именно механизм сокетов. Однако, существует способ создания простых сетевых сервисов, не требующий изучения программирования сокетов. Он напоминает объединение команд в конвейер, с тем отличием, что ввод-вывод должен производиться асинхронно. Стандартные же программы окружения Unix являются "фильтрами", которые преобразовывают ввод и выдают результат на вывод синхронно. Для решения это проблемы существует сетевой супер-сервер `xinetd`, а ранее эту же задачу решал сервер `inetd`. Сам он не реализует сетевого сервиса, но дает возможность сделать из программы, которая работает с синхронным вводом-выводом, сетевой сервис, который принимает данные по сети и передает их обратно в сеть.

Попробуем создать с помощью `xinetd` сетевую службу, которая будет выдавать текущую дату. Для этого создадим файл с полным именем `/etc/xinetd.d/test` и со следующим содержанием:

```
service test
{
        disable         = no
        type            = UNLISTED
        socket_type     = stream
        protocol        = tcp
        port            = 10000
        wait            = no
        user            = nobody
        server          = /bin/date
}
```

Эта запись означает, что создаваемый сервер будет принимать TCP-соединения на порт 10000 и передавать принятую информацию команде `/bin/date`, которая игнорирует ввод и выдает текущую дату. Запускаться эта команда будет от имени пользователя `nobody`. Теперь перезапустим сервер `xinetd`:

```
# service xinetd restart
Service xinetd is not running.   [PASSED]
Staring xinetd service:          [ DONE ]
```

Проверим работоспособность сервиса, подключившись к нему командой `netcat localhost 10000`:

```
$ netcat localhost 10000
Fri Jul  4 04:31:17 MSD 2008
```

А если мы укажем в поле `server` команду `/bin/cat`, то наш сервер будет отправлять полученные данные обратно отправителю (эхо-сервер).

В системе ALT Linux и ПСПО по умолчанию все встроенные службы супер-сервера `xinetd` выключены. Кроме того, по умолчанию в файле `/etc/xinetd.conf` указано, что к службам `xinetd` можно подключаться только с локальной машины, но не с другого компьютера по сети: `only_from = 127.0.0.1`. После любых изменениях в файле `/etc/xinetd.conf` и в файлах каталога `/etc/xinetd.d` супер-сервер следует перезапустить.

Преобразование имён и IP-адресов: теория

Назначение службы доменных имен DNS

Из всех многочисленных сетевых служб, работающих на прикладном уровне и обрабатывающие сетевые запросы клиентов, существует одна без которой и всем остальным работать очень тяжело, если вообще возможно. Эта служба называется DNS (*Domain Name Service* --- служба доменных имен).

Рассмотрим, чём назначение службы имен. В прошлый раз мы говорили о том, на каких условиях, как и кем раздаются ip-адреса (ICAAN, региональные провайдеры, локальные, и т.д.). Если посмотреть на технологию маршрутизации, то можно увидеть, что структура сети с точки зрения IP-адресов строго подчиняется топологии: если несколько компьютеров объединены в единую локальную сеть, то у них будет одинаковый адрес сети, а если компьютеры находятся в разных локальных сетях,то у них будут различающиеся IP-адреса в части адреса сети. Если какая-нибудь организация сначала получила десять адресов из диапазона, а затем ей оказалось этого мало и она получила ещё сто, то, тем не менее, эти два диапазона IP-адресов вполне могут быть совершенно разными, поскольку они отражают не тот факт, какая конкретная организация ими владеет, а тот факт, какая "часть" интернета ими занята, т.е. топологию.

Это очень неудобно с социальной точки зрения, так как мы не можем без дополнительной информации ответить на вопрос о том, чья же сеть перед нами, что это за компьютеры, и так далее. Другая проблема состоит в том, что человек довольно тяжело запоминает числа. Представьте себе, если бы мы набирали в браузере вместо имен сайтов четырехбайтные IP-адреса. Это крайне неудобно. Поэтому хочется решить сразу две задачи: присвоить всем компьютерам имена, чтобы идентифицировать их ими, и так составить эти имена, чтобы отражали не топологическую структуру интернета, а административную, то есть чтобы по имени компьютера было понятно, какой организации он принадлежит. Более того, было бы естественно, если бы сама процедура выдачи этих имён должна подчиняться административной структуре интернета.

Задача именования всех компьютеров в интернете сложнее, чем задача их перенумерации. Сложнее потому, что в отличии от перенумерации административная структура никак не подчиняется физическим аспектам сети. Соответственно, пытаться создавать централизованную базу всех имён всех компьютеров в интернете, как было на заре возникновения интернета, --- это шаг в заведомо неверном направлении, так как это очень сложно, и учитывая, что интернет испытывает сбои, такая база никогда не будет актуальна --- там будут устаревшие части и не будет вновь появившихся, к тому же нагрузка на эту базу будет слишком высокой. Поэтому возникла идея устроить не только процесс именования, но и процесс раздачи имён не с помощью единого хранилища, а в виде древовидно организованного распределенного хранилища.

Необходимо подчеркнуть, что все подключения по-прежнему осуществляются по IP-адресу , а служба доменных имен --- это всего лишь надстройка для удобства, хотя и очень полезная, но в теории не необходимая. Поэтому прежде чем произойдет обмен данными между машинами, если в качестве адреса машины указанно доменное имя, то необходимо преобразовать это имя в IP-адрес и затем соединиться. Преобразование происходит сначала по файлу `/etc/hosts`, а затем запросом службы DNS, как будет рассказано далее.

История возникновения DNS

Важное добавление: соответствие имени и ip-адреса. Во времена, когда адресов в интернете

было где-то около 15, существовал файл под названием `/etc/hosts`, который содержал базу данных по всем именам этих адресов. Его формат очень простой --- IP-адрес и имена, ему соответствующие (файл это используется и сейчас, но обычно хранит только имена самого компьютера).

Идея состоит в том, что даже когда компьютеров всего несколько десятков, может так случиться, что один компьютер будет иметь несколько имен. Почему? Потому что он может выступать в нескольких ипостасях --- с одной стороны это компьютер, принадлежащий какой-либо организации, а с другой стороны, к примеру --- компьютер, участвующий в таком-то проекте. И соответственно, если мы хотим подключиться к этой организации, мы вспомним, что его имя связано с именем организации, а если мы хотим подключиться к проекту --- что его имя связанно с именем проекта. В данном случае мы видим, что имена устроены весьма примитивно. К существующей схеме пришли не от хорошей жизни. Когда компьютеров стало очень много, то:

- во-первых, оказалось, что администраторы сетей почему-то хотят сами именовать свои компьютеры;
- во-вторых --- администраторы всех компьютеров на свете почему-то не хотят скачивать постоянно /etc/hosts из центрального хранилища, предпочитая писать /etc/hosts собственноручно, и он, естественно, не совпадает у всех компьютеров в интернете.

В какой-то момент стало очевидным то, что файлом /etc/hosts дело ограничиться не может и что надо создать некую систему, которая сама преобразовывала строковое имя в числовой адрес и обратно. Но если общего файла не существует, то надо использовать некую программу с этими функциями. А тогда появилась идея: поскольку администраторы сами пишут /etc/hosts, то пусть они и дальше сами раздают имена компьютерам в своей области владения, т.е.: ты администрируешь какие-то компьютеры --- значит, раздать им имена --- тоже твоя обязанность. Таким образом, решается задача --- преобразования доменных имен в адреса и обратно.

Решение задачи состоит в том что у каждого системного администратора есть свой файл преобразования хостов и IP-адресов. Точнее не файл, а специальная таблица (ее вид зависит от используемого DNS-сервера), распределенная по вышеизложенной схеме.

Организация системы доменных имен

Мировая система доменных имен организована следующим образом: существует несколько корневых серверов, которые ни в коем случае не знают все имена всех компьютеров в сети (и, более того, не обязаны этого знать). Вместо этого они содержат информацию о так называемых зонах DNS, состоящих из одного имени. Например, существуют зоны ru, com, info, de и так далее. Во-первых, они знают о том, какие имена первого уровня существуют в принципе, поскольку их относительно немного. Недавно их было совсем мало --- в старых справочниках по интернету указывались только com, net, org, edu, gov и еще несколько зон, связанных с государствами, по двух-буквенным кодам. В какой-то момент было принято решение сильно расширить диапазон, а недавно приняли решение, что для записи имен зон будет использоваться кодировка юникод, что позволит создать имена зон на национальных алфавитах.

Итак, корневые DNS-сервера знают о всех существующих зонах, то есть о том, какие вообще бывают окончания у доменных имен. Помимо этого, они также знают адреса серверов, обладающих информацией о содержимом этих зон --- т.е. о компьютерах, имена которых кончаются на окончание зоны. Указанные сервера называются серверами имен или NS-серверами (*nameservers*). Корневые сервера выдают ответ на вопрос довольно простого свойства --- во-первых, существует ли в принципе такое имя (проверяя окончание), и, если

оно существует, то где находится сервер (точнее, серверы) имен, знающие про имена с такими окончаниями.

Если посмотреть в словаре слово домен, мы узнаем, что оно появилось очень давно, и означает оно область владений феодального вассала. Феодальная структура была устроена пирамидально и самое главное - по принципу (соблюденному и в DNS) "вассал моего вассала --- не мой вассал". Т.е. сервер отвечает за своих непосредственных подчиненных, и не отвечает за прямых подчиненных, которые не являются непосредственными.

Рассмотрим доменное имя cs.msu.ru. Сервер, который отвечает за зону ru, отвечает только за сервера имен тех хостов, которые имеют двусложные имена, оканчивающиеся на "ru" (www.ru, msu.ru и т.д). А если в имени больше составляющих, то сервер имен зоны ".ru" не обязан знать ip-адрес сервера имен хоста, хотя он может и хранить его в силу разных причин. Но он обязан знать адрес сервера имен хоста msu.ru. Дальше запрашивается сервер имен msu.ru, а он уже обязан ответить, кто такой cmc.msu.ru либо сообщить его сервер имен, и так далее. В итоге мы получим в ответ либо ip-адрес, либо сообщение о том, что такого домена нету.

По сути, мы описали иерархическую распределённую базу данных. Если мы почитаем документацию к bind (сервер имен, часто использующийся в Linux),то увидим, что это не просто база данных, ведь там можно хранить что угодно --- любые текстовые поля, любые поля, соответствующие IP-адресам. Существует также RFC, описывающий, как хранить внутри текстовых полей данные разного типа.

Производительность и надежность системы доменных имен

В описанном процессе преобразования доменного имени в IP-адрес есть одна проблема: неужели, каждый раз, когда нам понадобится, находясь в каком-то другом домене, зайти на факультетский почтовой сервер, потребуется запрашивать корневой сервер? Тогда зачем все это было придумано, ведь число запросов на корневые сервера будет по прежнему огромно? Поэтому для того, чтобы эту процедуру сделать более эффективной, есть три идеи, которые сильно облегчают нагрузку.

1. Каждая таблица с записью соответствия IP-адреса доменному имени содержит в себе информацию. о том, в течение какого времени эта таблица, во-первых, действительна, а в-вторых, может не меняться, то есть ее время жизни и задержка на изменение. Таким образом, если мы один раз выяснили, что имени cs.msu.ru соответствует некий адрес, то в течение времени задержки на изменение --- т.е. времени, в течении которого эта запись гарантированно не будет меняться --- мы можем эту запись поместить в кеш доменных имен и не обращаться за ним к серверам имен в дальнейшем. Каждая такая таблица имеет свое время жизни (около недели), и время, в течение которого содержимое считается актуальным. Это сильно снижает нагрузку, поскольку в первый раз происходит взаимодействие по всей иерархии DNS, а в следующие разы взаимодействия не происходит, пока время задержки на изменение еще не истекло. И даже когда время актуальности истечёт, то возможно, что не будет происходить все преобразование имени, а произойдет всего лишь запрос вышестоящему серверу имен, не обновились ли данные у него. И если он отвечает "нет", то записью можно пользоваться дальше.
2. Чем больше в распределенной системе узлов, тем ниже совокупная надёжность самой системы. Для борьбы с этим тоже годится использование кеширования. Сильно повышает надежность требование, чтобы NS-серверов было более одного. И желательно, чтобы у них были существенно разные IP-адреса, например в разных сетях класса B. При этом для удобства администратора система устраивается так, что пользователю было абсолютно всё равно, какой из NS-серверов ему ответил. Все NS-сервера равноправны. Вопрос: а как несчастный администратор будет редактировать

файлы сразу на двух машинах, которые лежат в абсолютно разных местах интернета? Ответ: на самом деле среди этих машин есть главная, но об этом знает только администратор, и обычно он редактирует файл только на ней, а остальные просто скачивают с нее.

3. Некоторое ограничение свободы обычного пользователя. Дело вот в чём: если вы обращаетесь к какому-либо NS-серверу с запросом относительно преобразования IP-адреса, никакого отношения к домену, за который отвечает этот сервер, не имеющего, то он имеет право ответить "не знаю". Такой запрос (на который можно получить ответ "не знаю") называется нерекурсивным. Все DNS-сервера в сети отвечают на нерекурсивные запросы. Другой вариант: когда вы обращаетесь к серверу с т.н. рекурсивным запросом, тогда NS-сервер самостоятельно обращается к корневому и так далее. В итоге либо преобразование будет произведено, либо вам ответят, что нет такого имени, либо вам ответят, что время запроса превысило лимит. И в отличие от нерекурсивного запроса, рекурсивные запросы обычно разрешены только для "своих" машин, т.е. решение, удовлетворяет ли сервер рекурсивные запросы или нет, принимает системный администратор этого сервера. В результате получается так, что далеко не все компьютеры могут посылать далеко не всем NS-серверам рекурсивные запросы. Как правило существует два уровня: существует некая группа адресов, которым доверяют и они могут посылать рекурсивные запросы, а остальным нельзя.

Таким образом, конечные пользователи обычно работают с кеширующим сервером имен, расположенном в локальном сети, который, в свою очередь, обычно передает рекурсивные запросы к кеширующему серверу имен провайдера.

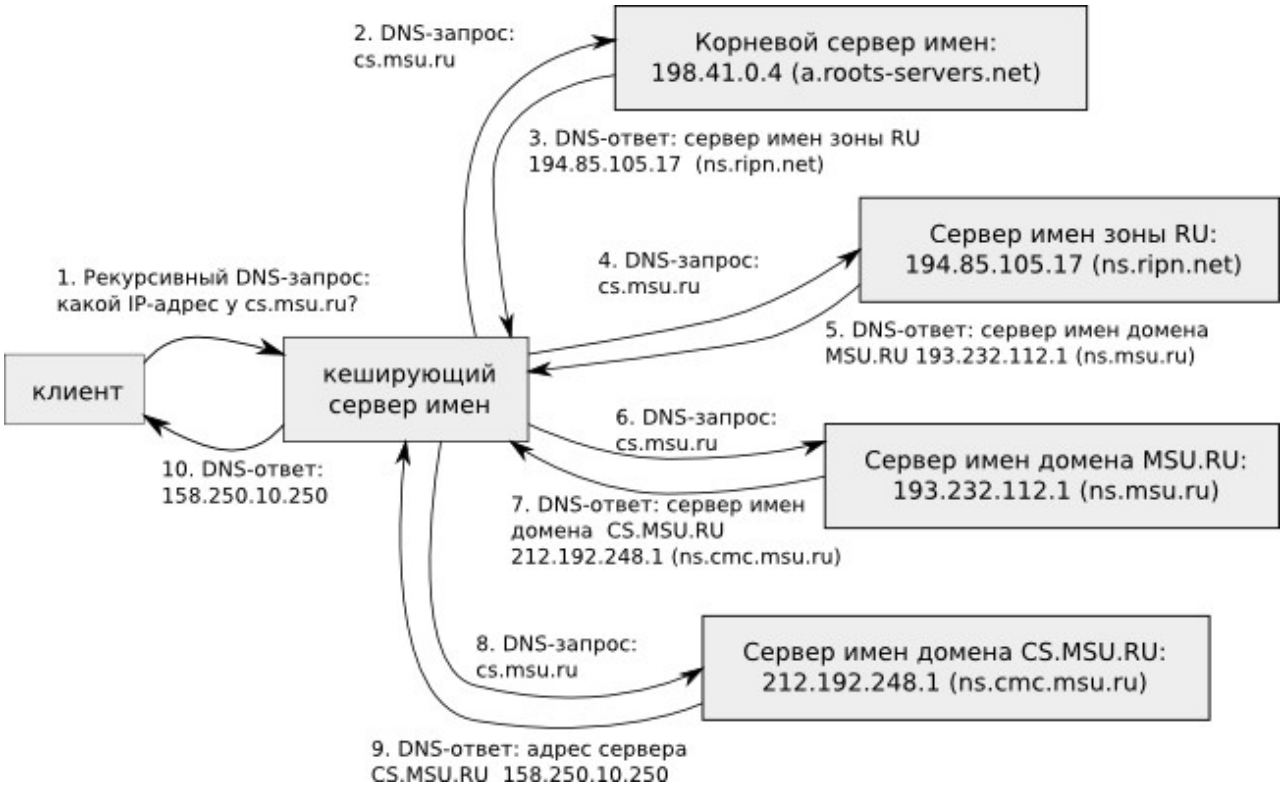

Схема преобразования DNS-имени cs.msu.ru в его ip-адрес с учетом всего сказанного выше представлена на рисунке. Предполагается, что требуемый адрес не найден ни в одном кеше, и поэтому система DNS выполняет максимально возможное число запросов. На практике каждый DNS-ответ возвращает адреса нескольких серверов имен (в данном случае --- до семи), но для простоты на рисунке указан только один.