ГОУВПО
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра программной инженерии

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ

к лабораторным работам по дисциплине

«Протоколы компьютерных сетей»

Для студентов направления подготовки 09.03.04
"ПРОГРАММНАЯ ИНЖЕНЕРИЯ"

Донецк-ДонНТУ-2016

ГОУВПО
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра программной инженерии

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ

к лабораторным работам по дисциплине

«Протоколы компьютерных сетей»

Для студентов направления подготовки 09.03.04
"ПРОГРАММНАЯ ИНЖЕНЕРИЯ"

Рассмотрено на заседании кафедры

Программной инженерии

Протокол № 1 от 30.08.2016

Утверждено на заседании

учебно-издательского Совета ДонНТУ

протокол № от

Донецк –2016

УДК 681.3

Методические указания и задания к лабораторным работам по дисциплине «Протоколы компьютерных сетей» для студентов направления подготовки 09.03.04 «Программная инженерия» / Сост.: Чернышова А.В. - Донецк, ДонНТУ, 2016 - 152 стр.

Приведены методические указания и задания к выполнению лабораторных работ по курсу «Протоколы компьютерных сетей» для студентов направления подготовки 09.03.04 «Программная инженерия». Излагаются вопросы, связанные с принципами работы протоколов, как FTP, TFTP, SMTP, POP3, IMAP, VNC, XMPP, UDP, TCP; стандарты передачи данных в беспроводных сетях, а также вопросы, связанные с разработкой клиент-серверных приложений.

Методические указания предназначены для усвоения теоретических основ и формирования практических навыков по дисциплине «Протоколы компьютерных сетей».

Составители: ст. преп. кафедры ПИ Чернышова А.В.

ЛАБОРАТОРНАЯ РАБОТА №1

Тема: Изучение протоколов FTP и TFTP

Цель: Теоретическое изучение прикладных протоколов FTP и TFTP, реализация клиента FTP и TFTP

Методические указания к выполнению лабораторной работы

В сетевой среде естественным является желание копировать файлы между компьютерными системами. Разработчики компьютеров уже создали сотни различных файловых систем, значительно или не очень существенно отличающихся друг от друга. Однако проблема связана не только с продуктами различных компаний. Иногда трудно копировать файлы между различными типами компьютеров одного и того же разработчика.

Среди проблем, с которыми обычно приходится сталкиваться при работе в многосистемном сетевом окружении, можно отметить следующие:

- Различные правила именования файлов.
- Различные правила перемещения по каталогам файловой системы.
- Ограничения на доступ к файлам.
- Различные способы представления текста и данных внутри файлов.

Разработчики стека протоколов TCP/IP старались найти не слишком сложное решение этих проблем и создали достаточно общий, но очень элегантный *протокол пересылки файлов* (File Transfer Protocol — FTP), который легко обслуживается и прост в использовании.

Протокол FTP создан для взаимодействия с интерактивным конечным пользователем или прикладной программой.

Пользовательский интерфейс разработан для клиента пересылки файлов операционной системы Berkeley Unix (BSD) и далее перенесен на различные типы многопользовательских компьютеров.

Основные функции пересылки файлов разрешают пользователю копировать файлы между системами, просматривать списки каталогов и выполнять файловые операции, подобные переименованию или удалению. Все эти функции являются частью стандартного стека протоколов TCP/IP.

Далее мы проанализируем *простейший протокол пересылки файлов* (Trivial File Transfer Protocol — TFTP), использующийся в базовых операциях по переносу файлов в определенных ситуациях, например при загрузке программного обеспечения в маршрутизаторы, мосты или бездисковые рабочие станции.

Компьютерные системы обычно требуют от пользователя идентификатор регистрации и пароль до того, как разрешить пользователю просматривать или манипулировать файлами. Однако иногда полезно создать возможность работы с общедоступными файлами. FTP обеспечивает как общедоступное совместное использование информации, так и частный доступ к файлам, предлагая два вида услуг:

- Доступ к общедоступным файлам через анонимную регистрацию.

- Доступ к личным файлам, разрешенный только для пользователя с системным идентификатором регистрации и паролем.

Протокол FTP (File Transfer Protocol).

Пример работы ftp-клиента (командная строка.

Сегодня многие имеют на своих настольных системах графические пользовательские интерфейсы (GUI) для пересылки файлов. Однако текстовый интерфейс позволяет лучше понять происходящие в процессе пересылки файлов события.

Традиционно обращение к общедоступным системам происходило через идентификатор *анонимного* (anonymous) *доступа.* В настоящее время больше применяется *ftp,* который легче напечатать. Общедоступные серверы для пересылки файла предполагают ввод пользователем адреса электронной почты в качестве пароля.

Приглашение *ftp* > выводится всякий раз, когда локальное приложение FTP ожидает ввода данных от пользователя. Строки, начинающиеся с чисел, содержат сообщения от удаленного файлового сервера. Ниже приведен пример подключения к ftp-серверу и передачей файлов.

```
> ftp ftp.internic.net
Connected to ftp.ds.internic   .net.
220-  InterNIC  Directory and
Database  Services
220-  .  .  .
220 ds.internic.net FTP server ready.
Name (ftp.internic.net:sfeit) : ftp
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
Ftp>
ftp> cd rfc

250 CWD command successful.
ftp> get rfcl842.txt nyrfc
200 PORT command successful.
150 Opening ASCII mode data
connection for rfcl842.txt
(24143 bytes).
226 Transfer complete.
local: newfile remote: rfcl842.txt
24818 bytes received in 0.53 seconds
(46 Kbytes/s)
ftp> quit 221 Goodbye.
```

Команда ftp запускает пользовательский интерфейс программы-клиента FTP. Пользователь хочет соединиться с удаленным хостом *ftp.intemic.net*. Локальный клиент FTP отчитывается об успешном соединении.

Это сообщение пришло от удаленной системы. Мы опустим приветствие.

Локальная клиентская программа FTP запрашивает ввод идентификатора пользователя. Локальная клиентская программа FTP запрашивает пароль. Пользователь переходит в удаленный каталог *rfc,* в котором и хранятся документы RFC. Команда изменения каталога (cd)

пересылается на сервер как CWD (изменить рабочий каталог). Каталог сервера изменяется на rfc, и можно начинать копирование документов RFC. Запрашивается копирование файла *rfcl842.txt,* для чего будет создано второе соединение. Локальный клиент FTP получил второй порт и послал на сервер команду PORT, указывая серверу на соединение через этот порт. Открытие соединения для пересылки файла. Завершение пересылки файла. • Создан новый локальный файл. Завершение сеанса.

Первая команда запрашивала у сервера переход в каталог *rfc.* Затем проведено копирование удаленного документа *rfcl842.txt* в локальный файл, названный *myrfc.* Если не вводить имя файла, локальный файл получит то же имя, что и удаленный файл. FTP позволяет записывать имена удаленных файлов так же, как это делают пользователи удаленного хоста. Копируя файл на локальный компьютер, можно присвоить ему локальное имя файла. Если имя не присваивается, то при необходимости FTP преобразует имя удаленного файла в формат, допустимый для локального хоста. Иногда это приводит к преобразованию, символов из нижнего регистра в верхний и к усечению имен.

Протокол FTP имеет характерный стиль операций. Всякий раз, когда должен быть скопирован файл, для пересылки данных открывается и используется второе соединение. После команды *get* (получить) в приведенном примере диалога локальный клиент FTP получает второй порт и указывает серверу на открытие соединения с этим портом:

200 PORT command successful.
150 Opening ASCII mode data connection for rfcl842.txt (24143 bytes)

Модель протокола FTP.

Как видно из приведенного выше диалога, пользователь взаимодействует с *локальным клиентом* FTP (точнее, с соответствующим процессом). Программное обеспечение локального клиента управляет преобразованием данных для *удаленного сервера* FTP через *управляющее соединение.* Когда конечный пользователь вводит команду пересылки или работы с файлом, эта команда транслируется в одно из специальных сокращений, используемых для управляющего соединения.

В сущности, управляющее соединение — это обычный сеанс *telnet* в режиме NVT. Клиент отправляет команду на сервер через управляющее соединение, а сервер возвращает ответ по этому же соединению.

Когда пользователь запрашивает пересылку файла, открывается отдельное соединение для передачи данных, и по нему пересылается файл. Это соединение используется и для пересылки содержимого каталогов. Модель FTP показана на рисунке 1.1 Обычно сервер использует порт 20 для соединения пересылки данных.

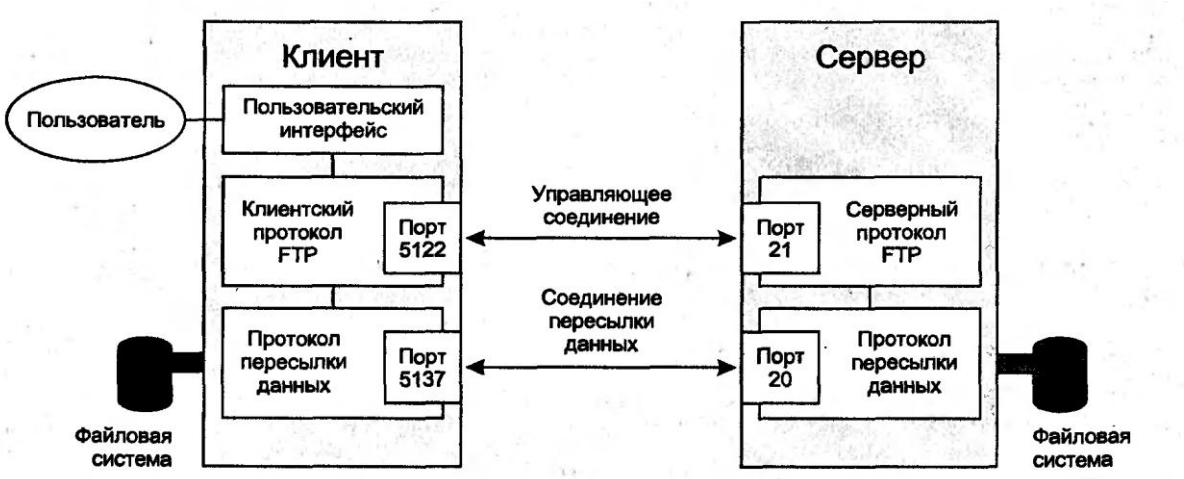


Рисунок 1.1 - Управляющее соединение и соединение пересылки данных в FTP

Во время вышерассмотренного диалога конечный пользователь вводил запросы на изменение удаленного каталога и пересылку файла. Эти запросы преобразовывались в формат команд FTP и пересылались по управляющему соединению на удаленный сервер FTP. Пересылка файлов производится по отдельному соединению, издаваемому для обмена данными.

Команды FTP.

Какие, команды можно передавать по управляющему соединению? Существуют команды аутентификации, дающие возможность пользователю указать идентификатор, пароль и регистрационную запись для работы с FTP.

Команды пересылки файлов позволяют:

- Копировать одиночный файл между хостами
- Копировать несколько файлов между хостами
- Добавлять содержимое локального файла к удаленному файлу
- Копировать файл и добавлять к его имени номер для формирования уникального имени (например, файлы ежедневной регистрации получат имена log.l, log.2 и т.д.)

Команды обслуживания файлов разрешают:

- Просмотреть список файлов каталога
- Узнать текущий каталог и изменить его на другой
- Создавать и удалять каталоги
- Переименовывать или удалять файлы

Управляющие команды служат для:

- Идентификации пересылки файлов ASCII, EBCDIC или двоичных файлов
- Проверки структурирования файла (как последовательность байт или как последовательность записей)
- Указания способа пересылки файла (например, как поток октетов)

Пересылаемые по управляющему соединению команды имеют стандартный формат. Например, команда *RETR* используется для копирования файла из сервера на сайт клиента.

Пользовательский интерфейс обычно имеет дополнительные команды для настройки локального окружения, например:

- Запросить FTP о выводе звукового сигнала при завершении пересылки файла
- Для текстового интерфейса запросить вывод символа диез (#) при пересылке каждого блока данных
- Установить автоматическое преобразование регистра символов в имени файла или таблицу трансляции символов

Использование команд в текстовом диалоге

Многие пользователи предпочитают графический интерфейс, доступный на настольных системах, но текстовый интерфейс позволяет лучше понять внутренние процессы протокола FTP.

Нижеприведенный текстовый диалог начинается с вывода справки. Существующие команды имеют синонимы, например *Is* и *dir*— для запроса сведений о каталоге, *put* и *send* — для копирования файла на удаленный хост, *get* и *recv* — для получения файла от удаленного хоста или *bye* и *quit* — для выхода из FTP.

Используя команды *mget* или *mput* и глобальные подстановочные символы можно одновременно копировать несколько файлов. Например, *mget* a* извлечет копии каждого файла с именем, начинающимся на букву a. Такой режим включается параметром *glob*, который разрешает или запрещает применение *глобальных* подстановочных символов.

В представленный ниже диалог включен вывод отладочной информации, чтобы дать некоторое представление о работе протокола:

Строки, начинающиеся на -->, показывают сообщения, посланные локальным хостом по управляющему соединению.

Строки, начинающиеся с числа, соответствуют сообщениям, посланным удаленным сервером для отчета о результате выполнения команды.

```
plum-feit > ftp
ftp> help
Commands may be abbreviated. Commands are:
```

1	cr	macdef	proxy	send
$	delete	mdelete	sendport	status
Account	debug	mdir	put	struct
Append	dir	mget	pwd	sunique
Ascii	disconnect	mkdir	quit	tenex
Bell	form	mis	quote t	trace
Binary	get	mode	recv	type
Bye	glob	raput	remotehelp	user
Case	hash	rmap	rename	verbose
Cd	help	ntrans	reset	
Cdup	led	open	rrodir	
Close	Is	prompt	runique	

```
ftp> debug
```

Debugging on (debug = 1).
ftp> **open tigger. jvnc.net:**
Connected to tigger.jvnc.net.
220 tigger.jvnc.net FTP server (Version wu-2.4(l) Fri Apr 15 13:54:36 EDT 1994)
ready.

Для обращения к личным файлам введены реальные идентификатор пользователя и пароль.

Name (tigger.jvnc.net:sfeit): **feit . .**
—> USER feit
331 Password required for feit.
Password:
—> PASS abcdl234
230 User feit logged in.

Команда *status* (статус) показывает текущие параметры сеанса FTP. Отметим, что *тип данных* (Type) указан как ASCII. При пересылке текстовых файлов FTP часто предполагает это значение по умолчанию.

ftp> **status**
Connected to tigger.jvnc.net.
No proxy connection.
Mode: stream; Type: ascii; Form: non-print; Structure: file
Verbose: on; Bell: off; Prompting: on; Globbing: on
Store unique: off; Receive unique: off
Case: off; CR stripping: on •
Ntrans: off
Nmap: off
Hash mark printing: off; Use of PORT cmds: on

Затем запрашивается список файлов каталога. Такой список может быть очень большим, поэтому FTP посылает его по соединению для данных:

ftp> **dir**

FTP необходим порт для пересылки данных. Клиент посылает команду *PORT*, которая идентифицирует его IP-адрес (4 байта) и новый порт (2 байта), чтобы использовать эти значения при пересылке данных. Байты преобразуются в десятичный формат и разделяются запятыми. IP-адрес 128.36.4.22 будет записан как 128,36,4,22, а порт 2613 — как 10,53.

—> **PORT** 128,36,4,22,10,53
200 **PORT** command successful.

Сервер откроет соединение по указанному адресу socket. Команда *LIST*— это формальное сообщение для запроса подробного списка файлов каталога:

—> LIST

Далее сервер открывает соединение с объявленным клиентом портом:

150 Opening ASCII mode data connection for /bin/ls.
total 531
-rw-r-r— 1 feit tigers 0 Oct 24 1994 .addressbook
-rw-r-r- 1 feit tigers 2808 Sep 23 1994 .article
-rw-r-r- 1 feit tigers 397 Mar 14 1993 . cshrc
-rw-r-r- 1 feit tigers 3113 Jul 31 13:29 subnets -rw-r-r- 1 feit tigers 59901 Jun 5 17:48 typescript 226 Transfer complete. 2239 bytes received in 0.31 seconds (7 Kbytes/s)

Сразу после пересылки списка файлов соединение данных будет закрыто. Затем мы можем получить файл.

ftp> get subnets

Клиент указывает новый адрес socket для переноса файла. Отметим, что на сей раз используется клиентский порт 2614 (10,54).

—> PORT 128,36,4,22,10,54
200 PORT command successful.

—> RETR subnets
150 Opening ASCII mode data connection for subnets (3113 bytes).,226 Transfer complete.

По завершении пересылки файла соединение для данных закрывается.

```
local: subnets remote: subnets
3187 bytes received in 0.27 seconds (11 Kbytes/s)
ftp> quit
—> QUIT
221 Goodbye.
plum-feit>
```

Отметим, что сценарий для соединения данных был таким:

- *Локальный клиент получил новый порт и использовал управляющее соединение, чтобы сообщить серверу FTP номер своего порта.*
- *FTP-сервер связался с новым портом данных клиента.*
- *Данные были переданы.*
- *Соединение было закрыто.*

Можно применять альтернативный сценарий. Если клиент посылает команду *PASV*, сервер возвращает номер порта и переходит к прослушиванию установки соединения данных от клиента. Ранее преобладало использование команды *PORT*. Однако теперь клиент может послать команду *PASV для* пересылки файлов через простую систему защиты (firewall), которая не разрешает установку соединений из поступающих сообщений.

При работе с файлами большого размера иногда обнаруживается, что пересылается не тот файл. Хорошая реализация должна позволять отменить пересылку. Для текстового интерфейса это обычно делается через комбинацию клавиш CONTROL-C, а в графическом интерфейсе — специальной кнопкой *Abort* (остановить).

Типы данных, структуры файлов и методы пересылки

На обоих концах соединения необходимо обеспечить единый формат для пересылаемых данных. Этот файл текстовый или двоичный? Он структурирован по записям или по блокам?

Для описания формата пересылки используются три атрибута: *тип данных* (data type), *структура файла* (file structure) и *режим пересылки* (transmission mode). Допустимые значения этих атрибутов рассмотрены ниже. В общем случае применяются:

- Пересылка текста ASCII или двоичных данных.
- Неструктурированный файл, который рассматривается как последовательность байт.
- Режим пересылки рассматривает файл как поток байт.

Однако есть и несколько исключений. Некоторые хосты структурируют текстовые файлы как последовательность записей. Хосты IBM используют для текстовых файлов кодирование EBCDIC и проводят обмен файлами как набором структурированных блоков, а не как потоком байт.

Типы данных

Файл может содержать текст ASCII, EBCDIC или двоичный образ данных (существует еще тип, называемый *локальным* или *логическим байтом* и применяемый для компьютеров с размером байта в 11 бит).

Текстовый файл может содержать обычный текст или текст, форматированный для вывода на принтер. В последнем случае в нем будут находиться коды вертикального форматирования:

- Символы вертикального форматирования *Telnet* для режима NVT (т.е. <CR>, <LF>, <NL>, <VT>, <FF>)
- Символы вертикального форматирования ASA (ФОРТРАН)

Типом данных по умолчанию является нераспечатываемый текст ASCII (т.е. текст без управляющих символов форматирования). Тип данных может быть изменен стандартной командой *TYPE,* пересылаемой по управляющему соединению.

Пересылка текста ASCII

Хотя текст ASCII является стандартным, компьютеры интерпретируют его по-разному из-за различия в кодах конца строки. Системы Unix используют для этого <LF>, компьютеры PC — <CR><LF>, а Macintosh — <CR>.

Для устранения этих различий FTP превращает локальный текстовый файл ASCII в формат NVT, а приемник преобразует NVT ASCII в собственный локальный формат. Например, если текстовый файл копируется с системы Unix на PC, все коды концов строк (в Unix — <LF>) при получении файла на PC нужно преобразовать в <CRXLF>.

Пересылка текста EBCDIC

Поддерживающие кодировку EBCDIC хосты обеспечивают весьма полезную команду пользовательского интерфейса, инициирующую пересылку по управляющему соединению команды *TYPE E.* Текстовые символы EBCDIC пересылаются по соединению в своем обычном 8-разрядном формате. Строки завершаются символом новой строки EBCDIC (<NL>).

Пересылка двоичных данных

С пересылки текстов ASCII легко переключиться на двоичный образ данных. В текстовом пользовательском интерфейсе для этого служит команда *binary,* а в графическом — командная кнопка binary (двоичные данные). Клиент меняет тип пересылаемых данных командой *TYPE I,* передаваемой по управляющему соединению.

Что произойдет, если пользователь забудет переключить тип данных с ASCII на двоичный при копировании двоичного файла? Хорошие реализации FTP предупредят, что задана ошибочная операция, и позволят до начала пересылки файла изменить тип данных. К сожалению, многие реализации идут еще дальше и "помогают" изменять все двоичные байты, которые

выглядят как символы конца строк (исправляя их на специальные заполнители или полностью удаляя их из текста). Некоторые действительно плохие реализации все же начинают пересылку файла и аварийно завершаются в середине выполнения такой операции.

Структуры файлов

В FTP поддерживаются две структуры (ранее использовалась также *страничная структура* для файлов DEC TOPS-20, сейчас устаревшая):

- *Файловая структура,* соответствующая неструктурированному файлу, который рассматривается как последовательность байт.
- *Структура записей,* которая применяется для файлов, состоящих из последовательности записей.

Более распространена *файловая структура,* которая применяется по умолчанию. Перейти на *структуру записей* можно стандартной командой *STRU R,* пересылаемой по управляющему соединению.

Режимы пересылки

Режим пересылки и структура файла определяют, как будут форматированы данные для обмена по соединению. Существуют три режима пересылки: *stream* (поток), *block* (блочный режим) и *compressed* (сжатые данные).

В режиме потока и файловой структуры файл передается как поток байт. FTP возлагает на TCP обеспечение целостности данных и не включает в данные никаких заголовков или разделителей. Единственным способом указания на конец файла будет нормальное завершение соединения для данных.

Для режима потока и структуры записей каждая запись отделяется 2-байтовым управляющим кодом конца записи (End Of Record — EOR), а конец файла отмечается символами конца файла (End Of File —EOF). EOR кодируется как X'FF 01, а EOF — X'FF 02. Для последней записи файла EOR и EOF записываются как X'FF 03. Если файл содержит байт данных из одних единиц, то такой байт представляется при пересылке как X'FF FF.

В блочном режиме файл пересылается как последовательность блоков данных. Каждый блок начинается 3-байтовым заголовком (см. рис. 14.4).

Режим сжатия данных используется крайне редко, поскольку обеспечивает очень неудачный метод архивирования, разрушающий последовательность повторяющихся байт. Обычно пользователю проще применить одну из более удачных программ сжатия, широко доступных на современных компьютерах, и далее пересылать полученный архивный файл как двоичные данные.

На рисунке 1.2 представлен формат заголовка блочного режима пересылки FTP.

8 бит 16 бит

Дескрипторные флаги	Счетчик байт
Конец блока — EOR Конец блока — EOF Restart Marker	Количество следующих далее байт

Рисунок 1.2 - Формат заголовка блочного режима пересылки FTP

Блок может содержать целую запись, или в записи объединяются несколько блоков. Дескриптор содержит:

- Флаг End Of Record для идентификации границы записи
- Флаг End Of File, который указывает, является ли блок последним при пересылке файла
- Флаг Restart Marker (маркер перезапуска), указывающий, содержит ли данный блок текстовую строку, которую можно использовать для указания точки перезапуска после неудачной пересылки файла в более поздней точке.

Режим потока наиболее распространен и используется по умолчанию. Изменить его на блочный режим можно стандартной командой *MODE B,* пересылаемой по управляющему соединению.

Преимущество структуры записей или блочного режима, состоит в том, что будет явно отмечен конец файла и после завершения его пересылки можно сохранить соединение для данных, а следовательно, использовать его для нескольких пересылок.

В показанном ранее диалоге ответ на команду *status* содержал:

Mode: stream; Type: ascii; Form: non-print; Structure: file

Т.е. по умолчанию был установлен поточный режим пересылки данных, тип данных ASCII без форматирования для печати и файловая структура (соответствующая неструктурированному файлу).

С протоколом FTP связаны следующие понятия:

- Команды и их параметры, пересылаемые по управляющему соединению
- Числовые коды, возвращенные в ответ на команду
- Формат пересылаемых данных

Ниже рассмотрен набор команд FTP. Они передаются по управляющему соединению. За последние годы набор команд существенно увеличился, однако хостам необязательно реализовывать все специфицированные команды.

Иногда локальный пользовательский интерфейс не поддерживает команды непосредственно, а оставляет их реализацию для удаленного хоста. Хорошая реализация FTP обеспечивает команду *quote* (цитата), которая позволяет вводить нужную команду в ее стандартном виде. Введенные пользователем символы далее пересылаются по управляющему соединению без каких-либо преобразований. Такой способ полезен, когда пользователю известны стандартные команды и их параметры.

Команды управления доступом

Команды и параметры, которые определяют доступ пользователя к хранилищу файлов удаленного хоста, определены в таблице 1.1.

Таблица 1.1 - Команды авторизации пользователя для доступа к архиву файлов

Команда	**Определение**	**Параметр(ы)**
USER	Идентифицирует пользователя	Идентификатор пользователя
PASS	Ввод пароля	Пароль
ACCT	Указание регистрационной записи пользователя	Идентификатор регистрационной записи
REIN	Повторная инициализация для указания состояния	Нет
QUIT	Выход	Нет
ABOR	Отмена предыдущей команды и запущенной этой командой пересылки данных	Нет

Команды управления файлами

Команды из таблицы 1.2 дают возможность выполнять типичные операции позиционирования на каталог и управления файлами удаленного хоста. Рабочим каталогом (working directory) называется текущий каталог пользователя.

Таблица 1.2 - Команды выбора каталога и управления файлами

Команда	**Определение**	**Параметр(ы)**
CWD	Перейти в другой каталог сервера	Имя каталога
CDUP	Перейти в родительский каталог	Нет
DELE	Удалить файл	Имя файла
LIST	Вывести информацию о файлах	Имя каталога, список файлов (без параметра — вывод информации о рабочем каталоге)
MKD	Создать каталог	Имя каталога
NLST	Вывести список файлов каталога	Имя каталога (для рабочего каталога может отсутствовать)
PWD	Вывести имя рабочего каталога	Нет
RMD	Удалить каталог	Имя каталога
RNFR	Указать файл, который будет переименован	Имя файла
RNTO	Переименовать файл	Имя файла
SMNT	Монтировать другую файловую систему	Идентификатор (Identifier)

Команды установки формата данных

Команды из таблицы 1.3 используются для указания формата данных, структуры файла и режима пересылки, которые будут применяться при копировании файлов.

Таблица 1.3 - Команды описания типа, структуры и режима

Команда	**Определение**	**Параметр(ы)**
TYPE STRU MODE	Указание типа данных и необязательного формата вывода на принтер Структура файла Формат пересылки	A (ASCII), E (EBCDIC), 1 (двоичный образ), N (не распечатываемые), T *{telnet),* C (ASA). F (файл) или R (записи) ё (поток), B (блок) или C (сжатие)

Команды пересылки файлов

Команды из таблицы 1.4 применяются с целью установки соединения для данных, копирования файлов и восстановления при перезапуске.

Таблица 1.4 - Команды поддержки пересылки Файлов

Команда	**Определение**	**Параметр(ы)**	
ALLO	Выделяет (резервирует) достаточное пространство для поступающих данных	Целое число байт	
APPE	Добавляет локальный файл в конец удаленного файла	Имя файла	
PASV	Запрашивает у сервера IP-адрес и порт для инициализируемого клиентом соединения пересылки данных.	Нет. Сервер возвратит и номер порта	IP-адрес
PORT REST RETR STOR STOU	Идентифицирует сетевой адрес и номер порта для инициируемого сервером соединения Устанавливает маркер перезапуска (вводится сразу за перезапускаемой командой пересылки) " Извлечение или получение файла Сохранение или помещение файла Сохранение файла с уникальным именем	IP-адрес и номер порта Значение маркера Имя файла (файлов) Имя файла (файлов) Имя файла	

Исполнительные команды

Последний набор команд (таблица 1.5) выводит конечному пользователю полезную информацию.

Таблица 1.5 - Дополнительные информационные команды

Команда	**Определение**	**Параметр(ы)**
HELP NOOP SITE SYST STAT	Вывод сведений о реализованных на сервере возможностях Запрос от сервера ответа ОК Используется для специфичных серверных подкоманд, которые не стандартизованы, но могут быть доступны на данном сервере Запрос к серверу о типе его операционной системы Запрос информации о параметрах и состоянии соединения	Нет Нет Нет Нет Нет

Команды сайта

Многие файловые серверы Unix используют программное обеспечение WU-FTP от Вашингтонского университета (Сент-Луис). Эта реализация имеет команду *SITE* для выполнения на файловом сервере различных специальных программ. Например, пользователь может сначала получить доступ по идентификатору *ftp,* а затем указать в команде *SITE* регистрационный идентификатор группы и пароль. В этом случае обеспечивается доступ к большему числу файлов, чем при анонимном доступе.

Восстановления после ошибок и перезапуск

Многим организациям необходимо пересылать очень большие файлы. Предположим, что во время пересылки такого файла произошла ошибка. Возникшие проблемы должна помочь решить служба перезапуска FTP. Она не является обязательной и, к сожалению, такую службу обеспечивали только немногие продукты TCP/IP.

В блочном режиме работы FTP и при реализации службы перезапуска пересылающая информацию сторона может передавать блоки, содержащие в нужных местах общего потока данных маркеры перезапуска. Каждый маркер представляет собой распечатываемую строку текста. Например, последовательные маркеры могли бы быть: 1, 2, 3 и т.д. Всякий раз, когда приемник получает маркер, он записывает принятые данные на энергонезависимое устройство хранения и отслеживает положение маркера в общем потоке данных.

Если информацию принимает клиент, о получении каждого маркера будет информироваться конечный пользователь (как только данные были сохранены в локальной системе). Если данные получает удаленный сервер, пользователю по управляющему соединению будет возвращено сообщение, указывающее, что данные до маркера были успешно сохранены на сервере.

При отказе системы пользователь может возобновить выполнение команды, указав значение маркера как аргумент команды. Эта операция должна быть инициирована сразу после команды, во время выполнения которой произошел крах системы.

Коды ответов

Каждой команде в диалоге соответствует ответ, состоящий из кода ответа и сообщения. Например:

```
ftp> get subnets
—>  PORT   128,36,0,22,10,54
200  PORT  command successful.
—>  RETR subnets
150  Opening ASCII mode data  connection  for  subnets   (3113 bytes).
226 Transfer  complete.
```

Коды ответов состоят из трех цифр, каждая из которых имеет определенное назначение:

- Коды от 200 до 300 указывают на успешное выполнение команды.
- Коды от 100 до 200 указывают на начало выполнения операции.,
- Коды от 300 до 400 указывают на успешное достижение промежуточной точки.
- Коды от 400 до 500 сигнализируют о временной ошибке.
- Коды от 500 свидетельствуют о постоянной ошибке (это плохие новости).

Вторая и третья цифры кодов более точно специфицируют ответ.

Безопасность

Проверка имен хоста клиента

Иногда пользователи сталкиваются с невозможностью анонимного доступа к файловому архиву. Если это происходит не часто, то обычно является следствием загруженности сервера. Однако если доступ невозможен постоянно, значит есть проблемы с именем домена.

Некоторые файловые серверы запрещают доступ клиентам, которые не перечислены в базе данных DNS. Сервер FTP может выполнять обратный поиск для всех входных IP-адресов. Если такого адреса нет в базе данных DNS — доступ блокируется. Единственным решением такой проблемы может быть обращение к администратору DNS для включения имени системы в базу данных. Некоторые серверы производят *двойную проверку* — транслируют адрес клиента в имя, а затем полученное имя опять в адрес для сравнения его с исходным адресом запроса. Благодаря этому исключаются обращения с подстановочными символами для элементов DNS.

PASV или PORT.

Организации обеспечивают безопасность своих сетей через средства защиты (firewall), применяющие к датаграммам определенный критерий фильтрации и ограничивающие входящий трафик. Часто простейшие

средства защиты разрешают пользователям локальной сети инициировать соединение, но блокируют все попытки создания соединения извне.

Исходная спецификация FTP определяет команду *PORT как* средство по умолчанию для установки соединения данных. В результате многие реализации основывают установку соединения только на этой команде. Однако команда *PORT* требует открытия соединения от внешнего файлового сервера к клиенту, что обычно блокируется средством защиты локальной сети.

К счастью, новые реализации поддерживают команду *PASV*, указывающую серверу на выделение нового порта для соединения данных с пересылкой IP-адреса и номера порта сервера в ответе клиенту. Далее клиент может самостоятельно открыть соединение с сервером.

Промежуточные прокси

Некоторые организации создают более изощренные системы безопасности. Каждый запрос реально пересылается на промежуточный прокси, реализующий систему зашиты локальной сети. Прокси становится единственной системой, которая будет видна из внешнего мира. Для работы через прокси клиент предоставляет:

- Имя или IP-адрес прокси
- Идентификатор пользователя и пароль для получения доступа к прокси
- Номер порта для доступа к прокси пользователям пересылки файлов (необязательно порт 21)
- Дополнительную информацию, зависящую от конкретной реализации данного прокси-агента

После ввода данных пользователь сможет работать с приложениями обычным образом. Промежуточные процессы не видны конечному пользователю (хотя это и зависит от типа прокси). Некоторые средства защиты требуют от локальных пользователей ввода идентификатора и пароля при доступе через средство защиты до того, как начнется реальная пересылка транзакций.

Замечания о производительности

На эффективность операций пересылки файлов влияют следующие факторы:

- Файловая система хоста и производительность его дисков
- Объем обработки по переформатированию данных
- Используемая служба TCP

Краткий отчет о пропускной способности приводится в конце каждой пересылки файла:

```
226 Transfer comlete
local: rfcl261 remote: rfcl261
4488 bytes sent in 0.037 seconds (1.2e + 02 Kbytes/s)
```

Средние значения производительности FTP и TCP можно получить при пересылке больших файлов.

Протокол TFTP (Trivial File Transfer Protocol)

Некоторым приложениям копирования файлов требуются очень простые реализации, например, для начальной загрузки программного обеспечения и конфигурационных файлов в маршрутизаторы, концентраторы или бездисковые рабочие станции.

Простейший протокол пересылки файлов (Trivial File Transfer Protocol — TFTP) используется как очень полезное средство копирования файлов между компьютерами. TFTP передает данные в датаграммах UDP (при реализации в другом стеке протоколов TFTP должен запускаться поверх службы пакетной доставки данных). Для этого не потребуется слишком сложное программное обеспечение — достаточно только IP и UDP. Особенно полезен TFTP для инициализации сетевых устройств (маршрутизаторов, мостов или концентраторов).

Характеристики TFTP:

- Пересылка блоков данных размером в 512 октетов (за исключением последнего блока)
- Указание для каждого блока простого 4-октетного заголовка
- Нумерация блоков от 1
- Поддержка пересылки двоичных и ASCII октетов
- Возможность чтения и записи удаленных файлов
- Отсутствие ограничений по аутентификации пользователей

Один из партнеров по TFTP пересылает нумерованные блоки данных одинакового размера, другой партнер подтверждает их прибытие сигналом АСК. Отправитель ожидает АСК для посланного блока до того, как пошлет следующий блок. Если за время тайм-аута не поступит АСК, выполняется повторная отправка того же самого блока. Аналогично, если к получателю не поступят данные за время тайм-аута, он отправляет еще один АСК.

Сеанс TFTP начинается запросами *Read Request* (запрос чтения) или *Write Request* (запрос записи). Клиент TFTP начинает работу после получения порта, посылая Read Request или Write Request на порт 69 сервера. Сервер должен идентифицировать различные номера портов клиентов и использовать их для последующей пересылки файлов. Он направляет свои сообщения на порт клиента. Пересылка данных производится как обмен блоками данных и сообщениями АСК.

Каждый блок (за исключением последнего) должен иметь размер в 512 октетов данных и завершаться EOF (конец файла). Если длина файла кратна 512, то заключительный блок содержит только заголовок и не имеет никаких данных. Блоки данных нумеруются от единицы. Каждый АСК содержит номер блока данных, получение которого он подтверждает.

Элементы данных протокола TFTP

В TFTP существуют пять типов элементов данных:

- Read Request (RRQ, запрос чтения)
- Write Request (WRQ, запрос записи)

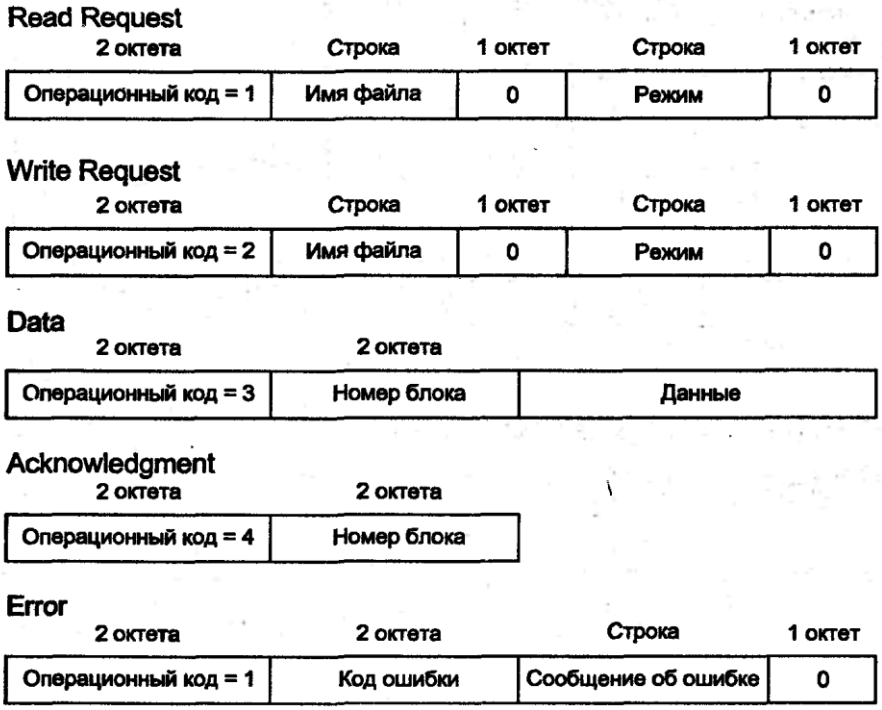


Рисунок 1.3 - Форматы элементов данных TFTP

- Data (DATA, данные)
- Acknowledgment (АСК, подтверждение)
- Error (ERROR, ошибка)

Сообщение об ошибке указывает на события, подобные таким: "файл не найден" или "для записи файла на диске нет места".

Каждый заголовок TFTP начинается операционным кодом, идентифицирующим тип элемента данных протокола (Protocol Data Unit — PDU). Форматы PDU показаны на рис. 1.3.

Отметим, что длина Read Request и Write Request меняется в зависимости от длины имени файла и полей режима, каждое из которых представляет собой текстовую строку ASCII, завершенную нулевым байтом. В поле режима могут присутствовать netascii (сетевой ASCII) или octet (октет).

Варианты TFTP

Улучшенный вариант TFTP разрешает согласование параметров через предварительные запросы чтения и записи. Его основная цель — позволить клиенту и серверу согласовывать между собой размер блока, когда он больше 512 байт (для увеличения эффективности пересылки данных).

Сценарий TFTP

Работу протокола TFTP можно проиллюстрировать простым сценарием. На рисунке 1.4 показано, как в TFTP реализуется чтение удаленного файла. После отправки запрашиваемой стороной блока данных она переходит в режим ожидания АСК на посланный блок и, только получив этот АСК, посылает следующий блок данных.

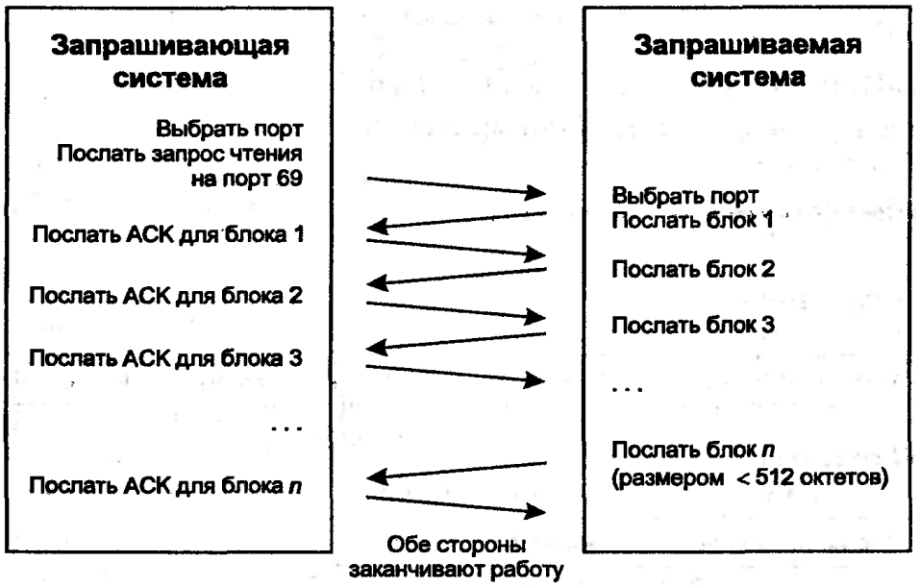


Рисунок 1.4 - Чтение удаленного файла в TFTP

Дополнительная литература

Протокол FTP определен в RFC 959, а TFTP — в RFC 1350.

Задание к лабораторной работе

Написать программу-клиент для реализации передачи файлов по протоколу ftp и клиент-серверное приложение для передачи файлов с помощью протокола tftp. Реализовать возможность анониммного подключения к серверу, просмотр текущего каталога, выбор файла и дальнейшая передача с возможностью переименования файла. Сервер для тестирования клиентской части программной реализации FTP-протокола из локальной сети университета: 194.44.183.2.

Требования к отчету по лабораторной работе №1

В отчете по лабораторной работе должны быть представлены:

1) Концепции взаимодействия программы-клиента и программы сервера для протоколов FTP и TFTP.
2) Листинг программ реализации клиентской части FTP протокола, и клиент-серверной программы реализации протокола TFTP.
3) Экранные формы, демонстрирующие работу программ.

Контрольные вопросы:

1) Какой порт использует протокол FTP?
2) Какой порт использует протокол TFTP?
3) Охарактеризуйте достоинства и недостатки протокола FTP.
4) Охарактеризуйте достоинства и недостатки протокола TFTP.
5) Что такое управляющее соединение? Какие управляющие команды Вы знаете?
6) Какие модели работы FTP протокола Вы знаете?

ЛАБОРАТОРНАЯ РАБОТА №2

Тема: Изучение протоколов передачи электронной почты.

Цель: Изучении теоретических основ работы протоколов передачи электронной почты и разработка простейшего почтового клиента (реализация протоколов SMTP, POP3, IMAP).

Методические указания к лабораторной работе

Прежде, чем рассматривать протоколы передачи почты, рассмотрим компоненты почтовой системы. На рисунке 2.1 представлены реальные компоненты почтовой системы Интернет.

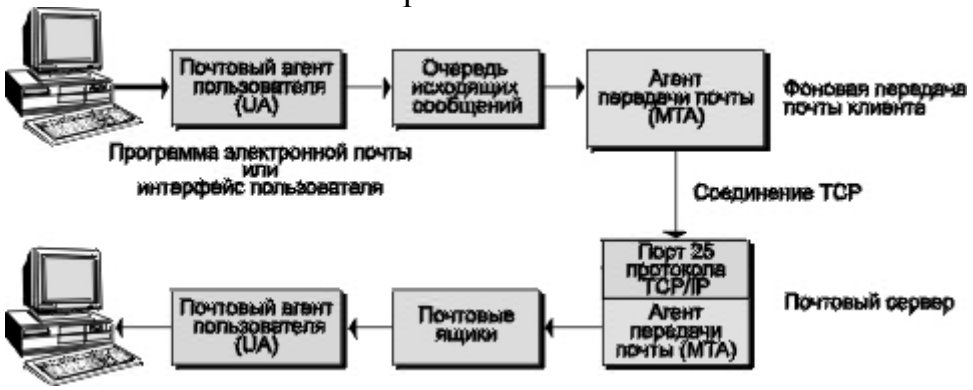


Рисунок 2.1 - компоненты почтовой системы

Термин «агент» довольно часто встречается в документации Интернет. «Агент»— это программа специального назначения, выполняющая действия для пользователя или другой программы. В большинстве случаев почтовая программа называется агентом пользователя (UA). Точно так же агент передачи почты (MTA) представляет собой клиент или сервер, выполняющий задачи по доставке или получению почты на сетевом компьютере.

Вы, как пользователь, взаимодействуете с агентом пользователя. Он, в свою очередь, взаимодействует с файлом-контейнером или агентом передачи сообщений за вас. В то же время, MTA ведет себя как представитель своего компьютера в сети. Агент пользователя защищает вас от необходимости общаться с различными почтовыми хостами, а MTA защищает компьютер от необходимости общаться с различными агентами пользователя или несколькими агентами передачи почты одновременно.

Пользовательский агент отделен от агента передачи почты. Конечно, их можно объединить в одной программе, но все равно это будут отдельные модули. Будучи взаимосвязаны, оба агента выполняют совершенно различные функции.

Система электронной почты представлена агентами передачи почты, MTA. До того как обсудить задачи пользовательского агента, необходимо узнать немного больше о том, что же такое MTA. MTA умеют устанавливать TCP-соединение для связи с другими MTA. Протоколом этого соединения, как правило, является простой протокол передачи почты (SMTP).

Простой протокол передачи почты (SMTP)

Агент передачи почты— основной компонент системы передачи почты Интернет. Как уже говорилось, MTA как бы представляет данный сетевой компьютер для сетевой системы электронной почты. Пользователи редко имеют дело с MTA, поскольку он не вполне «дружелюбен», однако без него не обходится ни одна почтовая система. После того как UA пошлет сообщение в выходную очередь, за дело принимается MTA. Он извлекает сообщение и посылает его другому MTA. Этот процесс продолжается до тех пор, пока сообщение не достигнет компьютера-получателя. Для передачи сообщений по TCP-соединению большинство MTA пользуются протоколом SMTP. Сообщения форматированы по правилам виртуального сетевого терминала (NVT), то есть в NVT ASCII. NVT подобен виртуальному сетевому протоколу и нужен затем, чтобы скрыть различия в восприятии разными компьютерами разных символов, например переводов каретки, переводов строки, маркеров конца строки, очистки экрана и т.д. Символ в NVT состоит из семи битов набора ASCII и является буквой, цифрой или знаком пунктуации. Семибитный набор ASCII часто называется NVT ASCII.

Команды SMTP

Простой протокол передачи почты обеспечивает двухсторонний обмен сообщениями между локальным клиентом и удаленным сервером MTA. MTA-клиент шлет команды MTA-серверу, а он, в свою очередь, отвечает клиенту. Другими словами, протокол SMTP требует получать ответы от приемника команд SMTP. Обмен командами и ответами на них называется почтовой транзакцией (mail transaction). Данные передаются в формате NVT ASCII. Кроме того, команды тоже передаются в формате NVT ASCII. Команды передаются в форме ключевых слов, а не специальных символов, и указывают на необходимость совершить ту или иную операцию. В табл. 2.1 приведен список ключевых слов (команд), определенный в спецификации SMTP— RFC 821.

Таблица 2.1 – Команды простого протокола передачи почты (SMTP).

Таблица 15.1. Команды простого протокола передачи почты (SMTP)

Команда	Обязательна	Описание
HELO	X	Идентифицирует модуль-передатчик для модуля-приемника (hello).
MAIL	X	Начинает почтовую транзакцию, которая завершается передачей данных в один или несколько почтовых ящиков (mail).
RCPT	X	Идентифицирует получателя почтового сообщения (recipient).
DATA		Строки, следующие за этой командой, рассматриваются получателем как данные почтового сообщения. В случае SMTP, почтовое сообщение заканчивается комбинацией символов: CRLF-точка-CRLF.
RSET		Прерывает текущую почтовую транзакцию (reset).
NOOP		Требует от получателя не предпринимать никаких действий, а только выдать ответ OK. Используется главным образом для тестирования. (No operation.)
QUIT		Требует выдать ответ OK и закрыть текущее соединение.
VRFY		Требует от приемника подтвердить, что ее аргумент является действительным именем пользователя. (См. примечание.)
SEND		Начинает почтовую транзакцию, доставляющую данные на один или несколько терминалов (а не в почтовый ящик).
SOML		Начинает транзакцию MAIL или SEND, доставляющую данные на один или несколько терминалов или в почтовые ящики.
SAML		Начинает транзакцию MAIL и SEND, доставляющие данные на один или несколько терминалов и в почтовые ящики.
EXPN		Команда SMTP-приемнику подтвердить, действительно ли аргумент является адресом почтовой рассылки и если да, вернуть адрес получателя сообщения (expand).
HELP		Команда SMTP-приемнику вернуть сообщение-справку о его командах.
TURN		Команда SMTP-приемнику либо сказать OK и поменяться ролями, то есть стать STMP-передатчиком, либо послать сообщение-отказ и остаться в роли SMTP-приемника.

В RFC 821 сказано, что команда VRFY не является обязательной для минимального набора команд SMTP. Однако в RFC 1123 «Требования для сетевых компьютеров Интернет— приложения и обеспечение работы» (Requirements for Internet Hosts— Application and Support, Braden, 1989), команда VRFY фигурирует в списке обязательных для Интернет команд реализации SMTP.

В соответствии со спецификацией команды, помеченные крестиком (X) в табл.2.1, обязаны присутствовать в любой реализации SMTP. Остальные команды SMTP могут быть реализованы дополнительно. Каждая SMTP-команда должна заканчиваться либо пробелом (если у нее есть аргумент), либо комбинацией CRLF. В описании команд употреблялось слово «данные», а не «сообщение». Этим подчеркивалось, что, кроме текста, SMTP позволяет

передавать и двоичную информацию, например графические или звуковые файлы. Другими словами, SMTP способен передавать данные любого содержания, а не только текстовые сообщения. Это значит, что, рассматривая вопросы, касающиеся SMTP, не забывайте, что термин «сообщение» обозначает не только текстовые данные.

Во многих операционных системах для системного администратора предусмотрена возможность послать сообщение многим пользователям одновременно. Например, перед остановкой системы администратор выполняет команду, которая автоматически рассылает предупреждающее сообщение на каждый подключенный терминал. С другой стороны, пользователь может отключить прием таких сообщений. Тот или иной режим приема сообщений устанавливается при входе пользователя в систему по умолчанию и может измениться позже.

Во многих компьютерах механизм рассылки таких сообщений подобен системе электронной почты. В SMTP существуют команды, позволяющие доставлять сообщения двумя путями— в почтовый ящик или на терминал. В терминах SMTP команда send означает «послать на терминал», а команда mail— «послать в почтовый ящик». Команда SEND является дополнительной, тогда как MAIL— обязательна в любой реализации.

Коды ответов SMTP

В спецификации SMTP требуется, чтобы сервер отвечал на каждую команду SMTP-клиента. MTA-сервер отвечает трехзначной комбинацией цифр, называемой кодом ответа. Вместе с кодом ответа, как правило, передается одна или несколько строк текстовой информации.

Несколько строк текста, как правило, сопровождают только команды EXPN и HELP. В спецификации SMTP, однако, ответ на любую команду может состоять из нескольких строк текста.

Каждая цифра в коде ответа имеет определенный смысл. Первая цифра означает, было ли выполнение команды успешно (2), неуспешно (5) или еще не закончилось (3). Как указано в приложении E документа RFC 821, простой SMTP-клиент может анализировать только первую цифру в ответе сервера, и на основании ее продолжать свои действия. Вторая и третья цифры кода ответа разъясняют значение первой. Если вы разрабатываете SMTP-приложение, обязательно изучите конструкцию всех кодов SMTP-ответа.

На рисунке 2.2 представлены коды ответа SMTP и их значение.

Код	Значение
211	Ответ о состоянии системы или помощь
214	Сообщение-подсказка (помощь)
220	<имя_домена> служба готова к работе
221	<имя_домена> служба закрывает канал связи
250	Запрошенное действие почтовой транзакции успешно завершилось
251	Данный адресат не является местным; сообщение будет передано по маршруту <forward-path>
354	Начинай передачу сообщения. Сообщение заканчивается комбинацией CRLF-точка-CRLF
421	<имя_домена> служба недоступна; соединение закрывается
450	Запрошенная команда почтовой транзакции не выполнена, так как почтовый ящик недоступен
451	Запрошенная команда не выполнена; произошла локальная ошибка при обработке сообщения
452	Запрошенная команда не выполнена; системе не хватило ресурсов
500	Синтаксическая ошибка в тексте команды; команда не опознана
501	Синтаксическая ошибка в аргументах или параметрах команды
502	Данная команда не реализована
503	Неверная последовательность команд
504	У данной команды не может быть аргументов
550	Запрошенная команда не выполнена, так как почтовый ящик недоступен
551	Данный адресат не является местным; попробуйте передать сообщение по маршруту <forward-path>
552	Запрошенная команда почтовой транзакции прервана; дисковое пространство, доступное системе, переполнилось
553	Запрошенная команда не выполнена; указано недопустимое имя почтового ящика
554	Транзакция не выполнена

Рисунок 2.2 - коды ответа SMTP и их значение

Ограничения по размерам

В стандарте SMTP сказано, что реализации SMTP не должны ограничивать максимальную длину обрабатываемых объектов (возможно, для будущих расширений стандарта). Однако в настоящий момент SMTP ограничивает допустимые размеры следующими величинами, приведенными в табл. 2.2.

Таблица 2.2 – Ограничении на размеры объектов SMTP

Таблица 15.3. Ограничения на размеры объектов SMTP

Объект SMTP	Ограничение
User	Максимальная длина имени пользователя: 64 символа
Domain	Максимальная длина имени домена: 64 символа
Path	Максимальная длина обратного маршрута или маршрута доставки, включая знаки пунктуации и символы-ограничители: 256 знаков
Command line	Максимальная длина командной строки, включая ключевое слово и символы CRLF: 512 знаков
Reply line	Максимальная длина строки ответа, включая код ответа и символы CRLF: 512 знаков
Text line	Максимальная длина текстовой строки, включая символы CRLF: 1000 знаков

В соответствии со спецификацией (RFC 821), если клиент MTA превысил ограничения на размер передаваемой информации, сервер MTA отвечает одним из следующих кодов:

500 Line too long.(Слишком длинная строка)

501 Path too long.(Слишком длинный путь)

552 Too many recipients.(Слишком много получателей)

552 Too much mail data.(Слишком много данных в сообщении)

Составные части сообщения электронной почты.

Сообщение электронной почты состоит из следующих частей: упаковки (envelope), для распознавания агентами передачи почты (MTA), заголовков (headers), используемых агентами пользователя (UA), и тела (body) сообщения— собственно информации, предназначенной адресату. Данные упаковки задаются полями «TO:» и «FROM:». (при рассмотрении SMTP-команды SEND и RCPT.) Другими словами, как указано в RFC 821, информация упаковки требуется для передачи сообщения от одного сетевого компьютера другому.

В RFC 822, «Стандарт оформления текстовых сообщений Интернет» (Standard for the Format of ARPA Internet Text Messages, Crocker, 1982), определены форматы двух остальных частей: заголовка и тела сообщения. (В профессиональной среде текстовые сообщения этого формата иногда называются просто «822». Например, можно встретить заявление типа: «В этом протоколе используется текст формата 822».) Для иллюстрации сложного заголовка сообщения в RFC 822 приводится следующий пример (рисунок 2.3):

Date : 27 Aug 76 0932 PDT
From : Ken Davis <KDavis@This-Host.This-net>
Subject: Re: The Syntax in the RFC
Sender : KSecy@Other-Host
Reply-To: Sam.Irving@Reg.Organization
To : George Jones <Group@Some-Reg.An-Org>,
Al.Neuman@MAD.Publisher
cc : Important folk: Tom Softwood <Balsa@Tree.Root>,

"Sam Irving"@Other-Host;,
Standard Distribution: /main/davis/people/standard@Other-Host,
"<Jones>standard.dist.3"@Tops-20-Host>;
Comment : Sam is away on business.
He asked me to handle his mail for him.
He'll be able to provide a more accurate explanation when he
returns next week.
In-Reply-To: <some.string@DBM.Group>, George's message
X-Special-action: This is a sample of user-defined field-names.
There could also be a field-name "Special-action" but, its name
might later be preempted
Message-ID: 4231.629.XYzi-What@Other-Host

Рисунок 2.3 – Пример сложного заголовка письма

Как уже отмечалось, тело сообщения содержит собственно информацию, которую необходимо доставить адресату. Как правило, оно состоит из текста формата NVT ASCII. Заголовок сообщения отделяется от тела пустой строкой.

Тело сообщения вводится пользователем в ходе почтовой транзакции при помощи агента пользователя (почтовой программы). Далее, агент пользователя добавляет необходимые поля заголовка, содержимое которых задается пользователем, и передает сообщение следующему агенту MTA, который осуществляет либо промежуточную, либо окончательную доставку.

Расширения SMTP.

Самым важным документом, посвященным расширениям SMTP, является RFC 1425, «Расширения службы SMTP» (SMTP Service Extensions, Klensin et al, 1993). Вдобавок к расширениям SMTP, в RFC 1425 описываются способы проектирования новых расширений, так сказать, на будущее. Реализация SMTP, работающая в соответствии с расширениями RFC 1425, называется «расширенный SMTP» или просто ESMTP (Extended SMTP). Расширенный SMTP работает почти так же, как и обычный SMTP, однако команда-приветствие у него другая: EHLO (Extended hello).

Чтобы выяснить, поддерживает ли MTA-сервер спецификацию ESMTP, MTA-клиент посылает команду EHLO. Если сервер поддерживает ESMTP, он отвечает кодом 250. Если нет, следует сообщение о синтаксической ошибке. В ответ на сообщение об ошибке, клиент может выдать обычную команду HELO и далее выполнять стандартные операции SMTP. Если сервер умеет обслуживать ESMTP, в ответ на приветствие, как правило, он выдает многострочный ответ (в известном вам формате). Каждая срока ответа содержит дополнительную команду ESMTP, с которой сервер знает, как работать. Вот пример реакции ESMTP-сервера в ответ на команду EHLO:

250-mail.server.com
250-EXPN
250-HELP
250 TURN

Вторая, третья и четвертая строки ответа содержат названия дополнительных команд сервера. Этот сервер обеспечивает обработку перечисленных дополнительных команд. На самом деле в качестве расширений RFC 1425 перечислены только дополнительные SMTP-команды. Другими словами, первыми шестью расширениями SMTP в RFC 1425 являются команды: EXPN, HELP, TURN, SEND, SOML и SAML. Существует также расширение SIZE, описанное в RFC 1427, «Расширения службы SMTP, касающиеся размеров сообщений» (SMTP Service Extension for Message Size Declaration, Klensin et al, 1993), позволяющее SMTP-клиенту и серверу сообщать друг другу размеры передаваемого сообщения. Если в ответе на команду EHLO присутствует ключевое слово SIZE, значит, данное расширение обрабатывается. Как известно, существует предел размеров передаваемого сообщения для сервера. Если клиент MTA попытается передать сообщение, превышающее этот предел, почтовая транзакция не состоится (закончится с ошибкой). Максимальная длина сообщения ограничивается по нескольким причинам. Основная состоит в том, что размеры жестких дисков сервера всегда ограничены, и слишком длинное сообщение может попросту не поместиться на них. С другой стороны, свободное пространство на дисках сервера может со временем увеличиться, и это сообщение будет принято при следующей попытке.

MTA-сервер ESMTP анализирует аргумент SIZE и решает, принимать ему сообщение такой длины или сообщить об ошибке. Все это происходит еще до начала передачи (однако в RFC 1427 описано несколько случаев, когда ошибка происходит уже во время передачи). Остальные несколько расширений SMTP, в общем, подчиняются тем же правилам, что и рассмотренная только что команда SIZE. То есть команда-расширение выдается в ответе сервера по запросу клиента EHLO. Расширения можно использовать в ваших программах в соответствии со спецификацией. В некоторых случаях расширение является просто дополнительной возможностью. В других случаях расширение— дополнительный аргумент к существующей команде (как в случае только что рассмотренной команды SIZE).

Местные расширения.

По определению местным расширением является любое поле, команда или название опции, начинающееся с буквы X.

```
X-Special-action: This is a sample of user-defined field-names.
There could also be a field-name "Special-action", but its name
might later be preempted
```

Имя этого поля начинается с X, поэтому оно местное, то есть задано пользователем. Разумеется, пользователь может придумать любое название поля, какое ему нравится. Однако при этом существует риск, что в дальнейших стандартах Интернет будет определено то же самое имя, но с

другим значением. При этом пользовательское и стандартное приложения вступят в конфликт. Поэтому при разработке собственных расширений почтовой системы необходимо, чтобы имена всех новых объектов начинались с буквы X. Например, пользовательский вариант декодирования тела сообщения должен называться не DECODE, как можно было бы предположить, а XDECODE. SMTP-сервер организации при этом должен включить местное расширение XDECODE в список, выдаваемый по команде EHLO (с кодом ответа 250):

250 XDECODE

MIME

В документе RFC 1521 под названием «Многоцелевые расширения почтовой системы Интернет» (MIME, Multipurpose Internet Mail Extensions, Borenstein and Bellcore, 1993) описано наиболее впечатляющее расширение для существующих почтовых систем. Система MIME не предполагает вмешательства в деятельность агентов передачи почты. Два агента пользователя, понимающие MIME, могут общаться друг с другом при помощи обыкновенных MTA. В MIME к сообщению просто добавляются несколько полей заголовка (в соответствии с RFC 822):

MIME-Version (версия MIME)
Content-Type (тип содержимого)
Content-Transfer-Encoding (тип кодировки содержимого)
Content-ID (идентификатор содержимого)
Content-Description (описание содержимого)

Номера версий MIME меняются по мере его развития. Поле MIME-Version задает номер версии расширения MIME, которое данный агент пользователя умеет обрабатывать. Номер версии в заголовке предохраняет агента от неправильной интерпретации сообщения, в случае, если версии MIME сообщения и агента не совпадают. Вот образец полей заголовка MIME-Version и Content-Type:

Mime-Version: 1.0Content-Type: TEXT/PLAIN;
charset=US-ASCII

Как видим, сообщение создано MIME версии 1.0. Тип содержимого— TEXT, подтип— PLAIN, кодовая таблица (набор символов) US-ASCII. В табл. 2.3 приведены существующие в данный момент типы и подтипы MIME.

Таблица 2.3 – Существующие типы и подтипы MIME

Таблица 15.4. Существующие типы и подтипы MIME

Тип	Подтип	Описание
Text	Plain	Неформатированный текст.
	Richtext	Текст с элементами форматирования и выделениями, например с курсивом, подчеркиваниями, жирными буквами и т. д.
	Enriched	Усовершенствованный и упрощенный вариант подтипа richtext.
Multipart	Mixed	Тело сообщения состоит из нескольких частей; обрабатывать последовательно.

	Parallel	Тело сообщения состоит из нескольких частей; обрабатывать параллельно.
	Digest	Дайджест электронной почты.
	Alternative	Тело сообщения состоит из нескольких частей; все части семантически идентичны.
Message	RFC822	В теле содержится почтовое сообщение стандарта RFC 822.
	Partial	Фрагмент почтового сообщения.
	External-Body	Указатель на действительное почтовое сообщение (не включенное в тело данного сообщения).
Application	Octet-Stream	Произвольные двоичные данные.
	Postscript	Программа на языке Postscript.
Image	JPEG	Формат ISO 10918.
	GIF	Графический формат фирмы Compuserve.
Audio	Basic	Звук в 8-битном ISDN-формате mu-law.
Video	MPEG	Формат ISO 11172.

Поля заголовка Content-ID и Content-Description могут отсутствовать. Первое служит для идентификации MIME-содержимого электронного письма, а второе может содержать дополнительное описание. Например, если MIME-содержимым является графический образ, в поле Content-Description можно поместить описание этого образа.

В табл. 2.4 перечислены возможные значения поля Content-Tranfer-Encoding, доступные в настоящее время.

Таблица 2.4 - возможные значения поля Content-Tranfer-Encoding

Таблица 15.5. Допустимые значения поля Content-Transfer-Encoding

Content-Transfer-Encoding	Описание
7bit	Формат NVT-ASCII – стандартный формат почтовых сообщений.
Quoted-printable	Полезен в случае, если текст содержит небольшое количество восьмибитных символов.
Base64	Формат, в котором три байта данных упакованы в четыре шестибитных значения.
8bit	Содержит текст, в котором не все символы принадлежат стандартному набору ASCII (то есть, в некоторых установлен восьмой бит).
Binary	Восьмибитные данные, как правило, без символов окончания строки.

По умолчанию формат почтовых сообщений удовлетворяет кодовому набору NVT-ASCII. 8-битные агенты MTA сейчас практически отсутствуют, но как только они получат широкое распространение, вероятно, передача бинарной и текстовой информации в 8-битной кодировке возрастет. В настоящий момент для передачи 8-битной информации по 7-битным каналам Интернет лучше всего использовать кодировки quoted-printable или base64.

Способы кодирования MIME

Для кодирования небольшого количества 8-битных данных в 7-битный формат NVT ASCII лучше всего подходит схема quoted printable. 8-битный символ в этой схеме представляется в виде последовательности из трех

символов. Последовательность всегда начинается со знака «равно» (=). Сразу за знаком «равно» следует двузначное шестнадцатиричное число, представляющее код ASCII кодируемого символа. Рассмотрим закодированную quoted printable последовательность JAMSA PRESS. Хоть она и не содержит 8-битных символов, зато позволяет хорошо проиллюстрировать сам принцип кодирования. Закодированное сочетание JAMSA PRESS выглядит так:

=4A=41=4D=53=41=20=50=52=45=53=53

Другими словами, буква J имеет шестнадцатиричный код ASCII 0x4A, буква A— 0x41 и т.д. Обратите внимание на то, что схема quoted printable передает ASCII код для каждого символа последовательности. То есть для знака A (ASCII 0x4A) передается код знака «равно» (ASCII 0x3D), код цифры 4 (ASCII 0x34) и код знака A (0x41). Данную схему довольно удобно использовать, но, к сожалению, она утраивает общее количество информации в сообщении. Таким образом, область применения quoted printable— сообщение с небольшим количеством символов, в которых установлен старший (восьмой) бит. Основная часть сообщения должна состоять все-таки из обычных 7-битных символов.

Схема quoted printable годится только для небольшого количества символов с установленным восьмым битом.

В отличие от quoted printable, кодирование Base-64 увеличивает размер сообщения всего лишь на одну треть. Каждая последовательность из трех байтов (24бита) превращается в четыре шестибитовых значения (опять-таки, 24 бита). Шестибитные символы соответствуют формату NVT ASCII и приведены в таблице 2.5.

Таблица 2.5 - Таблица кодировки Base-64

6 бит	ASCII	6 бит	ASCII	6 бит	ASCII	6 бит	ASCII
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Способ кодирования Base-64 идентичен способу, описанному в RFC 1421 (Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures, Linn, 1993), где рассматриваются приложения шифрования электронной почты (Privacy Enhanced Mail applications, PEM).

Если количество байтов (символов) в сообщении не кратно трем, используется дополняющий символ «равно». На рис. 2.4 приведены три примера кодирования методом Base-64.

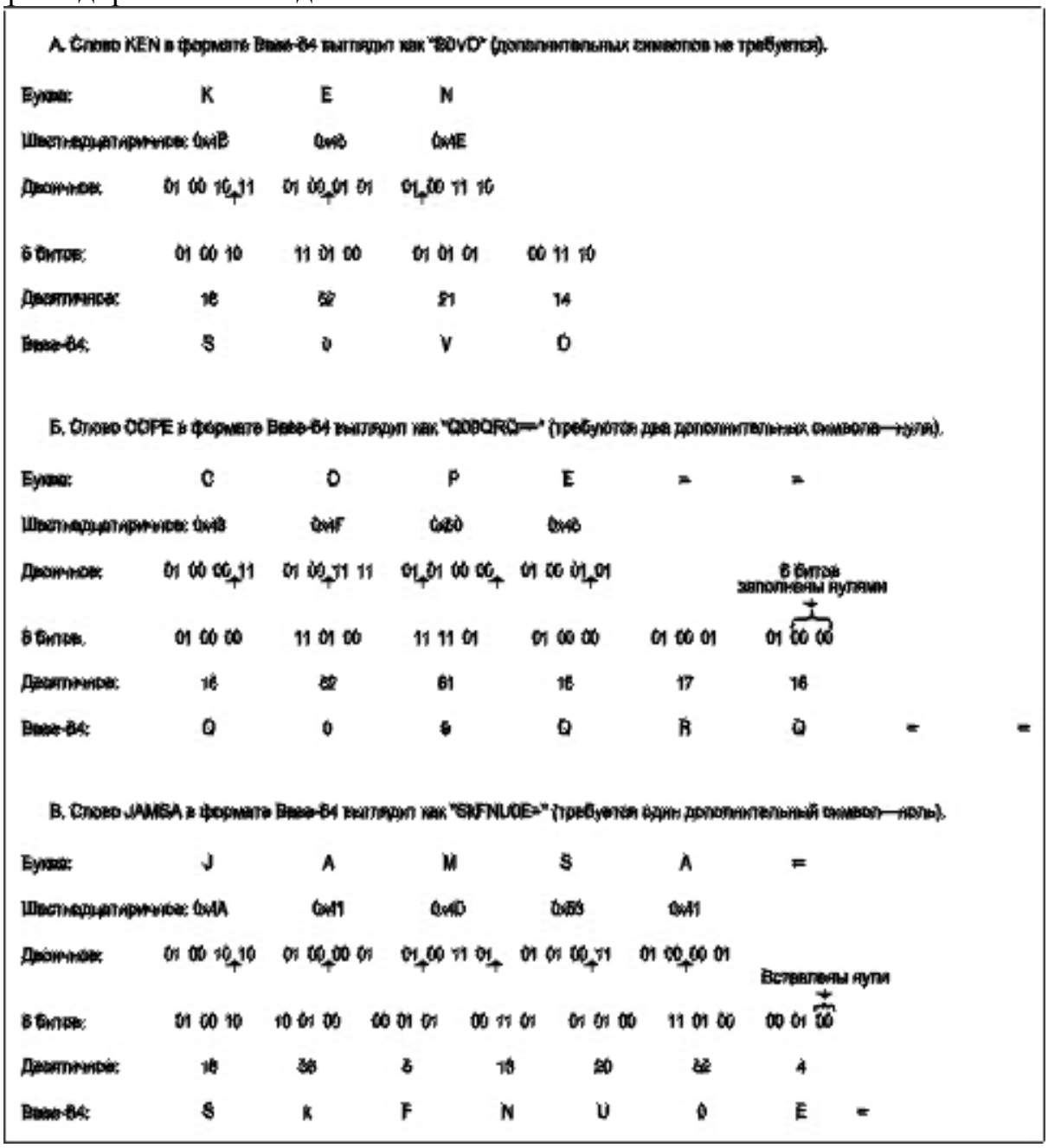


Рисунок 2.4 - Образцы кодирования методом Base-64 с дополнением

Протокол передачи почты POP

В некоторых небольших узлах Интернет бывает непрактично поддерживать систему передачи сообщений (MTS - Message Transport

System). Рабочая станция может не иметь достаточных ресурсов для обеспечения непрерывной работы SMTP-сервера [RFC-821]. Для “домашних ЭВМ” слишком дорого поддерживать связь с Интернет круглые сутки.

Но доступ к электронной почте необходим как для таких малых узлов, так и индивидуальных ЭВМ. Для решения этой проблемы разработан протокол POP3 (Post Office Protocol - Version 3, STD- 53. M. Rose, RFC-1939). Этот протокол обеспечивает доступ узла к базовому почтовому серверу.

POP3 не ставит целью предоставление широкого списка манипуляций с почтой. Почтовые сообщения принимаются почтовым сервером и сохраняются там, пока на рабочей станции клиента не будет запущено приложение POP3. Это приложение устанавливает соединение с сервером и забирает сообщения оттуда. Почтовые сообщения на сервере удаляются.

Более продвинутый и сложный протокол IMAP4 обсуждается в RFC-2060 (порт 143). Об аутентификации в POP3 можно прочесть в документе RFC-1734.

В дальнейшем ЭВМ-клиентом будет называться машина, пользующаяся услугами POP3, а ЭВМ-сервером - сторона, предлагающая услуги POP3.

Когда пользователь ЭВМ-клиента хочет послать сообщение, он устанавливает SMTP связь с почтовым сервером непосредственно и посылает все, что нужно через него. При этом ЭВМ POP3-сервер не обязательно является почтовым сервером.

В исходный момент ЭВМ POP3-сервер прослушивает TCP-порт 110. Если ЭВМ-клиент хочет воспользоваться услугами POP3-сервера, то устанавливает с ним TCP связь. По установлении связи POP3-сервер посылает клиенту уведомление (например, +OK POP3 server ready) и сессия переходит в фазу авторизации (см. также RFC-1734, -1957). После этого может производиться обмен командами и откликами.

Команды POP3 состоят из ключевых слов (3-4 символа), за которыми могут следовать аргументы. Каждая команда завершается парой символов CRLF. Как ключевые слова, так и аргументы могут содержать только печатаемые ASCII-символы. В качестве разделителя используются символы пробела. Каждый аргумент может содержать до 40 символов.

Сигнал отклика в POP3 содержит индикатор состояния и ключевое слово, за которым может следовать дополнительная информация. Отклик также завершается кодовой последовательностью CRLF. Длина отклика не превышает 512 символов, включая CRLF. Существует два индикатора состояния: положительный - "+OK" и отрицательный "-ERR" (все символы прописные).

Отклики на некоторые команды могут содержать несколько строк. В этом случае последняя строка содержит код завершения 046 ("."), за которым следует CRLF.

На практике многострочные отклики для исключения имитации завершаются последовательностью CRLF.CRLF.

процессе авторизации клиент должен представить себя серверу, передав имя и пароль (возможен вариант посылки команды APOP). Если авторизация успешно завершена, сессия переходит в состояние транзакции (TRANSACTION). При получении от клиента команды QUIT сессия переходит в состояние UPDATE, при этом все ресурсы освобождаются и TCP связь разрывается.

На синтаксически неузнанные и неверные команды, сервер реагирует, посылая отрицательный индикатор состояния.

POP3 сервер может быть снабжен таймером пассивного состояния (10 мин.), который осуществляет автоматическое прерывание сессии. Приход любой команды со стороны клиента сбрасывает этот таймер в нуль.

Сервер нумерует все передаваемые сообщения из своего почтового ящика и определяет их длину. Положительный отклик начинается с +OK, за ним следует пробел, номер сообщения, еще один пробел и длина сообщения в октетах. Завершается отклик последовательностью CRLF. Переданные сообщения удаляются из почтового ящика сервера. Все сообщения, передаваемые во время сессии POP3 должны следовать рекомендациям формата Интернет сообщений [RFC822].

В состоянии транзакции клиент может посылать серверу последовательность POP3 команд, на каждую из которых сервер должен послать отклик. Далее следует краткое описание команд, используемых в состоянии транзакция.

LIST [сообщение]

Аргументы: номер сообщения (опционно), который не может относиться к сообщению, помеченному как удаленное. Команда может быть выдана только в режиме TRANSACTION. При наличии аргумента сервер выдает положительный отклик, содержащий информационную строку сообщения. Такая строка называется скэн-листингом сообщения (scan listing). scan listing состоит из номера сообщения, за которым следует пробел и число октетов в сообщении. Сообщения, помеченные как удаленные, не пересылаются. Примером отрицательного отклика может служить: -ERR no such message.

Примеры использования команды LIST:

Клиент выдает команду: LIST
Сервер откликается: +OK 2 messages (320 octets)
Сервер: 1 120
Сервер: 2 200
Сервер: .
...
Клиент: LIST 2
Сервер: +OK 2 200
...
K: LIST 3
C: -ERR no such message, only 2 messages in maildrop

Здесь и далее символом К обозначается клиент, а символом С - сервер.

STAT - аргументов не использует, возможный отклик +OK nn mm, где nn - номер сообщения, а mm - его длина в байтах. Пример использования:

K: STAT
C: +OK 2 320

QUIT - аргументов не использует, возможный отклик +OK.

Сервер POP3 удаляет все сообщения, помеченные как удаленные из почтового ящика, посылает соответствующий отклик и разрывает TCP связь. Пример:

K: QUIT
C: +OK dewey POP3 server signing off.
RETR msg (msg - номер сообщения)

Если POP3-сервер выдал положительный отклик, то за начальным +OK следует сообщение с номером, указанным в аргументе. Отрицательный отклик имеет вид -ERR no such message.

Пример использования команды:

K: RETR 1
C: +OK 120 octets
C:
C: .

DELE msg (msg - номер сообщения)

Сервер POP3 помечает сообщение как удаленное. Любая ссылка на это сообщение в будущем вызовет ошибку. При этом само сообщение не удаляется пока сессия не войдет в режим UPDATE.

Пример использования команды:

K: DELE 1
C: +OK message 1 deleted
...
K: DELE 2
C: -ERR message 2 already deleted

NOOP (не использует каких-либо аргументов). При реализации этой команды сервер не делает ничего, лишь посылает положительный отклик.

RSET (не использует каких-либо аргументов)

Если какие-либо сообщения помечены как удаленные, сервер POP3 удаляет эту пометку и возвращает положительный отклик. Например:

K: RSET
C: +OK maildrop has 2 messages (320 octets)

Если сессия завершается не по команде клиента, то перехода в состояние UPDATE не производится, а сообщения не удаляются из почтового ящика. Далее следует описание команд, используемых в состоянии UPDATE.

Ряд команд не входят в перечень обязательных (являются опционными).

TOP msg n, где msg - номер сообщения, а n - число строк (используется только в режиме TRANSACTION).

При положительном отклике на команду TOP сервер посылает заголовки сообщений и вслед за ними n строк их текста. Если n больше числа строк в сообщении, посылается все сообщение.

UIDL [msg], где msg - номер сообщения является опционным (Unique-ID Listing).

Если сервер выдаст положительный отклик, будет выдана строка, содержащая информацию о данном сообщении. Эта строка называется уникальным идентификатором сообщения ("unique-id listing"). При отсутствии аргумента аналогичная информация выдается для каждого из сообщений в почтовом ящике. Уникальный идентификатор сообщения состоит из 1-70 символов в диапазоне от 0x21 до 0x7E. Сообщения в почтовом ящике должны характеризоваться различными идентификаторами. Пример использования команды:

K: UIDL
C: +OK
C: 1 whqtswO00WBw418f9t5JxYwZ
C: 2 QhdPYR:00WBw1Ph7x7

USER name, где name - характеризует почтовый ящик сервера.

Команда используется на фазе авторизации или после неудачного завершения команд USER или PASS. При авторизации клиент должен сначала послать команду USER и лишь после получения положительного отклика команду PASS. Команда может вызвать следующие отклики:

+OK name is a valid mailbox
-ERR never heard of mailbox name

Примеры использования команды USER:

K: USER frated
C: -ERR sorry, no mailbox for frated here
...
K: USER mrose
C: +OK mrose is a real hoopy frood

PASS string (string - пароль для доступа к почтовому серверу)

Команда работает в режиме авторизации сразу после команды USER. Когда клиент выдает команду PASS, сервер использует аргументы команд USER и PASS для определения доступа клиента к почтовому ящику. На команду PASS возможны следующие отклики:

+OK maildrop locked and ready
-ERR invalid password
-ERR unable to lock maildrop

Пример диалога при использовании команды PASS:

K: USER mrose
C: +OK mrose is a real hoopy frood
K: PASS secret
C: -ERR maildrop already locked
...
K: USER mrose

C: +OK mrose is a real hoopy frood
K: PASS secret
C: +OK mrose's maildrop has 2 messages (320 octets)

APOP name digest, где name - идентификатор почтового ящика, a digest - дайджест сообщения - MD5 (RFC-1828). Команда используется только на стадии авторизации.

Обычно любая сессия начинается с обмена USER/PASS. Но так как в некоторых случаях подключения к серверу POP3 может осуществляться достаточно часто, возрастает риск перехвата пароля. Альтернативным методом авторизации является использование команды APOP. Сервер, который поддерживает применение команды APOP, добавляет временную метку в свое стартовое уведомление. Синтаксис временной метки соответствует формату идентификаторов сообщений, описанному в [RFC822] и должны быть уникальными для всех заголовков уведомлений.

где `process-ID' представляет собой десятичное значение PID процесса, clock - десятичное показание системных часов, a hostname - полное имя домена, где размещен сервер POP3.

Клиент POP3 фиксирует временную метку и выдает команду APOP. Параметр `name' семантически идентичен параметру `name' команды USER. Параметр `digest' вычисляется с использованием алгоритма MD5 [RFC1321] для строки, состоящей из временной метки (включая угловые скобки) за которой следует строка пароля, которая известна только клиенту и серверу. Параметр `digest' содержит 16 октетов, которые пересылаются в шестнадцатеричном формате с использованием строчных ASCII символов. Сервер, получив команду APOP, проверяет принятый дайджест и если он корректен, посылает положительный отклик клиенту. Сессия при этом переходит в состояние транзакции. В противном случае посылается отрицательный отклик и состояние сессии не изменяется. С целью обеспечения безопасности для каждого конкретного пользователя и сервера должен использоваться либо метод доступа USER/PASS, либо APOP, но не в коем случае оба метода попеременно.

Сервер перед закрытием канала по команде QUIT должен удалить из почтового ящика все сообщения, которые были перенесены с помощью команд RETR.

Предполагается, что все сообщения, передаваемые в ходе сессии POP3, имеют текстовый формат Интернет в соответствии с документом [RFC822].

Задание к лабораторной работе №2 :

1) Изучить протоколы обмена электронной почты.

2) Разработать почтовый клиент, с возможностью передачи почты по протоколам SMTP и приема почты с помощью протоколов POP3, IMAP. Настроить клиент с учетом выбранного протокола и проверить его

функциональность с использованием бесплатного почтового сервера (например создать почтовый ящик на rambler.ru).

Требования к отчету по лабораторной работе №2

В отчете по лабораторной работе должны быть представлены:

1) Концепции взаимодействия программы-клиента и программы сервера для протоколов SMTP, POP3, IMAP.
2) Листинг программ реализации клиентской части протоколов SMTP, POP3, IMAP.
3) Экранные формы, демонстрирующие работу программ.

Контрольные вопросы:

1) Какой порт использует протокол SMTP?
2) Какой порт использует протокол POP3 и IMAP?
3) Охарактеризуйте достоинства и недостатки протокола POP3.
4) Охарактеризуйте достоинства и недостатки протокола IMAP.
5) Охарактеризуйте достоинства и недостатки протокола SMTP.
6) Что такое ESMTP?
7) Что такое Base64?

ЛАБОРАТОРНАЯ РАБОТА №3

Тема: Разработка простейших клиент-серверных программ с использованием UDP сокетов на базе ОС Linux.

Цель работы: Изучение принципов взаимодействия клиент-серверных программ с использованием UDP сокетов на базе OS Linux.

Методические указания к выполнению лабораторной работы

Организация связи между удаленными процессами с помощью датаграмм

Как уже упоминалось, более простой для взаимодействия удаленных процессов является схема организации общения клиента и сервера с помощью датаграмм, т. е. использование протокола UDP.

Процесс-сервер должен сначала совершить подготовительные действия:

1) *создать UDP-сокет;*
2) *связать его с определенным номером порта и IP-адресом сетевого интерфейса*
3) *настроить адрес сокета. При этом сокет может быть привязан к конкретному сетевому интерфейсу или к компьютеру в целом, то есть в полном адресе сокета может быть либо указан IP-адрес конкретного сетевого интерфейса, либо дано указание операционной системе, что информация может поступить через любой сетевой интерфейс, имеющийся в наличии.*
4) *После настройки адреса сокета операционная система начинает принимать сообщения, пришедшие на этот адрес и складывать их в сокет.*
5) *Сервер дожидается поступления сообщения, читает его, определяет, от кого оно поступило и через какой сетевой интерфейс, обрабатывает полученную информацию и отправляет результат по обратному адресу. После чего процесс готов к приему новой информации от того же или другого клиента.*

Процесс-клиент должен сначала совершить те же самые подготовительные действия: создать сокет и настроить его адрес. Затем он передает сообщение, указав, кому оно предназначено (IP-адрес сетевого интерфейса и номер порта сервера), ожидает от него ответа и продолжает свою деятельность.

Основные системные вызовы для работы с UDP-сокетами

- Создание сокета производится с помощью системного вызова socket().

- Для привязки созданного сокета к IP-адресу и номеру порта (настройка адреса) служит системный вызов bind().

- Ожиданию получения информации, ее чтению и, при необходимости, определению адреса отправителя соответствует системный вызов recvfrom().

- За отправку датаграммы отвечает системный вызов sendto().

Как известно, порядок байт в целых числах, представление которых занимает более одного байта, может быть для различных компьютеров неодинаковым. Есть вычислительные системы, в которых старший байт числа имеет меньший адрес, чем младший байт (big-endian byte order), а есть вычислительные системы, в которых старший байт числа имеет больший адрес, чем младший байт (little-endian byte order). *При передаче целой числовой информации от машины, имеющей один порядок байт, к машине с другим порядком байт мы можем неправильно истолковать принятую информацию. Для того чтобы этого не произошло, было введено понятие сетевого порядка байт, т.е. порядка байт, в котором должна представляться целая числовая информация в процессе передачи ее по сети* (на самом деле – это big-endian byte order). *Целые числовые данные из представления, принятого на компьютере-отправителе, переводятся пользовательским процессом в сетевой порядок байт, в таком виде путешествуют по сети и переводятся в нужный порядок байт на машине-получателе процессом, которому они предназначены*.

Для перевода целых чисел из машинного представления в сетевое и обратно используется четыре функции: htons(), htonl(), ntohs(), ntohl().

Прототипы функций

```
#include <netinet/in.h>
unsigned long int htonl(
   unsigned long int hostlong);
unsigned short int htons(
   unsigned short int hostshort);
unsigned long int ntohl(
   unsigned long int netlong);
unsigned short int ntohs(
   unsigned short int netshort);
```

Описание функций

Функция htonl осуществляет перевод целого длинного числа из порядка байт, принятого на компьютере, в сетевой порядок байт.

Функция htons осуществляет перевод целого короткого числа из порядка байт, принятого на компьютере, в сетевой порядок байт.

Функция ntohl осуществляет перевод целого длинного числа из сетевого порядка байт в порядок байт, принятый на компьютере.

Функция ntohs осуществляет перевод целого короткого числа из сетевого порядка байт в порядок байт, принятый на компьютере.

В архитектуре компьютеров i80x86 принят порядок байт, при котором младшие байты целого числа имеют младшие адреса. При сетевом порядке байт, принятом в Internet, младшие адреса имеют старшие байты числа.

Параметр у них – значение, которое мы собираемся конвертировать. Возвращаемое значение – то, что получается в результате конвертации. Направление конвертации определяется порядком букв h (host) и n (network) в названии функции, размер числа – последней буквой названия, то есть htons – это host to network short, ntohl – network to host long.

Для чисел с плавающей точкой все обстоит гораздо хуже. На разных машинах могут различаться не только порядок байт, но и форма представления такого числа. Простых функций для их корректной передачи по сети не существует. Если требуется обмениваться действительными данными, то либо это нужно делать на гомогенной сети, состоящей из одинаковых компьютеров, либо использовать символьные и целые данные для передачи действительных значений.

Функции преобразования IP-адресов inet_ntoa(), inet_aton()

Нам также понадобятся функции, осуществляющие перевод IP-адресов из символьного представления (в виде четверки чисел, разделенных точками) в числовое представление и обратно. Функция inet_aton() переводит символьный IP-адрес в числовое представление в сетевом порядке байт.

Функция возвращает 1, если в символьном виде записан правильный IP-адрес, и 0 в противном случае – для большинства системных вызовов и функций это нетипичная ситуация. Обратите внимание на использование указателя на структуру struct in_addr в качестве одного из параметров данной функции. Эта структура используется для хранения IP-адресов в сетевом порядке байт. То, что используется структура, состоящая из одной переменной, а не сама 32-битовая переменная, сложилось исторически, и авторы в этом не виноваты.

Для обратного преобразования применяется функция inet_ntoa().

Функции преобразования IP-адресов

Прототипы функций

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
int inet_aton(const char *strptr,
    struct in_addr *addrptr);
```

```
char *inet_ntoa(struct in_addr *addrptr);
```

Описание функций

Функция inet_aton переводит символьный IP-адрес, расположенный по указателю strptr, в числовое представление в сетевом порядке байт и заносит его в структуру, расположенную по адресу addrptr. Функция возвращает значение 1, если в строке записан правильный IP-адрес, и значение 0 в противном случае. Структура типа struct in_addr используется для хранения IP-адресов в сетевом порядке байт и выглядит так:

```
struct in_addr {
   in_addr_t s_addr;
};
```

То, что используется адрес такой структуры, а не просто адрес переменной типа in_addr_t, сложилось исторически.

Функция inet_ntoa применяется для обратного преобразования. Числовое представление адреса в сетевом порядке байт должно быть занесено в структуру типа struct in_addr, адрес которой addrptr передается функции как аргумент. Функция возвращает указатель на строку, содержащую символьное представление адреса. Эта строка располагается в статическом буфере, при последующих вызовах ее новое содержимое заменяет старое содержимое.

Функция bzero()

Функция 2 настолько проста, что про нее нечего рассказывать. Все видно из описания.

Функция bzero

Прототип функции
#include <string.h>
*void bzero(void *addr, int n);*

Описание функции

Функция bzero заполняет первые n байт, начиная с адреса addr, нулевыми значениями. Функция ничего не возвращает.

Создание сокета. Системный вызов socket()

Для создания сокета в операционной системе служит системный вызов socket(). Для транспортных протоколов семейства TCP/IP

существует два вида сокетов: UDP-сокет – сокет для работы с датаграммами, и TCP сокет – потоковый сокет. Однако понятие сокета не ограничивается рамками только этого семейства протоколов. Рассматриваемый интерфейс сетевых системных вызовов (socket(), bind(), recvfrom() , sendto() и т. д.) в операционной системе UNIX может применяться и для других стеков протоколов (и для протоколов, лежащих ниже транспортного уровня).

При создании сокета необходимо точно специфицировать его тип. Эта *спецификация производится с помощью трех параметров вызова socket(). Первый параметр указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства*.

Второй параметр служит для задания вида интерфейса работы с сокетом – будет это потоковый сокет, сокет для работы с датаграммами или какой-либо иной. Третий параметр указывает протокол для заданного типа интерфейса. В стеке протоколов TCP/IP существует только один протокол для потоковых сокетов – TCP и только один протокол для датаграммных сокетов – UDP, поэтому *для транспортных протоколов TCP/IP третий параметр игнорируется.*

В других стеках протоколов может быть несколько протоколов с одинаковым видом интерфейса, например, датаграммных, различающихся по степени надежности.

Для транспортных протоколов TCP/IP мы всегда в качестве первого параметра будем указывать предопределенную константу AF_INET (Address family – Internet) или ее синоним PF_INET (Protocol family – Internet).

Второй параметр будет принимать предопределенные значения SOCK_STREAM для потоковых сокетов и SOCK_DGRAM – для датаграммных.

Поскольку третий параметр в нашем случае не учитывается, в него мы будем подставлять значение 0.

Ссылка на информацию о созданном сокете помещается в таблицу открытых файлов процесса. *Системный вызов возвращает пользователю файловый дескриптор, соответствующий заполненному элементу таблицы, который далее мы будем называть дескриптором сокета*. Такой способ хранения информации о сокете позволяет, во-первых, процессам-детям наследовать ее от процессов-родителей, а, во-вторых, использовать для сокетов системные вызовы: close(), read(), write().

Системный вызов для создания сокета

Прототип системного вызова

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type,
        int protocol);
```

Описание системного вызова

Системный вызов socket служит для создания виртуального коммуникационного узла в операционной системе. За полной информацией обращайтесь к UNIX Manual.

Параметр domain определяет семейство протоколов, в рамках которого будет осуществляться передача информации Предопределенные значения параметра:

PF_INET – для семейства протоколов TCP/IP;

PF_UNIX – для семейства внутренних протоколов UNIX, иначе называемого еще UNIX domain.

Параметр type определяет семантику обмена информацией: будет ли осуществляться связь через сообщения (datagrams), с помощью установления виртуального соединения или еще каким-либо способом. Будем пользоваться только двумя способами обмена информацией с предопределенными значениями для параметра type:

SOCK_STREAM – для связи с помощью установления виртуального соединения;

SOCK_DGRAM – для обмена информацией через сообщения.

Параметр protocol специфицирует конкретный протокол для выбранного семейства протоколов и способа обмена информацией. Он имеет значение только в том случае, когда таких протоколов существует несколько. В нашем случае семейство протоколов и тип обмена информацией определяют протокол однозначно. Поэтому данный параметр мы будем полагать равным 0.

Возвращаемое значение

В случае успешного завершения системный вызов возвращает файловый дескриптор (значение большее или равное 0), который будет использоваться как ссылка на созданный коммуникационный узел при всех дальнейших сетевых вызовах. При возникновении какой-либо ошибки возвращается отрицательное значение.

Адреса сокетов. Настройка адреса сокета. Системный вызов bind()

Когда сокет создан, необходимо настроить его адрес. Для этого используется системный вызов bind(). Первый параметр вызова должен содержать дескриптор сокета, для которого производится настройка адреса. Второй и третий параметры задают этот адрес.

Во втором параметре должен быть указатель на структуру struct sockaddr, содержащую удаленную и локальные части полного адреса.

*Указатели типа struct sockaddr * встречаются во многих сетевых системных вызовах; они используются для передачи информации о том, к какому адресу привязан или должен быть привязан сокет. Рассмотрим этот тип данных подробнее. Структура struct sockaddr описана в файле <sys/socket.h> следующим образом:*

```
struct sockaddr {
   short sa_family;
   char sa_data[14];
};
```

Такой состав структуры обусловлен тем, что сетевые системные вызовы могут применяться для различных семейств протоколов, которые по-разному определяют адресные пространства для удаленных и локальных адресов сокета. По сути дела, этот тип данных представляет собой лишь общий шаблон для передачи системным вызовам структур данных, специфических для каждого семейства протоколов. *Общим элементом этих структур остается только поле short sa_family* (которое в разных структурах, естественно, может иметь разные имена, важно лишь, чтобы все они были одного типа и были первыми элементами своих структур) для описания семейства протоколов. Содержимое этого поля системный вызов анализирует для точного определения состава поступившей информации.

Для работы с семейством протоколов TCP/IP мы будем использовать адрес сокета следующего вида, описанного в файле <netinet/in.h>:

```
struct sockaddr _in{
   short sin_family;
   /* Избранное семейство протоколов
     – всегда AF_INET */
   unsigned short sin_port;
   /* 16-битовый номер порта в сетевом
     порядке байт */
   struct in_addr sin_addr;
   /* Адрес сетевого интерфейса */
   char sin_zero[8];
   /* Это поле не используется, но должно
```

```
        всегда быть заполнено нулями */
};
```

Первый элемент структуры – sin_family задает семейство протоколов. В него мы будем заносить предопределенную константу AF_INET.

Удаленная часть полного адреса – IP-адрес – содержится в структуре типа struct in_addr, (Функции преобразования IP-адресов inet_ntoa(), inet_aton())" .

Для указания номера порта предназначен элемент структуры sin_port, в котором номер порта должен храниться в сетевом порядке байт. Существует два варианта задания номера порта: фиксированный порт по желанию пользователя и порт, который произвольно назначает операционная система. Первый вариант требует указания в качестве номера порта положительного заранее известного числа и для протокола UDP обычно используется при настройке адресов сокетов и при передаче информации с помощью системного вызова sendto(). Второй вариант требует указания в качестве номера порта значения 0. В этом случае операционная система сама привязывает сокет к свободному номеру порта. Этот способ обычно используется при настройке сокетов программ клиентов, когда заранее точно знать номер порта для программисту необязательно.

Какой номер порта может задействовать пользователь при фиксированной настройке. *Номера портов с 1 по 1023 могут назначать сокетам только процессы, работающие с привилегиями системного администратора. Как правило, эти номера закреплены за системными сетевыми службами* независимо от вида используемой операционной системы, для того чтобы пользовательские клиентские программы могли запрашивать обслуживание всегда по одним и тем же локальным адресам. *Существует также ряд широко применяемых сетевых программ, которые запускают процессы с полномочиями обычных пользователей (например, X-Windows). Для таких программ корпорацией Internet по присвоению имен и номеров (ICANN) выделяется диапазон адресов с 1024 по 49151*, который нежелательно использовать во избежание возможных конфликтов. *Номера портов с 49152 по 65535 предназначены для процессов обычных пользователей.* ***Во всех примерах при фиксированном задании номера порта у сервера будем использовать номер 51000.***

IP–адрес при настройке также может быть определен двумя способами. Он может быть привязан к конкретному сетевому интерфейсу (т.е. сетевой плате), заставляя операционную систему принимать/передавать информацию только через этот сетевой интерфейс*, а может быть привязан и ко всей вычислительной системе в целом* (информация может быть

получена/отослана через любой сетевой интерфейс). *В первом случае в качестве значения поля структуры sin_addr.s_addr используется числовое значение IP-адреса конкретного сетевого интерфейса в сетевом порядке байт. Во втором случае это значение должно быть равно значению предопределенной константы INADDR_ANY, приведенному к сетевому порядку байт.*

Третий параметр системного вызова bind() должен содержать фактическую длину структуры, адрес которой передается в качестве второго параметра. Эта длина меняется в зависимости от семейства протоколов и даже различается в пределах одного семейства протоколов. *Размер структуры, содержащей адрес сокета, для семейства протоколов TCP/IP может быть определен как sizeof(struct sockaddr_in).*

Системный вызов для привязки сокета к конкретному адресу

Прототип системного вызова

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockd,
        struct sockaddr *my_addr,
        int addrlen);
```

Описание системного вызова

Системный вызов bind служит для привязки созданного сокета к определенному полному адресу вычислительной сети.

Параметр sockd является дескриптором созданного ранее коммуникационного узла, т. е. значением, которое вернул системный вызов socket().

Параметр my_addr представляет собой адрес структуры, содержащей информацию о том, куда именно мы хотим привязать наш сокет – то, что принято называть адресом сокета. Он имеет тип указателя на структуру-шаблон struct sockaddr, которая должна быть конкретизирована в зависимости от используемого семейства протоколов и заполнена перед вызовом.

Параметр addrlen должен содержать фактическую длину структуры, адрес которой передается в качестве второго параметра. Эта длина в разных семействах протоколов и даже в пределах одного семейства протоколов может быть различной (например, для UNIX Domain).

Возвращаемое значение

Системный вызов возвращает значение 0 при нормальном завершении и отрицательное значение – в случае ошибки.

Системные вызовы sendto() и recvfrom()

Для отправки датаграмм применяется системный вызов sendto(). В число параметров этого вызова входят:

дескриптор сокета, через который отсылается датаграмма;

адрес области памяти, где лежат данные, которые должны составить содержательную часть датаграммы, и их длина;

флаги, определяющие поведение системного вызова (в нашем случае они всегда будут иметь значение 0);

указатель на структуру, содержащую адрес сокета получателя, и ее фактическая длина.

Системный вызов возвращает отрицательное значение при возникновении ошибки и количество реально отосланных байт при нормальной работе. Нормальное завершение системного вызова не означает, что датаграмма уже покинула ваш компьютер. *Датаграмма сначала помещается в системный сетевой буфер, а ее реальная отправка может произойти после возврата из системного вызова. Вызов sendto() может блокироваться, если в сетевом буфере не хватает места для датаграммы.*

Для чтения принятых датаграмм и определения адреса получателя (при необходимости) служит системный вызов *recvfrom().*

В число параметров этого вызова входят:

Дескриптор сокета, через который принимается датаграмма.

Адрес области памяти, куда следует положить данные, составляющие содержательную часть датаграммы.

Максимальная длина, допустимая для датаграммы. Если количество данных датаграммы превышает заданную максимальную длину, то вызов по умолчанию рассматривает это как ошибочную ситуацию.

Флаги, определяющие поведение системного вызова (в нашем случае они будут полагаться равными 0).

Указатель на структуру, в которую при необходимости может быть занесен адрес сокета отправителя. Если этот адрес не требуется, то можно указать значение NULL.

Указатель на переменную, содержащую максимально возможную длину адреса отправителя. После возвращения из системного вызова в нее будет занесена фактическая длина структуры, содержащей адрес отправителя. Если предыдущий параметр имеет значение NULL, то и этот параметр может иметь значение NULL.

Системный вызов recvfrom() по умолчанию блокируется, если отсутствуют принятые датаграммы, до тех пор, пока датаграмма не появится. При возникновении ошибки он возвращает отрицательное значение, при нормальной работе – длину принятой датаграммы.

Системные вызовы sendto и recvfrom

```
Прототипы системных вызовов
#include <sys/types.h>
#include <sys/socket.h>
int sendto(int sockd, char *buff,
    int nbytes, int flags,
    struct sockaddr *to, int addrlen);
int recvfrom(int sockd, char *buff,
    int nbytes, int flags,
    struct sockaddr *from, int *addrlen);
```

Описание системных вызовов

Системный вызов sendto предназначен для отправки датаграмм. Системный вызов recvfrom предназначен для чтения пришедших датаграмм и определения адреса отправителя. По умолчанию при отсутствии пришедших датаграмм вызов recvfrom блокируется до тех пор, пока не появится датаграмма. Вызов sendto может блокироваться при отсутствии места под датаграмму в сетевом буфере. Данное описание не является полным описанием системных вызовов За полной информацией обращайтесь к UNIX Manual.

Параметр sockd является дескриптором созданного ранее сокета, т. е. значением, возвращенным системным вызовом socket(), через который будет отсылаться или получаться информация.

Параметр buff представляет собой адрес области памяти, начиная с которого будет браться информация для передачи или размещаться принятая информация.

Параметр nbytes для системного вызова sendto определяет количество байт, которое должно быть передано, начиная с адреса памяти buff. Параметр nbytes для системного вызова recvfrom определяет максимальное количество байт, которое может быть размещено в приемном буфере, начиная с адреса buff.

Параметр to для системного вызова sendto определяет ссылку на структуру, содержащую адрес сокета получателя информации, которая должна быть заполнена перед вызовом. Если параметр from для системного

вызова recvfrom не равен NULL, то для случая установления связи через пакеты данных он определяет ссылку на структуру, в которую будет занесен адрес сокета отправителя информации после завершения вызова. В этом случае перед вызовом эту структуру необходимо обнулить.

Параметр addrlen для системного вызова sendto должен содержать фактическую длину структуры, адрес которой передается в качестве параметра to. Для системного вызова recvfrom параметр addrlen является ссылкой на переменную, в которую будет занесена фактическая длина структуры адреса сокета отправителя, если это определено параметром from. Заметим, что перед вызовом этот параметр должен указывать на переменную, содержащую максимально допустимое значение такой длины. Если параметр from имеет значение NULL, то и параметр addrlen может иметь значение NULL.

Параметр flags определяет режимы использования системных вызовов. Рассматривать его не будем, и поэтому берем значение этого параметра равным 0.

Возвращаемое значение

В случае успешного завершения системный вызов возвращает количество реально отосланных или принятых байт. При возникновении какой-либо ошибки возвращается отрицательное значение.

Определение IP-адресов для вычислительного комплекса

Для определения IP-адресов на компьютере можно воспользоваться утилитой /sbin/ifconfig. Эта утилита выдает всю информацию о сетевых интерфейсах, сконфигурированных в вычислительной системе. Пример выдачи утилиты показан ниже:

```
eth0    Link encap:Ethernet HWaddr 00:90:27:A7:1B:FE
   inet addr:192.168.253.12 Bcast:192.168.253.255
   Mask:255.255.255.0
   UP BROADCAST NOTRAILERS RUNNING MULTICAST MTU:1500
   Metric:1 RX packets:122556059 errors:0 dropped:0
   overruns:0 frame:0 TX packets:116085111 errors:0
   dropped:0 overruns:0 carrier:0 collisions:0
   txqueuelen:100 RX bytes:2240402748 (2136.6 Mb)
   TX bytes:3057496950 (2915.8 Mb) Interrupt:10
   Base address:0x1000
lo    Link encap:Local Loopback
   inet addr:127.0.0.1 Mask:255.0.0.0
   UP LOOPBACK RUNNING MTU:16436 Metric:1
   RX packets:403 errors:0 dropped:0 overruns:0 frame:0
   TX packets:403 errors:0 dropped:0 overruns:0
```

carrier:0 collisions:0 txqueuelen:0
RX bytes:39932 (38.9 Kb) TX bytes:39932 (38.9 Kb)

Сетевой интерфейс eth0 использует протокол Ethernet. Физический 48-битовый адрес, зашитый в сетевой карте, – 00:90:27:A7:1B:FE. Его IP-адрес – 192.168.253.12.

Сетевой интерфейс lo не относится ни к какой сетевой карте. Это так называемый локальный интерфейс, который через общую память эмулирует работу сетевой карты для взаимодействия процессов, находящихся на одной машине по полным сетевым адресам. Наличие этого интерфейса позволяет отлаживать сетевые программы на машинах, не имеющих сетевых карт. Его IP-адрес обычно одинаков на всех компьютерах – 127.0.0.1.

Пример программы UDP-клиента

Рассмотрим, наконец, простой пример программы. Эта программа является UDP-клиентом для стандартного системного сервиса echo. Стандартный сервис принимает от клиента текстовую датаграмму и, не изменяя ее, отправляет обратно. За сервисом зарезервирован номер порта 7. Для правильного запуска программы необходимо указать символьный IP-адрес сетевого интерфейса компьютера, к сервису которого нужно обратиться, в качестве аргумента командной строки, например:

a.out 192.168.253.12

```
/* Простой пример UDP клиента для сервиса echo */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
int main(int argc, char **argv)
{
   int sockfd; /* Дескриптор сокета */
   int n, len; /* Переменные для различных длин и
      количества символов */
   char sendline[1000], recvline[1000]; /* Массивы
      для отсылаемой и принятой строки */
   struct sockaddr_in servaddr, cliaddr; /* Структуры для
   адресов сервера и клиента */
   /* Сначала проверяем наличие второго аргумента в
   командной строке. При его отсутствии ругаемся и прекращаем
   работу */
   if(argc != 2){
      printf("Usage: a.out <IP address>\n");
```

```
        exit(1);
    }
    /* Создаем UDP сокет */
    if((sockfd = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
        perror(NULL); /* Печатаем сообщение об ошибке */
        exit(1);
    }
    /* Заполняем структуру для адреса клиента: семейство
    протоколов TCP/IP, сетевой интерфейс – любой, номер порта
    по усмотрению операционной системы. Поскольку в структуре
    содержится дополнительное не нужное нам поле, которое
    должно     быть нулевым, перед заполнением обнуляем ее всю */
    bzero(&cliaddr, sizeof(cliaddr));
    cliaddr.sin_family = AF_INET;
    cliaddr.sin_port = htons(0);
    cliaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    /* Настраиваем адрес сокета */
    if(bind(sockfd, (struct sockaddr *) &cliaddr,
    sizeof(cliaddr)) < 0){
        perror(NULL);
        close(sockfd); /* По окончании работы закрываем
        дескриптор сокета */
        exit(1);
    }
    /* Заполняем структуру для адреса сервера:
семейство протоколов TCP/IP, сетевой интерфейс – из аргумента
командной строки, номер порта 7. Поскольку в
структуре содержится дополнительное не нужное нам
поле, которое должно быть нулевым, перед заполнением
обнуляем ее всю */
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(7);
    if(inet_aton(argv[1], &servaddr.sin_addr) == 0){
        printf("Invalid IP address\n");
        close(sockfd); /* По окончании работы закрываем
            дескриптор сокета */
        exit(1);
    }
    /* Вводим строку, которую отошлем серверу */
    printf("String => ");
    fgets(sendline, 1000, stdin);
    /* Отсылаем датаграмму */
    if(sendto(sockfd, sendline, strlen(sendline)+1,
    0, (struct sockaddr *) &servaddr,
    sizeof(servaddr)) < 0){
    perror(NULL);
    close(sockfd);
    exit(1);
    }
    /* Ожидаем ответа и читаем его. Максимальная
допустимая длина датаграммы – 1000 символов,
```

```
адрес отправителя нам не нужен */
   if((n = recvfrom(sockfd, recvline, 1000, 0,
   (struct sockaddr *) NULL, NULL)) < 0){
      perror(NULL);
      close(sockfd);
      exit(1);
   }
   /* Печатаем пришедший ответ и закрываем сокет */
   printf("%s\n", recvline);
   close(sockfd);
   return 0;
}
```

Пример программы UDP-сервера

Поскольку UDP-сервер использует те же самые системные вызовы, что и UDP-клиент, мы можем сразу приступить к рассмотрению примера UDP-сервера для сервиса echo.

```
/* Простой пример UDP-сервера для сервиса echo */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
int main()
{
   int sockfd; /* Дескриптор сокета */
   int clilen, n; /* Переменные для различных длин
      и количества символов */
   char line[1000]; /* Массив для принятой и
      отсылаемой строки */
   struct sockaddr_in servaddr, cliaddr; /* Структуры
      для адресов сервера и клиента */
   /* Заполняем структуру для адреса сервера: семейство
   протоколов TCP/IP, сетевой интерфейс – любой, номер порта
   51000. Поскольку в структуре содержится дополнительное не
   нужное нам поле, которое должно быть нулевым, перед
   заполнением обнуляем ее всю */
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(51000);
   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
   /* Создаем UDP сокет */
   if((sockfd = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
      perror(NULL); /* Печатаем сообщение об ошибке */
      exit(1);
   }
   /* Настраиваем адрес сокета */
```

```
    if(bind(sockfd, (struct sockaddr *) &servaddr,
    sizeof(servaddr)) < 0){
       perror(NULL);
       close(sockfd);
       exit(1);
    }
    while(1) {
       /* Основной цикл обслуживания*/
       /* В переменную clilen заносим максимальную длину
       для ожидаемого адреса клиента */
       clilen = sizeof(cliaddr);
       /* Ожидаем прихода запроса от клиента и читаем его.
       Максимальная допустимая длина датаграммы – 999
       символов, адрес отправителя помещаем в структуру
       cliaddr, его реальная длина будет занесена в
       переменную clilen */
       if((n = recvfrom(sockfd, line, 999, 0,
       (struct sockaddr *) &cliaddr, &clilen)) < 0){
          perror(NULL);
          close(sockfd);
          exit(1);
       }
       /* Печатаем принятый текст на экране */
       printf("%s\n", line);
       /* Принятый текст отправляем обратно по адресу
       отправителя */
       if(sendto(sockfd, line, strlen(line), 0,
       (struct sockaddr *) &cliaddr, clilen) < 0){
          perror(NULL);
          close(sockfd);
          exit(1);
       } /* Уходим ожидать новую датаграмму*/
    }
    return 0;
}
```

Дополнительные сведения

Проверка наличия компилятора - *which gcc*

Компиляция – gcc –o socket_server socket_server.c

Компиляция – gcc –o socket_client socket_client.c

Запуск скомпилированных программ для проверки работы осуществляется в разных консольных окнах:

./socket_server

./socket_client

Задание к лабораторной работе №3

Откомпилируйте программу клиента. Перед запуском уточнить, запущен ли в системе стандартный UDP-сервис echo и если нет, попросите

стартовать его. Запустите программу с запросом к сервису своего компьютера, к сервисам других компьютеров. Если в качестве IP-адреса указать несуществующий адрес, адрес выключенной машины или машины, на которой не работает сервис echo, то программа бесконечно блокируется в вызове recvfrom(), ожидая ответа. Протокол UDP не является надежным протоколом. Если датаграмму доставить по назначению не удалось, то отправитель никогда об этом не узнает.

Откомпилируйте программу UDP-сервера. Запустите ее на выполнение. Модифицируйте текст программы UDP-клиента, заменив номер порта с 7 на 51000. Запустите клиента с другого виртуального терминала или с другого компьютера и убедитесь, что клиент и сервер взаимодействуют корректно.

Каким образом можно осуществить одновременную обработку запросов клиентов сервером, допишите серверную часть программы, чтобы можно было осуществить одновременно несколько соединений с клиентами.

Требования к отчету по лабораторной работе №3

В отчете по лабораторной работе должны быть представлены:

1) Концепции взаимодействия программы-клиента и программы сервера, взаимодействующих по протоколу UDP.
2) Листинг программ реализации клиентской и серверной частей.
3) Экранные формы, демонстрирующие работу программ.

Контрольные вопросы:

1) На каком уровне стека протоколов TCP/IP находится протокол UDP?
2) Что такое сокет?
3) Что такое порт протокола?
4) В каких случаях при разработке клиент-серверных приложений в качестве транспортного протокола используют протокол UDP?
5) Охарактеризуйте достоинства и недостатки протокола UDP.
6) Есть ли возможность создать одновременно несколько соединений с программой сервером для приведенного в методических указаниях UDP-сервера.
7) Какие прикладные протоколы стека TCP/IP используют в качестве транспортного средства протокол UDP?

ЛАБОРАТОРНАЯ РАБОТА №4

Тема: Разработка простейших клиент-серверных программ с использованием TCP сокетов на базе ОС Linux.

Цель работы: Изучение принципов взаимодействия клиент-серверных программ с использованием TCP сокетов на базе OS Linux.

Методические указания к выполнению лабораторной работы

Протокол управления передачей (TCP) предназначен для использования в качестве надежного протокола общения между хост-компьютерами в коммуникационных компьютерных сетях с коммутацией пакетов, а также в системах, объединяющих такие сети.

TCP - это протокол обеспечения надежности прямых соединений, созданный для многоуровневой иерархии протоколов, поддерживающих межсетевые приложения. Протокол TCP обеспечивает надежность коммуникаций между парами процессов на хост-компьютерах, включенных в различные компьютерные коммуникационные сети, которые объединены в единую систему.

Протокол TCP взаимодействует с одной стороны с пользователем или прикладной программой, а с другой - с протоколом более низкого уровня, таким как протокол Internet.

Главной целью протокола TCP является обеспечение надежного, безопасного сервиса для логических цепей или соединений между парами процессов. Чтобы обеспечить такой сервис, основываясь на менее надежных коммуникациях Internet, система должна иметь возможности для работы в следующих областях:

- базовая передача данных
- достоверность
- управление потоком
- разделение каналов
- работа с соединениями
- приоритет и безопасность

Модель действия

Процесс пересылает данные, вызывая программу протокола TCP и передавая ей в качестве аргументов буферы с данными. Протокол TCP пакует данные из этих буферов в сегменты, а затем вызывает модуль Internet для передачи каждого сегмента на программу протокола TCP, являющуюся адресатом. Этот адресат в свою очередь помещает данные из сегмента в буферы получателя и затем оповещает своего клиента о прибытии предназначенных ему данных. Программы протокола TCP помещают в сегменты контрольную информацию, которая затем используется ими для проверки очередности передачи данных.

Модель Internet коммуникаций состоит в том, что с каждой программой протокола TCP связан модуль протокола Internet, обеспечивающий ей

интерфейс с локальной сетью. Данный модуль Internet помещает сегменты TCP в Internet датаграммы, а затем направляет их на другой Internet модуль или же промежуточный шлюз. Для передачи датаграммы по локальной сети она в свою очередь помещается в пакет соответствующего типа.

Коммутаторы пакетов могут осуществлять дальнейшую упаковку, фрагментацию или другие операции с тем, чтобы в локальной сети осуществить передачу пакетов по назначению на модуль Internet.

На шлюзах между локальными сетями датаграмма Internet освобождается от пакета локальной сети и исследуется с тем, чтобы определить, по какой сети она должна в дальнейшем идти. Затем Internet датаграмма упаковывается в пакет, соответствующий выбранной локальной сети, и посылается на следующий шлюз или же прямо к конечному получателю.

Шлюз имеет возможность разбивать Internet датаграмму на более мел кие датаграммы-фрагменты, если это необходимо для передачи по очередной локальной сети. Чтобы осуществить это, шлюз сам создает набор Internet датаграмм, помещая в каждую по одному фрагменты. В дальнейшем фрагменты могут быть снова разбиты следующими шлюзами на еще более мелкие части. Формат фрагмента Internet датаграммы спроектирован так, чтобы адресат - модуль Internet смог собрать фрагменты снова в исходные Internet датаграммы.

Internet модуль, являющийся адресатом, выделяет сегмент из датаграммы (после ее сборки в случае необходимости) и затем передает его по назначению на программу протокола TCP.

Надежные коммуникации

Поток данных, посылаемый на TCP соединение, принимается получателем надежно и в соответствующей очередности.

Передача осуществляется надежно благодаря использованию подтверждений и номеров очереди. Концептуально каждому октету данных присваивается номер очереди. Номер очереди для первого октета данных в сегменте передается вместе с этим сегментом и называется номером очереди для сегмента. Сегменты также несут номер подтверждения, который является номером для следующего ожидаемого октета данных, передаваемого в обратном направлении. Когда протокол TCP передает сегмент с данными, он помещает его копию в очередь повторной передачи и запускает таймер. Когда приходит подтверждение для этих данных, соответствующий сегмент удаляется из очереди. Если подтверждение не приходит до истечения срока, то сегмент посылается повторно.

Подтверждение протокола TCP не гарантирует, что данные достигли конечного получателя, а только то, что программа протокола TCP на компьютере у получателя берет на себя ответственность за это.

Для направления потока данных между программами протоколов TCP используется механизм управления потоками. Получающая программа протокола TCP сообщает "окно" посылающей программе. Данное окно

указывает количество октетов (начиная с номера подтверждения), которое принимающая программа TCP готова в настоящий момент принять.

Установка соединения и его отмена

Чтобы идентифицировать отдельные потоки данных, поддерживаемые протоколом TCP, последний определяет идентификаторы портов. Поскольку идентификаторы портов выбираются каждой программой протокола TCP независимо, то они не будут уникальны. Чтобы обеспечить уникальность адресов для каждой программы протокола TCP, мы объединяем идентифицирующий эту программу Internet адрес и идентификатор порта. В результате получаем сокет, который будет уникален во всех локальных сетях, объединенных в единое целое.

Соединение полностью определяется парой сокетов на своих концах. Локальный сокет может принимать участие во многих соединениях с различными чужими сокетами. Соединение можно использовать для передачи данных в обоих направлениях, иными словами, оно является "полностью дуплексным".

Протокол TCP волен произвольным образом связывать порты с процессами. Однако при любой реализации протокола необходимо придерживаться нескольких основополагающих концепций. Должны присутствовать общеизвестные сокеты, которые протокол TCP ассоциирует исключительно с "соответствующими им" процессами.

Соединение задается командой OPEN (открыть), сделанной с локального порта и имеющей аргументом чужой сокет. В ответ на такой запрос программа протокола TCP предоставляет имя локального (короткого) со единения. По этому имени пользователь адресуется к данному соединению при последующих вызовах. О соединениях следует помнить кое-какие вещи.

Запрос на пассивное открытие соединения означает, что процесс ждет получения извне запросов на соединение, вместо того, чтобы пытаться самому установить его. Часто процесс, сделавший запрос на пассивное открытие, будет принимать запросы на соединение от любого другого процесса. В этом случае чужой сокет указывается как состоящий целиком из нулей, что означает неопределенность. Неопределенные чужие сокеты могут присутствовать лишь в командах пассивного открытия.

Сервисный процесс, желающий обслужить другие, неизвестные ему процессы, мог бы осуществить запрос на пассивное открытие с указанием неопределенного сокета. В этом случае соединение может быть установлено с любым процессом, запросившим соединения с этим локальным сокетом. Такая процедура будет полезна, если известно, что выбранный локальный сокет ассоциирован с определенным сервисом.

Общеизвестные сокеты представляют собой удобный механизм априорного привязывания адреса сокета с каким-либо стандартным сервисом. Например, процесс "сервер для программы Telnet" жестко связан с конкретным сокетом. Другие сокеты могут быть зарезервированы за

передатчиком файлов, Remote Job Entry, текстовым генератором, эхо-сервером, а также Sink-процессами (последние три пункта связаны с обработкой текстов). Адрес сокета может быть зарезервирован для доступа к процедуре "просмотра", которая могла бы указывать сокет, через который можно было бы получить новообразованные услуги. Концепция общеизвестного сокета является частью TCP спецификации, однако собственно асоциирование сокетов с услугами выходит за рамки данного описания протокола.

Процессы могут осуществлять пассивные открытия соединений и ждать, пока от других процессов придут соответствующие запросы на активное открытие, а протокол TCP проинформирует их об установлении соединения. Два процесса, сделавшие друг другу одновременно запросы на активное открытие, получат корректное соединение. Гибкость такого подхода становится критичной при поддержке распределенных вычислений, когда компоненты системы взаимодействуют друг с другом асинхронным образом.

Когда осуществляется подбор сокетов для локального запроса пассивного открытия и чужого запроса на активное открытие, то принципиальное значение имеют два случая. В первом случае местное пассивное открытие полностью определяет чужой сокет. При этом подбор должен осуществляться очень аккуратно. Во втором случае во время местного пассивного открытия чужой сокет не указывается. Тогда в принципе может быть установлено соединение с любых чужих сокетов. Во всех остальных случаях подбор сокетов имеет частичные ограничения.

Процедура установки соединения использует флаг управления синхронизацией (SYN) и трижды обменивается сообщениями. Такой обмен называется трехвариантным подтверждением.

Соединение инициируется при встрече пришедшего сегмента, несущего флаг синхронизации (SYN), и ждущей его записи в блоке TCB. И сегмент и запись создаются пришедшими от пользователей запросами на открытие. Соответствие местного и чужого сокетов устанавливается при инициализации соединения. Соединение признается установленным, когда номера очередей синхронизированы в обоих направлениях между сокетами.

Отмена соединения также включает обмен сегментами, несущими на этот раз управляющий флаг FIN.

Коммуникация данных

Набор данных, передаваемых по соединению, можно рассматривать как поток октетов. Пользователь, отправляющий данные, указывает при запросе по посылку, следует ли данные, отправляемые при этом запросе, немедленно проталкивать через сеть к получателю. Указание осуществляется установкой флага PUSH (проталкивание).

Программа протокола TCP может собирать данные, принимаемые от пользователя, а затем передавать их в сеть по своему усмотрению в виде сегментов. Если же выставлен запрос на проталкивание, то протокол должен

передать все не отправленные ранее данные. Когда программа протокола TCP, принимающая данные, сталкивается с флагом проталкивания, ей не следует ожидать получения новых данных по сети до тех пор, пока уже имеющиеся данные не будут переданы ждущему их местному процессу.

Нет нужды привязывать функции проталкивания к границам сегмента. Данные, содержащиеся в каком-либо сегменте, могут быть результатом одного или нескольких запросов на посылку. Или же один запрос может породить несколько сегментов.

Целью функции проталкивания и флага PUSH является проталкивание данных через сеть от отправителя к получателю. Функция не осуществляет обработки самих данных.

Существует связь между функцией проталкивания и использованием буферов данных в интерфейсе между пользователем и протоколом TCP. Каждый раз, когда в буфер получателя приходят данные с флагом PUSH, содержимое этого буфера передается пользователю на обработку, даже если буфер и не был заполнен. Если приходящие данные заполняют буфер пользователя до того, как получена команда проталкивания, пользователю отправляется блок данных, соответствующий размеру буфера. Протокол TCP имеет также средства для сообщения получателю, что с некоторого момента он имеет дело со срочными данными. Протокол TCP не пытается определить, что именно пользователь делает со ждущими обработки срочными данными. Однако обычно предполагается, что получающий данные процесс будет предпринимать усилия для быстрой обработки срочных данных.

Приоритет и безопасность

Протокол TCP использует тип сервиса и опцию безопасности протокола Internet с тем, чтобы пользователям протокола TCP обеспечить приоритет и безопасность на каждом соединении. Не все модули протокола TCP обязательно будут действовать в многоуровневой системе обеспечения безопасности. Некоторые модули ограничиваются только обычными, неспецифическими соединениями, другие ограничиваются лишь первым уровнем безопасности и закрытости. Следовательно, некоторые реализации протокола TCP и услуг для пользователей могут использовать лишь часть многоуровневой системы безопасности.

Модули TCP, действующие в многоуровневой системе безопасности, должны адекватным образом выставлять в отсылаемых сегментах флаги безопасности и приоритета. Такие модули TCP должны также позволять своим клиентам или вышестоящим протоколам, таким как Telnet и THP, указывать требуемый уровень безопасности, закрытости и приоритета для устанавливаемых соединений.

Структура пакета TCP

Сегмент TCP состоит из TCP-заголовка, TCP-опций и данных, переносимых сегментом. На рис. 1 изображена структура сегмента TCP. Несмотря на то, что заголовок показан состоящим как бы из нескольких

уровней, на самом деле он является последовательным потоком данных, длиной как минимум в 20 байтов.

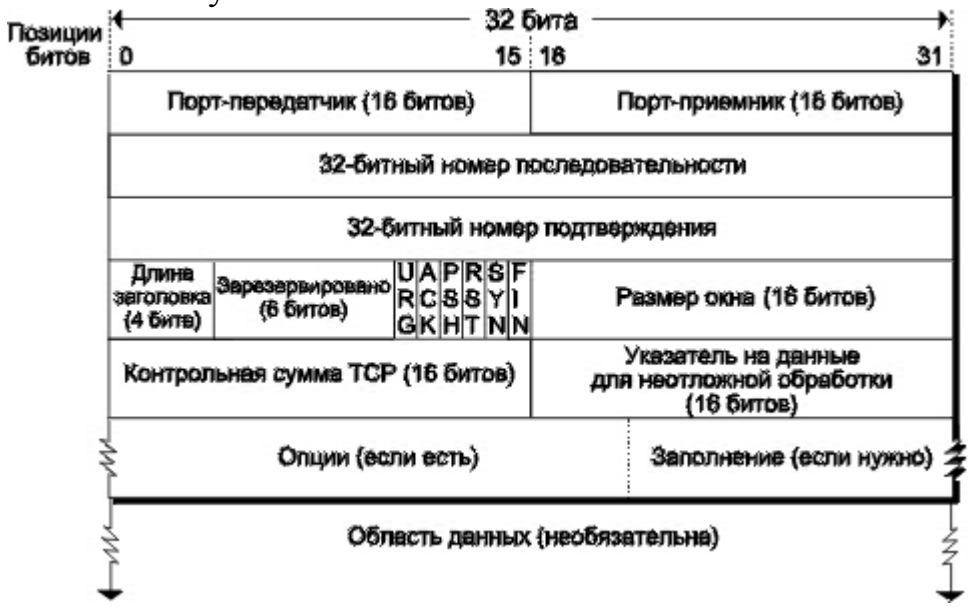


Рисунок 4.1 - Структура сегмента (сообщения) TCP

В табл. кратко описано назначение каждого поля заголовка TCP.

Поле заголовка	Назначение
Порт-передатчик	Обозначает порт протокола приложения-источника данных.
Порт-приемник	Обозначает порт протокола приложения-получателя данных.
Номер последовательности	Определяет первый байт данных в области данных сегмента TCP.
Номер подтверждения	Определяет следующий байт данных, который приемник рассчитывает получить из входного потока.
Длина заголовка	Длина TCP-заголовка, измеренная в 32-разрядных словах.
Флаг URG	Если установлен, извещает принимающий модуль TCP о том, что в сегменте находятся данные для неотложной обработки.
Флаг ACK	Указание принимающему модулю TCP на то, что поле номер подтверждения содержит соответствующие данные.
Флаг PSH	Требование принимающему модулю TCP передать данные приложению-получателю немедленно.
Флаг RST	Запрос принимающему модулю TCP сбросить соединение.
Флаг SYN	Запрос принимающему модулю TCP синхронизировать номера последовательности.
Флаг FIN	Сообщение принимающему модулю TCP об окончании передачи.
Размер окна	Сообщение принимающему модулю TCP о количестве байтов, которое способен принять модуль-передатчик.
Контрольная сумма TCP	Служит для обнаружения поврежденных при передаче данных.
Указатель на неотложные данные	Указывает на последний байт данных, требующих неотложной обработки, находящихся в области данных сегмента TCP.
Опции	Обычно используются совместно с опцией максимальная длина сегмента (MSS).

Схема взаимодействия клиента и сервера для протокола TCP

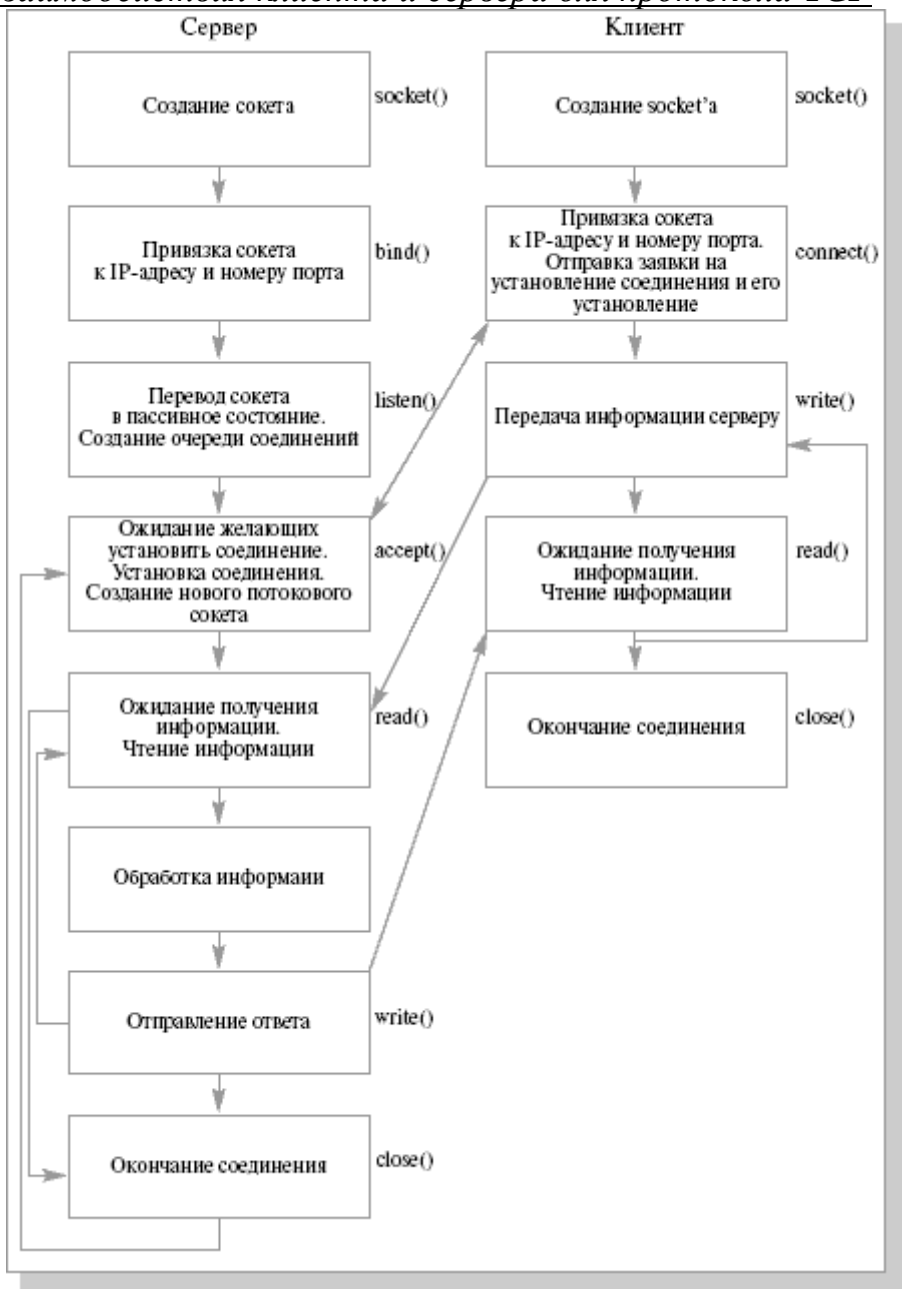


Рисунок 4.2 - Схема взаимодействия клиента и сервера для протокола TCP (последовательная обработка запросов на TCP-сервере)

Для создания используется системный вызов *socket()*.

Для привязки сервера к IP-адресу и номеру порта, как и в случае UDP-протокола, используется системный вызов *bind()*.

Для процесса клиента эта привязка объединена с процессом установления соединения с сервером в новом системном вызове *connect()* и скрыта от глаз пользователя. Внутри этого вызова операционная система осуществляет настройку сокета на выбранный ею порт и на адрес любого сетевого интерфейса.

Для перевода сокета на сервере в пассивное состояние и для создания очереди соединений служит системный вызов *listen()*.

Сервер ожидает соединения и получает информацию об адресе соединившегося с ним клиента с помощью си стемного вызова *accept()*. Поскольку установленное логическое соединение выглядит со стороны процессов как канал связи, позволяющий обмениваться данными с помощью потоковой модели, для передачи и чтения информации оба системных вызова используют уже известные нам системные вызовы *read() и write(),* а для завершения соединения – системный вызов *close().*

Необходимо отметить, что при работе с сокетами вызовы *read()* и *write()* обладают теми же особенностями поведения, что и при работе с pip'ами и FIFO.

Системный вызов connect()

Прототип системного вызова

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockd,
   struct sockaddr *servaddr,
   int addrlen);
```

Параметр sockd является дескриптором созданного ранее коммуникационного узла, т. е. значением, которое вернул системный вызов socket().

Параметр servaddr представляет собой адрес структуры, содержащей информацию о полном адресе сокета сервера. *Он имеет тип указателя на структуру-шаблон struct sockaddr, которая должна быть конкретизирована в зависимости от используемого семейства протоколов и заполнена перед вызовом.*

Параметр addrlen должен содержать фактическую длину структуры, адрес которой передается в качестве второго параметра. *Эта длина меняется в зависмости от семейства протоколов и различается даже в пределах одного семейства протоколов (например, для UNIX Domain).*

При установлении виртуального соединения системный вызов не возвращается до его установления или до истечения установленного в системе времени – timeout. При использовании его в connectionless связи вызов возвращается немедленно.

Возвращаемое значение:

Системный вызов возвращает значение 0 при нормальном завершении и отрицательное значение, если в процессе его выполнения возникла ошибка.

Пример программы TCP-клиента

Рассмотрим пример – программу.

Это простой TCP-клиент, обращающийся к стандартному системному сервису echo. Стандартный сервис принимает от клиента текстовую

датаграмму и, не изменяя ее, отправляет обратно. За сервисом зарезервирован номер порта 7.

Для того чтобы подчеркнуть, что после установления логического соединения клиент и сервер могут обмениваться информацией неоднократно, клиент трижды запрашивает текст с экрана, отсылает его серверу и печатает полученный ответ. Ниже представлен текст программы.

```
/* Простой пример TCP-клиента для сервиса echo */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
void main(int argc, char **argv)
{
   int sockfd; /* Дескриптор сокета */
   int n; /* Количество переданных или прочитанных
      символов */
   int i; /* Счетчик цикла */
   char sendline[1000],recvline[1000]; /* Массивы
      для отсылаемой и принятой строки */
   struct sockaddr_in servaddr; /* Структура для
      адреса сервера */
   /* Сначала проверяем наличие второго аргумента в
   командной строке. При его отсутствии прекращаем
   работу */
   if(argc != 2){
      printf("Usage: a.out <IP address>\n");
      exit(1);
   }
   /* Обнуляем символьные массивы */
   bzero(sendline,1000);
   bzero(recvline,1000);
   /* Создаем TCP сокет */
   if((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0){
      perror(NULL); /* Печатаем сообщение об ошибке */
      exit(1);
   }
   /* Заполняем структуру для адреса сервера: семейство
   протоколов TCP/IP, сетевой интерфейс – из аргумента
   командной строки, номер порта 7. Поскольку в структуре
   содержится дополнительное не нужное нам поле,
   которое должно быть нулевым, перед заполнением обнуляем
   ее всю */
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(51000);
   if(inet_aton(argv[1], &servaddr.sin_addr) == 0){
```

```
        printf("Invalid IP address\n");
        close(sockfd);
        exit(1);
    }
    /* Устанавливаем логическое соединение через
созданный сокет с сокетом сервера, адрес которого мы занесли
в структуру */
    if(connect(sockfd, (struct sockaddr *) &servaddr,
    sizeof(servaddr)) < 0){
        perror(NULL);
        close(sockfd);
        exit(1);
    }
    /* Три раза в цикле вводим строку с клавиатуры, отправляем
    ее серверу и читаем полученный ответ */
    for(i=0; i<3; i++){
        printf("String => ");
        fflush(stdin);
        fgets(sendline, 1000, stdin);
        if( (n = write(sockfd, sendline,
        strlen(sendline)+1)) < 0){
            perror("Can\'t write\n");
            close(sockfd);
            exit(1);
        }
        if ( (n = read(sockfd,recvline, 999)) < 0){
            perror("Can\'t read\n");
            close(sockfd);
            exit(1);
        }
        printf("%s", recvline);
    }
    /* Завершаем соединение */
    close(sockfd);
}
```

Прежде чем рассмотреть пример работы TCP-сервера, рассмотрим некоторые системные вызовы.

Системный вызов listen()

Системный вызов *listen()* является первым из еще неизвестных нам вызовов, применяемым на TCP–сервере. В его задачу входит перевод TCP–сокета в пассивное (слушающее) состояние и создание очередей для порождаемых при установлении соединения присоединенных сокетов, находящихся в состоянии не полностью установленного соединения и полностью установленного соединения. Для этого вызов имеет два параметра: дескриптор TCP–сокета и число, определяющее глубину создаваемых очередей.

Прототип системного вызова

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
int listen(int sockd, int backlog);
```

Параметр sockd является дескриптором созданного ранее сокета, который должен быть переведен в пассивный режим, т. е. значением, которое вернул системный вызов *socket()*. Системный вызов *listen* требует предварительной настройки адреса сокета с помощью системного вызова *bind()*.

Параметр backlog определяет максимальный размер очередей для сокетов, находящихся в состояниях полностью и не полностью установленных соединений.

Возвращаемое значение

Системный вызов возвращает значение 0 при нормальном завершении и значение -1 при возникновении ошибки.

Системный вызов accept()

Прототип системного вызова

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockd,
   struct sockaddr *cliaddr,
   int *clilen);
```

Описание системного вызова

Системный вызов *accept* используется сервером, ориентированным на установление связи путем виртуального соединения, для приема полностью установленного соединения.

Параметр sockd является дескриптором созданного и настроенного сокета, предварительного переведенного в пассивный (слушающий) режим с помощью системного вызова *listen()*.

Системный вызов *accept* требует предварительной настройки адреса сокета с помощью системного вызова *bind()*.

Параметр cliaddr служит для получения адреса клиента, установившего логическое соединение, и должен содержать указатель на структуру, в которую будет занесен этот адрес.

Параметр clilen содержит указатель на целую переменную, которая после возвращения из вызова будет содержать фактическую длину адреса клиента. Заметим, что перед вызовом эта переменная должна содержать максимально допустимое значение такой длины. Если параметр cliaddr имеет значение NULL, то и параметр clilen может иметь значение NULL.

Возвращаемое значение

Системный вызов возвращает при нормальном завершении дескриптор присоединенного сокета, созданного при установлении соединения для последующего общения клиента и сервера, и значение -1 при возникновении ошибки.

Системные вызовы read и write

Прототипы системных вызовов

```
#include <sys/types.h>
#include <unistd.h>
size_t read(int fd, void *addr,
          size_t nbytes);
size_t write(int fd, void *addr,
          size_t nbytes);
```

Описание системных вызовов

Системные вызовы *read* и *write* предназначены для осуществления потоковых операций ввода (чтения) и вывода (записи) информации над каналами связи, описываемыми файловыми дескрипторами, т.е. для файлов, pipe, FIFO и socket.

Параметр fd является файловым дескриптором созданного ранее потокового канала связи, через который будет отсылаться или получаться информация, т. е. значением, которое вернул один из системных вызовов *open(), pipe()* или *socket().*

Параметр addr представляет собой адрес области памяти, начиная с которого будет браться информация для передачи или размещаться принятая информация.

Параметр nbytes для системного вызова write определяет количество байт, которое должно быть передано, начиная с адреса памяти addr. Параметр nbytes для системного вызова read определяет количество байт, которое мы хотим получить из канала связи и разместить в памяти, начиная с адреса addr.

Возвращаемые значения

В случае успешного завершения системный вызов возвращает количество реально посланных или принятых байт. Заметим, что это значение (большее или равное 0) может не совпадать с заданным значением параметра nbytes, а быть меньше, чем оно, в силу отсутствия места на диске или в линии связи при передаче данных или отсутствия информации при ее приеме. При возникновении какой-либо ошибки возвращается отрицательное значение.

Системный вызов close

Прототип системного вызова

```
#include <unistd.h>
int close(int fd);
```

Описание системного вызова

Системный вызов *close* предназначен для корректного завершения работы с файлами и другими объектами ввода-вывода, которые описываются в операционной системе через файловые дескрипторы: pipe, FIFO, socket.

Параметр fd является дескриптором соответствующего объекта, т. е. значением, которое вернул один из системных вызовов *open(), pipe()* или *socket()*.

Возвращаемые значения

Системный вызов возвращает значение 0 при нормальном завершении и значение -1 при возникновении ошибки.

Пример простого TCP-сервера

```
/* Пример простого TCP-сервера для сервиса echo */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
void main()
{
   int sockfd, newsockfd; /* Дескрипторы для
слушающего и присоединенного сокетов */
   int clilen; /* Длина адреса клиента */
   int n; /* Количество принятых символов */
   char line[1000]; /* Буфер для приема информации */
   struct sockaddr_in servaddr, cliaddr; /* Структуры
      для размещения полных адресов сервера и
      клиента */
   /* Создаем TCP-сокет */
   if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
      perror(NULL);
      exit(1);
   }
   /* Заполняем структуру для адреса сервера: семейство
   протоколов TCP/IP, сетевой интерфейс – любой, номер
   порта 51000. Поскольку в структуре содержится
   дополнительное не нужное нам поле, которое должно
   быть нулевым, побнуляем ее всю перед заполнением */
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family= AF_INET;
   servaddr.sin_port= htons(51000);
   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
   /* Настраиваем адрес сокета */
   if(bind(sockfd, (struct sockaddr *) &servaddr,
   sizeof(servaddr)) < 0){
      perror(NULL);
      close(sockfd);
      exit(1);
   }
   /* Переводим созданный сокет в пассивное (слушающее)
   состояние. Глубину очереди для установленных
```

```
соединений описываем значением 5 */
if(listen(sockfd, 5) < 0){
    perror(NULL);
    close(sockfd);
    exit(1);
}
/* Основной цикл сервера */
while(1){
    /* В переменную clilen заносим максимальную
    длину ожидаемого адреса клиента */
    clilen = sizeof(cliaddr);
    /* Ожидаем полностью установленного соединения
    на слушающем сокете. При нормальном завершении
    у нас в структуре cliaddr будет лежать полный
    адрес клиента, установившего соединение, а в
    переменной clilen – его фактическая длина. Вызов
    же вернет дескриптор присоединенного сокета, через
    который будет происходить общение с клиентом.
    Заметим, что информация о клиенте у нас в
    дальнейшем никак не используется, поэтому
    вместо второго и третьего параметров можно
    было поставить значения NULL. */
    if((newsockfd = accept(sockfd,
    (struct sockaddr *) &cliaddr, &clilen)) < 0){
        perror(NULL);
        close(sockfd);
        exit(1);
    }
    /* В цикле принимаем информацию от клиента до
    тех пор, пока не произойдет ошибки (вызов read()
    вернет отрицательное значение) или клиент не
    закроет соединение (вызов read() вернет
    значение 0). Максимальную длину одной порции
    данных от клиента ограничим 999 символами. В
    операциях чтения и записи пользуемся дескриптором
    присоединенного сокета, т. е. значением, которое
    вернул вызов accept().*/
    while((n = read(newsockfd, line, 999)) > 0){
        /* Принятые данные отправляем обратно */
        if((n = write(newsockfd, line,
        strlen(line)+1)) < 0){
            perror(NULL);
            close(sockfd);
            close(newsockfd);
            exit(1);
        }
    }
/* Если при чтении возникла ошибка – завершаем работу */
if(n < 0){
        perror(NULL);
        close(sockfd);
        close(newsockfd);
```

```
            exit(1);
        }
        /* Закрываем дескриптор присоединенного сокета и
        уходим ожидать нового соединения */
        close(newsockfd);
    }
}
```

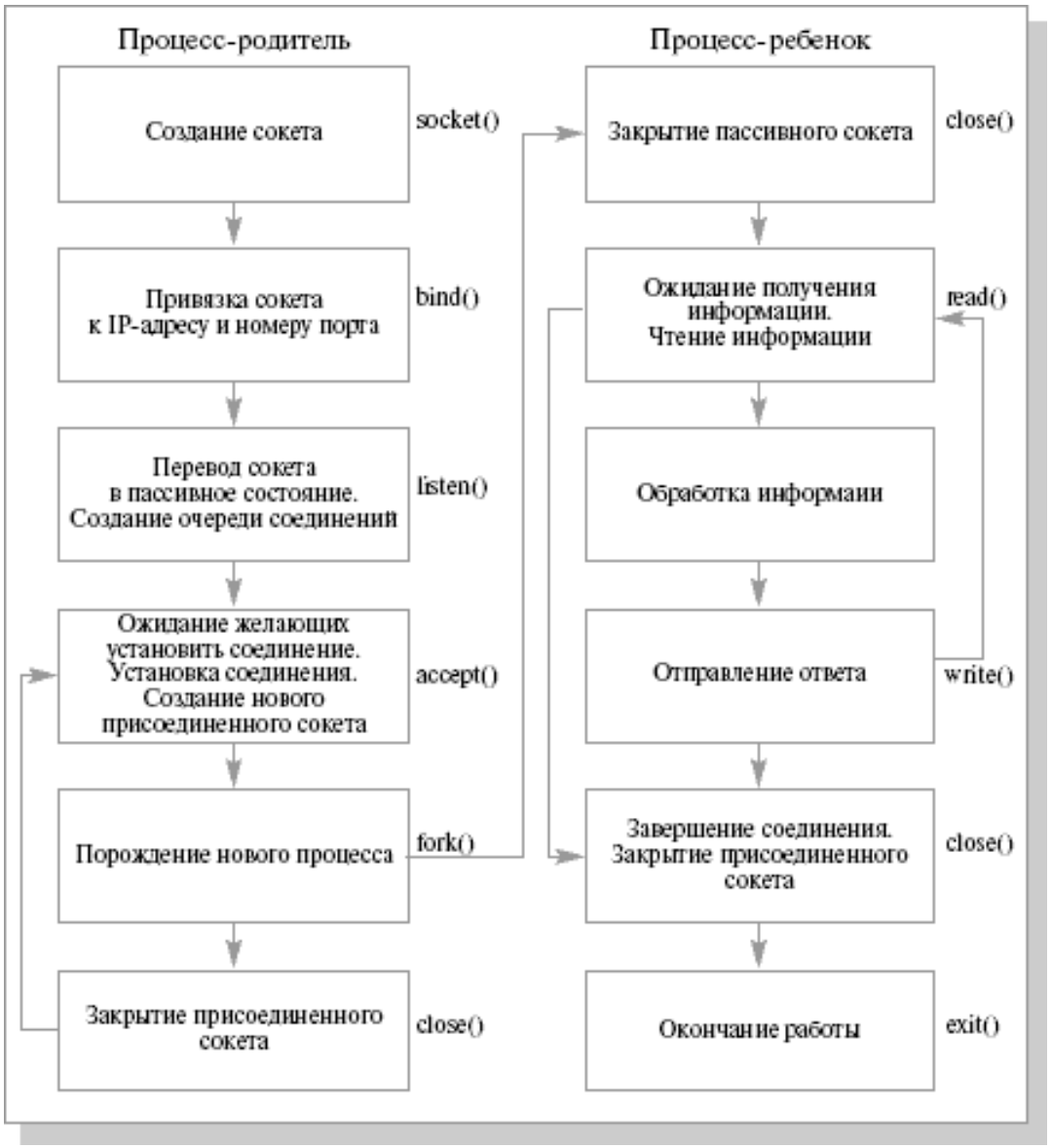


Рисунок 4.3 - Схема работы TCP-сервера с параллельной обработкой запросов

Дополнительные сведения

Проверка наличия компилятора - *which gcc*

Компиляция – gcc –o socket_server socket_server.c

Компиляция – gcc –o socket_client socket_client.c

Запуск скомпилированных программ для проверки работы осуществляется в разных консольных окнах:

./socket_server

./socket_client

Задание к лабораторной работе №4

Откомпилируйте программы простейших TCP-клиента и TCP-сервера (последовательная обработка запросов TCP-сервером). Убедитесь в правильном функционировании этих программ.

Изучите системные вызовы, которые используются в серверной и клиентской частях.

Напишите программу взаимодействия клиента и сервера на базе TCP соединения, используя параллельную обработку запросов TCP-сервером, ограничьте количество соединений (не более указанного числа) и время ожидания сервером запроса от клиента при уже установленном соединении.

Требования к отчету по лабораторной работе

В отчете по лабораторной работе должны быть представлены:

1) Концепции взаимодействия программы-клиента и программы сервера, взаимодействующих по протоколу TCP (последовательная обработка запросов сервером, параллельная обработка запросов сервером).
2) Листинг программ реализации клиентской и серверной частей.
3) Экранные формы, демонстрирующие работу программ.

Контрольные вопросы:

1) На каком уровне стека протоколов TCP/IP находится протокол TCP?
2) Что такое сокет?
3) Что такое порт протокола?
4) В каких случаях при разработке клиент-серверных приложений в качестве транспортного протокола используют протокол TCP?
5) Охарактеризуйте достоинства и недостатки протокола TCP.
6) Есть ли возможность создать одновременно несколько соединений с программой сервером для приведенного в методических указаниях TCP-сервера.
7) Какие прикладные протоколы стека TCP/IP используют в качестве транспортного средства протокол TCP?
8) Какие проблемы могут возникнуть при функционировании клиент-серверного программного обеспечения, использующие протокол TCP в качестве транспортного средства. Почему?

ЛАБОРАТОРНАЯ РАБОТА №5

Тема: Использование протокола VNC при разработке систем удаленного администрирования.

Цель: Ознакомиться с теоретическиеми вопросами, связанными с разработкой систем удаленного администрирования и реализовать простейшую клиент-серверную программу удаленного администрирования с использованием протокола VNC.

Методические указания к выполнению лабораторной работы

С развитием телекоммуникационных технологий и Интернет появляется все больше возможностей для обширного применения удаленного доступа. На сегодняшний день различное географическое расположение филиалов и отделений разнообразных компаний является нормальным явлением. Соответственно качественная и постоянная связь с этими отделенными филиалами, которые основываются на удаленных серверах и рабочий станциях, является обязательным требованием к сетевой инфраструктуре компании. И, как следствие, в задачи системного администратора входит обязанность поддержки этих географически удалённых серверов и рабочих станций. Используя современные технологии, стало возможным запуск различных приложений на удаленном компьютере с другого удаленного компьютера.

Существует множество разнообразных средств удаленного администрирования, разработанные под различные операционные системы. Также существуют встроенные средства для удаленного администрирования, такие как Telnet в Unix подобных системах, или «Удаленный рабочий стол» в Windows NT. В Windows XP такие встроенные возможности, поэтому стоит цель разработать программный продукт позволяющий производить удаленное администрирование.

Удаленный доступ — широкое понятие, которое включает в себя различные типы и варианты взаимодействия компьютеров, сетей и приложений.

Для удаленного доступа, как правило, характерна несимметричность взаимодействия, то есть с одной стороны имеется центральная крупная сеть или центральный компьютер, а с другой — отдельный удаленный терминал, компьютер или небольшая сеть, которые должны получить доступ к информационным ресурсам центральной сети.

В условиях современных распределенных информационных систем, учитывая рост затрат на их сопровождение и поддержку, заказчики все чаще применяют программные средства удаленного администрирования.

Самым известным программным обеспечением удаленного администрирования является Radmin и RealVNC .

Функциональные возможности систем удаленного администрирования

- запуск как служба;

- защита передаваемых данных;
- передача файлов;
- telnet-сервер;
- NT разрешения на доступ;
- защита паролем;
- передача буфера обмена;
- IP-фильтр;
- кроссплатформенность;
- Web - управление;
- стандарт;
- шифрование трафика;

Анализ достоинств и недостатков существующих средств для удаленного администрирования

Семейство UNIX. UNIX можно назвать операционной системой, хорошо приспособленной для задач системного и сетевого администрирования, но гораздо хуже — для офисных приложений. Поскольку речь идет о системе удаленного администрирования, а не о настольной системе, можно сказать, что благодаря сервисам telnet любой имеющий на то право пользователь может управлять сетью из любой точки земного шара, запустив на своем компьютере удаленный терминал. Единственный серьезный недостаток такого подхода — высокие требования к квалификации администратора: он должен хорошо владеть утилитами командной строки. Естественно, у неопытных администраторов возникают большие сложности, а рядовые пользователи просто паникуют при виде черного экрана с мигающим курсором командной строки.

В последнее время эта ситуация меняется в лучшую сторону — появляются клиент-серверные приложения, позволяющие удаленно администрировать UNIX/Linux-системы в графическом режиме. Примером может служить VNC Server для Suse Linux.

Семейство Windows. Сложность удаленного администрирования сервера Windows NT всегда удручала системных администраторов, сталкивавшихся с этой задачей. И хотя наиболее опытные освоили такие трюки, как использование RCMD (Remote Command Service, RCMD.EXE) в сочетании с программами regini или regedit, все равно удаленное администрирование Windows NT существенно отличается от своего локального аналога. В этом случае необходимо освоение специального инструментария, поскольку операционные системы персональных компьютеров всегда были тесно привязаны к локальным клавиатуре и дисплею. Этот пробел восполняется рядом продуктов третьих фирм-разработчиков.

Для решения таких задач есть достаточно большое количество программного обеспечения. Самыми известным программным обеспечением такого рода является Radmin и RealVNC, которые позволяют решать эти

проблемы. Для такого программного обеспечения характерны следующие функциональные возможности: запуск как служба, защита передаваемых данных, передача файлов, telnet сервер, NT разрешения на доступ, защита паролем, IP фильтр, передача буфера обмена, кроссплатформенность, Web - управление, стандарт, шифрование трафика.

Было рассмотрено и проанализировано несколько сравнительных характеристик программных продуктов. Результаты анализа представлены в таблице 5.1

Таблица 5.1 – Анализ возможностей существующих систем удаленного администрирования

Программа	Протокол	Лицензия	Клиент	Сервер	Шифрование	Передача файлов	Несколько сеансов	Linux	Mac OS	MS Windows
Apple Remote Desktop	VNC	Собственные	+	+	AES - 128	+	+	-	+	-
Ericom PowerTerm WebConnect	RDP, VNC	Собственные	+	+	SSL, TLS	+	+	+	-	+
RealVNC Free	VNC	GPL	+	+	-	-	-	-	-	+
RealVNC Privat	VNC	Собственные	+	+	AES - 128	+	+	-	-	+
RealVNC предпринимательства	VNC	Собственные	+	+	AES - 128	+	+	+	+	+
Radmin 2.2	VNC	Собственные	+	+	+	+	+	+	-	+

Недостатки рассматриваемых программных продуктов:

- дороговизна лицензии на программные продукты;
- функциональная ограниченность пробных версий.

Протокол VNC

RFB-клиентом или вьювером, называют удаленный компьютер, за которым находится пользователь. RFB-сервером, называют удаленный компьютер, с которого происходит удаленный доступ и с которого передаются framebuffer’ы.

Display протокол (протокол серверной части) базируется на простом графическом примитиве: «поместить прямоугольник, состоящий из пикселей, в позицию с координатами X,Y». Обновление этих прямоугольников

происходит при помощи запросов от клиента. То есть информация с изменениями пересылается только в одном направлении. Также изменения могут прогнозироваться на сервере, что в значительной мере сокращает трафик.

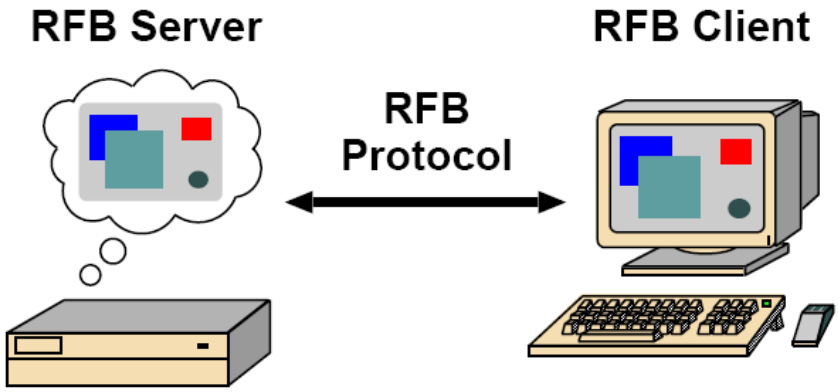


Рисунок 5.1 – Взаимодействие по протоколу RFB

Input протокол – это ни что иное, как клиентская часть протокола. В ней формируются сообщения, в которых указаны нажатие и отпускание клавиш, передвижения мышкой и нажатие ее клавиш.

Протокол может работать, как и со сплошным потоком, байт так и с сообщениями. На транспортном уровне использует протокол TCP/IP. Сообщения, посылаемые протоколом, можно разбить на три группы.

Первая - это группа сообщений, используемых при стадии «рукопожатия». В этой стадии устанавливается соединение, согласовываются версии протокола и системы безопасности

Стадия начинается с того, что сервер посылает клиенту сообщение ProtocolVersion, в котором указывается наибольшая и наименьшая версии протокола.

No. of bytes	Value
12	"RFB 003.003\n" (hex 52 46 42 20 30 30 33 2e 30 30 33 0a)

or

No. of bytes	Value
12	"RFB 003.007\n" (hex 52 46 42 20 30 30 33 2e 30 30 37 0a)

or

No. of bytes	Value
12	"RFB 003.008\n" (hex 52 46 42 20 30 30 33 2e 30 30 38 0a)

Рисунок 5.2 - сообщение ProtocolVersion

После сервер отсылает сообщение с типами безопасности, которые он поддерживает.

No. of bytes	Type [Value]	Description
1	U8	*number-of-security-types*
number-of-security-types	U8 array	*security-types*

Рисунок 5.3 - Сообщение с типами безопасности

Если сервер прислал хотя бы один тип безопасности, поддерживаемый клиентом, то клиент отсылает единственный байт назад, указывающий, какой тип безопасности должен использоваться при подключении.

No. of bytes	Type [Value]	Description
1	U8	*security-type*

Рисунок 5.4 - Тип безопасности

Если этот байт является нулем, то сервер закрывает соединение и отсылает клиенту сообщение с указанием причины.

No. of bytes	Type [Value]	Description
4	U32	*reason-length*
reason-length	U8 array	*reason-string*

Рисунок 5.5 - Сообщение с указанием причины

Последним сообщением стадии является сообщение SecurityResult, в котором сервер указывает, что соединение было установлено.

No. of bytes	Type [Value]	Description
4	U32	*status*:
	0	***OK***
	1	***failed***

Рисунок 5.6 - сообщение SecurityResult

В случае успеха стадии «рукопожатия» протокол переходит к стадии инициализации.

В этой фазе клиент и сервер обмениваются сообщениями ClientInit и ServerInit, в которых указываются параметры, используемые при дальнейшей работе сервера и клиента.

В сообщение ClientInit указывается, возможны ли подключения к серверу других клиентов, либо сервер должен дать монопольный доступ клиенту

No. of bytes	Type [Value]	Description
1	U8	*shared-flag*

Рисунок 5.7 - сообщение ClientInit

No. of bytes	Type [Value]	Description
2	U16	*framebuffer-width*
2	U16	*framebuffer-height*
16	PIXEL_FORMAT	*server-pixel-format*
4	U32	*name-length*
name-length	U8 array	*name-string*

Рисунок 5.8 - Сообщение ServerInit

No. of bytes	Type [Value]	Description
1	U8	*bits-per-pixel*
1	U8	*depth*
1	U8	*big-endian-flag*
1	U8	*true-colour-flag*
2	U16	*red-max*
2	U16	*green-max*
2	U16	*blue-max*
1	U8	*red-shift*
1	U8	*green-shift*
1	U8	*blue-shift*
3		*padding*

Рисунок 5.9 - PIXEL_FORMAT

Указание клиенту размера (ширины и высоты) framebuffer’a сервера, формат пикселя и названия компьютера - все это находится в сообщении ServerInit.

После обмена этими сообщениями фаза инициализации завершается. И протокол переходит нормальную рабочую стадию.

В нормальной стадии клиент посылает серверу следующие сообщения.

Сообщения клиент-сервер:

- SetPixelFormats в этом сообщении указывается формат пикселей, в которых должны быть переданы данные обновления framebuffer’a;

No. of bytes	Type	[Value]	Description
1	U8	0	*message-type*
3			*padding*
16	PIXEL_FORMAT		*pixel-format*

Рисунок 5.10 – SetPixelFormats

- SetEncoding - установка кодировки части framebuffer'a;

No. of bytes	Type	[Value]	Description
1	U8	2	*message-type*
1			*padding*
2	U16		*number-of-encodings*

Рисунок 5.11 – SetEncoding

- FrameBufferUpdateRequest - информирует сервер, что клиент интересуется областью framebuffer'a, с определенной позицией x,y, шириной и высотой;

No. of bytes	Type	[Value]	Description
1	U8	3	*message-type*
1	U8		*incremental*
2	U16		*x-position*
2	U16		*y-position*
2	U16		*width*
2	U16		*height*

Рисунок 5.12 - FrameBufferUpdateRequest

- KeyEvent - сообщение серверу о нажатие или отжатии клавиши; Используется ACII – кодировка.

No. of bytes	Type	[Value]	Description
1	U8	4	*message-type*
1	U8		*down-flag*
2			*padding*
4	U32		*key*

Рисунок 5.13 – KeyEvent

- PointerEvent – сообщает серверу о перемещении, нажатии или отпускании клавиши мыши.

No. of bytes	Type	[Value]	Description
1	U8	5	*message-type*
1	U8		*button-mask*
2	U16		*x-position*
2	U16		*y-position*

Рисунок 5.14 – PointerEvent

Сообщения сервер-клиент:

- FramebufferUpdate – сообщение, в котором пересылает обновляемая область framebuffer'a. Отсылается в ответ сообщение клиента FrameBufferUpdateRequest;

No. of bytes	Type	[Value]	Description
1	U8	0	*message-type*
1			*padding*
2	U16		*number-of-rectangles*

Рисунок 5.15 - FramebufferUpdate

В сообщении FramebufferUpdate указывается только номер передаваемого прямоугольника.

Сама прямоугольная область описывается следующей структурой:

No. of bytes	Type	[Value]	Description
2	U16		*x-position*
2	U16		*y-position*
2	U16		*width*
2	U16		*height*
4	S32		*encoding-type*

Рисунок 5.16 – Описание прямоугольной области

- SetColourMapEntries – установка и передача используемой палитры.

No. of bytes	Type	[Value]	Description
1	U8	1	*message-type*
1			*padding*
2	U16		*first-colour*
2	U16		*number-of-colours*

Рисунок 5.17 – SetColourMapEntries

Совокупность этих сообщений позволяет сполна реализовать все необходимые функции для удаленного доступа пользователю на сервер.

Одной из проблем использования VNC является обеспечение безопасности соединения. При необходимости можно использовать аутентификацию VNC. Если же аутентификация VNC используется, сервер

отправляет случайный 16-байтовый запрос клиенту, клиент шифрует запрос с помощью алгоритма DES, используя пароль, установленный пользователем как ключ, и отправляет 16-байтовую реакцию серверу. Это позволяет избежать нежелательные подключения. Возможны еще некоторые виды безопасности, например, Tight, Ultra, TSL и т.п. При необходимости возможно расширение протокола своими модулями. Например, для большей безопасности можно реализовать шифрование пароля каким-либо криптоустойчивым методом шифрования. Также VNC-соединение может быть установлено через SSH или VPN-туннель, что даст дополнительный уровень защиты с более мощным методом шифрования.

Еще одним узким местом для VNC является экономия трафика. Для этого в VNC реализованы несколько видов формата представление пикселей. Это 24, 16 и 8 бит на пиксель. 16 и 8-битное преставление пикселя задается при помощи карты цветов. Также для экономии трафика возможны различные виды кодирования. Такие, например, как CopyRect, RRE, Hextile, ZRLE, Cursor pseudo-encoding, DesctopSize pseudo-encoding. Без указания типа кодирования, по умолчанию, применяется метод Raw. Эти методы кодировки являются достаточно хорошими, что позволяет сильно сжать передаваемый трафик от сервера к клиенту.

Проектирование серверной части системы удаленного администрирования.

Так как для реализации был выбран протокол VNC, использующий на транспортном уровне стек протоколов TCP/IP, то всё проектирование серверной части будет зависеть от особенностей этих протоколов. Общая схема - проект серверной части показана на рисунке 5.18.

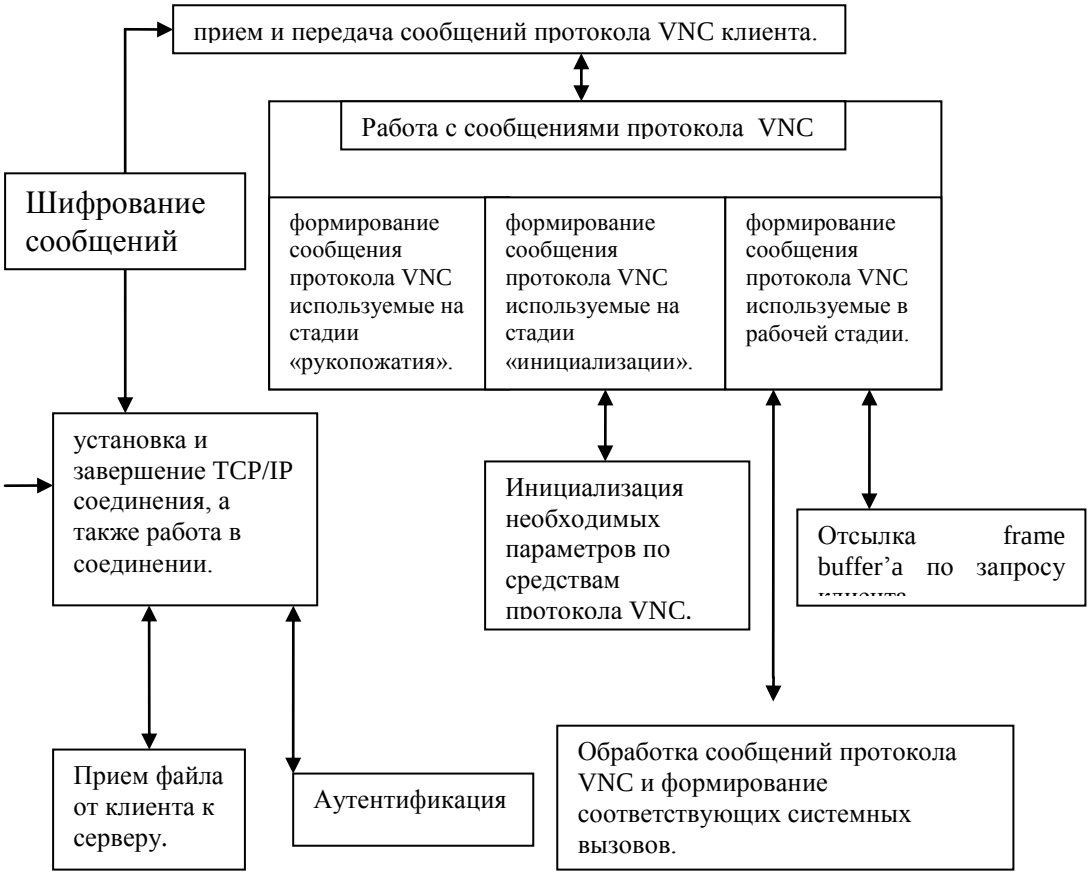


Рисунок 5.18 – Возможная схема работы серверной части системы удаленного администрирования

Каждый отдельный блок будет представлен в виде функций, либо процедур. Для ввода информации из внешнего мира служит получение сообщений из сети от клиентской части системы, либо реакция действия пользователя. Получением и отправкой сообщений от клиента и обратно будет осуществляться по средствам модуля «Установка и завершение TCP/IP соединения, а также работа в соединении». При получении этим модулем сообщения протокола VNC, полученное сообщение передается дальше модуль работы с сообщениями протокола VNC («Прием и передача сообщений протокола VNC»). В этом модуле определяется тип сообщения и передается в соответствующую часть модуля «Работа с сообщениями протокола VNC», где сообщение обрабатывается.

Из модуля «Работа с сообщениями протокола VNC» сообщение могут пойти по трем ветвям. Если полученное сообщение используется на стадии «Рукопожатия» протокола VNC, то оно передается на ветвь с названием «Формирование сообщения протокола VNC используемые на стадии «рукопожатия»». В этой ветки происходит прием, обработка и передача сообщений стадии «Рукопожатия», а так же выполнение инструкций соответствующий протокола VNC.

Если полученное сообщение используется на стадии «Инициализации» протокола VNC, то происходит прием всех необходимых

параметров. После приема вызывается модуль «Инициализация необходимых параметров по средствам протокола VNC», в котором инициализируется такие параметры как: тип сжатия передаваемых framebuffer’ов, количество бит на пиксель.

Если полученное сообщение используется в «Рабочей» стадии, то в соответствии с полученным сообщением, либо вызывается модуль «Отсылка ramebuffer’а по запросу клиента», либо «Обработка сообщений протокола VNC и формирование соответствующих системных вызовов». В первом происходит передача framebuffer’а клиентскому приложению. Во втором происходит анализ принятого сообщения и происходит системный вызов, в котором указывается изменение положения и нажатие кнопок манипулятора, или какая клавиша клавиатуры была нажата или отпущена.

Для большей безопасности предусмотренная аутентификация на основе логина и пароля. За аутентификацию отвечает модуль «Аутентификация». При указании соответствующих параметров можно использовать шифрование/расшифрование сообщений протокола VNC. За это отвечает модуль «Шифрование сообщений». Шифрование основано на модифицированной сети Фейстеля, использующей ключ, вычисляемый из логина и пароля.

Также спроектированная дополнительная возможность приема файлов от клиента к серверу. За прием файла отвечает модуль «Прием файла от клиента к серверу». Принятый файл сохраняется в каталог Downloads.

Проектирование структуры клиентской части системы удаленного администрирования.

Общая схема проекта клиентской стороны проекта похожа на серверную, но имеет ряд отличий, показана на рисунке 5.19. Клиентское приложение является инициатором соединения, в то время как сервер только ждет подключения клиента.

Рисунок 5.19 – Возможная схема работы клиентской части

Получением и отправкой сообщений от сервера и обратно занимается модуль «Установка и завершение TCP/IP соединения, а также работа в соединении». Если этот модуль получил сообщение протокола VNC, то он передает это сообщение в модуль работы с сообщениями протокола VNC («Прием и передача сообщений протокола VNC»), который в свою очередь передает сообщение в модуль «Работы с сообщениями протокола VNC», где сообщение обрабатывается.

Из модуля «Работа с сообщениями протокола VNC» сообщение могут пойти по трем ветвям. Если полученное сообщение используется на стадии «Рукопожатия» протокола VNC, то оно передается на ветвь с названием «Формирование сообщения протокола VNC используемые на стадии «рукопожатия»». В этой ветки происходит прием, обработка и передача сообщений стадии «Рукопожатия», а так же выполнение инструкций соответствующий протокола VNC.

Если полученное сообщение используется на стадии «Инициализации» протокола VNC, то происходит прием всех необходимых параметров. После приема вызывается модуль «Инициализация необходимых параметров по средствам протокола VNC», в котором инициализируется такие параметры как: тип сжатия передаваемых framebuffer'ов, количество бит на пиксель, размер экрана сервера.

Если полученное сообщение используется в «Рабочей» стадии, то в соответствии с полученным сообщением, либо вызывается модуль «получение обновляемых частей framebuffer'а и прорисовывание их в рабочей области», либо «формирование сообщений, передаваемых непосредственно на удаленный компьютер и управления им удаленно». В первом происходит прием framebuffer'а от серверного приложения и прорисовка его в рабочей области. Во втором происходит отслеживание движение и нажатии/отпускание клавиш манипулятора, нажатие и отпускание клавиш клавиатуры, и формирование соответствующий сообщений протокола VNC.

Для безопасности предусмотренная аутентификация на основе логина и пароля. За аутентификацию отвечает модуль «Аутентификация». При указании соответствующих параметров можно использовать шифрование/расшифрование сообщений протокола VNC. За это отвечает модуль «Шифрование сообщений». Шифрование основано на модифицированной сети Фейстеля, использующей ключ, вычисляемый из логина и пароля.

Также спроектированная дополнительная возможность приема файлов от клиента к серверу. За прием файла отвечает модуль «Прием файла от клиента к серверу». Принятый файл сохраняется в каталог Downloads.

Взаимодействие серверной и клиентской частей системы удаленного администрирования.

Представим взаимодействие серверной и клиентской частей.

В первую очередь на клиенте вызывается функция установки TCP-соединения между TCP-клиентом и TCP-сервером.

Если используется аутентификация и установка TCP-соединения прошла удачно, то от клиента передается сообщения с логином и паролем. Это сообщение принимается на сервере. Где принятые - логин и пароль сравниваются с введенными на сервере. Если они совпадают, то соединение считается установленным и взаимодействие переходит в следующую стадию. Если данные совпали, то сервер разрывает соединение. После установки соединение, начинается передача сообщений протокола VNC. Сначала проходит стадия «Рукопожатия». В этой стадии от сервера отсылаются сообщения с указанием версии протокола, используемыми типами безопасности, на эти сообщения клиент отсылает 1 байт с указанием типа безопасности. После завершения стадии «Рукопожатия» начинается стадия «Инициализации», в которой сервер отсылает клиенту сообщения с указанием размера экрана. А клиент в свою очередь отправляет сообщение, в котором указывает, что хочет использовать монопольный доступ.

После завершения стадии «Инициализации» серверная и клиентские части системы удаленного администрирования переходят в рабочую стадию. В начале это стадии клиент-часть передает серверной части сообщения с указанием количества бит на пиксель и используемый алгоритм сжатия

framebuffe'a. При возникновении события от манипулятора или клавиатуры формируются и передаются соответствующие сообщения. Если событие возникло от манипулятора, то передается новые координаты и маска для кнопок манипулятора, в которой указаны состояния кнопок. Если же событие было принято от клавиатуры, то передается сообщение с указанием кода клавиши и флагом, показывающем нажата или отпущенная данная клавиша. Сервер, принимая эти сообщения, делает соответствующие системные вызовы. Если сообщения от манипулятора, то сервер присваивает новые координаты, и нажимаются или отпускаются кнопки манипулятора. Если сообщение пришло от клавиатуры, то нажимает или отпускается клавиша код, которой был передан.

Если была выбрана опция передачи файла, клиент-часть вызывает диалог выбора файла, после передает сообщение с именем и расширением файла на сервер и размер файла. Сервер принимает сообщение с именем и размером файла. После клиент-часть начинает передачу файла, а сервер соответственно прием.

Также возможно расширить функции системы удаленного администрирования, добавив возможность шифрования передаваемых сообщений.

Задание к лабораторной работе №5

1) Изучить возможности протокола VNC.
2) Написать простейшию клиент-серверную программу для удаленного администрирования.

Требования к отчету по лабораторной работе

В отчете по лабораторной работе должны быть представлены:

1) Возможная структура клиентской и серверной частей системы удаленного администрирования.
2) Схема взаимодействия программы-клиента и программы-сервера.
3) Описание возможностей разработанной системы удаленного администрирования.
4) Листинг программы
5) Экранные формы, демонстрирующие работу программ.

Контрольные вопросы:

1) Охарактеризуйсте достоинства и недостатки известных Вам систем удаленного администрирования?
2) Какой протокол транспортного уровня использует Ваше клиент-серверное приложение? Почему?
3) Какие известные Вам средства можно использовать, чтобы сделать систему удаленного администрирования более защищенной?

Лабораторная работа №6

Тема: Изучение протокола обмена сообщениями и данными о присутствии XMPP. Разработка простейшего клиент-серверного приложения "Мессенджер".

Цель: Изучить протокол обмена сообщениями и данными о присутствии XMPP, разработать простейшее клиент-серверное программное обеспечение «Мессенджер»

Методические указания и задания к лабораторной работе:

Рассмотрим описание одного из популярных протоколов обмена сообщениями XMPP . В RFC 3920 представлено описание протокола XMPP.

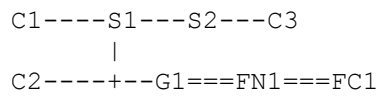
RFC 3920 определяет основные особенности протокола XMPP (eXtensible Messaging and Presence Protocol — расширяемый протокол обмена сообщениями и данными о присутствии), протокола, использующего XML (eXtensible Markup Language — расширяемый язык разметки) для обмена структурированной информацией в режиме близком к реальному времени между двумя любыми сетевыми оконечными точками.

Расширяемый протокол обмена сообщениями и данными о присутствии XMPP (Extensible Messaging and Presence Protocol) является открытым протоколом, использующими XML (Extensible Markup Language [XML]) для сервисов обмена сообщениями в режиме близком к реальному времени, данными о присутствии, запросами/откликами. Базовый синтаксис и семантика были разработаны первоначально в 1999 году сообществом разработчиков системы Jabber с общедоступным кодом. В 2002 рабочая группа XMPP занялась адаптацией протокола Jabber, который должен стать основой технологии обмена сообщениями в реальном масштабе времени (IM) и данными о присутствии сообщества IETF. Результатом работы группы XMPP стал данный документ, который определяет базовые черты XMPP 1.0; расширения, необходимые для реализации обмена сообщениями и данными о присутствии в реальном масштабе времени, определены в RFC-2779 [IMP-REQS] и специфицированы в протоколе XMPP: передача сообщений и данных о присутствии в реальном масштабе времени [XMPP-IM].

Обобщенная архитектура

Хотя XMPP не привязан к какой-то определенной сетевой архитектуре, реализация сессии осуществляется по схеме клиент-сервер, где клиент реализует подключение к серверу с помощью [TCP] TCP-транспорта, и сами серверы взаимодействуют друг с другом также, используя протокол TCP.

На рисунке ниже представлена схема высокоуровневой архитектуры подобной сессии ("-" коммуникации, использующие XMPP протокол, "=" — коммуникации, использующие какой-либо другой протокол).



Символы на рисунке имеют следующие значения:

- C1, C2, C3 = XMPP клиенты
- S1, S2 = XMPP серверы
- G1 = шлюз, который осуществляет согласования XMPP и протоколов, используемых в сети без поддержки XMPP.
- FN1 = внешняя сеть без поддержки XMPP
- FC1 = клиент сети без поддержки XMPP

Сервер

Сервер действует как интеллектуальный уровень абстракции для XMPP коммуникаций. Его основные обязанности:

- установление и поддержание соединения или сессии с другими объектами, в виде XML-потоков (раздел 4) к или от авторизованных клиентов, серверов и прочих объектов.
- маршрутизацию корректно адресованных строф (stanzas) (см. ниже) между такими объектами в рамках XML-потоков.

Большинство XMPP-совместимых серверов осуществляют также запоминание данных, которые используются клиентами (например, список контактов для пользователей системы обмена сообщениями и данными о присутствии с помощью XMPP); в этом случае, XML-данные обрабатываются непосредственно сервером для клиента, а не переадресуются другому объекту.

Клиент

Большинство клиентов подключается непосредственно к серверу посредством TCP и используют XMPP для получения полной функциональности, доступной на сервере. Для каждого авторизованного клиента к серверу могут одновременно подключаться несколько ресурсов (например, устройств или позиций). Каждый ресурс характеризуется идентификатором XMPP-адреса (например, <node@domain/home> или <node@domain/work>), как это определено схемой адресации (см. ниже). Рекомендованным номером порта для соединения клиента с сервером является 5222, как это фиксировано IANA (смотри "Номера портов" (см. ниже)).

Шлюз

Шлюз является специальной функцией сервера, задача которой заключается в трансляции протокола XMPP для внешних систем обмена сообщениями, которые не поддерживают XMPP, а также преобразовании данных, поступающих из удаленных сетей, не поддерживающих XMPP. Примерами могут служить шлюзы для электронной почты (смотри [SMTP]), IRC (Internet Relay Chat, смотри [IRC]), SIMPLE (смотри [SIMPLE]), SMS (Short Message Service), а также сервисы типа AIM, ICQ, MSN Messenger или Yahoo! Instant Messenger.

Сеть

Так как каждый сервер идентифицируется сетевым адресом и так как коммуникации сервер-сервер являются простым расширение протокола клиент-сервер, на практике система представляет собой сеть серверов, которые взаимодействуют друг с другом. Таким образом, например, <juliet@example.com> может обмениваться сообщениями и данными о присутствии, а также другой информацией с <romeo@example.net>. Эта схема подобна протоколам обмена сообщениями (таким как [SMTP]), которые используют стандарты сетевой адресации. Коммуникации между любыми двумя серверами являются опционными. Если они разрешены, такие коммуникации должны осуществляться посредством XML-потоков, которые сопряжены с TCP-соединениями. Рекомендуемым номером порта для соединения серверов является 5269 (смотри раздел "Номера портов").

Схема адресации

Overview

Объектом считается нечто, что может рассматриваться, как конечная точка сети (т.е., ID для сети) и что может осуществлять обмен с использованием XMPP. Все такие объекты являются уникально адресуемыми в формате, совместимом с RFC 2396 [URI]. По историческим причинам адрес XMPP-объекта называется идентификатором Jabber или JID. JID содержит набор элементов, образующих доменный идентификатор, идентификатор узла или ресурса.

Синтаксис JID описан ниже в формализме Бакуса-Наура, определенном в [ABNF].

```
jid             = [ node "@" ] domain [ "/" resource ]
domain          = fqdn / address-literal
fqdn            = (sub-domain 1*("." sub-domain))
sub-domain      = (internationalized domain label)
address-literal = IPv4address / IPv6address
```

Все JID базируются на следующей структуре. Наиболее общим использованием этой структуры является идентификация пользователя системы обмена сообщениями, сервера, к которому пользователь подключается и используемых ресурсов в форме <user@host/resource>. однако, возможны типы узлов, отличные от клиентов; например, специальная комната для разговоров, предлагаемая многопользовательским сервисом, может адресоваться как <room@service> (где "room" является именем chat room, а "service" имя машины, предлагающей этот вид сервиса. Определенный участник такой беседы может адресоваться как <room@service/nick> (где "nick" — имя/прозвище участника такой сессии). Возможно много других типов JID (например, <domain/resource> может быть скриптом или сервисом на стороне сервера).

Каждая из возможных частей JID (идентификаторы узла, домена и ресурса) не должны содержать более 1023 байт, что устанавливает максимальный размер, включая символы '@' и '/', равный 3071 байту.

Идентификатор домена

Идентификатор домена является первичным идентификатором и представляет собой единственный обязательный элемент JID (простой идентификатор домена является корректным значением JID). Он обычно представляет сетевой шлюз или "первичный" сервер, к которому подключаются другие объекты для осуществления XML-маршрутизации и управления данными. Однако объект, адресуемый идентификатором домена, не обязательно является сервером и может быть сервисом, который адресуется как субдомен сервера, который обеспечивает функциональность, неподдерживаемую сервером (например, пользовательский каталог или шлюз к удаленной системе обмена сообщениями).

Идентификатором домена для каждого сервера или сервиса, который будет обмениваться данными через сеть, может быть IP-адрес, но он должен обязательно соответствовать имени домена [DNS]. Идентификатор домена должен иметь международное доменное имя, как это определено в [IDNA]. Прежде чем сравнивать два идентификатора доменов сервер должен (также как и клиент) использовать профайл Nameprep для меток (как это задано в [IDNA]).

Идентификатор узла

Идентификатор узла является опционным вторичным идентификатором, размещаемым перед доменным идентификатором, и отделяемым от него символом '@'. Он обычно представляет собой объект, запрашивающий и использующий доступ к сети, предоставляемый сервером или шлюзом (т.е., клиент). Хотя он может представлять собой и другой объект. Объект, представляемый идентификатором узла адресуется в рамках

контекста конкретного домена; для задач обмена сообщениями и данными о присутствии приложений XMPP, этот адрес называется "чистым JID" и имеет вид <node@domain>.

Идентификатор узла должен быть сформатирован так, чтобы можно было использовать профайл Nodeprep [STRINGPREP]. Прежде чем сравнивать два идентификатора узлов, сервер должен сначала использовать профайл Nodeprep для каждого идентификатора.

Идентификатор ресурса

Идентификатор ресурса является опционным третичным идентификатором, размещаемым после идентификатора домена и отделенным от последнего символом '/'. Идентификатор ресурса может варьироваться от <node@domain> до <domain>. Он обычно представляет определенную сессию, соединение, например, устройство или положение), или объект (например, участник многопользовательского обмена сообщениями), принадлежащее сущности, ассоциированной с идентификатором узла. Идентификатор ресурса не прозрачен для сервера и других клиентов, и обычно определяется реализацией клиента, когда он выдает информацию, нужную для завершения подключения ресурса (Resource Binding, см. ниже). Объект может содержать несколько подключенных ресурсов. Каждый подключенный ресурс имеет свой идентификатор.

Идентификатор ресурса должен быть сформирован так, чтобы было можно применить профайл Resourceprep [STRINGPREP]. Прежде чем сравнивать два идентификатора ресурсов, сервер должен сначала использовать профайл Resourceprep для каждого идентификатора.

Определение адреса

После согласования SASL (см. ниже) и, если нужно, подключения ресурсов (см. ниже), принимающий объект должен определить начальное значение JID.

Для обменов сервер-сервер, начальное значение JID должно определять авторизационную идентичность, как это определено спецификацией уровня безопасности и простой аутентификации SASL (Simple Authentication and Security Layer) [SASL] (см. ниже).

Для коммуникаций клиент-клиент "чистый JID" (<node@domain>) должен определять авторизационную идентичность, как это определено спецификацией [SASL], (см. ниже). Секция идентификатора ресурса "чистого JID" (<node@domain/resource>) должна быть ресурсным идентификатором, согласованным клиентом и сервером при подключении ресурса (см. ниже).

Принимающий объект должен гарантировать, что результирующий JID (включая идентификатор узла, домена, ресурса и разделительные символы) соответствует правилам и форматам, определенным ранее в данном разделе; чтобы удовлетворить этим ограничениям, принимающий объект может быть вынужден заменить JID, присланный инициатором обмена.

XML потоки

Overview

Два фундаментальных принципа делают возможным быстрый асинхронный обмен при сравнительно небольшом поле данных: XML-потоки и XML-строфы (stanzas). Эти термины определяются следующим образом:

Определение XML-потока:

XML-поток является контейнером для обмена XML-элементами между объектами через сеть. Начало XML-потока однозначно определяется открывающим XML-тэгом <stream> (с декларацией соответствующих атрибутов названий позиций), в то время как конец XML-потока однозначно определяется закрывающим XML-тэгом </stream>. Во время существования потока, объект его инициировавший, может посылать произвольное число XML-элементов, либо элементов, используемых для согласования параметров потока (например, для согласования использования TLS (см. ниже), либо использования SASL (см. ниже)), либо XML-строф (как это определяется, <message/>, <presence/> или <iq/>). "Исходный поток" согласуется инициатором сессии (обычно клиентом или сервером) с принимающим объектом (обычно сервером). Исходный поток обеспечивает однонаправленную коммуникацию между инициатором и получателем; для того чтобы разрешить информационный обмен от принимающей стороны к инициатору, приемник должен согласовать поток в обратном направлении ("поток-отклик").

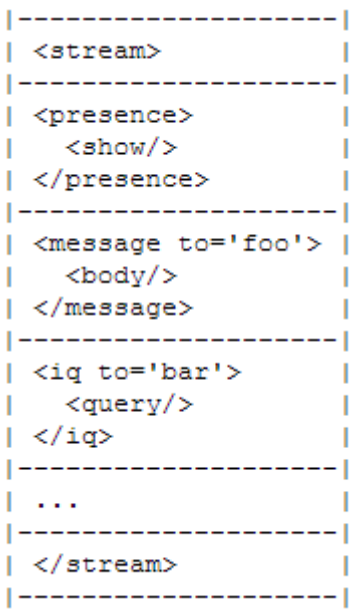
Определение XML строфы:

XML-строфа является дискретным семантическим блоком структурированной информации, который посылается одним объектом другому объекту через XML-поток. XML-строфа является дочерним элементом по отношению элементу <stream/>. Начало любой XML-строфа однозначно определяется стартовым тэгом XML-потока (depth=1) (например, <presence>), а конец любой XML-строфы однозначно задается закрывающим тэгом (например, </presence>). XML-строфа может содержать дочерние элементы (сопроводительные атрибуты и XML-символы). Единственными видами XML-строф, определенными здесь, являются: <message/>>, <presence/> и <iq/> (см. ниже). XML-элементы, посланные для целей согласования транспортной безопасности TLS (Transport Layer Security) (см.

ниже) и согласования SASL (см. ниже) не рассматриваются как XML-строфы.

Рассмотрим пример клиентской сессии взаимодействия с сервером. Для того чтобы соединиться с сервером, клиент должен инициировать XML-поток путем посылки серверу открывающего тэга <stream>, опционно перед ним может следовать текстовая декларация, специфицирующая XML-версию и символьное кодирование (смотри "Включение текстовых деклараций" (см. ниже); а также Кодирование символов (см. ниже)). В соответствии с локальной политикой и предоставляемыми сервисами сервер должен откликнуться и сформировать XML-поток к клиенту. Как только клиент завершает согласование SASL (см. ниже), он может посылать любое число XML-строф любому получателю в сети. Когда клиент захочет прервать сессию, он просто посылает серверу закрывающий тэг </stream> (поток может быть прерван также и сервером), после этого как клиент так и сервер разрывают TCP-соединение.

По существу, XML-поток функционирует как конверт для всех XML-строф, посланных во время сессии. Мы можем представит это в упрощенной форме как:



Связь с TCP

Хотя нет необходимости связывать XML-поток с TCP-соединением [TCP] (например, два объекта могут установить соединение друг с другом посредством другого механизма, данная спецификация предполагает работу только с TCP-протоколом. В контексте коммуникаций клиент-сервер, сервер должен позволить клиенту разделять одно TCP-соединение для XML-строф, посланных клиентом серверу и от сервера клиенту. В контексте коммуникаций сервер-сервер, сервер должен использовать одно TCP-

соединение для XML-строф, посланных сервером своему партнеру, и другое TCP-соединение (инициированное партнером) для передачи строф от партнера серверу.

Безопасность потока

Когда согласуется формирование XML-потоков в XMPP 1.0, следует использовать TLS, а SASL должен использоваться только при определенных условиях, описанных ниже. Безопасности исходного потока (т.е., потока от инициатора получателю) и поток-отклик (т.е., поток от получателя к инициатору) должны обеспечиваться независимо. Объект не должен пытаться посылать XML-строфы (см. ниже) через поток, прежде чем он будет аутентифицирован, но если он это сделает, тогда другой объект не должен воспринимать такие строфы и должен возвращать уведомление об ошибке <not-authorized/> и затем прерывать XML-поток и TCP-соединение; заметим, что здесь вовлечены только XML-строфы (т.е., элементы '<message/>, <presence/> и <iq/>), а не XML элементы, используемые для согласования параметров потока (например, элементы, используемые для согласования работы с TLS (см. ниже) или с SASL (см. ниже)).

Атрибуты потока

Существуют следующие атрибуты элемента потока:

- to - Следует использовать только в заголовке XML-потока между инициатором и получателем. Значение этого атрибута должно соответствовать имени машины получателя. Не следует использовать атрибут 'to' в заголовках XML-потоков, где получатель посылает отклик инициатору; однако, если атрибут 'to' использован в таком контексте, он должен быть молча проигнорирован инициатором.
- from - Следует использовать только в заголовках XML-потоков от получателя к инициатору, и его значение должно быть равно имени машины, обслуживаемой получателем. Не должно быть атрибутов 'from' в заголовках XML-потоков между инициатором и получателем; однако, если атрибут 'from' все же вставлен, он должен молча игнорироваться получателем.
- id - Следует использовать только в заголовках XML-потоков от получателя к инициатору. Этот атрибут является уникальным идентификатором, сформированным получателем, для использования в качестве ключа сессии для потоков инициатора, и должен быть уникальным для принимающих приложений (в норме для сервера). Заметим, что ID-потока может быть критичным в отношении безопасности и, следовательно, непредсказуемым (рекомендации смотри в [RANDOM]). Не следует использовать атрибут 'id' в заголовках XML-потоков, посылаемых инициатором получателю;

однако, если атрибут 'id' все же включен, он должен молча игнорироваться получателем.

- xml:lang - Следует включать инициатором в заголовок, чтобы специфицировать язык по умолчанию для любых посылаемых им символьных данных. Если атрибут включен, получатель должен запомнить его в качестве значения по умолчанию как для исходного потока, так и для потока-отклика; если атрибут не был включен, получатель должен использовать для обоих потоков значение из конфигурации, которое он должен пересылать в заголовке потока-отклика. Для всех строф, переданных в исходном потоке, если инициатор не использовал атрибут 'xml:lang', получатель должен использовать значение по умолчанию; если инициатор включил атрибут 'xml:lang', получатель не должен модифицировать или удалять его (смотри также xml:lang (см ниже)). Значение атрибута 'xml:lang' должно быть NMTOKEN (как это определено of [XML]) и должно следовать формату, определенному в RFC 3066 [LANGTAGS].
- version - Присутствие атрибута версии с значением, равным по крайней мере "1.0" сигнализирует о поддержке потоковых протоколов, описанных в данной спецификации.

Итак:

to	Имя машины получателя	молча игнорируется
from	молча игнорируется	Имя машины получателя
id	молча игнорируется	ключ сессии
xml:lang	язык по умолчанию	язык по умолчанию
version	Сигнал поддержки XMPP 1.0	Сигнал поддержки XMPP 1.0

Поддержка версии

Версия XMPP, специфицированная здесь, равна "1.0"; в частности, подразумевает протоколы, ориентированные на потоки, (использование TLS , использование SASL и ошибки в потоках), а также семантику трех определенных типов XML-строф (<message/>, <presence/> и <iq/>). Схема нумерации версий XMPP имеет формат "<major>.<minor>". Старший и младший коды версии должны рассматриваться как отдельные целые числа и каждое число может инкрементироваться. Таким образом, "XMPP 2.4" будет более ранней версией чем "XMPP 2.13", которая в свою очередь младше, чем версия "XMPP 12.3". Лидирующие нули (например, "XMPP 6.01") должны игнорироваться получателями и не должны пересылаться.

Старший код номера версии следует инкрементировать только, если форматы потока или строф, или необходимые действия изменились настолько сильно, что объект старой версии не сможет их интерпретировать корректно. Младший код номера версии индицирует новые возможности, и

он должен игнорироваться объектом более ранней версии. Например, младший код номера версии может указывать на возможность обработки вновь определенных типов атрибута сообщения, данных о присутствии, или IQ-строф. Объект с большим значением младшего кода версии будет знать, что его партнер не способен правильно воспринять новый тип атрибута, и не будет его посылать.

Следующие правила следует использовать при генерации и обработке атрибута 'version' в заголовках потоков:

1. Инициатор должен установить значение атрибута 'version' в заголовке исходного потока равным наивысшей поддерживаемой версии (например, если наивысшая поддерживаемая им версия равна описанной в данном документе, он должен установить ее равной "1.0").
2. Приемник должен установить значение атрибута 'version' в заголовке потока-отклика равным либо значению версии, присланной инициатором, либо наивысшему поддерживаемому значению приемника, в зависимости от того какое значение ниже. Приемник должен осуществлять числовое сравнение старшего и младшего кодов версии, а не использовать значение строки "<major>.<minor>".
3. Если номер версии, включенный в заголовок потока-отклика является по крайней мере в старшей части меньше, чем код версии, включенный в заголовок исходного потока, сущности новой версии не смогут работать. Инициатор должен сформировать уведомление об ошибки <unsupported-version/> и прервать XML-поток и разорвать TCP-соединение.
4. Если какой-либо объект получает заголовок без атрибута 'version', объект должен считать поддерживаемую версию объекта равной "0.0" и не должен включать атрибут 'version' в заголовок потока, который он посылает в виде отклика.

Декларации пространства имен

Потоковый элемент должен содержать декларации потокового пространства имен и пространства имен по умолчанию ("декларация пространства имен" определено XML спецификации пространства имен [XML-NAMES]).

Характеристики потока

Если инициатор включает атрибут 'version' в заголовок исходного потока со значением по крайней мере "1.0", приемник должен послать дочерний элемент <features/> (перед ним размещается префикс пространства имен потока) инициатору, для того чтобы анонсировать любые характеристики поточного уровня, которые могут быть согласованы (или возможности, которые нужно анонсировать). Сейчас, это используется только для

анонсирования использования TLS, использования SASL и подключаемых ресурсов (см. ниже), а также для установления сессий, как это определено в [XMPP-IM]; однако, функциональность потоковых характеристик может использоваться для анонсирования других согласуемых параметров в будущем. Если объект не понимает или не поддерживает некоторые возможности, он должен их молча игнорировать.

Ошибки потока

Корневой потоковый элемент может содержать дочерний элемент <error/>, перед которым следует префикс пространства имен потока. Дочерний элемент ошибки должен быть послан соответствующим объектом (обычно сервером, а не клиентом), если он понимает, что на поточном уровне произошла ошибка.

Правила

Следующие правила используются в отношении ошибок уровня потока:

- Предполагается, что все ошибки потокового уровня являются неисправимыми; следовательно, если ошибка случается на уровне потока, объект, который детектирует ошибку должен послать сигнал потоковой ошибки другому объекту, послать закрывающий тэг </stream>, и завершить TCP-соединение.
- Ошибка происходит, когда поток сформирован, получатель должен послать открывающий тэг <stream>, включить элемент <error/> в качестве дочернего, послать закрывающий тэг </stream> и разорвать TCP-соединение. В этом случае, если инициатор предлагает неизвестную машину в атрибуте 'to' (или вообще не предлагает атрибута 'to'), сервер должен выдать перед терминацией заслуживающее доверия имя машины в атрибуте 'from' заголовка потока.

Синтаксис

Синтаксис потоковых ошибок имеет следующий формат:

```
<stream:error>
  <defined-condition xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
  <text xmlns='urn:ietf:params:xml:ns:xmpp-streams'
        xml:lang='langcode'>
    OPTIONAL descriptive text
  </text>
  [OPTIONAL application-specific condition element]
</stream:error>
```

Элемент <error/>:

- должен содержать дочерний элемент, соответствующий одной из выявленных ошибок строф; этот элемент должен быть задан пространством имен 'urn:ietf:params:xml:ns:xmpp-streams'
- содержать дочерний элемент <text/>, содержащий XML символьные данные, которые характеризуют ошибку более детально; этот элемент должен быть связан с пространством имен 'urn:ietf:params:xml:ns:xmpp-streams' и должен иметь атрибут 'xml:lang', характеризующий язык, используемый символьными данными.
- содержать дочерний элемент для ошибки, специфичной для определенного состояния приложения; этот элемент должен быть привязан к прикладному пространству имен, а его структура определяется этим пространством имен.

Элемент <text/> является опционным. В случае включения он должен использоваться только для описательных и диагностических данных, которые являются дополнительным пояснением условий. Он не должен интерпретироваться приложением программно. Он не должен использоваться в качестве сообщения об ошибке пользователю, но может служить дополнением к сообщению об ошибке.

Определенные условия

Определены следующие условия для ошибок уровня потока:

- <bad-format/> - объект послал XML, который не может быть обработан; эта ошибка может использоваться вместо каких-то более специфических связанных с XML ошибок, таких как <bad-namespace-prefix/>, <invalid-xml/>, <restricted-xml/>, <unsupported-encoding/> и <xml-not-well-formed/>, хотя предпочтительнее более специфические сообщения об ошибках.
- <bad-namespace-prefix/> - объект послал префикс пространства имен, которое не поддерживается, или не послал префикс пространства имен на элемент, который требует такого префикса (смотри XML имена пространства имен и префиксы (см. ниже)).
- <conflict/> - сервер закрывает активный поток для этого объекта, так как инициирован новый поток, который конфликтует с существующим.
- <connection-timeout/> -объект не генерировал трафик через поток в течение некоторого времени (сконфигурирован исключительно для локальной работы).
- <host-gone/> - значение атрибута 'to', проверенное инициатором в заголовке потока соответствует машине, которая более не обслуживается сервером.
- <host-unknown/> - значение атрибута 'to', выданное инициатором в заголовке потока, не соответствует машине, обслуживаемой сервером.

- <improper-addressing/> - строфа, пересланная между серверами, не содержит атрибута 'to' или 'from' (или атрибут не имеет значения).
- <internal-server-error/> - сервер имеет ошибку в конфигурации или имеет место другая внутренняя ошибка, мешающая обработке потока.
- <invalid-from/> - JID имя машины, представленное в адресе 'from', не соответствует авторизованному JID или соответствующему согласованному домену (с помощью SASL или dialback).
- <invalid-id/> - ID потока или dialback ID некорректны или не соответствуют присланному ранее ID.
- <invalid-namespace/> - имя пространства имен потоков не соответствует "http://etherx.jabber.org/streams" или имя пространства имен dialback не совпадает с "jabber:server:dialback" (смотри "XML имена пространства имен и префиксы").
- <invalid-xml/> - объект послал некорректный XML через поток серверу, который выполняет валидацию (смотри "Валидация" (см. ниже)).
- <not-authorized/> - объект попытался послать данные, прежде чем поток оказался аутентифицирован, или он не авторизован для выполнения согласования формирования потока; приемник не должен обрабатывать предлагаемую строфу до посылки сообщения об ошибке.
- <policy-violation/> - объект нарушил некоторые правила внутренней политики; сервер может специфицировать политику в элементе <text/>.
- <remote-connection-failed/> - сервер не может корректно подключится к удаленному объекту, который необходим для авторизации или аутентификации.
- <resource-constraint/> - сервер не имеет достаточных ресурсов, чтобы обслужить поток.
- <restricted-xml/> - объект попытался послать нечто с ограниченным применением, например, комментарий, инструкцию обработки, DTD, ссылку на объект или недопустимый символ (смотри "Ограничения" (см. ниже)).
- <see-other-host/> - сервер не будет осуществлять сервис для инициатора, но переадресует трафик другой машине; сервер должен специфицировать имя альтернативной машины или IP-адрес (который должен являться корректным доменным идентификатором) в виде элемента XML символьных данных <see-other-host/>.
- <system-shutdown/> - сервер выключается и все активные потоки закрываются.
- <undefined-condition/> - обстоятельства ошибки не совпадает ни с одним из перечисленных выше; это условие ошибки следует использовать только совместно с условиями специфическими для приложения.
- <unsupported-encoding/> - инициатор использовал кодирование потока, которое не поддерживается сервером (смотри "Кодировка символов" (см. ниже)).

- <unsupported-stanza-type/> - инициатор послал дочерний поток первого уровня, который не поддерживается сервером.
- <unsupported-version/> - значение атрибута 'version', выданное инициатором в заголовке потока, специфицирует версию XMPP, которая не поддерживается сервером; сервер может специфицировать версии, поддерживаемого элемента с помощью <text/>.
- <xml-not-well-formed/> - инициатор послал XML, который некорректно сформирован (смотри [XML]).

Специфические для приложения условия

Как было отмечено, приложение может выдать свою информацию об ошибке потока путем включения дочернего элемента ошибки. Специфический для приложения элемент должен быть дополнением к стандартному элементу. Таким образом, элемент <error/> будет содержать 2-3 дочерних элементов:

```
<stream:error>
     <xml-not-well-formed
         xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
     <text xml:lang='en' xmlns='urn:ietf:params:xml:ns:xmpp-streams'>
       Some special application diagnostic information!
     </text>
     <escape-your-data xmlns='application-ns'/>
   </stream:error>
   </stream:stream>
```

Упрощенные примеры потоков

Ниже представлены два упрощенных примера сессий, базирующихся на потоках клиент-сервер (где в "C"-строках данные пересылаются от клиента к серверу, а в "S"-строках — от сервера к клиенту); эти примеры включены для целей иллюстрации концепции, изложенной далее.

Базовая "сессия":

```
C: <?xml version='1.0'?>
   <stream:stream
       to='example.com'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       version='1.0'>
S: <?xml version='1.0'?>
   <stream:stream
       from='example.com'
       id='someid'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       version='1.0'>
```

... шифрование, аутентификация и подключение ресурсов ...

```
C:   <message from='juliet@example.com'
             to='romeo@example.net'
             xml:lang='en'>
C:     <body>Art thou not Romeo, and a Montague?</body>
C:   </message>
S:   <message from='romeo@example.net'
             to='juliet@example.com'
             xml:lang='en'>
S:     <body>Neither, fair saint, if either thee dislike.</body>
S:   </message>
C: </stream:stream>
S: </stream:stream>
```

Сессия реализована плохо:

```
C: <?xml version='1.0'?>
   <stream:stream
       to='example.com'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       version='1.0'>
S: <?xml version='1.0'?>
   <stream:stream
       from='example.com'
       id='someid'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       version='1.0'>
```

... шифрование, аутентификация и подключение ресурсов ...

```
C: <message xml:lang='en'>
     <body>Bad XML, no closing body tag!
   </message>
S: <stream:error>
    <xml-not-well-formed
        xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
   </stream:error>
S: </stream:stream>
```

Использование TLS

Overview

XMPP включает в себя метод обеспечения безопасности потока от фальсификации и подслушивания. Метод криптозащиты канала использует протокол безопасности транспортного уровня [TLS], с расширениями "STARTTLS", которые смоделированы для протоколов IMAP [IMAP], POP3 [POP3] и ACAP [ACAP], как это описано в RFC 2595 [USINGTLS]. Имя

пространства имен для расширения STARTTLS = 'urn:ietf:params:xml:ns:xmpp-tls'.

Администратор данного домена может потребовать использования TLS для коммуникаций клиент-сервер и сервер-сервер. Клиенты должны использовать TLS, чтобы обеспечить безопасность потоков, прежде чем пытаться завершить согласование SASL, а серверы должны использовать TLS между двумя доменами для целей обеспечения безопасности коммуникаций сервер-сервер.

Используются следующие правила:

1. Инициатор, который следует данной спецификации должен включить в заголовок потока атрибут 'version', содержащий значение "1.0".
2. Если согласование TLS происходит между двумя серверами, коммуникации не должны происходить до тех пор, пока DNS не распознает имена машин, введенные серверами (смотри "Коммуникации сервер-сервер").
3. Когда приемник, который следует данной спецификации, получает заголовок исходного потока, который содержит атрибут 'version', равный по крайней мере "1.0", он должен включить элемент <starttls/> (привязанный к пространству имен 'urn:ietf:params:xml:ns:xmpp-tls') вместе со списком других характеристик потока.
4. Если приемник намерен использовать TLS, согласование параметров TLS должно быть завершено до согласования использования SASL; такой порядок диалога необходим, чтобы защитить аутентификационную информацию, посланную во время согласования применения SASL.
5. Во время согласования использования TLS, объект не должен посылать каких-либо символов пробелов в элементе корневого потока в качестве сепараторов между элементами (любой пробел, имеющийся в примерах TLS ниже, включен исключительно из соображений читаемости); этот запрет помогает гарантировать корректность байт на уровне безопасности.
6. Приемник должен осуществлять согласование применения TLS сразу после посылки завершающего символа ">" элемента <proceed/>. Инициатор должен осуществлять согласование применения TLS сразу после получения завершающего символа ">" элемента <proceed/> от приемника.
7. Инициатор должен проверить сертификат, представленный приемником; (смотри "Проверка сертификата").
8. Сертификаты должны проверяться по поводу имени машины, выданного инициатором, (например, пользователем), а не имя машины, полученное с помощью DNS; например, если пользователь специфицирует имя машины "example.com", а DNS SRV прислал

"im.example.com", сертификат должен проверять версию "example.com". Если JID для любого XMPP-объекта (например, клиента или сервера) присутствует в сертификате, он должен быть представлен, в виде UTF8String в пределах имени некоторого объекта (otherName) внутри subjectAltName. Делается это с привлечением объектного идентификатора [ASN.1] "id-on-xmppAddr".

9. Если согласование применения TLS прошло успешно, приемник должен ликвидировать любые данные, полученные ранее от инициатора небезопасным способом.
10. Если согласование применения TLS прошло успешно, инициатор должен аннулировать любые данные, полученные ранее от приемника небезопасным способом.
11. Если согласование применения TLS прошло успешно, приемник не должен предлагать инициатору расширения STARTTLS, а также другие возможности, которые предложены, когда поток рестартовал.
12. Если согласование применения TLS прошло успешно, инициатор должен приступить к согласованию SASL.
13. Если согласование применения TLS завершилось неудачей, приемник должен прервать XML-поток и разорвать TCP-соединение.
14. По поводу механизмов, которые должны быть непременно поддержаны, смотри в "Обязательные для использования технологии".

Идентификатор объекта ASN.1 для XMPP-адреса

Идентификатор объекта [ASN.1] "id-on-xmppAddr", описанный выше, определяется следующим образом:

```
id-pkix OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
        dod(6) internet(1) security(5) mechanisms(5) pkix(7) }

id-on  OBJECT IDENTIFIER ::= { id-pkix 8 }  -- other name forms

id-on-xmppAddr  OBJECT IDENTIFIER ::= { id-on 5 }

XmppAddr ::= UTF8String
```

Этот объектный идентификатор может быть представлен в точечном формате вида "1.3.6.1.5.5.7.8.5".

Диалог

Когда инициатор обеспечивает безопасность потока с помощью TLS, реализуются следующие шаги:

1. Инициатор открывает TCP-соединение и инициирует поток путем посылки XML-заголовка получателю. В заголовок потока вставляется атрибут 'version', со значением как минимум "1.0".
2. Получатель откликается установлением TCP-соединения и посылкой XML-заголовка потока инициатору, включая при этом в заголовок атрибут 'version', содержащий значение как минимум "1.0".

3. Получатель предлагает инициатору расширение STARTTLS, включив его вместе с другими возможностями потока (если для взаимодействия с получателем требуется TLS, он должен сигнализировать об этом с помощью включения элемента <required/> в качестве дочернего элемента <starttls/>).
4. Инициатор выдает команду STARTTLS (т.е., элемент <starttls/>, соотнесенный с пространством имен 'urn:ietf:params:xml:ns:xmpp-tls'), чтобы уведомит получателя, что он хочет начать согласование применения TLS.
5. Получатель должен откликнуться либо элементом <proceed/>, либо элементом <failure/>, соотнесенным с пространством имен 'urn:ietf:params:xml:ns:xmpp-tls'. Если произойдет сбой, получатель должен прервать XML-поток и разорвать TCP-соединение. Если все в порядке, участники должны попытаться завершить согласование применения TLS через имеющееся TCP-соединение и до завершения этого процесса не должны посылать какие-либо XML-данные.
6. Инициатор и получатель пытаются согласно с [TLS] завершить согласование TLS.
7. Если реализация TLS оказалась неудачной, получатель должен разорвать TCP-соединение. В случае же успеха, инициатор должен сформировать новый поток, послав открывающий XML-заголовок получателю (необязательно посылать сначала закрывающий тэг </stream>, так как получатель и инициатор должна рассматривать исходный поток закрытым после успешного завершения согласования применения TLS).
8. После получения нового заголовка потока от инициатора, получатель должен откликнуться посылкой инициатору нового заголовка XML-потока вместе со всеми доступными возможностями (возможность STARTTLS не включается).

Пример клиент-сервер

Шаг 1: Клиент формирует поток к серверу:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Шаг 2: Сервер откликается посылкой тэга потока клиенту:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_123'
    from='example.com'
    version='1.0'>
```

Шаг 3: Сервер посылает клиенту расширение STARTTLS и данные о механизме аутентификации и других особенностях потока:

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
    <required/>
  </starttls>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
</stream:features>
```

Шаг 4: Клиент посылает серверу команду STARTTLS:

```
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Шаг 5: Сервер информирует клиента о том, что он может продолжить работу:

```
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Шаг 5 (alt): Сервер информирует клиента, что согласование TLS не состоялось и следует прервать поток и разорвать TCP-соединение:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
</stream:stream>
```

Шаг 6: Клиент и сервер пытаются завершить согласование применения TLS через существующее TCP-соединение.

Шаг 7: Если согласование TLS успешно, клиент формирует новый поток к серверу:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Шаг 7 (alt): Если согласование TLS не получилось, сервер закрывает TCP-соединение.

Шаг 8: Сервер реагирует посылкой клиенту заголовка потока и любых характеристик потока:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='example.com'
    id='c2s_234'
    version='1.0'>
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
    <mechanism>EXTERNAL</mechanism>
  </mechanisms>
</stream:features>
```

Шаг 9: Клиент продолжает согласование SASL

Пример сервер-сервер можно посмотреть в RFC по протоколу (скачать можно по адресу http://webi.ru/webi_files/xmpp_protocol.html)

Использование SASL

Overview

MPP содержит в себе метод аутентификации потока с помощью XMPP-профайла протокола SASL (Simple Authentication and Security Layer) [SASL]. SASL предоставляет обобщенный метод добавления поддержки аутентификации для протоколов с установлением соединения, и XMPP использует универсальный профайл XML пространства имен для SASL, который соответствует требованиям [SASL].

Используются следующие правила:

1. Если согласование применения SASL происходит между двумя серверами, коммуникации не должны продолжаться до тех пор, пока с помощью DNS не выяснены имена присвоенные серверам (смотри "Коммуникации сервер-сервер").
2. Если инициатор может согласовать применение SASL, он должен включить в заголовок исходного потока атрибут 'version' со значением по крайней мере равным "1.0".
3. Если получатель может согласовать применение SASL, он должен анонсировать один или более аутентификационных механизмов в элементе <mechanisms/>, сопряженным с пространством имен 'urn:ietf:params:xml:ns:xmpp-sasl', в отклике на открывающий тэг потока, полученный от инициатора (если открывающий тэг потока включает в себя атрибут 'version' со значением по крайней мере равным "1.0").
4. Во время согласования применения SASL, объект не должен в качестве сепараторов элементов посылать символы пробелов (см. процедуру формирования содержимого в [XMPP-IM] и [XML]) в элементе корневого потока (любой символ пробела, представленный в примерах SASL, введен исключительно с целью обеспечения читабельности текста); этот запрет помогает гарантировать корректность представления материала на уровне безопасности.
5. Любые XML символьные данные, содержащиеся в XML-элементах, и используемые при согласовании SASL должны иметь кодировку base64, где кодировка привязана к определениям из RFC 3548 [BASE64].
6. Если предоставление "simple username" поддерживается выбранным механизмом SASL (например, это поддерживается механизмами DIGEST-MD5 и CRAM-MD5, но не поддерживается механизмами EXTERNAL и GSSAPI), во время аутентификации, инициатор должен обеспечить в качестве простого имени пользователя свой домен (IP-

адрес или полное доменное имя, как оно записано в доменном идентификаторе) в случае коммуникаций сервер-сервер, или свое имя аккаунта (имя пользователя или узла, содержащегося в XMPP-идентификаторе узла) в случае коммуникаций клиент-сервер.

7. Если инициатор хочет действовать в интересах другого объекта и выбранный механизм SASL поддерживает передачу авторизационной идентичности, инициатор должен предоставить авторизационную идентичность в процессе согласования применения SASL. Если инициатор не хочет действовать в интересах другого объекта, он не должен предоставлять идентичность авторизации. Как специфицировано в [SASL], инициатор не должен предоставлять идентичность авторизации, за исключением случая, когда идентичность авторизации отличается от идентичности по умолчанию, полученной согласно [SASL]. Если такие данные предоставляются, значение идентичности авторизации должно иметь формат <domain> (т.е., только идентификатор домена) для серверов и формат <node@domain> (т.е., идентификатор узла и идентификатор домена) для клиентов.
8. В случае успешного согласования SASL, которое включает в себя согласование уровня безопасности, получатель должен ликвидировать любую информацию, полученную от инициатора, которая не получена непосредственно в процессе согласования SASL.
9. В случае успешного согласования SASL, которое включает в себя согласование уровня безопасности, инициатор должен ликвидировать любую информацию, полученную от получателя, которая не получена непосредственно в процессе согласования SASL.
10. В отношении механизмов, которые нужно поддерживать смотри "Обязательные для использования технологии".

Narrative

Когда инициатор аутентифицирует получателя, используя SASL, процедура включает в себя следующие шаги:

1. Инициатор запрашивает аутентификацию SASL путем включения атрибута 'version' в открывающий заголовок XML потока, направленного получателю, со значением равным "1.0".
2. После посылки заголовка XML-потока в виде отклика, получатель анонсирует список доступных механизмов аутентификации SASL; каждый из этих элементов <mechanism/> включаются в качестве дочерних в контейнерный элемент <mechanisms/>, соотнесенный с пространством имен 'urn:ietf:params:xml:ns:xmpp-sasl', который в свою очередь является дочерним элементом <features/> в пространстве имен потока. Если TLS нужно установить, прежде чем использовать какой-то конкретный механизм аутентификации, получатель не должен

помещать этот механизм в список механизмов аутентификации SASL, до согласования TLS. Если инициатор предоставляет корректный сертификат при предварительном согласовании TLS, получатель при согласовании SASL должен предложить инициатору механизм SASL EXTERNAL (смотри [SASL]), хотя механизм EXTERNAL может быть предложен также и при других обстоятельствах.

3. Инициатор выбирает механизм путем посылки элемента <auth/>, соотнесенного с пространством имен 'urn:ietf:params:xml:ns:xmpp-sasl', получателю. Элемент содержит также значение атрибута 'mechanism'. Этот элемент может содержать символьные XML-данные (в терминологии SASL, "исходный отклик"), если механизм поддерживает или требует этого. Если инициатор должен послать исходный отклик нулевой длины, он должен передать отклик в виде одиночного символа равенства ("="), который указывает, что этот отклик не содержит данных .
4. Если необходимо, получатель направляет вызов инициатору путем посылки инициатору элемента <challenge/>, соотнесенного с пространством имен 'urn:ietf:params:xml:ns:xmpp-sasl'. Этот элемент может содержать символьные данные (которые должны быть обработаны в соответствии с определением механизма SASL, который выбрал инициатор).
5. Инициатор реагирует на вызов посылкой получателю элемента <response/>, соотнесенного с пространством имен 'urn:ietf:params:xml:ns:xmpp-sasl'. Этот элемент может содержать символьную XML-информацию (которая должна быть обработана в соответствии с определением механизма SASL, который выбрал инициатор).
6. Если необходимо, получатель посылает несколько вызовов, а инициатор отправляет несколько откликов.

Эти последовательности вызов/отклик продолжаются до тех пор пока не случится одно из трех событий:

1. Инициатор обрывает диалог посылкой получателю элемента <abort/>, соотносящегося с пространством имен 'urn:ietf:params:xml:ns:xmpp-sasl'. После получения элемента <abort/> получатель должен разрешить конфигурируемое но разумное число повторных попыток (по крайней мере 2), после которых он должен прервать TCP-соединение. Это позволяет инициатору (например, конечному пользователю — клиенту) перепроверить неверно выданные параметры авторизации (например, опечатку в пароле) без необходимости выполнения повторного соединения.
2. Получатель сообщает о неудаче диалога посылкой инициатору элемента <failure/>, соотносящегося пространству имен 'urn:ietf:params:xml:ns:xmpp-sasl', конкретная причина неудачи должна

быть сообщена в соответствующем дочернем элементе элемента <failure/>, как это определено в "Ошибки SASL". В случае неудачи получатель должен разрешить конфигурируемое, но разумное число повторных попыток (по крайней мере 2), после которых он должен прервать TCP-соединение. Это позволяет инициатору (например, конечному пользователю — клиенту) перепроверить неверно выданные параметры авторизации (например, опечатку в пароле) без необходимости выполнения повторного соединения.

3. Получатель сообщает об успехе диалога посылкой инициатору элемента <success/>, соотносимого с пространством имен 'urn:ietf:params:xml:ns:xmpp-sasl'. Этот элемент может содержать символьную XML-информацию (в терминологии SASL "дополнительные данные при успехе"), если это требуется выбранным механизмом SASL. После получения элемента <success/> инициатор должен сформировать новый поток, путем посылки открывающего заголовка XML-потока получателю (посылать закрывающий тэг </stream> не нужно, так как приемник и инициатор должны рассматривать исходный поток закрытым после отправки или получения элемента <success/>). После получения заголовка нового потока от инициатора получатель должен реагировать посылкой инициатору нового заголовка потока и информировать партнера об имеющихся возможностях посредством элемента <features/>.

Определение SASL

Требования профайла [SASL] определяют необходимость предоставления следующей протокольной информации:

- имя сервиса:
- "xmpp"
- стартовая последовательность:
- после того как инициатор выдает открывающий заголовок потока и получатель соответственно откликается, получатель выдает список приемлемых методов аутентификации. Инициатор выбирает один из методов и посылает получателю значение атрибута 'mechanism', содержащегося в элементе <auth/>.
- последовательность обмена:
- вызовы и отклики реализуются посредством обмена элементами <challenge/>, направляемыми от получателя инициатору, и элементами <response/>, отправляемыми инициатором получателю. Получатель сообщает о неудаче с помощью посылки элемента <failure/>, а об успехе — посредством посылки элемента <success/>. Инициатор абортирует обмен путем посылки элемента <abort/>. После успешного согласования параметров потока, обе стороны считают исходный поток закрытым и посылают друг другу заголовки нового потока.

- согласование на уровне безпасности:
- уровень безопасности начинает функционировать сразу после посылки завершающего символа ">" элемента <success/> получателю, и сразу после получения инициатором завершающего символа ">" элемента <success/>.
- использование авторизационной идентичности:
- авторизационная идентичность может использоваться xmpp, чтобы указать <node@domain> клиента (если это не значение по умолчанию) или посылки <domain> сервера.

Ошибки SASL

Определены следующие условия ошибок, сопряженных с SASL:

- <aborted/>
- Получатель подтверждает получение элемента <abort/>, посланного инициатором; посылается в ответ на элемент <abort/>.
- <incorrect-encoding/>
- Данные предоставленные инициатором не могут быть обработаны из-за того, что кодировка [BASE64] оказалась некорректной (например, потому что кодировка не соответствует [BASE64]); в ответ посылается элемент <response/> или элемент <auth/> вместе с данными исходного отклика.
- <invalid-authzid/>
- authzid, предоставленный инициатором, либо по причине неправильного формата, либо из-за того, что инициатор не имеет разрешения авторизовать данный идентификатор, присланный в элементах отклика <response/> или <auth/>.
- <invalid-mechanism/>
- Инициатор не предоставил в элементе <auth/> отклика механизма или запросил механизм, который не поддерживается получателем.
- <mechanism-too-weak/>
- Механизм, запрошенный инициатором, слабее предусмотренного политикой сервера для данного инициатора; присылается в элементе <response/> или <auth/> отклика.
- <not-authorized/>
- Аутентикация не прошла из-за того, что инициатор не предоставил корректные параметры (это может быть, например, неизвестное имя пользователя); посылается в элементах <response/> или <auth/> отклика.
- <temporary-auth-failure/>
- Аутентикация не прошла из-за временной ошибки на стороне получателя; посылается в в элементах <auth/> или <response/> отклика.

Пример клиент-сервер

Следующий пример показывает схему обмена данными при аутентификации клиента сервером с привлечением SASL, это делается в норме после согласования применения TLS (заметим: альтернативные шаги, показанные ниже, приведены для иллюстрации работы протокола в случае ошибок).

Шаг 1: Клиент формирует поток до сервер:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Шаг 2: Сервер реагирует посылкой клиенту тэга потока:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_234'
    from='example.com'
    version='1.0'>
```

Шаг 3: Сервер информирует клиента об имеющихся механизмах аутентификации:

```
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
</stream:features>
```

Шаг 4: Клиент выбирает механизм аутентификации:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='DIGEST-MD5'/>
```

Шаг 5: Сервер посылает закодированный вызов [BASE64] клиенту:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cmVhbG09InNvbWVyZWFsbSIsbm9uY2U9Ik9BNk1HOXRFUUdtMmhoIixxb3A9ImF1dGgi
LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1kNS1zZXNzCg==
</challenge>
```

Декодированный вызов имеет вид:

```
realm="somerealm",nonce="OA6MG9tEQGm2hh",\
qop="auth",charset=utf-8,algorithm=md5-sess
```

Шаг 5 (alt): Сервер возвращает клиенту сообщение об ошибке:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <incorrect-encoding/>
</failure>
</stream:stream>
```

Шаг 6: Клиент посылает закодированный отклик [BASE64] на вызов:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
dXNlcm5hbWU9InNvbWVub2RlIixyZWFsbT0ic29tZXJlYWxtIixub25jZT0i
T0E2TUc5dEVRR20yaGgiLGNub25jZT0iT0E2TUhYaDZWcVRyUmsiLG5jPTAw
MDAwMDAxLHFvcD1hdXRoLGRpZ2VzdC11cmk9InhtcHAvZXhhbXBsZS5jb20i
LHJlc3BvbnNlPWQzODhkYWQ5MGQ0YmJkNzYwYTE1MjMyMWYyMTQzYWY3LGNo
YXJzZXQ9dXRmLTgK
</response>
```

Декодированный отклик имеет вид:

```
username="somenode",realm="somerealm",\
nonce="OA6MG9tEQGm2hh",cnonce="OA6MHXh6VqTrRk",\
nc=00000001,qop=auth,digest-uri="xmpp/example.com",\
response=d388dad90d4bbd760a152321f2143af7,charset=utf-8
```

Шаг 7: Сервер посылает еще один закодированный [BASE64] вызов клиенту:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cnNwYXV0aD1lYTQwZjYwMzM1YzQyN2I1NTI3Yjg0ZGJhYmNkZmZmZAo=
</challenge>
```

Декодированный вызов имеет вид:

```
rspauth=ea40f60335c427b5527b84dbabcdfffd
```

Шаг 7 (alt): Сервер посылалает клиенту сообщение об ошибке:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <temporary-auth-failure/>
</failure>
</stream:stream>
```

Шаг 8: Клиент откликается на вызов:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Шаг 9: Сервер информирует клиента об успешной аутентификации:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Шаг 9 (alt): Сервер информирует клиента об ошибке аутентификации:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <temporary-auth-failure/>
</failure>
</stream:stream>
```

Шаг 10: Клиент формирует новый поток к серверу:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Шаг 11: Сервер откликается посылкой заголовка потока клиенту, сообщая о любых дополнительных имеющихся возможностях (если таких возможностей нет, посылается пустой элемент):

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_345'
    from='example.com'
    version='1.0'>
<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</stream:features>
```

Пример сервер-сервер см. в RFC по протоколу (http://webi.ru/webi_files/xmpp_protocol.html).

Подключение ресурсов

После согласования применения SASL с получателем, инициатор может хотеть или не нуждаться в подключении к потоку специфических ресурсов. В основном это относится только к клиентам: для того чтобы адаптироваться к формату адресации и правил доставки строф, специфицированных в данном документе. Должен существовать идентификатор ресурса, ассоциированный с <node@domain> клиента (который либо генерируется сервером, либо предлагается приложением клиента). Это гарантирует, то, что адрес используемый для данного потока, является "полным JID" формата <node@domain/resource>.

После получения информации об успехе согласования SASL, клиент должен послать серверу новый заголовок, на который сервер должен откликнуться заголовком потока и прислать список доступных возможностей потока. Точнее, если сервер требует, чтобы клиент после успешного согласования SASL подключил ресурс к потоку, он должен включить пустой элемент <bind/>, сопряженный с пространством имен 'urn:ietf:params:xml:ns:xmpp-bind', в список возможностей, которые он предлагает клиенту после отправки заголовка для потока-отклика, посланного после успешного согласования SASL (но не до):

Сервер анонсирует возможность подключения ресурса клиенту:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_345'
    from='example.com'
    version='1.0'>
<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
</stream:features>
```

После того как клиент оказывается проинформирован о необходимости подключения ресурса, он должен выполнить подключение ресурса к потоку, посредством посылки серверу IQ-строфы типа "set" (смотри "IQ семантика"), содержащей данные, соотнесенные с пространством имен 'urn:ietf:params:xml:ns:xmpp-bind'.

Если клиент хочет позволить серверу сформировать идентификатор ресурса, он посылает IQ-строфу типа "set", которая содержит пустой элемент <bind/>:

Клиент просит сервер подключить ресурс:

```
<iq type='set' id='bind_1'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
</iq>
```

Сервер, который поддерживает подключение ресурсов, должен быть способен генерировать идентификатор ресурса для клиента. Идентификатор ресурса, генерируемый сервером, должен быть уникальным для данного <node@domain>.

Если клиент хочет специфицировать идентификатор ресурса, он посылает IQ-строфу типа "set", которая содержит желаемый идентификатор ресурса в виде текстового элемента <resource/>, который является дочерним элементом <bind/>:

Клиент подключает ресурс:

```
<iq type='set' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
</iq>
```

Раз сервер сформировал идентификатор ресурса для клиента или воспринял идентификатор, предложенный клиентом, он должен послать клиенту IQ-строфу типа "result", которая должна содержать дочерний элемент <jid/>, который специфицирует полный JID для подключенного ресурса, как это определено сервером:

Сервер информирует клиента об успешном подключении ресурса:

```
<iq type='result' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <jid>somenode@example.com/someresource</jid>
  </bind>
</iq>
```

Сервер должен воспринять идентификатор ресурса, присланный клиентом, Но может заменить его на идентификатор, сгенерированный сервером; в этом случае, сервер не должен присылать клиенту строфу ошибки (например, <forbidden/>), а вместо этого прислать клиенту свой сгенерированный идентификатор ресурса в IQ результата, как было показано выше.

Когда клиент выдает идентификатор ресурса, возможна присылка следующей строфы условий ошибки (смотри "Строфы ошибок"):

- Представленный идентификатор ресурса не может обрабатываться сервером в соответствии с Resourceprep (см. приложение к описанию протокола).
- Клиенту не позволяется привязывать ресурсы к потоку (например, потому, что узел или пользователь достиг предельного числа подключаемых ресурсов).
- Предложенный идентификатор ресурса уже используется, а сервер не разрешает подключения нескольких ресурсов с одинаковыми идентификаторами.

Протокол обработки условия ошибки представлен ниже.

Идентификатор ресурса не может быть обработан при условии:

```
<iq type='error' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Клиент не разрешает подключение ресурса:

```
<iq type='error' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Идентификатор ресурса уже используется:

```
<iq type='error' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
  <error type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Если, до завершения этапа подключения ресурса клиент попытается послать XML-строфу, отличную от IQ-строфы с дочерним <bind/>, привязанным к пространству имен 'urn:ietf:params:xml:ns:xmpp-bind', сервер не должен обрабатывать строфу и ему следует вернуть клиенту строфу ошибки <not-authorized/>.

Метод сервера Dialback ("обратный дозвон")

Overview

Протоколы Jabber, которые адаптированы для XMPP, были приспособлены к использованию метода сервера "dialback" для защиты от фальсификации домена, таким образом делая сложным фальсификацию XML-строф. Обратный дозвон сервера (dialback) не является механизмом безопасности и связан исключительно со слабыми средствами верификации идентичности сервера (смотри "Коммуникации Сервер-сервер", где рассмотрены характеристики безопасности этого метода). Домены, требующие надежной безопасности, должны использовать TLS и SASL. Если для аутентификации сервер-сервер используется SASL, обратный дозвон применять не следует (просто бессмысленно).

Метод обратного дозвона сервера возможен благодаря наличию DNS (Domain Name System), так как сервер может выяснить наличие другого

сервера в пределах данного домена. Так как обратный дозвон зависит от DNS, междоменные коммуникации не должны запускаться, пока объекты не будут локализованы с помощью DNS сервера).

Обратный дозвон сервера является однонаправленным, по этой причине обеспечивает слабую верификацию идентичности для потока лишь в одном направлении. Так как обратный дозвон сервера не является механизмом аутентификации, двухсторонняя аутентификация здесь невозможна. Следовательно, чтобы реализовать двунаправленный обмен между доменами, обратный дозвон должен быть выполнен для обоих направлений.

Метод генерации и верификации ключей, используемый в dialback сервером должен учитывать используемые имена машин, идентификатор потока, формируемый сервером-приемником и секретный ключ, известный всем авторизованным в сети серверам. Идентификатор потока является критическим параметром безопасности в процедуре dialback и, следовательно, должен быть непредсказуемым и неповторимым (смотри [RANDOM]).

Любая ошибка, которая происходит во время согласования dialback должна рассматриваться как ошибка потока, которая приводит к прерыванию потока и разрыву TCP-соединения. Возможные условия ошибки специфицированы ниже в описании протокола.

Здесь используется следующая терминология:

- Originating Server — исходный сервер
- сервер, который пытается установить соединение между двумя доменами.
- Receiving Server — принимающий сервер
- сервер, который пытается проверить, что исходный сервер представляет тот самый домен, который он анонсирует.
- Authoritative Server — управляющий сервер
- сервер, который отвечает, руководствуясь именем, введенным исходным сервером; для базовой конфигурации это исходный сервер, но может быть и другая машина в сети исходного сервера.

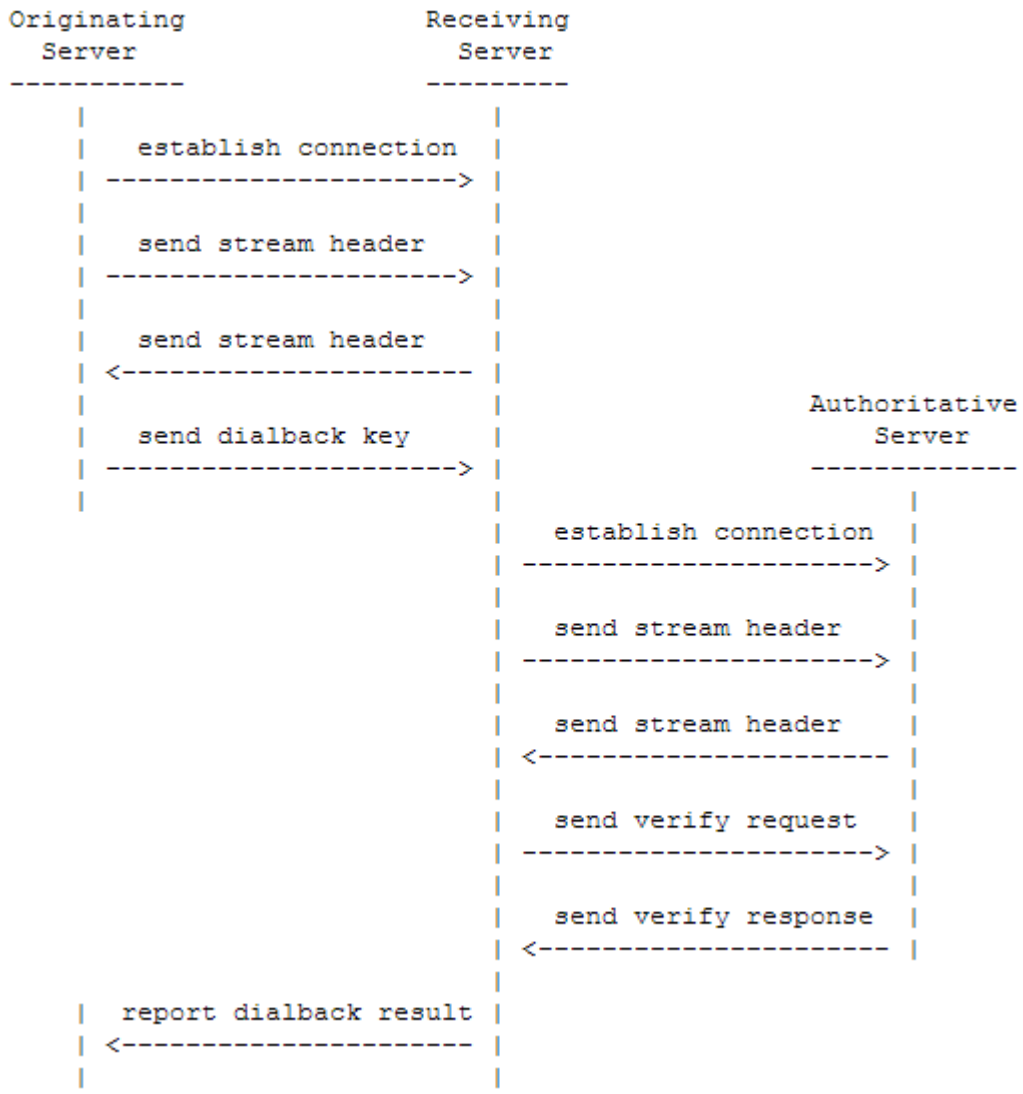
Порядок событий

Ниже представлено краткое описание последовательности событий при dialback:

1. Исходный сервер устанавливает соединение с принимающим сервером.
2. Исходный сервер посылает через соединение значение 'key' принимающему серверу.

3. Принимающий сервер устанавливает соединение с управляющим сервером.
4. Принимающий сервер посылает этот ключ управляющему серверу.
5. Управляющий сервер подтверждает или не подтверждает корректность ключа.
6. Принимающий сервер информирует исходный сервер, аутентифицирован он или нет.

Мы можем представить схему диалога следующим образом:



1. Исходный сервер устанавливает TCP-соединение с принимающим сервером.
2. Исходный сервер посылает заголовок потока принимающему серверу:

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
```

```
    xmlns='jabber:server'
    xmlns:db='jabber:server:dialback'>
```

Заметим: атрибуты 'to' и 'from' являются опционными для элемента корневого потока. Включение декларации пространства имен xmlns:db с приведенным именем указывает принимающему серверу, что исходный сервер поддерживает dialback. Если имя пространства имен некорректно, тогда принимающий сервер должен сформировать ошибку потока <invalid-namespace/> и завершить XML-поток, а также разорвать TCP-соединение.

3. Принимающий сервер должен отослать назад заголовок потока исходному серверу, включая уникальный ID этого взаимодействия:

```
<stream:stream
   xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:server'
   xmlns:db='jabber:server:dialback'
    id='457F9224A0...'>
```

Заметим: Если имя пространства имен некорректно, тогда исходный сервер должен сформировать ошибку потока <invalid-namespace/>, прервать поток и разорвать TCP-соединение. Принимающий сервер должен ответить, но может молча завершить XML-поток, в зависимости от принятой политики безопасности; однако, если принимающий сервер хочет продолжить, он должен послать исходному серверу заголовок потока.

4. Исходный сервер посылает ключ dialback принимающему серверу:

```
<db:result

   to='Receiving Server'
    from='Originating Server'>
    98AF014EDC0...
</db:result>
```

Замечание: Этот ключ не просматривается принимающим сервером, так как он не хранит информацию об исходном сервере по завершении сессии. Ключ, генерируемый исходным сервером, должен базироваться частично на значении ID, предоставленном принимающим сервером на предыдущем шаге, а также частично на секретном ключе, совместно используемом исходным и управляющим серверами. Если значение адреса 'to' не соответствует имени машины, распознанному принимающим сервером, тогда последний должен генерировать сообщение ошибки потока <host-unknown/> и прерывать XML-поток. Если значение адреса 'from' соответствует домену, с которым принимающий сервер уже установил соединение, тогда он должен поддерживать существующее соединение до тех пор, пока не будет установлено новое соединение. Кроме того, принимающий сервер может сформировать сообщение об ошибке потока <not-authorized/> для нового

соединения и затем прервать XML-поток и его TCP-соединение, связанные с новым запросом.

5. Принимающий сервер устанавливает TCP-соединение с доменным именем, представленным исходным сервером, как результ установления соединения с управляющим сервером.
6. Принимающий сервер посылает заголовок потока управляющему серверу:

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:server'
    xmlns:db='jabber:server:dialback'>
```

Замечание: атрибуты 'to' и 'from' являются опционными для элемента корневого потока. Если имя пространства имен некорректно, тогда управляющий сервер должен генерировать сообщение ошибки потока <invalid-namespace/> и прерывать XML-поток и TCP-соединение.

7. Управляющий сервер посылает заголовок потока принимающему серверу:

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:server'
    xmlns:db='jabber:server:dialback'
    id='1251A342B...'>
```

Замечание: Если имя пространства имен некорректно, тогда принимающий сервер должен сформировать сообщение ошибки потока <invalid-namespace/> и прервать XML-поток и соответствующее TCP-соединение между ним и сервером управления. Если ошибка потока случится между принимающим и управляющим серверами, тогда принимающий сервер должен сформировать сообщение об ошибке потока <remote-connection-failed/> и прервать поток и TCP-соединение с исходным сервером.

8. Принимающий сервер посылает управляющему серверу запрос верификации ключа:

```
<db:verify
    from='Receiving Server'
    to='Originating Server'
    id='457F9224A0...'>
  98AF014EDC0...
</db:verify>
```

Замечание: Здесь уже получены имена машин, оригинальный идентификатор из заголовка потока принимающего сервера исходному серверу на шаге 3, и ключ, который исходный сервер послал принимающему серверу на шаге 4. На основе этой информации, а также на секретном, совместно используемом ключе сети управляющего сервера ключ

верифицируется. Для генерации ключа может использоваться любой метод верификации. Если значение адреса 'to' не соответствует имени машины, выявленному управляющим сервером, тогда управляющий сервер должен сформировать сообщение об ошибке потока <host-unknown/> и разорвать XML-поток и TCP-соединение. Если значение адреса 'from' не соответствует имени машины, представленному принимающим сервером при открытии TCP-соединения, тогда управляющий сервер должен сформировать ошибку потока <invalid-from/>.

9. Управляющий сервер проверяет, является ли ключ корректным:

```
<db:verify
    from='Originating Server'
    to='Receiving Server'
    type='valid'
    id='457F9224A0...'/>
```

или

```
<db:verify
    from='Originating Server'
    to='Receiving Server'
    type='invalid'
    id='457F9224A0...'/>
```

Замечание: Если ID не соответствует тому, который предложен принимающим сервером на шаге 3, тогда принимающий сервер должен сгенерировать ошибку потока <invalid-id/> и прервать XML-поток и соответствующее TCP-соединение. Если значение адреса 'to' не соответствует имени машины, распознанному принимающим сервером, тогда принимающий сервер должен выработать ошибку потока <host-unknown/> и разорвать TCP-соединение. Если значение адреса 'from' не соответствует имени машины, представленному исходным сервером при открытии TCP-соединения, тогда принимающий сервер должен выработать ошибку потока <invalid-from/> и разорвать TCP-соединение. После возвращения флага верификации принимающему серверу, управляющий сервер должен прервать поток между ними.

10. Принимающий сервер информирует исходный сервер о результате:

```
<db:result
    from='Receiving Server'
    to='Originating Server'
    type='valid'/>
```

Замечание: В этой точке, соединение либо верифицировано type='valid', либо объявлено некорректным. Если соединение некорректно, принимающий сервер должен прервать XML-поток и разорвать TCP-соединение. Если соединение верифицировано, исходный сервер начинает посылать данные,

которые читаются принимающим сервером; перед этим все XML-строфы, посланные принимающему серверу должны быть молча отброшены.

Результатом всего предшествующего является то, что принимающий сервер проверил идентичность исходного сервера, таким образом управляющий сервер может посылать, а принимающий сервер принимать XML-строфы через "исходный поток" (т.е., поток от исходного к принимающему серверу). Для того чтобы верифицировать идентичности объектов, используя "поток-отклик" (т.е., поток от принимающего сервера к исходному серверу), должен быть осуществлен также dialback ("обратный дозвон") в обратном направлении.

После успешного выполнения процедуры dialback, принимающий сервер должен воспринять последующие пакеты <db:result/> (например, запросы валидации, посланные субдомену или другой машине, обслуживаемой принимающим сервером) от исходного сервера через имеющееся верифицированное соединение.

Даже если согласование dialback успешно, сервер должен проверить, что все XML-строфы полученные от другого сервера содержат атрибуты 'from' и 'to'; если строфа не следует этому ограничению, сервер, который получает строфу должен сформировать сообщение об ошибке потока <improper-addressing/> и разорвать TCP-соединение. Кроме того, сервер должен проверить, что атрибут 'from' строфы, полученной от другого сервера, содержит верифицированное имя домена для потока; если строфа не отвечает этим ограничениям, сервер, который получает строфу, должен сформировать ошибку потока <invalid-from/> и разорвать соединение.

XML-строфы

После согласования применения TLS, SASL и подключения ресурсов, если требуется, можно начать пересылку через поток XML-строф. Определены три типа XML-строф для пространств имен 'jabber:client' и 'jabber:server': <message/>, <presence/> и <iq/>. Кроме того, существует пять общих атрибутов для трех видов строф. Ниже описаны эти общие атрибуты, так же как базовая семантика трех видов строф, более детальную информацию о синтаксисе XML-строф и их применении можно найти в [XMPP-IM].

Общие атрибуты

Следующие пять атрибутов являются общими для сообщений, данных о присутствии и IQ-строф:

Атрибут 'to'

Атрибут 'to' специфицирует JID получателя строфы.

В пространстве имен 'jabber:client', строфа должна иметь атрибут 'to', хотя строфа, посланная от клиента серверу, для обработки сервером (например, данные о присутствии, посланные серверу для широковещательной рассылки другим объектам) не должна содержать атрибут 'to'.

В пространстве имен 'jabber:server' строфа должна иметь атрибут 'to'; если сервер получает строфу, которая не отвечает этим ограничениям, он должен сформировать ошибку потока <improper-addressing/> и разорвать соединение.

Если значение атрибута 'to' некорректно, объект, выяснивший этот факт, должен вернуть соответствующую ошибку отправителю, установив атрибут 'from' строфы-ошибки равной содержимому атрибуту 'to' исходной строфы.

Атрибут 'from'

Атрибут 'from' специфицирует JID отправителя.

Когда сервер получает XML-строфа в контексте аутентифицированного потока, сопряженного с пространством имен 'jabber:client', он должен сделать следующее:

1. Проверить, что значение атрибута 'from', предложенное клиентом, соответствует подключенному ресурсу для ассоциированного объекта
2. Добавить адрес 'from' к строфе, чье значение равно чистому JID (<node@domain>) или полный JID (<node@domain/resource>), определенный сервером для подключенного ресурса, который сформировал строфу (смотри "Определение адресов")

Если клиент пытается послать XML-строфу, для которой значение атрибута 'from' не соответствует одному из подключенных ресурсов для данного объекта, сервер должен прислать клиенту ошибку потока <invalid-from/>. Если клиент пытается послать XML-строфу через поток, который еще не аутентифицирован, сервер должен вернуть клиенту ошибку потока <not-authorized/>. Если имеют место оба эти условия, со соединение прерывается. Это помогает исключить атаку отказа в обслуживании, запущенную недобросовестным клиентом.

Когда сервер генерирует строфу от своего имени для доставки подключенному клиенту (например, в контексте сервиса памяти,

предоставляемого сервером клиенту), строфа либо не должна включать в себя атрибут 'from' либо включать атрибут 'from', чье значение является чистым JID аккоунта (<node@domain>) или полным JID клиента (<node@domain/resource>). Сервер не должен посылать клиенту строфу без атрибута 'from', если строфа не была сформирована самим сервером. Когда клиент получает строфу, которая не содержит атрибута 'from', он должен полагать, что строфа пришла от сервера, с которым клиент соединен.

В пространстве имен 'jabber:server', строфа должна иметь атрибут 'from'; если сервер получает строфу, которая не отвечает этому ограничению, он должен генерировать ошибку потока <improper-addressing/>. Кроме того, секция доменного идентификатора JID, содержащаяся в атрибуте 'from', должна соответствовать имени машины отправляющего сервера (или любой проверенный домен, такой как субдомен сервера-отправителя). Если сервер получает строфу, которая не отвечает этому ограничению, он должен сформировать сообщение об ошибке <invalid-from/>.

Атрибут 'id'

Опционный атрибут 'id' может использоваться объектом-отправителем для внутреннего отслеживания строф, которые он отправляет и получает (в частности для отслеживания взаимодействий типа запрос-отклик, типичных для семантики IQ-строф). Для атрибута 'id' глобальная уникальность является опционной в рамках домена или потока.

Атрибут 'type'

Атрибут 'type' специфицирует данные о целях или контексте сообщения, данные о присутствии или IQ-строфы. Конкретные допустимые значения атрибута 'type' сильно зависят от того, является ли строфа сообщением, данными о присутствии или IQ. Значения для строф сообщений и данных о присутствии являющиеся специфичными для технологии сообщений реального времени, они описаны в [XMPP-IM], тогда как значения для IQ-строф специфицируют роль IQ-строф в структурированном диалоге запрос-отклик (раздел 9). Единственным общим для всех трех типов строф является ошибка; смотри "Ошибки строф".

Атрибут 'xml:lang'

Строфа должна иметь атрибут 'xml:lang', если строфа содержит символьные данные, которые предназначены для обычного пользователя-человека (как описано в RFC 2277 [CHARSET], "Интернационализация для людей"). Значение атрибут 'xml:lang' специфицирует язык по умолчанию любых символьных данных, которое может быть переписано атрибутом 'xml:lang' дочернего элемента. Если строфа не имеет атрибута 'xml:lang', реализация должна предполагать, что действует значение языка по

умолчанию, специфицированное для потока, как это определено выше. Значение атрибута 'xml:lang' должно быть NMTOKEN и должно соответствовать формату, определенному в RFC 3066 [LANGTAGS].

Базовая семантика

Семантика сообщений

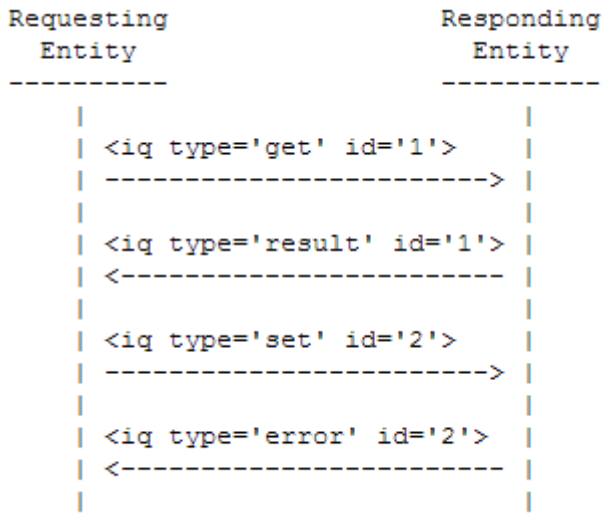
Тип строф <message/> можно рассматривать как "push"-механизм, поскольку один объект "проталкивает" информацию другому объекту, подобно коммуникациям, которые осуществляются в системах типа электронной почты. Все строфы сообщений имеют атрибут 'to', который специфицирует получателя сообщения. После получения такой строфы сервер должен переадресовать или доставить ее адресату (смотри "Серверные правила для обработки XML-строф")

Семантика присутствия

Элемент <presence/> может рассматриваться как механизм широковещательного уведомления, посредством чего многие объекты получают информацию об объекте, на которую они подписались (в данном случае информация о доступности сети). Вообще, уведомляющий объект должен послать строфу присутствия без атрибута 'to', серверу, который должен широковещательно переслать ее всем объектам-подписчикам. Однако уведомляющий объект может также послать строфу присутствия с атрибутом 'to', в этом случае сервер должен обеспечить доставку строфы конкретному адресату. Общие правила переадресации и доставки строф смотри в главе "Серверные правила обработки XML-строф" и в [XMPP-IM].

Семантика IQ

Info/Query, или IQ, представляет собой механизм запросов-откликов, сходный в некотором роде с [HTTP]. Семантика IQ делает возможными для объекта отправку запросов и получение откликов другого объекта. Информационное содержание запроса и отклика определяется декларацией пространства имен дочернего объекта IQ-элемента, а диалог отслеживается запрашивающим объектом с помощью атрибута 'id'. Таким образом, за стандартным обменом структурированными данными, такими как get/result или set/result следует IQ-диалог (хотя в отклике на запрос при определенных условиях может быть прислано сообщение об ошибке):



Для того чтобы усилить данную семантику, применены следующие правила:

1. Для IQ-строф необходим атрибут 'id'.
2. Для IQ-строф необходим атрибут 'type'. Значение должно быть из числа перечисленных ниже.
    - get - строфа является запросом информации или требований.
    - set - строфа предоставляет необходимые данные, устанавливает новые значения или замещает существующие величины.
    - result - строфа является откликом на успешный запрос get или set.
    - error - произошла ошибка, связанная с обработкой или доставкой выданной перед этим строфы (get или set). (смотри "Строфы ошибок" .
3. Объект, который получает IQ-запрос типа "get" или "set" должен ответит IQ-откликом типа "result" или "error" (отклик должен сохранить атрибут 'id' запроса).
4. Объект, который получает строфу типа "result" или "error" не должен откликаться на строфу посылкой следующего IQ-отклика типа "result" или "error"; однако, как показано выше, запрашивающий объект может послать еще один запрос (например, IQ типа "set" для того чтобы предоставить нужную информацию, выявленную в результате обмена get/result).
5. IQ-строфа типа "get" или "set" должна содержать один и только один дочерний элемент, который специфицирует семантику конкретного запроса или отклика.
6. IQ-строфа типа "result" должна содержать нуль или один дочерний элемент.
7. IQ-строфа типа "error" должна содержать дочерний элемент, содержащийся в соответствующем "get" или "set" и должна включать в себя дочерний элемент <error/>. Подробности смотри в главе "ошибки строф"

Ошибки строф

Ошибки, сопряженные со строфами обрабатываются аналогично ошибкам потока. Однако, в отличии от ошибок потока, ошибки строф являются исправимыми; следовательно ошибки строф содержат намеки на действия, которые следует предпринять исходному отправителю, для того чтобы исправить ошибку.

Правила

В отношении ошибок, связанных со строфами, используются следующие правила:

- Принимающий или обрабатывающий объект, который детектирует состояние ошибки в отношении строфы, должен прислать отправителю строфу того же сорта (сообщение, присутствие или IQ), чей атрибут 'type' установлен равным "error" (такая строфа называется "строфой ошибки").
- Объект, который генерирует строфу ошибки должен включить исходный посланный XML, так чтобы отправитель мог проверить, если необходимо, корректность XML, прежде чем предпринимать попытку повторной передачи.
- Строфа-ошибка должна содержать дочерний элемент <error/>.
- Дочерний элемент <error/> не должен включаться, если атрибут 'type' имеет значение, отличное от "error" (или если атрибута 'type' нет вообще).
- Объект, который получает ошибку строфы, не должен реагировать на строфу с еще одной ошибкой; это предотвращает зацикливание.

Синтаксис

```
<stanza-kind to='sender' type='error'>
  [RECOMMENDED to include sender XML here]
  <error type='error-type'>
    <defined-condition xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <text xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'
          xml:lang='langcode'>
      OPTIONAL descriptive text
    </text>
    [OPTIONAL application-specific condition element]
  </error>
</stanza-kind>
```

Вид строфы может принимать значение message (сообщение), presence (присутствие) или iq.

Значение элемента <error/> атрибута 'type' должно принимать одно из следующих значений:

- cancel - не предпринимать повторных попыток (ошибка неисправима)

- continue - продолжать (это всего лишь предупреждение)
- modify - повторить попытку после изменения посылаемых данных
- auth - повторить попытку после изменения параметров авторизации
- wait - повторить попытку после выдержки (ошибка является временной)

Элемент <error/>:

- должен содержать дочерний элемент, соответствующий одному из условий, специфицированных ниже; этот элемент должен соотноситься с пространством имен 'urn:ietf:params:xml:ns:xmpp-stanzas'.
- Может содержать дочерний элемент <text/>, который несет в себе символьные XML-данные, которые описывают ошибку более детально. Этот элемент должен соотноситься с пространством имен 'urn:ietf:params:xml:ns:xmpp-stanzas' и должен иметь атрибут 'xml:lang' .
- Может содержать дочерний элемент для ошибок, специфических для приложения; этот элемент должен соотноситься с пространством имен приложения, а его структура определяется этим пространством имен.

Элемент <text/> является опционным. Когда включается, он должен использоваться только для передачи описательной и диагностической информации, которая является дополнительной и служит для пояснения специфических условий приложения. Он не должен интерпретироваться приложением программно. Он не должен использоваться в качестве сообщения об ошибке пользователю.

Наконец, для обеспечения обратной совместимости, схема (специфицированная в [XMPP-IM]) позволяет опционное включение атрибута 'code' элемента <error/>.

Определенные условия

Определены следующие условия использования ошибок строф.

- <bad-request/> - отправитель послал XML с некорректным форматом или который не может быть обработан (например, IQ-строфа, которая включает в себя нераспознаваемое значение атрибут 'type'); сопряженным типом ошибки должен быть "modify".
- <conflict/> - доступ не может быть предоставлен, так как существует ресурс или сессия с тем же именем или адресом; сопряженным типом ошибки должен быть "cancel".
- <feature-not-implemented/> - запрошенная возможность не реализуется получателем или сервером и, следовательно, не может быть осуществлена; сопряженным типом ошибки должен быть "cancel".

- <forbidden/> - запрошенный объект не имеет требующегося разрешения для выполнения данного действия; сопряженным типом ошибки должен быть "auth".
- <gone/> - получатель или сервер не может более контактировать с этим адресом (строфа ошибки может содержать новый адрес в виде символьных XML-данных элемента <gone/>); сопряженным типом ошибки должен быть "modify".
- <internal-server-error/> - сервер не может обработать строфу, из-за неверной конфигурации или неопределенной внутренней ошибки сервера; сопряженным типом ошибки должен быть "wait".
- <item-not-found/> - адресованный JID или запрошенный элемент не может быть найден; сопряженным типом ошибки должен быть "cancel".
- <jid-malformed/> - посылающий объект предоставил или переслал XMPP-адрес (например, значение атрибута 'to') или нечто в этом же роде (например, идентификатор ресурса), который не стыкуется с синтаксисом, определенным в главе "Схема адресации" ; сопряженным типом ошибки должен быть "modify".
- <not-acceptable/> - получатель или сервер понимает запрос, но отказывается обрабатывать его, так как он не отвечает критериям, определенным получателем или сервером (например, локальная политика в отношении приемлемых слов в сообщениях); сопряженным типом ошибки должен быть "modify".
- <not-allowed/> - получатель или сервер не позволяют любому объекту выполнять операцию; сопряженным типом ошибки должен быть "cancel".
- <not-authorized/> - отправитель должен предоставить правильные параметры авторизации, прежде чем ему будет позволено выполнить соответствующеие действия, или он выдал неверные параметры авторизации; сопряженным типом ошибки должен быть "auth".
- <payment-required/> - запрашивающий объект не авторизован для доступа к запрашиваему сервису, так как требуется оплата; сопряженным типом ошибки должен быть "auth".
- <recipient-unavailable/> - адресат временно недоступен; сопряженным типом ошибки должен быть "wait" (заметим: приложение не должно возвращать эту ошибку, если, делая это, выдаст информацию о сетевой доступности адресата объекту, неавторизованному для получения такой информации).
- <redirect/> - получатель или сервер переадресуют запрос для этой информации другому объекту, обычно временно (ошибочная строфа должна содержать альтернативный адрес, который должен быть корректным JID, в текстовом элементе <redirect/>); сопряженным типом ошибки должен быть "modify".

- <registration-required/> - запрашиваемый объект не авторизован для доступа к запрошенному сервису, так как требуется регистрация; сопряженным типом ошибки должен быть "auth".
- <remote-server-not-found/> - удаленный сервер или сервис, специфицированный в качестве части или всего JID получателя, не существует; сопряженным типом ошибки должен быть "cancel".
- <remote-server-timeout/> - удаленный сервер или сервис, специфицированный в качестве части или всго JID получателя, не досягаемы в пределах разумного времени; сопряженным типом ошибки должен быть "wait".
- <resource-constraint/> - получатель или сервер имеют мало системных ресурсов для обслуживания запроса; сопряженным типом ошибки должен быть "wait".
- <service-unavailable/> - получатель или сервер в настоящее время не предоставляют запрашиваемого сервиса; сопряженным типом ошибки должен быть "cancel".
- <subscription-required/> - запрашиваемый объект не авторизован для доступа к запрашиваемого сервиса, так как требуется подписка; сопряженным типом ошибки должен быть "auth".
- <undefined-condition/> - условие ошибки не совпадает ни с одним, описанным в этом перечне; любой тип ошибки может быть ассоциирован с этим условием, и это условие может использоваться только совместно с условием, специфичным для приложения.
- <unexpected-request/> - получатель или сервер понимают запрос, но не ожидали его в данный момент; сопряженным типом ошибки должен быть "wait".

Условия специфические для приложения

Как было замечено, приложение может выдавать специфическую информацию строф ошибок путем включения дочернего элемента ошибки, соответственно привязанного к пространству имен. Специфический элемент приложения должен дополнять или прояснять определенный элемент ошибки. Таким образом, элемент <error/> будет содержать два или три дочерних элемента:

```
<iq type='error' id='some-id'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <too-many-parameters xmlns='application-ns'/>
  </error>
</iq>

<message type='error' id='another-id'>
  <error type='modify'>
    <undefined-condition
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <text xml:lang='en'
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      Some special application diagnostic information...
    </text>
    <special-application-condition xmlns='application-ns'/>
  </error>
</message>
```

Правила сервера для обработки XML-строф

Адаптивные реализации сервера должны гарантировать надлежащую обработку XML-строф между любыми двумя объектами.

За пределами требований нормальной обработки реализация каждого сервера будет содержать свои собственные "деревья доставки" для обслуживания приходящих к ним строф. Такое дерево определяет, нуждается ли строфа в переадресации в другой домен, будет ли обработана локально или доставлена ресурсу, сопряженному с соседним узлом. Здесь используются следующие правила:

Нет адреса 'to'

Если строфа не имеет атрибута 'to', сервер должен обработать ее для объекта, ее пославшего. Так как все строфы, полученные от других серверов, должны иметь атрибут 'to' , это правило приложимо только к строфам, полученных от зарегистрированных объектов (таких как клиент), т.е. подключенных к серверу. Если сервер получает строфу присутствия без атрибута 'to', сервер должен послать ее широковещательно объектам, которые подписаны на посылку данных присутствия, если это возможно (семантика широковещательных данных для обмена сообщениями и данными присутствия определена в [XMPP-IM]). Если сервер получает IQ-строфу типа "get" или "set" без атрибута 'to' и он понимает пространство имен, которое соотнесено с содержимым строфы, он должен либо обработать строфу от имени отправителя (где под словом "обработать" подразумевается семантика соответствующего пространства имен) или прислать отпраителю сообщение об ошибке

Чужой домен

Если имя машины секции идентификатора домена JID содержит атрибут 'to', который не согласуется со сконфигурированными именами самого сервера или субдомена, сервер должен переадресовать строфу внешнему домену. Возможны два случая:

Между двумя доменами уже имеется поток сервер-сервер: существуют маршруты строф сервера к управляющему серверу для внешнего домена через существующий поток.

Между двумя доменами не существует потока сервер-сервер:

1. Выясняет имя машины для внешнего домена (как это определено для коммуникаций сервер-сервер),
2. Согласует поток сервер-сервер между двумя доменами (как это определено для применения TLS и использует SASL, и
3. Переадресует строфу управляющему серверу для внешнего домена через вновь созданный поток.

Если маршрутизация до сервера получателя не удалась, сервер должен прислать сообщение об ошибке отправителю; если сервер получателя достижим, но доставка строфы сервером получателя невозможна, сервер получателя должен прислать сообщение ошибки отправителю.

Субдомен

Если имя машины секции идентификатора домена JID, содержащегося в атрибуте 'to', соответствует субдомену одного из сконфигурированных имен самого сервера, сервер должен либо обработать строфу сам, либо переадресовать ее специализированному сервису, который ответственен за субдомен (если субдомен сконфигурирован), или послать уведомление об ошибке отправителю (если субдомен не сконфигурирован).

Полный домен или специфические ресурсы

Если имя машины секции идентификатора домена JID, содержащегося в атрибуте 'to', соответствует сконфигурированному имени самого сервера, а JID, содержащий атрибут 'to', имеет формат <domain> или <domain/resource>, сервер (или определенный ресурс) должен либо обрабатывать строфу или возвращать отправителю ошибку.

Узел в том же домене

Если имя машины секции идентификатора домена JID, содержащееся в атрибуте 'to', соответствует субдомену одного из сконфигурированных имен

самого сервера, а JID, содержащий атрибут 'to', имеет формат <node@domain> или <node@domain/resource>, сервер должен доставить строфу адресату строфы, согласно JID, содержащемуся в атрибуте 'to'. Используются следующие правила:

1. Если JID содержит идентификатор ресурса (т.е., имеет форму <node@domain/resource>) и существует подключенный ресурс, который соответствует полному JID, сервер получателя должен вставить строфу в поток или сессию, которая соответствует идентификатору ресурса.
2. Если JID содержит идентификатор ресурса и не существует подключенного ресурса, который соответствует полному JID, сервер получателя должен прислать отправителю ошибку строфы <service-unavailable/>.
3. Если JID имеет формат <node@domain> и в узле существует, по крайней мере, один подключенный ресурс, сервер получателя должен доставить строфу одному из подключенных ресурсов, согласно правилам приложения (набор правил доставки сообщения и данных о положении определен в [XMPP-IM]).

Использование XML в XMPP

Ограничения

XMPP является упрощенным и специализированным протоколом для потоковых элементов XML, для того чтобы обмениваться структурированной информацией в режиме близком к реальному времени. Так как XMPP не требует парсинга произвольных XML-документов, не существует никаких требований, которые должен поддерживать XMPP [XML]. В частности, используются следующие ограничения.

Что касается XML-генерации, XMPP-реализации не должны вводить в XML-поток следующие вещи:

- Комментарии;
- инструкции обработки;
- внутренние или внешние DTD-субнаборы;
- внутренние или внешние ссылки на объекты за исключением заранее определенных сущностей;
- символьные данные или значения атрибута, содержащие символы без эскэйпов, которые могут указывать на предопределенные объекты; следует избегать использования таких символов

С учетом XML-обработки, если XMPP-реализация получает такие запрещенные данные, она должна их игнорировать.

Имена XML-пространства имен и префиксов

Пространство имен XML [XML-NAMES] используется в рамках XMPP-подобных XML, чтобы сформировать четкие границы принадлежности данных. Основной функцией пространства имен является разделение различных словарей XML-элементов, которые структурно перемешаны. Гарантия того, что XMPP-подобные XML знакомы с пространством имен, делает возможным структурное перемешивание любых допустимых XML с любыми элементами данных в рамках XMPP. Правила для пространства имен XML и префиксов определены в следующих подразделах.

Пространство имен потоков

Декларацией пространства имен потока для всех заголовков XML-потока является обязательной. Имя пространства имен потока должно быть 'http://etherx.jabber.org/streams'. Элемент имен элемента <stream/> и его дочерних элементов <features/> и <error/> должны быть заданы во всех случаях префиксом пространства имен потоков. Реализация должна генерировать для этих элементов только префикс 'stream:'.

Пространство имен по умолчанию

Декларация пространства имен по умолчанию является обязательной, она используется во всех XML-потоках, для того чтобы определить для корневого потока допустимые дочерние элементы первого уровня. Эта декларация пространства имен должна быть идентичной для исходного потока и для потока-отклика, так чтобы оба потока были совместимыми. Декларация пространства имен по умолчанию применяется для потока и всех строф, посланных в рамках потока.

Реализация сервера должна поддерживать следующие два пространства имен по умолчанию:

- jabber:client - это пространство имен по умолчанию декларируется, когда поток используется для коммуникаций между клиентом и сервером.
- jabber:server - это пространство имен по умолчанию декларируется, когда поток используется для коммуникаций между двумя серверами.

Реализация клиента должна поддерживать пространство имен по умолчанию 'jabber:client'.

Реализация не должна генерировать префиксы пространства имен для элементов в пространстве имен по умолчанию, если это пространство имен 'jabber:client' или 'jabber:server'. Реализация не должна генерировать префиксов пространства имен для элементов, квалифицируемых по

содержанию (в противоположность потоку) пространства имен, отличного от 'jabber:client' и 'jabber:server'.

Замечание: Пространства имен 'jabber:client' и 'jabber:server' являются почти идентичными, но используются в различных контекстах (коммуникации клиент-сервер для 'jabber:client' и коммуникации сервер-сервер для 'jabber:server'). Единственное отличие между ними заключается в том, что атрибуты 'to' и 'from' являются опционными для строф, посланных в рамках 'jabber:client', в то время как они обязательны для строф, посланных в рамках 'jabber:server'. Если реализация принимает поток, который соотносится с пространствами имен 'jabber:client' или 'jabber:server', она должна поддерживать общие атрибуты и базовую семантику всех трех главных видов строф (сообщение, присутствие и IQ).

Пространство имен Dialback

Декларация пространства имен dialback является обязательной для всех элементов, используемых в серверном dialback. Имя пространства имен dialback должно быть 'jabber:server:dialback'. Все элементы, соотнесенные с этим пространством имен, должны иметь префикс. Реализация должна генерировать для таких элементов только префикс 'db:' и может воспринимать только префикс 'db:'.

Валидация

За исключением, как отмечено в отношении адресов 'to' и 'from' для строф из пространства имен 'jabber:server', сервер не несет ответственности за валидацию XML-элементов, переадресуемых клиенту или другому серверу. Конкретная реализация может решить предоставлять только проверенные информационные элементы, но это является опционным (хотя реализация не должна воспринимать XML, которые плохо сформатированы). Клиенты не должны полагаться на возможность посылки данных, не следующих заданным формам, и должны игнорировать любые нестандартные элементы или атрибуты во входном XML-потоке. Валидация XML-потоков и строф является опционной.

Включение текстовых деклараций

Реализации должны посылать текстовые декларации до посылки заголовка потока. Приложения должны следовать правилам из [XML], в зависимости от обстоятельств, при которых включается текстовая декларация.

Кодирование символов

Реализации должны поддерживать UTF-8 (RFC 3629 [UTF-8]) преобразования универсального символьного набора (ISO/IEC 10646-1

[UCS2]), как этого требует RFC 2277 [CHARSET]. Реализации не должны пытаться использовать любые другие виды кодирования.

Основные требования совместимости

Этот раздел суммирует специфические аспекты протокола XMPP, которые должны поддерживаться серверами и клиентами, для того чтобы быть совместимыми. Для целей совместимости мы проводим различие между базовыми протоколами (которые должны поддерживаться любым сервером или клиентом, вне зависимости от специфичности приложений) и протокола отправки сообщений в реальном масштабе времени (IM) (которые должны поддерживаться IM и приложениями присутствия, работающими поверх базовых протоколов). Требования совместимости, которые приложимы ко всем серверам и клиентам, специфицированы ниже. Требования совместимости для IM серверов и клиентов описаны в разделе [XMPP-IM].

Серверы

В дополнение ко всем определенным требованиям с учетом безопасности сервер должен поддерживать следующие базовые протоколы:

- Применение профайлов [NAMEPREP], Nodeprep (Приложение A) и Resourceprep (Приложение B) [STRINGPREP] к адресам (включая гарантию того, что идентификаторы доменов являются международными доменными именами, как это определено в [IDNA])
- XML-потоки, включая использование TLS, использование SASL и подключение ресурсов
- Базовая семантика трех основных видов строф (т.е., <message/>, <presence/> и <iq/>).
- Генерация синтаксиса ошибок и семантик, относящаяся к потокам, TLS, SASL и XML-строфам

Кроме того сервер может поддерживать серверный dialback

Клиенты

Клиент должен поддерживать следующие базовые протоколы, чтобы считаться адекватным:

- XML-потоки, включая использование TLS, использование SASL и подключение ресурсов.
- Базовая семантика трех определенных видов строф (т.е., <message/>, <presence/> и <iq/>), как это специфицировано в семантике строф.
- Обработка (и, где возможно, генерация) синтаксиса ошибок и семантики, сопряженной с потоками, TLS, SASL и XML-строфами.

Кроме того, клиент должен поддерживать следующие базовые протоколы:

- Генерация адресов, в которых профайлы [NAMEPREP], Nodeprep и Resourceprep [STRINGPREP] могут успешно работать.

Порядок слоев

Порядок протокольных слоев в стеке должен быть следующим:

1. TCP
2. TLS
3. SASL
4. XMPP

Разумность такого порядка заключается в том, что он является основой [TCP] упорядочения для всего стека протоколов, работающих поверх TCP.

Имя сервиса GSSAPI

IANA зарегистрировала "xmpp" в качестве имени сервиса GSSAPI [GSS-API], как это задано в рамках определения SASL.

Номера портов

IANA зарегистрировала ключевые слова xmpp-client и xmpp-server для обозначения номеров портов для TCP-обмена 5222 и 5269, соответственно. Эти номера портов следует использовать для обменов клиент-сервер и сервер-клиент.

Соображения интернационализации, соображения безопасности, соображения IANA см. в RFC к протоколу.

Задание к лабораторной работе:

Изучить протокол обмена сообщениями и данными о присутствии XMPP, разработать простейшее клиент-серверное программное обеспечение «Мессенджер».

Требование к отчету:

1) Представить краткое описание используемого или собственно разработанного протокола обмена сообщениями для клиент-серверного ПО «Мессенджер».
2) Представить основной программный код клиент-серверного приложения.
3) Представить экранные формы работы приложения.

Контрольные вопросы:

4) Основные принципы работы протокола xmpp?
5) Какие порты используются при работе приложений на базе xmpp?
6) Какие средства защиты предусмотрены в протоколе xmpp?
7) Каким образом хранятся сообщения в приложениях типа «Мессенджер»?
8) Какой транспортный протокол используется в Вашем приложении?
9) Какое кол-во клиентов могут одновременно подключаться к серверу, от чего это зависит?
10) Как избежать «отказа в обслуживании» сервера при большом кол-ве подключенных клиентов?

Лабораторная работа №7

Тема: Исследование протоколов беспроводной сети

Цель работы: Ознакомиться со структурой протоколов, принципами организации и работы в беспроводных сетях 802.11b и 802.11g

Методические указания к лабораторной работе:

1. Общее описание протоколов

Рассмотрим все существующие стандарты IEEE 802.11, которые предусматривают использование определенных методов и скоростей передачи данных, методов модуляции, мощности передатчиков, полос частот, на которых они работают, методов аутентификации, шифрования.

С самого начала сложилось так, что некоторые стандарты работают на физическом уровне, некоторые - на уровне среды передачи данных, а другие - на высших уровнях модели взаимодействия открытых систем ISO / OSI.

Существуют следующие группы стандартов:

• IEEE 802.11a, IEEE 802.11b и IEEE 802.11g описывают работу сетевого оборудования (физический уровень)

• IEEE 802.11d, IEEE 802.11e, IEEE 802.11i, IEEE 802.11j, IEEE 802.11h и IEEE 802.11r - параметры среды, частоты радиоканала, средства безопасности, способы передачи мультимедийных данных;

• IEEE 802.11f и IEEE 802.11c - принцип взаимодействия точек доступа между собой, работу радиомостов.

1.1. Стандарт IEEE 802.11

Стандарт IEEE 802.11 был «первенцем» среди стандартов беспроводной сети. Работу над ним начали еще в 1990 году. Как и полагается, этим занималась рабочая группа из IEEE, целью которой было создание единого стандарта для радиооборудования, которое работало на частоте 2,4 ГГц. При этом ставилась задача достичь скорости 1 и 2 Мбит/с при использовании методов DSSS (англ. Direct Sequence Spread Spectrum) и FHSS (англ. Frequency Hopping Spectrum Spreading) соответственно. Работа над созданием стандарта закончилась через 7 лет. Цель была достигнута, но скорость, которую обеспечивал новый стандарт, оказалась очень маленькой для современных нужд. Поэтому рабочая группа по IEEE начала разработку новых, более скоростных, стандартов.

Разработчики стандарта 802.11 учитывали особенности ячеечной архитектуры системы. Потому что волны распространяются в разные стороны на определенный радиус. Так что внешне зона напоминает элементарную ячейку. Работа каждой ячейки осуществляется под

управлением базовой станции, в качестве которой используется точка доступа. Часто ячейку называют базовой зоной обслуживания.

Чтобы базовые зоны обслуживания могли общаться между собой, существует специальная распределительная система (Distribution System, DS). Недостатком распределительной системы стандарта 802.11 является невозможность роуминга. Стандарт IEEE 802.11 предусматривает работу компьютеров без точки доступа, в составе одной ячейки. В этом случае функции точки доступа выполняют сами рабочие станции.

Этот стандарт разработан и ориентирован на оборудование, функционирует в полосе частот 2400-2483,5 МГц. При этом радиус ячейки достигает 300 м, не ограничивая топологию сети.

1.2. Стандарт IEEE 802.11a

IEEE 802.11a - наиболее перспективный стандарт беспроводной сети, который рассчитан на работу в двух радиодиапазонах, - 2,4 и 5 ГГц. Используемый метод OFDM (англ. Orthogonal frequency-division multiplexing) позволяет достичь максимальной скорости передачи данных 54 Мбит/с. Кроме этой скорости, спецификациями предусмотрены и другие скорости:

• обязательные - 6, 12 и 24 Мбит / с;

• необязательные - 9, 18, 36, 48 и 54 Мбит / с.

Этот стандарт также имеет свои преимущества и недостатки. Из преимуществ можно отметить следующие:

• использование параллельной передачи данных

• высокая скорость передачи;

• возможность подключения большого количества компьютеров.

Недостатки стандарта IEEE 802.11a такие:

• Ограниченный радиус действия сети при использовании диапазона 5 ГГц (примерно 100 м);

• высокая стоимость оборудования по сравнению с оборудованием других стандартов;

• для использования диапазона 5 ГГц необходимо специальное разрешение от государственных организаций по надзору за радио эфиром.

Для достижения высоких скоростей передачи данных стандарта IEEE 802.11a использует в своей работе технологию квадратурной амплитудной модуляции QAM.

1.3. Стандарт IEEE 802.11b

Работа над стандартом IEEE 802.11b (другое название - IEEE 802.11 High rate, высокая пропускная способность) была закончена в 1999 году. Работа данного стандарта основана на методе прямого расширения спектра DSSS с использованием восьмиразрядных последовательностей Уолша. При этом каждый бит данных кодируется с помощью последовательности

дополнительных кодов ССК. Что позволяет достичь скорости передачи данных 11 Мбит / с.

Как и базовый стандарт, IEEE 802.11b работает с частотой 2,4 ГГц, используя не более трех каналов, которые не перекрываются. Радиус действия сети при этом составляет около 300м.

Особенностью этого стандарта является то, что при необходимости (например, при ухудшении качества сигнала, большом расстоянии от точки доступа и действия других помех) скорость передачи данных может уменьшаться до 1 Мбит/с. Если качество сигнала возросло, сетевое оборудование автоматически повышает скорость передачи до максимальной. Этот механизм называется динамическим регулированием скорости.

Кроме оборудования стандарта IEEE 802.11b, часто встречается оборудования IEEE 802.11b+. Отличие между этими стандартами заключается только в скорости передачи данных. В этом случае она доходит до 22 Мбит/с, благодаря использованию метода двоичного пакетного сверточного кодирования PBCC.

1.4. Стандарт IEEE 802.11d

Стандарт IEEE 802.11d определяет параметры физических каналов и сетевого оборудования. Он описывает правила, касающиеся разрешенной мощности излучения передатчиков в диапазонах разрешенных частот. Этот стандарт очень важен, поскольку для работы сетевого оборудования используются радиоволны. Если они не соответствуют указанным параметрам, то могут создавать помехи другим устройствам, работающим в этом или близком диапазоне частот.

1.5. Стандарт IEEE 802.11e

Поскольку в сетях могут передаваться данные различных форматов и важности, существует потребность в механизме, который бы определял их важность и присваивал необходим приоритет. За это отвечает стандарт IEEE 802.11, специально разработанный с целью передачи потоковых видео - или аудио данных с гарантированным качеством и доставкой.

1.6. Стандарт IEEE 802.11f

Стандарт IEEE 802.11f разработан с целью обеспечения аутентификации сетевого оборудования (рабочей станции) при перемещении компьютера пользователя от одной точки доступа к другой, то есть между сегментами сети. При этом вступает в действие протокол обмена служебной информацией IAPP (Inter-Access Point Protocol), который необходим для передачи данных между точками доступа. При этом достигается эффективная организация работы распределенных беспроводных сетей.

1.7. Стандарт IEEE 802.11g

Наиболее употребительным в настоящее время стандартом можно считать стандарт IEEE 802.11g, который унаследовал лучшие качества стандарта IEEE 802.11 и IEEE 802.11b и, кроме того, обладает многими собственными полезными качествами. Целью создания данного стандарта было достижение скорости передачи данных 54 Мбит/с.

Как и IEEE 802.11b, стандарт IEEE 802.11g разработан для работы в частотном диапазоне 2,4 ГГц. IEEE 802.11g регламентирует обязательные и возможные скорости передачи данных:

• обязательные - 1; 2; 5,5; 6; 11; 12 и 24 Мбит/с;

• возможные - 33, 36, 48 и 54 Мбит/с.

Для достижения таких показателей используется кодирование с помощью последовательности дополнительных кодов (ССК), метод ортогонального частотного мультиплексирования (OFDM), метод гибридного кодирования (CCK-OFDM) и метод двоичного пакетного сверточного кодирования (PBCC). Стоит отметить, что одну и ту же скорость можно достичь разными методами, однако обязательные скорости передачи данных достигаются только с помощью методов ССК и OFDM, а возможные скорости - с помощью методов CCK-OFDM и PBCC.

Преимуществом оборудования стандарта IEEE 802.11g является совместимость с оборудованием IEEE 802.11b. Что позволяет легко использовать любой компьютер с сетевой картой стандарта IEEE 802.11 для работы с точкой доступа стандарта IEEE 802.11g, и наоборот.

1.8. Стандарт IEEE 802.11h

Стандарт IEEE 802.11h разработан с целью эффективного управления мощностью излучения передатчика, выбором частоты передачи и генерации нужных отчетов. Он вносит некоторые новые алгоритмы в протокол доступа к среде MAC (Media Access Control, управление доступом к среде), а также в физический уровень стандарта IEEE 802.11a.

В первую очередь это связано с тем, что в некоторых странах диапазон 5 ГГц используется для трансляции спутникового телевидения, радарного слежения за объектами, что может вносить помехи в работу передатчиков беспроводной сети.

Смысл работы алгоритмов стандарта IEEE 802.11h заключается в том, что при обнаружении отраженных сигналов (интерференции) компьютеры беспроводной сети (или передатчики) могут динамически переходить в другой диапазон, а также снижать или повышать мощность передатчиков. Это позволяет более эффективно организовать работу уличных и офисных радиосетей.

1.9. Стандарт IEEE 802.11i

Стандарт IEEE 802.11i разработан специально для повышения безопасности работы беспроводной сети. С этой целью созданы различные алгоритмы шифрования и аутентификации, функции защиты при обмене информацией, возможность генерирования ключей:

• AES (Advanced Encryption Standard, передовой алгоритм шифрования данных) - алгоритм шифрования, который позволяет работать с ключами длиной 128, 192 и 256 бит;

• RADIUS (Remote Authentication Dial-In User Service, служба дистанционной аутентификации пользователя) - система аутентификации с возможностью генерирования ключей для каждой сессии и управления ими, включая алгоритмы проверки достоверности пакетов;

• TKIP (Temporal Key Integrity Protocol, протокол целостности временных ключей) - алгоритм шифрования данных;

• WRAP (Wireless Robust Authenticated Protocol, устойчивый беспроводной протокол аутентификации) - алгоритм шифрования данных;

• Ccmp (Counter with Cipher Block Chaining Message Authentication Code Protocol) - алгоритм шифрования данных.

1.10. Стандарт IEEE 802.11j

Стандарт IEEE 802.11j разработан специально для использования беспроводных сетей в Японии, а именно - для работы в дополнительном диапазоне радиочастот 4,9-5 ГГц. Спецификация предназначена для Японии и расширяет стандарт 802.11a добавочным каналом 4,9 ГГц.

На данный момент частота 4,9 ГГц рассматривается как дополнительный диапазон для использования в США. Из официальных источников известно, что этот диапазон планируется для использования органами общественной и национальной безопасности.

Данным стандартом расширяется диапазон работы устройств стандарта IEEE 802.11a.

1.11. Стандарт IEEE 802.11n

В настоящее время стандарт IEEE 802.11n - самый перспективный из всех стандартов, используемых в беспроводных сетях. Но он только разрабатывается, хотя возможности, которые он открывает, выглядят очень привлекательно. Данный стандарт должен обеспечить скорость передачи данных, минимальное значение которой будет 100 Мбит/с, что фактически равняется наиболее распространенной скорости в проводных сетях стандарта Ethernet 802.3. IEEE 802.11n использовать метод ортогонального частотного мультиплексирования (OFDM) и квадратурная модуляция (QAM). Это должно обеспечить не только высокую скорость передачи данных, но и полную совместимость со стандартами IEEE 802.11a, IEEE 802.11b и IEEE 802.11g. Для увеличения скорости передачи данных планируется использовать несколько новых технологий, одной из которых является

технология с множественным вводом /выводом (MIMO - Multiple Input Multiple Output). Ее смысл заключается в параллельной передачи данных по различным каналам с применением нескольких передающих антенн. Кроме того, используется расширение частотного канала до 40 МГц.

1.12. Стандарт IEEE 802.11r

Ни в одном беспроводном стандарте до конца не описаны правила роуминга, то есть перехода клиента от одной зоны к другой. Это должны сделать в стандарте IEEE 802.11r.

2. Физический уровень протокола 802.11

Физический уровень протокола 802.11, хотя и не встречается в чистом виде, в то же время является основой всех остальных протоколов. В стандарте 802.11, как и во всех остальных стандартов данного семейства, предусмотрено использование частотного диапазона от 2400 до 2483,5 МГц, то есть частотный диапазон шириной 83,5 МГц, который, как будет показано далее, разбитый на несколько частотных подканалов.

3. Архитектура беспроводных сетей

Развитие беспроводных сетей, как и многое другое, проходит под постоянным контролем соответствующих организаций. Главной среди них является IEEE (Institute of Electrical and Electronic Engineers, Международный институт инженеров электротехники и электроники). В частности, беспроводные стандарты, сетевое оборудование и все, что относится к беспроводным сетям контролирует рабочая группа WLAN (Wireless Local Area Network, беспроводная локальная сеть), в которую входят более 100 представителей различных университетов и компаний-разработчиков сетевого оборудования. Эта комиссия собирается несколько раз в год с целью совершенствования существующих стандартов и создание новых, основанных на последних исследованиях и компьютерных достижениях.

На сегодняшний день используется два варианта архитектуры или, проще говоря, варианты построения сети: независимая конфигурация (Ad-hoc) и инфраструктурная конфигурация. Различия между ними незначительны, однако они кардинально влияют на такие показатели, как количество пользователей, подключаемых радиус сети помехоустойчивость и так далее. Какая конфигурация сети не была бы выбрана, стандарты определяют один тип протокола доступа к носителю и различные спецификации для физических каналов.

3.1 Независимая конфигурация (Ad-hoc)

Режим независимой конфигурации (IBSS - Independent Basic Service Set, независимый базовый набор служб), который часто называют «точка-точка», - самый простой в применении. Соответственно, проще является построение и настройка сети с использованием независимой конфигурации (рис. 1).

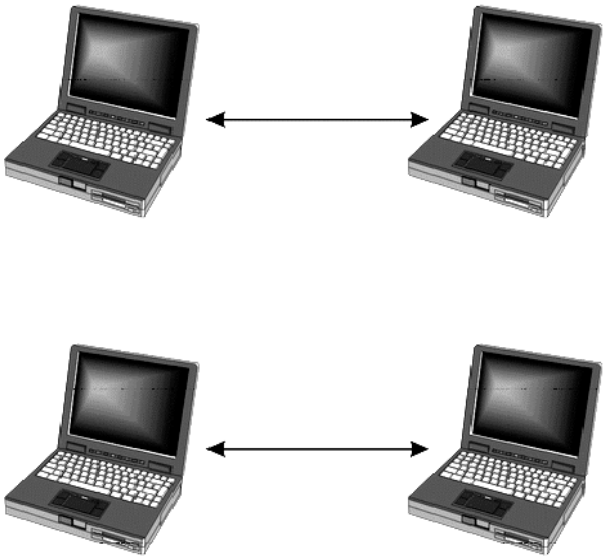


Рисунок 1 - Режим независимой конфигурации

Чтобы объединить компьютеры в беспроводную сеть, достаточно оборудовать каждый компьютер адаптером беспроводной связи. Как правило, такими адаптерами комплектуются переносные компьютеры. Построение такой сети водится к настройке соответствующих ресурсов и ограничений.

Обычно такой способ используется для организации хаотичной или временной сети, а также в том случае, если другой способ построения сети по каким-либо причинам не подходит.

Хотя режим независимой конфигурации прост в построении, существует несколько недостатков, главными из которых являются малый радиус действия сети и низкая устойчивость к помехам, что накладывает определенные ограничения на местоположение компьютеров сети. Кроме того, подключиться к внешней сети или к Интернету в таком случае очень непросто.

3.2. Инфраструктурная конфигурация

Инфраструктурная конфигурация, или, как ее еще часто называют, «режим клиент-сервер», - перспективный и широко используемый вариант беспроводной сети.

Инфраструктурная конфигурация имеет много преимуществ, среди которых возможность подключения достаточно большого количества пользователей, хорошая помехоустойчивость, высокий уровень контроля

подключений. Есть возможность использования комбинированной топологии и проводных сегментов сети.

Для организации беспроводной сети с использованием инфраструктурной конфигурации необходимо иметь как минимум одну точку доступа (Access Point) (рис. 2).

Рисунок 2 - Точка доступа

В этом случае конфигурация называется базовым набором служб (BSS - Basic Service Set). Точка доступа может работать автономно или в составе проводной сети и может выполнять функцию моста между проводными и беспроводными сегментами сети. При такой конфигурации сети компьютеры «общаются» только с точки доступа, которая управляет передачей данных между компьютерами.

Конечно, одной точкой доступа сеть может не ограничиваться. В этом случае базовые наборы служб образуют единую сеть, конфигурация которой носит название расширенного набора служб (ESS - Extended Service Set). При такой конфигурации сети точки доступа обмениваются между собой информацией, переданной с помощью проводного соединения или с помощью радиомостов. Это позволяет эффективно организовывать трафик между сегментами сети (фактически - точками доступа).

Дополнительную информацию можно получить из источников, представленных в литературе к методическим указаниям к лабораторным работам.

Задание к лабораторной работе

Изучить протоколы обмена информацией в беспроводных сетях. Вывести список Wi-Fi точек.

SSID (*Service Set Identifier*) - уникальное наименование беспроводной сети, которое отличает одну сеть Wi-Fi от другой.

Варианты заданий

1. Вывести SSID и тип сети.
2. Вывести SSID и проверку подлинности.
3. Вывести SSID и тип шифрования.
4. Вывести SSID и BSSID.
5. Вывести SSID и тип сети.
6. Вывести SSID и сигнал.
7. Вывести SSID и тип радио.

8. Вывести SSID и канал.
9. Вывести SSID и базовую скорость.
10. Вывести SSID сетей, у которых проверка подлинности WPA2.
11. Вывести SSID сетей, у которых сигнал больше 50%.
12. Вывести SSID и количество найденных сетей.
13. Вывести SSID сетей, у которых тип сети - Инфраструктура.
14. Вывести SSID сетей, у которых шифрование - CCMP.
15. Вывести SSID сетей, у которых нет шифрования.

Содержание отчета

1. Краткое описание теоретических сведений для работы с Wi-Fi – точками.
2. Код программы
3. Скриншоты работы программы.

1. Пример реализации на C# (необходим ManagedWifi

   http://managedwifi.codeplex.com/releases/view/7718)

```
WlanClient client = new WlanClient();
  foreach (WlanClient.WlanInterface wlanIface in client.Interfaces)
  {
     Wlan.WlanAvailableNetwork[] networks =
wlanIface.GetAvailableNetworkList(0);
     foreach (Wlan.WlanAvailableNetwork network in networks)
     {
        Console.WriteLine("Found network with SSID {0}.",
        GetStringForSSI(network.dot11Ssid));
     }
     Console.ReadKey();
  }
}

static string GetStringForSSI(Wlan.Dot11Ssid ssid)
{
   return Encoding.ASCII.GetString( ssid.SSID, 0, (int) ssid.SSIDLength );
}
```

2. Пример реализации на Java

```
ProcessBuilder builder = new ProcessBuilder(
        "cmd.exe", "/C","netsh", "wlan","show", "all");
```

```
        builder.redirectErrorStream(true);
        Process p = builder.start();
        BufferedReader r = new BufferedReader(new
    InputStreamReader(p.getInputStream(),"cp866"));
        String line;
        while (true) {
            line = r.readLine();
      if (line == null) { break; }
      System.out.println(line);
}
```

Литература

1. Йон Снейдер. – Эффективное программирование TCP/IP. – Питер, 2002 г., 320 стр.
2. Э. Джонс, Д.Оланд, - Программирование в сетях Windows., «Русская редакция», 2002 г., 608 стр.
3. Д. Камер, Д.Стивенс, - Сети TCP/IP, «Вильямс», 2002, 592 стр.
4. У. Стивенс, - «Unix. Разработка сетевых приложений», «Питер», 2003 г., 1088 стр.
5. Ш.Уолтон, - «Создание сетевых приложений в среде Linux», «Вильямс», 2001 г., 464 стр.
6. RFC по протоколам: ftp, tftp, pop, smtp, imap, udp, tcp, arp, rarp, ip, rip, ospf.
7. Энциклопедия. Технология корпоративних сетей.
8. Олифер В.Г., Н.А.Олифер «Компьютерые сети», учебник, Сн-Петербург, 2001 г.
9. Ресурсы Internet
10. В.Г. Олифер , Н.А. Олифер «Компьютерные сети, принципы, технологии, протоколы» - ИД «Питер» 1999 г.
11. М.Гук “Аппаратные средства локальных сетей - энциклопедия”, ИД “Питер”, 2001 г.
12. http://webi.ru/webi_files/xmpp_protocol.html
13. P. Roshan, J. Leary, “802.11 Wireless LAN fundamentals”. Cisco, Indianapolis, Ind., 2004.
14. J. Geier. “Wireless networks first-step”. Cisco, Indianapolis, Ind., 2005.
15. R. Flickenger. “Building Wireless Community Networks, Second Edition”. O'Reilly, 2003, 182p.
16. J. Khan, A. Khwaja. “Building secure wireless networks with 802.11”. Wiley, Indianapolis, Ind., 2003.
17. Managed Wifi API — Managed Wifi API [Электронный ресурс]. - https://managedwifi.codeplex.com/
18. How to find a list of wireless networks (SSID’s) in Java, C# and/or C? – Stack Overflow [Электронный ресурс]. - http://stackoverflow.com/questions/917910/how-to-find-a-list-of-wireless-networks-ssids-in-java-c-and-or-c/979806#979806

СОДЕРЖАНИЕ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ И ЗАДАНИЯ

к лабораторным работам по дисциплине

«Протоколы компьютерных сетей»

Для студентов направления подготовки 09.03.04

"ПРОГРАММНАЯ ИНЖЕНЕРИЯ"

Составители: Алла Викторовна Чернышова