1. **Протокол IP. Особенности его программирования в защищенном режиме работы процессора.**

**Internet Protocol** или **IP** (англ. internet protocol – межсетевой протокол) - маршрутизируемый сетевой протокол сетевого уровня семейства TCP/IP.

Протокол IP используется для негарантированной доставки данных, разделяемых на так называемые пакеты от одного узла сети к другому. Это означает, что на уровне этого протокола (третий уровень сетевой модели OSI) не даётся гарантий надёжной доставки пакета до адресата. В частности, пакеты могут прийти не в том порядке, в котором были отправлены, продублироваться (когда приходят две копии одного пакета - в реальности это бывает крайне редко), оказаться повреждёнными (обычно повреждённые пакеты уничтожаются) или не прибыть вовсе. Гарантии безошибочной доставки пакетов дают протоколы более высокого (транспортного) уровня сетевой модели OSI - например, TCP - который использует IP в качестве транспорта.

В протоколе IP 4 версии каждому узлу сети ставится в соответствие IP-адрес длиной 4 октета (1 октет состоит из 8 бит). При этом компьютеры в подсетях объединяются общими

начальными битами адреса. Количество этих бит, общее для данной подсети, называется маской подсети.

IP-пакет представляет собой форматированный блок информации, передаваемый по вычислительной сети. Соединения вычислительных сетей, которые не поддерживают пакеты, такие как традиционные соединения типа «точка-точка» в телекоммуникациях, просто передают данные в виде последовательности байтов, символов или битов. При использовании пакетного форматирования сеть может передавать длинные сообщения более надежно и эффективно.

IP-адреса разделяются на 5 классов: A, B, C, D, E. Адреса классов A, B и C делятся на две логические части: номер сети и номер узла.



Идентификатор сети, также называемый адресом сети, обозначает один сетевой сегмент в более крупной объединенной сети, использующей протокол TCP/IP. IP-адреса всех систем, подключенных к одной сети, имеют один и тот же идентификатор сети. Этот

идентификатор также используется для уникального обозначения каждой сети в более крупной объединенной сети.

Идентификатор узла, также называемый адресом узла, определяет узел TCP/IP (рабочую станцию, сервер, маршрутизатор или другое устройство) в пределах каждой сети. Идентификатор узла уникальным образом обозначает систему в том сегменте сети, к которой она подключена.

**Маска подсети** – это 32-разрядное двоичное число, в разрядах расширенного префикса содержащая единицу; в остальных разрядах находится ноль. Расширенный сетевой префикс получается побитным сложением по модулю два (операция XOR) IP-адреса и маски подсети.

**Особенности программирования протокола:**

Основные функции из библиотеки Winsock:

Функция **socket** (int af, int type, int protocol) нужна для создания и инициализации сокета. Здесь af – сведения о семействе адресов. Для интернет-протоколов указывается константа AF\_INET; type – тип передаваемых данных (поток или дейтаграммы). В данной лабораторной работе используется константа SOCK\_DGRAM; protocol – протокол передачи данных. Для протокола IP используется константа IPPROTO\_IP. Функция возвращает дескриптор созданного сокета.

Функция **bind** (IPEndPoint ep) привязывает адрес и порт к ранее созданному сокету Здесь ep – локальная конечная точка.

Для определения IP адреса по имени используется функция **struct hostent FAR \* gethostbyname (const char FAR \* name)**. В качестверезультата, функция возвращает структуру hostent.В случае автоматического распределения адресов и портов узнать

какой адрес и порт присвоен сокету можно при помощи функции **getsockname (SOCKET s, struct sockaddr FAR\* name, int FAR\*** **namelen).** Если операция выполнена успешно, возвращает 0, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при

помощи функции WSAGetLastError.

Функция **sendto (const char FAR \* buf, IPEndPoint ep)** служит для передачи

данных. Здесь buf - указатель на буфер с данными, которые необходимо переслать; ep – конечная локальная точка. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **recvfrom (char FAR\* buf, IPEndPoint ep)** служит для приема

данных. Если операция выполнена успешно, возвращает количество

полученных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **close ()** нужна для закрытия сокета. Она возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

1. **Протоколы POP3 и SMTP. Особенности их программирования в защищенном режиме работы процессора.**

**POP3 (англ. Post Office Protocol Version 3)** – стандартный Интернет-протокол прикладного уровня, используемый клиентами электронной почты для извлечения электронного сообщения с

удаленного сервера по TCP/IP-соединению. Этот протокол обеспечивает доступ узла к базовому

почтовому серверу.

POP3 поддерживает простые требования «загрузи-и-удали» для доступа к удаленным почтовым ящикам. Хотя большая часть POP-клиентов предоставляют возможность оставить почту на сервере после загрузки, использующие POP клиенты обычно соединяются, извлекают все письма, сохраняют их на пользовательском компьютере как новые сообщения, удаляют их с сервера, после чего разъединяются.

POP3-сервер прослушивает порт 110. Шифрование связи для POP3 запрашивается после запуска протокола, с помощью либо команды STLS (если она поддерживается), либо POP3S, которая соединяется с сервером используя TLS или SSL по TCP-порту 995.

Доступные сообщения клиента фиксируются при открытии почтового ящика POP-сессией и определяются количеством сообщений для сессии, или, по желанию, с помощью уникального

идентификатора, присваиваемого сообщению POP-сервером. Этот уникальный идентификатор является постоянным и уникальным для почтового ящика и позволяет клиенту получить доступ к одному и тому же сообщению в разных POP-сессиях. Почта извлекается и помечается для удаления с помощью номера сообщения. При выходе клиента из сессии помеченные сообщения удаляются из почтового ящика.

Команды РОРЗ состоят из ключевых слов, состоящих из ASCII-символов, и одним или несколькими параметрами, отделяемыми друг от друга символом "пробела" - <SP>. Все команды заканчиваются символами "возврата каретки" и "перевода строки" - <CRLF>. Длина ключевых слов не превышает четырех символов, а каждого из аргументов может быть до 40 символов.

Ответы РОРЗ-сервера на команды состоят из строки статус-индикатора, ключевого слова, строки дополнительной информации и символов завершения строки. Длина строки ответа может достигать 512 символов. Строка статус-индикатора принимает два значения: положительное ("+ОК") и отрицательное ("-ERR"). Любой сервер РОРЗ обязан отправлять строки статус-индикатора в верхнем регистре, тогда как другие команды и данные могут приниматься или отправляться как в нижнем, так и в верхнем регистрах.

Ответы РОРЗ-сервера на отдельные команды могут составлять несколько строк. В этом случае строки разделены символами <CRLF>. Последнюю строку информационной группы завершает

строка, состоящая из символа "." (код — 046) и <CRLF>, т. е. последовательность "CRLF.CRLF"

Команды протокола POP3:

· USER - идентифицирует пользователя с указанным именем;

· PASS - указывает пароль для пары клиент-сервер;

· QUIT - закрывает TCP-соединение;

· STAT - сервер возвращает количество сообщений в почтовом ящике плюс размер почтового ящика;

· LIST - сервер возвращает идентификаторы сообщений вместе с размерами сообщений;

· RETR - извлекает сообщение из почтового ящика;

· DELE - отмечает сообщение для удаления;

· NOOP - Сервер возвращает положительный ответ, но не совершает никаких действий;

· LAST - Сервер возвращает наибольший номер сообщения из тех, к которым ранее уже обращались;

· RSET - Отменяет удаление сообщения, отмеченного ранее командой DELE.

**SMTP (Simple Mail Transfer Protocol)** – широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP.

Упрощенно схема взаимодействия представлена на рисунке ниже (объемными стрелками показано направление движения почтовых сообщений).



Со стороны пользователя обычно одна и та же программа выступает в роли и POP3 клиента, и SMTP клиента отправителя.

На схеме у пользователя есть клиентское ПО, а у провайдера – серверная часть приложения. На самом деле это немного не так. Протокол SMTP делает возможным смену сторон даже в ходе одного сеанса. Условно принято считать клиентом ту сторону, которая начинает взаимодействие и хочет отослать почту, а сервером ту, что принимает запросы. После того, как клиент посылает серверу несколько служебных команд и получает положительные ответы на них, он отправляет SMTP серверу собственно тело сообщения. SMTP сервер получает сообщение, вносит в него дополнительные заголовки, указывающие на то, что он обработал данное послание, устанавливает связь со следующим SMTP сервером по пути следования письма.

Общение между любыми SMTP серверами происходит по той же схеме. Инициирует переговоры клиент, сервер на них отвечает, а затем получает корреспонденцию и "ставит штампик" в теле письма (в его заголовочной части). Все это очень напоминает обычную бумажную почту, где работу по сортировке и отправке почты выполняют люди.

Как только почта достигнет конечного пункта (SMTP сервера адресата сообщения), она будет сложена в почтовый ящик абонента, который всегда сможет в удобное для него время изъять ее по протоколам POP3 или IMAP, в зависимости от того, какой из них поддерживается провайдером. В арсенал SMTP клиента, равно как и сервера, входит около 10 команд, но, воспользовавшись только пятью из них, уже можно легально послать почтовое сообщение. Это HELO, MAIL, RCPT, DATA, QUIT. Их использование подразумевается именно в такой последовательности. HELO предназначена для идентификации отправителя, MAIL указывает адрес отправителя, RCPT – адрес назначения. После команды DATA и ответа на нее, клиент посылает серверу тело сообщения, которое должно заканчиваться строкой, содержащей лишь одну точку.

**Особенности программирования протокола:**

Функция message.Attachments.Add(attachment) для добавления вложений в сообщение. Cient.Send(message) для отправки сообщения указанному в свойствах клиента адресату, эта функция осуществляет отправку специальных команд SMTP-серверу.

client.Dispose() отправка серверу сообщения о завершении сеанса.

client.Connect(mailServ, port, enableSSL) –для подключения к серверу почты mailServ (pop3-сервер), с использование порта port и с включением/отключением SSL – enableSSL.

client.Authenticate(login, pass) – аутентификация с логином login и паролем pass на pop3-сервере.

client.GetMessageCount() – получение количества сообщений в почтовом ящике. client.GetMessageHeaders(i) – получение заголовка i-го письма. client.GetMessageSize(i) – получение размера i-го письма в КБ. client.GetMessage(i) – получение полного i-го письма.

mess. GetBodyAsText() – получение текста из письма. mess.Save(f) – сохранение письма в файл ( для сохранения вложений). client.DeleteMessage(i) – удаление i-го сообщения. client.Disconnect() – отсоединение от pop3-сервера.

1. **Протокол HTTP. Особенности его программирования в защищенном режиме работы процессора.**

**HTTP (Hyper Text Transfer Protocol** – протокол передачи гипертекста) – протокол прикладного уровня стека протоколов TCP/IP, предназначенный для передачи данных по сети с использованием транспортного протокола TCP.

Основой HTTP является технология «клиент-сервер». HTTP- клиенты отсылают HTTP-запросы, которые содержат метод, обозначающий потребность клиента. Также такие запросы содержат

универсальный идентификатор ресурса, указывающий на желаемый ресурс. Обычно такими ресурсами являются хранящиеся на сервере файлы. По умолчанию HTTP-запросы передаются на порт 80. HTTP-сервер отсылает коды состояния, сообщая, успешно ли выполнен HTTP-запрос или же нет.

Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы, но ими могут быть логические объекты или что-то абстрактное.

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

Метод протокола HTTP – это команда, передаваемая HTTP-клиентом HTTP-серверу.

Основные методы HTTP

-GET – Используется для запроса содержимого ресурса, на который указывает URI, содержащийся в запросе.

-HEAD – Используется для извлечения метаданных или проверки наличия ресурса, на который указывает URI, содержащийся в запросе.

-POST – Применяется для передачи данных заданному ресурсу. Данный метод предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого.

-PUT – Применяется для передачи данных заданному ресурсу. Данный метод предполагает, что передаваемое клиентом содержимое соответствует находящемуся по данному URI ресурсу.

-OPTIONS – Используется для определения возможностей HTTP-сервера или параметров соединения для конкретного ресурса.

-DELETE – Применяется для удаления ресурса, на который указывает URI.

Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Название метода чувствительно к регистру. Каждый сервер обязан поддерживать как минимум

методы GET и HEAD.

Каждое HTTP-сообщение состоит из трех частей, которые передаются в следующем порядке:

1. Стартовая строка – определяет тип сообщения;

2. Заголовки – характеризуют тело сообщения, параметры

передачи и прочие сведения;

3. Тело сообщения – непосредственно данные сообщения.

Протокол HTML позволяет достаточно легко создавать клиентские приложения. Возможности протокола можно расширить благодаря внедрению своих собственных заголовков, с помощью которых можно получить необходимую функциональность при решении нетривиальной задачи. При этом сохраняется совместимость с другими клиентами и серверами: они будут просто игнорировать неизвестные им заголовки.

**Особенности программирования протокола:**

Start() – установка TcpListener в состояние прослушивания. AcceptTcpClient() – принятие подключения от нового клиента. GetStream().Read(Buffer, 0, Buffer.Length) – чтение данных из потока клиента в Buffer. GetStream().Write(HeadersBuffer, 0, HeadersBuffer.Length) – запись в поток клиента. Close() – отключение клиента.

1. **Протокол DNS. Особенности программирования протокола DNS в защищенном режиме работы процессора.**

Доменная система имен реализована в Интернете, но она может работать и как автономная система имен в любой крупной корпоративной сети, которая также использует стек TCP/IP, но не связана с Интернетом.

Соответствие между доменными именами и IP-адресами может устанавливаться как средствами локального хоста, так и средствами централизованной службы. На раннем этапе развития Интернета на каждом хосте вручную создавался текстовый файл с известным именем hosts.txt. Этот файл состоял из некоторого количества строк, каждая из которых содержала одну пару «IP-адрес — доменное имя».

В настоящее время используется масштабируемая служба для разрешения имен — система доменных имен (Domain Name System, DNS).

Процедура разрешения DNS-имени во многом аналогична процедуре поиска файловой системой адреса файла по его символьному имени. Для определения IP-адреса по доменному имени необходимо просмотреть все DNS-серверы, обслуживающие цепочку поддоменов, входящих в имя хоста, начиная с корневого домена. Существенным отличием является то, что файловая система расположена на одном компьютере, а служба DNS по своей природе является распределенной.

Для ускорения поиска IP-адресов DNS-серверы широко применяют процедуру кэширования проходящих через них ответов. Чтобы служба DNS могла оперативно отрабатывать изменения, происходящие в сети, ответы кэшируются на определенное время — обычно от нескольких часов до нескольких дней.

**Особенности программирования протокола:**

Библиотека ip Helper:

GetInterfaceInfo — получает список интерфейсов и информацию о них.

1. **Протокол DHCP. Особенности программирования протокола DHCP в защищенном режиме работы процессора.**

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

количество ошибок. Протокол DHCP используется в большинстве сетей TCP/IP.

Протокол DHCP предоставляет три способа распределения IP-адресов:

1. *Ручное распределение*

2. *Автоматическое распределение.*

3. *Динамическое распределение.*

Некоторые реализации службы DHCP способны автоматически обновлять записи DNS, соответствующие клиентским компьютерам, при выделении им новых адресов.

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

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

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

Некоторыми из наиболее часто используемых опций являются:

- IP-адрес маршрутизатора по умолчанию;

- маска подсети;

- адреса серверов DNS;

- имя домена DNS.

**Особенности программирования протокола:**

* IpReleaseAddress — освобождает адрес переданного адаптера;
* IpRenewAddress — получает адрес для переданного адаптера;
* GetInterfaceInfo — получает список интерфейсов и информацию о них.

1. **Протокол TCP. Особенности программирования протокола TCP в защищенном режиме работы процессора.**

**Transmission Control Protocol (TCP)** (протокол управления передачей) - один из основных сетевых протоколов Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Выполняет функции протокола транспортного уровня модели OSI.

TCP - это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. В отличие от UDP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи.

TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере.

TCP протокол базируется на IP для доставки пакетов, но добавляются две важные вещи: во-первых, устанавливается соединение – это позволяет ему, в отличие от IP, гарантировать доставку пакетов, во-вторых – используются порты для обмена пакетами между приложениями, а не просто узлами.

Протокол TCP предназначен для обмена данными — это «надежный» протокол, потому что он, во-первых, обеспечивает надежную доставку данных, так как предусматривает установления логического соединения; во-вторых, нумерует пакеты и подтверждает

их прием квитанцией, а в случае потери организует повторную передачу; в-третьих, делит передаваемый поток байтов на части — сегменты - и передает их нижнему уровню, на приемной стороне снова собирает их в непрерывный поток байтов.

**Особенности программирования протокола:**

Перевод сокета в состояние “прослушивания” (для TCP) осуществляется функцией **Start**(int backlog), где backlog – максимальный размер очереди входящих сообщений на соединение. Используется сервером, чтобы информировать ОС, что он ожидает (“слушает”) запросы связи на данном сокете. Без этой функции всякое требование связи с сокетом будет отвергнуто.

Функция **Connect** (IPAddress ipAddr, int socket) нужна для соединения с сокетом, находящимся в состоянии “прослушивания” (для TCP). Она ипользуется процессом-клиентом для установления связи с сервером. В случае успешного установления соединения Connect возвращает 0, иначе SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **AcceptTcpClient**() служит для подтверждения запроса на соединение (для TCP). Функция используется для принятия связи на сокет. Сокет должен быть уже слушающим в момент вызова функции. Если сервер устанавливает связь с клиентом, то данная функция возвращает новый сокет-дескриптор, через который и производит общение клиента с сервером. Пока устанавливается связь клиента с сервером, функция блокирует другие запросы связи с данным сервером, а после установления связи “прослушивание” запросов возобновляется.

Функция **GetStream**() использовалась для получения потока который используется для приема и передачи данных.

Для передачи использовались функции Read и Write, имеющий одинаковую сигнатуру (byte [] mess, int offset, int BLOCK\_SIZE), где mess – принимаемое(передаваемое сообщение), offset начальная позиция сообщения в массиве mess, BLOCK\_SIZE – конечная позиция сообщения в массиве mess.

1. **Протокол UDP. Особенности программирования протокола UDP в защищенном режиме работы процессора.**

**Протокол UDP (User Datagram Protocol)** является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP. Протокол UDP обеспечивает доставку дейтограмм, но не требует подтверждения их получения.

Протокол UDP не требует соединения с удаленным модулем UDP ("бессвязный" протокол). К заголовку IP-пакета UDP добавляет поля: порт отправителя и порт получателя, которые обеспечивают мультиплексирование информации между различными прикладными процессами, а также, поля: длина UDP-дейтограммы и контрольная сумма, позволяющие поддерживать целостность данных. Таким

образом, если на уровне IP для определения места доставки пакета

используется адрес, на уровне UDP - номер порта.

Протокол UDP ориентирован на транзакции, получение датаграмм и защита от дублирования не гарантированы. Приложения, требующие гарантированного получения потоков данных, должны использовать протокол управления пересылкой (Transmission Control Protocol - TCP).

UDP – минимальный ориентированный на обработку сообщений протокол транспортного уровня, задокументированный в RFC 768.

UDP не предоставляет никаких гарантий доставки сообщения для протокола верхнего уровня и не сохраняет состояния отправленных сообщений.

UDP обеспечивает многоканальную передачу (с помощью номеров портов) и проверку целостности (с помощью контрольных сумм) заголовка и существенных данных. Надежная передача в случае необходимости должна реализовываться пользовательским

приложением. Из-за недостатка надежности, приложения UDP должны быть

готовы к некоторым потерям, ошибкам и дублированиям.

**Особенности программирования протокола:**

Создавался сокет для обмена данными.

Для передачи данных по протоколу UDP используется функция **Send** (const char FAR \* buf, int len, IPEndPoint IPendPoint), buf - указатель на буфер с данными, которые

необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; IPendPoint – конечная точка отправки сообщения. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Для приема данных по протоколу UDP используется **Receive**(IPEndPoint IPendPoint). Если операция выполнена успешно, возвращает принятое сообщение.

1. **Протокол ARP/RARP. Особенности программирования протоколов ARP/RARP в защищенном режиме работы процессора.**

**ARP** (**Address Resolution Protocol** - протокол определения адреса) – протокол канального уровня, предназначенный для определения MAC-адреса (адреса канального уровня) по известному IP-адресу (адресу сетевого уровня). Наибольшее распространение этот протокол получил благодаря распространению сетей IP, построенных поверх Ethernet, поскольку практически в 100% случаев при таком сочетании используется протокол ARP.

Функциональность протокола ARP сводится к решению двух задач. Одна часть протокола определяет физические адреса при посылке дейтаграммы, другая отвечает на запросы устройств в сети. Протокол ARP предполагает, что каждое устройство «знает» как свой IP -адрес, так и свой физический адрес.

Преобразование адресов выполняется путем поиска в таблице. Эта таблица, называемая ARP-таблицей, хранится в памяти и содержит строки для каждого узла сети. В двух столбцах содержатся IP- и Ethernet-адреса. Если требуется преобразовать IP-адрес в Ethernet-адрес, то ищется запись с соответствующим IP-адресом.

В ARP-таблице могут содержаться как статические, так и динамические записи. Динамические записи добавляются и удаляются автоматически, статические заносятся вручную. Так как большинство устройств в сети поддерживает динамическое разрешение адресов, то

администратору, как правило, нет необходимости собственноручно указывать записи протокола ARP в таблице адресов. Кроме того, ARP-таблица всегда содержит запись с физическим

широковещательным адресом (OxFFFFFFFFFFFF) для локальной сети. Эта запись позволяет устройству принимать широковещательные ARP-запросы.

Каждая запись в ARP-таблице имеет свое время жизни, например для операционной системы Microsoft Windows 2000 оно составляет 10 минут. При добавлении записи для нее активируется

таймер. Если запись не востребована в первые две минуты, она удаляется. Если используется — будет существовать на протяжении 10 минут. В некоторых реализациях протокола ARP новый таймер устанавливается после каждого обращения к записи в ARP -таблице.

ARP-таблица необходима потому, что IP-адреса и Ethernet-адреса выбираются независимо, и нет какого-либо алгоритма для преобразования одного в другой. IP-адрес выбирает менеджер сети с

учетом положения машины в сети internet. Если машину перемещают в другую часть сети internet, то ее IP-адрес должен быть изменен.

Ethernet-адрес выбирает производитель сетевого интерфейсного оборудования из выделенного для него по лицензии адресного пространства. Когда у машины заменяется плата сетевого адаптера, то меняется и ее Ethernet-адрес.

ARP-таблица заполняется автоматически модулем ARP, по мере необходимости. Когда с помощью существующей ARP-таблицы не удается преобразовать IP-адрес, то происходит следующее:

1. По сети передается широковещательный ARP-запрос.

2. Исходящий IP-пакет ставится в очередь.

Каждый сетевой адаптер принимает широковещательные передачи. Все драйверы Ethernet проверяют поле типа в принятом Ethernet-кадре и передают ARP-пакеты модулю ARP.

Каждый модуль ARP проверяет поле искомого IP-адреса в полученном ARP-пакете и, если адрес совпадает с его собственным IP-адресом, то посылает ответ прямо по Ethernet-адресу отправителя

запроса. Этот ответ получает машина, сделавшая ARP-запрос. Драйвер этой машины проверяет поле типа в Ethernet-кадре и передает ARP-пакет модулю ARP. Модуль ARP анализирует ARP-пакет и добавляет запись в свою ARP-таблицу. Если в сети нет машины с искомым IP-адресом, то ARP-ответа не будет и не будет записи в ARP-таблице.

**Протокол RARP** – это протокол, решающий обратную задачу – нахождение IP-адреса по известному локальному адресу. Он называется реверсивный ARP - RARP (Reverse Address Resolution Protocol) и используется при старте бездисковых станций, не знающих в начальный момент своего IP-адреса, но знающих адрес своего сетевого адаптера. Этот протокол требует наличия в сети сервера RARP , подключенного к тому же сегменту сети, что и интерфейс маршрутизатора.

**Особенности программирования протокола:**

Для поиска в таблице ARP mac-адреса по известному IP- адресу использовалась функция SendARP(int dstIp, int srcIp, byte[] mac, ref int macLen), где dstIp – искомый ip-адрес, srcIp - используется для выбора интерфейса для отправки запросана добавление в ARP, mac – выходной параметр, в нем будет занесен мак-адрес , macLen – длина мак-адреса.

Функция возвращает нулевое значение, если адрес найден.

Функция Process.Start(startInfo) использовалась для запуска таблицы ARP. startInfo задает параметры запуска приложения.

Функция CreateIpNetEntry( PMIB\_IPNETROW pArpEntry) для создания новой записи в таблице ARP. В этой функции параметр pArpEntry – указатель на структуру в которой занесен IP-адрес, MAC- адрес и их длины.

1. **Протокол SPX. Формат пакета SPX.**

**SPX** – протокол последовательного обмена пакетами (Sequenced Packet Exchange Protocol), разработанный Novell. Система адресов протокола SPX аналогична системе адресов протокола IPX и также состоит из 3 частей: номера сети, адреса станции и сокета.

Протокол SPX использует такой же блок ECB для передачи и приёма пакетов, что и протокол IPX. Однако, пакет, передаваемый при помощи протокола SPX, имеет более длинный заголовок. Дополнительно к 30 байтам стандартного заголовка пакета IPX добавляется еще 12 байт.



Поле **ConnControl** представляет собой как набор битовых флагов,

управляющих передачей данных по каналу SPX.

Поле **DataStreamType** состоит из однобитовых флагов, которые используются для классификации данных, передаваемых или принимаемых при помощи протокола SPX.

Поле **SourceConnID** содержит номер канала связи передающей программы, присвоенный драйвером SPX при создании канала связи. Этот номер должен указываться функции передачи пакета средствами SPX.

Поле **DestConnID** содержит номер канала связи принимающей стороны. Так как все пакеты приходят на один номер сокета и могут принадлежать разным каналам связи (на одном сокете можно открыть несколько каналов связи), необходимо классифицировать приходящие пакеты по номеру канала связи.

Поле **SeqNumber** содержит счетчик пакетов, переданных по каналу в одном направлении. На каждой стороне канала используется свой счетчик. После достижения значения FFFFh счетчик сбрасывается в нуль, после чего процесс счета продолжается. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

Поле **AckNumber** содержит номер следующего пакета, который должен быть принят драйвером SPX. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

Поле **AllocNumber** содержит количество буферов, распределенных программой для приема пакетов. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

1. **Протокол SPX. Особенности программирования протокола SPX в защищенном режиме работы процессора.**

**SPX** – протокол последовательного обмена пакетами (Sequenced Packet Exchange Protocol), разработанный Novell. Система адресов протокола SPX аналогична системе адресов протокола IPX и также состоит из 3 частей: номера сети, адреса станции и сокета.

Протокол SPX использует такой же блок ECB для передачи и приёма пакетов, что и протокол IPX. Однако, пакет, передаваемый при помощи протокола SPX, имеет более длинный заголовок. Дополнительно к 30 байтам стандартного заголовка пакета IPX добавляется еще 12 байт.

**Основные функции:**

Функция **socket**(int af, int type, int protocol) возвращает либо дескриптор созданного сокета, либо ошибку INVALID\_SOCKET. Расширенный код ошибки можно получить при помощи функции GetLastError. Параметр af содержит сведения о семействе протоколов

(AF\_INET, AF\_IPX). В данной лабораторной работе необходимо использовать константу AF\_IPX. Параметр type – тип передаваемых данных (поток или дейтаграммы). В данной лабораторной работе для IPX необходимо использовать константу DGRAM, а для SPX – константу SEQPACKET, которая означает, что пакеты будут отсылаться последовательно и в порядке очереди. Параметр protocol – протокол передачи данных. Для протокола IPX используется константа IPX, для SPX – SPX.

Чтобы работать дальше с созданным сокетом его нужно привязать к какой-либо локальной конечной точке. Этим занимается функция **bind** (EndPoint ep). ep – объект типа EndPoint который необходимо создать, он определяет сетевой адрес.

В конце работы сокет необходимо закрыть функцией **close**.

**EndPoint**(int networkNumber, long nodeNumber, short socketNumber) –создание объекта EndPoint, который определяет сетевой адрес. networkNumber – номер сети, nodeNumbe – адрес в сети, socketNumber – сокет.

**SendTo**(byte [] Message, EndPoint endPoint) – осуществляет отправку сообщения Message по адресу endPoint. В данном случае, отправка осуществлялась широковещательно.

**ReceiveFrom**(byte [] message, EndPoint endPoint) – осуществляет прием сообщения с адреса endPoint и записывает это сообщение в message.

**Особенности программирования протокола:**

**Listen**(int length) – устанавливает сокет в состояние прослушивания, length – максимальная долина очереди ожидающих подключения.

**Accept**() – создает новый сокет для созданного подключения.

**Connect**(EndPoint endPoint) – осуществляет подключение к endPoint.

1. **Протокол сетевого уровня IPX. Особенности программирования протокола IPX в защищенном режиме работы процессора.**

**Протокол IPX** – это протокол сетевого уровня модели взаимодействия открытых систем реализующий передачу пакетов (сообщений) между станциями сети на уровне датаграмм. Датаграмма – это сообщение, доставка которого получателю не гарантируется.

В процессе обмена сообщениями на уровне сеанса связи участвуют только две станции сети. На уровне датаграмм есть возможность посылать сообщение одновременно всем станциям сети.

Система адресов, используемая в протоколе IPX, представлена несколькими компонентами: это номер сети, адрес станции в сети и идентификатор программы на рабочей станции.

**Номер сети** - это номер сегмента сети, определяемого системным

администратором. Если в общей сети есть мосты, каждая отдельная

сеть, подключенная через мост, должна иметь свой, уникальный номер

сети.

**Адрес станции** - это число, которое является уникальным для

каждой рабочей станции. При использовании адаптеров Ethernet

уникальность обеспечивается изготовителем сетевого адаптера.

Специальный адрес FFFFFFFFFFFFh используется для рассылки

данных всем станциям данной сети одновременно.

**Идентификатор программы на рабочей станции (сокет)** -

число, которое используется для адресации определенной программы, работающей на компьютере.

Формат передаваемых с использованием протокола IPX по сети пакетов представлен ниже.



Пакет можно разделить на две части, поле Data представляет собой передаваемые данные, все остальные же поля представляют собой заголовок пакета.

Поле **Checksum** предназначено для хранения контрольной суммы передаваемых пакетов. При формировании пакетов не нужно заботиться о содержимом этого поля, так как проверка данных по контрольной сумме выполняется драйвером сетевого адаптера.

Поле **Length** определяет общий размер пакета. Длина заголовка фиксирована и равна 30 байт. Размер передаваемых в поле **Data** данных может составлять от 0 до 546 байт. При формировании пакетов не нужно проставлять длину пакета в поле **Length**, протокол IPX

делает это сам.

Поле **TransportControl** является счетчиком мостов, которые проходит пакет на своем пути от передающей станции к принимающей. Каждый раз, когда пакет проходит через мост,

значение этого счетчика увеличивается на единицу. IPX перед передачей пакета сбрасывает содержимое этого поля в нуль. Так как IPX сам следит за содержимым этого поля, при формировании пакетов не нужно изменять или устанавливать его в какое-либо состояние.

Поле **PacketType** определяет тип передаваемого пакета. Для IPX следует установить значение равное 4.

Поле **DestNetwork** определяет номер сети, в которую передается пакет.

Поле **DestNode** определяет адрес рабочей станции, которой предназначен пакет.

Поле **DestSocket** предназначено для адресации программы, запущенной на рабочей станции, которая должна принять пакет.

Поля **SourceNetwork, SourceNode** и **SourceSocket** содержат соответственно номер сети, из которой посылается пакет, адрес передающей станции и сокет программы, передающей пакет.

Поле **Data** содержит передаваемые данные.

Прикладные программы все свои запросы на прием и передачу пакетов направляют драйверу IPX, который, в свою очередь, обращается к драйверу сетевого адаптера. Для приема или передачи пакета прикладная программа должна подготовить пакет данных,

сформировав его заголовок, и построить так называемый блок управления событием ECB (Event Control Block).

В блоке ECB задается адресная информация для передачи пакета, адрес передаваемого пакета в оперативной памяти и некоторая другая информация.

Подготовив блок ECB, прикладная программа передает его адрес соответствующей функции IPX для выполнения операции приема или передачи пакета. Функции IPX, выполняющие прием или передачу пакетов выполняются асинхронно, т.е. не дожидаются завершения передачи или приема данных, а завершают их в фоновом режиме. Для определения завершения передачи или приема данных в блоке ECB есть специальное поле, которое показывает завершена или нет операция. Программа может периодически опрашивать это поле.

Существует и другой способ: В блоке ECB можно указать адрес процедуры, которая будет

вызвана при завершении выполнения операции передачи данных. Такой способ

предпочтительнее, так как прикладная программа не будет тратить время на

периодическую проверку блока ECB.

Блок ECB состоит из фиксированной части размером 36 байт и массива дескрипторов, описывающих отдельные фрагменты передаваемого или принимаемого пакета данных.

Структура блока ECB представлена ниже.



Поле **Link** предназначено для организации списков, состоящих из блоков ECB. Драйвер IPX использует это поле для объединения переданных ему блоков ECB в списки, записывая в него полный адрес в формате [сегмент : смещение].

Поле **ESRAddress** содержит полный адрес программного модуля (в формате [сегмент : смещение]), который получает управление при завершении процесса чтения или передачи пакета IPX. Этот модуль называется программой обслуживания события ESR (Event Service Routine). Если программа не использует ESR, она должна записать в поле **ESRAddress** нулевое значение. В этом случае о завершении выполнения операции чтения или передачи можно узнать по изменению содержимого поля **InUse**.

Поле **InUse** может служить индикатором завершения операции приема или передачи пакета. Перед тем как вызвать функцию IPX, программа записывает в поле **InUse** нулевое значение. Пока операция передачи данных, связанная с данным ECB, не завершилась, поле **InUse** содержит следующие значения:

· FFh - ECB используется для передачи пакета данных;

· FEh - ECB используется для приема пакета данных, предназначенного программе с конкретным сокетом;

· FDh - ECB используется функциями асинхронного управления событиями AES (Asynchronous Event Sheduler), ECB находится в состоянии ожидания истечения заданного временного интервала;

· FBh - пакет данных принят или передан, но ECB находится во внутренней очереди IPX в ожидании завершения обработки.

Программа может постоянно опрашивать поле InUse, ожидая завершения процесса передачи или приема данных. Как только в этом поле окажется нулевое значение, программа считает, что запрошенная функция выполнена. Результат выполнения можно получить в поле CCode, где после выполнения функции IPX содержится код результата

выполнения.

Если с данным ECB была связана команда приема пакета, в поле **CCode** могут находиться следующие значения:

· 00 - пакет был принят без ошибок;

· FFh - указанный в ECB сокет не был предварительно открыт программой;

· FDh - переполнение пакета: либо поле количества фрагментов в пакете FragmentCnt равно нулю, либо буферы, описанные дескрипторами фрагментов, имеют недостаточный размер для записи принятого пакета;

· FCh - запрос на прием данного пакета был отменен специальной функцией драйвера IPX.

Если ECB использовался для передачи пакета, в поле **CCode** после завершения передачи могут находиться следующие значения:

· 00 - пакет был передан без ошибок (но это не означает, что пакет был доставлен по назначению и успешно принят станцией-адресатом);

· FFh - пакет невозможно передать физически из-за неисправности в сетевом адаптере или в сети;

· FEh - пакет невозможно доставить по назначению, так как станция с указанным адресом не существует или неисправна;

· FDh - пакет сбойный, т.е. либо имеет длину меньше 30 байт, либо первый фрагмент пакета по размеру меньше размера стандартного заголовка пакета IPX, либо поле количества фрагментов в пакете FragmentCnt равно нулю;

· FCh - запрос на передачу данного пакета был отменен специальной функцией драйвера IPX.

Поле **Socket** содержит номер сокета, связанный с данным ECB. Если ECB используется для приема, это поле содержит номер сокета, на котором выполняется прием пакета. Если же ECB используется для передачи, это поле содержит номер сокета передающей программы.

Поле **IPXWorkspace** зарезервировано для использования драйвером IPX. Приложение не должно инициализировать или изменять содержимое этого поля, пока обработка ECB не завершена.

Поле **DriverWorkspace** зарезервировано для использования драйвером сетевого адаптера. Программа не должна инициализировать или изменять содержимое этого поля, пока обработка ECB не завершена.

Поле **ImmAddress** (Immediate Address - непосредственный адрес) содержит адрес узла в сети, в который будет направлен пакет. Если пакет передается в пределах одной сети, поле **ImmAddress** будет содержать адрес станции-получателя (такой же, как и в заголовке

пакета IPX). Если же пакет предназначен для другой сети и будет проходить через мост, поле **ImmAddress** будет содержать адрес этого моста в сети, из которой передается пакет.

Поле **FragmentCnt** содержит количество фрагментов, на которые нужно разбить принятый пакет, или из которых необходимо собрать передаваемый пакет.

Далее располагаются дескрипторы фрагментов, состоящие из указателя в формате [сегмент : смещение] на фрагмент **Address** и поля размера фрагмента **Size**. Если программ- ме нужно разбить принятый пакет на несколько частей, то нужно установить в поле **FragmentCnt** значение, равное количеству требуемых фрагментов. Для каждого фрагмента необходимо создать дескриптор, в котором указать адрес буфера и размер фрагмента. Аналогичные действия выполняются и при сборке пакета перед передачей из нескольких фрагментов.

**Особенности программирования протокола:**

Функция **socket**(int af, int type, int protocol) возвращает либо дескриптор созданного сокета, либо ошибку INVALID\_SOCKET. Расширенный код ошибки можно получить при помощи функции GetLastError. Параметр af содержит сведения о семействе протоколов

(AF\_INET, AF\_IPX). В данной лабораторной работе необходимо использовать константу AF\_IPX. Параметр type – тип передаваемых данных (поток или дейтаграммы). В данной лабораторной работе для IPX необходимо использовать константу DGRAM, а для SPX – константу SEQPACKET, которая означает, что пакеты будут отсылаться последовательно и в порядке очереди. Параметр protocol – протокол передачи данных. Для протокола IPX используется константа IPX, для SPX – SPX.

Чтобы работать дальше с созданным сокетом его нужно привязать к какой-либо локальной конечной точке. Этим занимается функция **bind** (EndPoint ep). ep – объект типа EndPoint который необходимо создать, он определяет сетевой адрес.

В конце работы сокет необходимо закрыть функцией **close**.

**EndPoint**(int networkNumber, long nodeNumber, short socketNumber) –создание объекта EndPoint, который определяет сетевой адрес. networkNumber – номер сети, nodeNumbe – адрес в сети, socketNumber – сокет.

**SendTo**(byte [] Message, EndPoint endPoint) – осуществляет отправку сообщения Message по адресу endPoint. В данном случае, отправка осуществлялась широковещательно.

**ReceiveFrom**(byte [] message, EndPoint endPoint) – осуществляет прием сообщения с адреса endPoint и записывает это сообщение в message.

1. **Протокол сетевого уровня IPX. Особенности программирования протокола IPX в реальном режиме работы процессора.**

**См. вопрос 12.**

**Особенности программирования протокола:**

Функция **IPXOpenSocket** предназначена для получения сокетов. На входе ВХ = 00h, AL =

Тип сокета: 00h - короткоживущий; FFh - долгоживущий, DX = Запрашиваемый номер

сокета или 0000h, если требуется получить динамический номер сокета. Байты номера

сокета находятся в перевернутом виде. На выходе AL = Код завершения (00h - сокет

открыт; FFh – этот сокет уже был открыт раньше; FEh - переполнилась таблица сокетов),

DX = Присвоенный номер сокета.

Функция **IPXCloseSocket** предназначена для освобождения сокетов, т.к. сокеты являются

ограниченным ресурсом. На входе ВХ = 0lh, DX = Номер закрываемого сокета. На выходе

регистры не используются.

Функция **IPXListenForPacket** предназначена для начала процесса приема пакетов данных

из сети. На входе ВХ = 04h, ES:SI = Указатель на заполненный блок ЕСВ. На выходе

регистры не используются. Сразу после вызова функции IPXListenForPackets в поле InUse

блока ECB устанавливается значение FEh, которое означает, что для данного блока ECB

ожидается прием пакета.

Функция **IPXSendPacket** подготавливает блок ECB и связанный с ним заголовок пакета

для передачи пакета по сети. На входе ВХ = 03h, ES:SI = Указатель на заполненный блок

ЕСВ. На выходе регистры не используются. Сразу после вызова функции IPXSendPacket,

IPX-служба ставит блок ECB в очередь на передачу и возвращает управление вызвавшей

ее программе, не дожидаясь прихода пакета, в поле InUse блока ECB устанавливается

значение FFh. Сама передача пакета происходит асинхронно по отношению к вызывавшей

ее программе.

Функция **IPXRelinquishControl** выделяет драйверу IPX процессорное время, необходимое

для его правильной работы. Если программа не использует ESR, она, должна в цикле

опрашивать поле InUse блока ECB, для которого выполняется ожидание завершения

процесса приема или передачи пакета. Однако для правильной работы драйвера IPX в

цикле ожидания необходимо вызывать эту функцию.

1. **Протокол сетевого уровня IPX. Взаимодействие прикладной программы с драйвером IPX. Формат блока ECB.**

**См. вопрос 12.**

Перед началом работы с протоколом IPX нужно проверить, установлен ли драйвер этого протокола, затем, нужно получить адрес вызова этого драйвера – точку входа API. В дальнейшем можно будет вызывать драйвер при помощи команды межсегментного вызова процедуры по адресу точки входа API драйвера IPX.

Для того чтобы проверить, загружен ли драйвер IPX, необходимо загрузить в регистр AX значение 7A00h и вызвать мультиплексное прерывание INT 2Fh. Если после возврата из прерывания в регистре AL будет значение FFh, то драйвер IPX загружен. Адрес точки входа для вызова API драйвера при этом будет находиться в регистровой паре ES:DI. Если же содержимое регистра AL после возврата из прерывания INT 2Fh будет не равно FFh, то драйвер IPX/SPX не загружен.

Для вызова API в регистр BX необходимо загрузить код выполняемой функции. Значения,

загружаемые в другие регистры, зависят от выполняемой функции.

Блок ECB состоит из фиксированной части размером 36 байт и массива дескрипторов, описывающих отдельные фрагменты передаваемого или принимаемого пакета данных.

Структура блока ECB представлена ниже.



1. **Протокол сетевого уровня IPX. Система адресации IPX. Структура пакета IPX.**

**См. вопрос 12.**

Система адресов, используемая в протоколе IPX, представлена несколькими компонентами: это номер сети, адрес станции в сети и идентификатор программы на рабочей станции.

**Номер сети** - это номер сегмента сети, определяемого системным администратором. Если в общей сети есть мосты, каждая отдельная сеть, подключенная через мост, должна иметь свой, уникальный номер сети.

**Адрес станции** - это число, которое является уникальным для каждой рабочей станции. При использовании адаптеров Ethernet уникальность обеспечивается изготовителем сетевого адаптера. Специальный адрес FFFFFFFFFFFFh используется для рассылки

данных всем станциям данной сети одновременно.

**Идентификатор программы на рабочей станции (сокет)** - число, которое используется для адресации определенной программы, работающей на компьютере.

Формат передаваемых с использованием протокола IPX по сети пакетов представлен ниже.



Пакет можно разделить на две части, поле Data представляет собой передаваемые данные, все остальные же поля представляют собой заголовок пакета.

Поле **Checksum** предназначено для хранения контрольной суммы передаваемых пакетов. При формировании пакетов не нужно заботиться о содержимом этого поля, так как проверка данных по контрольной сумме выполняется драйвером сетевого адаптера.

Поле **Length** определяет общий размер пакета. Длина заголовка фиксирована и равна 30 байт. Размер передаваемых в поле **Data** данных может составлять от 0 до 546 байт. При формировании пакетов не нужно проставлять длину пакета в поле **Length**, протокол IPX

делает это сам.

Поле **TransportControl** является счетчиком мостов, которые проходит пакет на своем пути от передающей станции к принимающей. Каждый раз, когда пакет проходит через мост, значение этого счетчика увеличивается на единицу. IPX перед передачей пакета сбрасывает содержимое этого поля в нуль. Так как IPX сам следит за содержимым этого поля, при формировании пакетов не нужно изменять или устанавливать его в какое-либо состояние.

Поле **PacketType** определяет тип передаваемого пакета. Для IPX следует установить значение равное 4.

Поле **DestNetwork** определяет номер сети, в которую передается пакет.

Поле **DestNode** определяет адрес рабочей станции, которой предназначен пакет.

Поле **DestSocket** предназначено для адресации программы, запущенной на рабочей станции, которая должна принять пакет.

Поля **SourceNetwork, SourceNode** и **SourceSocket** содержат соответственно номер сети, из которой посылается пакет, адрес передающей станции и сокет программы, передающей пакет.

Поле **Data** содержит передаваемые данные.

1. **Протокол SNMP. Особенности его программирования в защищенном режиме работы процессора.**

**SNMP** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Simple Network Management Protocol* — простой протокол сетевого управления) — стандартный интернет-протокол для управления устройствами в IP-сетях на основе архитектур [TCP](https://ru.wikipedia.org/wiki/TCP)/[UDP](https://ru.wikipedia.org/wiki/UDP). К поддерживающим SNMP устройствам относятся маршрутизаторы, коммутаторы, серверы, рабочие станции, принтеры, модемные стойки и другие. Протокол обычно используется в системах сетевого управления для контроля подключенных к сети устройств на предмет условий, которые требуют внимания администратора. SNMP определен Инженерным советом интернета (IETF) как компонент [TCP/IP](https://ru.wikipedia.org/wiki/TCP/IP). Он состоит из набора стандартов для сетевого управления, включая протокол прикладного уровня, схему баз данных и набор объектов данных.

SNMP предоставляет данные для управления в виде переменных, описывающих конфигурацию управляемой системы. Эти переменные могут быть запрошены (а иногда и заданы) управляющими приложениями.