агентство по образованию
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра теоретических основ информатики

УДК 004.4:004.7

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, проф., д. т. н.
_____ Ю.Л. Костюк
«____»_____ 2006 г.

Ишина Анна Дмитриевна

РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ ДЛЯ УЧЁТА СЕТЕВОГО ТРАФИКА ПОД ОС СЕМЕЙСТВА WINDOWS NT

Дипломная работа

Научный руководитель,
ст. преп. каф. теоретических
основ информатики, к. т. н. В. А. Лавров

Исполнитель,
студ. гр. 1411 А. Д. Ишина

Электронная версия дипломной работы помещена
в электронную библиотеку.
Администратор

Томск – 2006

Реферат

Дипломная работа 34 с., 12 рис., 18 табл., 11 источников, 2 прил.

LAYERED SERVICE PROVIDER, TRANSPORT SERVICE PROVIDER, WINDOWS SOCKETS 2 SPI, DLL, УЧЁТ СЕТЕВОГО ТРАФИКА, OVERLAPPED INPUT/OUTPUT, INPUT/OUTPUT COMPLETEION PORTS, WINDOWS.

Объект исследования – сервис провайдеры, Winsock SPI, стек протоколов TCP/IP;

Цель работы – разработка приложения для учёта сетевого трафика;

Метод проведения работы – теоретический и практический;

В результате был разработан свой сервис провайдер в виде динамической библиотеки (DLL), установщик и пользовательское приложение для работы с сервис провайдером;

Содержание

Введение

Бурное развитие сетевых технологий привело к тому, что сейчас сложно себе представить изолированный компьютер, который тем или иным способом не использовал бы подключение к Интернету. По данным Miniwatts Marketing Group на 31.03.2006 количество пользователей глобальной сети в России составило 23,7 млн. человек, то есть каждый шестой житель нашей страны имеет выход в Интернет. Причём за последние пять лет число людей, использующих глобальную сеть, выросло более чем в 6,5 раз [1]. Средние мировые оценки мало отличаются от российских (см. табл. 1).

Таблица 1 – Мировая статистика использования Интернет [1]

Часть света	Население, чел. (оценка на 2006 год)	Население, % от всего населения мира	Использует Интернет, чел. (данные за 31.03.2006 года)	Использует Интернет, % от всего населения (проникновение)	Рост числа пользователей Интернет за 2000-2005 годы
Африка	915 210 928	14,1%	23 649 500	2,6%	423,9%
Азия	3 667 774 066	56,4%	364 270 713	9,9%	218,7%
Европа	807 289 020	12,4%	291 600 898	36,1%	177,5%
Ближний Восток	190 084 161	2,9%	18 203 500	9,6%	454,2%
Северная Америка	331 473 276	5,1%	227 303 500	68,6%	110,3%
Латинская Америка	553 908 632	8,5%	79 962 809	14,4%	342,5%
Австралия и Океания	33 956 977	0,5%	17 872 707	52,6%	134,6%
Всего	6 499 697 060	100%	1 022 863 307	15,7%	183,4%

Все пользователи сети обмениваются друг с другом информацией, порождая потоки данных, которые необходимо контролировать. Поэтому учёт сетевого трафика – одна из наиболее актуальных задач, с которой сталкивается каждый администратор сети и, очень часто, сам пользователь.

В рамках данной работы предложен вариант реализации приложения для MS Windows в виде сервис провайдера, которое выполняет учёт сетевого трафика, как на отдельной локальной машине, так и на сервере. Оно состоит из трёх компонентов:

- динамической библиотеки;
- приложения для установки динамической библиотеки;
- пользовательского приложения для просмотра статистики.

1 Постановка задачи

1.1 Необходимость учёта трафика

Сейчас всё более распространёнными становятся выделенные каналы доступа к Интернету, при использовании которых оплата берётся не за время, проведённое в сети, а за входящий трафик. В такой ситуации его учёт просто необходим.

В любом случае трафик контролируется провайдером. Но если одним каналом пользуется несколько человек, может возникнуть необходимость считать трафик для каждого из них по отдельности. Например, это необходимо в Интернет-клубах.

В корпоративной сети также необходимо следить, какой сотрудник сколько скачивает информации. Анализируя трафик, можно также определить, что это за информация, а, следовательно, и проконтролировать, сколько времени сотрудник тратит на работу, а сколько на другие занятия.

Также учёт трафика часто нужен для диагностики сети. Например, для выявления её «узких» мест. Отслеживая реальную скорость передачи данных между хостами, это легко определить [2].

Неконтролируемый доступ к выделенному каналу может приносить также и косвенные убытки, в связи с перегрузкой корпоративной локальной сети. С этим можно бороться за счёт ограничения сетевого трафика. [3]. Это может быть необходимым и в целях контроля расходов или регулирования доступа к какой-то информации.

Проблема управления сетевым трафиком также актуальна для владельцев web- и ftp-серверов на платформе MS Windows. Это связано с тем, что сейчас широко распространены программы для скачивания информации, позволяющие открывать одновременно несколько сессий, что ведёт к перегрузке канала. Выходом из этой ситуации может стать введение ограничений на скорость и количество подключений к серверу. К сожалению, стандартные средства MS Windows не обладают необходимой для этого функциональностью.

1.2 Основные способы, используемые для учёта трафика в существующих системах

Осознав необходимость учёта и контроля сетевого трафика, пользователи обычно сталкиваются с другой проблемой: как выбрать подходящее приложение. Существует достаточно много программных продуктов, предоставляющих возможность управления сетевым трафиком. Но б*о*льшая их часть предназначена для ОС семейства UNIX. Многие из программ, работающих под Windows, слишком дороги, перегружены функциональностью и не отличаются гибкостью настроек.

В основном, приложения для управления трафиком основываются на трёх технологиях: прослушивание пакетов с помощью включения promiscuous mode на сетевой плате, анализ информации, предоставляемой другой программой (proxy-сервером, брандмауэром) или аппаратурой (например, поддерживающей CISCO NetFlow [4]) и обработка потоков данных специальным драйвером. Рассмотрим достоинства и недостатки этих подходов.

Программы для прослушивания сетевого трафика обычно называют снифферами. Прослушивание возможно благодаря особенности архитектуры Ethernet. Информацию, передаваемую по сети, получает каждое устройство. По умолчанию сетевая плата компьютера видит только то, что предназначено именно для неё. Но прослушивающие программы включают режим приёма всех пакетов – promiscuous mode. В основе снифферов лежат сетевые драйверы и библиотеки, которые осуществляют б*о*льшую часть работы. До недавнего времени создание таких приложений было уделом квалифицированных специалистов, но с появлением MS Windows 2000 создать программу для прослушивания сегмента сети стало

достаточно просто. При интенсивной загрузке локальной, сети компьютер и программа-сниффер могут оказаться не способными обработать всю поступающую информацию. На этом факте основаны некоторые программы для борьбы с прослушиванием. Они посылают в сеть множество пакетов с поддельной информацией, и в результате снифферы «захлёбываются» [5]. Ещё один минус данного подхода: ограниченные возможности такой программы. Этим способом можно считать трафик, но нельзя его ограничивать.

Главный недостаток второго метода заключается в том, что приложения, считающие трафик, на основе информации полученной из другой программы или специального оборудования, не являются полностью самостоятельными и независимыми. Это требует дополнительных финансовых затрат и лишает данный подход универсальности. К тому же при работе через proxy-сервер подсчитывается не весь трафик, так как не учитываются заголовки пакетов и служебный TCP трафик, что приводит к занижению общего трафика на 5-15% [6]. Преимущество данного подхода в том, что исходная информация остаётся нетронутой, поэтому при изменении условий можно перенастроить фильтры в анализаторе логов и сделать пересчёт трафика. Также этот способ может быть удобным, если программа, формирующая логи, уже используется для каких-то других целей, тогда для учёта трафика будет достаточно установить нужный plug-in.

Приложения для учёта трафика, основанные на драйвере, отличаются точностью и высокой производительностью. Хотя и здесь возможны проблемы: при большой нагрузке некоторые пакеты могут теряться. Нужно учесть и тот факт, что любой драйвер работает в режиме ядра, поэтому неполадки в программе могут оказать губительное воздействие на всю систему.

Способ учёта трафика, предложенный в данной работе, основан на реализации сервис провайдера, который получает и обрабатывает информацию обо всех операциях ввода / вывода, запрашиваемых пользовательским приложением через Windows Sockets (Winsock) Application Programming Interface (API). Эта программа может использоваться независимо от установленного оборудования и программного обеспечения на платформе MS Windows. Она полностью самостоятельна и точно подсчитывает трафик. Хотя возможна ситуация, когда пользовательское приложение использует Transport Data Interface (TDI), минуя Winsock. Пакеты, посланные таким образом, не будут учтены [7]. Но обращение к TDI напрямую в приложениях используется редко, поэтому этим фактом можно пренебречь. Одно из преимуществ сервис провайдеров заключается в том, что программы, построенные по этой технологии, лишены недостатка, присущего сетевым драйверам, так как они загружаются в контексте определённых процессов и не затрагивают ядро.

Приложения для учёта сетевого трафика, построенные предложенным способом пока не распространены. Хотя эта технология активно используется для реализации других функций.

1.3 Выводы

Учёт и управление сетевым трафиком – актуальная проблема практически для каждого пользователя глобальной сети, решение которой не всегда очевидно. Это связано с тем, что существующие на данный момент программные продукты обладают рядом недостатков, которые не всегда позволяют их использовать. Среди них основными являются:

- дороговизна;
- недостаточная точность;
- недостаточная функциональность.

В связи с этим, было принято решение разработать программный продукт, ориентированный на платформы, работающие под ОС Windows, который при этом обладал бы следующими качествами:

- не зависел от установленного в системе ПО (кроме ОС) и аппаратуры;
- точно подсчитывал TCP и UDP трафик;
- обладал возможностями для расширения функциональности (можно было добавить ограничение трафика и количества подключений с одного IP-адреса).

2 Структура приложения для учёта сетевого трафика

2.1 Архитектура библиотеки Winsock

Winsock – библиотека, разработанная компанией Microsoft для реализации сетевых приложений под ОС Windows. Она построена на основе модели Windows Open System Architecture (WOSA) (см. рис. 1) и позволяет разрабатывать собственные сервис провайдеры (сетевые службы), которые будут влиять на все приложения, использующие Winsock API. При этом не нужно будет переписывать код этих приложений или заменять Winsock 2 DLL (Ws2_32.dll). Для разработки сервис провайдеров Winsock включает в себя Service Provider Interface (SPI).

<table>
<tr><td colspan="2">Пользовательское приложение</td><td colspan="2">Пользовательское приложение</td></tr>
<tr><td colspan="4">Winsock 2 API</td></tr>
<tr><td colspan="4">Winsock 2 DLL
WS2_32.DLL</td></tr>
<tr><td colspan="2">Транспортные функции

Winsock 2 SPI для транспортных сервис провайдеров</td><td colspan="2">Функции для работы с пространством имён (namespace)

Winsock 2 SPI для работы с пространством имён</td></tr>
<tr><td>Транспортный сервис провайдер</td><td>Транспортный сервис провайдер</td><td>Сервис провайдер пространства имён</td><td>Сервис провайдер пространства имён</td></tr>
</table>

Рисунок 1 – Архитектура библиотеки Winsock 2 [8]

Winsock 2 SPI позволяет разрабатывать два типа сервис провайдеров: транспортные сервис провайдеры и сервис провайдеры пространства имён. Транспортные сервис провайдеры выполняют функции, отвечающие за установку соединения, передачу данных, управление трафиком и контроль количества ошибок. Сервис провайдеры пространства имён предоставляют функции разрешения имён.

Транспортные сервис провайдеры в свою очередь делятся на базовые (base service providers) и многоуровневые (layered service providers). Базовые реализуют низкоуровневые функции транспортного протокола, многоуровневые выполняют функции более высокого уровня и полагаются на нижележащий базовый сервис провайдер (см. рис. 2).

Базовые сервис провайдеры и провайдеры пространства имён обычно поставляются производителями операционной системы или стека протоколов. Расширение базовых функций возможно с помощью разработки дополнительных многоуровневых сервис провайдеров. Но при этом нужно иметь в виду, что воспользоваться этой функциональностью смогут только приложения, использующие Winsock API [8].

Каждый транспортный сервис провайдер поддерживает один или несколько протоколов. Например, TCP/IP провайдер должен поддерживать, как минимум, протоколы TCP и UDP, IPX/SPX провайдер – IPX, SPX и SPX II. Каждый протокол, поддерживаемый конкретным провайдером, описывается структурой WSAPROTOCOL_INFOW, а набор таких структур может быть представлен как каталог установленных протоколов [9].

Многоуровневые и базовые сервис провайдеры связываются вместе и образуют цепочку протокола. Для описания цепочки в целом также используется структура WSAPROTOCOL_INFOW. Она описывает порядок, в котором связаны провайдеры [8].

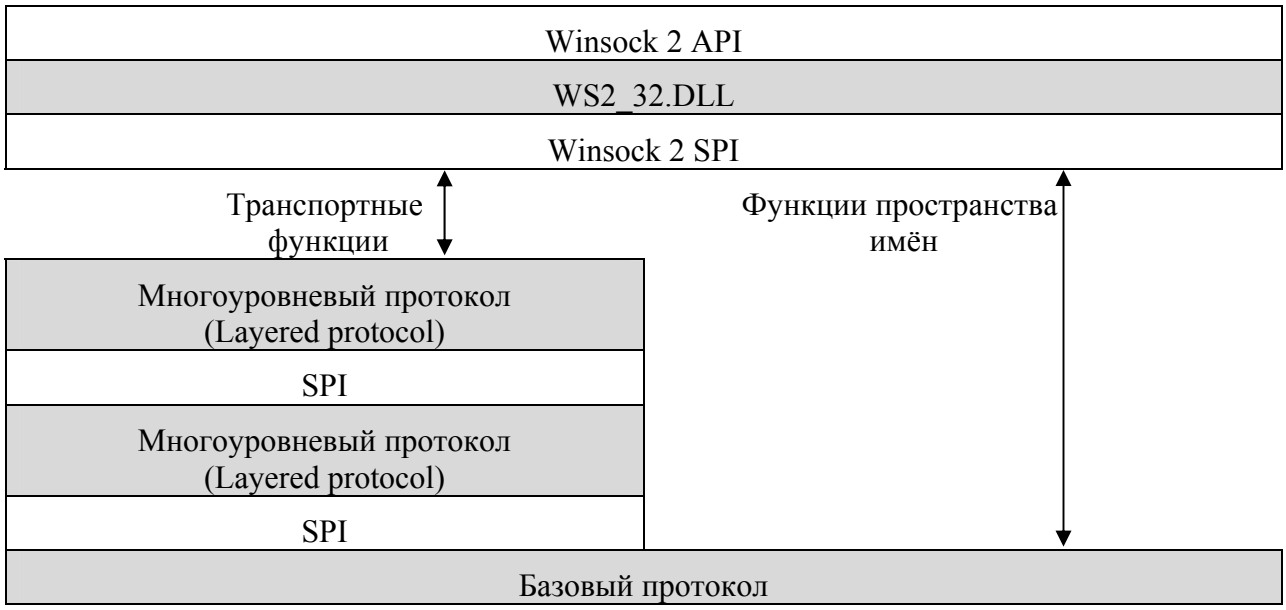

Рисунок 2 – Схема взаимодействия Ws2_32.dll и сервис провайдеров [8]

2.2 Структура многоуровневого сервис провайдера

Любой многоуровневый сервис провайдер представляет собой динамическую библиотеку с одной экспортируемой точкой входа – инициализирующей функцией: WSPStartup(). Доступ ко всем остальным функциям сервис провайдера Ws2_32.dll получает через таблицу адресов функций (dispatch table), предоставляемую провайдером. SPI также позволяет транспортным сервис провайдерам вызывать функции Ws2_32.dll. Для этого Ws2_32.dll передаёт свою таблицу адресов функций (upcall dispatch table) как параметр в инициализирующую функцию [9].

Структура WSPPROC_TABLE содержит функции, которые должны быть реализованы в многоуровневом сервис провайдере и чьи адреса используются для заполнения выходного параметра функции WSPStartup(). Каждый многоуровневый сервис провайдер должен реализовать 30 функций. Но в большинстве случаев полная реализация необходима только для некоторых из них. Внутри остальных функций будет достаточно просто передать вызов нижележащему сервис провайдеру.

Динамическая библиотека сервис провайдера загружается либо другим провайдером, либо библиотекой Ws2_32.dll (в зависимости от положения провайдера в цепочке). Сразу после загрузки вызывается функция WSPStartup(). Она может вызываться несколько раз. Для каждого успешного вызова WSPStartup() будет вызвана WSPCleanup(). Поэтому каждый сервис провайдер должен иметь счётчик ссылок, который будет увеличиваться на единицу при вызове WSPStartup() и уменьшаться при вызове WSPCleanup(). Если этот счётчик станет равным нулю, сервис провайдер должен подготовиться к выгрузке.

Когда пользовательское приложение вызывает какую-либо Winsock API-функцию, Ws2_32.dll вызывает соответствующую Winsock SPI-функцию. Но не все Winsock API-функции имеют соответствующие Winsock SPI-функции. Некоторые из них реализованы в Ws2_32.dll и их вызовы не передаются сервис провайдеру. Также не передаются сервис провайдерам вызовы функций для работы с объектами синхронизации и функции ожидания: они напрямую передаются службам ОС Windows [8].

2.3 Установка сервис провайдера

Для того чтобы функциональность, предоставляемая сервис провайдером, стала доступна, он должен быть корректно установлен и зарегистрирован библиотекой Winsock. Для этого используется отдельное инсталляционное приложение, которое обычно предоставляет разработчик сервис провайдера. Инсталляционная программа добавляет необходимую информацию в каталог Ws2_32.dll. Для этого используется функция WSCInstallProvider(). С помощью вызова WSCDeinstallProvider() эту информацию можно удалить из каталога Winsock.

Порядок, в котором сервис провайдеры устанавливаются, определяет очерёдность, в которой они будут вызываться при обработке запроса клиента на создание сокета. При регистрации новый сервис провайдер помещается в конец каталога. Поэтому, чтобы сделать его сервис провайдером по умолчанию, каталог Ws2_32.dll нужно переупорядочить. Winsock предоставляет такую возможность с помощью функции WSCWriteProviderOrder(), которая реализована в дополнительной динамической библиотеке, Sporder.dll.

Структура WSAPROTOCOL_INFOW, предоставляемая каждым провайдером при установке, содержит информацию о том, какого типа этот провайдер - базовый, многоуровневый или это цепочка протоколов. Установка цепочки протоколов возможна только после успешной инсталляции всех её компонентов [8, 9].

2.4 Обработка операций ввода / вывода

Для учёта трафика с помощью сервис провайдера необходимо соответствующим образом реализовать функции, отвечающие за операции ввода / вывода. Это такие функции как WSPRecv(), WSPRecvFrom(), WSPSend(), WPSendTo() и WSPIoctl().

2.4.1 Блокирующий ввод / вывод

Библиотека Winsock поддерживает три вида операций ввода/вывода. Самый простой из них – блокирующий, он используется по умолчанию. Блокирующая операция ввода / вывода возвращает управление только после полного её завершения. Поэтому каждый поток может одновременно выполнять только одну операцию. Например, если поток принимает данные, но на текущий момент ничего не поступило, поток блокируется. Этот вид операций ввода / вывода прост, но не всегда эффективен [9].

2.4.2 Неблокирующий ввод / вывод

Если сокет переведён в неблокирующеее состояние, то любая операция будет сразу выполняться или возвращать код ошибки WSAEWOULDBLOCK, обозначающий, что операция не может быть завершена немедленно. В последнем случае необходим механизм, который бы позволял определить момент, в который нужно вызвать операцию повторно, чтобы она успешно завершилась. Для этой цели был определён набор сетевых событий. Для переключения сокета в режим асинхронной доставки сетевых событий используются функции WSPAsyncSelect() и WSPEventSelect(). После этого можно использовать функцию WSPSelect() для определения состояния сокета [9].

2.4.3 Отложенный ввод / вывод

Winsock 2 поддерживает также отложенный (overlapped) ввод / вывод и требует, чтобы все транспортные сервис провайдеры его поддерживали. Операции отложенного

ввода / вывода можно совершать над сокетами, которые были созданы с помощью WSPSocket() с указанием флага WSA_FLAG_OVERLAPPED.

Для получения данных клиент использует WSPRecv() или WSPRecvFrom(). При этом он предоставляет буферы для входной информации. Если вызов был сделан до поступления данных, то они сразу помещаются в буфер клиента, что позволяет избежать лишней операции копирования. В случае поступления данных до предоставления буфера, сервис провайдер прибегает к синхронному стилю операций, когда входные данные сохраняются во внутреннем буфере до тех пор, пока клиент не вызовет операцию WSPRecv() или WSPRecvFrom().

Для отправки данных используются функции WSPSend() и WSPSendTo(), в качестве параметров которых клиент передаёт указатель на буфер с данными. При этом подразумевается, что содержимое буфера не будет изменяться, пока данные не будут отправлены.

WSPIoctl() используется для вызова расширенных операций ввода / вывода [9].

Отложенные операции отправки и получения данных возвращают управление немедленно. Если они возвращают 0, это означает, что операция была завершена успешно. То есть связанный с данной операцией объект типа «событие» переведён в сигнальное состояние, либо соответствующая завершающая процедура поставлена в очередь. Если же операция возвращает SOCKET_ERROR, и при этом код ошибки равен WSA_IO_PENDING, это означает, что операция была успешно начата и в дальнейшем будет передано уведомление об отправке или получении данных. Любой другой исход сигнализирует об ошибке [9].

2.4.4 Порты завершения ввода / вывода

Порт завершения ввода/вывода – это механизм, с помощью которого приложение может использовать пул потоков (pool of threads) для обработки запросов на асинхронный ввод/вывод (asynchronous I/O requests). Этот способ является наиболее эффективным [9].

Один порт завершения ввода/вывода можно связать с одним или более дескриптором файла (или сокета) с помощью вызова CreateIoCompletionPort(). Когда отложенная операция ввода/вывода заканчивается, в очередь к порту посылается пакет с информацией о завершённой операции (I/O completion packet). Таким образом, этот механизм может быть использован для создания одной точки синхронизации для нескольких дескрипторов объектов.

Управляющий поток может использовать функцию GetQueuedCompletionStatus() для ожидания пакета с информацией о завершённой операции. Потоки блокируются в очереди к порту на завершение и освобождаются в порядке LIFO (last-in-first-out): когда поступает пакет завершения, система пробуждает последний заблокированный поток [9].

2.5 Схема работы приложения для учёта сетевого трафика

Схема работы приложения для учёта сетевого трафика представлена на рис. 3. В качестве примера приведён порядок передачи и обработки вызова recv().

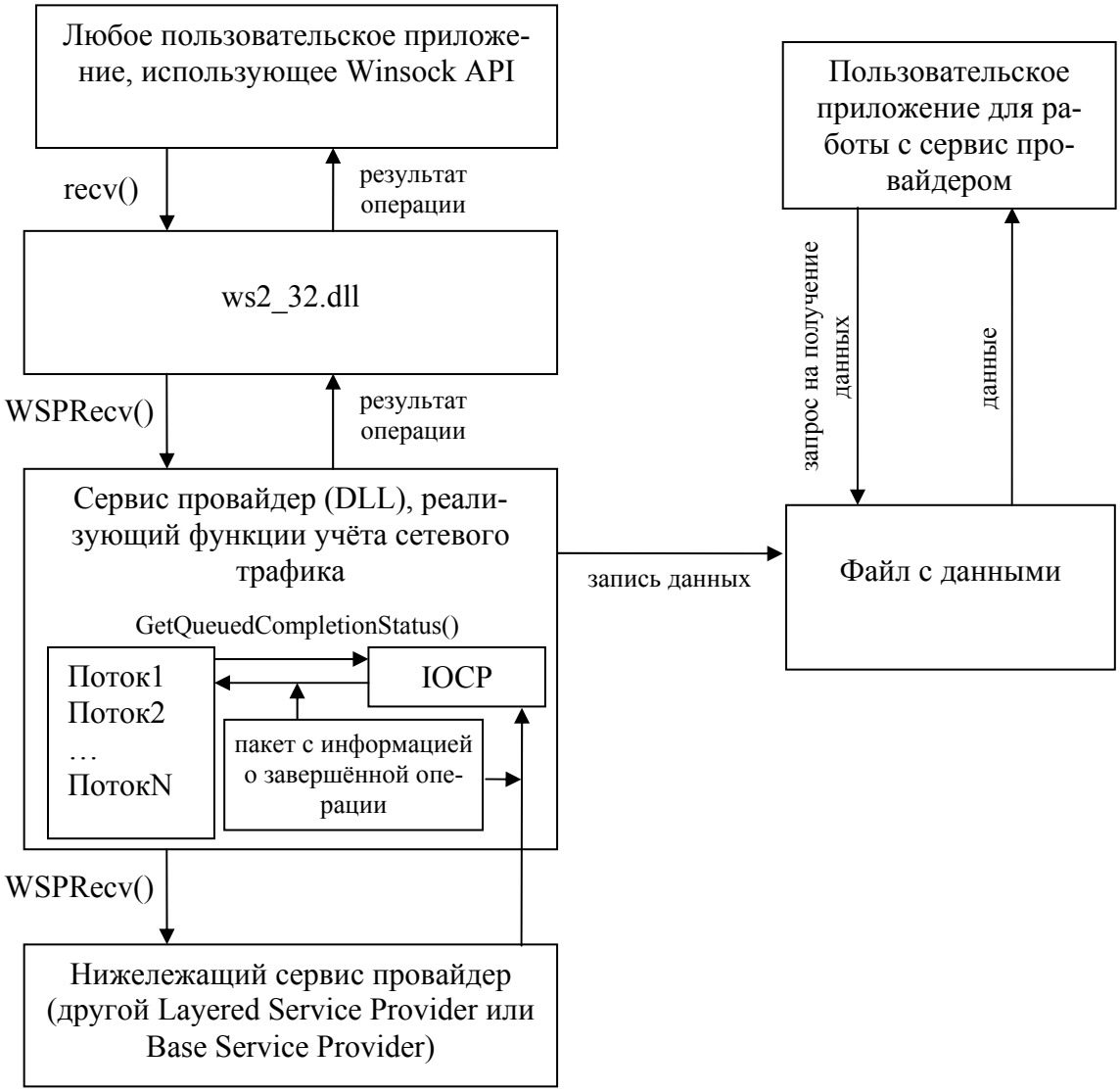

Рисунок 3 – Схема работы приложения для учёта сетевого трафика

2.6 Хранение и обработка динамической информации

При выборе формата хранения динамической информации, было рассмотрено три варианта:

- база данных;
- XML-файл;
- CSV-файл.

В итоге был выбран третий способ. Такой выбор был сделан в связи с тем, что структура данных, которую необходимо хранить, слишком проста, и применение базы данных в этом случае было бы слишком громоздким. Хранение статистики в XML-файле в данном случае также представляется не оптимальным, так как этот формат по сравнению с CSV сложнее и требует б*о*льших накладных расходов [11].

Таким образом, было принято решение необходимые данные хранить в CSV-файле. Запись в файл осуществляет сервис провайдер при закрытии сокета. При этом сохраняется следующая информация:

- время с точностью до миллисекунд;

- IP-адрес и порт источника;
- IP-адрес и порт назначения;
- тип протокола, по которому были переданы данные;
- количество переданных байт;
- количество принятых байт.

Для обработки файла с данными внутри пользовательского приложения реализован конечный автомат, который преобразует текстовую информацию в более удобную для обработки структуру. Затем полученная структура отображается в понятной пользователю форме.

2.7 Выводы

В результате проведённого исследования были разработаны структуры:

- собственного сервис провайдера;
- приложения для установки динамической библиотеки;
- приложения для обработки накопленных данных.

3 Реализация приложения для учёта сетевого трафика

Предложенный в данной работе программный продукт реализован в виде трёх компонентов:

- windows-приложения для установки сервис провайдера;
- динамической библиотеки (сервис провайдера);
- windows-приложения, предназначенного для управления сервис провайдером.

Для разработки использовалась среда Microsoft Visual Studio.NET 2003, язык C++.

Рассмотрим каждый из этих компонентов отдельно.

3.1 Приложение для установки сервис провайдера

Общая схема работы приложения приведена на рис. 4.

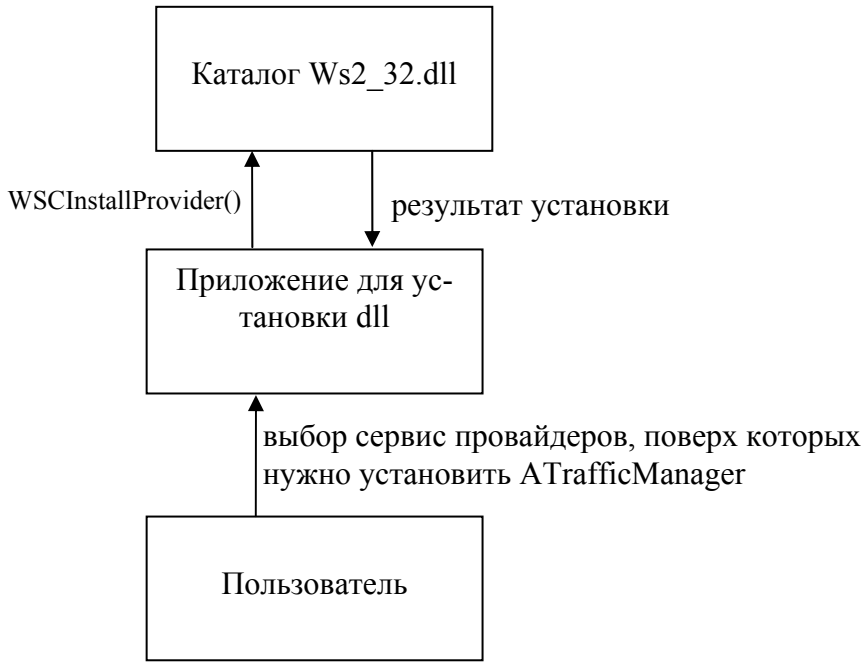

Рисунок 4 – Схема работы приложения для установки сервис провайдера.

Входными данными для этого приложения являются идентификаторы сервис провайдеров, поверх которых необходимо установить динамическую библиотеку. Они зависят от конкретной версии операционной системы, а также от уже установленных сервис провайдеров. Поэтому наиболее оптимальным вариантом в данной ситуации является предоставление пользователю возможности выбрать самому сервис провайдеры, поверх которых нужно установить динамическую библиотеку. Для этого установочное приложение выводит список установленных сервис провайдеров с краткой информацией о них, из которого пользователь может выбрать необходимые.

После получения входной информации, приложение начинает установку сервис провайдера. Сначала динамическая библиотека устанавливается как спрятанный элемент каталога с помощью функции WSCInstallProvider(). При этом в структуре WSAPROTOCOL_INFOW dwProviderFlags должен содержать бит PFL_HIDDEN. Это необходимо для получения идентификатора элемента каталога, которому будет соответствовать

провайдер. Затем регистрируются цепочки провайдеров. Для этого также используется функция WSCInstallProvider(). При этом для каждой цепочки необходимо сгенерировать свой глобальный идентификатор. После успешной установки цепочки, содержащие провайдер, помещаются в начало каталога с помощью функции WSCWriteProviderOrder().

Удаление провайдера также производит установочное приложение. Деинсталляция происходит в два этапа: сначала удаляются все цепочки, в которые входит сервис провайдер, а затем удаляется сам провайдер. Это делается с помощью функции WSCDeinstall-Provider(). При этом если в системе есть провайдеры, установленные поверх удаляемого, то их также необходимо переустановить. Для этого они сначала удаляются, а затем после коррекции цепочек заново устанавливаются с помощью функции WSCInstallProvider().

3.2 Динамическая библиотека, реализующая функции управления трафиком

Основная функциональность динамической библиотеки заключена в реализации функций WSPRecv(), WSPRecvFrom(), WSPSend(), WSPSendTo() и WSPIoctl(). Все они построены примерно по одной схеме. На рисунках 5 и 6 приведены примеры работы сервис провайдера при вызове пользовательским приложением функции recv() в двух разных случаях: при использовании отложенного ввода / вывода и при использовании блокирующего или неблокирующего ввода / вывода.

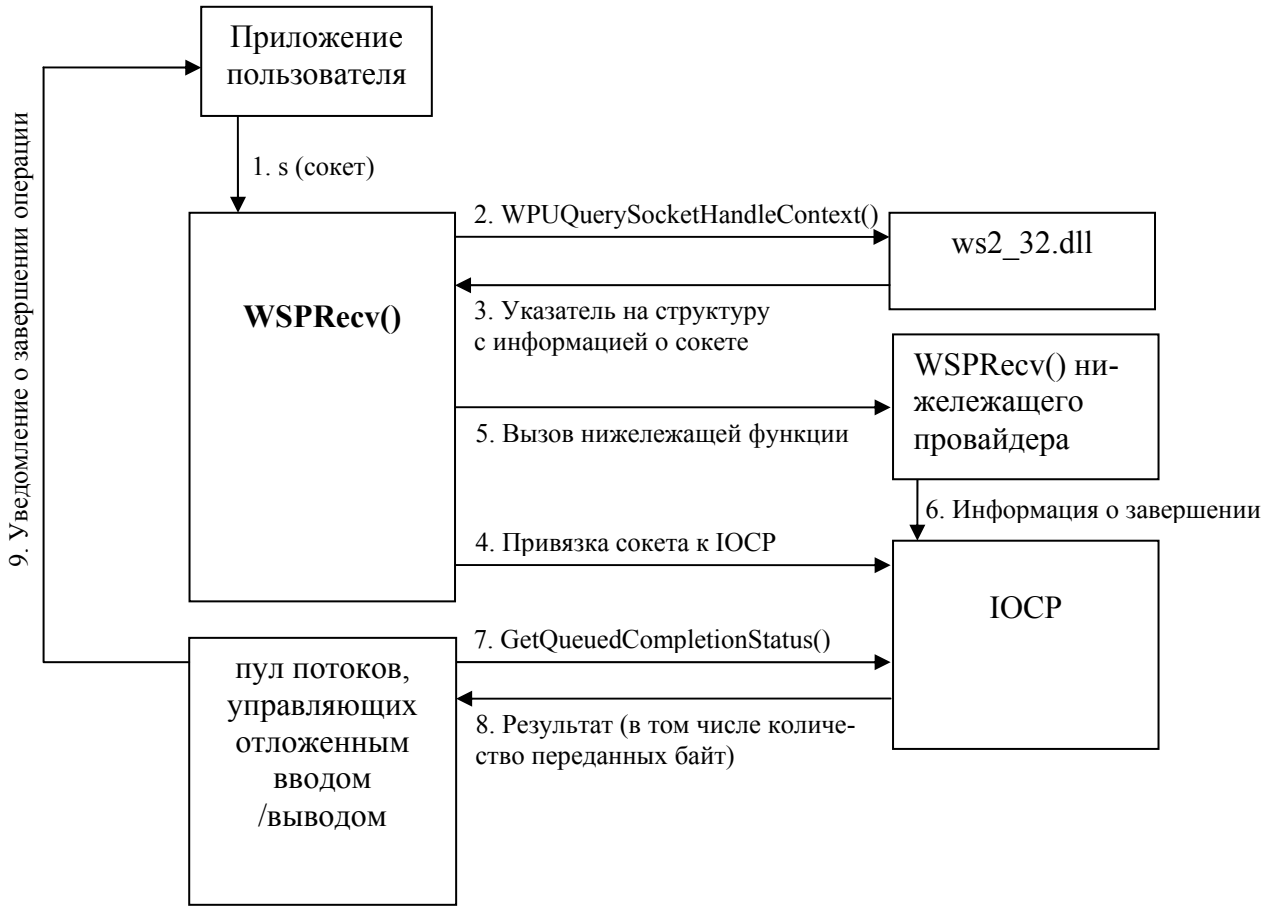

Рисунок 5 – Схема обработки вызова recv() в случае отложенного ввода / вывода.

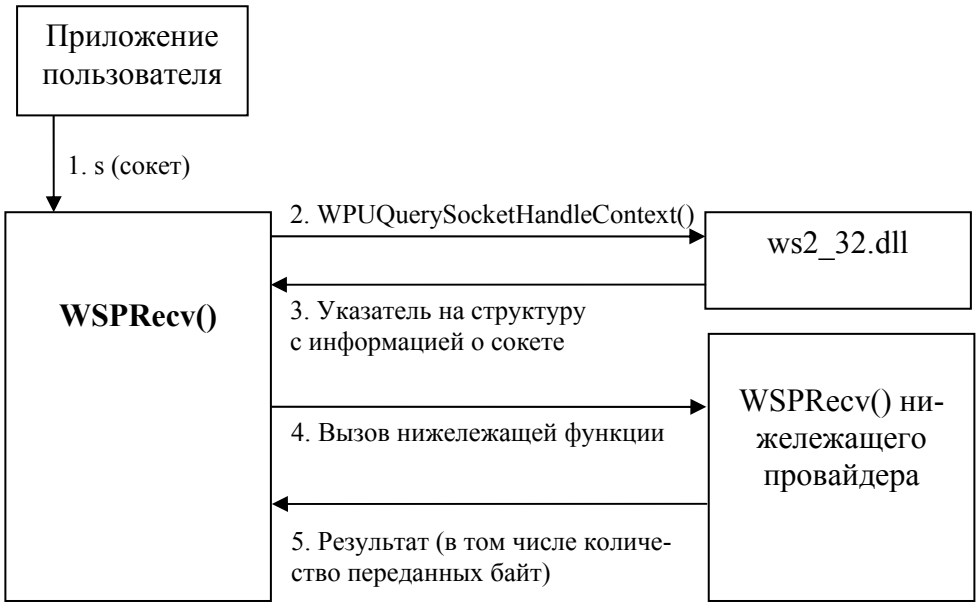

Рисунок 6 – Схема обработки вызова recv() в случае блокирующего или неблокирующего ввода / вывода.

Вся необходимая информация о сокете хранится в специальной структуре. При закрытии сокета клиентским приложением, то есть при вызове WSPSocket(), данные из этой структуры сохраняются на диск в CSV-файл, расположенный в папке «Мои документы» текущего пользователя. При этом если файл данных ещё не существует, он создаётся и в него записывается заголовок. Если же файл уже был создан ранее, данные просто дописываются в конец.

Информация записывается в файл в следующем порядке:

- время с точностью до миллисекунд;
- IP-адрес источника;
- порт источника;
- IP-адрес назначения;
- порт назначения;
- тип сокета (TCP / UDP);
- количество переданных байт;
- количество принятых байт.

3.3 Пользовательское приложение для работы с сервис провайдером

Общая схема работы пользовательского приложения представлена на рис 7.

По запросу пользователя приложение обращается к CSV-файлу, считывает его и с помощью конечного автомата преобразует в удобную для дальнейшей обработки структуру. После обработки данные выводятся в понятной человеку форме. Матрица переходов конечного автомата представлена в таблице 2.

Рисунок 7 – Схема работы пользовательского приложения

Таблица 2 – Матрица переходов конечного автомата, обрабатывающего CSV-файл с данными о сетевом трафике

	Любой символ (кроме «,», возврата каретки, перехода на новую строчку и нулевого символа)	«,»	Конец буфера	Символ возврата каретки, перехода на новую строчку или нулевой символ
S	A	B	S	S
A	A	B	A	O
B	B	C	B	O
C	C	D	C	O
D	D	E	D	O
E	E	F	E	O
F	F	G	F	O
G	G	H	G	O
H	H	O	H	Z

Список состояний конечного автомата:

- S – начальное;
- A – считывание времени;
- B – считывание IP-адреса источника;
- C – считывание порта источника;
- D – считывание IP-адреса назначения;
- E – считывание порта назначения;
- F – считывания типа сокета;
- G – считывание количества переданных байт;
- H – считывание количества принятых байт;
- O – ошибочное;
- Z – конечное.

3.4 Выводы

Таким образом, в рамках данной работы было реализовано следующее:

- установка сервис провайдера;
- обработка операций ввода / вывода сервис провайдером;
- сохранение динамических данных в CSV-файл;
- преобразование CSV-файла во внутреннюю структуру данных с помощью конечного автомата;
- обработка преобразованных данных и вывод их пользователю.

Заключение

В результате проведённой работы нами было сделано следующее:

- изучена архитектура Winsock SPI;
- разработана архитектура собственного сервис провайдера, приложения для его установки и пользовательского приложения для обработки собранных данных;
- разработанные архитектуры реализованы в виде windows-приложений, работающих под ОС версии Windows 2000 и выше.

Сервис провайдер разрабатывался с учётом дальнейшего расширения набора его свойств, таких как ограничение трафика (по объёму и скорости), а также количества подключений.

Список использованных источников

1. Internet usage statistics. // Internet World Stats. [Электронный ресурс]. – 2006. – Режим доступа: http://www.internetworldstats.com/, свободный.
2. TMeter. Описание. // TMeter Web Online Help. [Электронный ресурс]. – 2006. – Режим доступа: http://www.tmeter.ru/tmeter/manual/, свободный.
3. Абсалямов А. Учёт Internet-трафика в локальных сетях. // Windows IT Pro. [Электронный ресурс]. – 2002. – №5. – Режим доступа: http://www.osp.ru/win2000/, свободный.
4. Cisco IOS IPsec Accounting with Cisco IOS NetFlow. // Сайт Cisco Systems. [Электронный ресурс]. – 2006. – Режим доступа: http://www.cisco.com/ , свободный.
5. Максимов К. Сниффер: щит и меч. // RSDN. [Электронный ресурс]. – 2005. – Режим доступа: http://www.rsdn.ru/article/net/sniffer.xml, свободный.
6. Traffic Inspector. Сравнение с другими продуктами. // Сайт SMART-SOFT. [Электронный ресурс]. – 2005. – Режим доступа: http://www.smart-soft.ru/
7. Windows Network Data and Packet Filtering. // NDIS Developer's Reference. [Электронный ресурс]. – 2003. – Режим доступа: http://www.ndis.com/, свободный.
8. Hua W., Ohlund J, Butterklee B. Unraveling the Mysteries of Writing a Winsock 2 Layered Service Provider // Microsoft Systems Journal. [Электронный ресурс]. – May 1999. – V. 14. - № 5. – Режим доступа к журн.: http://www.microsoft.com/msj/, свободный.
9. MSDN Library. [Электронный ресурс]. – January 2006.
10. Jones A., Ohlund J. Network Programming for Microsoft Windows [Электронный ресурс]. – 1999.
11. Repici D. J. The Comma Separtated Value (CSV) File Format // Creativyst Docs. [Электронный ресурс]. – 2006. – Режим доступа: http://www.creativyst.com/, свободный.

Приложение А. Руководство пользователя

Пользователю приложение ATrafficManager поставляется в виде динамической библиотеки ADll.dll (сервис провайдер), двух исполняемых файлов: Installer.exe (приложение для установки и удаления сервис провайдера) и UserApp.exe (приложение для работы с сервис провайдером).

Для работы приложения необходима ОС Windows, в состав которой входит библиотека Winsock версии 2.2. (В Windows 2000/XP/2003 она уже включена, для более ранних версий Windows необходимо установить обновление).

Установка и удаление сервис провайдера

Для установки сервис провайдера:

1. Скопируйте динамическую библиотеку ADll.dll в папку %SystemRoot%\System32 (%SystemRoot% - системная папка Windows, например «C:\Windows»).
2. Запустите файл Installer.exe.
3. Выберите пункт «Установка» и нажмите кнопку «Далее» (см. рис. А.1).

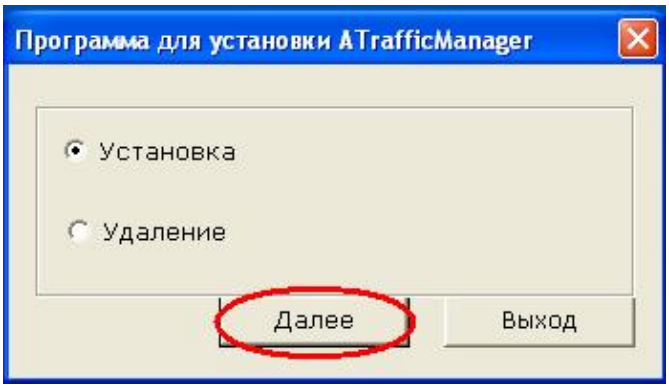

Рисунок А.1 – Установка сервис провайдера

4. Данную динамическую библиотеку нужно установить поверх протоколов TCP, UDP и RAW IP. Если в вашей системе не устанавливались никакие сервис провайдеры, помимо тех, которые идут вместе с ОС Windows, необходимо выбрать "MSAFD Tcpip [TCP/IP]", "MSAFD Tcpip [UDP/IP]" и "MSAFD Tcpip [RAW/IP]" (см. рис. А.2). Выберите соответствующие провайдеры из списка и нажмите «ОК».
5. После успешной установки вы получите сообщение: «ATrafficManager был успешно установлен».

Если на шаге 5 вы получили сообщение «ATrafficManager уже установлен», это означает, что в вашей системе данный сервис провайдер зарегистрирован, и если это сделано корректно, то нет необходимости устанавливать его снова. Если же вы хотите переустановить ATrafficManager, то перед этим его сначала нужно удалить.

Если же вы получили одно из сообщений: «Ошибка при установке элемента ATrafficManager каталога Winsock», «Ошибка при установке цепочки» или «Ошибка при переупорядочивании каталога Winsock», проверьте, достаточно ли у вас прав. Для того чтобы установить сервис провайдер, вы должны обладать правами администратора.

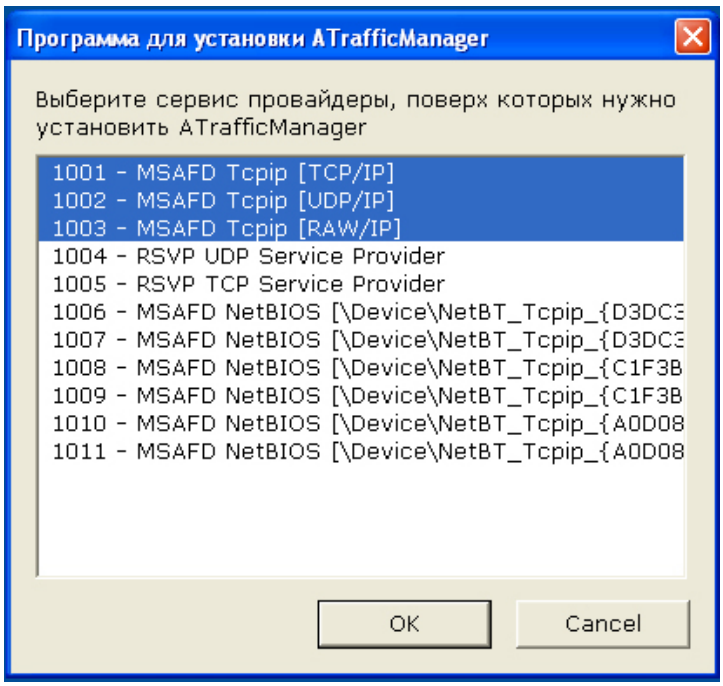

Рисунок А.2 – Выбор сервис провайдеров, поверх которых необходимо установить ATrafficManager

Чтобы удалить сервис провайдер из системы достаточно ещё раз запустить Installer.exe, выбрать опцию «Удаление» и нажать кнопку «Далее» (см. рис. А.3). В случае успешной деинсталляции вы увидите сообщение «ATrafficManager был успешно удалён». Если вы получили сообщение «ATrafficManager не был установлен», это означает, что данный сервис провайдер не зарегистрирован в вашей системе.

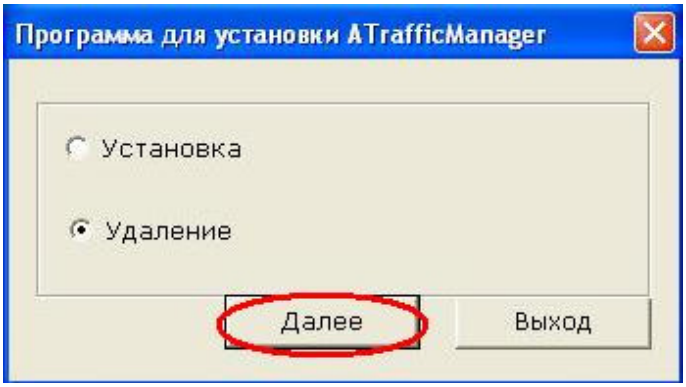

Рисунок А.3 – Удаление сервис провайдера

Просмотр статистики использования сетевого трафика

Для просмотра статистики использования сетевого трафика достаточно запустить исполняемый файл UserApp.exe. Для обновления информации нужно нажать кнопку «Обновить» (см. рис. А.4).

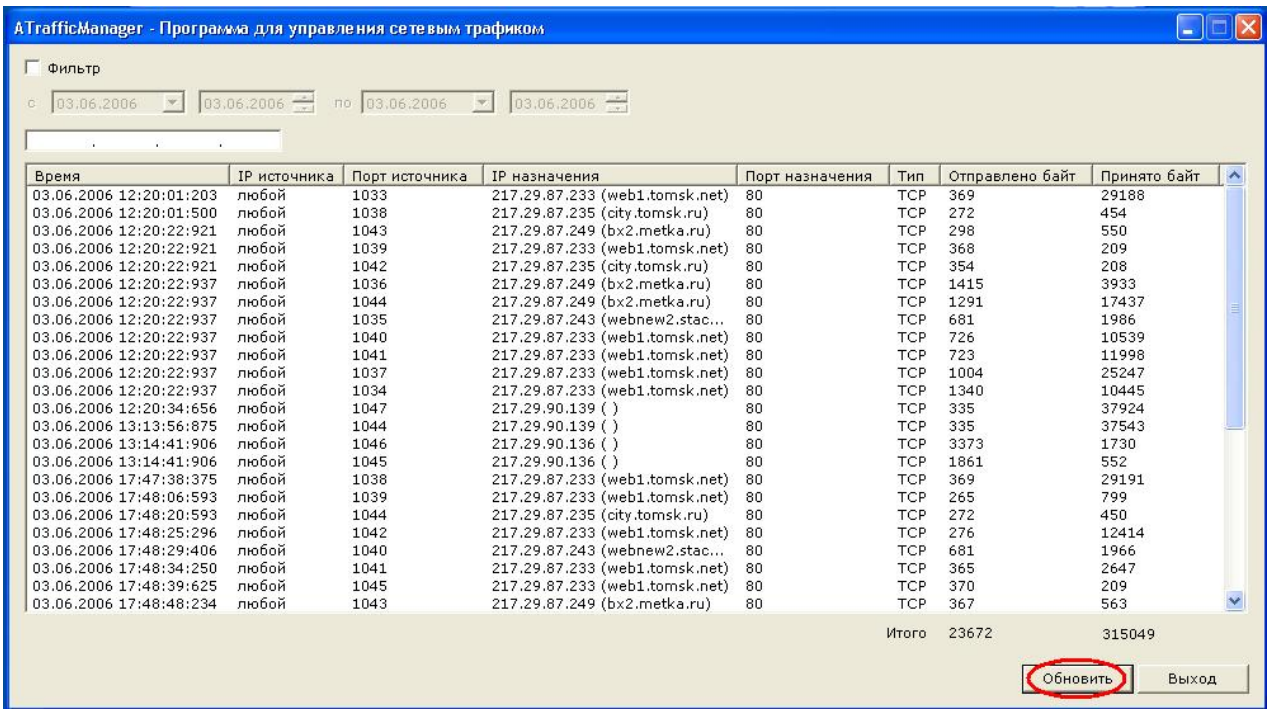

Рисунок А.4 – Просмотр статистики использования сетевого трафика

Чтобы отфильтровать собранные данные по времени или IP-адресу назначения, необходимо выбрать опцию «Фильтр», заполнить соответствующие поля и нажать кнопку «Обновить» (см. рис. А.5).

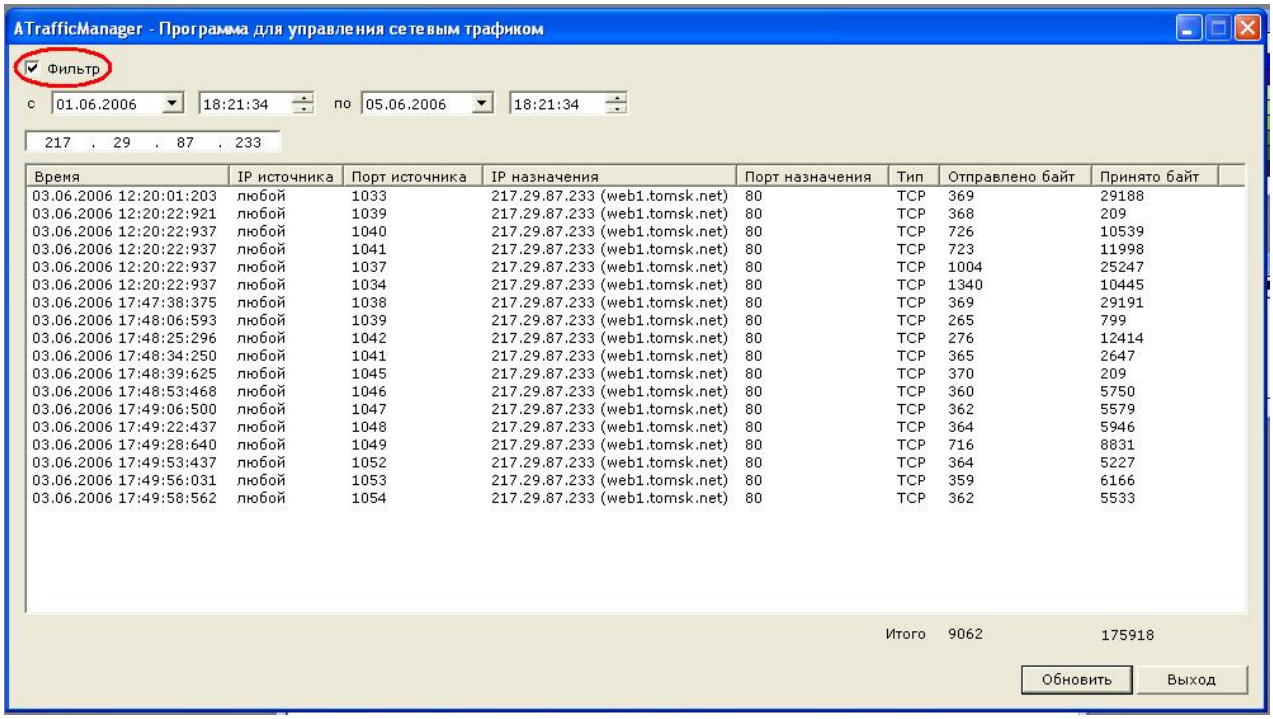

Рисунок А.5 – Фильтрация данных

Приложение Б. Руководство программиста

Приложение для учёта сетевого трафика реализовано в виде одного решения (solution'a) ATrafficManager, включающего в себя три проекта:

- Installer (Windows-приложение для установки сервис провайдера;
- ATrafficMngrDll (динамическая библиотека, реализующая основную функциональность сервис провайдера);
- UserApp (Windows-приложение, предназначенное для работы с сервис провайдером).

В качестве инструмента разработки использовалась среда MS Visual Studio .NET 2003, язык C++.

Приложение для установки и удаления сервис провайдера

Проект Installer содержит три заголовочных файла, три файла с кодом и один файл ресурсов (см. табл. Б.1) .

Таблица Б.1 – Файлы проекта Installer

Имя файла	Описание
ATrafficMngrCommon.cpp	Содержит функции, которые используются и динамической библиотекой, и установочным приложением.
ATrafficMngrCommon.h	Содержит прототипы функций из ATrafficMngrCommon.cpp, а также глобальный идентификатор (GUID) провайдера.
ATrafficMngrInstaller.cpp	Содержит функции для установки и удаления динамической библиотеки.
ATrafficMngrInstaller.h	Содержит прототипы функций, из ATrafficMngrInstaller.cpp, а также определения некоторых констант.
Installer.cpp	Содержит функцию wWinMain, а также функции диалоговых окон приложения.
Installer.h	Содержит прототипы функций из Installer.cpp.
Installer.rc	Содержит 2 диалоговых окна.

Содержание файлов ATrafficMngrCommon.cpp, ATrafficMngrInstaller.cpp, Installer.cp см. в таблицах Б.2, Б.3 и Б.4 соответственно.

Таблица Б.2 – Функции, реализованные в ATrafficMngrCommon.cpp

Название	Возвращаемое значение	Параметры	Описание
GetProvidersInfo()	LPWSAPROTOCOL_INFOW – указатель на первый установленный в системе провайдер	[out] LPINT lpProvidersCount – количество установленных в системе провайдеров	Возвращает информацию обо всех установленных в системе провайдерах. Для этого используется функция WSCEnumProtocols(), которая вызывается 2 раза: первый раз для того, чтобы определить, сколько необходимо выделить памяти, затем выделяется память с помощью функции VirtualAlloc(), после этого WSCEnumProtocols() вызывается второй раз, уже для получения нужной информации.

Таблица Б.2 (продолжение)

Название	Возвращаемое значение	Параметры	Описание
FreeProvi-dersInfo()	void	LPWSAPROTO-COL_INFOW all-Providers – указа-тель на первый провайдер	Освобождает память, выделенную функцией GetProvidersInfo(). Со-стоит из одного вызова Virtu-alFree().

Таблица Б.3 – Функции, реализованные в ATrafficMngrInstaller.cpp

Название	Возвращаемое значение	Параметры	Описание
Install()	int – при ус-пешном завер-шении установ-ки возвращает 0, иначе -1.	void	Устанавливает ATrafficManager (регистрирует динамическую биб-лиотеку в системе).
Uninstall()	int – при ус-пешном завер-шении деин-сталляции воз-вращает 0, ина-че -1.	void	Удаляет ATrafficManager из систе-мы.
RemoveId-FromChain()	int	DWORD dwId, LPWSAPROTO-COL_INFOW Protocol	Удаляем идентификатор dwId из цепочки протокола Protocol (Proto-col->ProtocolChain). Используется для удаления ATrafficManager, в случае, когда поверх него установ-лены другие сервис провайдеры.

Таблица Б.4 – Функции, реализованные в Installer.cpp

Название	Возвращаемое значение	Параметры	Описание
wWinMain()	int	HINSTANCE hInstance, HINSTANCE, PWSTR pszCmdLine, int nCmdShow	Главная функция приложения. Управляет выводом диалоговых окон.
Dialog-StartWndProc()	BOOL	HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam	Оконная процедура диалогового окна для выбора установки или удаления.
DialogList-WndProc()	BOOL	HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam	Оконная процедура диалогового окна для вывода списка установ-ленных сервис провайдеров.

Динамическая библиотека

Список файлов динамической библиотеки приведён в табл. Б.5.

Таблица Б.5 – Файлы проекта ATrafficMngrDll

Название файла	Описание
ATrafficMngrDll.def	Def-файл библиотеки. Содержит её название и имена экспортируемых функций.
ATrafficMngrDll.cpp	Основной файл библиотеки. Содержит DllMain() и экспортируемую функцию WSPStartup().
ATrafficMngrDll.h	Содержит прототип WSPStartup() и определение структуры A_PROVIDER, которая используется для хранения информации о провайдере.
ATrafficMngrCommon.cpp	Содержит функции, которые используются и динамической библиотекой, и установочным приложением.
ATrafficMngrCommon.h	Прототипы функций из ATrafficMngrCommon.cpp, а также глобальный идентификатор (GUID) провайдера.
SPI.cpp	Реализация 30 функций из WSPPROC_TABLE.
SPI.h	Прототипы функций из SPI.cpp
Context.cpp	Реализация функций, предназначенных для работы с контекстом сокета.
Context.h	Прототипы функций из Context.cpp, а также определение структуры для хранения контекста сокета.
AsyncSelect.cpp	Реализация функций, предназначенных для корректной обработки вызова WSAAsynSelect() (создание спрятанного окна и получение сообщений).
AsyncSelect.h	Прототипы из AsyncSelect.cpp, а также определение некоторых констант.
OverlappedIO.cpp	Реализация функций предназначенных, для обработки запросов на отложенный ввод/вывод
OverlappedIO.h	Прототипы функций из OverlappedIO.cpp, определение некоторых констант и структур, перечисление для идентификации запрошенной операции ввода/вывода.
Extensions.cpp	Реализация функций, предназначенных для корректной обработки вызова WSAIoctl().
Extensions.h	Определение структуры, содержащей таблицу функций-расширений, прототипы из Extensions.cpp.
Debug.cpp	Реализация функций для отладки приложения.
Debug.h	Прототипы функций из Debug.cpp.
DataManager.cpp	Реализация класса для работы с файлом данных (содержит методы для открытия, закрытия и записи в файл)
DataManager.h	Описание класса для работы с файлом данных, также содержит определение некоторых констант
Record.cpp	Реализация класса представляющего собой запись в файле данных
Record.h	Описание класса записи в файле данных

Содержимое файлов Context.cpp, AsyncSelect.cpp и OverlappedIO.cpp см. в таблицах Б.6, Б.7 и Б.8 соответственно.

Таблица Б.6 – Содержимое файла Context.cpp

Название	Возвращаемое значение	Параметры	Описание
CreateContext()	LPSOCKET_-CONTEXT	SOCKET nextSocket, LPA_PROVI-DER Provider	Выделяет память в куче под структуру, в которой содержится информация о сокете. А также помещает эту структуру в общий список контекстов, относящихся к провайдеру Provider.
FreeCon-textMemory()	void	LPA_PROVI-DER Provider	Освобождает выделенную с помощью CreateContext() память.
RemoveCon-text()	void	LPA_PROVI-DER Provider, LPSOCKET_-CONTEXT SocketContext	Удаляет указанную структуру SocketContext из списка контекстов провайдера Provider.
GetContext()	LPSOCKET_-CONTEXT	SOCKET s	Ищет среди всех провайдеров контекст, содержащий информацию о сокете s.
QueryCon-textAndIncRef-Count()	LPSOCKET_-CONTEXT	SOCKET s, [out] int *lpErrno	Запрашивает с помощью WPUQue-rySocketHandleContext контекст сокета s, а также увеличивает на единицу счётчик ссылок на структуру с информацией о сокете.
DecRefCount()	void	LPSOCKET_-CONTEXT lpContext	Уменьшает на единицу счётчик ссылок на контекст lpContext.
LockContext()	void	LPSOCKET_-CONTEXT lpContext	Выполняет вход в критическую секцию контекста.
ReleaseCon-text()	void	LPSOCKET_-CONTEXT lpContext	Выполняет выход из критической секции контекста.

Таблица Б.7 – Содержимое файла AsyncSelect.cpp

Название	Возвращаемое значение	Параметры	Описание
GetAsyncWin-dow()	HWND	void	Возвращает описатель спрятанного окна. Если оно ещё не создано, то создаёт его.
AsyncThread-Proc()	DWORD	LPVOID lpParameter	Процедура потока, который принимает сообщения и передаёт их оконной процедуре.
AsyncWnd-Proc()	LRESULT	HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam	Оконная процедура спрятанного окна.
StopAsync-Thread()	void	void	Уничтожает спрятанное окно и закрывает его описатель.

Таблица Б.8 – Содержимое файла OverlappedIO.cpp

Название	Возвращаемое значение	Параметры	Описание
AllocateFree-Pool()	int	void	Выделяет в куче память для хранения списка свободных структур для передачи параметров операции отложенного ввода / вывода.
InitOver-lappedIO()	int	void	Подготавливает сервис провайдер к обработке операций отложенного ввода вывода: создаёт порты завершения ввода / вывода и пул потоков для работы с ними.
StartOver-lappedIO()	void	void	Вызывает InitOverlappedIO().
StopOver-lappedIO()	int	void	Посылает пулу потоков специальный пакет, сообщающий о необходимости завершить работу. А также закрывает описатели потоков и портов завершения ввода / вывода.
Over-lappedThread-Proc()	DWORD	LPVOID lpParam	Процедура потоков, обрабатывающего пакеты, которые поступают от портов завершения ввода / вывода.
GetOverlapped-Struct()	LPEXTENDED_ WSAOVER-LAPPED	LPSOCKET_CO NTEXT lpContext	Возвращает указатель на следующую свободную структуру из списка свободных.
FreeOver-lappedStruct()	void	LPEXTENDED_ WSAOVER-LAPPED lpExt-Overlapped	Помещает указанную структуру в список свободных.
Intermedi-ateCR()	void	DWORD dwEr-ror, DWORD cbTransferred, LPWSAOVERL APPED lpOver-lapped, DWORD dwFlags	Промежуточная процедура завершения операции отложенного ввода / вывода. Необходима для подсчёта объёма принятых и переданных данных в результате операции отложенного ввода /вывода.
QueueOver-lappedOp()	int	LPEXTENDED_ WSAOVERLAP PED lpExtOver-lapped, LPSOCKET_-CONTEXT lpContext	Связывает структуру lpContext с портом завершения ввода / вывода и запускает операцию ввода / вывода на выполнение.
ExecuteOver-lappedOp()	int	LPEXTENDED_ WSAOVER-LAPPED op	Вызывает WSPIoctl() нижележащего провайдера, при этом в качестве процедуры завершения ввода / вывода передаёт IntermediateCR() – для последующей обработки результатов.

Таблица Б.8 (продолжение)

Название	Возвращаемое значение	Параметры	Описание
CallUser-ApcProc()	void	ULONG_PTR Context	Вызывает пользовательскую процедуру завершения ввода / вывода. После обработки результатов внутри сервис провайдера - пользовательскому приложению нужно сообщить о завершении операции.
Context-Cleanup()	void	LPEXTENDED_WSAOVERLAPPED lpExtOverlapped	Уменьшает счётчик ссылок на контекст, связанный с завершенной операцией отложенного ввода / вывода. В случае если счётчик становится равен нулю, освобождает память.

Файл Extensions.cpp содержит функции, предназначенные для обработки вызовов расширенных функций (через WSPIoctl()).

Файл Debug.cpp содержит только одну функцию – PrintDebugInfo() с помощью которой можно сделать форматированный вывод данных в окно для вывода отладочных данных.

Основные методы классов DataManager и Record перечислены в таблицах Б.9 и Б.10.

Таблица Б.9 – Основные методы класса DataManager

Название	Возвращаемое значение	Параметры	Описание
InitWriting()	int – возвращает 0 в случае успешного открытия файла и -1 в случае ошибки	void	Открывает файл данных на запись. При этом если файла не существует, то создаёт новый и записывает в него заголовок.
WriteData()	int – возрващает 0 в случае успешной записи и -1 в случае ошибки	LPSOCKET_-CONTEXT lpContext	Записывает информацию из структуры lpContext в файл данных.

Таблица Б.10 – Основные методы класса Record

Название	Возвращаемое значение	Параметры	Описание
GetCurTime()	int – возвращает 0 в случае успешного завершения и -1 в случае ошибки	[out] TCHAR* buffer, INT size	Записывает в buffer размера size текущее время с точностью до миллисекунд. Для этого используется функция GetLocalTime().
AddZeroes()	void	[in, out] TCHAR* str, INT len	Дополняет строку str таким количеством нулей в начале, чтобы длина строки была равна len. Данный метод используется для приведения времени к нужному формату.

Таблица Б.10 (продолжение)

Название	Возвращаемое значение	Параметры	Описание
Save()	int – возвращает 0 в случае успешного завершения и -1 в случае ошибки	[out] TCHAR *buffer	Представляет все члены класса в виде единой строки buffer.

Пользовательское приложение для работы с сервис провайдером

Список файлов проекта UserApp см. в табл. Б.11.

Таблица Б.11 – Файлы проекта UserApp

Название файла	Описание
UserApp.cpp	Основной файл приложения.
UserApp.h	Прототипы функций из UserApp.cpp, а также определения некоторых констант.
UserApp.rc	Диалоговое окно приложения.
DataReader.cpp	Реализация класса для чтения из файла данных
DataReader.h	Описание класса для чтения из файла, также содержит определения некоторых констант
UserRecord.cpp	Реализация класса для представления записи из файла данных в виде структуры, удобной для дальнейшей обработки
UserRecord.h	Описание класса для представления записи
FSM.cpp	Реализация класса конечного автомата
FSM.h	Описание класса конечного автомата

Содержимое файла UserApp.cpp см. в табл. Б.12.

Таблица Б.12 – Содержимое файла UserApp.cpp

Название	Возвращаемое значение	Параметры	Описание
wWinMain()	int	HINSTANCE hInstance, HINSTANCE, PWSTR pszCmdLine, int nCmdShow	Главная функция приложения. Создаёт и выводит на экран диалоговое окно.
AppWndProc()	BOOL	HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam	Оконная процедура диалогового окна.
Refresh()	void	WND hDlg, BOOL bFilter	Функция для обновления содержимого окна.

Таблица Б.12 (продолжение)

Название	Возвращаемое значение	Параметры	Описание
ShowRecords()	void	HWND hDlg, UserRecord *RecordSet, DWORD dwCount	Отображает массив записей RecordSet в диалоговом окне
CountFilteredRecords()	DWORD	UserRecord* RecordSet, DWORD size, SYSTEMTIME from, SYSTEMTIME to, DWORD ip, BOOL bIp	Возвращает число записей из массива RecordSet, соответствующих заданному фильтру
GetFilteredRecords()	void	UserRecord* RecordSet, DWORD dwSetSize, [out] UserRecord* FilteredSet, DWORD dwFiltered, SYSTEMTIME from, SYSTEMTIME to, DWORD ip, BOOL bIp	Выбирает записи из массива RecordSet, соответствующие заданному фильтру и записывает их в массив FilteredSet
CompareTime()	long	LPSYSTEMTIME a1, LPSYSTEMTIME a2	Сравнивает два времени в формате SYSTEMTIME. Для этого оба параметра переводятся в формат FILETIME и используются функция CompareFileTime().

Основные методы классов DataReader и FSM перечислены в таблицах Б.13 и Б.14 соответственно.

Таблица Б.13 – Основные методы класса DataReader

Название	Возвращаемое значение	Параметры	Описание
InitReading()	int – в случае успешного завершения возвращает 0, в случае ошибки --1	void	Открывает файл данных на чтение.
CountRecords()	DWORD	void	Считает текущее количество записей в файле данных

Таблица Б.13 (продолжение)

Название	Возвращаемое значение	Параметры	Описание
ReadData()	int – в случае успешно завершения возвращает 0, в случае ошибки - -1, в случае конца файла – 1.	[out] TCHAR* buffer, DWORD dwSize, [out] DWORD *lpRead	Считывает из файла данных dwSize байт и записывает их в buffer. В переменную lpRead записывает количество прочитанных байт.

Таблица Б.14 – Основные методы класса FSM

Название	Возвращаемое значение	Параметры	Описание
ProcessData()	void	[out] UserRecord* RecordSet, DWORD dwCount	Преобразует файл данных в массив объектов UserRecord. Для этого используется конечный автомат. Для считывания файла используется метод ReadData() класса DataReader. Результат записывается в буфер RecordSet размера dwCount.
GetRecordSetSize()	DWORD	void	Считает необходимый размер буфера объектов UserRecord, для того чтобы в нём можно было хранить все записи файла данных. Использует метод CountRecords() класса DataReader.
StartField()	void	void	Семантическая процедура. Используется для начала формирования поля.
ContinueField()	void	void	Семантическая процедура. Используется для накопления значения поля.
FinishTime()	void	void	Семантическая процедура. Завершает формирование поля представляющего собой время.
FinishSourceIp()	void	void	Семантическая процедура. Завершает формирование IP-адреса источника.
FinishSourcePort()	void	void	Семантическая процедура. Завершает формирование порта источника.
FinishDestIp()	void	void	Семантическая процедура. Завершает формирование IP-адреса назначения.
FinishDestPort()	void	void	Семантическая процедура. Завершает формирование порта назначения.
FinishType()	void	void	Семантическая процедура. Завершает формирование типа

Таблица Б.14 (продолжение)

Название	Возвращаемое значение	Параметры	Описание
FinishBytes-Sent()	void	void	Семантическая процедура. Завершает формирование количества переданных байт
FinishBytes-Recvd()	void	void	Семантическая процедура. Завершает формирование количества принятых байт
ReadBuffer()	void	void	Семантическая процедура. Считывает очередную порцию из файла данных.
FinishRecord()	void	void	Семантическая процедура. Завершает формирование записи.
Empty()	void	void	Пустая семантическая процедура

В таблице Б.15 приведено описание основных членов класса FSM.

Таблица Б.15 – Основные члены класса FSM

Имя	Тип	Описание
TransitionMatrix	int**	Матрица переходов
dr	DataReader	Объект для работы с файлом данных
ur	UserRecord*	Массив преобразованных записей
curRecord	UserRecord	Текущая формируемая запись
m_iRecordSetSize	int	Размер массива записей
m_iRecordCount	int	Номер текущей формируемой записи
m_lpszField	TCHAR[BUF_SIZE]	Текущее формируемое поле
m_iFieldCount	int	Номер текущего символа в формируемом поле
m_lpszBuffer	TCHAR[BUF_SIZE]	Текущая преобразуемая порция файла данных
m_iBufferCount	int	Номер текущего обрабатываемого символа из буфера m_lpszBuffer

Класс UserRecord используется для создания объектов типа запись, полученных из CSV-файла. Он содержит методы для установки и получения значений его членов, а также методы, приведённые в таблице Б.16.

Таблица Б.16 – Основные методы класса UserRecord

Название	Возвращаемое значение	Параметры	Описание
GetDestHost-Name()	void	[out] TCHAR *hostName	Возвращает имя хоста назначения, полученного по его IP-адресу с помощью функции getsockname();
CompareIp()	BOOL - возвращает TRUE в случае совпадения, иначе FALSE.	DWORD dwIp	Сравнивает IP-адрес назначения текущего объекта, с переданным ему в качестве параметра.