УДК 004.732

Н.И. Алишов

ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА МОДЕЛИРОВАНИЯ МЕТОДОВ ДОСТУПА В ЛОКАЛЬНЫХ СЕТЯХ

Рассматриваются архитектура, алгоритмы работы и характеристики функционирования программного комплекса для исследования методов доступа, реализуемых в локальных сетях компьютеров. Его отличительной особенностью является универсальная полнота средств моделирования. Разработанный инструментальный комплекс может служить также для статистического анализа сетевых трафиков, моделирования алгоритмов доступа в канал и проектирования сетевых структур с общим разделяемым каналом.

Введение

В Институте кибернетики им. В.М. Глушкова НАН Украины разработана технология универсального множественного доступа в распределенных системах для повышения эффективной производительности на канальном уровне, а также для динамической оптимизации взаимодействия узлов локальной сети. Основу данной технологии составляет адаптивный стековый алгоритм *ANIO*, который является надстройкой над классическим методом доступа *CSMA/CD*. Благодаря данному не требуется разработка новой элементной базы, что позволяет реализовать предложенный метод в существующих локальных сетях. Его реализация предполагает наличие в каждом узле сети унифицированного стекового механизма, который используется для улучшения характеристик дисциплины обслуживания стохастических запросов. Универсальная полнота предложенных алгоритмов позволяет реализовать практически любой протокол взаимодействия узлов локальной сети на канальном уровне [1−3].

Постановка задачи

Отсутствие проблемно-ориентированной системы для исследования методов доступа канального уровня, их сравнения и оценки обусловливает разработку специализированной системы для оценки производительности протоколов канального уровня. Такая система позволит классифицировать методы доступа по их эффективной производительности, облегчить их усовершенствование, а также реализовать новые, более совершенные методы доступа на канальном уровне. И прежде всего необходимо разработать методику проведения экспериментов, определить критерии оценки эффективности протокола, разработать программную реализацию протокола (сетевую службу, драйвер), подобрать или разработать средства сбора и анализа результатов, спроектировать общую структуру операционной среды для проведения экспериментов.

Состояние проблемы

Для исследования сетевого трафика обычно используются сетевые анализаторы (*Network Sniffers*) типа *Tcpdump, WinDump, Ethereal* и т.п. [4]. Как правило, анализаторы являются частью более широкой категории аппаратного и программного обеспечения, предназначенного для мониторинга сети и позволяющего администраторам контролировать свои локальные и корпоративные сетевые службы и сетевой трафик. Существуют также инструментальные средства проектирования компьютерных сетей, позволяющие оценивать проектные решения, моделировать различные режимы функционирования сетей и т.д.. Наиболее известные среди них − *ComNet, NetMarker XA, OpNet* и др. [5]. Однако перечисленные системы не позволяют решать поставленные в данной работе задачи должным образом, поэтому необходимо разработать такие инструментальные средства, которые, с одной стороны, обладают функциями сетевых анализаторов, а с другой − позволяют моделировать и проектировать локальные сети с новыми протоколами организации взаимодействия сетевых ресурсов.

Методы и алгоритмы решения задачи

Алгоритм *ANIO* является концепцией множественного доступа в канал передачи данных, основанной на исключении арбитражных и случайных методов регулирования после установления логико-виртуального соединения. Чтобы исследовать эффективность функционирования данного алгоритма в операционной среде, была разработана его программная модель для проведения экспериментов. Программно стековый механизм реализован как отдельный класс. Разработано программное обеспечение, позволяющее внедрить данного механизма в операционную среду, организовать сохранение статистической информации в *LOG*-файлах, подобрать средства оценки эффективной производительности локальной сети и промоделировать различные методы доступа в канал [6, 7].

Программная реализация стека *ANIO* предполагает получение в реальном масштабе времени *MAC*-адреса узла, осуществившего передачу. В результате работы стек должен выдавать информацию о том, что наступила очередь передачи пакета; это необходимо для реализации доступа к каналу по данному алгоритму. Для нормальной работы стек должен поддерживать три типа операций (табл. 1): добавление нового элемента в стек, вращение стека в процессе формирования очереди и удаление элемента из стека.

Алгоритм должен поддерживать следующие функции работы со стеком: инициализация стека, добавление нового элемента, сравнение элементов, поиск элемента, удаление элемента, сдвиг очереди, активный и пассивный режим работы, обработку наступления тайм-аута. Псевдопрограммная модель упрощенного принципа работы стекового механизма показана на рис. 1.

В табл. 2 приведены схемы формирования и соответствующие алгоритмы стековых операций.

Описание общей структуры алгоритма ANIO. Работа алгоритма начинается в момент получения *MAC*-адреса или сигнала тайм-аута (рис. 2). Если тайм-аут наступил в активном режиме, то алгоритм переходит в пассивный режим. При наступлении тайм-аута в пассивном режиме удаляется второй элемент стека (*MAC*-адрес узла, пропустившего передачу).

Как только получен *MAC*-адрес, он сразу добавляется в стек в качестве нового элемента, счетчик (*count*) увеличивается на единицу. Затем выполняется поиск дубликатов, новый элемент стека сравнивается с каждым элементом стека. В результате будет получен указатель (*pAnioP*) на дубликат и номер (*Position*) элемента стека.

Position = 0 – дубликаты не обнаружены, новый *MAC*-адрес;

Таблица 1. Основные операции управления стеком

Операция		Причина		Действия		Режим
				Активный	Пассивный	
Добавление		Новый узел начал передачу		Добавить новый элемент в конец очереди	Добавить новый элемент в конец очереди	Начальный режим
Сдвиг		Очередной узел совершил передачу		Добавить новый элемент в конец очереди	Добавить новый элемент в конец очереди	Стационарный режим
			MAC	Удалить 1-й элемент	Удалить 2-й элемент	
			MyMAC	Переход из активного режима в пассивный	Переход из пассивного режима в активный	
Удаление	Ошибка	Узел стоит в очереди, но начал передачу раньше времени		Добавить новый элемент в конец очереди Удалить N-й элемент	Добавить новый элемент в конец очереди Удалить N-й элемент	Сбой
	Тайм-аут	Очередной узел пропустил передачу	MAC	Удалить 1-й элемент	Удалить 2-й элемент	Рабочий режим
			MyMAC	Переход из активного режима в пассивный	Переход из пассивного режима в активный	

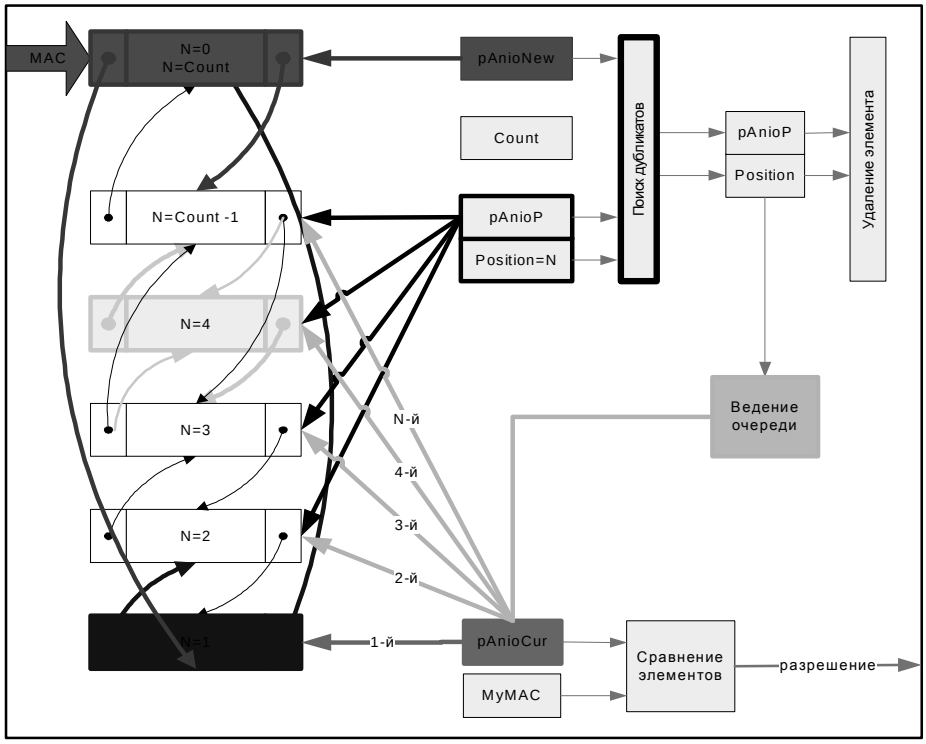


Рис. 1. Упрощенный принцип работы стекового алгоритма

Position = 1 – новый элемент *pAnioNew* совпадает с текущим *pAnioCur;* очередной узел передал пакет в сеть, необходимо удалить дубликат элемента из позиции 1.

Если до этого был пассивный режим и *pAnioNew* совпадает с локальным адресом *MyMAC*, значит локальный узел возобновил обмен данными и перешел в активный режим.

Position = 2 – новый элемент *pAnioNew* совпадает с *pAnioCur – >next;* очередной узел передал пакет в сеть, необходимо удалить дубликат элемента из позиции 2.

Position = n – произошел сбой, узел поспешил с передачей пакета, необходимо удалить дубликат элемента из позиции *n*.

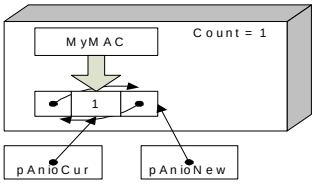
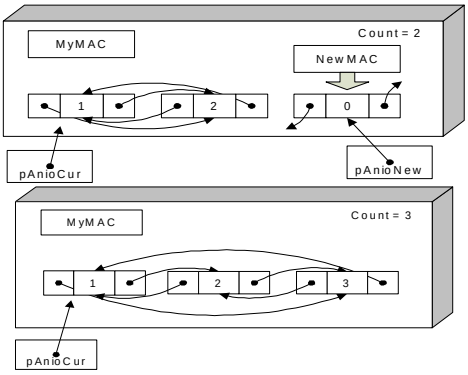
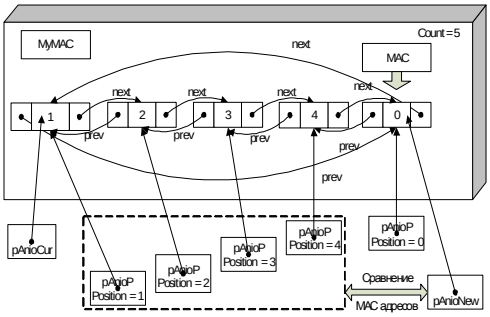
После завершения операций со стеком нужно обеспечить движение очереди, т.е. перейти к следующему элементу стека. Чтобы определить наступление очереди передачи для локального узла, необходимо сравнить *pAnioCur* и *MyMAC*; и, если они совпадают, выдать сигнал о разрешении передачи пакета в сеть.

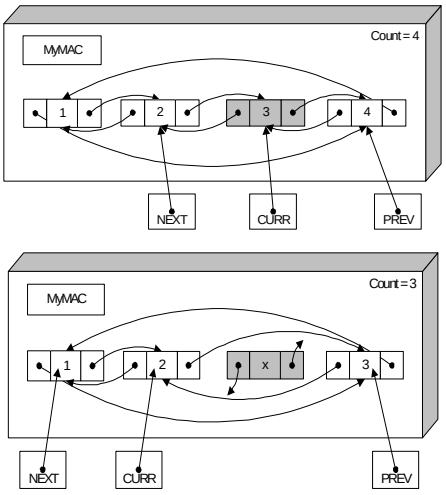
Функция анализа адресов в качестве параметра получает *MAC*-адрес узла, который в данный момент отправил в сеть пакет. Если вместо *MAC*-адреса было передано значение *NULL*, это значит, что произошел тайм-аут. В результате своего выполнения функция возвращает значение логического типа *BOOLEAN*: *TRUE* – наступила очередь передачи пакета в сеть, *FALSE* – очередь не наступила.

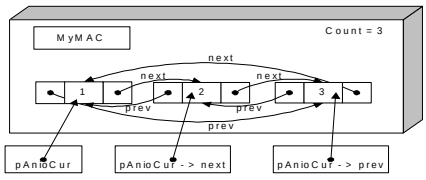
Когда алгоритм находится в пассивном режиме, функция анализа состояния локальной сети все время возвращает значение *TRUE*. При этом ведётся стек, но без участия локального узла: локальный узел имеет право в любой момент возобновить передачу.

К основным функциям алгоритма *ANIO* прежде всего следует отнести: ведение списка очереди по алгоритму, пассивный мониторинг сетевых пакетов (на сетевом либо на более высоком уровне), пассивный мониторинг доступа к сетевым системным библиотекам (*DLL*) и сетевым сервисам, активный мониторинг сетевых пакетов, т. е. мониторинг и задержка исходящих пакетов для ожидания очереди (на более высоком уровне − мониторинг и фильтрация запросов к сетевым системным *DLL* и сетевым сервисам) [8]. Комбинируя перечисленные функции, можно по

Таблица 2. Алгоритмы реализации стековых операций

	Стек	Алгоритм
Инициализация		MyMAC ? локальный MAC-адрес pAnioCur = pAnioNew; pAnioCur– >next = pAnioCur; pAnioCur– >prev = pAnioCur; pAnioNew– >MAC = MyMAC; Count = 1;
Добавление		Count – количество элементов в стеке MAC – MAC-адрес узла, передавшего в данный момент пакет в сеть pAnioNew– > prev = pAnioCur– > prev; pAnioCur– > prev– >next = pAnioNew; pAnioCur– > prev = pAnioNew; pAnioNew– > next = pAnioCur; pAnioNew– > MAC = MAC; Count++;
Поиск		Position – номер позиции в стеке pAnioP – позиция в стеке while (!pAnioP == pAnioNew) { Position--; pAnioP = pAnioP – > prev;} Position = 1 – pAnioCur ? очередной элемент в активном режиме Position = 2 – pAnioCur –> next ? очередной элемент в пассивном режиме Position = (3 .. Count-1) ? MAC-адрес одного из узлов, ожидающего своей очереди передачи Position = (0 и Count) – pAnioNEW ? новый элемент стека

Удаление		_AnioStack * PREV = pAnioS– > prev;
		_AnioStack * CURR = pAnioS;
		_AnioStack * NEXT = pAnioS– > next;
		NEXT– >prev = PREV;
		PREV– >next = NEXT;
		ExFreePool(CURR);
		Count--;
		pAnioS = NEXT;

Сдвиг		pAnioCur ? текущий элемент pAnioCur – > next ? следующий элемент pAnioCur – > prev ? предыдущий элемент pAnioCur = pAnioCur – > next

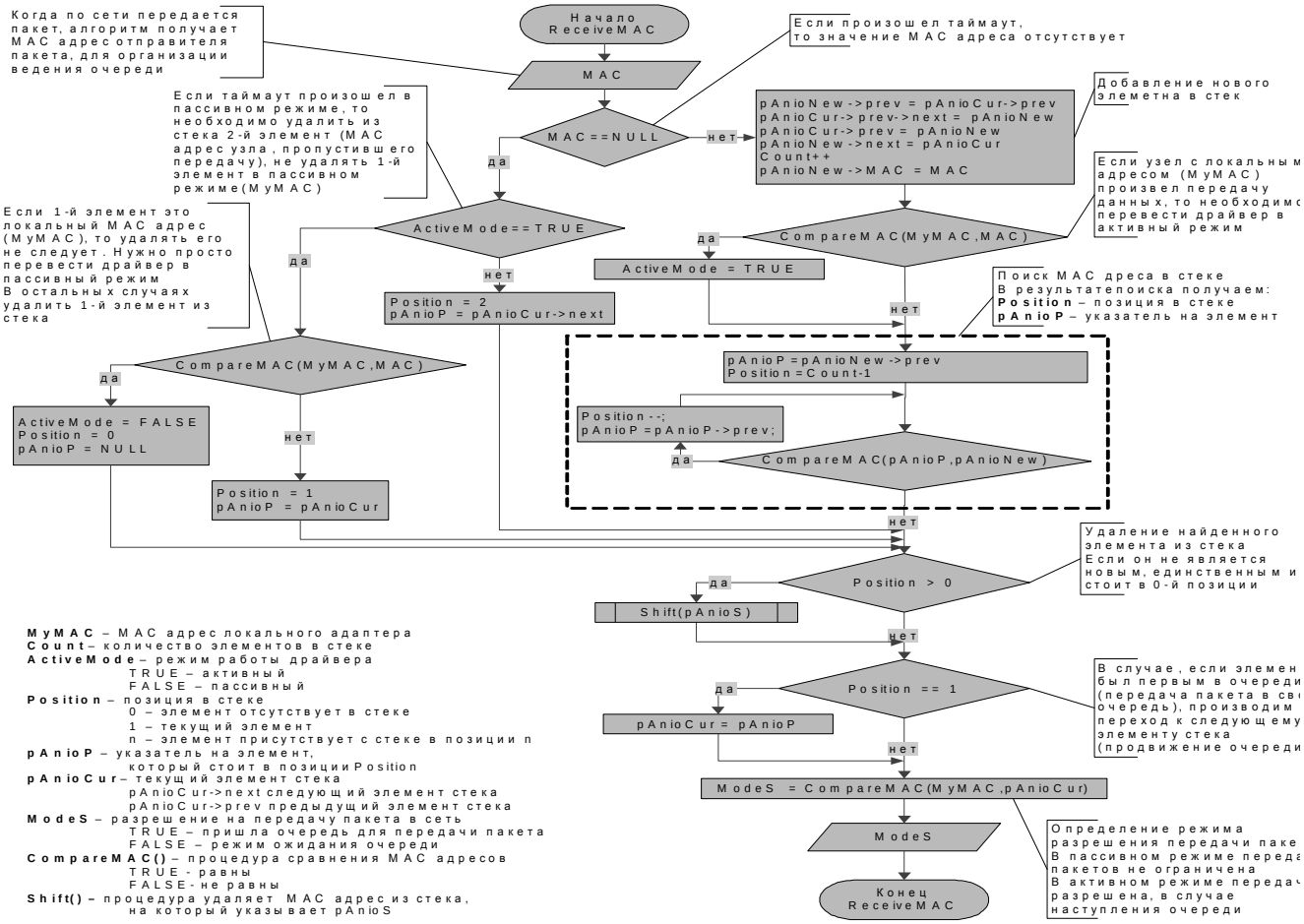


Рис. 2. Блок-схема алгоритма *ANIO*

лучить любой класс системы сетевого обмена − от простого сетевого монитора для отслеживания сетевой активности приложения до брандмауэра с возможностями поддержки виртуальных соединений.

Разработка приложений или *DLL* с функциями алгоритма − это самая тривиальная и обеспечивающая наименьшую гибкость его реализации. Алгоритм на уровне приложения не является прозрачным, в данном случае пользователь должен сам решать, нужно ли его использовать.

Для того чтобы данные других приложений могли обрабатываться приложением, реализующим алгоритм передачи этих данных, должны использоваться специальные механизмы межпроцессной коммуникации: копирование данных из одного адресного пространства в другое или создание области совместно используемой памяти, видимой из обоих адресных пространств [9]. Следовательно, приложение, данные которого необходимо передавать по алгоритму *ANIO*, должно быть разработано для взаимодействия с приложением, обрабатывающим его данные, с использованием данных механизмов. Но оно не обязано знать о существовании данного алгоритма.

Реализация алгоритма на уровне собственной *DLL* также не обеспечивает прозрачной передачи данных. Для того чтобы передачу данных приложений выполняла некоторая библиотека *DLL*, необходимо, чтобы приложения были разработаны так, чтобы вызывать функции из данной *DLL*. Затем, как исполняемый код *DLL* проецируется на адресное пространство вызывающего процесса (с помощью неявной компоновки при загрузке приложения, или явной – в период выполнения), код и данные *DLL* просто становятся частью процесса приложения. Таким образом, уровень приложений и собственных *DLL*

не имеет всех возможностей по реализации функций алгоритма. Интерфейсы *API*, предоставляемые операционной системой (ОС) приложениям и *DLL*, не позволяют им выполнять пассивный или активный мониторинг сетевых пакетов или контролировать доступ к сетевым системным *DLL* и сетевым сервисам. Только лишь во взаимодействии с собственным драйвером приложение может реализовать алгоритм, причем в некоторых случаях приходится использовать недокументированные методы.

Возможности реализации алгоритма на уровне ядра ОС

Сетевые компоненты, исполняющиеся в привилегированном режиме − режиме ядра, являются драйверами. Прежде чем перейти к дальнейшему рассмотрению сетевой архитектуры *Windows NT*, позволяющая как и куда встраивать модули алгоритма на уровне ядра, необходимо вспомнить механизм драйверов-фильтров, который может использоваться в качестве средства реализации алгоритма на любом уровне в стеке драйверов.

Один из методов расширения возможностей системы ввода/вывода − разработка и применение драйверов-фильтров (рис. 3). Перехватив запрос, драйвер-фильтр либо расширяет, либо замещает функциональность, обеспечиваемую первоначальным получателем запроса. При этом драйвер-фильтр использует либо сервисы драйвера, которому изначально предназначался запрос, либо сервисы других программных модулей уровня ядра для обеспечения дополнительной функциональности. Драйверы-фильтры могут встраиваться в любое место в стеке драйверов, их может быть несколько, в том числе расположенных подряд. Драйвер-фильтр обычно исследует, модифицирует, завершает или передает дальше полученный пакет. На всех уровнях, которые будут обсуждаться в дальнейшем, возможны реализация и применение драйверов-фильтров.

Реализация алгоритма на уровне транспортного драйвера

Транспортные драйверы являются, фактически, стандартными драйверами промежуточного уровня и реализуют в своей верхней части стандартный интерфейс, соответ-ствующий *TDI*-спецификации. Этот интерфейс в основном базируется на получении и обработке пакетов *IRP_MJ_INTERNAL_DEVICE_CONTROL*, содержащих различные значения контрольных кодов *TDI_XXX*, которые определяются *TDI*-спецификацией. В нижней части *TDI* драйвер взаимодействует с *NDIS*-библиотекой. Разработка драйверов транспорта хорошо документирована в *Windows NT DDK* (глава Network drivers, раздел Intermediate NDIS drivers and TDI drivers).

Возможна реализация драйвера-фильтра, который будет присоединяться к объектам-устройствам, создаваемым драйвером транспорта, например, к объектам-устройствам *\Device\Tcp* и *\Device\Udp*, создаваемым драйвером транспорта *TCP/IP*. Учитывая, что драйвер транспорта должен предоставлять единый интерфейс, облегчается задача его разработки. Про-

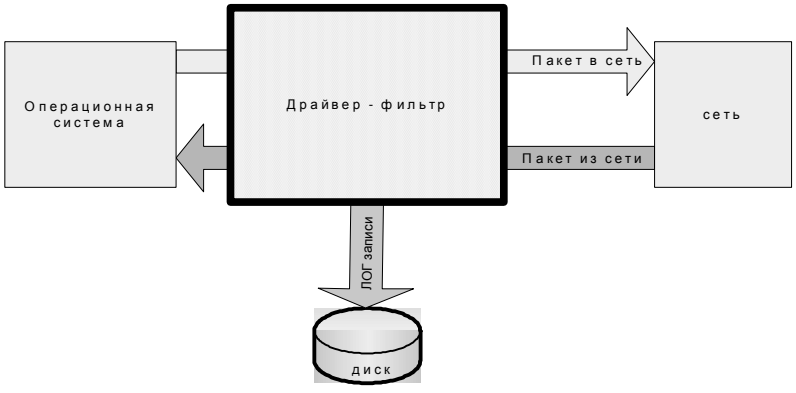


Рис. 3. Входящие и исходящие данные драйвера-фильтра

цесс происходит с помощью вызова драйвером транспорта функций, указатели на которые передаются *TDI*-клиентом драйверу транспорта в самом начале сетевых операций в пакете *IRP_MJ_INTERNAL_DEVICE_ CONTROL* с контрольным кодом *TDI_SET_ EVENTJHANDLER*, и регистрируются драйвером транспорта. *TDI*-клиент вполне может использовать подобный механизм функций обратного вызова в качестве альтернативы обычным пакетам запросов к транспорту с контрольными кодами *TDI_XXX*. Но когда драйвер транспорта вызывает подобную функцию, он может передать *TDI*-клиенту в качестве параметров лишь ограниченное количество данных. И тогда драйвер-фильтр, присоединенный к драйверу транспорта, не получит возможности проконтролировать эти данные.

Подобное свойство расширения возможностей взаимодействия клиента с определенным драйвером транспорта приводит к частичной закрытости интерфейса между ними. В результате чего затрудняется разработка драйвера-фильтра, присоединяемого к транспортному драйверу. При этом сохраняется возможность разработки собственного драйвера транспорта *DDK*, который реализует требуемые функции алгоритма.

Реализация алгоритма на уровне сетевого драйвера промежуточного уровня, поддерживающего интерфейс NDIS

Разработка *NDIS*-драйверов промежуточного уровня − один из хорошо документированных механизмов расширения возможностей системы ввода/вывода ОС *Windows NT*. Типичные варианты реализации промежуточного драйвера показаны на рис. 4. Промежуточный драйвер выглядит для драйвера транспорта как драйвер виртуальной сетевой карты. Используя механизм привязок, можно все транспорты привязать снизу к требуемому промежуточному драйверу, который таким образом получит возможность контролировать все пакеты, идущие в/из сети. Это позволит ему реализовать такие функции защиты, как преобразование форматов пакетов, преобразование пользовательских данных в пакетах, фильтрация пакетов, регистрация пакетов, контроль содержимого пакетов.

Благодаря таким правилам взаимодействия посредством использования библиотеки *NDIS*, встраивание промежуточного драйвера между драйвером транспорта, поддерживающим интерфейс *NDIS* в нижней части, и *NDIS*-драйвером сетевой карты не приводит к нарушению работы стека сетевых драйверов.

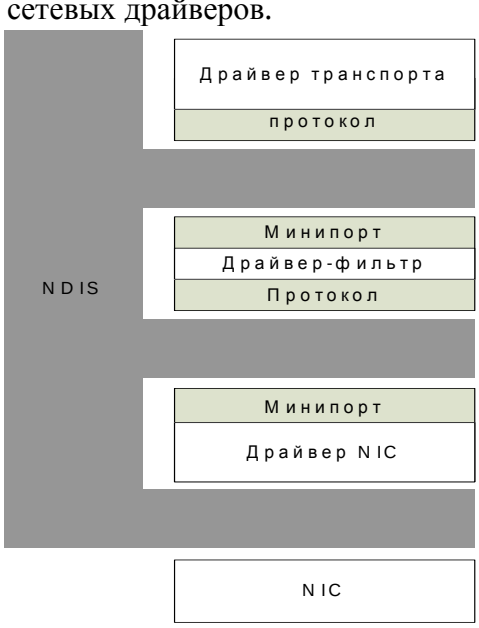


Рис. 4. Типы промежуточных драйверов

Промежуточный драйвер, расположенный над драйвером сетевой карты, фактически эквивалентен драйвер-фильтру, присоединенному к драйверу физического устройства. Только последний встраивается в цепь драйверов, взаимодействующих посредством пакетов *IRP*. Чтобы контролировать трафик на уровне драйверов, не обязательно заменять их собственными аналогами со встроенными функциями алгоритма *ANIO* можно создать промежуточный драйвер, реализующий функции алгоритма *ANIO* и расположить его выше или ниже этих драйверов.

Промежуточные драйверы являются неотъемлемой частью многих систем защиты сетевых ресурсов, реализующих пассивный мониторинг сетевых пакетов. Примером такой системы безопасности может служить программный анализатор сетевого трафика *Microsoft Network*

Monitor. Подобные драйверы являются частью многих систем безопасности, выполняющих фильтрацию и шифрование сетевого трафика, например *Guardian Windows NT Firewall.*

Реализация алгоритма на уровне драйверов сетевых устройств

Реализация алгоритма на уровне драйверов сетевых карт (рис. 5), зависит от разработанного драйвера устройства в соответствии со стандартом *NDIS*, т. е. это *NDIS*-драйвер сетевой карты локальной или глобальной сети, или это драйвер устройства, управляемый пакетами *IRP*. В первом случае лучше всего перейти к реализации промежуточного драйвера, а во втором − к реализации драйвер-фильтра. На этом уровне возможна также реализация собственного драйвера устройства со встроенными функциями алгоритма. Разработка собственных драйверов сетевых карт для глобальных и локальных сетей, поддерживаемых интерфейсом *NDIS*, хорошо документирована в *DDK*.

Сравнительный анализ способов реализации алгоритма ANIO

Анализ реализаций алгоритма выполним только для способов, имеющих возможность встраивания модуля алгоритма и расширения своей функциональности на данном уровне. Относительно следующих факторов: объем контролируемых данных, сложность реализации, прозрачность алгоритма, возможности, предоставляемые ОС коду алгоритма (табл. 3).

При оценке объема контролируемых данных необходимо определить, какие данные проходят через алгоритм и, следовательно, могут им контролироваться и обрабатываться [10].

Сложность реализации отражает степень ее документированности и сложность отладки. Данный фактор оценивается по трехбалльной шкале: низкая, средняя, высокая. Низкая сложность означает, что при разработке средства защиты используются хорошо документированные интерфейсы и доступные средства отладки; средняя, − что используются доку-

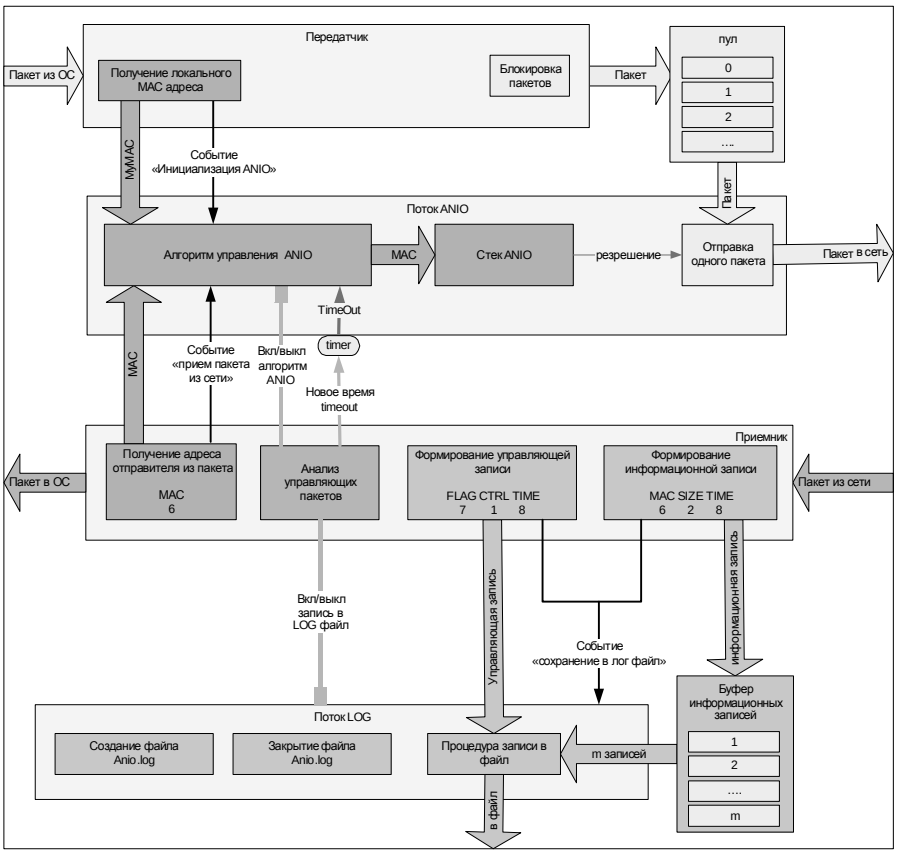


Рис. 5. Принцип работы драйвера

Таблица 3. Сравнительный анализ возможностей реализации алгоритма

Реализация алгоритма	Объем контролируемых данных	Сложность реализации	Прозрачность	Возможности, предоставляемые ОС	Уровень OSI
Приложение	Только данные самого приложения	Низкая	-	Минимальные	7
Собственная DLL	Данные приложений, использующих эту DLL	Низкая	-	Минимальные	6
Транспортный драйвер	Данные приложений, использующих этот транспорт, + данные приложений, взаимодействующих с другим транспортом напрямую	Высокая	+	Максимальные	3
Промежуточный драйвер	Данные приложений, использующих транспорты, привязанные снизу к этому промежуточному драйверу, + данные приложений, взаимодейст-вующих с промежуточным драйвером напрямую	Средняя	+	Максимальные	2
Драйвер сетевого устройства	Данные всех приложений	Средняя	+	Максимальные	2
Аппаратная	Весь трафик узла	Высокая	+	Максимальные	1

ментированные интерфейсы, но отладка средства защиты уже не столь проста, поскольку данное средство реализуется в режиме ядра, а следовательно, большинство ошибок в нем могут привести к «падению» ОС и, кроме того, при отладке приходится иметь дело с ассемблерным кодом; высокая, – что проблемы с отладкой усугубляются частичным или полным отсутствием документации для разработки.

Прозрачность алгоритма определяет, должен ли пользователь предпринять какие-либо действия для того, чтобы передать свои данные с помощью данного алгоритма.

Возможности, предоставляемые ОС коду алгоритма

В зависимости от того, исполняется ли алгоритм в режиме пользователя или в привилегированном режиме − режиме ядра, ему предоставляются различные возможности. В режиме ядра разрешено выполнение всех команд процессора и доступна системная область памяти и оборудование, тогда как в пользовательском режиме некоторые команды запрещены, а системные области памяти недоступны [11].

Выбор конкретного способа реализации алгоритма зависит от первоначально предъявляемых к данной системе требований. Из вышеприведенного анализа сетевой архитектуры и сравнительной характеристики различных способов реализации средств защиты сетевой информации следует, что реализации на уровне промежуточных драйверов и драйверов сетевых устройств наиболее предпочтительны (даже с учетом фактора реализации дополнительной функциональности, не свойственной данному уровню относительно модели *OSI*, т. е, по сути дела, разработки некоторых функций драйвера транспорта) (табл. 4).

ОС *Windows NT* позволяет внедрять разработанные драйверы в стек сетевых драйверов без необходимости модификации существующих компонентов. Они могут взаимодействовать с компонентами исполнительной системы, ядром и *HAL*, получая тем самым максимальные возможности для реализации функций алгоритма. Данные драйверы способны контролировать весь сетевой трафик, через них будут проходить данные всех приложений, исполняющихся в системе [12].

В результате исследования сетевой архитектуры можно сделать вывод, что не на всех ее уровнях операционная система

Таблица 4. Зависимость способа реализации алгоритма от предъявляемых к нему требований

Требования к системе безопасности	Рекомендуемый уровень реализации средства защиты
Необходимо реализовать программу, обеспечивающую пассивный мониторинг компьютерной сети	Реализация на уровне приложения и *DLL*
Необходимо обеспечить активный моніторинг и управление трафиком	Реализация на уровне драйвера *ndis.sys*, драйвера транспорта, промежуточного драйвера или драйвера устройства

Windows NT позволяет расширять свои возможности и, в частности, возможности по реализации управления передачей данных по сети. Зачастую, если даже сетевая архитектура предоставляет возможность расширения своей функциональности, *Microsoft* ее не всегда документирует.

Наилучшим вариантом расширения функциональности ОС является реализация алгоритмов в виде промежуточных драйверов, располагающихся между драйверами транспортов и драйверами сетевых карт. Но в этом случае может возникнуть проблема корректной работы такого драйвера в сети, поскольку ОС позволяет встраивать собственные модули управления на нижних уровнях взаимодействия сетевых узлов.

Для реализации алгоритма на программном уровне необходимо обеспечить мониторинг трафика в сегменте с общим доступом к среде передачи данных, организовать управление доступом к среде передачи данных, разработать и внедрить программную реализацию стекового алгоритма *ANIO,* организовать сбор статистики в файл и разработать методику анализа для оценки производительности сети.

С учётом вышеизложенного для разрабатываемой программы устанавливаются такие требования:

- программа должна отслеживать прием и передачу пакетов, приходящих к драйверу сетевого адаптера как от пользовательских процессов, так и от компонент ОС, и сохранять передаваемые данные в *LOG*-файле;
- от программы не требуется определять протоколы, однако при анализе *LOG*-файлов должно быть известно, кто передавал и принимал пакет, в какой момент времени и какого размера;
- на основе полученной информации программа должна управлять передачей пакетов в сеть в соответствии с алгоритмом;
- пользователь должен иметь возможность управления программой (локально и удаленно), сохранять настройки файлов для последующей автоматической перенастройки программы;
- программа не должна приводить к сбоям в работе ОС или к существенным задержкам ввода/вывода.

Поток LOG. Необходимо организовать протоколирование данных, обрабатывающие в драйвере на высоких приоритетах *IRQL*, а функции *ZwCreateFile* и *ZwWriteFile* должны выполняться только на уровне *PASSIVE_LEVEL*. Подобная задача достаточно легко решается, если высокоприоритетный программный код будет помещать записи в промежуточный буфер, который будет сброшен на диск программным потоком, выполняющимся на подходящем для этого уровне *PASSIVE_LEVEL*. Системные программные потоки создаются вызовом *PsCreateSystemThread,* а завершиться они должны самостоятельно − выполнением вызова *PsTerminateSystemThread.* Поэтому сетевая служба порождает поток *LOG,* работающий на уровне *PASSIVE_LEVEL* и выполняющий все необходимые действия по сбору статистической информации.

Структура информационной записи *LOG*-файла:

```
struct LogFile // Основная структура
{
        UCHARMAC[6];    // MAC-адрес отправителя пакета
        short   Size;   // Размер пакета
        //Время прибытия пакета
        LARGE_INTEGER Time;
};
struct LogFileEx        // Дополнительная структура
{
        UCHAR MAC[6]; // MAC-адрес получателя
        short Flags;            // Флаги
        LONG SetTimeOut; // Установка времени ожидания тайм-аута
        //Количество тайм-аутов
        ULONG CountTimeOut;
};
```

Поток ANIO. Чтобы не блокировать работой алгоритма процесс приема пакетов из сети, управление стеком *ANIO*, необходимо организовать в отдельном потоке. Сетевая служба порождает поток *ANIO*, создает стек, вносит туда локальный *MAC*-адрес. Затем в замкнутом цикле выполняются такие действия:

- перевод потока в спящий режим;
- ожидание события приема пакета (или прекращение ожидания по тайм-ауту);
- определение причины, по которым поток был пробуждён (тайм-аут, прием пакета из сети, завершение работы потока);
- в случае получения пакета из сети производится оповещение алгоритма *ANIO* и передача указателя на *MAC*-адрес (если наступил тайм-аут, то в качестве указателя на *MAC*- адрес алгоритму передается значение *NULL*).

Алгоритм *ANIO* определяет момент наступления очереди передачи пакета в сеть (возвращает значение *TRUE*, если пора передать пакет); в данном случае запускается процедура *FreeSend*, пропускающая в сеть один пакет. При необходимости завершить поток обеспечивается его выход из замкнутого цикла, удаляется стек, завершается работа потока.

Рассматриваемое программное обеспечение (ПО) работает корректно только под Windows 2000 Profesional, поскольку разрабатывалось и тестировалось только под управлением данной ОС. Использована возможность работы с виртуальными машинами на основе *VMware* (рис. 6). С помощью *Dbgview.exe* можно наблюдать за работой службы и сохранять все результаты в LOG-файле. К данному ПО прилагается руководство пользователя *Dbgview.chm.*

Алгоритм *ANIO* работает в отдельном потоке и организует вращение стека, поэтому ему необходима информация о том, кто в данный момент времени пере-

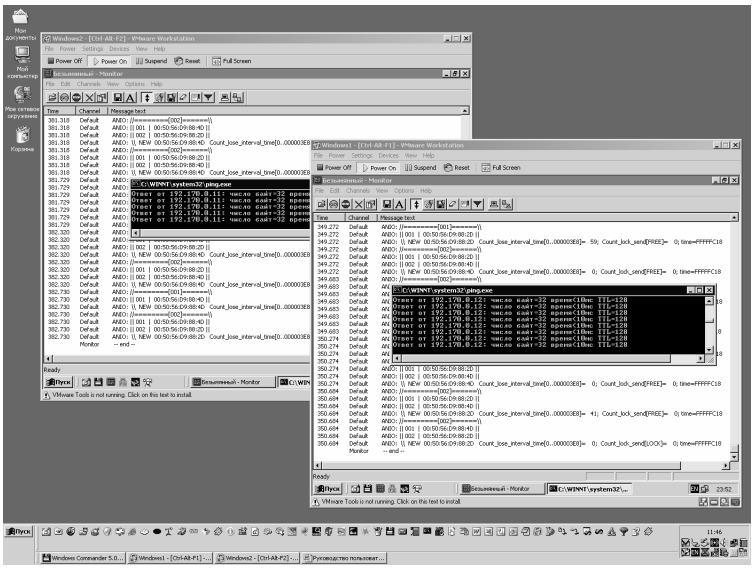


Рис. 6. Работа с виртуальными машинами на основе *VMware*

дает пакет. Для поддержания очереди необходимо запретить на некоторое время передачу очередного пакета, до наступления очереди передачи. Режим *TIME OUT* позволяет исключить возможную блокировку сетевой службы: по истечении определенного промежутка времени считается, что очередной узел пропустил свою передачу и будет исключен из стека.

Сетевая служба является промежуточным звеном между драйвером сетевого адаптера и стеком протоколов *tcp/ip*. Через нее проходят все пакеты, передающие в сеть или принимающие из сети. Данная служба может запретить или разрешить передачу в сеть любого пакета. Кроме того, она переводит адаптер в режим *NDIS_PACKET_TYPE_PROMISCUOUS.* Программа, соответственно, может просматривать все передающиеся по сети пакеты. В результате работы появляется *anio.log* − файл со статистикой.

Для эффективной работы алгоритма необходимо определить оптимальное время *TIME OUT*. При небольшом значении данной величины стек будет вращаться слишком быстро и узлы не будут успевать передавать свои пакеты. В этом случае процессор будет загружен выполнением операций со стеком, а ОС может «зависнуть». Если *TIME OUT* слишком велик, то передача данных будет выполняться медленно и много узлов сможет выполнить передачу пакета вне очереди.

Управление сетевой службой

Сетевая служба работает на уровне ядра, поэтому не имеет интерфейса управления. Но *ANIO* нуждается в централизованном управлении. Для управления работой сетевой службы разработано специальное программное обеспечение (приложение *ctrlANIO.exe*), позволяющее передавать в сеть пакеты любого содержания. Данное приложение является снифером, потому что позволяет просматривать все пакеты, передающиеся по сети. Управление выполняется с помощью пакетов определенного формата. Данное ПО использует библиотеку *WinPcap*, поэтому требует установки *WinPcap_3_x*.

Перед началом работы необходимо выбрать адаптер и запустить его. При этом выполняется перевод адаптера в режим *PROMISCUOUS* и начинается перехват пакетов из сети, который можно прекратить, нажав кнопку СТОП. Чтобы приостановить вывод пакетов на экран, следует нажать кнопку ПАУЗА. Реализована возможность очистки окна от ненужных пакетов.

Для удобства есть окно редактирования пакета − РЕДАКТОР УПРАВЛЯЮЩЕГО ПАКЕТА (рис. 7), его можно вызвать из меню или щелкнув на одном из пакетов из списка. При выборе пакета из

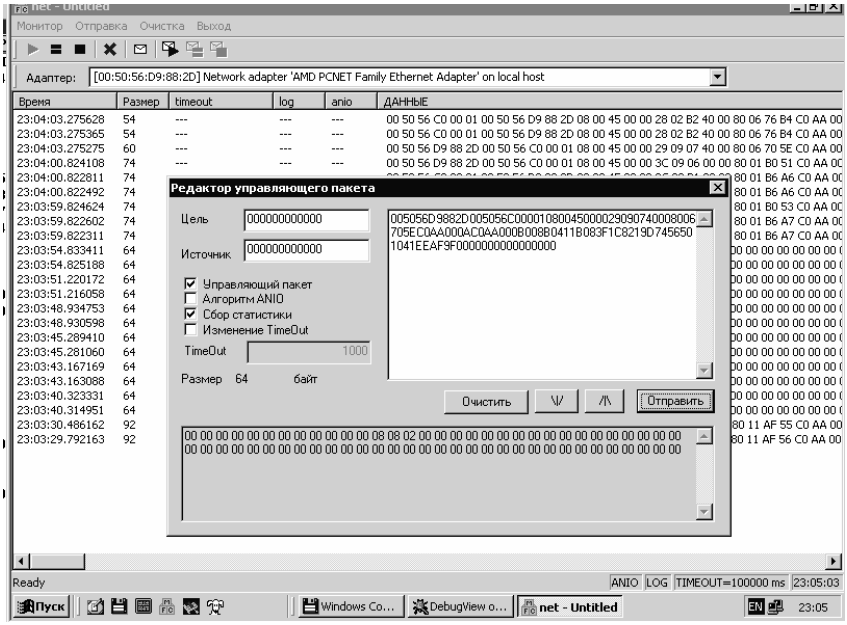


Рис. 7. Редактор управляющего пакета

списка его содержимое помещается в редактор, затем пакет можно повторно отправить в сеть. Имеется возможность передачи пакета как одиночно, так и многочисленно. Многочисленную передачу можно приостановить (кнопка ПАУЗА) либо прекратить (кнопка СТОП).

Выбор опции «управляющий пакет» активизирует элементы редактирования и генерацию управляющего пакета, позволяющий изменить режим работы сетевой службы *ANIO:*

- разрешить или запретить сбор статистики;
- запустить или остановить алгоритм *ANIO;*
- выполнить установку времени *TIME OUT;*
- указать *TIME OUT*;
-указать, от кого и кому предназначен пакет.

Все адаптеры в режиме *PROMISCUOUS* видят этот пакет и выполняют его команды, если установлена служба *ANIO.*

Обработка статистики

Обработка статистики осуществляется приложением *LogANIO* (рис. 8), открывающие файл *anio.log* и помещает его в базу данных *base\anio.db*. Если БД пуста, то необходимо загрузить данные из файла *anio.log*. В приложении разработан набор стандартных *SQL*-запросов, кроме того реализована возможность формирования собственных запросов. По результатам запросов можно построить диаграммы, характеризующие различные свойства сетевых протоколов доступа в канал.

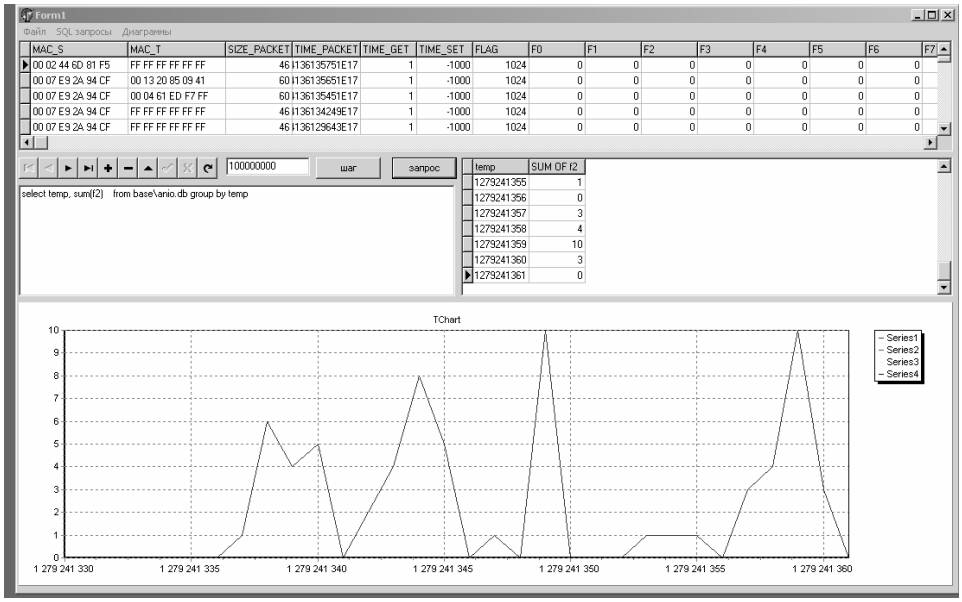


Рис. 8. Программное обеспечение *LogANIO.exe* для обработки статистики

Заключение

Разработанный программный работает в режиме ядра ОС и устанавливается как драйвер-фильтр поверх драйвера сетевого адаптера, что позволяет управлять режимами приема, просматривать все входящие и исходящие кадры, разрешать или запрещать прием определенных кадров из сети и их передачу в сеть. Сетевой драйвер, сетевые службы и ряд системных программ интегрированы в рамках инструментальных средств, позволяющих моделировать в реальных локальных сетях множество сетевых протоколов канального уровня на базе универсального алгоритма *ANIO*, с целью оптимизации характеристик функционирования существующих и вновь разрабатываемых методов доступа в локальных сетях компьютеров.

1. *Алишов Н.И.* Адаптивный стековый алгоритм универсального множественного доступа в распределенных системах и сетях компьютеров // УСиМ. – 2004. – № 1. – С. 59–72.
2. *Алишов Н.И., Петришина О.В.* Конфликты доступа в локальных вычислительных сетях // Математичні машини і системи. – 2006. – № 4. – С. 111–123.
3. *Устройство* для передачи информации: А.с. 1509970 СССР, МКИ G 08 C 19/28, G 06 F 13/00 / Б.Н. Малиновский, Н.И. Алишов, А.В. Кушнарев. – № 4360107/24; Заявл. 07.01.88; Опубл. 23.09.89, Бюл. № 35. – 10 с.
4. *AbdelallahElhadj H., Khelalfa H.M., Kortebi H.M.* An Experimental Sniffer Detector: Snifferwall Securite des Communications sur Internet. – Proc. of SECI02, 2002. – P. 69−79.
5. *Bhatt S., Fujimoto R., Ogieski A.* Parallel Simulation Techniques for Large-Scale Networks // IEEE Communication Magazine. – 1998 August. – P. 42–47.
6. *Петришина О.В.* Інструментальне середовище дослідження множинного доступу в локальних мережах // Вісті академії інженерних наук України. – 2005. – № 4(27). – С. 38–40.
7. *Петришина О.В.* Технология реализация универсального множественного доступа // Комп'ютерні засоби, мережі та системи. – 2006. – № 5. – С. 80–85.
8. *Полный* справочник по C++ / Пер. с англ. – М.: Изд. дом «Вильямс», 2004. – 800 с.
9. *Шеферд Дж.* Программирование на Microsoft Visual C++ .NET / Пер. с англ. – М.: Изд.-торг. дом «Русская редакция», 2003. – 928 с.
10. *Oney W.* Programming the Microsoft Windows Driver Model. – Redmond, Washington: Microsoft Press., 2002. – 675 p.
11. *Программирование* драйверов Windows. – М.: ООО «Бином-Пресс», 2004. – 480 с.
12. *Baker A., Lozano J.* The Windows 2000 Device Driver Book // A Guide for Programmers, Second Edition. – Prentice Hall PTR, 2000. – 480 p.

Получено 17.10.2007

Об авторе:

Алишов Надир Исмаил-оглы,

доктор технических наук.

Место работы автора:

Институт кибернетики
им. В.М. Глушкова НАН Украины,
03680, Киев-187,
проспект Академика Глушкова, 40.
Тел.: (044) 526 3427.
e-mail: anio@ukrtel.com