МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт Факультет приборостроения, информационных технологий и электроники	Кафедра «Информационная безопасность систем и технологий»
Специальность 10.05.03 –	Информационная безопасность автоматизированных систем
Специализация –	Защищенные автоматизированные системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНЫЙ МОДУЛЬ ПРОТИВОДЕЙСТВИЯ СКРЫТЫМ КАНАЛАМ ПЕРЕДАЧИ ИНФОРМАЦИИ В ПРОГРАММНО-АППАРАТНОМ УСТРОЙСТВЕ ШИФРОВАНИЯ IP-ТРАФИКА

Студент Скоркин Дмитрий Алексеевич

Руководитель Савенко П.Ю.

Нормоконтролёр Иванов А.П.

Рецензент зам. ген. директора АО «ПНИЭИ» по спецвопросам Сомкин С.А.

Работа допущена к защите (протокол заседания кафедры от 26.06.18 № 16)

Заведующий кафедрой С.Л. Зефиров

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 27.06.18 № 10)

Секретарь ГЭК А.П. Иванов

Пенза 2018

Подпись п.ю.савенко удостоверяю
инженер 1 кат. ОК

Проверил ВКР – 100 баллов
22.06.18г.

Оглавление

Введение 5
1 Используемые способы построения скрытых каналов 7
1.1 Анализ существующих способов построения скрытых каналов 7
1.2 Скрытые каналы, использующие межпакетные интервалы для скрытой передачи информации 12
1.3 Скрытые каналы, использующие размер передаваемых пакетов для скрытой передачи информации 13
2 Используемые способы противодействия скрытым каналам 14
2.1 Способы противодействия скрытым каналам 14
2.2 Схема противодействия скрытым каналам 15
2.3 Используемые в разрабатываемом модуле способы противодействия скрытым каналам 17
3 Разработка алгоритма работы программного модуля противодействия скрытым каналам 20
3.1 Общий алгоритм работы программного модуля противодействия скрытым каналам 20
3.2 Блок-схема алгоритма работы программного модуля противодействия скрытым каналам 22
3.3 Алгоритмы функций для программной реализации модуля противодействия скрытым каналам 23
4 Написание и отладка программного обеспечения модуля противодействия скрытым каналам 26
4.1 Используемые инструменты для разработки программного модуля противодействия скрытым каналам передачи информации 26
4.2 Формат структур данных для хранения информации 26
4.3 Программная реализация функций модуля 29
4.4 Отладка программного модуля противодействия скрытым каналам 50

5 Тестирование программного обеспечения модуля противодействия скрытым каналам 51
5.1 Описание процесса тестирования программного модуля 51
5.2 Тестирование модуля и оценка влияния работы модуля противодействия скрытым каналам на эффективную полосу пропускания полезных данных в канале связи 52
6 Вредные психофизиологические факторы, влияющие на психоэмоциональное состояние специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие 60
Заключение 63
Список используемых источников 64
Приложение А Алгоритмы функций программного модуля противодействия скрытым каналам 66
Приложение Б Тексты программного модуля противодействия скрытым каналам 69

Обозначения и сокращения

В настоящей пояснительной записке применяются следующие сокращения с соответствующими расшифровками:

- IP — internet protocol;
- MTU — maximum transmission unit;
- TCP — transmission control protocol;
- UDP — user datagram protocol;
- АС — автоматизированная система;
- ВКР — выпускная квалификационная работа;
- Н — нарушитель;
- ПК — персональный компьютер;
- ПО — программное обеспечение;
- ПЭВМ — персональная электронно-вычислительная машина;
- СЗ — средство защиты;
- СК — скрытый канал.

Введение

В настоящее время широкое распространение получили скрытые каналы в сетях пакетной передачи данных, поскольку возможно использование особенностей стека протоколов TCP/IP для негласной передачи информации.

Потребность в создании и непрерывном улучшении способов противодействия утечке информации через скрытые каналы обусловлена тем, что такие каналы могут быть построены, несмотря на использование традиционных методов сетевой защиты, таких как межсетевое экранирование, туннелирование, и т.д. Как видно из уже существующих исследований данной проблемы, угроза сохраняется, даже при передаче информации в зашифрованном виде.

Актуальность настоящей выпускной квалификационной работы (ВКР) заключается в разработке программного модуля противодействия скрытым каналам передачи информации в программно-аппаратном устройстве шифрования IP-трафика, с помощью которого коммерческие и государственные организации получат возможность повысить защищенность информационных систем.

Целью работы является программная реализация модуля противодействия скрытым каналам передачи информации в программно-аппаратном устройстве шифрования IP-трафика.

В первом разделе рассмотрены основные способы построения скрытых каналов, представлена схема функционирования скрытого канала, классификация скрытых каналов, а также описаны типы скрытых каналов, противодействие которым осуществляется при помощи разрабатываемого программного модуля.

Во втором разделе представлена схема противодействия скрытым каналам, рассмотрены существующие и используемые разрабатываемым программным модулем способы противодействия скрытым каналам.

В третьем разделе описывается общий алгоритм работы программного модуля противодействия скрытым каналам, разрабатывается блок-схема алгоритма работы программного модуля противодействия скрытым каналам, на основе которой реализуется программное обеспечение модуля.

В четвёртом разделе описывается процесс написания и отладки программного обеспечения модуля противодействия скрытым каналам, основные функции, выполняемые программой.

В пятом разделе описан процесс тестирования программного обеспечение модуля противодействия скрытым каналам. Также в данном разделе осуществляется оценка влияния работы модуля противодействия скрытым каналам передачи информации в сетях передачи IP-трафика на эффективную полосу пропускания полезных данных в канале связи.

Результатом выполнения ВКР является программный модуль противодействия скрытым каналам передачи информации в программно-аппаратном устройстве шифрования IP-трафика.

1 Используемые способы построения скрытых каналов

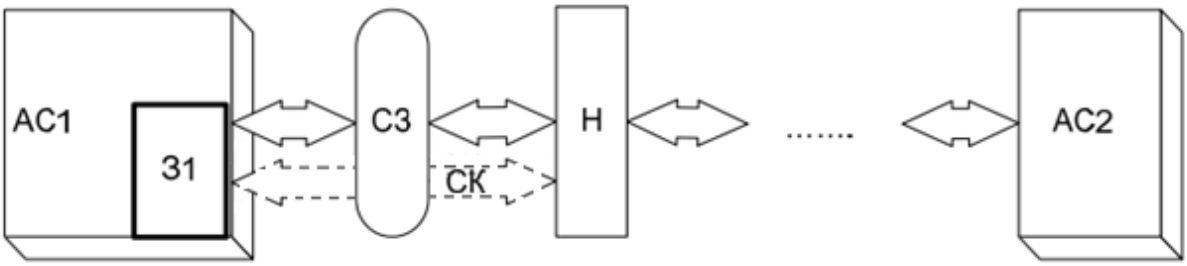
1.1 Анализ существующих способов построения скрытых каналов

1.1.1 Механизм функционирования скрытого канала

Скрытым каналом (СК) является коммуникационный канал, не предусмотренный разработчиком, использующийся для нарушения политики безопасности системы [1].

Наличие или возможность построения скрытых каналов в автоматизированной системе (АС) может повлечь за собой как утечку конфиденциальной информации из АС, так и негативное влияние извне на деятельность АС.

На рисунке 1 представлена схема взаимодействия скрытого канала с автоматизированной системой и средствами защиты [2].



Примечание:

АС1, АС2 – два сегмента автоматизированной системы, соединенных через сеть общего пользования через средства защиты (СЗ);

СЗ – средства защиты АС;

Н – внешний нарушитель, осуществляющий несанкционированный доступ к АС либо оказывающий негативное влияние на неё;

З1 – внутренний нарушитель системы, формирующий скрытый канал для связи с внешним нарушителем (Н).

Рисунок 1 – Схема взаимодействия СК и АС

В качестве внутреннего нарушителя в данном взаимодействии может выступать не человек, а внедренное средство, с помощью которого возможно построение скрытого канала.

В российском стандарте ГОСТ Р 53113.1–2008 скрытые каналы подразделяются по механизму передачи информации на каналы по памяти и каналы по времени [1].

При разработке модуля противодействия скрытым каналам стоит учитывать, что он является составной частью устройства шифрования IP-трафика, обеспечивающего комплексную защиту от скрытых каналов. В соответствии с техническим заданием, разрабатываемый модуль противодействия скрытым каналам должен противодействовать скрытым каналам, построение которых нарушителем возможно с помощью:

— изменения длин передаваемых пакетов;

— изменения длин межпакетных интервалов.

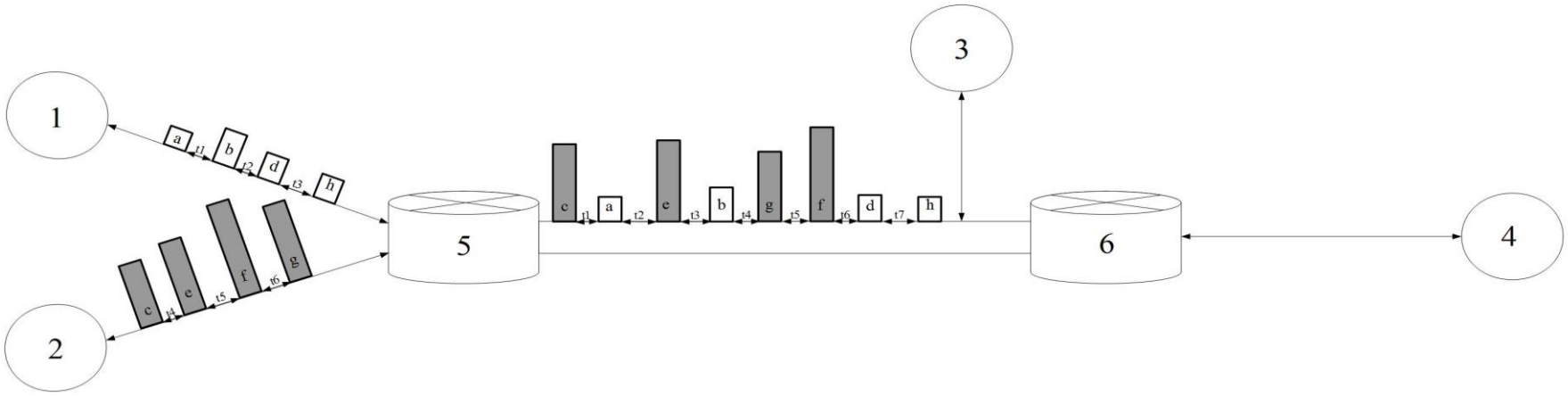
Механизм скрытой передачи информации на основе изменения длин передаваемых пакетов является скрытым каналом по памяти, а механизм на основе изменения длин межпакетных интервалов — скрытым каналом по времени. Каждый из механизмов отдельно рассмотрены в следующих подразделах.

Представленную на рисунке 1 схему можно видоизменить применительно к типу скрытых каналов, противодействие которым осуществляется модулем.

В качестве средства защиты на представленной далее схеме будет выступать маршрутизатор со встроенным устройством шифрования трафика, составной частью которого является модуль противодействия скрытым каналам. Маршрутизаторы каждого сегмента сети связаны между собой туннельным интерфейсом. Внутри туннеля организуются потоки, каждый из которых контролируется модулем противодействия скрытым каналам.

Функционирование скрытого канала по данной схеме возможно только при выключенном модуле противодействия скрытым каналам.

Схема функционирования скрытого канала приведена на рисунке 2.



1 — Субъект, имеющий санкционированный доступ к информации
2 — Нарушитель (субъект, имеющий санкционированный доступ к информации или внедренное средство нарушителя безопасности)
3 — Злоумышленник, с которым 2 осуществляет несанкционированное информационное взаимодействие
4 — Субъект, с которым 1 осуществляет санкционированное информационное взаимодействие
5, 6 — Маршрутизаторы, связанные между собой туннельным интерфейсом
a, b, d, h — Пакеты, отправляемые субъектом 1
c, e, f, g — Пакеты, отправляемые нарушителем или внедренным агентом

Рисунок 2 — Схема функционирования скрытого канала

Схема (см. рисунок 2) изображает механизм функционирования скрытых каналов, действие которых основано на использовании межпакетных интервалов и размера передаваемых пакетов для скрытой передачи информации.

Субъектом, имеющим санкционированный доступ, является работник организации, который осуществляет санкционированное взаимодействие с другими работниками. Нарушителем является работник, взаимодействующий со злоумышленником, не имеющим доступа к информации. Также нарушителем может быть внедренное средство нарушителя безопасности. Обязательным условием того, что канал передачи информации скрытый является то, что информационное взаимодействие между нарушителем и злоумышленником невидимо для остальных работников и средств защиты.

Злоумышленник наблюдает за зашифрованным трафиком в туннеле между двумя сегментами сети. Несмотря на то, что все данные зашифрованы, для наблюдения остаются доступными длины пакетов и межпакетные интервалы.

1.1.2 Скрытые каналы по памяти

Скрытый канал по памяти использует память, в которую отправитель записывает данные, а получатель считывает их, причем сторонний наблюдатель не знает того места в памяти, куда помещена записываемая информация. В скрытых каналах по памяти информация передается через доступ отправителя на запись и получателя на чтение к одним и тем же ресурсам или объектам.

Скрытые каналы по памяти подразделяются на несколько групп:

— скрытые каналы, основанные на модификации битов полей заголовков передаваемых пакетов;

— скрытые каналы, основанные на модификации данных в теле передаваемых пакетов;

— скрытые каналы, основанные на изменении длин передаваемых пакетов [3].

Разрабатываемый модуль противодействия скрытым каналам применим для третьей группы скрытых каналов. Противодействие скрытым каналам, которые относятся ко второй и третьей группам осуществляется модулем шифрования, входящим в состав устройства шифрования IP-трафика, составной частью которого является модуль противодействия скрытым каналам передачи информации.

1.1.3 Скрытые каналы по времени

Скрытый канал по времени основан на том, что отправитель кодирует информацию с помощью интервалов между событиями, которыми может управлять. В данном случае информация кодируется в межпакетных интервалах. Скрытый канал по времени характеризуется доступом отправителя и получателя к одному и тому же процессу или изменяемому во времени атрибуту.

Для построения скрытых каналов по времени нарушитель должен обладать одной или несколькими из следующих возможностей:

— возможность формировать собственные ложные IP-пакеты произвольной длины, то есть генерировать собственный фиктивный трафик;

— возможность буферизовать все пакеты, подлежащие передаче из внутренней или внешней сети;

— возможность передавать пакеты по каналу в заранее определенный момент времени.

Разрабатываемый модуль применяется для противодействия скрытым каналам, основанным на изменении межпакетных интервалов.

Среди скрытых каналов, основанных на изменении скорости передаваемых пакетов, выделяют простейший случай, где нулем кодируется

бездействие отправителя, а единицей – отправка пакетов в течение некоторого интервала времени.

Скрытые каналы по времени всегда являются каналами с шумом. Это обусловлено следующими факторами:

— существует ненулевая вероятность потери пакетов;

— время следования пакета зависит от характеристик сети [3].

1.2 Скрытые каналы, использующие межпакетные интервалы для скрытой передачи информации

Для организации нарушителем данного типа скрытых каналов необходимо наличие у него следующих возможностей:

— генерация собственного фиктивного трафика;

— буферизация всех пакетов, подлежащих передаче из внутренней или внешней сети;

— передача пакетов в канал в заранее определенный момент времени.

Построение данного типа скрытых каналов возможно при использовании двух подходов — изменение межпакетных интервалов и изменении скорости передаваемых пакетов.

Примером скрытого канала, основанного на изменении скорости передаваемых пакетов является случай, когда нулем кодируется бездействие отправителя, а единицей отправка пакетов в течение определенного промежутка времени.

Отправитель выбирает скорость передачи пакетов для каждого интервала времени, а получатель измеряет скорость передачи пакетов и восстанавливает таким способом скрытую информацию

Способ кодирования скрытой информации путем изменения длин межпакетных интервалов подразумевает кодирование одного отрезка длин межпакетных интервалов нулем, а другого — единицей [3].

Емкость скрытого канала существенно меняется во времени в зависимости от объема и интенсивности трафика, который передается по основному каналу.

1.3 Скрытые каналы, использующие размер передаваемых пакетов для скрытой передачи информации

Чтобы создать скрытые каналы, основанные на изменении длин передаваемых пакетов, нарушителю необходимо иметь одну или несколько из следующих возможностей:

— изменение длины любого пакета;

— генерация собственного фиктивного трафика;

— буферизация всех пакетов, подлежащих передаче из внутренней или внешней сети;

— передача пакетов по каналу в заранее определенный момент времени [3].

Емкость скрытого канала зависит от сервисов, которые работают внутри защищенных сегментов.

2 Используемые способы противодействия скрытым каналам

2.1 Способы противодействия скрытым каналам

Для обеспечения защиты автоматизированных систем от организации скрытых каналов в средствах защиты в настоящее время применяются следующие механизмы защиты:

— заполнение неиспользуемых полей пакетов фиксированными значениями;

— выравнивание длин пакетов;

— выравнивание длин межпакетных интервалов;

— нормализация процесса передачи пакетов различным адресатам;

— контроль и фильтрация пакетов по заданным правилам (межсетевое экранирование);

— переформирование пакетов, передаваемых или принимаемых из коммуникационной среды (использование прокси-серверов);

— туннелирование трафика с использованием алгоритмов криптографической защиты, таких как шифрование и имитозащита [4].

Полную защиту от организации скрытых каналов можно обеспечить, применяя средство защиты, обеспечивающее туннелирование трафика с использованием алгоритмов шифрования и имитозащиты, выравнивание длин пакетов, длин межпакетных интервалов и обеспечивающее равномерную передачу пакетов всем возможным обособленным сегментам автоматизированной системы [2].

Однако алгоритмы, обеспечивающие выравнивание длин межпакетных интервалов и равномерную передачу данных всем абонентам сети, негативно сказываются на общей пропускной способности коммуникационных каналов.

На практике данная совокупность мер защиты должна применяться только для случая соединения двух сегментов сети по схеме точка–точка с использованием выделенных линий связи. При этом основное негативное

влияние на работоспособность сети связи, оказывает мера защиты, обеспечивающая постоянную длину межпакетного интервала, то есть, по сути, установление постоянной скорости передачи информации.

Таким образом, для критически важных объектов необходимо использовать меры защиты, обеспечивающие полную защиту от организации скрытых каналов. В остальных случаях необходимо реализовывать совокупность мер защиты, обеспечивающих снижение пропускной способности возможных скрытых каналов до уровня, не представляющего серьёзной угрозы безопасности автоматизированной системы.

Вопрос определения допустимого уровня пропускной способности скрытого канала конкретной АС является отдельной сложной задачей и в настоящее время полностью не решен [2].

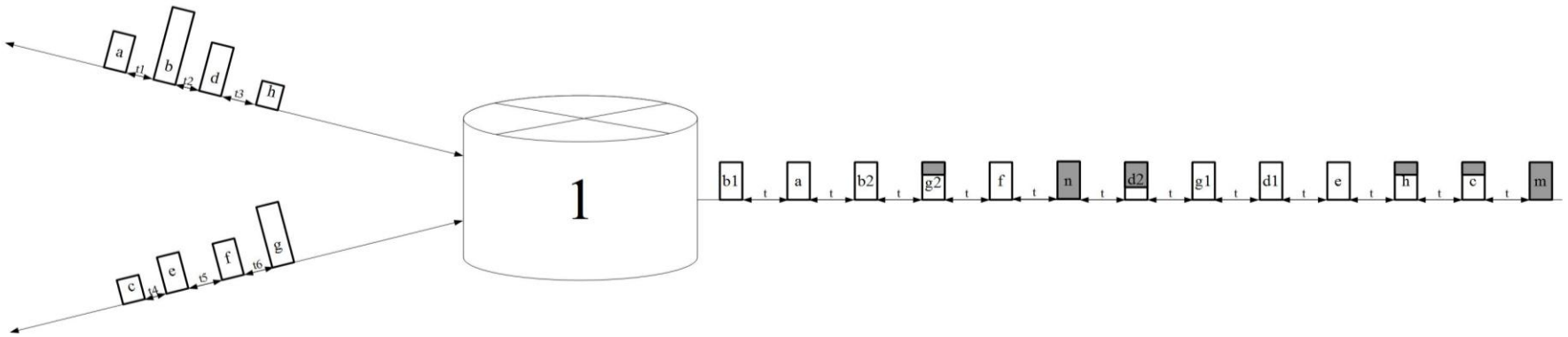
Пропускная способность скрытых каналов вычисляется методами теории информации. Часто пропускная способность скрытых каналов измеряется отношением количества скрытно переданной информации к количеству информации в легальном процессе.

В силу того, что скрытые каналы обладают, как правило, небольшой пропускной способностью, может сложиться мнение, что они не представляют опасности. Часто в таких случаях устанавливается порог на пропускную способность, ниже которого канал считается не опасным[5].

Таким образом, защиту указанными механизмами возможно реализовать с помощью устройства шифрования IP-трафика, в состав которого входит разрабатываемый модуль противодействия скрытым каналам.

2.2 Схема противодействия скрытым каналам

Противодействие скрытым каналам, организованным с использованием межпакетных интервалов и длин передаваемых пакетов осуществляется в модуле противодействия по схеме, которая представлена на рисунке 3.



1 — Маршрутизатор со встроенным устройством шифрования IP-трафика
a, b, c, d, e, f, g, h — Пакеты, имеющие разную длину и межпакетные интервалы
a1, b1, b2, c, d1,d2, e, f, g1, g2, h1 — Пакеты с фиксированной длиной l и межпакетными интервалами t
m, n — Сгенерированные модулем противодействия пакеты фиксированной длины

Рисунок 3 — Схема противодействия скрытым каналам

На вход модуля противодействия скрытым каналам передачи информации подаются пакеты различной скоростью, размер пакетов может быть любым. С выхода модуля считывается трафик, в котором все пакеты имеют постоянную длину и равные межпакетные интервалы.

То есть, задачей модуля является выравнивание межпакетных интервалов t1-t7 до фиксированной величины t и, в зависимости от размера принятого пакета, разбиение его на несколько пакетов, дополнение нулевыми данными до необходимой длины или генерация пакетов, содержащих нулевые данные.

2.3 Используемые в разрабатываемом модуле способы противодействия скрытым каналам

Используемые в модуле способы противодействия основаны на выравнивании размеров пакетов, передаваемых во внешние сети, и исключении зависимости межпакетных интервалов на выходе прибора от межпакетных интервалов на его входе. Таким образом, исключаются скрытые каналы передачи информации, возникающие путем манипуляции размерами пакетов и интервалами их передачи.

Используемым подходом к реализации защиты от утечки по скрытым каналам передачи информации является простая реализация механизма выравнивания всех пакетов до единого размера и генерации «ложных» пакетов таким образом, чтобы интенсивность внешнего трафика оставалась неизменной и не зависела от интенсивности внутреннего трафика.

Для включения данного механизма необходимо задать основные параметры модуля противодействия скрытым каналам. Такими параметрами являются: максимальная пропускная способность внешнего канала (Vmax), максимальная пропускная способность канала на указанных направлениях (Vn_max), минимально допустимая полоса пропускания на заданном направлении (Vn_min), размер пакетов на указанном направлении (Psize)

(берется равным MTU данного направления), время поддержания активности канала (Ts).

При включенном модуле противодействии скрытым каналам, между парами устройств шифрования организуются виртуальные каналы связи, удовлетворяющие заданным характеристикам.

Все передаваемые пакеты между ними, перед зашифрованием выравниваются до размера P_size, если же входящие пакеты по размеру больше чем P_size, то они разбиваются на пакеты длиной P_size и остаток. Остаток перед зашифрованием так же выравнивается до размера P_size. Таким образом, на выходе устройства шифрования трафика, во внешнюю сеть передаются пакеты размером (P_size+Paddition), где Paddition – размер служебных данных и IP заголовка, добавляемых шифратором для передачи пакета через внешние сети. Скорость трафика так же поддерживается на заданном уровне, таким образом, что скорость на данном направлении не превышает Vn_max, а общая скорость на выходе устройства не превышает Vmax. При этом если отсутствует входящий трафик, то производится генерация ложных пакетов, которые во внешнем канале не отличаются от обычного трафика.

Общая внешняя полоса пропускания, установленная как Vmax, делится равномерно (или же по какому-либо алгоритму распределения) между всеми активными направлениями. Направление считается активным, если по нему передаются данные между устройствами шифрования. Началом передачи данных, служит начало передачи данных из внутренней сети. Активность канала поддерживается заданное время (Ts), после начала передачи данных. Если по истечению времени Ts, данные на передачу из внутренней сети отсутствуют, то канал связи разрывается и полоса, выделенная для данного направления, освобождается. Если же данные на передачу присутствуют, то активность поддерживается еще один интервал времени Ts.

Таким образом, модуль противодействия скрытым каналам с одной стороны вносит избыточность как во внутренний передаваемый трафик, за счет

дополнения всех передаваемых пакетов до размера P_size, так и во внешний трафик, за счет генерации ложных пакетов, при отсутствии входящего трафика на заданном направлении, а так же ограничивает полосу пропускания заданными настройками.

3 Разработка алгоритма работы программного модуля противодействия скрытым каналам

3.1 Общий алгоритм работы программного модуля противодействия скрытым каналам

Для понимания принципа функционирования разрабатываемого модуля в устройстве шифрования IP-трафика рассмотрим схему, отражающую место программного модуля противодействия скрытым каналам в архитектуре устройства шифрования IP-трафика. Схема представлена на рисунке 4.

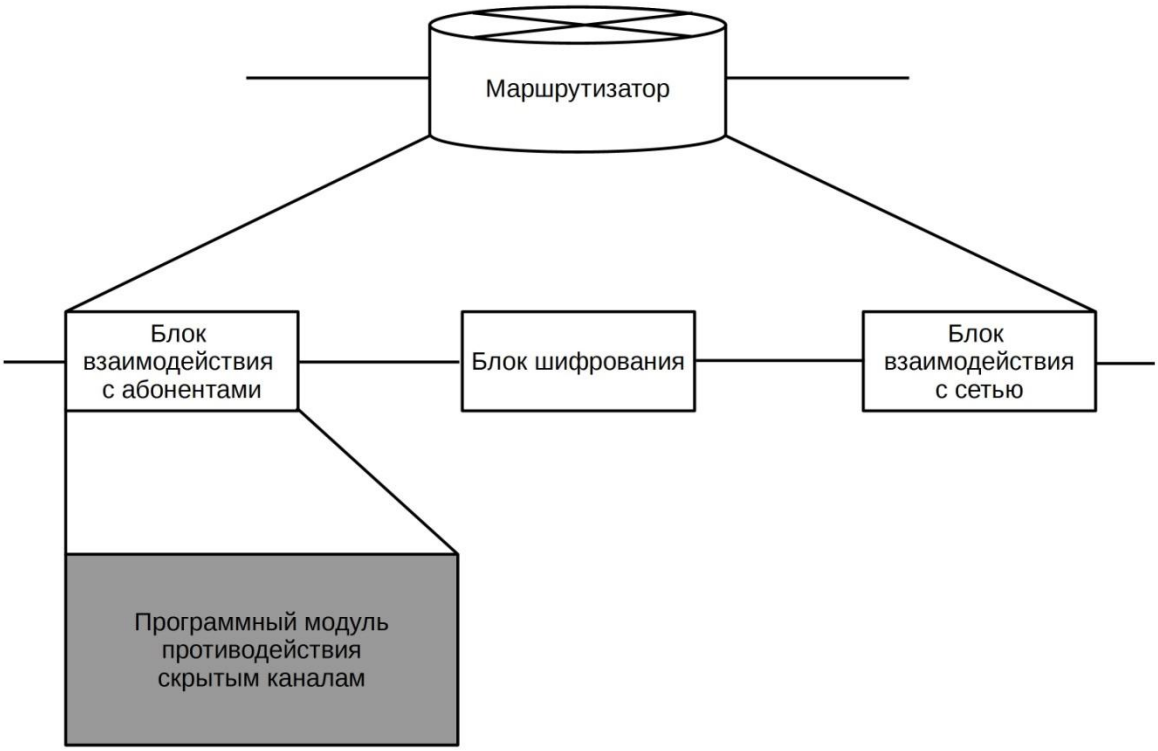


Рисунок 4 — Место программного модуля противодействия скрытым каналам в архитектуре устройства шифрования IP-трафика

На схеме (см. рисунок 4) устройством шифрования IP-трафика является маршрутизатор, в состав которого входят: блок взаимодействия с абонентами, блок шифрования и блок взаимодействия с сетью.

Разрабатываемый программный модуль противодействия скрытым каналам входит в состав блока взаимодействия с абонентами.

Более подробно алгоритм работы программного модуля противодействия скрытым каналам во взаимодействии с внутренней сетью отражает схема, представленная на рисунке 5.

Рисунок 5 — Общий алгоритм работы программного модуля противодействия скрытым каналам

Драйвер стыка с Ethernet принимает пакеты из внутренней сети и отправляет их на маршрутизацию.

После маршрутизации пакеты передаются в очередь на зашифрование.

Модуль противодействия скрытым каналам принимает пакеты из очереди на зашифрование по сигналу, который периодически формируется таймером.

После обработки модулем выравнивания, пакеты помещаются в очередь пакетов в стык с блоками шифрования.

Драйвер стыка с блоком шифрования забирает пакеты из очереди и передает их в блок шифрования.

3.2 Блок-схема алгоритма работы программного модуля противодействия скрытым каналам

На рисунке 6 представлена блок-схема алгоритма работы программного модуля противодействия скрытым каналам.

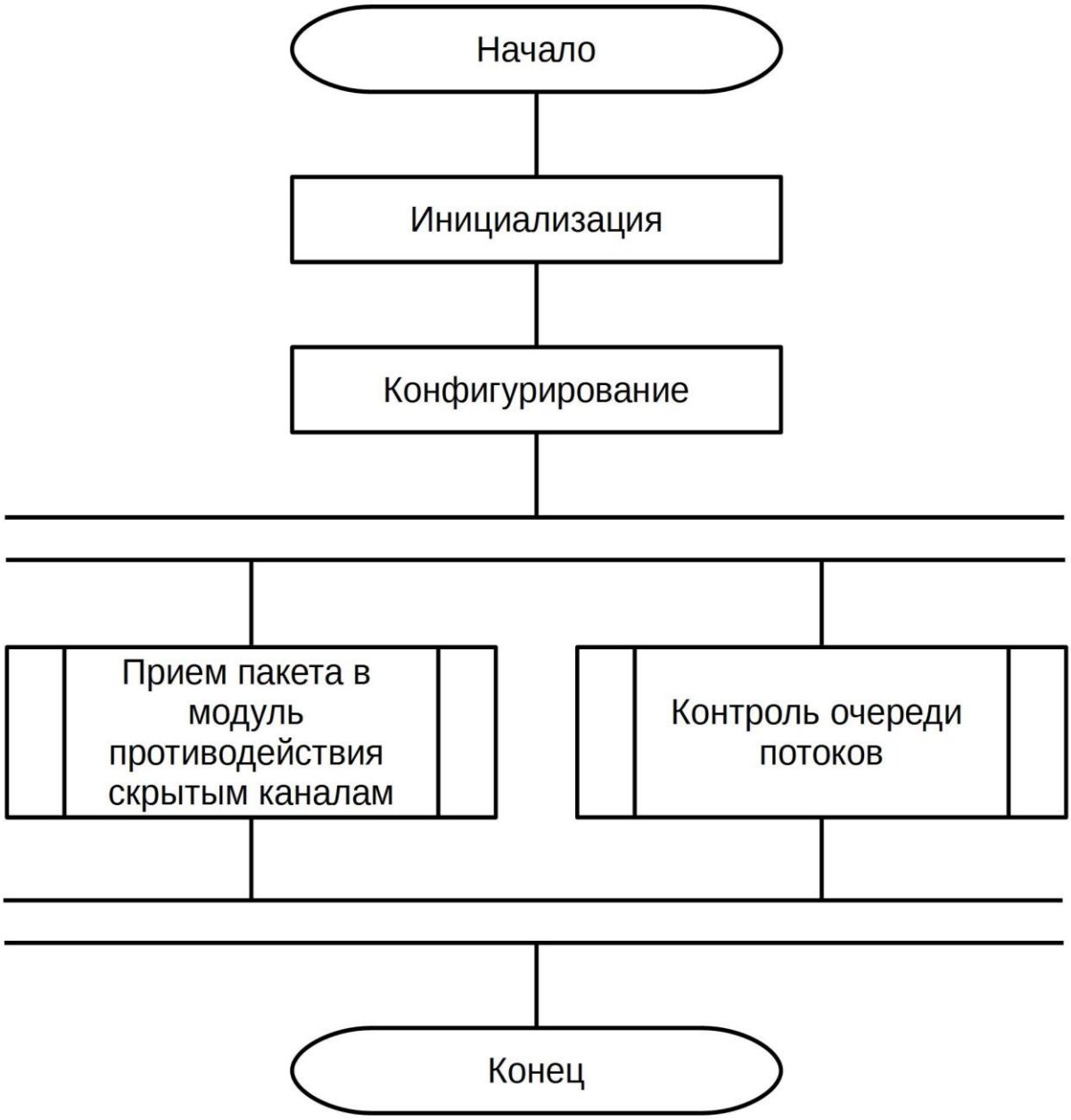


Рисунок 6 — Блок-схема алгоритма работы программного модуля противодействия скрытым каналам

Для подготовки модуля к работе необходимо произвести его инициализацию.

Далее осуществляется конфигурирование модуля. В процессе конфигурирования задаются параметры нормализации трафика, конфигурации потоков внутри туннеля, устанавливаются параметры потока на выходе устройства.

Прием пакета в модуль противодействия скрытым каналам и контроль очереди потоков выполняются модулем параллельно.

3.3 Алгоритмы функций для программной реализации модуля противодействия скрытым каналам

Инициализацию можно разделить на два процесса:

— инициализация нормализации;

— сброс прежних конфигураций.

Работа устройства начинается с инициализации. Инициализация нормализации необходима при пуске модуля противодействия скрытым каналам.

При повторном старте модуля необходим сброс прежних конфигураций.

Конфигурирование состоит из следующих процессов:

— задание параметров нормализации трафика;

— указание конфигураций потоков при создании потоков в туннеле.

Задание параметров нормализации трафика включает установку времени поддерживания потока в секундах (t) и максимально допустимой скорости исходящего трафика (Vmax).

При создании потоков в туннеле необходимо указать их конфигурации. Конфигурациями потока являются номер потока (num_sa), длина пакета (mtu), максимальная пропускная способность канала на указанных направлениях (Vn_max), минимально допустимая полоса пропускания на заданном направлении (Vn_min).

В соответствии с техническим заданием на ВКР модуль должен выполнять три основные функции:

— настройка фиксированной длины передаваемых пакетов;

— настройка фиксированного интервала между моментами выдачи IP-пакетов;

— прием IP-пакета в программный модуль для обработки.

Функция приема пакета в программный модуль для обработки и функция настройки фиксированной длины передаваемых пакетов реализуется в модуле с помощью функции приема пакета в модуль противодействия скрытым каналам.

Функция приема пакета в модуль противодействия скрытым каналам начинает выполняться приема пакета функцией и проверки правильности ссылки на поток нормализации. Далее осуществляется проверка условия нахождения потока. При условии, что поток найден происходит добавление нового потока в очередь, в противном случае проверяется условие успешного добавления найденного потока в очередь. В случае успешного добавления завершается выполнение данной функции, при неуспешном добавлении функция выполняется с проверки правильности ссылки на поток нормализации.

Блок-схема рассмотренной функции представлена в приложении А на рисунке А.1.

Функция настройки фиксированного интервала между моментами выдачи IP-пакетов и функция настройки фиксированной длины передаваемых пакетов реализуются в модуле с помощью контроля очереди потоков.

Функция контроля очереди потоков начинает выполняться с проверки очереди потоков. При наличии потока в очереди проверяется условие необходимости извлечения пакета из очереди. При неуспешной проверке данного условия ожидается увеличение системного времени. В случае успеха проверяется наличие пакета в очереди. Если в очереди есть пакет, то пакет извлекается и выравнивается до фиксированной длины mtu, в случае отсутствия пакета, формируется нулевой пакет.

Далее проверяется время жизни потока. Если оно истекло, то проверяется условие «полезности» увеличенного пакета. При выполнении условия продлевается время жизни потока и происходит отправка пакета во внешний модуль. Если условие не выполняется, то поток удаляется. После происходит смена потока, и функция выполняется снова.

Блок-схема данной функции приведена в приложении А на рисунке А.2.

Следующим шагом в разработке модуля противодействия скрытым каналам является его программная реализация. Для этого необходимо определить набор инструментов для написания и отладки модуля.

4 Написание и отладка программного обеспечения модуля противодействия скрытым каналам

4.1 Используемые инструменты для разработки программного модуля противодействия скрытым каналам передачи информации

Для разработки проектов на языке программирования C в среде разработки Eclipse IDE был установлен модуль расширения Eclipse IDE for C/C++ Developers.

Для создания make-файлов для сборки ARM-проектов установлен модуль расширения GNU ARM Eclipse Plug-in.

Компилятор языка C для процессоров семейства ARM arm-none-eabi-gcc содержится в наборе программ, необходимых для компиляции и генерации выполняемого кода из исходных текстов Sourcery CodeBench Lite Toolchain.

4.2 Формат структур данных для хранения информации

Для хранения информации о потоках в туннеле, об очереди потоков в туннеле и конфигурировании параметров нормализации трафика используются структуры.

Структура потока в туннеле:

```
struct thread{
    struct zas_prio_queue pkt_queue; // Очередь пакетов, отложенных на передачу
    unsigned long long time_call; // Расчетное время отправки пакета из очереди
    unsigned int zas_ip; // Адрес аппарата, с которым устанавливается сеанс
    unsigned int time_dead; // Время окончания действия потока (в единицах системного времени)
    unsigned int num_sa; // Номер профиля туннелей
    struct thread *next_ln, *prev_ln; // Очередь потоков
```

```
    struct aks_selector* sel; // Ссылка на туннель
};
```

Структура очереди потоков в туннеле:

```
struct queue_thread{
    unsigned int flg_max_thread; // флаг достижение максимального количества потоков
    unsigned int size_queue_thread; // количество потоков в очереди потоков
    struct thread *next; // указатель на первый поток в очереди
    struct thread *current; // указатель на текущий обрабатываемый поток
    unsigned int num_sa; // номер профиля туннелей
    unsigned int mtu_thread; // размер выравнивания пакетов
    unsigned int Vmax_thread; // максимальная скорость потока (байт/сек)
    unsigned int Vmin_thread; // минимальная скорость потока (байт/сек)
    unsigned int Vpkt_max_thread; // максимальная пакетная скорость потока (пак/сек)
    unsigned int Vpkt_min_thread; // минимальная скорость потока (пак/сек)
    unsigned int Vcurrent_thread; // текущая скорость потока (байт/сек)
    unsigned int Vpkt_current_thread; // текущая скорость потока (пак/сек)
    unsigned int init_thread; // количество инициализированных потоков
    unsigned int Vmax_thread_ext; // максимальная
```

```
скорость потока на выходе устройства(байт/сек)
    unsigned int Vmin_thread_ext; // минимальная скорость
потока на выходе устройства(байт/сек)
    unsigned int Vmax_ext; // максимальная скорость
потока (байт/сек)
    unsigned int Vmin_ext; // минимальная скорость потока
(байт/сек)
    unsigned int Vmax; // максимальная скорость потока
(байт/сек)
    unsigned int Vmin; // минимальная скорость потока
(байт/сек)
    unsigned int Vpkt_max; // максимальная пакетная
скорость потока в целом (пак/сек)
    unsigned int Vpkt_min; // минимальная пакетная
скорость потока в целом (пак/сек)
    unsigned int Vcurrent; // текущая скорость потока
    unsigned int Vpkt_current; // текущая пакетная
скорость потока
    unsigned int Tcurrent; // текущий расчитанный
интервал срабатывания таймера (микро сек)
    unsigned int Tmin; // минимальное расчетное время
срабатывания таймера при максимальной скорости (мк сек)
    unsigned int Tmax; // максимальное расчетное время
срабатывания таймера при минимальноей скорости (мк сек)
    unsigned long long save_time; // сохраненное
системное время (для расчета приоритета)
};
```

Структуры конфигурирования параметров нормализации трафика:

```
struct norm_cfg{
    unsigned Vmax; // максимальная выходная интенсивность
```

```
внешнего потока (байт/сек)
    unsigned Time; // гарантированное время поддержания
интенсивности трафика на каждом открытом направлении
    unsigned char flg_enable; // включение механизма
нормализации трафика (0 - выключен, 1 - включен)
} m_cfg;
struct norm_stat{
    unsigned transfer_pkt[2]; // пакетов в обход модуля
нормализации
    unsigned input_pkt[2]; // всего пакетов поставлено в
очередь
    unsigned all_pkt[2]; // всего пакетов отправлено
    unsigned good_pkt[2]; //отправлено полезных пакетов
    unsigned null_pkt[2]; // отправлено ложных пакетов
    unsigned frag_pkt[2]; // фрагментированно пакетов
    unsigned dpopped_pkt[2]; // отброшено полезных
пакетов
    unsigned transfer_queue_pkt[2]; // перемещеннных
пакетов из одной очереди в другую
    unsigned add_err[2]; // ошибки добавления новых
потоков
    unsigned long long save_time; // последнее время
вывода статистики
} m_stat;
```

4.3 Программная реализация функций модуля

Процесс реализации функций модуля противодействия скрытым каналам, действие которого основано на нормализации трафика состоит из нескольких этапов.

Работа устройства начинается с инициализации. Инициализация нормализации необходима при пуске модуля противодействия скрытым каналам.

Инициализация модуля состоит в выполнении функций инициализации нормализации и сброса прежних конфигураций при повторном пуске модуля.

Ниже приведен код функции инициализации нормализации.

```
void init_normalize(){
    unsigned i;
    DEBUG_NORM("\n init_normalize...");
    memset(&m_cfg, 0, sizeof(m_cfg));
    memset(&m_stat, 0, sizeof(m_stat));
        for(i = 0; i < CONFIG_MAX_SA; i++) {
        memset(&m_queue_thread[i], 0, sizeof(struct
queue_thread));
        m_queue_thread[i].num_sa = i;
        task_init(&task_norm[i],
              control_thread, (void
*)&m_queue_thread[i]);
        timer_init(&timer_norm[i],
         control_thread_time, (void
*)&m_queue_thread[i]);
    }
    // инициализация свободной очереди потоков
    free_thread = &m_thread[0];
    for(i = 0; i < (CONFIG_MAX_SESSIONS - 1); i++)
    {
        m_thread[i].next_ln = &m_thread[(i+1)];
        zas_prio_queue_init(&m_thread[i].pkt_queue,
MAX_PKT_QUEUE);
    }
```

```
    m_thread[i].next_ln = 0;
    normalize_config_flush();
    return;
}
```

При повторном старте модуля необходим сброс прежних конфигураций. Код функции сброса конфигураций модуля нормализации представлен ниже.

```
void normalize_config_flush(){
    unsigned i;
    DEBUG_NORM("\n normalize_config_flush...");
    memset(&m_cfg, 0, sizeof(m_cfg));
    memset(&m_stat, 0, sizeof(m_stat));
    for(i = 0; i < CONFIG_MAX_SA; i++) {
        memset(&m_queue_thread[i], 0, sizeof(struct
queue_thread));
        m_queue_thread[i].num_sa = i;
        timer_stop(&timer_norm[i]);
    }
```

Следующим шагом в работе модуля является конфигурирование. Конфигурирование состоит из следующих функций:

— задание параметров нормализации;

— задание конфигурации потоков внутри туннеля;

— задание параметров потока на выходе устройства.

Сначала задаются параметры нормализации трафика. Параметр time - время поддерживания потока (сек), параметр Vmax - максимально допустимая скорость исходящего трафика. Код функции задания параметров нормализации представлен ниже:

```
void set_normalize_cfg(unsigned time, unsigned Vmax)
{
    DEBUG_NORM("\n set_normalize_cfg: time = %d, Vmax =
%d", time, Vmax);
```

```
    m_cfg.Time = time*1000;
    m_cfg.Vmax = Vmax;
    if(time == 0) {
        DEBUG_NORM("\n set_normalize_cfg: Error time ==
0");
        m_cfg.Time = NORMALIZE_TIME_THREAD;
    }
    if((m_cfg.Vmax == 0) || (m_cfg.Vmax >
NORMALIZE_MAX_SPEED)) {
        m_cfg.Vmax = NORMALIZE_MAX_SPEED;
    }
    m_cfg.Vmax = m_cfg.Vmax/8;
    return;
}
```

При создании потоков в туннеле необходимо указать их конфигурации. Задание конфигурации потоков внутри туннеля и на выходе устройства в виде кода представлено ниже.

```
void set_normalize_sa_thread(unsigned num_sa,  unsigned
mtu, unsigned Vmax, unsigned Vmin)
{
    ASSERT(num_sa < CONFIG_MAX_SA, "\n
set_normalize_sa_thread: ASSERT num_sa == %d", num_sa);
    m_queue_thread[num_sa].Vmax = 0;
    m_queue_thread[num_sa].Vmin = 0;
    m_queue_thread[num_sa].Vmax_ext = 0;
    m_queue_thread[num_sa].Vmin_ext = 0;
    m_queue_thread[num_sa].Vmax_thread = Vmax/8;
    m_queue_thread[num_sa].Vmin_thread = Vmin/8;
    if((Vmax == 0) && (Vmin == 0)){
        m_queue_thread[num_sa].init_thread = 0;
```

```
        return;
    }
    if((m_queue_thread[num_sa].Vmax_thread == 0) ||
(m_queue_thread[num_sa].Vmax_thread > m_cfg.Vmax))
        m_queue_thread[num_sa].Vmax_thread = m_cfg.Vmax;
    if(m_queue_thread[num_sa].Vmin_thread == 0)
        m_queue_thread[num_sa].Vmin_thread =
NORMALIZE_MIN_SPEED/8;
    if(m_queue_thread[num_sa].Vmin_thread >
m_queue_thread[num_sa].Vmax_thread)
        m_queue_thread[num_sa].Vmin_thread =
m_queue_thread[num_sa].Vmax_thread;
    m_queue_thread[num_sa].Vmax += Vmax;
    m_queue_thread[num_sa].Vmin += Vmin;
    m_queue_thread[num_sa].mtu_thread = mtu;
    if(m_queue_thread[num_sa].mtu_thread < MIN_MTU)
m_queue_thread[num_sa].mtu_thread = MIN_MTU;
    if(m_queue_thread[num_sa].mtu_thread > MAX_MTU)
m_queue_thread[num_sa].mtu_thread = MAX_MTU;
    m_queue_thread[num_sa].Vpkt_max_thread =
m_queue_thread[num_sa].Vmax_thread/mtu;
    m_queue_thread[num_sa].Vpkt_min_thread =
m_queue_thread[num_sa].Vmin_thread/mtu;
    m_queue_thread[num_sa].Vpkt_max +=
m_queue_thread[num_sa].Vpkt_max_thread;
    m_queue_thread[num_sa].Vpkt_min +=
m_queue_thread[num_sa].Vpkt_min_thread;
```

Далее задаем параметры потока на выходе устройства.

```
    m_queue_thread[num_sa].Vmax_thread_ext =
```

```
m_queue_thread[num_sa].Vmax_thread*(mtu+EXTERN_HEADER_PKT
)/mtu;
    m_queue_thread[num_sa].Vmin_thread_ext =
m_queue_thread[num_sa].Vmin_thread*(mtu+EXTERN_HEADER_PKT
)/mtu;
    m_queue_thread[num_sa].Vmax_ext +=
m_queue_thread[num_sa].Vmax_thread_ext;
    m_queue_thread[num_sa].Vmin_ext +=
m_queue_thread[num_sa].Vmin_thread_ext;
    m_queue_thread[num_sa].init_thread = 1;
}
```

Функции приема пакета в модуль противодействия скрытым каналам и контроль очереди потоков выполняются модулем параллельно.

Функция контроля очереди потоков начинает выполняться с проверки наличия потока в очереди. При наличии потока в очереди проверяется условие необходимости извлечения пакета из очереди. При неуспешной проверке данного условия ожидается увеличение системного времени. В случае успеха проверяется наличие пакета в очереди. Если в очереди есть пакет, то вызывается функкция извлечения пакета для нормализации из потока. Данная функция извлекает пакет и выравнивает его до фиксированной длины mtu, в случае отсутствия пакета, формируется нулевой пакет.

Текст функции контроля очереди потоков представлен ниже.

```
void control_thread(struct task_handler* ctx){
        struct   queue_thread   *m_queue   =   (struct
queue_thread *)ctx->private_data;
        struct thread *thrd;
        struct net_pkt *pkt = 0;
        unsigned    long    long    current_time    =
core_clock_ticks();
    //int k = m_queue->current_prio+1;
```

```
    if(m_queue->size_queue_thread == 0) return;
    if(m_queue->current == 0) m_queue->current = m_queue-
>next;
    pkt = get_pkt_normalize_tread(m_queue->current);
    if(!pkt) {
// если пакета нет, то необходимо дождаться увеличения
системного времени
        task_post(ctx);
        //timer_start(&timer_norm[m_queue->num_sa], 0);
        return;
    }
    DEBUG_NORM("\n control_thread: pktm_refcnt = %d",
pkt->refcnt);
    ASSERT(pkt->refcnt == 1,"\n control_thread:
pktm_refcnt = %d", pkt->refcnt);
//если время жизни потока истекло
    if(current_time > m_queue->current->time_dead) {
//если в очереди потока нет пакетов, то поток удаляется
      if(pkt->pkt_payload_type == EMPTY_PAYLOAD_DATA) {
            thrd = m_queue->current;
            DEBUG_NORM("\n delete thrd ip = %x", thrd-
>zas_ip);
            m_queue->current = thrd->next_ln;
            delete_thread_queue(thrd);
            update_time_ctrl();
            net_pkt_release(pkt);
            return;
        }
        else{    // продление времени жизни потока
    m_queue->current->time_dead = current_time +
```

```
m_cfg.Time;
        }
    }
    if(pkt->pkt_payload_type == EMPTY_PAYLOAD_DATA){
        aks_tun_output_empty_packet(pkt,          m_queue-
>current->sel);
        net_pkt_release(pkt);
        m_stat.null_pkt[0]++;
    }
    else{
        aks_tun_output_packet(pkt);
        net_pkt_release(pkt);
        m_stat.good_pkt[0]++;
    }
    m_stat.all_pkt[0]++;
    m_queue->current = m_queue->current->next_ln;
    task_post(ctx);
    return;
}
```

Из данной функции вызывается функция извлечения пакета для нормализации из потока get_pkt_normalize_tread(). С ее помощью производится извлечение пакета из очереди и выравнивание его до фиксированной величины mtu.

Код функции представлен ниже.

```
struct  net_pkt*  get_pkt_normalize_tread(struct  thread
*thrd){
     struct net_pkt* pkt = 0;
     unsigned long long current_time;
     unsigned mtu;
     current_time = core_clock_ticks_mksec();
```

```
// если необходимо извлечь пакет из очереди
     if((current_time <= thrd->time_call) ||
        (m_queue_thread[thrd->num_sa].Tcurrent   ==   0))
return 0;
     thrd->time_call       +=       m_queue_thread[thrd-
>num_sa].Tcurrent;
     mtu = m_queue_thread[thrd→num_sa].mtu_thread;
// если в очереди есть пакет
     if(zas_prio_queue_empty(&thrd->pkt_queue)){
         DEBUG_NORM("\n  get_pkt_normalize_tread:  create
null pkt");
         pkt = net_pkt_create(mtu);
         if (pkt) {
             //DEBUG_NORM("\n control_thread NULL pkt-len
= %d", m_queue->mtu);
            memset(net_pkt_add_tail(pkt, mtu), 0, mtu);
            pkt->pkt_payload_type = EMPTY_PAYLOAD_DATA;
            pkt->priority = 1;    // низкий приоритет
         }
     }
     else{
// извлечение пакета и выравнивание его до mtu
         struct ipv4_hdr *iph;
         pkt = zas_prio_queue_get(&thrd->pkt_queue);
         ASSERT(pkt  !=  0,  "  get_pkt_normalize_tread:
ASSERT pkt == %x", (unsigned) pkt);
         DEBUG_NORM("\n get_pkt_normalize_tread: get good
pkt");
         pkt->pkt_payload_type = IP_PAYLOAD_DATA;
// удаление «хвоста»
```

```
        iph = (struct ipv4_hdr*)(pkt->head);
        if(ntohs(iph->tot_len) < net_pkt_size(pkt)) {
            net_pkt_cut_tail(pkt,  net_pkt_size(pkt)  -
ntohs(iph->tot_len));
        }
        if(mtu > net_pkt_size(pkt))
            memset(net_pkt_add_tail(pkt,     (mtu     -
net_pkt_size(pkt))), 0, (mtu - net_pkt_size(pkt)));
     }
     return pkt;
}
```

При условии, что увеличенный функцией пакет не является полезным, происходит удаление потока из очереди потоков посредством функции void delete_thread_queue(). Текст функции представлен ниже.

```
void delete_thread_queue(struct thread *thrd)
{
        unsigned int num_sa = thrd->num_sa;
    unsigned i;
    DEBUG_NORM("\n  delete_thread_queue  ip  =  %x  sa%d",
thrd->zas_ip, num_sa);
        m_queue_thread[num_sa].size_queue_thread--;
//сброс флагов достижения предельного количества потоков
во всех очередях
    for(i = 0; i < CONFIG_MAX_SA; i++) {
        m_queue_thread[i].flg_max_thread = 0;
    }
        // удаление записи из очереди
        if(thrd->next_ln) {
            thrd->next_ln->prev_ln = thrd->prev_ln;
        }
```

```
        if(thrd->prev_ln) {
            thrd->prev_ln->next_ln = thrd->next_ln;
        }
        else    m_queue_thread[num_sa].next    =    thrd-
>next_ln;
//очистка очереди пакетов
    zas_prio_queue_flush(&thrd->pkt_queue);
// добавление в очередь свободных записей
        set_free_thread(thrd);
        return;
}
```

Выравнивание межпакетных интервалов производится посредством корректировки запуска таймеров для очереди потоков. Из главной функции вызывается функция корректировки запуска таймеров для соответствующей очереди потоков. С помощью данной функции происходит корректировка скорости в потоке и вычисляется интервал отправки пакетов в микросекундах.

```
unsigned update_time_ctrl(){
        unsigned i, Vsum_max = 0, Vsum_min = 0, Vpkt_max
= 0, Vpkt_min = 0;
        unsigned k;  // поправочный коэффециент
    DEBUG_NORM("\n update_time_ctrl...");
// расчет суммарной скорости потоков на выходе устройства
        for(i = 0; i < CONFIG_MAX_SA; i++) {
            Vsum_max  +=  (m_queue_thread[i].Vmax_ext  *
m_queue_thread[i].size_queue_thread);
            Vsum_min  +=  (m_queue_thread[i].Vmin_ext  *
m_queue_thread[i].size_queue_thread);
        Vpkt_max    +=    m_queue_thread[i].Vpkt_max    *
m_queue_thread[i].size_queue_thread;
        Vpkt_min    +=    m_queue_thread[i].Vpkt_min    *
```

```
m_queue_thread[i].size_queue_thread;
         }
         DEBUG_NORM("\n Vsum_min = %d, Vsum_max = %d "
               "Vpkt_max = %d Vpkt_min = %d",
               Vsum_min, Vsum_max,
               Vpkt_max, Vpkt_min);
         if((Vsum_min  >  m_cfg.Vmax)  ||  (Vpkt_min  >
NORMALIZE_MAX_SPEED_PKT)) {
             DEBUG_NORM("\n Vsum_min > m_cfg.Vmax(%d) ||
Vpkt_min > %d",
                m_cfg.Vmax, NORMALIZE_MAX_SPEED_PKT);
             return 1;
         }
// корректировка побайтной скорости
    if(Vsum_max > m_cfg.Vmax){
        k = (unsigned)((((__u64)(m_cfg.Vmax - Vsum_min))
* 1000)/(Vsum_max - Vsum_min));   // расчет поправочного
коэффициента
    DEBUG_NORM("\n Vsum_max = %d,> Vmax = %d, k = %d ",
               Vsum_max, m_cfg.Vmax, k);
        Vpkt_max = 0;
        for(i = 0; i < CONFIG_MAX_SA; i++){
            if(m_queue_thread[i].size_queue_thread) {
                m_queue_thread[i].Vcurrent                =
m_queue_thread[i].Vmin   +   ((m_queue_thread[i].Vmax   -
m_queue_thread[i].Vmin)*k)/1000;
                m_queue_thread[i].Vcurrent_thread         =
m_queue_thread[i].Vmin_thread +
    ((m_queue_thread[i].Vmax_thread                       -
m_queue_thread[i].Vmin_thread)*k)/1000;
```

```
                m_queue_thread[i].Vpkt_current_thread   =
m_queue_thread[i].Vcurrent_thread/m_queue_thread[i].mtu_t
hread;
     Vpkt_max += m_queue_thread[i].Vpkt_current_thread;
            }
        }
    }
    else{
        for(i = 0; i < CONFIG_MAX_SA; i++){
            if(m_queue_thread[i].size_queue_thread) {
                m_queue_thread[i].Vpkt_current_thread   =
m_queue_thread[i].Vpkt_max_thread;
                m_queue_thread[i].Vcurrent_thread       =
m_queue_thread[i].Vmax_thread;
            }
        }
    }
// корректировка пакетной скорости
    if(Vpkt_max > NORMALIZE_MAX_SPEED_PKT){
        k = (unsigned)((((__u64)NORMALIZE_MAX_SPEED_PKT -
Vpkt_min) * 1000)/(Vpkt_max - Vpkt_min));  //      расчет
поправочного коффециента
        DEBUG_NORM("\n Vsum_pkt_max = %d,> %d, k = %d ",
               Vpkt_max, NORMALIZE_MAX_SPEED_PKT, k);
        for(i = 0; i < CONFIG_MAX_SA; i++){
            if(m_queue_thread[i].size_queue_thread) {
                m_queue_thread[i].Vpkt_current_thread   =
m_queue_thread[i].Vpkt_min_thread                       +
((m_queue_thread[i].Vpkt_current_thread                 -
m_queue_thread[i].Vpkt_min_thread)*k)/1000;
```

```
                m_queue_thread[i].Vcurrent_thread       =
m_queue_thread[i].Vpkt_current_thread*m_queue_thread[i].m
tu_thread;
            }
        }
    }
// вычисление интервала отправки пакетов в микросекундах.
    for(i = 0; i < CONFIG_MAX_SA; i++){
        if(m_queue_thread[i].size_queue_thread){
            DEBUG_NORM("\n  update_time_ctrl:  sa  =  %d
cnt_thrd = %d  ",
                    i,
m_queue_thread[i].size_queue_thread);
            if(m_queue_thread[i].Vcurrent_thread) {
                m_queue_thread[i].Tcurrent              =
1000000*m_queue_thread[i].mtu_thread/m_queue_thread[i].Vc
urrent_thread;
            }
            else m_queue_thread[i].Tcurrent = 0;
            task_post(&task_norm[i]);
        }
    }
    DEBUG_NORM("\n update_time_ctrl: OK  ");
         return 0;
}
```

Функция приема пакета в модуль противодействия скрытым каналам начинает выполняться с приема пакета функцией и проверки правильности ссылки на поток нормализации. После происходит проверка условия нахождения потока. Если поток найден, то происходит добавление нового потока в очередь, в противном случае проверяется условие успешного

добавления найденного потока в очередь. В случае успешного добавления завершается выполнение данной функции, при неуспешном добавлении функция выполняется с проверки правильности ссылки на поток нормализации.

```
void normalize_output_pkt(struct net_pkt* pkt){
    struct ipv4_route_entry* rt = (struct ipv4_route_entry*)(pkt->dst);
    struct thread *thrd;
    struct aks_selector* sel;
    if(!rt) return;
    if(!rt->m_ipsec_ptr) return; // если маршрут не найден
    sel = (struct aks_selector*) rt->m_ipsec_ptr;
    thrd = (struct thread *)sel->m_thread_normalize;
    // проверка правильности потока
    if(thrd) {
        if(thrd->zas_ip != sel->gw.s_addr){
            sel->m_thread_normalize = 0;
            thrd = 0;
        }
    }
    // DEBUG_NORM("\n output ip dst = %x  ", dst.s_addr);
    // Если поток не найден
    if(!thrd) {
        thrd = get_free_thread();
        if(thrd == 0){
            DEBUG_NORM("\n Warning get_thread: Error create thread ");
            m_stat.add_err[0]++;
            return;
        }
```

```
        // инициализация потока
        thrd->zas_ip = sel->gw.s_addr;
     thrd->time_dead = core_clock_ticks() + m_cfg.Time;
        thrd->num_sa = sel->profile;
        sel->m_thread_normalize = thrd;
        thrd->sel = sel;
        // добавление потока в очередь
        if(add_thread_queue(thrd)) // если поток в очередь
не добавлен
        {
            set_free_thread(thrd);
            m_stat.add_err[0]++;
            return;
        }
// инициализация начального значения времени отправки
        if(thrd->next_ln) {
            thrd->time_call = thrd->next_ln->time_call;
        }
    }
    // разбиение пакета и добавление в очередь потока
    add_pkt_normalize_thread(thrd, pkt);
    return;
}
```

После проверки правильности потока проверяется условие нахождения потока. Если поток найден, то выполняется функция добавления потока в очередь.

Текст функции добавления потока в очередь потоков представлен ниже.

```
unsigned add_thread_queue(struct thread *thrd){
         unsigned int num_sa = thrd->num_sa;
         DEBUG_NORM("\n add_thread_queue: ip = %x sa = %d
```

```
time_dead = %d", thrd->zas_ip, num_sa, thrd->time_dead);
    if(m_queue_thread[num_sa].flg_max_thread) return 1;
         // произвести перерасчет запуска таймеров
         m_queue_thread[num_sa].size_queue_thread++;
         if(update_time_ctrl()){
// если перерасчет завершился ошибкой, то поток в очередь
не добавляется
         m_queue_thread[num_sa].size_queue_thread--;
        m_queue_thread[num_sa].flg_max_thread = 1;
            return 1;
         }
    // добавление в начало очереди
    thrd->next_ln = m_queue_thread[num_sa].next;
         thrd->prev_ln = 0;
         if(thrd->next_ln) {
            thrd->next_ln->prev_ln = thrd;
         }
         m_queue_thread[num_sa].next = thrd;
         return 0;
}
```

В случае успешного добавления нового потока в очередь производится добавление пакета в очередь потока нормализации с помощью соответствующей функции.

Текст функции добавления пакета в очередь потока нормализации представлен ниже.

```
void add_pkt_normalize_thread(struct thread *thrd, struct
net_pkt* pkt){
    unsigned num_sa = thrd->num_sa;
    DEBUG_NORM("\n add_pkt_normalize_thread: ... body_len
= %d ref = %d ", body_len, pkt->refcnt);
```

```
    if(!m_queue_thread[num_sa].init_thread) {
        m_stat.dpopped_pkt[0]++;
        return;
    }
    DEBUG_NORM("\n  add_pkt_normalize_thread:  num_thrd  =
%d          (mtu          =          %d)",          num_thrd,
m_queue_thread[num_sa].mtu_thread);
    // добавление пакета в очередь потока с разбиением
его по частям
    if(put_fragment(pkt,                &thrd->pkt_queue,
m_queue_thread[num_sa].mtu_thread) == false)
        m_stat.dpopped_pkt[0]++;
    m_stat.input_pkt[0]++;
    DEBUG_NORM("\n add_pkt_normalize_thread end");
    return;
}
```

Из этой функции вызывается функция разбиения пакета с постановкой фрагментов в очередь put_fragment(). Данная функция проверяет необходима ли фрагментация, разбивает пакет на фрагменты, формирует пакет с фрагментом, дублирует заголовок, копирует фрагменты данных, обновляет некоторые поля заголовков и отправляет фрагмент в сеть. После выполнения функция осуществляет переход к следующему пакету.

То есть, посредством данной функции осуществляется разбиение пакета с постановкой фрагментов в очередь на нормализацию.

```
bool    put_fragment(struct    net_pkt*    pkt,    struct
zas_prio_queue  *pkt_queue, unsigned mtu) {
         struct  ipv4_hdr*  iph  =  (struct  ipv4_hdr*)(pkt-
>nh);
         const int hlen = iph->ihl << 2;
         const int pdu = mtu - hlen;
```

```
/*
* Бит MF исходного пакета дублируется во всех фрагментах
* (учет повторной фрагментации фрагмента)
         */
         const int old_mf = ntohs(iph->frag_off) &
IPV4_HDR_MF;
             /*
         * Смещение относительно исходного пакета
         * (учет повторной фрагментации фрагмента)
         */
         int offset = (ntohs(iph->frag_off) &
IPV4_HDR_OFFSET) << 3;
         int copy_offset = 0;
         int left = net_pkt_size(pkt) - hlen;
        if(left <= pdu) { // Если разбиение не требуется
          if(zas_prio_queue_put_data_pkt(pkt_queue,pkt )
== -1){
             if(pkt_queue->m_max_size == pkt_queue-
>m_size[3]) {
                    DEBUG_NORM("\n put_fragment dropped 1
queue size = %d,  max = %d mtu = %d",
                               pkt_queue->m_size[3],
                               pkt_queue->m_max_size, mtu);
                }
                return false;
            }
            return true;
        }
        /* Необходима фрагментация */
        if ((iph->frag_off & ntohs(IPV4_HDR_DF))) {
```

```
            icmpv4_send(pkt,
                        ICMPV4_TYPE_DEST_UNREACH,
                        ICMPV4_CODE_FRAG_NEEDED,
                        htonl(mtu));
            return false;
        }
            /* Разбиение пакета */
            while (left > 0) {
            int len = left < pdu ? left : pdu;
/* Все фрагменты, кроме последнего, кратны 8-ми */
            if (len < left) {
                len &= ~7;
            }
            struct       net_pkt*       frag_pkt       =
net_pkt_create(hlen + len);
            if (!frag_pkt) {
                return false;
            }
            /* Формирование пакета с фрагментом */
            frag_pkt->indev = pkt->indev;
            frag_pkt->dst = pkt->dst;
            if(pkt->dst) net_dst_addref(pkt->dst);

            /* Дублирование заголовка */
            frag_pkt->nh   =   net_pkt_add_tail(frag_pkt,
hlen);
            memcpy(frag_pkt->nh, pkt->nh, hlen);

            /* Копирование фрагмента данных */
            memcpy(net_pkt_add_tail(frag_pkt,      len),
```

```
((char*)pkt->nh + hlen) + copy_offset, len);

            /* Обновление некоторых полей заголовка */
            iph = (struct ipv4_hdr*)(frag_pkt->nh);
            iph->frag_off = htons((offset >> 3));
            if (left > len || old_mf) {
            // для теста таймера сборки if (left > 0 ||
old_mf) {
            iph->frag_off |= htons(IPV4_HDR_MF);
            }
            iph->tot_len = htons(hlen + len);
            iph->check = 0;
            iph->check = ipv4_hdr_csum(iph, iph->ihl);

            /* Отправка фрагмента в сеть */
     if(zas_prio_queue_put_data_pkt(pkt_queue,  frag_pkt)
== -1){
                DEBUG_NORM("\n  put_fragment  dropped  1
queue size = %d,%d,%d,%d,  max = %d ",
                           pkt_queue->m_size[0],
                           pkt_queue->m_size[1],
                           pkt_queue->m_size[2],
                           pkt_queue->m_size[3],
                           pkt_queue->m_max_size);
                return false;
            }
                net_pkt_release(frag_pkt);
                /* Переход к следующему фрагменту */
            offset += len;
            copy_offset += len;
```

```
            left -= len;
           m_stat.frag_pkt[0]++;
            }
            return true;
}
```

Полный исходный код программного модуля противодействия скрытым каналам представлен в приложении Б.

4.4 Отладка программного модуля противодействия скрытым каналам

В результате отладки разрабатываемого программного модуля противодействия скрытым каналам были успешно исправлены найденные ошибки и подтверждена его работоспособность.

5 Тестирование программного обеспечения модуля противодействия скрытым каналам

5.1 Описание процесса тестирования программного модуля

После отладки программного обеспечения модуля противодействия скрытым каналам необходимо произвести его тестирование. В результате тестирования должны быть получены оценки влияния работы модуля на эффективную полосу пропускания полезных данных в канале связи.

Для тестирования была собрана схема тестирования, которая состоит из двух персональных электронно-вычислительных машин (ПЭВМ) клиента и сервера, ПЭВМ мониторинга и двух устройств шифрования IP-трафика со встроенными модулями противодействия скрытым каналам. Схема тестирования представлена на рисунке 7.

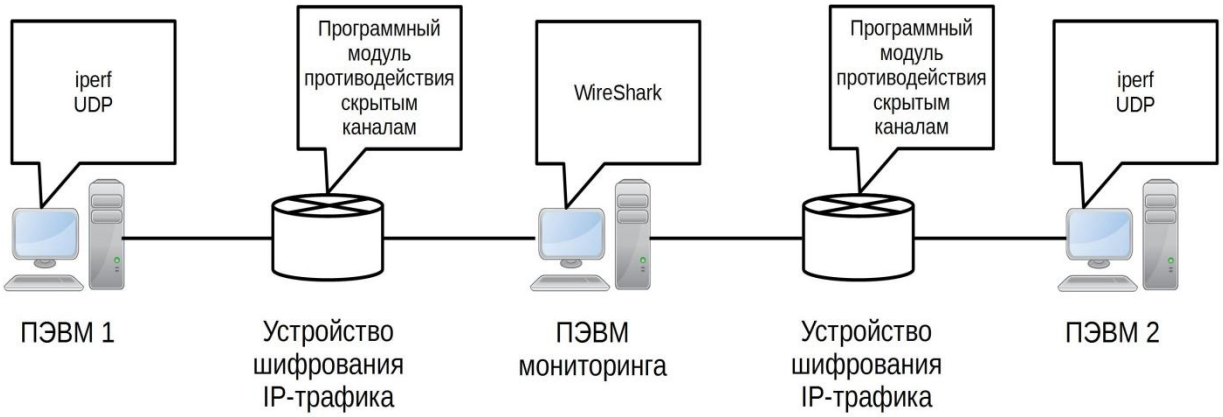


Рисунок 7 — Схема тестирования программного модуля противодействия скрытым каналам

На схеме (см. рисунок 7) ПЭВМ 1 (сервер) — это персональный компьютер (ПК), к Ethernet-порту которого подключено устройство шифрования IP-трафика. На ПЭВМ 1 запущена утилита iperf, которая задает максимальный размер пакета и размер буфера порта.

На рисунке 8 представлены параметры серверной части утилиты iperf.

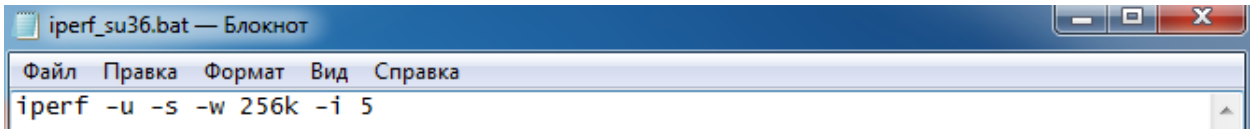


Рисунок 8 — Параметры серверной части утилиты iperf

ПЭВМ 2 (клиент) — это ПК, к Ethernet-порту которого подключено устройство шифрования IP-трафика. На ПЭВМ 2 запущена программа настройки модуля противодействия скрытым каналам и утилита iperf, задающая: адрес сервера, размер буфера порта, длину пакета, скорость передачи пакетов (бит/с), время передачи, интервал времени, количество потоков.

На рисунке 9 представлены параметры клиентской части утилиты iperf.

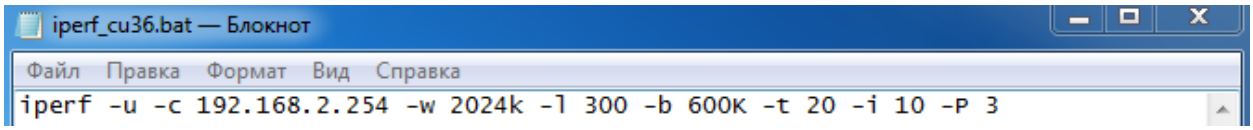


Рисунок 9 — Параметры клиентской части утилиты iperf

ПЭВМ мониторинга представляет собой ПК, предназначенный для отслеживания передаваемого во внешнюю сеть трафика посредством Wireshark. Данная ПЭВМ имеет две сетевые карты Ethernet, соединенные в мост (Bridge). ПЭВМ мониторинга работает под управлением Windows 7.

В модуле противодействия скрытым каналам могут задаваться следующие параметры:

— параметр включения/выключения нормализации;
— максимальная скорость внешнего трафика;
— время поддержания интенсивности потока;
— максимальный размер пакета.

5.2 Тестирование модуля и оценка влияния работы модуля противодействия скрытым каналам на эффективную полосу пропускания полезных данных в канале связи

Процесс тестирования программного модуля противодействия скрытым каналам заключается в проведении ряда опытов, связанных с варьированием

параметров нормализации и определении влияния работы модуля на эффективную полосу пропускания полезных данных в канале связи.

Первый этап тестирования состоит в сборке схемы тестирования и загрузки программного обеспечения модуля противодействия скрытым каналам в программно-аппаратное устройство шифрования IP-трафика.

На втором этапе были заданы необходимые параметры модуля шифрования и настроена ПЭВМ мониторинга.

Третий этап включал в себя задание параметров нормализации трафика модулем противодействия скрытым каналам и запуск программного обеспечения мониторинга.

Для оценки влияния работы модуля на эффективную полосу пропускания полезных данных в канале связи были заданы параметры тестирования программного модуля в соответствии с таблицей 1.

Таблица 1 — Параметры тестирования программного модуля

Номер теста	Статус нормализации	Размер входящих пакетов, байт	Размер нормализованных пакетов (P_size), байт
1	выключена	200	—
2		1000	
3		200 и 1000	
4	включена	200	1400
		1000	
		200 и 1000	
5		200	1000
		1000	
		200 и 1000	
6		200	600
		1000	
		200 и 1000	
7	включена	200	300
		1000	
		200 и 1000	

Далее были проведены тесты при включенной и выключенной нормализации, однородном или разнородном трафике, при изменении длин передаваемых пакетов. Значение параметра выравнивания длины пакетов (P_size) при включенной нормализации изменялось от значения по умолчанию (1400 байт) до значения близкого к размеру входящих пакетов в модуль нормализации (300 байт).

Внешняя полоса пропускания Vmax и максимальная скорость на направлении Vn_max были ограничены 2 Мбит/сек. Скорость внешнего трафика при включенной нормализации фиксирована и ограничена Vmax, а при выключенной нормализации ограничивалась скорость генерируемого трафика Vn_max 2 Мбит/с и оценивалось их отношение.

При проведении первого теста источник из внутренней сети, генерирует пакеты одинаковой длины MTU равной 200 байт для отправки через шифратор, с максимально возможной интенсивностью. Во втором тесте P_size берется равной 1000 байт, в третьем тесте генерируется разнородный трафик с длиной пакетов 200 и 1000 байт. С ПЭВМ мониторинга была получена скорость передачи во внутреннем канале Vn_max. Скорость передачи во внешнем канале Vmax фиксирована и имеет значение 2 Мбит/с.

В результате проведения первых трех тестов можно построить график, демонстрирующий влияние работы модуля противодействия скрытым каналам на эффективную полосу пропускания полезных данных в канале связи при выключенной нормализации и различных длинах генерируемых пакетов.

По оси абсцисс откладывается размер генерируемого пакета, а по оси ординат — отношение скорости передачи во внутреннем канале к скорости передачи во внешнем канале. Параметры при выключенной нормализации представлены в таблице 2.

Таблица 2 — Оцениваемые параметры при выключенной нормализации

Размер входящих пакетов, байт	Vn_max, Мбит/с	Vmax, Мбит/с	Vn_max/Vmax, %
200	1321205	2097152	63
200 — 80% 100 — 20%	1447034		69
200 — 20% 100 — 80%	1740636		83
1000	1908408		91

По данным из таблицы был построен график, отражающий влияние работы модуля на эффективную полосу пропускания полезных данных в канале связи при выключенной нормализации. Построенный график представлен на рисунке 10.

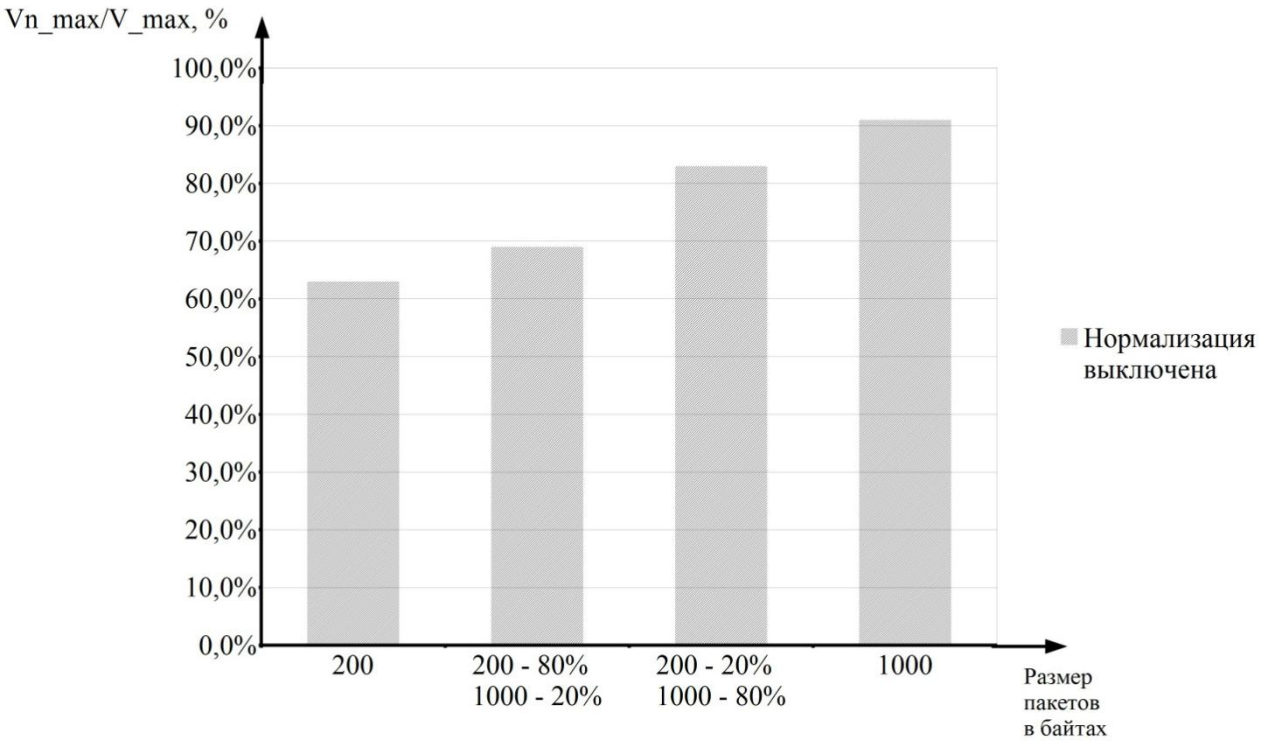


Рисунок 10 — Эффективная полоса пропускания полезных данных в канале связи при отключенном модуле нормализации

На графике (см. рисунок 10) видно, что эффективная полоса пропускания полезных данных в канале связи при отключенной нормализации снижается. Это связано с внесением избыточности. При длине генерируемых пакетов 200

байт избыточность составит 20 процентов из-за добавления заголовков. При 80 процентах трафика с длиной пакетов 200 байт и 20 процентах с длиной 1000 байт, избыточность близка к значению избыточности при генерации пакетов длиной 200 байт. При 20 процентах трафика с длиной 200 байт и 80 процентах пакетов с длиной 1000 байт избыточность уже будет близка к значению избыточности при генерации пакетов длиной 1000 байт. При генерации только пакетов длиной 1000 байт избыточность будет иметь самое малое значение, так как соотношение вносимой избыточности к величине генерируемого пакета является наименьшим.

При проведении последующих трех тестов был и построены графики отражающий влияние работы модуля на эффективную полосу пропускания полезных данных в канале связи при включенной нормализации с длиной пакетов 1400, 1200, 600 и 300 байт. Параметры представлены в таблице 3.

Таблица 3 — Параметры тестирования программного модуля

Размер входящих пакетов, байт	Размер нормализованных пакетов (P_size), байт	Vn_max, бит/с	Vmax, бит/с	Vn_max/Vmax, %
200	1400	288000	2097152	14,4
200 — 80% 100 — 20%		518400		25,9
200 — 20% 100 — 80%		1209600		60,5
1000		1440000		72
200	1200	332800		16,6
200 — 80% 100 — 20%		599040		30
200 — 20% 100 — 80%		1397760		69,9
1000		1664000		83,2
200	600	635200		31,8
200 — 80% 100 — 20%		824960		41,2
200 — 20% 100 — 80%		1394240		69,7
1000		1588000		79,4

Продолжение таблицы 3

200	300	1164800	2097152	58,2
200 — 80% 100 — 20%		1223040		61,2
200 — 20% 100 — 80%		1397760		69,9
1000		1456000		72,8

На основе представленных данных (см. таблицу 3) были построены графики, отражающие влияние работы модуля на эффективную полосу пропускания полезных данных в канале связи при включенной нормализации. Построенные графики представлены на рисунке 11.

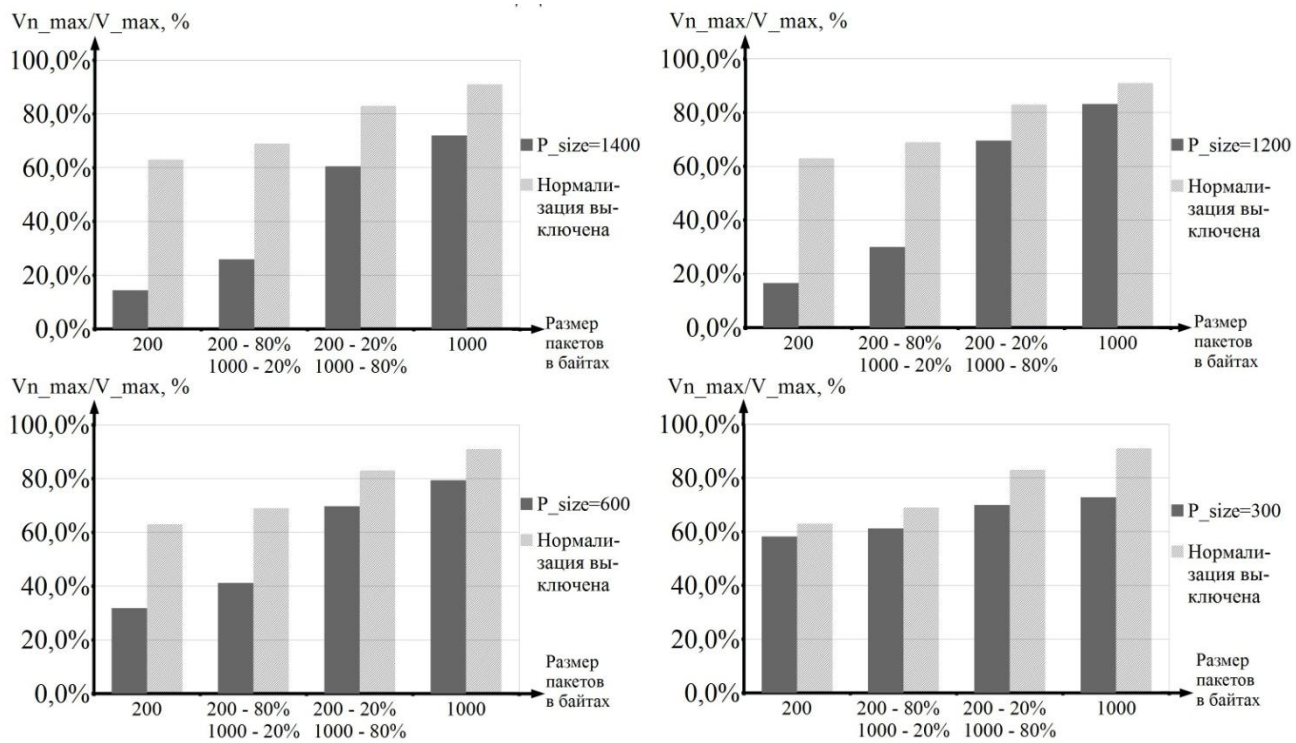


Рисунок 11 — Влияние работы модуля на эффективную полосу пропускания полезных данных в канале связи при включенной нормализации

На графиках (см. рисунок 11) видно, что максимальная эффективность работы модуля противодействия скрытым каналам достигается при длине нормализованных пакетов, которая обеспечивает минимальное внесение избыточности в передаваемые пакеты.

Для генерируемых пакетов длиной 200 байт оптимальным значением является P_size=300 байт, так как при дополнении пакетов длины 200 байт до

P_size вносит меньшую избыточность, чем, например, при дополнении пакетов длины 200 байт до длины 1400 байт.

При генерации пакетов длины 1000 байт и P_size=300, первые разбиваются на четыре и последний дополняется до длины P_size, что также вносит минимальную избыточность по сравнению с другими длинами нормализованных пакетов.

При генерации разнородного трафика длины 200 и 1000 байт длиной, при которой достигается максимальная эффективность также является P_size равное 300 байт.

Таким образом, при включенном механизме нормализации трафика все входящие пакеты дополняются до указанного размера, и соответственно чем меньше размер пакетов на входе, тем большая вносится избыточность, и тем меньше реальная полезная скорость передачи данных. Следовательно, наибольшая эффективность достигается при передачи пакетов длиной равной размеру пакета при нормализации. При превышении данного размера, начинается фрагментация каждого входящего пакета на несколько и последний дополняется до длины нормализованного пакета. Из-за этого вносится большая избыточность и наблюдается падение скорости передачи во внутреннем канале связи почти в два раза. Затем при дальнейшем увеличении длины пакетов, вносимая избыточность уменьшается и скорость увеличивается.

Однако имеется недостаток модуля нормализации. Выражающийся в том, что модуль нормализации трафика поддерживает в течении заданного времени T постоянную интенсивность передачи данных во внешнем канала, это негативно сказывается на протоколах верхнего уровня, обеспечивающих передачу данных между пользователями. К таковым относится протокол TCP, реализующий гарантированную передачу данных между узлами назначения, и управляющий интенсивностью трафика, в зависимости от качества канала связи. Протокол TCP при обнаружении потерь в канале связи, обеспечивает повторную передачу потерянных данных, а так же уменьшает интенсивность

(скорость) передачи данных [6]. Однако при включенном механизме нормализации, уменьшение скорости передачи данных вызовет увеличение скорости генерации «ложных» пакетов для поддержания постоянной скорости во внешней сети. Таким образом, уменьшение интенсивности передачи данных во внутренней сети, в случае потерь пакетов в канале связи, не уменьшит вероятность потерь пакетов. И протокол TCP при обнаружении потерь будет стремиться уменьшить скорость до минимума.

Таким образом, влияние модуля нормализации при некорректных настройках скорости передачи или при нестабильном внешнем канале связи оказывает крайне негативное влияние на протоколы транспортного уровня.

6 Вредные психофизиологические факторы, влияющие на психоэмоциональное состояние специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Производительность труда специалиста по защите информации зависит от обеспечения комфортных и безопасных условий труда.

Вредные психофизиологические факторы, влияющие на психоэмоциональное состояние специалиста по защите информации, по характеру действия подразделяются на:

— на физические перегрузки;

— нервно-психические перегрузки.

Физические перегрузки организма специалиста по защите информации характеризуются такими показателями, как:

— стереотипные рабочие движения;

— статическая нагрузка;

— рабочая поза.

Нервно-психические перегрузки подразделяют:

— на умственное перенапряжение, в том числе вызванное информационной нагрузкой;

— перенапряжение анализаторов, в том числе вызванное информационной нагрузкой;

— монотонность труда;

— эмоциональные перегрузки.

Нервно-психические перегрузки специалиста по защите информации, характеризуются следующими показателями:

— длительность сосредоточенного наблюдения;

— активное наблюдение за ходом производственного процесса;

— число производственных объектов одновременного наблюдения;

— плотность сигналов и сообщений в единицу времени;

— нагрузка на слуховой анализатор;

— нагрузка на голосовой аппарат;

— работа с оптическими приборами [7].

При работе с компьютером возникает производственная усталость.

Особую роль в снижении вредных психофизиологических факторов, влияющих на психоэмоциональное состояние специалиста по защите информации, играют:

— организация рабочего места;

— правильное рабочее положение;

— выбор ритма работы;

— рационализация трудового процесса;

— использование эмоциональных стимулов;

— внедрение рациональных режимов труда и отдыха.

Для профилактики усталости работников применяются специфические методы, к которым можно отнести средства восстановления функционального состояния зрительного и опорно-двигательного аппарата, уменьшение гиподинамии, усиления кровообращения, оптимизации мыслительной деятельности.

Значимость при работе с компьютером приобретает психоэмоциональный стресс, который в большей или меньшей степени проявляется у специалиста по защите информации.

При организации труда, связанной с использованием компьютера, для сохранения здоровья работника, предупреждения профессиональных заболеваний и поддержания работоспособности предусматриваются перерывы для отдыха.

Режимы труда и отдыха содержат дополнительные непродолжительные перерывы в периоды, предшествующие появлению объективных и субъективных признаков утомления и снижения работоспособности

При выполнении работ, связанных с использованием компьютера, которые занимают не менее 50% рабочего времени, на протяжении рабочей смены должны предусматриваться:

— перерывы для отдыха и приема пищи;

— перерывы для отдыха и личных потребностей;

— дополнительные перерывы, вводимые для отдельных профессий с учетом особенностей трудовой деятельности

При 8-часовой дневной рабочей смене в зависимости от характера труда устанавливаются режимы труда и отдыха при работе с ПЭВМ.

В день специалисту по защите информации не рекомендуется работать за компьютером более 6 часов.

Методом снижения вредных психофизиологических факторов также является проведение сеансов психофизиологической разгрузки, которые основаны на сознательном применении комплекса взаимосвязанных приемов психической саморегуляции и выполнении несложных физических упражнений со словесным самовнушением. Главное внимание при этом уделяется приобретению и закреплению навыков мышечного расслабления.

Данные сеансы, должны проводиться в комнате психофизиологической разгрузки с соответствующим интерьером и цветовым оформлением.

После сеансов психофизиологической разгрузки у работников уменьшается чувство усталости, появляются бодрость, хорошее настроение и, как следствие, воздействие вредных психофизиологических факторов снижается.

Заключение

В процессе выполнения дипломного проектирования:

— были изучены способы построения скрытых каналов, противодействие которым производится с помощью разработанного модуля, а именно:

— способ построения скрытых каналов, использующих межпакетные интервалы для скрытой передачи информации;

— способ построения скрытых каналов, использующие размер передаваемых пакетов для скрытой передачи информации;

— были изучены способы противодействия скрытым каналам, которые применяются в разрабатываемом модуле, а именно:

— способ противодействия, при котором все выходящие из модуля пакеты передаются через равные интервалы времени;

— способ противодействия, при котором обеспечивается одинаковый размер передаваемых пакетов;

— был разработан алгоритм работы программного модуля противодействия скрытым каналам;

— было реализовано на языке программирования C и отлажено программное обеспечение модуля противодействия скрытым каналам;

— разработанный модуль противодействия скрытым каналам был протестирован;

— были описаны вредные психофизиологические факторы, влияющие на психоэмоциональное состояние специалиста по защите информации, а также методы и средства физической культуры, снижающие их воздействие.

Таким образом, все пункты технического задания на выпускную квалификационную работу были выполнены в полном объеме.

Список используемых источников

1 Общие положения [Электронный ресурс]: ГОСТ Р 53113.1-2008 Информационная технология (ИТ). Защита информационных технологий и автоматизированных систем от угроз информационной безопасности, реализуемых с использованием скрытых каналов. Часть 1. Общие положения. Режим доступа: URL: http://docs.cntd.ru/document/1200075568, свободный. – Загл. с экрана.

2 Схема взаимодействия скрытого канала и автоматизированной системы [Электронный ресурс]: Матвеев С. В.: Меры защиты от скрытых каналов в автоматизированных системах и их пропускная способность при некоторых способах противодействия. Режим доступа: URL: https://journal.tusur.ru/storage/45717/074.pdf?1466682342, свободный. – Загл. с экрана.

3 Систематизация скрытых каналов [Электронный ресурс]: Скрытые каналы передачи данных. Режим доступа: URL: http://cryptowiki.net/index.php, свободный. – Загл. с экрана.

4 Примеры методов противодействия СК [Электронный ресурс]: ГОСТ Р 53113.2-2009 Информационная технология (ИТ). Защита информационных технологий и автоматизированных систем от угроз информационной безопасности, реализуемых с использованием скрытых каналов. Часть 2. Рекомендации по организации защиты информации, информационных технологий и автоматизированных систем от атак с использованием скрытых каналов Режим доступа: URL: http://docs.cntd.ru/document/1200076494, свободный. – Загл. с экрана.

5 Пропускная способность скрытых каналов [Электронный ресурс]: Об оценке пропускной способности скрытых информационных каналов, основанных на изменении длин передаваемых пакетов. Режим доступа: URL: http://www.infokosmo.ru/file/article/15403.pdf, свободный. – Загл. с экрана.

6 Сетевые технологии [Электронный ресурс]: Протокол IP/TCP/UDP. Режим доступа: URL: http://net.e-publish.ru/p241aa1.html, свободный. – Загл. с экрана.

7 ГОСТ 12.0.003-2015. Система стандартов по безопасности труда. Опасные и вредные производственные факторы. Классификация [Электронный ресурс]: Режим доступа: URL: http://www.internet-law.ru/gosts/gost/62075, свободный — Загл. с экрана.

Приложение А

(обязательное)

Алгоритмы функций программного модуля противодействия скрытым каналам передачи информации

Рисунок А.1 — Алгоритм функции приема пакета в модуль противодействия скрытым каналам

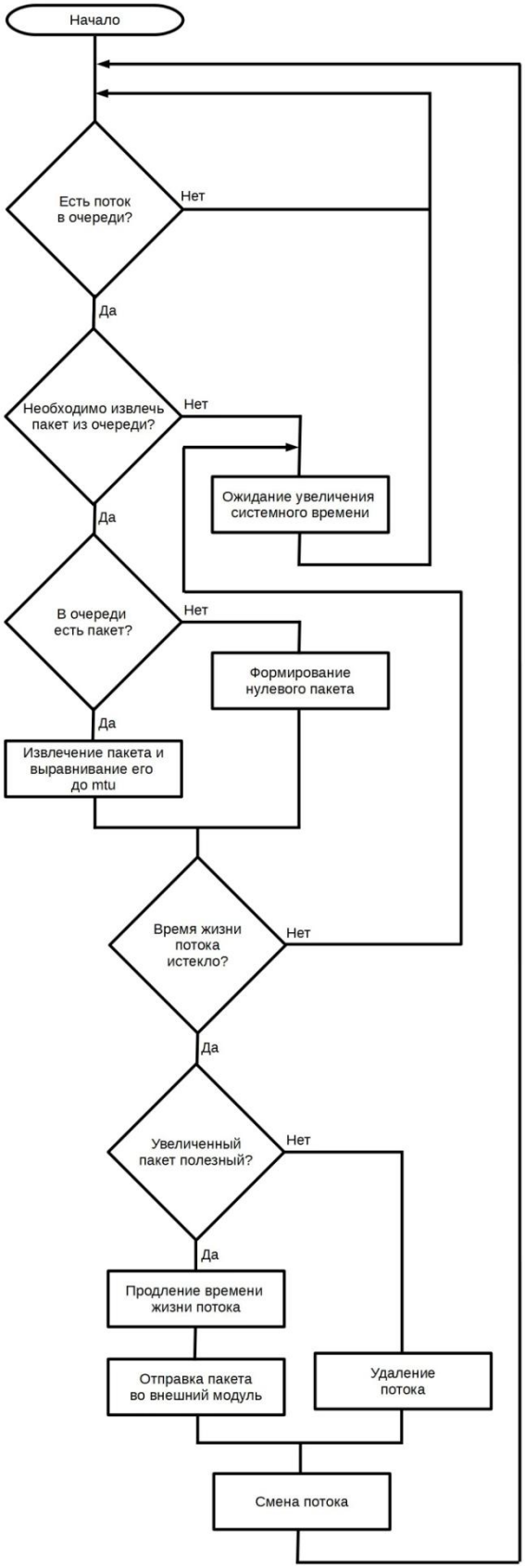


Рисунок А.2 — Алгоритм функции контроля очереди потоков

Приложение Б

(обязательное)

Тексты программного модуля противодействия скрытым каналам

Файл normalize.h

```
#ifndef NORMALIZE_H
#define NORMALIZE_H
#include <command_shell.h>
#include <net.h>
int cmd_norms_init();
void init_normalize();
// останов модуля нормализации и очистка конфигурации
void normalize_config_flush();
void set_normalize_cfg(unsigned time, unsigned Vmax); // должно
быть вызвано раньше set_normalize_sa
// задать конфигурацию потоков внутри туннеля (вызывать только
после set_normalize_sa)
void set_normalize_sa_thread(unsigned num_sa, unsigned mtu,
unsigned Vmax, unsigned Vmin);
void normalize_on();
void normalize_off();
unsigned is_normalize_on();
void normalize_output_pkt(struct net_pkt* pkt);
#endif
```

Файл normalize.c

```
#include <normalize.h>
#include <diag.h>
#include <system_config.h>
#include <core.h>
#include <aks_tun.h>
#include <net.h>
#include <ipv4.h>
#include <packet.h>
////////////////////////////////////////
//#define DEBUG
#ifdef DEBUG
    #define DEBUG_NORM(_msg_...) do { diag_printf(_msg_); } while
(0)
#else
    #define DEBUG_NORM(_msg_...) do { } while (0)
#endif
////////////////////////////////////////////////////////////////
#define EXTERN_HEADER_PKT           70        // Размер внешнего
заголовка добавляемого после зашифрования (Eth, IP, ESP....)
// максимально допустимая скорость внешенго трафика при включенной
нормализации (бит/сек)
#define NORMALIZE_MAX_SPEED        12582912
// минимально допустимая скорость внешенго трафика при включенной
нормализации (бит/сек)
#define NORMALIZE_MIN_SPEED        512
//время поддержания интенсивности потока по умолчанию
#define NORMALIZE_TIME_THREAD 30000
```

```
// максимально допустимая скорость внешенго трафика при включенной нормализации (пакетов/сек)
#define NORMALIZE_MAX_SPEED_PKT    8000    // Максимальная скорость трафика в пакетах в сек.
#define MAX_PKT_QUEUE    1024      //    максимальное    количесвто пакетов в очереди потока
struct thread;
// структура потока
struct thread{
    struct zas_prio_queue     pkt_queue;          //        Очередь пакетов отложенных на передачу
    unsigned long long time_call;  //  расчетное  время  отправки пакета из очереди
        unsigned int zas_ip;      //Адрес  удаленного  устнойства, на который организован поток
//  (адрес  противоположного  аппарата,  с  которым  устанавливаются сеансы связи)
        unsigned int time_dead;         //     время     окончания действия потока (в еденицах системного времени)
        unsigned int num_sa;       // номер профиля туннелей
        struct thread *next_ln, *prev_ln;         //        очередь потоков
    struct  aks_selector*  sel;                          // ссылка на соответсвующий туннель
};
// структура очереди потоков
struct queue_thread{
    unsigned  int  flg_max_thread;          //  флаг  достижение максимального количества потоков
        unsigned int size_queue_thread;           //     количество потоков в очереди потоков
        struct thread *next;            //  указатель  на  первый поток в очереди
        struct thread *current;              //    указатель    на текущий обрабатываемый поток
        unsigned int num_sa;            // номер профиля туннелей
    unsigned int mtu_thread;       // размер выравнивания пакетов
    unsigned int Vmax_thread;      // максимальная скорость потока (байт/сек)
    unsigned int Vmin_thread;      // минимальная скорость потока (байт/сек)
    unsigned int Vpkt_max_thread;  //     максимальная     пакетная скорость потока (пак/сек)
    unsigned int Vpkt_min_thread;  // минимальная скорость потока (пак/сек)
    unsigned int Vcurrent_thread;  //   текущая   скорость   потока (байт/сек)
    unsigned int Vpkt_current_thread;   // текущая скорость потока (пак/сек)
    unsigned  int  init_thread;                    //  количество инициализированных потоков
```

```
    /////////// Характеристики потоков на выходе из устройства с
учетом
    /////////  добавление внешнего загловка (IP+ESP)
    unsigned int Vmax_thread_ext;       //  максимальная  скорость
потока на выходе устройства(байт/сек)
    unsigned int Vmin_thread_ext;       //   минимальная   скорость
потока на выходе устройства(байт/сек)
    unsigned int Vmax_ext;              //  максимальная  скорость
потока (байт/сек)
    unsigned int Vmin_ext;              //   минимальная   скорость
потока (байт/сек)
        unsigned int Vmax;              //  максимальная  скорость
потока (байт/сек)
        unsigned int Vmin;              //   минимальная   скорость
потока (байт/сек)
    unsigned int Vpkt_max;              //  максимальная  пакетная
скорость потока в целом (пак/сек)
    unsigned int Vpkt_min;              //   минимальная   пакетная
скорость потока в целом (пак/сек)
    unsigned int Vcurrent;         // текущая скорость потока
    unsigned int Vpkt_current;          //     текущая     пакетная
скорость потока
    unsigned int Tcurrent;         // текущий расчитанный интервал
срабатывания таймера (микро сек)
        unsigned int Tmin;              //  минимальное  расчетное
время срабатывания таймера при максимальной скорости (микро сек)
        unsigned int Tmax;              //  максимальное  расчетное
время срабатывания таймера при минимальноей скорости (микро сек)
    unsigned long long save_time;   // сохраненное системное время
(для расчета приоритета)
};
////////////////////////////////////////////////////////////
struct task_handler task_norm[CONFIG_MAX_SA];     //     обработчик
очереди потоков
struct timer_handler timer_norm[CONFIG_MAX_SA];   //     обработчик
очереди потоков

struct thread m_thread[CONFIG_MAX_SESSIONS];      // буфер потоков
struct thread *free_thread;   ///  очередь  свободных  потоков  и
действующих
struct queue_thread  m_queue_thread[CONFIG_MAX_SA];         //
очереди потоков (своя очередь для каждого профеля туннеля)

//структура конфигурирования параметров нормализации трафика
struct norm_cfg{
        unsigned    Vmax;               //  максимальная  выходная
интенсивность внешнего потока (байт/сек)
        unsigned    Time;               //  гарантированное  время
поддержания интенсивности трафика на каждом открытом направлении
        unsigned  char      flg_enable;    // включение механизма
нормализации трафика (0 - выключен, 1 - включен)
} m_cfg;
```

```
struct norm_stat{
    unsigned  transfer_pkt[2];       //  пакетов  в  обход  модуля
нормализации
    unsigned input_pkt[2];   // всего пакетов поставлено в очередь
    unsigned all_pkt[2];   // всего пакетов отправлено
    unsigned good_pkt[2];  // отправлено полезных пакетов
    unsigned null_pkt[2];  // отправлено ложных пакетов
    unsigned frag_pkt[2];  // фрагментированно пакетов
    unsigned dpopped_pkt[2];  // отброшено полезных пакетов
    unsigned  transfer_queue_pkt[2];   // перемещеннных  пакетов  из
одной очереди в другую
    unsigned add_err[2];   // ошибки добавления новых потоков
    unsigned  long  long  save_time;     //  последнее  время  вывода
статистики
} m_stat;
//=============================================================
unsigned update_time_ctrl();
void delete_thread_queue(struct thread *thrd);
void control_thread(struct task_handler* ctx);
void control_thread_time(struct timer_handler* ctx);
void init_normalize(){
        unsigned i;
        DEBUG_NORM("\n init_normalize...");
        memset(&m_cfg, 0, sizeof(m_cfg));
    memset(&m_stat, 0, sizeof(m_stat));
        //-----------------------------
        for(i = 0; i < CONFIG_MAX_SA; i++) {
          memset(&m_queue_thread[i],        0,        sizeof(struct
queue_thread));
          m_queue_thread[i].num_sa = i;
        task_init(&task_norm[i],
               control_thread, (void *)&m_queue_thread[i]);
        timer_init(&timer_norm[i],
               control_thread_time, (void *)&m_queue_thread[i]);
        }
        // инициализация свободной очереди потоков
        free_thread = &m_thread[0];
        for(i = 0; i < (CONFIG_MAX_SESSIONS - 1); i++)
        {
          m_thread[i].next_ln = &m_thread[(i+1)];
        zas_prio_queue_init(&m_thread[i].pkt_queue,
MAX_PKT_QUEUE);
        }
        m_thread[i].next_ln = 0;
    normalize_config_flush();
        return;
}
void normalize_config_flush(){
    unsigned i;
    DEBUG_NORM("\n normalize_config_flush...");
    memset(&m_cfg, 0, sizeof(m_cfg));
    memset(&m_stat, 0, sizeof(m_stat));
```

```
    //----------------------------
    for(i = 0; i < CONFIG_MAX_SA; i++) {
        memset(&m_queue_thread[i],         0,         sizeof(struct
queue_thread));
        m_queue_thread[i].num_sa = i;
        timer_stop(&timer_norm[i]);
    }
    // инициализация свободной очереди потоков
    free_thread = &m_thread[0];
    for(i = 0; i < (CONFIG_MAX_SESSIONS - 1); i++)
    {
        m_thread[i].next_ln = 0;
        m_thread[i].next_ln = &m_thread[(i+1)];
        m_thread[i].zas_ip = 0;
        m_thread[i].sel = 0;
        zas_prio_queue_flush(&m_thread[i].pkt_queue);
    }
    m_thread[i].next_ln = 0;
    return;
}
// задание параметров нормализации трафика
// time - время поддреживания потока (сек)
// Vmax - максимально допустимая скорость исходящего трафика
void set_normalize_cfg(unsigned time, unsigned Vmax)
{
        DEBUG_NORM("\n set_normalize_cfg: time = %d, Vmax = %d",
time, Vmax);
        m_cfg.Time = time*1000;
        m_cfg.Vmax = Vmax;
        if(time == 0) {
          DEBUG_NORM("\n set_normalize_cfg: Error time == 0");
          m_cfg.Time = NORMALIZE_TIME_THREAD;
        }
        if((m_cfg.Vmax == 0) || (m_cfg.Vmax > NORMALIZE_MAX_SPEED))
{
          m_cfg.Vmax = NORMALIZE_MAX_SPEED;
        }
        m_cfg.Vmax = m_cfg.Vmax/8;
        return;
}
// задать конфигурацию потоков внутри туннеля
void  set_normalize_sa_thread(unsigned  num_sa,    unsigned  mtu,
unsigned Vmax, unsigned Vmin)
{
    ASSERT(num_sa  <  CONFIG_MAX_SA,  "\n  set_normalize_sa_thread:
ASSERT num_sa == %d", num_sa);
    m_queue_thread[num_sa].Vmax = 0;
    m_queue_thread[num_sa].Vmin = 0;
    m_queue_thread[num_sa].Vmax_ext = 0;
    m_queue_thread[num_sa].Vmin_ext = 0;
    m_queue_thread[num_sa].Vmax_thread = Vmax/8;
    m_queue_thread[num_sa].Vmin_thread = Vmin/8;
```

```
    if((Vmax == 0) && (Vmin == 0)){
        m_queue_thread[num_sa].init_thread = 0;
        return;
    }
    if((m_queue_thread[num_sa].Vmax_thread      ==      0)      ||
(m_queue_thread[num_sa].Vmax_thread > m_cfg.Vmax))
        m_queue_thread[num_sa].Vmax_thread = m_cfg.Vmax;
    if(m_queue_thread[num_sa].Vmin_thread == 0)
        m_queue_thread[num_sa].Vmin_thread                          =
NORMALIZE_MIN_SPEED/8;
    if(m_queue_thread[num_sa].Vmin_thread                           >
m_queue_thread[num_sa].Vmax_thread)
        m_queue_thread[num_sa].Vmin_thread                          =
m_queue_thread[num_sa].Vmax_thread;
    m_queue_thread[num_sa].Vmax += Vmax;
    m_queue_thread[num_sa].Vmin += Vmin;
    m_queue_thread[num_sa].mtu_thread = mtu;
    if(m_queue_thread[num_sa].mtu_thread        <         MIN_MTU)
m_queue_thread[num_sa].mtu_thread = MIN_MTU;
    if(m_queue_thread[num_sa].mtu_thread        >         MAX_MTU)
m_queue_thread[num_sa].mtu_thread = MAX_MTU;
    m_queue_thread[num_sa].Vpkt_max_thread                          =
m_queue_thread[num_sa].Vmax_thread/mtu;
    m_queue_thread[num_sa].Vpkt_min_thread                          =
m_queue_thread[num_sa].Vmin_thread/mtu;
    m_queue_thread[num_sa].Vpkt_max                                 +=
m_queue_thread[num_sa].Vpkt_max_thread;
    m_queue_thread[num_sa].Vpkt_min                                 +=
m_queue_thread[num_sa].Vpkt_min_thread;
    // параметры потока на выходе устройства
    m_queue_thread[num_sa].Vmax_thread_ext                          =
m_queue_thread[num_sa].Vmax_thread*(mtu+EXTERN_HEADER_PKT)/mtu;
    m_queue_thread[num_sa].Vmin_thread_ext                          =
m_queue_thread[num_sa].Vmin_thread*(mtu+EXTERN_HEADER_PKT)/mtu;
    m_queue_thread[num_sa].Vmax_ext                                 +=
m_queue_thread[num_sa].Vmax_thread_ext;
    m_queue_thread[num_sa].Vmin_ext                                 +=
m_queue_thread[num_sa].Vmin_thread_ext;
    m_queue_thread[num_sa].init_thread = 1;
}
// включение/выключение нормализации трафика
void normalize_on(){
        m_cfg.flg_enable = 1;
        DEBUG_NORM("\n normalize_on...");
}
// выключение нормализации трафика
void normalize_off(){
        unsigned i;
        DEBUG_NORM("\n normalize_off...");
        m_cfg.flg_enable = 0;
        for(i = 0; i < CONFIG_MAX_SA; i++) {
          m_queue_thread[i].current = 0;
```

```
          while(m_queue_thread[i].next){
               delete_thread_queue(m_queue_thread[i].next);
          }
        }
        update_time_ctrl();
        return;
}
// возвращает 1, если нормализация включена
unsigned is_normalize_on(){
        return m_cfg.flg_enable;
}
////////////////////////////////////////////////////////////////
//////////
// корректировка запуска таймеров для соответствующей очереди
потоков
unsigned update_time_ctrl(){
        unsigned i, Vsum_max = 0, Vsum_min = 0, Vpkt_max = 0,
Vpkt_min = 0;
        unsigned k;  // поправочный коэффециент
    DEBUG_NORM("\n update_time_ctrl...");
        // расчет суммарной скорости потоков на выходе устройства
        for(i = 0; i < CONFIG_MAX_SA; i++) {
          Vsum_max      +=      (m_queue_thread[i].Vmax_ext      *
m_queue_thread[i].size_queue_thread);
          Vsum_min      +=      (m_queue_thread[i].Vmin_ext      *
m_queue_thread[i].size_queue_thread);
        Vpkt_max       +=       m_queue_thread[i].Vpkt_max       *
m_queue_thread[i].size_queue_thread;
        Vpkt_min       +=       m_queue_thread[i].Vpkt_min       *
m_queue_thread[i].size_queue_thread;
        }
        DEBUG_NORM("\n Vsum_min = %d, Vsum_max = %d "
               "Vpkt_max = %d Vpkt_min = %d",
               Vsum_min, Vsum_max,
               Vpkt_max, Vpkt_min);
        if((Vsum_min    >    m_cfg.Vmax)    ||    (Vpkt_min    >
NORMALIZE_MAX_SPEED_PKT)) {
          DEBUG_NORM("\n Vsum_min > m_cfg.Vmax(%d) || Vpkt_min >
%d",
                    m_cfg.Vmax, NORMALIZE_MAX_SPEED_PKT);
          return 1;
        }
    // корректировка побайтной скорости
    if(Vsum_max > m_cfg.Vmax){
        k   =   (unsigned)((((__u64)(m_cfg.Vmax   -   Vsum_min))   *
1000)/(Vsum_max - Vsum_min)); // расчет поправочного коффециента
        DEBUG_NORM("\n Vsum_max = %d,> Vmax = %d, k = %d ",
               Vsum_max, m_cfg.Vmax, k);
        Vpkt_max = 0;
        for(i = 0; i < CONFIG_MAX_SA; i++){
            if(m_queue_thread[i].size_queue_thread) {
```

```
                m_queue_thread[i].Vcurrent                          =
m_queue_thread[i].Vmin      +      ((m_queue_thread[i].Vmax      -
m_queue_thread[i].Vmin)*k)/1000;
                m_queue_thread[i].Vcurrent_thread                   =
m_queue_thread[i].Vmin_thread +

((m_queue_thread[i].Vmax_thread                                     -
m_queue_thread[i].Vmin_thread)*k)/1000;
                m_queue_thread[i].Vpkt_current_thread               =
m_queue_thread[i].Vcurrent_thread/m_queue_thread[i].mtu_thread;
                Vpkt_max += m_queue_thread[i].Vpkt_current_thread;
            }
        }
    }
    else{
        for(i = 0; i < CONFIG_MAX_SA; i++){
            if(m_queue_thread[i].size_queue_thread) {
                m_queue_thread[i].Vpkt_current_thread               =
m_queue_thread[i].Vpkt_max_thread;
                m_queue_thread[i].Vcurrent_thread                   =
m_queue_thread[i].Vmax_thread;
            }
        }
    }

    //корректировка пакетной скорости скорости
    if(Vpkt_max > NORMALIZE_MAX_SPEED_PKT){
        k    =    (unsigned)((((__u64)NORMALIZE_MAX_SPEED_PKT    -
Vpkt_min) * 1000)/(Vpkt_max - Vpkt_min));    //              расчет
поправочного коффециента
        DEBUG_NORM("\n Vsum_pkt_max = %d,> %d, k = %d ",
               Vpkt_max, NORMALIZE_MAX_SPEED_PKT, k);

        for(i = 0; i < CONFIG_MAX_SA; i++){
            if(m_queue_thread[i].size_queue_thread) {
                m_queue_thread[i].Vpkt_current_thread               =
m_queue_thread[i].Vpkt_min_thread +
                    ((m_queue_thread[i].Vpkt_current_thread       -
m_queue_thread[i].Vpkt_min_thread)*k)/1000;
                m_queue_thread[i].Vcurrent_thread                   =
m_queue_thread[i].Vpkt_current_thread*m_queue_thread[i].mtu_thread
;
            }
        }
    }
    // вычисление интервала отправки пакетов в микросекундах
    for(i = 0; i < CONFIG_MAX_SA; i++){
        if(m_queue_thread[i].size_queue_thread){
            DEBUG_NORM("\n update_time_ctrl: sa = %d cnt_thrd = %d
",
                    i, m_queue_thread[i].size_queue_thread);
            if(m_queue_thread[i].Vcurrent_thread) {
```

```
                m_queue_thread[i].Tcurrent                              =
1000000*m_queue_thread[i].mtu_thread/m_queue_thread[i].Vcurrent_th
read;
            }
            else m_queue_thread[i].Tcurrent = 0;
            task_post(&task_norm[i]);
        }
    }
    DEBUG_NORM("\n update_time_ctrl: OK  ");
        return 0;
}
struct thread* get_free_thread(){
        struct thread *thrd;
        DEBUG_NORM("\n     get_free_thread:     fr     =     %x",
(__u32)free_thread);
        if(free_thread == 0) return 0;
        thrd = free_thread;
        free_thread = thrd->next_ln;
    thrd->time_call = core_clock_ticks()*1000;
        DEBUG_NORM("\n     get_free_thread:     fr2     =     %x",
(__u32)free_thread);
        return thrd;
}
void set_free_thread(struct thread *uk){
        DEBUG_NORM("\n set_free_thread: fr = %x uk = %x ip = %x",
               (__u32)free_thread, (__u32)uk, uk->zas_ip);
        uk->next_ln = free_thread;
        free_thread = uk;
        uk->zas_ip = 0;
        DEBUG_NORM("\n     set_free_thread:     fr2     =     %x",
(__u32)free_thread);
        return;
}
// добавление потока в очередь потоков
unsigned add_thread_queue(struct thread *thrd){
        unsigned int num_sa = thrd->num_sa;
        DEBUG_NORM("\n add_thread_queue: ip = %x sa = %d  time_dead
= %d", thrd->zas_ip, num_sa, thrd->time_dead);
    if(m_queue_thread[num_sa].flg_max_thread) return 1;
        // произвести перерасчет запуска таймеров
        m_queue_thread[num_sa].size_queue_thread++;
        if(update_time_ctrl()){
          // если перерасчет завершился с ошибкой, то поток в
очередь не добавляется
          m_queue_thread[num_sa].size_queue_thread--;
        m_queue_thread[num_sa].flg_max_thread = 1;
          return 1;
        }
    // добавление в начало очереди
    thrd->next_ln = m_queue_thread[num_sa].next;
        thrd->prev_ln = 0;
        if(thrd->next_ln) {
```

```
          thrd->next_ln->prev_ln = thrd;
        }
        m_queue_thread[num_sa].next = thrd;
        return 0;
}
// удаление потока из очереди потоков, очистка кэш и помещение
записи в очередь свободных записей
void delete_thread_queue(struct thread *thrd)
{
        unsigned int num_sa = thrd->num_sa;
    unsigned i;

    DEBUG_NORM("\n  delete_thread_queue  ip  =  %x  sa%d",  thrd-
>zas_ip, num_sa);
        m_queue_thread[num_sa].size_queue_thread--;
    // сбросить флаги достижения предельного количества потоков
    // во всех очередях
    for(i = 0; i < CONFIG_MAX_SA; i++) {
        m_queue_thread[i].flg_max_thread = 0;
    }
        // удаление записи из очереди
        if(thrd->next_ln) {
          thrd->next_ln->prev_ln = thrd->prev_ln;
        }
        if(thrd->prev_ln) {
          thrd->prev_ln->next_ln = thrd->next_ln;
        }
        else m_queue_thread[num_sa].next = thrd->next_ln;
    // очистка очереди пакетов
    zas_prio_queue_flush(&thrd->pkt_queue);
        // добавление в очередь свободных записей
        set_free_thread(thrd);
        return;
}
// разбиение пакета с постановкой фрамментов в очередь
bool  put_fragment(struct  net_pkt*  pkt,  struct  zas_prio_queue
*pkt_queue, unsigned mtu) {
          struct ipv4_hdr* iph = (struct ipv4_hdr*)(pkt->nh);
          const int hlen = iph->ihl << 2;
          const int pdu = mtu - hlen;
               /*
           *  Бит  MF  исходного  пакета  дублируется  во  всех
фрагментах
           * (учет повторной фрагментации фрагмента)
           */
          const int old_mf = ntohs(iph->frag_off) & IPV4_HDR_MF;

          /*
           * Смещение относительно исходного пакета
           * (учет повторной фрагментации фрагмента)
           */
```

```
          int offset = (ntohs(iph->frag_off) & IPV4_HDR_OFFSET) <<
3;
          int copy_offset = 0;
          int left = net_pkt_size(pkt) - hlen;
        if(left <= pdu) { // Если разбиение не требуется
            if(zas_prio_queue_put_data_pkt(pkt_queue,pkt ) == -1){
                if(pkt_queue->m_max_size  ==  pkt_queue->m_size[3])
{
                    DEBUG_NORM("\n  put_fragment  dropped  1  queue
size = %d,  max = %d mtu = %d",
                               pkt_queue->m_size[3],
                               pkt_queue->m_max_size, mtu);
                }
                return false;
            }
            return true;
        }
        /* Необходима фрагментация */
        if ((iph->frag_off & ntohs(IPV4_HDR_DF))) {
            icmpv4_send(pkt,
                      ICMPV4_TYPE_DEST_UNREACH,
                      ICMPV4_CODE_FRAG_NEEDED,
                      htonl(mtu));
            return false;
        }
          /* Разбиение пакета */
          while (left > 0) {
               int len = left < pdu ? left : pdu;
               /*  Все  фрагменты,  кроме  последнего,  обязательно
кратны 8-ми */
               if (len < left) {
                    len &= ~7;
               }
               struct  net_pkt*  frag_pkt  =  net_pkt_create(hlen  +
len);
               if (!frag_pkt) {
                    return false;
               }

               /* Формирование пакета с фрагментом */
               frag_pkt->indev = pkt->indev;
               frag_pkt->dst = pkt->dst;
               if(pkt->dst) net_dst_addref(pkt->dst);

               /* Дублирование заголовка */
               frag_pkt->nh = net_pkt_add_tail(frag_pkt, hlen);
               memcpy(frag_pkt->nh, pkt->nh, hlen);

               /* Копирование фрагмента данных */
               memcpy(net_pkt_add_tail(frag_pkt,              len),
((char*)pkt->nh + hlen) + copy_offset, len);
```

```
                /* Обновление некоторых полей заголовка */
                iph = (struct ipv4_hdr*)(frag_pkt->nh);
                iph->frag_off = htons((offset >> 3));
                if (left > len || old_mf) {
                     // для теста таймера сборки if (left > 0 ||
old_mf) {
                     iph->frag_off |= htons(IPV4_HDR_MF);
                }
                iph->tot_len = htons(hlen + len);
                iph->check = 0;
                iph->check = ipv4_hdr_csum(iph, iph->ihl);

                /* Отправка фрагмента в сеть */
             if(zas_prio_queue_put_data_pkt(pkt_queue,  frag_pkt)  ==
-1){
                 DEBUG_NORM("\n put_fragment dropped 1 queue size =
%d,%d,%d,%d,  max = %d ",
                              pkt_queue->m_size[0],
                              pkt_queue->m_size[1],
                              pkt_queue->m_size[2],
                              pkt_queue->m_size[3],
                              pkt_queue->m_max_size);
                  return false;
             }
                net_pkt_release(frag_pkt);

                /* Переход к следующему фрагменту */
                offset += len;
                copy_offset += len;
                left -= len;
            m_stat.frag_pkt[0]++;
           }
           return true;
}
void add_pkt_normalize_thread(struct thread *thrd, struct net_pkt*
pkt){
    unsigned num_sa = thrd->num_sa;
    DEBUG_NORM("\n add_pkt_normalize_thread: ... body_len = %d ref
= %d ", body_len, pkt->refcnt);
    if(!m_queue_thread[num_sa].init_thread) {
        m_stat.dpopped_pkt[0]++;
        return;
    }
    DEBUG_NORM("\n add_pkt_normalize_thread: num_thrd = %d (mtu =
%d)", num_thrd, m_queue_thread[num_sa].mtu_thread);
    // добавление пакета в очередь потока с разбиением его по
частям
    if(put_fragment(pkt,                              &thrd->pkt_queue,
m_queue_thread[num_sa].mtu_thread) == false)
        m_stat.dpopped_pkt[0]++;
    m_stat.input_pkt[0]++;
    DEBUG_NORM("\n add_pkt_normalize_thread end");
```

```
    return;
}
struct net_pkt* get_pkt_normalize_tread(struct thread *thrd){
     struct net_pkt* pkt = 0;
     unsigned long long current_time;
     unsigned mtu;
     current_time = core_clock_ticks_mksec();
     // если необходимо извлеч пакет
     if((current_time <= thrd->time_call) ||
        (m_queue_thread[thrd->num_sa].Tcurrent == 0)) return 0;
     thrd->time_call += m_queue_thread[thrd->num_sa].Tcurrent;
     mtu = m_queue_thread[thrd->num_sa].mtu_thread;
     // Если очередь пакетов по прежнему пуста, то отправляется
"нулевой" пакет
     if(zas_prio_queue_empty(&thrd->pkt_queue)){
         DEBUG_NORM("\n   get_pkt_normalize_tread:   create   null
pkt");
         pkt = net_pkt_create(mtu);
         if (pkt) {
             //DEBUG_NORM("\n control_thread NULL pkt-len = %d",
m_queue->mtu);
             memset(net_pkt_add_tail(pkt, mtu), 0, mtu);
             pkt->pkt_payload_type = EMPTY_PAYLOAD_DATA;
             pkt->priority = 1;    // низкий приоритет
         }
     }
     else{ // Извлечение пакета с выравниванием до mtu
         struct ipv4_hdr *iph;
         pkt = zas_prio_queue_get(&thrd->pkt_queue);
         ASSERT(pkt != 0, " get_pkt_normalize_tread: ASSERT pkt ==
%x", (unsigned) pkt);
         DEBUG_NORM("\n get_pkt_normalize_tread: get good pkt");
         pkt->pkt_payload_type = IP_PAYLOAD_DATA;
         // удаление хвоста
         iph = (struct ipv4_hdr*)(pkt->head);
         if(ntohs(iph->tot_len) < net_pkt_size(pkt)) {
             net_pkt_cut_tail(pkt,  net_pkt_size(pkt)  -  ntohs(iph-
>tot_len));
         }
         ////////////////////////////////////////////////////
         if(mtu > net_pkt_size(pkt))
             memset(net_pkt_add_tail(pkt,          (mtu          -
net_pkt_size(pkt))), 0, (mtu - net_pkt_size(pkt)));
     }
     return pkt;
}
///============================================================


void control_thread_time(struct timer_handler* ctx){
    struct  queue_thread  *m_queue  =  (struct  queue_thread  *)ctx-
>private_data;
```

```
    task_post(&task_norm[m_queue->num_sa]);
    return;
}
// функция контроля очереди потоков
//  проверяет за раз 1 поток, берет пакет из его очереди и
отправляет
// если пакетов в очереди потока нет, то отправляет пустой пакет
void control_thread(struct task_handler* ctx){
        struct queue_thread *m_queue = (struct queue_thread *)ctx-
>private_data;
        struct thread *thrd;
        struct net_pkt *pkt = 0;
        unsigned long long current_time = core_clock_ticks();
    //int k = m_queue->current_prio+1;
    if(m_queue->size_queue_thread == 0) return;
    if(m_queue->current == 0) m_queue->current = m_queue->next;
    pkt = get_pkt_normalize_tread(m_queue->current);
    if(!pkt)  {  //  если  пакета  нет,  то  необходимо  дождаться
увеличения системного времени
        task_post(ctx);
        //timer_start(&timer_norm[m_queue->num_sa], 0);
        return;
    }
    DEBUG_NORM("\n  control_thread:  pktm_refcnt  =  %d",  pkt-
>refcnt);
    ASSERT(pkt->refcnt == 1,"\n control_thread: pktm_refcnt = %d",
pkt->refcnt);
    // если время жизни потока истекло
    if(current_time > m_queue->current->time_dead) {
        // если в очереди потока нет пакетов, то поток удаляется
        if(pkt->pkt_payload_type == EMPTY_PAYLOAD_DATA) {
            thrd = m_queue->current;
            DEBUG_NORM("\n delete thrd ip = %x", thrd->zas_ip);
            m_queue->current = thrd->next_ln;
            delete_thread_queue(thrd);
            update_time_ctrl();
            net_pkt_release(pkt);
            return;
        }
        else{  // продление времени жизни потока
            m_queue->current->time_dead    =    current_time    +
m_cfg.Time;
        }
    }
    if(pkt->pkt_payload_type == EMPTY_PAYLOAD_DATA){
        aks_tun_output_empty_packet(pkt, m_queue->current->sel);
        net_pkt_release(pkt);
        m_stat.null_pkt[0]++;
    }
    else{
        aks_tun_output_packet(pkt);
        net_pkt_release(pkt);
```

```
        m_stat.good_pkt[0]++;
    }
    m_stat.all_pkt[0]++;
    m_queue->current = m_queue->current->next_ln;
    task_post(ctx);
    return;
}
void normalize_output_pkt(struct net_pkt* pkt){
    struct ipv4_route_entry* rt = (struct ipv4_route_entry*)(pkt-
>dst);
    struct thread *thrd;
    struct aks_selector* sel;
    if(!rt) return;
    if(!rt->m_ipsec_ptr) return; // если маршрут не найден
    sel = (struct aks_selector*) rt->m_ipsec_ptr;
    thrd = (struct thread *)sel->m_thread_normalize;
    // проверка правильности потока
    if(thrd) {
        if(thrd->zas_ip != sel->gw.s_addr){
            sel->m_thread_normalize = 0;
            thrd = 0;
        }
    }
    // DEBUG_NORM("\n output ip dst = %x  ", dst.s_addr);
    // Если поток не найден
    if(!thrd) {
        thrd = get_free_thread();
        if(thrd == 0){
            DEBUG_NORM("\n Warning get_thread: Error create thread
");
            m_stat.add_err[0]++;
            return;
        }
        // инициализация потока
        thrd->zas_ip = sel->gw.s_addr;
        thrd->time_dead = core_clock_ticks() + m_cfg.Time;
        thrd->num_sa = sel->profile;
        sel->m_thread_normalize = thrd;
        thrd->sel = sel;
        // добавление потока в очередь
        if(add_thread_queue(thrd)) //  если  поток  в  очередь  не
добавлен
        {
            set_free_thread(thrd);
            m_stat.add_err[0]++;
            return;
        }
        //  инициализация  начального  значения  времени  отправки
пакетов.
        if(thrd->next_ln) {
            thrd->time_call = thrd->next_ln->time_call;
        }
```

```
    }
    // разбиение пакета и добавление в очередь потока
    add_pkt_normalize_thread(thrd, pkt);
    return;
}
void output_norm_stat(struct stdio_file* file){
    unsigned long long current_time = core_clock_ticks();
    unsigned long long delta_time = current_time -
m_stat.save_time;
    unsigned integrity_pkt[2];
    int i;
    integrity_pkt[0] = m_stat.good_pkt[0] - m_stat.frag_pkt[0];
    integrity_pkt[1] = m_stat.good_pkt[1] - m_stat.frag_pkt[1];
    stdio_fprintf(file, "Normalize all stat: \n");
    stdio_fprintf(file, "Transfer pkt = %d, dwelta = %d (%d
p/s)\n",
                  m_stat.transfer_pkt[0], m_stat.transfer_pkt[0] -
m_stat.transfer_pkt[1],
                  (__u32)(((__u64)(m_stat.transfer_pkt[0] -
m_stat.transfer_pkt[1])*1000)/delta_time));
    stdio_fprintf(file, "All packet input: %d \n",
                  m_stat.input_pkt[0]);
    stdio_fprintf(file, "All packet output: %d (good pkt = %d
(frag = %d, integr=%d), null pkt = %d)\n",
                  m_stat.all_pkt[0], m_stat.good_pkt[0],
m_stat.frag_pkt[0],
                  integrity_pkt[0], m_stat.null_pkt[0]);
    stdio_fprintf(file, "All packet transfer queue pkt: %d \n",
m_stat.transfer_queue_pkt[0]);
    stdio_fprintf(file, "All packet dropped: %d \n",
m_stat.dpopped_pkt[0]);
    stdio_fprintf(file, "Add err: %d \n", m_stat.add_err[0]);
    stdio_fprintf(file, "\nDelta time (%d ms) stat: \n",
(__u32)delta_time);
    stdio_fprintf(file, "All packet input: %d (%d p/s) \n",
                  m_stat.input_pkt[0]-m_stat.input_pkt[1],
                  (__u32)(((__u64)(m_stat.input_pkt[0] -
m_stat.input_pkt[1])*1000)/delta_time));
    stdio_fprintf(file, "All packet output: %d (%d p/s),\n"
                        "good pkt = %d (%d p/s),\n "
                        "null pkt = %d (%d p/s),\n"
                        "integrity pkt = %d (%d p/s),\n"
                        "frag pkt = %d (%d p/s),\n"
                        "transfer queue pkt = %d (%d p/s),\n",
                  m_stat.all_pkt[0] - m_stat.all_pkt[1],
                  (__u32)(((__u64)(m_stat.all_pkt[0] -
m_stat.all_pkt[1])*1000)/delta_time),
                  m_stat.good_pkt[0] - m_stat.good_pkt[1],
                  (__u32)(((__u64)(m_stat.good_pkt[0] -
m_stat.good_pkt[1])*1000)/delta_time),
                  m_stat.null_pkt[0] - m_stat.null_pkt[1],
```

```
                    (__u32)(((__u64)(m_stat.null_pkt[0]            -
m_stat.null_pkt[1])*1000)/delta_time),
                    integrity_pkt[0] - integrity_pkt[1],
                    (__u32)(((__u64)(integrity_pkt[0]              -
integrity_pkt[1])*1000)/delta_time),
                    m_stat.frag_pkt[0] - m_stat.frag_pkt[1],
                    (__u32)(((__u64)(m_stat.frag_pkt[0]            -
m_stat.frag_pkt[1])*1000)/delta_time),
                    m_stat.transfer_queue_pkt[0]                   -
m_stat.transfer_queue_pkt[1],
                    (__u32)(((__u64)(m_stat.transfer_queue_pkt[0]  -
m_stat.transfer_queue_pkt[1])*1000)/delta_time));
    stdio_fprintf(file, "All packet dropped: %d (%d p/s) \n",
                    m_stat.dpopped_pkt[0] - m_stat.dpopped_pkt[1],
                    (__u32)(((__u64)(m_stat.dpopped_pkt[0]         -
m_stat.dpopped_pkt[1])*1000)/delta_time));
    stdio_fprintf(file,  "Add  err:  %d  \n",  m_stat.add_err[0]-
m_stat.add_err[1]);
    m_stat.transfer_pkt[1]   = m_stat.transfer_pkt[0];
    m_stat.input_pkt[1] = m_stat.input_pkt[0];
    m_stat.all_pkt[1]   = m_stat.all_pkt[0];
    m_stat.good_pkt[1]  = m_stat.good_pkt[0];
    m_stat.null_pkt[1]  = m_stat.null_pkt[0];
    m_stat.dpopped_pkt[1] = m_stat.dpopped_pkt[0];
    m_stat.frag_pkt[1] = m_stat.frag_pkt[0];
    m_stat.transfer_queue_pkt[1] = m_stat.transfer_queue_pkt[0];
    m_stat.add_err[1] = m_stat.add_err[0];
    m_stat.save_time = current_time;
    for(i = 0; i < CONFIG_MAX_SA; i++) {
        if(m_queue_thread[i].size_queue_thread)
stdio_fprintf(file,    "sa%d    =    %d    thread    \n",    i,
m_queue_thread[i].size_queue_thread);
    }
    return;
}
////////////////////////////////////////////
static int cmd_norms_func(struct cmd_entry *cmd, char *av[], int
ac, struct stdio_file* file){
    output_norm_stat(file);
    return 0;
}
/// Обработчик консольных команд
static struct cmd_entry cmd_norms;
int cmd_norms_init()
{
        cmd_entry_init(&cmd_norms, cmd_norms_func, "norms",
                "norms",
                "normalize statistic commands", 0);
        return shell_cmd_add(&cmd_norms);
}
```