электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.052

Метод обнаружения некорректного завершения сетевых соединений прикладного уровня

Шушвар Б. В., *магистр*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение и информационные технологии»

Научный руководитель: Крищенко В.А., к.т.н, доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана

Введение

Современное ПО для взаимодействия по сети зачастую использует сложные протоколы прикладного уровня, позволяющие эффективно управлять процессом обмена данными. Зачастую подобные протоколы реализуют модель взаимодействия «клиент-сервер», подразумевающую установку «логического» соединия между взаимодействующими системами. От корректности реализации протокола зависит эффективность и качество работы ПО, поэтому возникает задача проверки реализации протокола. В частности, такая проблема, как некорректное завершение соединения протокола прикладного уровня может оказать негативное влияние на работу сетевой службы. Например, в случае незакрытого сокета, будут неэффективно тратиться системные ресурсы, такие, как файловые дескрипторы. Также возможна ситуация некорректной обработки принимаемых или отправляемых данных, что недопустимо.

Существуют различные методы обнаружения утечки ресурсов. Так, в работе [1] описывается статический метод обнаружения локальных утечек. Однако в случае дескрипторов сокетов этот метод не применим из-за того, что создание и закрытие сокета может происходить в разных функциях или потоках, и неочевиден момент, когда сокет сетевого соединения должен быть закрыт.

Для отслеживания утечек ресурсов в сложных сценариях возможно применение динамических методов анализа. Так, в [2] описано средство Valgrind, позволяющее динамически проверять корректность освобождения ресурсов. Однако в случае с «зависшим» сетевым соединением незакрытый файловый дескриптор сокета будет считаться корректной ситуацией, хотя с точки зрения логики соединения сокет должен быть закрыт. Кроме того, подобные программные средства проверяют факт закрытия файловых дескрипторов на момент завершения программы, но сетевые службы могут

работать в течение неопределенно большого периода времени. Также на момент получения сигнала на завершение процесса невозможно гарантировать корректное завершение соединения с точки зрения протокола прикладного уровня.

Таким образом, существующие средства не в полной мере подходят для решения проблемы, и актуальна становится задача разработки метода обнаружения некорректно закрытых сетевых соединений.

Классификация состояний соединения

Для выявления факторов, влияющих на корректность закрытия сетевого соединения, была рассмотрена общая схема сетевого взаимодействия (рисунок 1). Большинство протоколов используют подобную схему с установлением «логического» соединения, когда для закрытия соединения необходимо отправить соответствующее сообщение («FIN»). Благодаря этому возможно обнаружение факта завершения логического соединения без необходимости использования полной спецификации протокола.

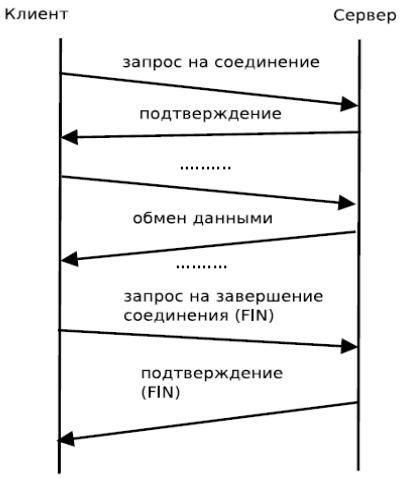

Рис. 1. Последовательность действий клиента и сервера при сетевом взаимодействии

На рисунке 2 показана типичная схема взаимодействия приложения с сетевым стеком POSIX-совместимой ОС. Следует выделить такие важные сущности, как конечный автомат (КА) обработки сообщений протокола прикладного уровня, буферы сокета (буферы уровня ядра) и буферы приложения.

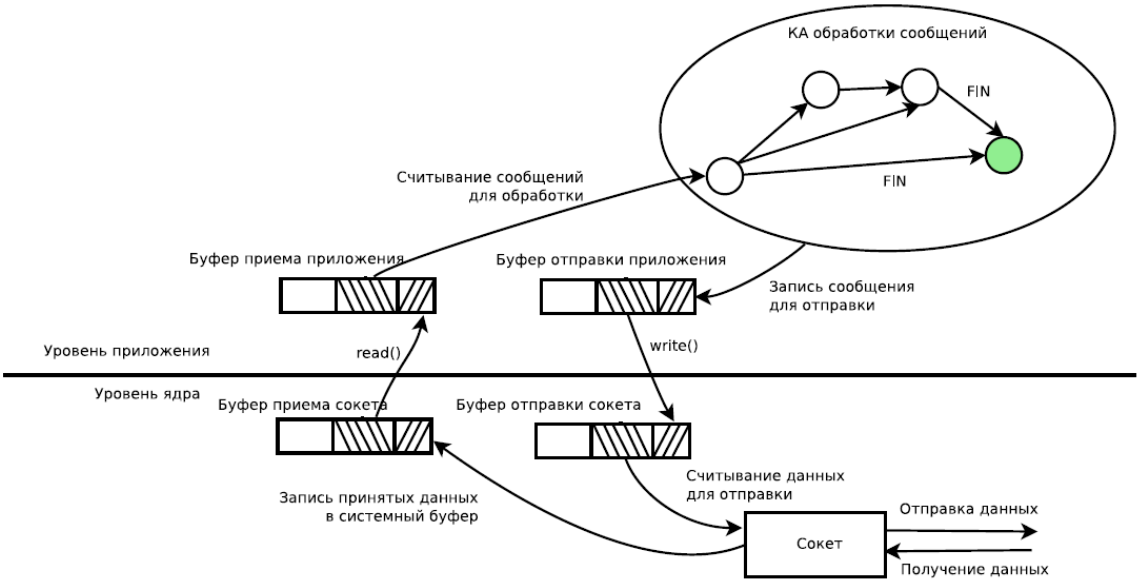

Рис. 2. Схема сетевого взаимодействия приложения с ОС

Исходя из данной схемы, были выделены следующие факторы, влияющие на корректность состояния соединения.

1) Состояние буферов
2) Состояние конечного автомата (КА) обработки сообщений
3) Состояние файлового дескриптора сокета

Таким образом, указанные факторы будут определять корректность текущего состояния соединения. Была разработана классификация состояний соединения, приведенная в таблице 1. Таблица содержит полное декартово произведение факторов, а потому полностью описывает все возможные состояния соединения. Для каждого из возможных состояний соединения указаны характерные признаки такого состояния.

Необходимо отметить, что рассматривается взаимодействие с полностью корректным партнером (т.е. таким партнером, который корректно реализует протокол и не выполняет каких-либо злонамеренных действий). Это позволяет гарантировать то, что возникшие проблемы сигнализируют о проблемах только со стороны клиента.

Таблица 1

Классификация состояний соединения

	Корректное состояние		Некорректное состояние		
	Случай 1	Случай 2	Случай 3	Случай 4	Случай 5
Состояние буферов	*пусты*	*не имеет значения*	*не пусты*	*не меняется*	*не имеет значения*
Состояние КА	*конечное*	*не конечное*	*конечное*	*не имеет значения*	*не конечное*
Состояние сокета	*закрыт*	*не закрыт*	*не имеет значения*	*«зависшее»* *(не закрыт)*	*закрыт*

Случай 1. Соединение в момент корректного завершения: все буферы пусты (т.е. все данные обработаны и отправлены), конечный автомат находится в конечном состоянии, а файловый дескриптор сокета закрыт.

Случай 2. Соединение в активном состоянии: приложения ведут обмен данными, и соединение не должно быть завершено.

Случай 3. Некорректное состояние соединения: не все данные были считаны или отправлены перед завершением соединения. При этом достаточно того факта, что конечный автомат находится в конечном состоянии, так как некорректное состояние буферов говорит об ошибке в логике обработки данных приложением, и корректность закрытия сокета уже не играет роли.

Случай 4. Соединение не закрыто, но приложение некоторое время не ведет сетевой активности. При выполнении ресурсоемких вычислений, требующих значительного времени, рекомендуется разрывать сетевое соединение, а по окончании вычислений и необходимости возобновления передачи данных – устанавливать сетевое соединение заново [3], [5].

Таким образом, данную ситуацию можно считать ситуацией с некорректно завершенным соединением.

Случай 5. Некорректное состояние соединения: сокет был закрыт до отправки сообщения о завершении соединения. В этом случае также неважно состояние буферов системы и приложения, так как нарушено более сильное условие.

На основе приведенной классификации состояний соединения был сформулирован **базовый критерий** корректного завершения соединения: соединение считается завершенным корректно, если на момент его завершения выполнены следующие условия:

4) все данные считаны и обработаны;
5) КА обработки сообщений находится в конечном состоянии;
6) все данные отправлены;
7) файловый дескриптор сокета закрыт.

Соответствие состояния соединения данному критерию говорит о корректном использовании системных ресурсов на протяжении всего процесса сетевого взаимодействия и корректном освобождении по его окончании.

Однако данный критерий невозможно проверить на практике. Так, нет возможности статически определить состояние буферов приложения, так как оно зависит от конкретных данных, использующихся в приложении. Задача определения того факта, что все данные обработаны приложением, также не является решаемой в общем случае.

В связи с этим, на основе полного критерия был сформулирован **проверяемый критерий**: соединение считается завершенным корректно, если на момент его завершения выполнены следующие условия:

1) буфер приема в ядре пуст;
2) КА обработки сообщений находится в конечном состоянии;
3) последняя порция данных была отправлена полностью;
4) файловый дескриптор сокета закрыт приложением.

Так как первое условие проверяемого критерия слабее соответствующего условия базового критерия, то возможно появление ложно-положительных результатов~--- приложение считало все данные из буфера, и, не обработав их, закрыло соединение (что не является корректным завершением соединения по базовому критерию). Такие ситуации редки, поэтому в большинстве случаев ложноотрицательных срабатываний не будет. Второе условие проверяемого критерия полностью совпадает с условием основного критерия. Третье условие ослаблено тем, что отправка считается успешной после того, как успешно завершился соответствующий системный вызов. Не проверяется факт того, что данные действительно были отправлены системой адресату. Программы основываются на том, что системный вызов завершится без сбоев (прерывание выполнения системного вызова из-за сигнала не является сбоем системного вызова). Если при системном вызове произошел сбой, то это нарушает стабильность работы всей системы, и говорить о корректном завершении сетевого соединения нет смысла.

Метод динамической проверки критерия

Так как условия критерия имеют динамическую природу, то был применен метод динамической проверки. Корректность состояния соединения на момент его завершения с точки зрения критерия зависит от хода процесса сетевого взаимодействия, поэтому для проверки критерия необходимо анализировать состояние соединения на всем протяжении взаимодействия.

Первый пункт критерия проверяется следующим образом. Перед моментом закрытия сокета (например, внутри вызова close() или shutdown()) необходимо проверить системным вызовом select() факт наличия данных для считывания. Наличие данных свидетельствует о том, что состояние соединения не соответсвтует критерию. Учитывая условную корректность партнера, можно утверждать следующее.

1) Если проверяемая программа~--- сервер, то после получения от клиента сообщения о завершении соединения серверу не будут отправлены никакие другие данные, кроме, возможно, повторных сообщений о завершении соединения.
2) Если проверяемая программа~--- клиент, то после получения от сервера сообщения-подтверждения о завершении соединения клиенту не будут отправлены никакие другие данные, кроме, возможно, повторных сообщений-подтверждений о завершении соединения.

Таким образом, после того, как проверяемой программой будет полностью считано последнее сообщение о завершении соединения, можно быть уверенным в том, что буфер приема в ядре пуст.

Проверка второго пункта критерия возможна путем перехвата передаваемых сообщений и их анализа на соответствие предоставленной спецификации сообщений о завершении соединения.

Проверка третьего пункта критерия осуществляется следующим образом. При каждом вызове send() или write() проверяется, что результат вызова равнялся длине переданного в него сообщения, и запоминается состояние последней проверки. Если на момент закрытия сокета последнее сравнение дало отрицательный результат, то текущее состояние соединения не соответствует критерию. При этом отрицательный результат указанного сравнения на протяжении взаимодействия не свидетельствует об ошибочной ситуации (т.к. системный вызов может отправить часть данных).

Четвертый пункт критерия, факт закрытия файлового дескриптора сокета, проверяется путем анализа результата системного вызова close() или shutdown().

Таким образом, для реализации проверки критерия необходимо получать информацию о следующих системных вызовах: socket, read/recv, write/send, close/shutdown, accept. Кроме того, необходимо иметь информацию о виде сообщений о завершении соединения для того, чтобы можно было отслеживать момент их приема и передачи. Данная информация может быть представлена в виде спецификации сообщений о завершении соединения.

Текущее состояние соединения относительно критерия было формализовано в виде конечных автоматов (КА).

На рисунке 3 изображен детерминированный по построению конечный автомат проверки критерия для сервера. Для клиента конечный автомат аналогичен серверному с соответствующими изменениями. Состояния автомата представляют собой состояния логики проверки критерия при динамическом анализе состояния соединения: «1» – начальное состояние, «5» – конечное состояние (корректное завершение соединения), «e» – состояние ошибки, означающее некорректное завершение соединения. Остальные состояния являются промежуточными, которые отражают возможные ситуации при анализе состояния соединения. Дуги помечены отслеживаемыми событиями, которые фиксируются путем анализа данных соответствующих системных вызовов, а именно:

1) **send/recv** означает чтение или отправку данных, не являющихся сообщением о завершении соединения;
2) **recvFin** означает получение сообщения о завершении соединения;
3) **sendFin** означает отправку сообщения о завершении соединения;
4) **close** означает закрытие файлового дескриптора сокета;
5) **timeout** означает наступление тайм-аута, соединение признано «зависшим»;
6) **other** означает наступление события, для которого нет отдельной помеченной дуги.

Если конечный автомат проверки критерия в ходе анализа перешел в состояние ошибки, то это означает, что критерий нарушен, и соединение завершено некорректно.

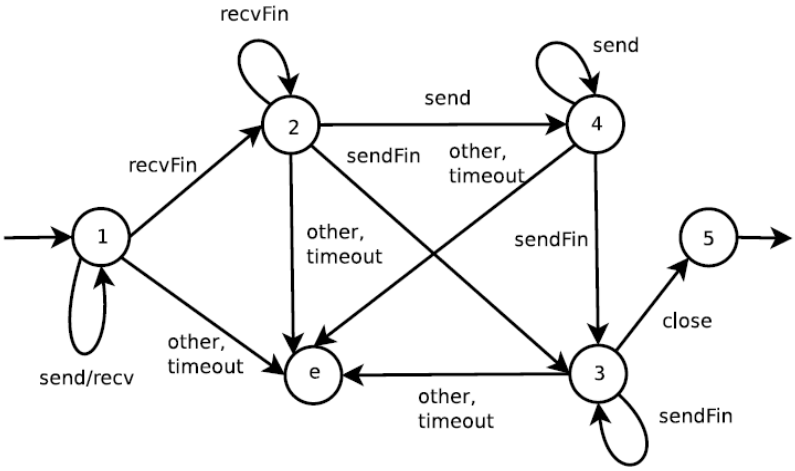

Рис. 3. Конечный автомат проверки критерия для сервера

Алгоритм проверки критерия и особенности реализации

На рисунке 4 изображен алгоритм обработки данных системного вызова. Переход конечного автомата проверки критерия в соответствующее состояние происходит в зависимости от данных о системном вызове, которые анализируются при осуществлении перехода.

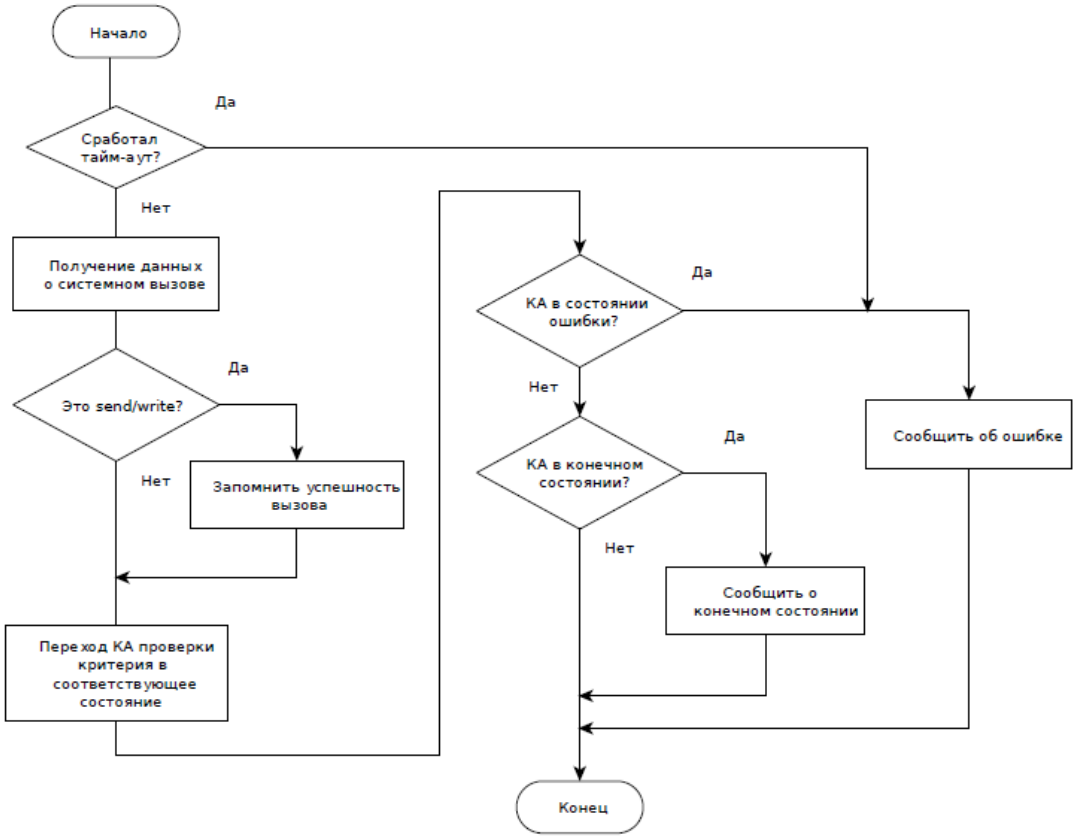

Рис. 4. Алгоритм обработки данных системного вызова

На рисунке 5 изображена общая структура программного комплекса. Стрелками указаны потоки данных между частями комплекса. Серым цветом выделены части программного комплекса, которые были разработаны в рамках данной работы. Остальные части являются внешними и используются в вспомогательных целях.

Были выделены следующие основные части.

1) **Тестируемое приложение.** Программа, которую необходимо проверить на корректность завершения соединений. Она использует библиотеку подмененных системных вызовов вместо оригинальной; использование является «прозрачным».
2) **Библиотека подмененных функций.** Содержит подмененные системные вызовы, которые выполняют оригинальные системные вызовы и передают информацию модулю анализа данных.
3) **Библиотека оригинальных функций.** Оригинальная библиотека системных вызовов.
4) **Модуль проверки критерия.** В данном модуле проверяется критерий корректности завершения соединения на основе данных, получаемых от библиотеки подмененных функций. Это основной модуль программного комплекса.
5) **Тестирующее приложение.** Программа, корректно реализующая протокол прикладного уровня и взаимодействующая с тестируемой программой.

Для получения данных о системных вызовах используется способ перехвата вызовов стандартной библиотеки libc, с помощью которых реализуются системные вызовы. Была разработана библиотека с модифицированными вызовами libc, которая подгружается при запуске проверяемой программы. Вызов подмененной функции является полностью прозрачным для вызывающей стороны и никак не влияет на ее работу. Для передачи данных в модуль проверки критерия используется интерфейс сокетов UNIX, передача в подмененном вызове выполняется асинхронно для минимизации возможных побочных эффектов подмены вызова.

Разработанный метод позволяет работать как с текстовыми, так и с бинарными протоколами. В реализации поддерживаются только текстовые протоколы, так как их анализ является относительно простой задачей. В дальнейшем целесообразно вынести анализ сообщений протокла в отдельный модуль, что позволит разрабатывать и использовать модули для различных протоколов вне зависимости от их природы.

Результаты применения метода

Для тестирования реализованного метода на реальных программах в качестве протокола был выбран SMTP [4]. Это текстовый протокол, позволяющий легко отследить сообщение о завершении соединения в процессе сетевого взаимодействия. Кроме того, существует много открытых реализаций данного протокола в виде агентов почтовых служб (Mail Transfer Agent, MTA), например, **exim4** и **postfix**.

Было произведено тестирование реализаций MTA, выполненные студентами кафедры ИУ7 в рамках курсовой работы по курсу «Протоколы вычислительных сетей». Были протестированы две реализации отправителя почты (клиенты) в составе MTA и две – получателя (серверы).

Серверы тестировались способом имитации корректного клиента, так как не было известно, корректны ли имеющиеся программы клиентов. Для этого запускалась программа-анализатор, затем – тестируемый сервер с использованием подмененной библиотеки. Далее производилось подключение к работающему серверу с помощью утилиты **telnet**, и имитировались запросы корректного клиента.

В качестве корректной программы для взаимодействия при тестировании клиентских программ использовались ранее протестированные программы серверов с допущением, что они являются полностью корректными. Тестирование осуществлялось путем отправки клиентом писем на сервер почты.

В результате тестирования выяснилось, что обе тестируемые клиентские программы завершают соединения некорректно. Так, первый клиент для обработки исходящей почты открывает несколько соединений, через которые он пытается одновременно отправить накопившуюся почту серверу. После того, как все сообщения были успешно отправлены, сокеты, открытые для передачи данных, не закрываются. По истечении периода тайм-аута приложение-анализатор сообщило об ошибочной ситуации. В итоге, клиент нарушает критерий корректности завершения соединения. Второй клиент также не закрывает сокет после завершения передачи данных, однако обрабатывает почту в один поток.

Для того, чтобы удостовериться, что факт обнаружения некорректно завершенных соединений не является ложным, были просмотреты исходные коды тестируемых приложений, в результате чего были обнаружены ошибки в реализации, соответствующие указанным проблемам.

Общие результаты тестирования приведены в таблице 2.

Таблица 2

Результаты применения разработанного программного комплекса

Программа	**Результат**	**Причина ошибки**
Отправитель почты в составе MTA 1	Критерий нарушен	Не закрывает сокет после завершения взаимодействия
Отправитель почты в составе MTA 2	Критерий нарушен	Не закрывает сокет после завершения взаимодействия
Получатель почты в составе MTA 1	Критерий не нарушен	–
Получатель почты в составе MTA 1	Критерий не нарушен	–

Таким образом, исходя из результатов применения разработанного метода к реальным программам, был сделан вывод, что разработанный метод обнаружения некорректного завершения соединений применим в практических целях.

Заключение

Проведен анализ проблемы некорректного завершения соединений протокола прикладного уровня, проанализирована предметная область, выявлены виды ошибок при завершении соединения, сфомулирован и формализован критерий корректного завершения соединения.

На осонове данного критерия разработан метод, позволяющий обнаруживать ситуации некорректного завершения соединения. Также спроектирован программный комплекс для обнаружения случаев некорректного завершения соединения.

Спроектированный комплекс разработан и протестирован с помощью модульных и синтетических тестов. Также проведена проверка реальных программ, являющихся MTA на основе протокола SMTP, на соответствие критерию. В результате проверки были обнаружены случаи некорректного завершения соединений, не являющиеся ложными. Таким образом, разработанный метод можно применять в практических целях.

В качестве дальнейшего развития метода и его реализации можно отметить следующие направления.

1) Добавление возможности подключения модуля анализа произвольных протоколов.

2) Доработка модуля перехвата системных вызовов с целью обеспечения возможности его использования с приложениями, работающими в фоновом режиме (демоны Unix).

Работа выполнена при частичной поддержке Российского фонда фундаментальных исследований (грант № 13-07-00918).

Список литературы

1. Медников А.В., Крищенко В.А. Проверка корректности освобождения ресурсов, локальных для функции на языке С. Режим доступа http://engjournal.ru/catalog/it/hidden/1098.html . Дата доступа: 15.11.2014
2. Nethercote, Nicholas. Valgrind: A framework for heavyweight dynamic binary instrumentation / Nicholas Nethercote, Julian Seward // Programming Language Design and Implementation Conference: Proceedings (San Diego, CA, USA, June 11 – 13, 2007), 2007, pp. 89-100.
3. RFC. TCP RFC Standard. Available at: http://tools.ietf.org/html/rfc793. Accessed: 15.11.2014
4. RFC. SMTP RFC Standard. Available at: http://tools.ietf.org/html/rfc5321. Accessed: 15.11.2014
5. Jon C. Snader Effective TCP/IP Programming. Boston: Addison-Wesley, 2000. 299 p.