УДК 004.715
К. П. СТОРЧАК, канд. техн. наук, доцент;
Т. П. ДОВЖЕНКО, аспирант,
Государственный университет телекоммуникаций, Киев

Исследование сети TCP/IP с применением программного каркаса ERLANG/OTP

Рассмотрен программный каркас (фреймворк) Erlang/OTP и проведено исследование сети TCP/IP с применением данного фреймворка.
Ключевые слова: Erlang/OTP; PI; RED; AQM; TCP/IP-протокол.

Введение

Значительный рост скоростей каналов передачи данных неизбежно приводит к возникновению заторов в телекоммуникационной сети. Пакетная коммутация позволяет повысить эффективность использования каналов, но при этом приводит к снижению надежности доставки. При перегрузке канала с пакетной коммутацией данные на входе канала не смогут поместиться во входной буфер и будут сброшены. Для обеспечения гарантированной доставки пакетов по каналам без гарантированной доставки были разработаны специальные протоколы, одним из которых является протокол TCP. Традиционные протоколы управления очередями и предотвращения перегрузок не справляются с управлением трафиком со сложной динамикой и нелинейностью изменения нагрузки, что приводит к возникновению перегрузок, уменьшению эффективной скорости передачи данных и ухудшает параметры качества, такие как доля (в процентах) потерянных пакетов, задержки и вариации задержек.

Цель статьи — исследование сети TCP/IP с применением фреймворка **OTP** (*Open Telecom Platform*), представленного на языке Erlang, при использовании системы **AQM** (*Active Queue Managment*) активного управления очередью на базе таких алгоритмов, как **RED** (*Random Early Detection*) и **PI-controller** (*Proportional-Integral controller*).

Основная часть

Постановка задачи

Провести исследование работы AQM-системы в условиях изменения трафика при использовании OTP-фреймворка.

Обзор системы AQM

Главные задачи, решаемые при помощи алгоритмов управления очередью, сводятся к минимизации средней длины очереди при одновременном обеспечении высокого коэффициента использования канала, а также справедливое распределение буферного пространства между различными потоками данных [1]. Схемы управления очередью различаются, преимущественно, критерием, по которому отбрасываются пакеты, и местом в очереди (ее начало или конец), откуда осуществляется отбрасывание пакетов. Наиболее простым критерием для отбрасывания пакетов является достижение очередью определенного порога, называемого *максимальной длиной очереди*.

Система алгоритмов AQM обеспечивает заблаговременное обнаружение перегрузки. Основными целями указанных алгоритмов являются:

- минимизация задержки пакетов благодаря контролю среднего размера очереди;
- предотвращение эффекта глобальной синхронизации TCP-трафика;
- обеспечение непредвзятого обслуживания трафика, характеризующегося кратковременными всплесками;
- строгое ограничение максимального среднего размера очереди.

Обзор основных AQM-алгоритмов представлен в [2].

Язык программирования Erlang и фреймворк OTP

Erlang — разработанный и поддерживаемый компанией Ericsson [3] функциональный язык программирования со строгой динамической типизацией, предназначенный для создания распределенных вычислительных систем.

Популярность Erlang начала расти по мере расширения области его применения (телекоммуникационные системы) с охватом высокопараллельных распределенных систем, обслуживающих миллионы пользователей, таких как системы управления содержимым, веб-серверы и распределенные, требующие масштабирования базы данных, кластерные операционные системы (Clustrx), системы управления коммутаторами и другим сетевым оборудованием (программный коммутатор ECSS-10, ПО коммутаторов широкополосных телефонных линий).

В Erlang имеется набор инструментов для эффективной организации параллельных вычислений. Отличительной особенностью языка

является применение облегченных процессов в соответствии с моделью акторов (математическая модель параллельных вычислений). Данный подход позволяет выполнять одновременно сотни тысяч и даже миллионы таких процессов, каждый из которых может иметь скромные требования, касающиеся памяти [4].

Существующие процессы могут порождать другие процессы, выполняться одновременно, обмениваться сообщениями, реагировать на завершение друг друга. Процессы, о которых идет речь, характеризуются изолированностью друг от друга и не имеют общего состояния, но между ними можно установить связь и получать сообщения об их течении.

Для взаимодействия процессов используется асинхронный обмен сообщениями. Каждый процесс имеет свою очередь сообщений, обработка которой использует сопоставление с образцом. Процесс, отправивший сообщение, не получает уведомления о доставке. Таким образом, ответственность за правильно организованное взаимодействие между процессами лежит на разработчике.

Процесс можно связать с другим процессом, в результате чего между ними устанавливается двунаправленное соединение. В случае, если один из процессов завершается аномально, всем связанным с ним процессам передается сигнал выхода. Процессы, получившие этот сигнал, завершаются, распространяя полученный сигнал далее. Причина завершения передается по цепочке завершающихся процессов.

Процесс может осуществить перехват ошибки, если у него установлен флаг перехвата выхода. Такой процесс получает сигналы выхода связанных с ним процессов в виде обычных сообщений с той же структурой. Перехваченный сигнал выхода более не передается процессом, связанным с процессом-перехватчиком. Сигнал выхода с причиной, являющейся атомом normal (нормальное завершение процесса), не вызывает завершения связанного процесса. Если же причина — атом kill, процесс завершается безусловно (независимо от флага перехвата выхода), а связанным с ним процессам в качестве причины отправляется атом killed, что дает им возможность отреагировать.

В Erlang есть возможность установить и однонаправленное соединение. При завершении наблюдаемого процесса процесс-наблюдатель получает сообщение с указанием причины завершения.

Процесс также может остановить сам себя или другой процесс, вызвав соответствующую функцию.

Erlang с самого начала проектировался для распределенных вычислений и масштабируемости. Работающий экземпляр среды выполнения Erlang называется ***узлом***. Программы, написанные на Erlang, способны работать на нескольких узлах. Узлами могут быть процессоры, группы ядер одного процессора и даже целый кластер машин. Узел имеет имя и «знает» о существовании других узлов на данной машине или в сети. Создание и взаимодействие процессов различных узлов не отличается от организации взаимодействия процессов внутри узла. Синтаксис отправки сообщения процессу на своем узле и на удаленном один и тот же. Благодаря встроенным в язык возможностям распределенных вычислений, объединение в кластер, балансировка нагрузки, добавление узлов и серверов, повышение надежности — все это вызывает лишь небольшие затраты на дополнительный код [5].

Программы на высокоуровневом языке Erlang могут быть использованы в системах мягкого реального времени (которое иногда переводят как «псевдореальное» или «квазиреальное»). Автоматизированное управление памятью и сборка мусора действуют в рамках одного процесса, что дает возможность создавать системы с миллисекундным временем отклика (даже несмотря на необходимость сборки мусора), не испытывающие ухудшения пропускной способности при высокой нагрузке.

Erlang был целенаправленно разработан для применения в распределенных, отказоустойчивых параллельных системах реального времени, для реализации которых кроме средств самого языка имеется стандартная библиотека модулей и библиотека шаблонных решений (так называемых поведений) — ***фреймворк OTP***.

OTP является хорошо отлаженным набором полезных поведений (behavior) процессов в рамках модели акторов. В модулях OTP определены общие стандартизированные шаблоны для конструирования параллельных приложений. Наиболее популярными поведениями являются обобщенный сервер (gen_server) и наблюдатель (supervisor), но имеются и другие: конечный автомат (gen_fsm), обработчик событий (gen_event).

OTP-поведения (рис. 1) делятся на рабочие процессы (worker processes), выполняющие собственно обработку запросов, и процессы-наблюдатели или супервайзеры (supervisors).

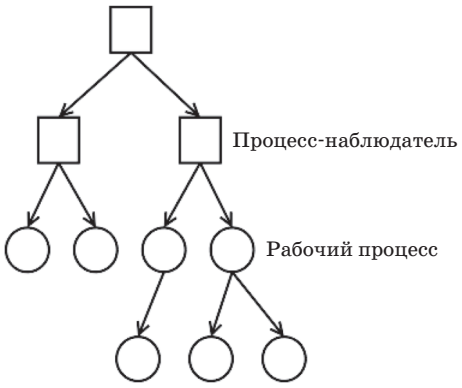

Рис. 1. Дерево процессов OTP-фреймворка

В задачу последних входит слежение за рабочими процессами и другими процессами-наблюдателями. Деревья наблюдателей составляют OTP-приложение (application). OTP-приложение — это компонент, реализующий ту или иную функциональность, которая может быть независимо запущена на исполнение и остановлена как целое, а также повторно использована в других системах. Разработчик приложения пишет код модулей функций обратного вызова (callback module), в которых и находится специфичная для данного приложения часть функциональности [6].

Отказоустойчивость приложений, написанных на Erlang, обеспечивается на трех уровнях. ***Во-первых***, процессы изолированы друг от друга. И если в одном из процессов возникает ошибка, то прерывается только его работа. Вся остальная система продолжает работать. ***Во-вторых***, процессы работают под присмотром супервайзеров. Если процесс останавливается, то супервайзер запускает его заново. В случае, когда ошибка повторяется снова и снова, а процесс каждый раз аварийно завершается, перегрузки не помогают, и после нескольких попыток супервайзер завершается сам. В этом случае его перезагружает родительский супервайзер. То есть перезагрузка происходит на все более высоком уровне, пока или проблема не решится, или не завершится корневой супервайзер. ***Третий уровень защиты*** — распределенность. Erlang-узлы могут быть объединены в кластер, и это позволяет сохранять работоспособность при падении одного из узлов.

Помимо всего перечисленного Erlang-система позволяет выполнять интеграцию с системами на других языках программирования. Имеются механизмы для сетевого взаимодействия с C, Java, Лисп, Perl, Python, Ruby. Например, для более эффективного синхронного вызова небольших функций на C можно использовать так называемые **NIF-функции** (*Natively Implemented Function*). Высокоуровневые библиотеки позволяют Erlang-системе представлять C или Java-узлы как обычные Erlang-узлы. Другие языки могут быть более тесно сопряжены со средой выполнения Erlang при помощи драйверов или сетевых сокетов на базе протоколов вроде HTTP, SNMP.

Эмпирическое исследование показало, что для изученных телекоммуникационных приложений код на Erlang был на 70-85% короче, чем на C++, а производительность системы при переписывании кода с C++ на Erlang возросла почти на 100%. Для одного из использованных в исследовании проектов разница была объяснена написанием дополнительного кода C++ в рамках защитного программирования, управления памятью, а также кода для высокоуровневой коммуникации, т. е. возможностями, которые являются частью языка Erlang и библиотек OTP [7].

Решение задачи

В статье [2] было описано исследование сети, которое проводилось с использованием программного комплекса NS-2 [8]. Схема сети (рис. 2) включает в себя 4 FTP источника сообщения (S1, S2, S3, S4), которые при помощи маршрутизатора передают информацию на TCP-приемник.

Рис. 2. Схема сети для имитационного моделирования

Скорость канала между источниками сообщения и маршрутизатором составляет 10 Мбит/с, задержка для каждого источника разная: S1 = 2 мс, S2 = 3 мс, S3 = 4 мс, S4 = 5 мс; объем каждого пакета — 210 байт. Скорость канала между маршрутизатором и TCP-приемником составляет 4 Мбит/с (канал с перегрузкой), а задержка — 20 мс. При исследовании моделируемой сети нагрузка на маршрутизатор будет постепенно увеличиваться. Первым начнет работу источник S1. Затем, через 4 с — S2. На 8-й секунде включится S3, а на 12-й — S4. Процесс моделирования длится 25 с.

После выполнения программы получим графики зависимости длины очереди (пакеты) от времени работы сети для каждого алгоритма (рис. 3–6).

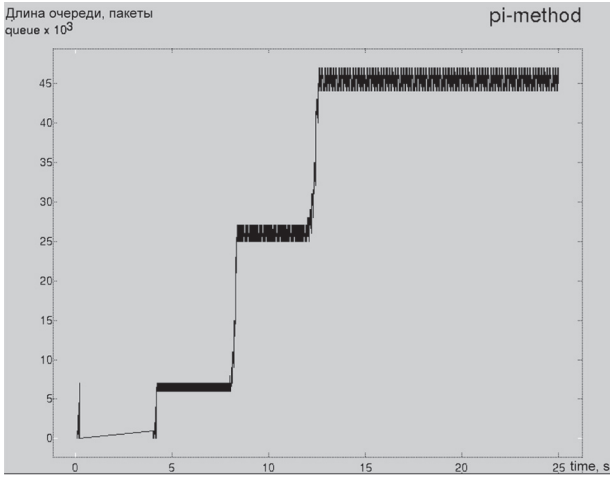

Рис. 3. Длина очереди с использованием PI-алгоритма

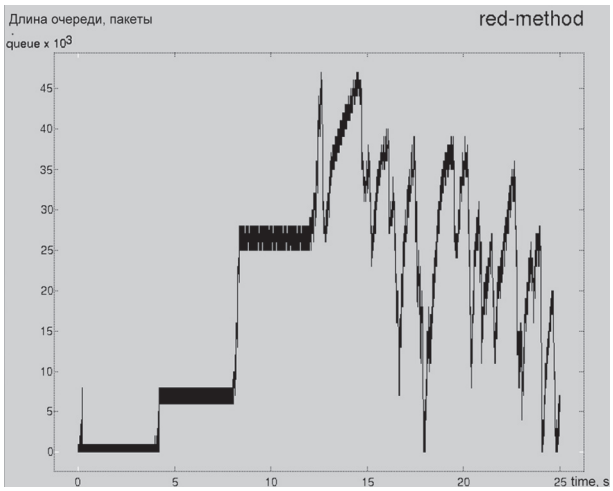

Рис. 4. Длина очереди с использованием RED-алгоритма

Теперь построим аналогичную модель сети при помощи OTP-фреймворка. Далее представлена часть программного кода, отвечающая за создание узлов сети.

```
-module(sbroker_example).
-behaviour(sbroker).
-export([start_link/0]).
-export([init/1]).
start_link() ->
 sbroker:start_link(?MODULE, undefined).

init(_) ->
 QueueSpec = {squeue_timeout, 200, out, 16, drop},
 Interval = 100,
 {ok, {QueueSpec, QueueSpec, Interval}}.
{ok, Broker} = sbroker_example:start_link(),
*** = spawn_link(fun() -> sbroker:ask_r(Broker) end),
{go, _Ref, ***, _SojournTime} = sbroker:ask(Broker).
{AsyncRef, {go, Ref, ***, SojournTime}}
{AsyncRef, {drop, SojournTime}}
{ok, Broker} = sbroker_example:start_link().
{await, AsyncRef, Broker}= sbroker:async_ask(Broker).
ok = sbroker:cancel(Broker, AsyncRef).
*** - Место вставки AQM-метода
```

После выполнения данной программы получим следующие результаты.

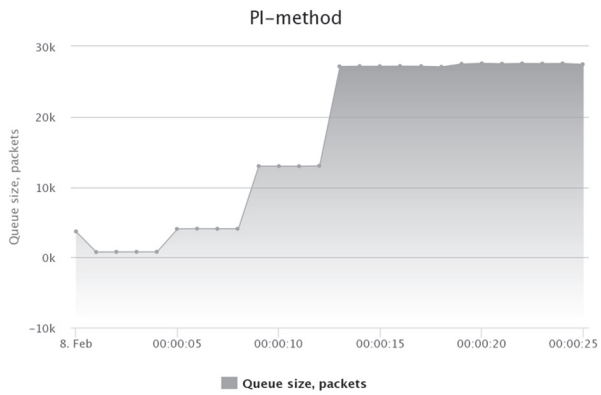

Рис. 5. Длина очереди с использованием PI-алгоритма и Erlang/OTP-фреймворка

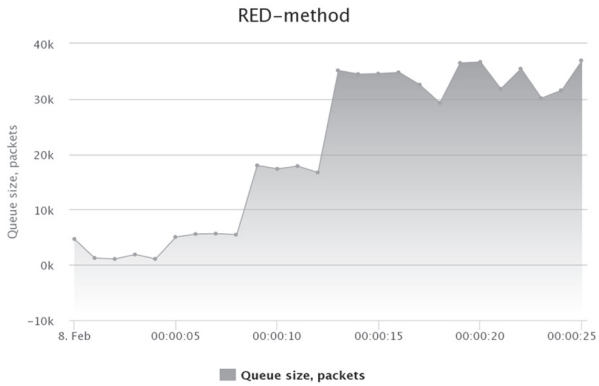

Рис. 6. Длина очереди с использованием RED-алгоритма и Erlang/OTP-фреймворка

Выводы

Полученные результаты (см. рис. 3 и 4) показывают, что PI-алгоритм является более подходящим для использования в сетевых маршрутизаторах, чем RED, поскольку обладает высокой стабильностью длины очереди.

При использовании программного каркаса Erlang/OTP уменьшается очередь маршрутизатора, а также увеличивается ее стабильность (см. рис. 5 и 6).

Список использованной литературы

1. ***Коваленко, Т. Н.*** *Модель активного управления очередями в расределенных инфокоммуникационных системах, представленная сетью Петри / Т. Н. Коваленко // Проблемы телекоммуникаций.— 2012.— № 2 (7).— С. 58–67.*

2. ***Гостев, В. И.*** *Исследование сети TCP/IP с применением основных алгоритмов активного управления очередью / В. И. Гостев, Т. П. Довженко, А. С. Артющик // Системи управління, навігації та зв'язку.— 2014.— №2 (30).— С. 87–91.*

3. ***Erlang*** *programming language [Электронный ресурс].— Режим доступа:*
http://www.erlang.org/.

4. ***Thompson, S. J.*** *Erlang Programming: A Concurrent Approach to Software Development / S. J. Thompson, F. Cesarini. 1st ed. Sebastopol, California: O'Reilly Media, Inc., 2009.— 496 p.*

5. ***Глебов, А. Н.*** *Параллельное программирование в функциональном стиле / А. Н. Глебов [Электронный ресурс].— Режим доступа:*
http://www.softcraft.ru/parallel/ppfs.shtml.

6. ***Чезарини, Ф.*** *Программирование в Erlang / Ф. Чезарини, С. Томпсон.— М., 2012.— 487 с.*

7. ***Nyström, J. H.*** *High-level distribution for the rapid production of robust telecoms software: comparing C++ and ERLANG / J. H. Nyström, P. W. Trinder, D. J. King // Concurrency and Computation: Practice and Experience.— 2008.— Т. 20.— № 8.— P. 941–968.*

8. ***The*** *Network Simulator NS-2 [Электронный ресурс].— Режим доступа:*
http://www.isi.edu/nsnam/ns/

Рецензент: доктор техн. наук, профессор **В. И. Гостев**, Государственный университет телекоммуникаций, Киев.

К. П. Сторчак, Т. П. Довженко
ДОСЛІДЖЕННЯ МЕРЕЖІ TCP/IP З ВИКОРИСТАННЯМ ПРОГРАМНОГО КАРКАСУ ERLANG/OTP
Розглянуто програмний каркас (фреймворк) Erlang/OTP і проведено дослідження мережі TCP/IP із застосуванням цього фреймворка.
Ключові слова: Erlang/OTP; RED; PI; AQM; алгоритм активного управління чергою; TCP/IP-протокол.

K. Storchak, T. Dovzhenko
RESEARCH OF TCP/IP NETWORK USING THE ERLANG/OTP FRAMEWORK
In this article considered a software framework (framework) Erlang / OTP and investigated TCP/IP network using the framework.
Keywords: Erlang/OTP; RED; PI; AQM; active queue management algorithm; TCP/IP-protocol.

УДК 004.738

С. І. ОТРОХ, канд. техн. наук, доцент;
В. О. ЯРОШ, аспірант;
Є. П. ГОРОХОВСЬКИЙ, здобувач;
Ю. М. ЗІНЕНКО, здобувач,
Державний університет телекомунікацій, Київ

Оцінювання показників стійкості мережі майбутнього (FN) до зовнішніх дестабілізуючих факторів

Подано визначення мережі майбутнього (FN), головна відмітна особливість якої — це здатність до самовідновлення та самоорганізації завдяки притаманній сталості та стійкості до дії стихійних лих. Сформовано методи забезпечення стійкості мережі та її відновлення після впливу стихії. Розроблено систему оцінювання показників стійкості до потужного електромагнітного та іонізуючого випромінювання, а також подано рекомендації стосовно підвищення стійкості мереж FN до дії зовнішніх дестабілізуючих факторів.

Ключові слова: мережі майбутнього; стихійні лиха; зовнішні дестабілізуючі фактори; стійкість.

Вступ

У наш час, коли людство потерпає від небачених досі катастроф техногенного та природного характеру, посилюються загрози локальних конфліктів, дедалі важливішого значення набуває завдання забезпечити стале функціонування мережі майбутнього (**FN** — *Future Networks*) — результат еволюційного розвитку мереж наступного покоління [1]. Концепція FN характеризується безперервною зміною вимог до телекомунікаційних мереж, появою принципово нових прикладних сфер дистанційного керування побутовою та іншою технікою (Internet of Things), створенням «розумних» мереж (Smart Gridg) із використанням хмарних обчислень (Cloud Computing).

Згідно з визначенням МСЕ, мережа майбутнього [2] являє собою глобальну інформаційну інфраструктуру, яка включає в себе наявні інформаційно-комунікаційні мережі з урахуванням тих компонентів, які тільки плануються до впровадження. Єдиним центром управління глобальною інформаційною інфраструктурою забезпечується здатність надавати повний спектр телекомунікаційних послуг (у будь-якому географічному районі, причому гарантованої якості, прийнятної вартості та в будь-який час) на базі новітніх та інноваційних технологій. Головна істотна характеристика FN полягає в її здатності до самовідновлення та самоорганізації за рахунок сталості та стійкості до дії стихійних лих. Безвідмовне функціонування FN під час дії стихійного лиха — це чинник забезпечення потреб щодо управління державою, підтримання її обороноздатності, гарантування безпеки, охорони правопорядку, господарського комплексу країни, а також потреб фізичних і юридичних осіб щодо високоякісних послуг телекомунікацій.

Отже, в епоху глобальних змін, передбачуваних і непередбачуваних загроз необхідно сформувати такі вимоги до FN, щоб вона не втрачала стійкості під впливом будь-яких зовнішніх дестабілізуючих факторів (ЗДФ). Для того, аби якомога надійніше захистити мережу, необхідно розробити певні рекомендації стосовно проектування FN із метою мінімізації дії ЗДФ та відновлення сталого функціонування мережі. ЗДФ — це певний вид зовнішніх впливів, параметри якого перевищують ті значення, на які було розраховано елемент мережі при його проектуванні. До ЗДФ віднесено також стихійні лиха, про які йдеться в Рекомендації L.392 сектору стандартизації Міжнародного союзу електрозв'язку, поданій наприкінці 2016 року [3].

Основна частина

Стійкість мережі та її відновлення після дії стихійного лиха досягаються за допомогою багатократних методів [3]. *Головний метод* передбачає ***максимальне посилення*** (*укріплення*) ***мереж***,