ктури SDN, як фізичної, так і віртуальною. Завдяки централізації управління пристроями рівня інфраструктури OpenFlow. До переваг протоколу OpenFlow можна віднести:

- спрощення управління мережею;
- розширення можливостей програмування;
- спрощує управління мережею та програмування мережевих пристроїв;
- динамічна зміна потоків трафіку [4].

Мережа може швидше реагувати на мінливі бізнес-потреби. Контролер SDN підтримує відкритий інтерфейс програмування (API), який дозволяє програмувати його ззовні, створюючи таким чином середовище для автоматизації, контролю, а також масштабувати функціонал для майбутніх додатків. По суті це дає можливість застосування підходу SDN як у великих компаніях і телекомах, так і малому та середньому бізнесі.

Висновок

Реалізація концепції SDN на практиці дозволить підприємствам і операторам зв'язку отримати вендоронезалежний контроль над всією мережею з єдиного місця, що значно спростить її експлуатацію. Що не менш важливо, конфігурування мережі сильно спроститься і адміністраторам не доведеться вводити сотні рядків коду окремо для різних комутаторів або маршрутизаторів. Характеристики мережі можна буде оперативно змінювати в режимі реального часу, відповідно, терміни впровадження нових додатків і сервісів значно скоротяться.

Список літератури

1. Thomas D. Nadeau, Ken Gray, SDN: Software Defined Networks, O'Reilly, 2013. 2.

2. Бакланов И. Г. SDN-NGSDN; практический взгляд на развитие транспортных сетей. Современный язык систем эксплуатации связи. – М. : Метротек, 2006.

3. Коломеец А. Е., Сурков Л. В. Программно-конфигурируемые сети sdn–принципиально новый подход к построению сетей //ббк 32.973 С56. – 2014. – С. 33.

4. Гайдур Г. І. Визначення оптимального маршруту для забезпечення надійності системи управління безпроводовими мережами //Телекомунікаційні та інформаційні технології. – 2014. – №. 4. – С. 68-73.

Надійшла до редколегії 29.01.2015

Рецензент: доктор технічних наук, проф. С.В. Козелков, Державний університет телекомунікацій, Київ.

БУДУЩЕЕ СЕТЕЙ - SDN

Г.И. Гайдур

В статье рассмотрено, как должны изменяться сети, что напрямую связано с передачей данных, которые динамично изменяются. Сети SDN позволяют централизовать управление устройствами, что позволяет легко управлять трафиком, масштабировать сеть и повысить ее производительность.

Ключевые слова: *сеть, SDN, архитектура, протокол, OpenFlow.*

FUTURE NETWORKS - SDN

G.I. Haydur

The article considers how the network should be changed, which is associated with the transmission of data, which are dynamically changing. SDN network enable centralized management of devices, making it easy to manage traffic, network scale and improve its performance.

Keywords: *network, SDN, architecture, protocol, OpenFlow.*

УДК 004.715

Т.П. Довженко

Государственный университет телекомуникаций, 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.*

Вступление

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 поместится 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 процент потерянных пакетов, задержки и вариации задержек. В данной работе производиться исследование сети TCP/IP с применением фремвока OTP (Open Telecom Platform), написаном на языке Erlang, при использовании системы активного управления очередями AQM (Active Queue Managment) таких как: RED (Random Early Detection) и PI-controller (Proportional-Integral controller).

Постановка задачи

Главная цель работы заключается в исследовании работы AQM-системы в условиях изменения нагрузки трафика при использовании OTP-фреймворка.

Обзор системы AQM

Основные задачи алгоритмов управления очередями – минимизация средней длины очереди при одновременном обеспечении высокого коэффициента использования канала, а также справедливое распределение буферного пространства между различными потоками данных[1]. Схемы управления очередями различаются, в основном, критерием, по которому отбрасываются пакеты, и местом в очереди, откуда производится отбрасывание пакетов (начало или конец очереди). Наиболее простым критерием для отбрасывания пакетов является достижение очередью определенного порога, называемого максимальной длиной очереди.

Система активного управления очередями (AQM), обеспечивает заблаговременное обнаружение перегрузки. Основными целями алгоритмов AQM являются:

- минимизация задержки пакетов путем контроля 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 был приведен в источнике[2].

Язык программирования Erlang и фреймворк OTP

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

Популярность Erlang начала расти в связи с расширением его области применения (телекоммуникационные системы) на высокопараллельные распределённые системы, обслуживающие миллионы пользователей, такие как системы управления содержимым, веб-серверы и распределённые, 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 облегчённых процессов в соответствии с моделью акторов(математическая модель паралельных вычислений). кой подход позволяет выполнять одновременно сотни тысяч и даже миллионы таких процессов, каждый из которых может иметь скромные требования по памяти [4].

Процессы могут порождать другие процессы, выполняться одновременно, обмениваться сообщениями, реагировать на завершение друг друга. Процессы изолированы друг от друга и не имеют общего состояния, но между ними можно установить связь и получать сообщения об их состоянии. 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, даже если идентификатор процесса-получателя недействителен или получатель игнорирует сообщение. 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 ненормально, всем связанным с ним процессам передаётся сигнал выхода. Процессы, получившие сигнал, завершаются, распространяя сигнал дальше. Причина завершения передаётся 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 сигнала выхода более не передаётся связанным с процессом-перехватчиком процессам. Сигнал выхода с причиной – 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 даёт им возможность среагировать.

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 функцию.

Erlang с самого начала проектировался для распределённых вычислений и масштабируемости. Работающий экземпляр среды выполнения Erlang называется узлом. Программы, написанные на Erlang, способны работать на нескольких узлах. Узлами могут быть процессоры, многие ядра 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 разных узлов не отличается от организации взаимодействия процессов внутри узла. Синтаксис отправки сообщения процессу на своём узле и удалённом один и тот же. Благодаря встроенным в язык возможностям распределённых вычислений объединение в кластер, балансировка нагрузки, добавление узлов и серверов, повышение надёжности вызывают 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 даёт 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 распределённых, 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 обобщённый сервер (gen_server) и наблюдатель (supervisor), но имеются и другие: конечный автомат (gen_fsm), обработчик событий (gen_event).

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

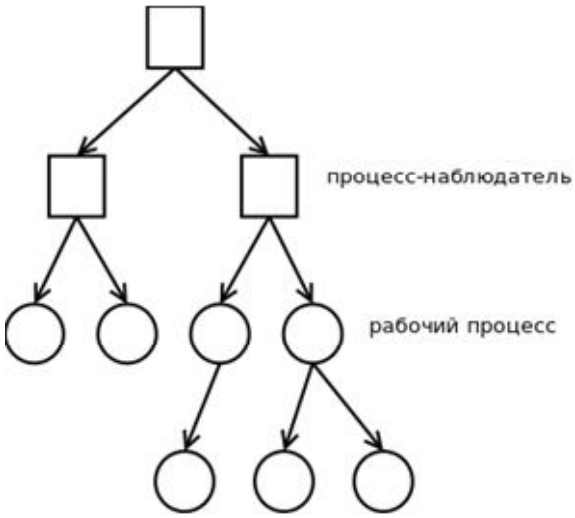

Рис. 1. Дерево процессов OTP-фреймворка

В задачу последних входит слежение за рабочими процессами и другими процессами-наблюдателями. Деревья наблюдателей составляют OTP-приложение (application). OTP-приложение – это компонент, реализующий некоторую 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, и процесс все время 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 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 вышеперечисленного 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 с помощью драйверов или сетевых сокетов посредством 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, то есть возможностями, которые являются частью языка Erlang и библиотек OTP [7].

Решение задачи

В статье[2] было описано исследование сети проводилось с использованием программного комплекса NS-2[8]. Схема сети (см. Рисунок 3) состоит из 4 FTP источников сообщения(S1,S2,S3,S4), которые с помощью маршрутизатора передают информацию на TCP-приемник.

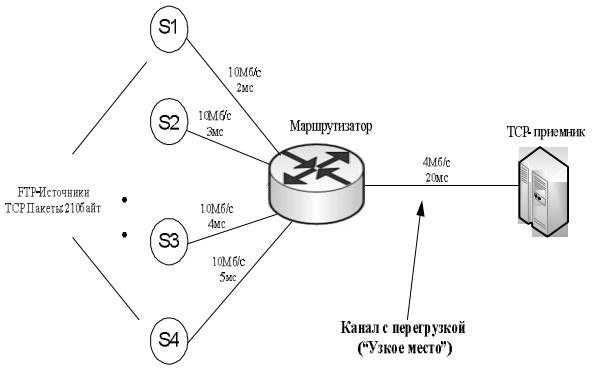

Рис. 2. Схема сети для имитационного моделирования

Скорость канала между источниками сообщения и маршрутизатором составляет 10 Мб/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 4Мб/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 сек - S2. На 8-й секунде включиться S3, а в момент времени 12 сек - S4. Продолжительность процесса моделирования составляет 25 сек.

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

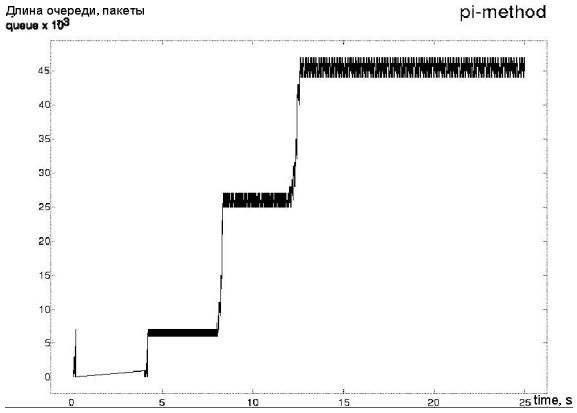

Рис. 3. Длина очереди с использованием PI-алгоритма

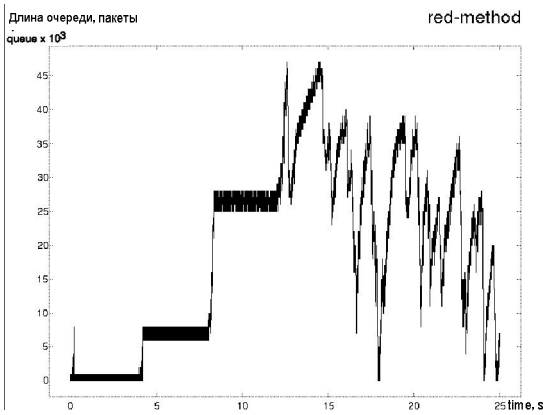

Рис. 4. Длина очереди с использованием RED-алгоритма

Теперь построим аналогичную модель сети с помощью OTP-фреймворка. Ниже 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.

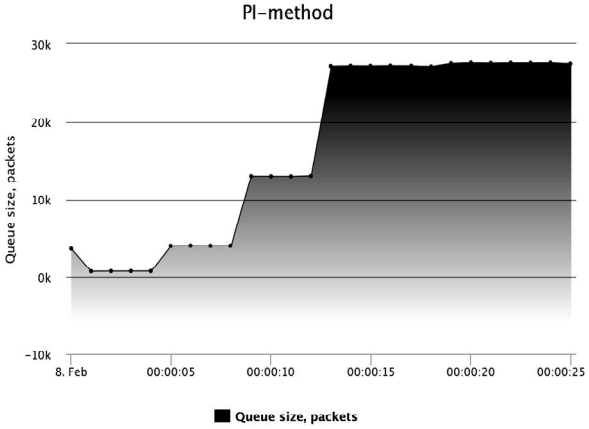

Рис. 5. Длина очереди с использованием PI-алгоритма и Erlang/OTP-фреймворка

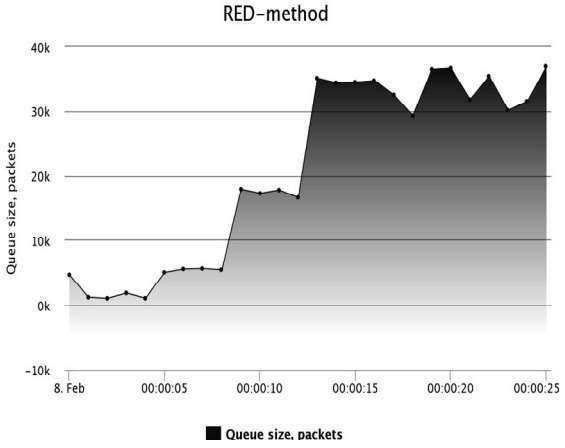

Рис. 6. Длина очереди с использованием RED-алгоритма и Erlang/OTP-фреймворка

Выводы

Учитывая полученные результаты(см. Рис. 3, 4), можно сделать вывод о том, что PI-алгоритм является более подходящим для использования в сетевых маршрутизаторах, чем RED, потому что обладает высоким значением стабильности длины очереди. При использовании программного каркаса Erlang/OTP уменьшается значение очереди маршрутизатора, а также увеличивается ее стабильность(см. Рис. 5, 6).

Список литературы

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

2. Гостев В.И., Довженко Т.П., Артющик А.С., Исследование сети TCP/IP с применением основных алгоритмов активного управления очередью//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. – 496p.

5. Глебов, А. Н. Параллельное программирование в функциональном стиле. – 2003 [Электронный ресурс]. http://www.softcraft.ru/parallel/ppfs.shtml.

6. Программирование в Erlang/ Чезарини Ф., Томпсон С./ М: ДМК-2012 – 487с.

7. Nyström, J. H., Trinder, P. W., King, D. J. 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. *8.The Network Simulator NS-2, http://www.isi.edu/nsnam/ns/*

Поступила в редакцию 2.02.2015

Рецензент: д-р техн. наук, проф. В.И. Гостев, Государственный университет телекомуникаций, Киев.

ДОСЛІДЖЕННЯ МЕРЕЖІ TCP/IP З ВИКОРИСТАННЯМ ПРОГРАМНОГО КАРКАСУ ERLANG/OTP

Т.П. Довженко

У статті розглянуто програмний каркас (фреймворк) Erlang / OTP і проведено дослідження мережі TCP / IP із застосуванням даного фреймворку.

Ключові слова: *Erlang/OTP, RED, PI, AQM, алгоритм активного управління чергою, TCP / IP-протокол.*

RESEARCH OF TCP / IP NETWORK USING THE ERLANG/OTP FRAMEWORK

T.P. Dovzhenko

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.*