Равномерное распределение нагрузки аппаратно-программного ядра в UNIX-системах

Е.В. Пальчевский <teelxp@inbox.ru>
А.Р. Халиков <khalikov.albert.r@gmail.com>
ФГБОУ ВО Уфимский государственный авиационный технический университет,
450000, Россия, г. Уфа, ул. К. Маркса, 12

Аннотация. В данной статье рассматривается задача максимального увеличения пропускной способности сетевого стека при взаимодействии с аппаратно-программным ядром для обеспечения стабильности работы физического сервера. Разработаны алгоритмы и программные коды для оптимизации распределения нагрузки на центральные процессоры методом параллелизации ядра. Также рассмотрены статистические данные повышающейся мощности распределенных атак, влияющих на сетевую инфраструктуру. Показано влияние приложений, имеющих выход во внешнюю глобальную сеть, на производственную часть физического сервера, представляемой в виде физических ресурсов. С помощью разработанного и реализованного алгоритма (на языке «*BASH*»), произведено распределение нагрузочной способности по физическим ядрам сервера с целью дальнейшего снижения нагрузочной способности на вычислительную мощность центрального процессора. Продемонстрированы блок-схемы алгоритмов, а также итоговые результаты тестирования каждого этапа разработки. Реализована оптимизация сетевого режима «*AF_PACKET*», благодаря которой появилась возможность принимать внешние сетевые пакеты без каких-либо блокировок, что, в свою очередь, повышает эффективность достижения заданной цели (при запросе от сервера к клиенту). Анализируется возможность принимать до десяти миллионов входящих сетевых пакетов с помощью программных средств физического сервера, что позволяет обеспечить стабильную обработку информации для бесперебойной работы при *DDoS*-атаках «*SYN*-флуда», у которых реализована возможность перегрузки многомиллионными сетевыми пакетами. Подобное количество входящих сетевых пакетов может привести к заполнению внешнего сетевого канала с последующим повышением нагрузочной способности сетевого TCP/IP стека, что перекрывает возможность зоны удаленного управления физическим сервером в кратчайшие сроки, а также нарушает работоспособность рабочей среды.

Ключевые слова: нагрузочная способность; *CPU*; параллелизация; параллелизация процессорной нагрузки; оптимизация; защита от *DDoS*-атак; фильтрация трафика; фильтрация вредоносного трафика.

DOI: 10.15514/ISPRAS-2016-28(1)-6

Для цитирования: Пальчевский Е.В., Халиков А.Р. Равномерное распределение нагрузки аппаратно-программного ядра в UNIX-системах. Труды ИСП РАН, 2016, том 28, вып. 1, с. 93-102.DOI: 10.15514/ISPRAS-2016-28(1)-6

1. Введение

При массированных *DDoS*-атаках в программном ядре происходят многочисленные сбои, ошибки и перегрузки [1, 2]. Это приводит к замедленной работе всего компьютера и происходит вызываету величение нагрузки на центральные процессоры [3]. Данные по мощности внешних сетевых угроз, за счет которых можно произвести сетевую перегрузку, представлены на рис. 1.

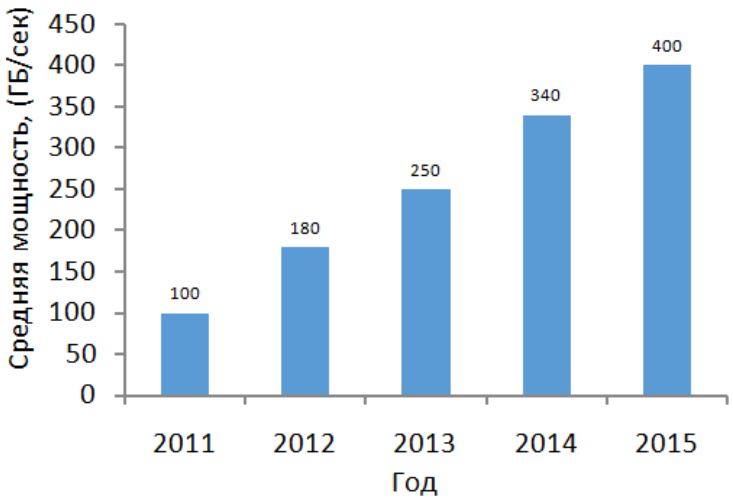

Рис.1. График DDoS-атак в годовом эквиваленте за 2011-2015 [4]
Fig. 1. Schedule of DDoS-attacks in annual terms for the 2011-2015 [4]

Параллелизация программного ядра – это равномерное распределение нагрузочной способности центральных процессоров по физическим ядрам сервера. Подобная операция дает возможность повысить производительность. Появляется возможность повышения устойчивости против разносторонней отправки вредоносного сетевого трафика, направленного с разных ЭВМ [4, 5].

2. Перераспределение нагрузки по физическим ядрам

Ядро – это централизованная часть операционной системы, которая обеспечивает различным программам и приложениям доступ к физическим ресурсам ЭВМ, а также обеспечивает взаимосвязь с сетевым стеком [6].

Для перераспределения нагрузки по ядрам необходимо включить технологию «*PROMISC*», за счет этого в сетевом стеке срабатывает ускорительный режим

для одного ядра. Становится возможным многократное увеличение входящих сетевых пакетов. Запуск производится командой «*ifconfig eth*6 *promisc*». При *DDoS*-атаке после выполненной процедуры возникает максимально возможная загруженность первого ядра. Это позволяет распределять нагрузки *TCP*/*IP*-стека по всем ядрам сервера.

Для того, чтобы организовать просмотр скорости (кбит/с) внешнего сетевого интерфейса нужно применить разработанный скрипт, который написан на языке программирования «*BASH*» [7]:

Блок-схема «*.sh*-скрипта» показана на рис. 2.

```
#!/bin/bash
 INTERVAL="1"  # update interval in seconds
 if [ -z "$1" ]; then
        echo
        echo usage: $0 [network-interface]
        echo
        echo e.g. $0 eth0
        echo
        echo shows packets-per-second
        exit
fi
 IF=$1
 while true
do
        R1=`cat /sys/class/net/$1/statistics/rx_packets`
        T1=`cat /sys/class/net/$1/statistics/tx_packets`
        sleep $INTERVAL
        R2=`cat /sys/class/net/$1/statistics/rx_packets`
        T2=`cat /sys/class/net/$1/statistics/tx_packets`
        TXPPS=`expr $T2 - $T1`
        RXPPS=`expr $R2 - $R1`
        echo "TX $1: $TXPPS pkts/s RX $1: $RXPPS pkts/s"
done
```

Пример использования вышеприведенного кода в скриптовом файле: «*bash /root/speed.sh eth0*». Файл «*speed.sh*» создается в системной папке «*root*» с помощью команды «*touch speed.sh*» со вставкой программных строк, которые приведены выше. В ответ, при *DDoS*-атаке, будет выведено сообщение о том, что скорость на внешнем интерфейсе около четырех миллионов пакетов в секунду. Количественная размерность сетевых пакетов зависит от настроек физического сервера.

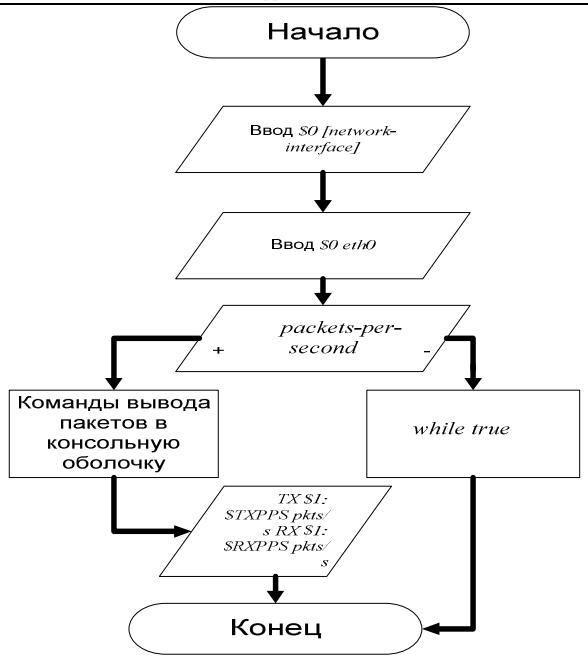

Рис. 2. Блок-схема вывода пакетов и увеличения пропускной способности.

Fig. 2. Block diagram of the output packets, and increase throughput.

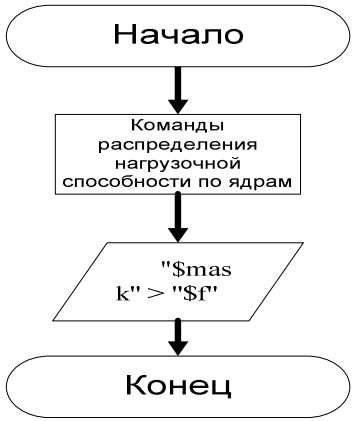

Рис. 3. Блок-схема распределения нагрузки по нескольким ядрам.

Fig. 3. A block diagram of load-balancing across multiple cores.

Для того, чтобы уменьшить нагрузку на первом ядре, был написан специальный скрипт на языке программирования «*BASH*», который позволяет реализовать возможность равномерного распределения нагрузки по всем физическим ядрам:

Блок-схема «*.sh*-кода» показана на рис. 3.

```
#!/bin/bash
ncpus=`grep -ciw ^processor /proc/cpuinfo`
test "$ncpus" -gt 1 || exit 1
n=0
for irq in `cat /proc/interrupts | grep eth | awk '{print $1}' | sed s/\://g`
do
  f="/proc/irq/$irq/smp_affinity"
  test -r "$f" || continue
  cpu=$[$ncpus - ($n % $ncpus) - 1]
  if [ $cpu -ge 0 ]
      then
        mask=`printf %x $[2 ** $cpu]`
        echo "Assign SMP affinity: eth queue $n, irq $irq, cpu $cpu, mask 0x$mask"
        echo "$mask" > "$f"
        let n+=1
  fi
done
```

После запуска вышеприведенного кода при атаке в $6x10^6$ входящих пакетов в секунду производительность *CPU* повысилась в несколько раз (за счет распределения нагрузочной способности на центральные процессоры), что представлено в таблице 1.

Табл. 1.- Нагрузка на ядра при применении оптимизации распределенной нагрузки на физический сервер

Table 2. The load on the core when applying optimization of distributed load on a physical server

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Нагрузка, %	55,1	52,5	62,5	62,5	57,7	47,7	55,9	61,4	55,1	52,5	62,5	62,5
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Нагрузка, %	57,7	47,7	55,9	61,4	55,1	52,5	62,5	62,5	57,7	47,7	55,9	61,4

С помощью двух скриптов «*BASH*» мы достигли повышенной производительности при принятии сетевых пакетов *TCP*/*IP* стеком.

3. Разработка и использование алгоритма для равномерной нагрузки на ядра

Для оптимизации аппаратно-программного ядра написан код на языке программирования *C++*, алгоритм которого предназначен для сдерживания атаки до $10x10^6$ пакетов в секунду, посредством равномерно распределенной нагрузки на физические серверы.

Многократное тестирование показало, что разработанный алгоритм не вызывает ошибок в ядре, фрагмент кода, отвечающий за переключения свитча в режим «*PROMISC*», приведен ниже. Режим «*PROMISC*» позволяет принимать сетевой плате все входящие пакеты, направленные различным адресатам, которые находятся на физическом сервере.

```
 struct packet_mreq sock_params;
memset(&sock_params, 0, sizeof(sock_params));
sock_params.mr_type = PACKET_MR_PROMISC;
sock_params.mr_ifindex = interface_number;
  int set_promisc = setsockopt(packet_socket, SOL_PACKET, PACKET_ADD_MEMBERSHIP, (void *)&sock_params, sizeof(sock_params));
if (set_promisc == -1) {
  printf("Can't enable promisc mode\n");
  return -1;   }
struct sockaddr_ll bind_address;
memset(&bind_address, 0, sizeof(bind_address));
bind_address.sll_family = AF_PACKET;
bind_address.sll_protocol = htons(ETH_P_ALL);
bind_address.sll_ifindex = interface_number;
struct tpacket_req3 req;
memset(&req, 0, sizeof(req));
req.tp_block_size = blocksiz;
req.tp_frame_size = framesiz;
req.tp_block_nr = blocknum;
req.tp_frame_nr = (blocksiz * blocknum) / framesiz;
req.tp_retire_blk_tov = 60; // Timeout in msec
req.tp_feature_req_word = TP_FT_REQ_FILL_RXHASH;
int setsockopt_rx_ring = setsockopt(packet_socket, SOL_PACKET , PACKET_RX_RING , (void*)&req , sizeof(req));
  if (setsockopt_rx_ring == -1) {
  printf("Can't enable RX_RING for AF_PACKET socket\n");
  return -1;   }
```

Операционная система на ядре *LINUX* версии «4.5.2» в вышеприведенном алгоритме увеличила показатели производительности фактически в 2.5 раза (см. таблицу 2). Тестирование проводилось на следующей конфигурации [8]:

- *CPU* 2 x *Intel Xeon* 5660 (в сумме 12 ядер / 24 потока по 2.8*GHz*, 12*Mb Cache*, 6.40 *GT/s*)
- *RAM* 48*Gb DDR*3-10600 *ECC REG*;
- *Intel S*3710 *SSDSC*2*BA*012*T*401.

Табл. 2. Оптимизированный вариант нагрузки на ядра сервера
Table. 2 - The optimized version of the load on the Server Core

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Нагрузка, %	25,1	22,5	32,5	32,5	27,7	17,7	25,9	31,4	25,1	22,5	32,5	32,5
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Нагрузка, %	27,7	17,7	31,4	25,9	31,4	25,1	22,5	32,5	32,5	27,7	17,7	31,4

4. Оптимизация «AF_PACKET» для решения проблемы с блокировкой сетевых пакетов

«*AF_PACKET*» – это режим, обладающий большим быстродействием и требующий наличие двух внешних сетевых интерфейсов. Он разрешает ядрам *UNIX*-подобных систем принимать и отправлять различные сокеты, что позволяет контактировать с внешней глобальной сетью. Главным компонентом работы является настройка системы в качестве шлюза. Такой алгоритм работы позволяет организовать полную блокировку входящих и исходящих пакетов вредоносного трафика при *DDoS*-атаке.

Чтобы избежать ложных блокировок и оптимизировать «*AF_PACKET*», рекомендуется подключить функцию «*FANOUT*», что поможет разделить входящий поток данных. Оптимальные значения рассчитаны и приведены в таблице 3.

В каждом ядре нужен как минимум один разветвитель входящих пакетов, чтобы распределить нагрузочную способность физического и логического потоков в равномерном порядке. Необходимость подключения двух разделителей в первом, двенадцатом и двадцать четвертом ядрах объясняется тем, что в операционных системах вида «*UNIX*» используется упорядоченная система нагрузки: при снижении производительности в первом ядре, загруженность которого будет приближаться к 80%, процесс в автоматическом режиме переносится на следующее ядро. На ядре под номером «двенадцать» заканчиваются физические ядра сервера по технологии «*HT*»: необходимое количество подключений «FANOUT» приравнивается к двум, с целью минимизировать нагрузочную способность на логические потоки, начинающиеся с номера «тринадцать».

Табл. 3. Расчет значений опции «FANOUT» для физических и логических ядер сервера
Table. 3. Calculation «FANOUT» option values for physical and logical server cores

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Число включений, ед.	2	1	1	1	1	1	1	1	1	1	1	2
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Число включений, ед.	1	1	1	1	1	1	1	1	1	1	1	2

Включение «*FANOUT*» производится добавлением истинного значения «*true*», к переменной «*use_multiple_fanout_processes*». Пример кода включения вышеописанного параметра: «*bool use_multiple_fanout_processes = true*».

В конечном итоге, оптимизировав «*AF_PACKET*», нагрузочная способность снизилась на несколько процентов и повлекло за собой повышенную продуктивную способность физического сервера справляться с сетевыми перегрузками, что показано в таблице 4.

Табл. 4. Нагрузочная способность логических и физических потоки сервера, при оптимальном режиме «AF_PACKET»
Table. 4. Load capacity logic and physical server threads with optimal «AF_PACKET

№ ядра	1	2	3	4	5	6	7	8	9	10	11	12
Нагрузка, %	20,1	18,5	27,5	19,5	24,7	13,7	15,9	26,4	22,1	20,5	29,5	30,5
№ ядра	13	14	15	16	17	18	19	20	21	22	23	24
Нагрузка, %	24,7	12,7	27,4	23,9	19,4	22,1	20,5	22,5	22,5	21,2	14,6	28,3

Наши результаты позволяют сделать вывод о эффективности параллелизации аппаратно-программного ядра в *UNIX*-системах посредством использования специализированных режимов, кодов и алгоритмов. Т.о. равномерная нагрузка на физических ядрах позволяет увеличить производительность физического сервера и повышает пропускную сетевую способность в несколько раз.

5. Заключение

В данной статье была рассмотрена возможность увеличения нагрузочной способности физического сервера для обеспечения отказоустойчивости к *DDoS*-атакам типа «*SYN*», в которых имеется многомиллионное сочетание сетевых пакетов. Это делает возможным в кратчайшие сроки заполнить канал и сетевой стек и вывести физический сервер из зоны удаленного обслуживания. Были приведены как примеры кодов на языках «*C++*» и «*BASH*», которые помогли организовать мульти-оптимизацию. Представлены

результаты тестирования разработанных алгоритмов. На начальной стадии получена возможность обработки сетевых пакетов, скорость поступления которых составляет порядка 4 миллионов в секунду. После применения двух специальных скриптов нагрузка на физические ядра снизилась и, за счет этого, стало возможным принимать сетевые пакеты до 6 миллионов в секунду. Разработанный алгоритм на языке высокого уровня «*C++*» позволил более эффективно снизить нагрузку на физические ядра сервера и обрабатывать сетевые пакеты до $10x10^6$ в секунду. В конечном результате представленный алгоритм может быть применим для сдерживания *DDoS*-атаки различных типов и видов.

Список литературы

[1]. Crist E.F., Keijser J.J. Mastering OpenVPN / E.F. Crist., Keijser J.J. – Изд-во: «Packt Publishing», – 2015. – 364 с.

[2]. Ватаманюк, А.К. Создание, обслуживание и администрирование сетей на 100% – Изд-во: «Питер», – 2010. – 350 с.

[3]. Dubrova E. Fault-Tolerant Design / E. Dubrova – Изд-во: «Springer», 2013. – 185 с.

[4]. Palchevsky E., Khalikov A. TCP/IP network STACK optimization under high load on UNIX-like systems – DSPTech'2015. Proceedings v.1. USATU, UFA, 2015. – С. 130-135.

[5]. Раго С., Стивенс У. UNIX. Профессиональное программирование, 3-е издание – С. Раго, У. Стивенс – Изд-во: «Санкт-Петербург», 2013. – 1104 с.

[6]. Krylov V., Kravtsov K. DDoS attack and interception resistance IP fast hopping based protocol – 23rd international conference on software engineering and data engineering, sede 2014. New Orleans, LA, 2014. – С. 43-48.

[7]. Блум Р., Бреснахэн К. Командная строка Linux и сценарии оболочки / Р. Блум, К. Бреснахэн – Изд-во: «Вильямс», 2013. – 784 с.

[8]. Е.В. Пальчевский, А.Р. Халиков. Техника инструментирования кода и оптимизация кодовых строк при моделировании фазовых переходов на языке C++. Труды ИСП РАН, том 27 (выпуск 6), 2015 г., стр. 87-96. DOI: 10.15514/ISPRAS-2016- 27(6)-5.

Uniformly distributed load of hardware and software core in the UNIX-based systems

E.V Palchevsky <teelxp@inbox.ru>
A.R Khalikov <khalikov.albert.r@gmail.com>
USATU, 450000, Russia, Ufa, st. Marx, 12

Abstract. In this article we consider the problem of maximizing the capacity of the network stack to the interaction of hardware and software core to ensure the stability of the physical server. The algorithms and program codes are proposed to optimize the load capacity of the CPU by core parallelization. The paper also considers statistics of improved power of

distributed attacks affecting the network infrastructure. It proved the impact of any application with access to the external global network to the production of the physical server presented in the form of physical resources. With the help of the developed and implemented the algorithm (in the language of «BASH»), produced by the distribution of the load capacity of the physical server cores, to further reduce the load capacity on the processing power of the CPU is provided. Showcased flowcharts, as well as the final test results of each stage of development are discussed. Implemented network optimization mode «AF_PACKET», which has given the opportunity to accept external network packets without any locks that, in turn, increases the efficiency of achievement of the set goals (upon request from the server to the client). The possibility of taking up to ten million incoming network packets by software physical server, which allows for stable processing of information for the smooth operation under DDoS-attacks «SYN-flood" who realized the possibility of overload multimillion network packets. A similar number of incoming network packets provides an opportunity to fill the external network channel, with a consequent increase in the load capacity of the network TCP / IP stack that covers the remote control area physical server as soon as possible. Also adversely affect the performance of the working environment.

Keywords: carrying capacity; CPU; parallelization; parallelization of CPU load; optimization; protection against DDoS-attacks; traffic filtering; filtering malicious traffic.

DOI: 10.15514/ISPRAS-2016-28(1)-6

For citation Palchevsky E.V. Khalikov A.R. Uniform load distribution of hardware and software core in the UNIX-based systems. Trudy ISP RAN /Proc. ISP RAS, 2016, vol. 28, issue 1, 2016. pp. 93-102 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-6

References

[1]. Crist E.F., Keijser J.J. Mastering OpenVPN / E.F. Crist., Keijser J.J. - Publishing house: «Packt Publishing», - 2015. - 364 p.

[2]. Vatamanyuk, A.K. Establishment, maintenance and administration of networks 100% / A.K. Vatamanyuk - Publishing house "Peter" - 2010. - 350 p.

[3]. Dubrova E. Fault-Tolerant Design / E. Dubrova - Publishing house: «Springer», 2013. - 185 p.

[4]. Palchevsky E., Khalikov A. TCP/IP network STACK optimization under high load on UNIX-like systems // DSPTech'2015. Proceedings v.1. USATU, UFA, 2015. – С. 130-135.

[5]. Rago C., Stevens W. UNIX. Vocational Programming, 3rd Edition - / C. Rago, W. Stevens - Publishing House: "St. Petersburg", 2013 - 1104.

[6]. Krylov V., Kravtsov K. DDoS attack and interception resistance IP fast hopping based protocol // 23rd international conference on software engineering and data engineering, sede 2014. New Orleans, LA, 2014. - S. 43-48.

[7]. Bloom R., Bresnahen K. Linux command line and shell / R. Bloom scenarios K. Bresnahen - Publishing House "Williams", 2013. - 784 p.

[8]. E.V. Palchevsky, A.R. Khalikov. Tehnika instrumentirovanija koda i optimizacija kodovyh strok pri modelirovanii fazovyh perehodov na jazyke C++ [Code instrumentation technique and optimization of lines of code in the simulation of phase transitions in the language C ++]. Trudy ISP RAN [Proceedings of ISP RAS], volume 27 (issue 6), 2015, pp. 87-96 (in Russian). DOI: 10.15514/ISPRAS-2015- 27(6)-5.