7. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. БХВ-Петербург. 2002. С. 329. 13.

8. Страница проекта BNB-Simulator https://github.com/fominandrey/bnb-simulator.

9. Evtushenko Y., Posypkin M., Sigal I. A framework for parallel large-scale global optimization //Computer Science-Research and Development. 2009. Т. 23. №. 3-4. С. 211-215.

10. Страница проекта BNB-Solver https://github.com/mposypkin/BNB-solver.

Е.В. Пальчевский, А.Р. Халиков

ПАРАЛЛЕЛИЗАЦИЯ НАГРУЗКИ АППАРАТНО-ПРОГРАММНОГО ЯДРА В UNIX-СИСТЕМАХ

(Уфимский государственный авиационный технический университет)

В данной статье рассматривается задача максимального увеличения пропускной способности сетевого стека с взаимодействием аппаратно-программного ядра для обеспечения стабильности работы физического сервера [1]. Анализируется возможность принимать до десяти миллионов входящих сетевых пакетов с помощью программных средств физического сервера, которая позволяет обеспечить стабильную обработку информации ля бесперебойной работы при *DDoS*-атаках.

При *DDoS*-атаках, в программном ядре, происходят многочисленные сбои, ошибки и перегрузки [2]. Это приводит к замедленной работе всей электронно-вычислительной машины и происходит повышение нагрузки на центральные процессоры. Данные по мощности внешних сетевых угроз, за счет которых можно произвести сетевую перегрузку, представлены на рисунке 1.

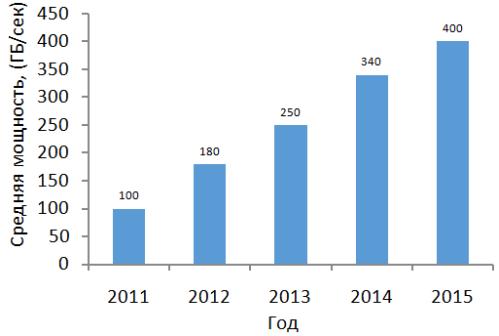

Рисунок 1 – График *DDoS*-атак в годовом эквиваленте за 2011-2015 [2]

Параллелизация программного ядра – это равномерное распределение нагрузочной способности центральных процессоров по физическим ядрам сервера. Подобная операция дает возможность повысить производительность [3].

Появляется возможность повышения устойчивости против разносторонней отправки вредоносного сетевого трафика, направленного с разных ЭВМ [4, 5].

1. Перераспределение нагрузки по физическим ядрам

Для перераспределения нагрузки по ядрам, необходимо включить технологию «*PROMISC*», за счет этого в сетевом стеке срабатывает ускорительный режим для одного ядра [6]. Становится возможным многократное увеличение входящих сетевых пакетов. Запуск производится командой «*ifconfig eth*6 *promisc*». При *DDoS*-атаке, после проделанной процедуры, возникает загруженность первого ядра на максимально возможную. 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, фрагмент которого написан на языке программирования «*BASH*» [7]:

```
#!/bin/bash
    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.

Таблица – 1 Нагрузка на ядра при применении оптимизации распределенной нагрузки на физический сервер

№ 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

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:

```
#!/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, повысилась производительность *CPU* в несколько раз (за счет распределения нагрузочной способности на центральные процессоры), что представлено в таблице 1.

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

2. 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.

Таблица – 2 Оптимизированный вариант нагрузки на ядра сервера

№ 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

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, которые направлены различным адресатам, находящемуся на физическом сервере.


```
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));
```

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 (машине):

- *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

После выполнения вышеперечисленных методов равномерная нагрузка на физических ядрах позволяет увеличить производительность физического сервера и повышает пропускную сетевую способность в несколько раз.

Вывод

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 выводит физический сервер из зоны удаленного обслуживания. Были приведены как примеры кодов на языках «*C++*» и «*BASH*», которые помогли организовать мульти-оптимизацию так, и представлено тестирование разработанных алгоритмов. На начальной стадии, получена возможность обработки сетевых пакетов, размерность которых составляет порядка 4 миллионов. 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 применяться для сдерживания *DDoS*-атаки различных типов и видов.

Литература

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

2. 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.

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

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

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.

С.О. Пономаренко, Р.Р. Диязитдинов

ТОЧНОСТЬ ОПРЕДЕЛЕНИЯ НАПРАВЛЕНИЯ НА ИСТОЧНИК СИГНАЛА В SMART-АНТЕННАХ

(Поволжский государственный университет телекоммуникаций и информатики)

В настоящее время разрабатываются устройства приема и способы предоставления связи, которые постепенно расширяют границы использования мобильных устройств, преодолевая эти препятствия. Одной из перспективных в коммерческом плане разработок являются Smart-антенны, теоретические основы которых были заложены не так давно – 10-15 лет назад.

«Умные» антенны являются разновидностью секторных антенн, имеющие ряд преимуществ. Проще говоря, основной принцип работы Smart-антенн – формирование диаграммы направленности в направлении полезного сигнала и глубокое подавление ДН в направлении помех (см. Рис.1). Их главная особенность заключается в том, что при особой структуре антенной решетки они могут формировать диаграмму направленности в зависимость от места положения абонента.За счет этих особенностей SMART-антенны выигрывают в сравнении с другими видами антенн.

Проектирование Smart-антенн представляет собой сложный комплексный процесс, при котором необходимо задействовать множество различных математических алгоритмов, а также использовать реальные устройства. Также существует ряд проблем связанных с настойкой Smart-антенн и их юстировкой. В результате этого, основным подходом к анализу и расчету Smart-антенн является компьютерное моделирование.

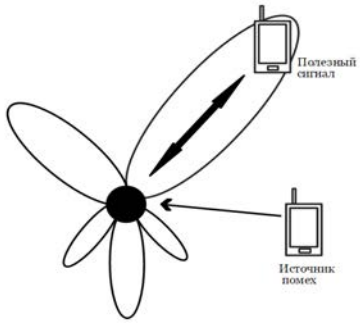

Рис.1 – Принцип работы Smart-антенн

Пусть Smart-антенна состоит из двух элементов, расстояние между которыми равно *d*. На антенну приходит два сигнала:

– $S(t)$ – полезный сигнал;

– $N(t)$ – мешающий сигнал.

Весовые коэффициенты для антенной решетки, состоящей из двух элементов:

$$w = \begin{pmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{pmatrix};$$

Зная весовые коэффициенты можно найти коэффициент усиления по формуле: $G = 10\lg\left(\frac{w^T \cdot a(\theta_S)^2}{w^H w}\right)$ [1].

В работе моделируется ошибка определения направления на источник сигнала. Для этого вводится отклонение угла прихода сигнала-помехи от номинального значения. Для определения характеристик было разработано модельное программное обеспечение, графический интерфейс которого представлен на рис. 2.

Отклонение угла прихода помехи приводит к тому, что весовые коэффициенты усиления не совпадают с весовыми коэффициентами для номинальных значений. А это в свою очередь ведет к появлению погрешностей в усилении сигналов (см. рис. 3).