УДК 004.724, 004.728

И. П. Иванов

ИНТЕГРАЛЬНАЯ ОЦЕНКА СОСТОЯНИЯ РЕСУРСОВ ПОЛЬЗОВАТЕЛЬСКОГО МАРШРУТА В КОРПОРАТИВНОЙ СЕТИ

Рассмотрены программные модули, реализующие интегральную оценку состояния ресурсов транспортной подсистемы корпоративной сети на маршруте пользователей. Приведены результаты тестирования разработанных модулей в корпоративной сети МГТУ им. Н.Э. Баумана.

E-mail: ivanov@bmstu.ru

Ключевые слова: *сеть, хост, нагрузка, трафик, ресурс, интерфейс.*

Принцип интегральной оценки состояния ресурсов на маршруте между источником информации и ее приемником в компьютерных сетях с помощью измерения Round Trip Time (RTT — время двойного оборота) реализован и широко используется в технологиях установления соединения с подтверждением (протокол Transmission Control Protocol стека TCP/IP) и в технологиях без установления соединения, но с подтверждением [1, 2]. В этих случаях все необходимые измерения могут выполняться на источниках информации с минимальной доступной для источника погрешностью, чем снимается весьма серьезная проблема синхронизации работы узлов сети. Для режимов передачи информации без подтверждения (User Datagram Protocol стека TCP/IP) интегральная оценка состояния ресурсов может быть выполнена утилитой *ping*, распознаваемой и выполняемой всеми аппаратно-программными модулями сетей TCP/IP [1–3], однако оценка состояния ресурсов на исследуемом маршруте может не соответствовать действительности из-за приоритета пакетов протокола Internet Control Message Protocol (ICMP — межсетевой протокол управляющих сообщений), которому принадлежат эхо-запрос и эхо-ответ стандартной утилиты [3, 4]. Еще одним недостатком утилиты *ping* является факт определения RTT между сетевыми уровнями взаимодействующих хостов. Для определения состояния ресурсов на маршрутах пользователей в настоящей работе предлагается мониторинг UDP-пакетами (рис. 1).

Один из компьютеров сети выполняет роль клиента для генерации (*1*) UDP-сегментов, отправляемых через сеть другому компьютеру (*2*), т.е. аналогично утилите *ping* посылается запрос в сеть. Этот

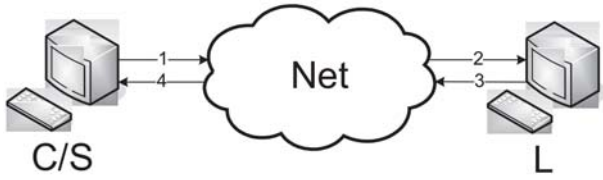

Рис. 1. Схема UDP-мониторинга

же компьютер выполняет роль сервера, обрабатывая поступающие (*4*) от запрашиваемого компьютера эхо-ответы, т.е. выполняет серверную часть обработки результатов получения пакетов. На рис. 1 этот хост обозначен C/S (Client/Server). Запрашиваемый компьютер (*2*), возвращающий (*3*) запросы клиента, обозначен на этом же рисунке буквой L (Loop). Генерируемый клиентом UDP-трафик запросов может меняться по интенсивности, для этого предусматривается возможность изменения числа передаваемых сегментов в секунду, размеров каждого сегмента и общего числа посылаемых в сеть сегментов. Для достижения минимальной погрешности все временные интервалы в хостах C/S и L измеряются в тактах процессора (аналогично экспериментам, описанным в работе [4]). На компьютере L происходит замена IP-адресов источника и приемника и возврат принятых пакетов, т.е. реализуется петля. Вернувшись на компьютер C/S, отправленная информация обрабатывается серверным S процессом. Серверный процесс обработки поступающей информации может быть запущен и на компьютере, осуществляющем возврат информации.

Пакет для тестирования производительности сети UDPPING состоит из трех модулей. Модуль UDPCLIENT отправляет согласно заданному профилю пакеты модулю UDPLOOP, который, в свою очередь, возвращает их модулю UDPSERVER. Модули UDPCLIENT и UDPSERVER работают на одном компьютере, а UDPLOOP — на другом.

Согласно заданным аргументам модуль UDPCLIENT отравляет в сеть пакеты, в каждый помещая номер пакета и время отравления, измеряемое в тактах процессора.

Запуск модуля осуществляется командой

udpclient <ipaddr> <port> <plen> <pps> <pcount>,

где <ipaddr> — IP-адрес или DNS — имя системы с модулем UDPLOOP, осуществляющей возврат пакетов; <port> — UDP-порт, прослушиваемый модулем UDPLOOP (9930 по умолчанию); <plen> — размер поля данных ($60 < plen <= 1472$); <pps> — число пакетов в секунду (при указании очень больших значений пакеты отправляются непрерывно); <pcount> — число пакетов в тесте ($2 < pcount <= 1000000$).

Текст программы модуля приведен на рис. 2.

```
int main(int argc, char *argv[]) {

   //Обработка аргументов для запуска модуля
   long pl, pps, pc;
   int lport;
   struct hostent *he;

   if (argc !=6) {
     printf("Usage: %s <ipaddr> <port> <plen> <pps> <pcount>\n",argv
[0]);
     printf("port=9930\n60<plen<=1472\n0<pps\n2<pcount<=1000000\n");
     exit(1);
   }
   if ((he=gethostbyname(argv[1]))==NULL) {
     printf("gethostbyname() error\n");
     exit(1);
   }
   lport=atoi(argv[2]);
   pl=atoi(argv[3]);
   pps=atoi(argv[4]);
   pc=atoi(argv[5]);

   //Вычисление числа тактов процессора между пакетами согласно
указанному pps
   u_int32_t low, high;
   u_int64_t v, oldv, delta;
   __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
   oldv = (low | ((u_int64_t)high << 32));
   sleep(1);
   __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
   v = (low | ((u_int64_t)high << 32));
   delta=(v-oldv)/pps;

   //Инициализация сокета для отправки пакетов и его файлового
дескрипторов
   struct sockaddr_in si_send;
   int s;

   if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1) exit(1);

   memset((char *) &si_send, 0, sizeof(si_send));
   si_send.sin_family = AF_INET;
   si_send.sin_port = htons(lport);
   si_send.sin_addr = *((struct in_addr *)he->h_addr);

 //Цикл отправки пакетов
    int ctry=0;          //счетчик попыток отправки пакетов
    struct sticks {              //структура пакета
          int npack;             //номер пакета
          u_int64_t ctick;       //время отправки из UDPCLIENT в
 тактах процессора
          u_int64_t ltick;       //время прохождения через UDPLOOP в
 тактах процессора

          u_int64_t stick;       //время приема в UDPSERVER в тактах
 процессора
    };
    char buf[BUFLEN];    //содержимое пакета
```

Рис. 2. Листинг текста программы модуля UDPCLIENT (см. также с. 51)

```
    for (int i=0; i<pc; i++) {

      //Заполнение двух первых полей пакета
      __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
      v=ctics[i]=(low | ((u_int64_t)high << 32));
      ((struct sticks *)buf)->npack=i;
      ((struct sticks *)buf)->ctick=v;

      //Отправка пакетов с фиксацией числа попыток
      ctry++;
      while (sendto(s, buf, pl, 0, (struct sockaddr*)&si_send, slen)
==-1) {ctry++;}

      //Выдержка паузы между пакетами согласно требуемому числу
пакетов в секунду
      while (v<(ctics[i]+delta)) {
        __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
        v=(low | ((u_int64_t)high << 32));
      }
    }

    //Отправка 10 пакетов, сигнализирующих об окончании передачи
    //Поле npack содержит недопустимый номер пакета
    //Поле ctick содержит действительное число оправленных пакетов
    #define NPACK 1000010

    for (i=0; i<10; i++) {
      ((struct sticks *)buf)->npack=NPACK;
      ((struct sticks *)buf)->ctick=pc;
      while (sendto(s, buf, pl, 0, (struct sockaddr*)&si_send, slen)
==-1) {}
    }
     //Окончание работы программы
    close(s);
    printf("Try %d packets, send %d packets\n", ctry, pc);
    return 0;
}
```

Рис. 2. (Окончание)

Модуль UDPLOOP принимает пакеты от UDPCLIENT, фиксирует в каждом пакете время прохождения (в тактах процессора) и отправляет пакет обратно на порт, прослушиваемый UDPSERVER. После передачи блока пакетов фиксируется число переданных пакетов и число попыток передачи. Запуск модуля осуществляется программой

udploop <listenport> <sendport>,

где <listenport> — UDP-порт, используемый модулем UDPLOOP для приема пакетов; <sendport> — UDP-порт, на который UDPLOOP отправляет пакеты модулю UDPSERVER.

Текст программы модуля приведен на рис. 3.

```
int main(int argc, char *argv[]) {

   //Обработка аргументов для запуска модуля
   int listenport, sendport;

   if (argc !=3) {
     printf("Usage: %s <listenport> <sendport>\n",argv[0]);
     exit(1);
   }

   listenport=atoi(argv[1]);
   sendport=atoi(argv[2]);

   //Вычисление числа тактов процессора в секунду
   u_int32_t low, high;
   u_int64_t v, oldv, tps;

   __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
   oldv = (low | ((u_int64_t)high << 32));
   sleep(1);
   __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
   v = (low | ((u_int64_t)high << 32));
   tps=v-oldv;

   //Инициализация сокетов для прослушивания udp порта (si_listen)
   //и отправки пакетов (si_send) и их файловых дескрипторов
   struct sockaddr_in si_listen, si_send;
   int sr, ss;

   if ((sr=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1) exit(1);

  memset((char *) &si_listen, 0, sizeof(si_listen));
   si_listen.sin_family = AF_INET;
   si_listen.sin_port = htons(listenport);
   si_listen.sin_addr.s_addr = htonl(INADDR_ANY);
   if (bind(sr, (struct sockaddr*)&si_listen, sizeof(struct
sockaddr))
==-1)
     exit(1);

   if ((ss=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1) exit(1);

   memset((char *) &si_send, 0, sizeof(si_send));
   si_send.sin_family = AF_INET;
   si_send.sin_port = htons(sendport);
```

Рис. 3. Листинг текста программы модуля UDPLOOP (см. также с. 53)

```
  //Определение состояний программы и перевод ее в состояние
ожидания
   #define WAIT 0
   #define READ 1
   #define STOP 2
   int state=WAIT;

   int lsend=ltry=0; //Обнуляем счетчики переданных пакетов и
попыток
передать пакет

   //Запуск бесконечного цикла обработки
   for(;;) {

     //Ожидание пакета
     recvlen=recvfrom(sr, buf, BUFLEN, 0, (struct
sockaddr*)&si_recv,
&slen);
     if (recvlen==-1) exit(1);

     //Определение адреса отправителя
     si_send.sin_addr=si_recv.sin_addr;

     //Проверка, является ли пакет финальным
     if (((struct sticks *)buf)->npack==NPACK) {

       //Если этому предшествовало состояние передачи тестовых
пакетов
       if (state==READ) {

         //Вывод результатов работы программы
        printf("================================\n");
         printf("Try %d packets, send %d packets\n", ltry, lsend);
         lsend=ltry=0; //обнуляются счетчики переданных пакетов и
попыток передать пакет
       }

       //Передача в финальных пакетах числа тактов процессора в
секунду на этом компьютере
       ((struct sticks *)buf)->ltick=tps;
       state=STOP; //Состояние программы - передача финальных
пакетов
     } else { //Пакет обычный

       state=READ; //Состояние программы - чтение пакетов

       //Фиксация в пакете времени прохождения через этот компьютер
       __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
       v=(low | ((u_int64_t)high << 32));
       ((struct sticks *)buf)->ltick=v;
     }
     //Отправка пакета на UDPSERVER
     while (sendto(ss, buf, recvlen, 0, (struct sockaddr*)&si_send,
slen)==-1);
   }
}
```

Рис. 3. (Окончание)

Модуль UDPSERVER принимает пакеты от UDPLOOP и по окончании передачи выполняет расчет:

— полученных пакетов в процентах от числа отправленных;

— числа тактов процессора на компьютерах UDPCLIENT/UDPSERVER (stps) и UDPLOOP (ltps);

— среднего времени возврата пакета от UDPCLIENT к UDPSERVER в тактах процессора (crtpavg);

— среднего времени между отправлением пакетов UDPCLIENT (cavg), прохождением пакетов через UDPLOOP (lavg) и возвращением пакетов UDPSERVER (savg);

— дисперсии времени между отправлением пакетов UDPCLIENT (cdisp), прохождением пакетов через UDPLOOP (ldisp) и возвращением пакетов UDPSERVER (sdisp).

Запуск модуля осуществляется командой

udpserver <listenport>,

где <listenport> — UDP-порт, используемый UDPSERVER для приема пакетов.

Приведем пример результата работы модуля UDPSERVER:

udpserver 9931

==========

Received 930 packets of 1000, 93.00

stps: 764676779 ltps: 764292646 comptics: 1.00

crtpavg: 17835047

cavg: 8218838 lavg: 8222883 savg: 8226646

cdisp: 14255103640951 ldisp: 32300226589229 sdisp: 23772342591085

Текст программы модуля показан на рис. 4.

```
//Организация массива данных для обработки тестовых блоков пакетов
struct sticks {                 //структура пакета
        int npack;              //номер пакета
        u_int64_t ctick;        //время отправки из UDPCLIENT в
тактах
процессора
        u_int64_t ltick;        //время прохождения через UDPLOOP в
тактах
процессора
        u_int64_t stick;        //время приема в UDPSERVER в тактах
процессора
};
#define NPACK 1000010
struct sticks ticks[NPACK];

int main(int argc, char *argv[]) {
```

Рис. 4. Листинг текста программы модуля UDPSERVER (см. также с. 55–57)

```
   //Обработка аргументов для запуска модуля
   int listenport;

   if (argc !=2) {
     printf("Usage: %s <listenport>\n",argv[0]);
     exit(1);
   }

   listenport=atoi(argv[1]);

   //Вычисление числа тактов процессора в секунду
   u_int32_t low, high;
   u_int64_t v, oldv, stps;

   __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
   oldv = (low | ((u_int64_t)high << 32));
   sleep(1);
   __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
   v = (low | ((u_int64_t)high << 32));
   stps=v-oldv;

   //Определение состояний программы и перевод ее в состояние
ожидания
   #define WAIT 0
   #define READ 1
   #define STOP 2
   int state=WAIT;

   //Запуск бесконечного цикла обработки блоков тестовых пакетов
   for(;;) {

     //Инициализация сокета для прослушивания порта и ассоциация его
с файловым дескриптором
     struct sockaddr_in si_me;
     int s;

     if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP))==-1)
       exit(1);

     memset((char *) &si_me, 0, sizeof(si_me));
     si_me.sin_family = AF_INET;
     si_me.sin_port = htons(SPORT);
     si_me.sin_addr.s_addr = htonl(INADDR_ANY);
     if (bind(s, (struct sockaddr*)&si_me, sizeof(struct sockaddr))
==-1) exit(1);
     //Обнуление счетчика пакетов
     int i=0;

     //Цикл обработки блока тестовых пакетов
     while (i<NPACK) {
       //Получение пакета и помещение его в буфер
       struct sockaddr_in si_other;
       int slen=sizeof(si_other);
       char buf[BUFLEN];
       if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr*)&si_other,
&slen)==-1) exit(1);
       //Проверка, является ли пакет финальным
       if (((struct sticks *)buf)->npack==NPACK) {
```

Рис. 4. (Продолжение)

```
 //Если этому предшествовало состояние передачи тестовых пакетов
         if (state==READ) {

          //Фиксируется число полученных пакетов
          int recpackets=i;

           //Из пакета извлекается число тактов процессора на
компьютере с UDPLOOP
           int ltps=((struct sticks *)buf)->ltick;

          //Из пакета извлекается информация о числе пакетов
переданных
UDPCLIENT
          int sendpackets=((struct sticks *)buf)->ctick;

          //Фиксируется состояние окончания обработки тестовой серии
пакетов
           state=STOP;

           //Цикл обработки тестовой серии пакетов прерывается
           break;
         }
       } else { //Пакет обычный
        state=READ; //Состояние программы - чтение пакетов

         //Фиксирование параметров пакета в массиве статистической
информации
         ticks[i].npack=((struct sticks *)buf)->npack;
         ticks[i].ctick=((struct sticks *)buf)->ctick;
         ticks[i].ltick=((struct sticks *)buf)->ltick;
         __asm __volatile("rdtsc" : "=a" (low), "=d" (high));
        ticks[i].stick=(low | ((u_int64_t)high << 32));

        i++; //Увеличение счетчика пакетов
       }
     }
     close(s); //Закрытие сокета

     //приведение тактов процессора в системах UDPCLIENT/UDPSERVER и
UDPLOOP
     //к общему знаменателю
     int comptics=(0.0+stps)/ltps;

     //Использование полученного знаменателя в массиве
статистических
данных
     for(i=0;i<recpackets;i++) ticks[i].ltick=(u_int64_t)
(comptics*ticks[i].ltick;

     //Расчет среднего времени возврата пакета от UDPCLIENT к
UDPSERVER в тактах процессора
     u_int64_t crtpsum=0;
     for(i=1;i<=recpackets;i++) {
       crtpsum+=(ticks[i].stick-ticks[i].ctick);
     }
     u_int64_t crtpavg=(u_int64_t)crtpsum/recpackets;
```

Рис. 4. (Продолжение)

```
    //Расчет среднего времени между отправлением пакетов UDPCLIENT
(cavg)
     //прохождением пакетов через UDPLOOP (lavg) и
     //возвращением пакетов UDPSERVER (savg)
     u_int64_t csum=lsum=ssum=0;
     for(i=1;i<recpackets;i++) {
       csum+=(ticks[i].ctick-ticks[i-1].ctick);
       lsum+=(ticks[i].ltick-ticks[i-1].ltick);
       ssum+=(ticks[i].stick-ticks[i-1].stick);
     }
     cavg=(u_int64_t)csum/(recpackets-1);
     lavg=(u_int64_t)lsum/(recpackets-1);
     savg=(u_int64_t)ssum/(recpackets-1);

     //Расчет дисперсии времени между отправлением пакетов UDPCLIENT
(cdisp),
     //прохождением пакетов через UDPLOOP (ldisp) и
     //возвращением пакетов UDPSERVER (sdisp)
     csum=lsum=ssum=0;
     for(i=1;i<recpackets;i++) {
       csum+=((ticks[i].ctick-ticks[i-1].ctick)-cavg)*((ticks
[i].ctick-ticks[i-1].ctick)-cavg);
       lsum+=((ticks[i].ltick-ticks[i-1].ltick)-lavg)*((ticks
[i].ltick-ticks[i-1].ltick)-lavg);
       ssum+=((ticks[i].stick-ticks[i-1].stick)-savg)*((ticks
[i].stick-ticks[i-1].stick)-savg);
     }
     cdisp=csum/(recpackets-1);
     ldisp=lsum/(recpackets-1);
     sdisp=ssum/(recpackets-1);
    //Вывод результатов работы программы
     printf("=====================================\n");
     printf("Received %d packets of %d, %.2f\n",
        recpackets, sendpackets, (float)recpackets/sendpackets*100);
     printf("stps: %llu ltps: %llu comptics: %.2f
\n",stps,ltps,comptics);
     printf("crtpavg: %llu\n", crtpavg);
     printf("cavg: %llu lavg: %llu savg: %llu\n", cavg, lavg, savg);
     printf("cdisp: %llu ldisp: %'llu sdisp: %llu\n", cdisp, ldisp,
sdisp);

   }
}
```

Рис. 4. (Окончание)

Поскольку участвующие в процессе UDP-обмена информацией компьютеры могут различаться производительностью, то при обработке следует учитывать различные тактовые частоты их процессоров. Серверный процесс (S) может быть запущен и на компьютере, осуществляющем отправку эхо-ответов, что даст возможность сравнить параметры прибываемых и отправляемых потоков информации на обоих компьютерах. Это позволит более тщательно исследовать дуплексный режим взаимодействия двух хостов через сеть. Если в качестве IP-адреса назначения использовать loopback (127.0.0.1), то можно исследовать процесс генерации пакетов и обработать их без задействования ресурсов сети и уровня сетевых интерфейсов стека

TCP/IP самого компьютера (канального и физического уровней модели ISO/OSI). Все три процесса C, S и L можно запустить на одном хосте и в качестве IP-адреса получателя указать IP-адрес источника. Это дает возможность включить в процесс UDP-мониторинга RTT ресурсы канального и физического уровней хоста и минимальные ресурсы сети (интерфейса коммутатора, к которому подключен данный хост, однако при этом требуются изменения и перекомпиляция отдельных блоков операционной системы). Параллельный запуск нескольких пар C/S процессов на одном хосте с IP-адресами 127.0.0.1 (loopback) дает возможность тестировать компоненты операционной системы хоста, обеспечивающие функционирование стека TCP/IP. Аналогично может быть осуществлено тестирование минимальных ресурсов сети (интерфейса ближайшего коммутатора) и сетевого адаптера хоста.

Более того, клиентский процесс позволяет, меняя интенсивность отправки пакетов, изменять интенсивность трафика не только на маршруте между процессами C и L, но даже в том случае, если процесс L не запущен на хосте с IP-адресом получателя, хотя в этом случае дуплексная связь нагружается только в одном направлении. Таким образом, можно искусственно нагружать исследуемую сеть на всех дугах покрывающего дерева (соответствующего определенной виртуальной сети), если в качестве IP-адреса приемника указать широковещательный IP-адрес. Иными словами можно использовать предлагаемые компоненты приведенных программ для искусственной организации широковещательных штормов и штормов на маршруте от C/S процесса (хоста) до L процесса (хоста) при исследовании ресурсных возможностей отдельных участков сети.

Для проверки возможностей разработанного программного обеспечения была создана экспериментальная сеть, топология которой представлена на рис. 5.

Сеть включает в себя два коммутатора — SW1 и SW2 — и три хоста: H1, H2 и H3. Два хоста (H1 и H2) подключаются к интерфейсам коммутатора SW1, а третий — к интерфейсу коммутатора SW2. Коммутаторы также связываются между собой. В качестве хостов при проведении

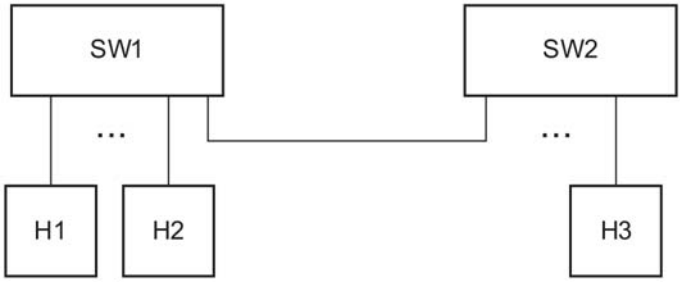

Рис. 5. Топологическая схема экспериментальной сети

экспериментальных исследований использовались компьютеры в следующей конфигурации: процессор Intel Celeron с тактовой частотой 764,2 МГц, объем жесткого диска 20 Гбайт, тип подключения жесткого диска IDE UDMA/33, сетевой адаптер на базе чипсета Intel 8280 IBA/CAM Pro 100/Ethernet, операционная система FreeBSD Release 6.1 patch-level 15. Коммутация в сети осуществлялась ненагруженным коммутатором Cisco Catalyst 2950 (SW1) и функционирующим в корпоративной сети МГТУ им. Н.Э. Баумана коммутатором Cisco Catalyst 6500 (SW2). На всех хостах экспериментальной сети были установлены три программных модуля.

Первый из запланированных экспериментов был предназначен для определения среднего значения RTT между хостами H1 и H3 при отсутствии иных потоков информации между всеми указанными хостами. В результате средним значением оказалась величина порядка 270000 тактов процессора, что соответствует длительности 353 мкс. В последующих экспериментах между хостами H2 и H3 возбуждался UDP-трафик с различной интенсивностью, что меняло степень загрузки сегмента между коммутаторами и граничного сегмента между интерфейсом коммутатора и хостом H3. Оказалось, что даже в случае 100 %-ной загрузки маршрута между H2 и H3 (порядка 12500 Кбайт/с, что соответствует технологии 100BaseT, 100 Мбит/с) и обязательного столкновения передаваемых от хостов H1 и H2 на хост H3 кадров в коммутаторе SW1 (блокировка впереди стоящим) измерение RTT дает те же значения (в пределах разброса из-за случайных факторов), что и для ненагруженного маршрута до тех пор, пока нагрузка потока измеряющего хоста H1 на интерфейс коммутатора SW1 не станет соизмеримой с такой же нагрузкой на свой интерфейс того же коммутатора от хоста H2. Это объясняется тем, что в коммутаторах Cisco Catalyst в буферном пространстве организуется две и более очереди в случае, если нескольким интерфейсам требуется коммутация с каким-либо другим (одним и тем же) интерфейсом, т.е. двум и более входным кадрам, прибывшим по разным портам коммутатора, практически одновременно понадобится один и тот же выходной порт. В соответствии с дисциплиной WFQ предпочтение всегда отдается менее длинной очереди кадров, т.е. кадр исследующего потока будет передан первым на выходной порт коммутатора SW1 и далее по всему маршруту в ущерб исследуемому потоку кадров. Лишь при интенсивности потока исследующих кадров, приводящей к сравнимой с исследуемым потоком загрузке буферного пространства, возможна заметная их задержка, отражаемая в увеличении RTT. Возникает парадоксальная ситуация, при которой, для того чтобы определить степень загрузки какого-либо сегмента в сети, мы должны нагрузить этот сегмент трафиком, сравнимым по интенсивности с уже имеющимся.

При этом реальная нагрузка выявляется с высокой достоверностью только при условии превышения суммой потребных пропускных способностей исследуемого и исследующего трафиков предельной пропускной способности сегмента. Эта ситуация невозможна при дисциплине обслуживания FIFO, которая практически не используется в современных информационно-коммуникационных технологиях. Для любых иных алгоритмов обслуживания очередей непосредственное измерение RTT в маршруте между источником и стоком информации приводит к неоправданному росту накладных расходов (около 100 %) и может быть рекомендовано для режимов передачи информации без установления соединений и без подтверждения только при максимальном размере зондирующих кадров, не получающих приоритетов при их коммутации. Для тех же режимов передачи, где так или иначе предусмотрены кадры с подтверждением, измерение RTT вполне допустимо из-за малости накладных затрат (т.е. реальный трафик не искажается). Если использовать модуль UDPCLIENT для создания искусственной нагрузки на маршрутах корпоративной сети, то учет приоритетности обслуживания требует выполнять загрузку маршрута пакетами минимального размера.

СПИСОК ЛИТЕРАТУРЫ

1. Таненбаум Э. Компьютерные сети. – СПб.: Питер, 2009.
2. Олифер В. Г., Олифер Н. А. Компьютерные сети. – СПб.: Питер, 2007.
3. TCP/IP: Архитектура, протоколы, реализация (включая IPv6 и IP Security) / Д-р Синди Фейт. – М.: Лори. – 2009.
4. Иванов И. П. Математические модели коммутаторов локальных вычислительных сетей // Вестник МГТУ им. Н.Э. Баумана. Сер. “Приборостроение”. – 2009. – № 2. – С. 84–92.
5. Разработка методов и алгоритмов управления интенсивностью трафика в коммутируемых сегментах корпоративной сети / М. Бойченко и др. // Отчет о НИР 1.10.09 – М.: МГТУ им. Н.Э. Баумана, 2009.

Статья поступила в редакцию 26.04.2009

Игорь Потапович Иванов родился в 1955 г., окончил в 1979 г. МВТУ им. Н.Э. Баумана. Проректор по информатизации, заведующий кафедрой “Теоретическая информатика и компьютерные технологии” МГТУ им. Н.Э. Баумана. Канд. техн. наук, доцент. Автор более 30 научных работ в области информационно-коммуникационных технологий.

I.P. Ivanov (b. 1955) graduated from the Bauman Moscow Higher Technical School in 1979. Ph. D. (Eng.), vice-rector on informatization, head of “Theoretical Bases of Information and Computer Technologies” department of the Bauman Moscow State Technical University. Author of more than 30 publications in the field of data and communication technologies.