Multimedia Tools and Applications, 24, 5–28, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Packet Video Broadcasting with General-Purpose Operating Systems in an Ethernet

EDUARDO MAGAÑA emagana@eecs.berkeley.edu
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720, USA

JAVIER ARACIL javier.aracil@unavarra.es
JESÚS VILLADANGOS jesusv@unavarra.es
Departamento de Automática y Computación, Universidad Pública de Navarra, Pamplona, Spain

Abstract. Video transmission with general-purpose PCs poses a number of requirements that radically differ from those of high-end dedicated video servers. We analyze the scenario of an Ethernet local area network in which a number of PCs are transmitting video streams, while other TCP/IP applications are also running concurrently. Our findings show that since the operating system clock resolution cannot cope with the transmission timing requirements the following holds: if the video transmission is performed with exact timing accuracy to maintain a constant rate then CPU load grows to 100%, thus blocking the PC for other user applications; on the other hand, if transmission is performed in a bursty manner, i.e. with sleep system calls, then CPU load decreases dramatically but the increased burstiness of the video stream has a negative impact on network performance (for example, capture effect in the Ethernet). Furthermore, the impact of video transmission over the rest of TCP/IP applications running on the same network depends heavily on the packet size. We provide an integrated analysis of operating system and network parameters to achieve video broadcasting while preserving timing requirements and minimizing the impact on other applications.

Keywords: packet video broadcasting, general-purpose video servers, Ethernet

1. Introduction

The increasing demand for multimedia services is spurring packet video transmission in the wide area and local area networks. Not only high-resolution digital TV is becoming more popular but also other services such as low to medium-rate packet video services for tele-conferencing, tele-education and tele-medicine applications. Our case study consists of a general-purpose PC connected to an Ethernet and acting as a video server. Such scenario is consistent with the technological trend towards increasing PC processing power, which, together with the availability of cost-effective hardware for coding/decoding video streams, is making *video at the desktop* a reality. For instance, a standard PC can effectively perform MPEG video coding with inexpensive hardware while decoding at the client can be achieved by software. The data rate for such video coding/decoding schemes, in the order of Mbps for MPEG1, provides a subjective quality ideally suited for a number of multimedia applications. On the other hand, most local area networks today provide sufficient bandwidth for multimedia traffic transport. Additionally, the broadcast nature of such LANs, for example the Ethernet, greatly simplifies the deployment of multimedia services, which normally require point-to-multipoint connectivity.

Packet video broadcasting poses significant challenges for both video server and transport network. However, the challenges imposed by both server and network may differ to a large extent depending on the broadcasting scenario. Traditionally, the research effort has been focused on dedicated video server engineering. Indeed, video transmission with dedicated servers is an area of intensive research, and there is a vast literature concerning admission control [1], traffic models [8], traffic shaping [4] and other video server engineering aspects [6]. Since dedicated servers perform video broadcasting to a large number of users, the challenge is to make efficient use of the available hardware resources (HD, memory, I/O buses) in order to serve as many users as possible and the same applies to the high-speed transport network. However, we note a fundamental difference in comparison to our case study: while high-speed video broadcasting demands dedicated resources, low-rate packet video consumes shared resources since both server and network are not devoted exclusively to the video application. Precisely, a distinguishing feature of this paper is the analysis of the impact of video broadcasting on other concurrent applications running on the same server and using the same LAN. In fact, multimedia applications spawn a number of concurrent processes besides raw video servers. Such processes are in charge of signaling aspects, bulk data transfer for shared document editing, electronic bulletin boards, etc. Thus, the challenge is to provide video service with a minimal impact on concurrent applications. Most importantly, we note that the transport network, at least in the access segment, is also a shared resource, since users are normally attached to a broadcast LAN in most corporate and academic environments. As a conclusion, we consider a case study in which the video application is just another process in a shared system, from both server and network standpoint. Figure 1 shows the dedicated server scenario in comparison to our case study of shared multimedia terminals.

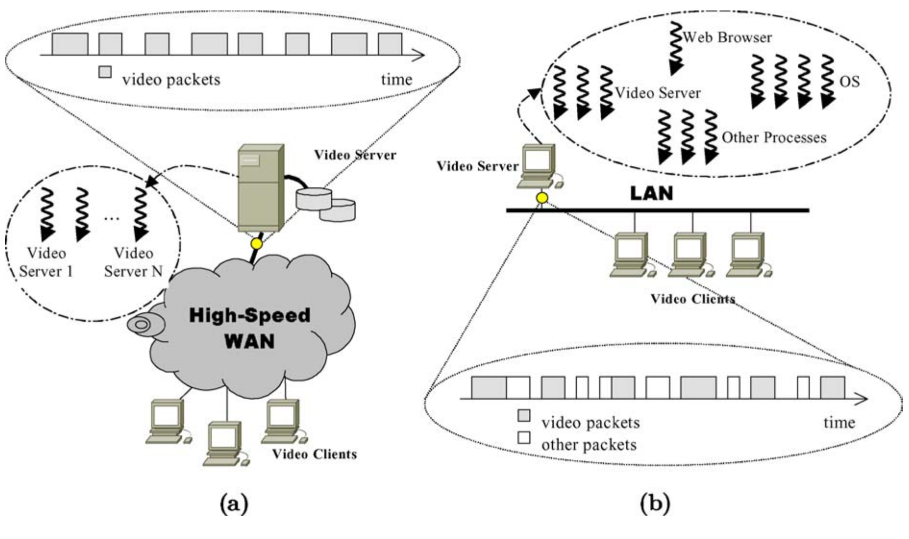


Figure 1. Dedicated server (a) versus general-purpose PC (b).

Our research serves to identify the pros and cons of ad-hoc solutions for video broadcasting. In fact, ad-hoc solutions are common in the software market since multimedia capabilities are being incorporated massively to current operating systems, with little changes in the kernel algorithms and system hardware, based on belief that the increased network bandwidth and CPU power will suffice for video broadcasting at low-medium rates. In this paper we actually identify fundamental trade-offs that must be specifically addressed when video broadcasting in a general-purpose system is a requisite.

For example, emerging services such as teleducation require voice, video and data processing capabilities from the user terminal. Moreover, the terminal cost should be low in order to cost-effectively offer service to a large student population. Thus, the PC is the platform of choice for many teleducation applications [11]. Nevertheless, the trade-offs presented in this paper clearly indicate that changes must be performed to the actual PC architecture, that are also examined in the paper. It must be stressed that our research is targeted to the general-purpose scenario, meaning that the video server will not be devoted to video broadcasting solely, but also to other processes running concurrently. Furthermore, communication resources are not reserved in advance but, instead, video traffic will be sharing the network with IP traffic. This is a common scenario for multimedia and *video at the desktop* applications running on a LAN. Furthermore, since an extraordinary increase in video services demand is foreseen, video applications are expected to play an essential role in the user personal computer and communication network. The lessons learned from our analysis help to provide design guidelines for both operating system and network, that overcome the limitations of current technology and set the pace for well-engineered systems.

While the scenario depicted in figure 1(a) applies to a number of broadcast networks such as HFC or wireless networks this paper only considers the Ethernet case, a most popular LAN (figure 1(b)). Our findings show an interesting and fundamental trade-off: if video transmission is performed in a non-aggressive manner for the rest of data applications then CPU load increases to nearly 100%. Thus, the PC is blocked for other user applications and strictly devoted to the video transmission. If otherwise the transmission clock requirements are relaxed, thus alleviating the CPU burden at the server, then the increased traffic burstiness severely affects the rest of TCP/IP applications. However, such effect is not as significant as the CPU overload produced by the non-aggressive transmission mode.

The rest of the paper is organized as follows: in Section 2, we report on the dynamics of packet video transmission over the Ethernet. Next, the impact of video transmission in a general-purpose server architecture is analyzed in Section 3. Section 4 presents two different transmission modes, accurate and bursty, together with a performance analysis including empirical results on a real network configuration. Section 5 is devoted to results and discussion, followed by the conclusions that can be drawn from this study.

2. Broadcasting continuous media over the Ethernet

Ethernets are the most extended local area networks nowadays, their success being based on the simplicity and ease of hardware implementation, which makes Ethernets straightforward to deploy and very cost-effective. The Ethernet is based on the well-known CSMA/CD protocol, in which all stations gain access to the network in contention. Although simple and cost-effective, the Ethernet cannot offer deterministic guarantees for bandwidth, delay or jitter, as opposed to Token Ring or ATM networks. Most importantly, transmission of bursty streams over the Ethernet may lead to the capture effect, that we describe next.

The retransmission delay after the occurrence of a collision is determined by the Binary Exponential Backoff algorithm (BEB). If subsequent transmission attempts from any of the colliding hosts also result in collisions, the BEB algorithm provides longer retransmission delays. As the network load increases, collisions become more likely, so the backoff time also increases rapidly. Consider that two stations collide so that the BEB algorithm imposes a certain retransmission delay, during which a third host attempts transmission successfully and takes over the channel for continuous transmission. Should any of the two colliding stations transmit again after backoff time expiration, the chance of collision with the third one is very high. However, since the number of consecutive collisions for the latter is only one then the backoff time given by the BEB algorithm will be lower. As a result, the third station gains access to the channel again, thus capturing the network in an unfair manner. After the occurrence of a third collision the third station delays retransmission for a relatively short time again, since the BEB algorithm reinitializes upon successful transmission. However, the first and second stations experience longer delays since the number of consecutive collisions is now three. As a result one of the stations is capturing the network while the others are neglected transmission. Such unfair effect has been called capture effect [12]. While some methods have been proposed to reduce the capture effect like the Capture Avoidance BEB (CABEB) [10] and Binary Logarithmic Arbitration Method (BLAM) [7], the BEB is the current standard. The capture effect is particularly harmful when there is a server dumping traffic into the network in a continuous manner. Such host will surely monopolize the network bandwidth, preventing other hosts from transmission.

We note that since the network load produced by video transmission is high and the server is likely to transmit in a continuous manner chances are that the rest of applications suffer a capture effect on the same LAN. In the Ethernet scenario, few papers have already analyzed the issue of video transmission with background data traffic [2, 5], focusing on audio/video quality and obtaining the maximum number of video streams that can be simultaneously transmitted. However, the impact of the video transmission over the rest of generic TCP/UDP applications has not been studied in detail yet. We note that due to the broadcast nature of the Ethernet, the usual applications such as WWW, distributed file sharing (NFS), etc. will surely be affected. Therefore, it is of primary importance to perform video transmission in a non-aggressive manner, so that the impact over the rest of applications is minimized while the video timing requirements are preserved.

3. Impact of video service in general-purpose PCs

As noted before, video transmission with general purpose PCs should be provided with minimal impact over the CPU performance, that is being used for other applications. While dedicated video servers are usually limited by input/output to hard disk, the case is different for a general-purpose PC, in which only a few streams are transmitted together. For instance, the prototype system under analysis incorporates the MPEG1 standard at different rates ranging from 1.5 to 6 Mbps. Such video rates allow for compression and decompression performed by inexpensive hardware, still keeping an acceptable quality for multimedia applications. Furthermore, we assess that the bottleneck is not in the system hardware (hard disk, I/O buses), as happens in dedicated video servers. While specialized file systems are required for the latter, video transmission at few Mbps does not require input/output capabilities other than those provided by standard IDE disks and PCI I/O buses.

The design challenges in our scenario come from the fact that the video transmission should not take over the system CPU, so that the PC is not blocked for the rest of the applications, while hard disk and I/O to video card and network interfaces card are of less concern in this type of video servers. We claim that large CPU resources are consumed in the video server mainly due to the timing mismatch between operating system clock and network clock. In order to fully understand this phenomenon, let us provide a quick glimpse on operating system timing issues.

3.1. Operating system timing issues

Process scheduling is a most important issue for multitask operating systems. In an overly simplistic view, a process can be either in the sleep state or in the active state. A kernel part called the scheduler is in charge of determining which process should enter the CPU or be removed from CPU. Such transitions to the active state from the sleep state and vice versa take place at regular instants which are determined by the operating system clock.[1] The OS clock provides the system heartbeat, so that the kernel is woken up at regular time intervals in order to perform system housekeeping, namely determine which processes should be run in the next time slice and other tasks such as file system maintenance and so on. Note that the kernel tasks consume CPU resources since a context switching is necessary from the active process to the kernel processes and back either to the active process or to another process awaiting for service. We stress the difference between operating system clock and hardware clock, which is the physical clock that drives the CPU program execution. Usually, the OS clock granularity is in the order of millions of hardware clock ticks (for instance 0.01 seconds or approximately 4.5 M hardware clock cycles in a 450 MHz Pentium III with Linux operating system). Note that such OS clock granularity cannot be made arbitrarily small since otherwise the most part of CPU time would be devoted to kernel tasks. On the other hand, note that a user process cannot be put to sleep for less than an OS clock tick. Such procedure is performed with the *sleep()* system call.

3.2. Network clock and OS clock

Since video is a delay and jitter sensitive service it would be desirable to transmit and receive data with strict timing. However, since video packets are encapsulated in UDP/IP protocols and transmitted through the PC network interface card timing is performed by the CPU. Since OS clock granularity is far too coarse for data transmission (usually 10 ms) there is no chance to provide a constant rate stream by putting the video server to sleep between consecutive packet transmissions. Alternatively, accurate timing can be provided by means of active waiting on the hardware clock. Namely, the video server reads the value of the hardware clock continuously, thus always remaining in the active state. As a result, the video transmission server, which is one among the many applications run in the PC, takes over the CPU, leading to starvation of the rest of the applications and noticeable slowdown in overall PC performance.

We note that the timing adaptation between the system clock and the network clock is of fundamental importance. A general-purpose operating system provides a system clock whose granularity allows for task scheduling in the kernel, but is not suitable for data transmission. Since the video server will necessarily use the socket layer for communications, normally over UDP—or the DLPI link layer services—the transmission clock is provided by the operating system. Instead of providing strict timing, i.e. identical packet inter-arrival times, we may allow some jitter so the video server can be put to sleep. Such procedure necessarily leads to bursty transmission in the Ethernet but provides considerable savings in CPU load.

4. Burstiness—CPU load trade-off analysis

If the OS clock cycle duration exceeds the packet interarrival time it turns out that the only way to achieve constant rate transmission is through active waiting. Alternatively, we may transmit several packets in a burst per OS clock cycle. Thus, we note that we may use different video transmission modes that balance CPU load and network impact. Specifically, we consider the following transmission modes:

– **Accurate mode:** packets are released with the most precise timing that the CPU can deliver. In order to do so, the video server is performing continuous polling over the hardware clock in order to transmit packets right on time. As a result, the processing of system calls that results from polling increases CPU utilization severely, while the traffic stream is constant rate. The accurate mode algorithm in pseudo-code is:

```
to = packet interarrival time
tIRQ = OS clock cycle time
while (1)
   {
   tnow=read(hardware clock)
   if ( to > tIRQ )
      sleep(floor(to/tIRQ)*tIRQ)  /* Process is put to
                                     sleep. CPU load 0% */
   while ( read(hardware clock)-tnow < to )
      {
      /* Active waiting loop.
      CPU load 100% */
      }
   transmit(packet)
   }
```

The *read(hardware clock)* is normally performed using the UNIX system call *gettimeofday()* while *transmit(packet)* uses the socket library call *sendto*. Since *read(hardware clock)* is executed continuously the accurate mode leads to 100% CPU load.

– **Bursty mode:** the video server is put to sleep between subsequent packet transmissions in order to save CPU power. Since the *sleep()* or *nanosleep()* system call both have coarse resolution (in the other of 10 milliseconds) then transmission is necessarily bursty in other to meet the video stream rate requirements. Indeed, we note that the *sleep()* system call can only be performed if the packet interarrival time is larger than the OS clock cycle, which is normally infeasible. For example, packet interarrival time is in the order of milliseconds with MTU size packets (1500 bytes) for a case with eight low-rate 1.5 Mbps streams, one order of magnitude below OS clock cycle granularity. The pseudo-code for the inaccurate mode algorithm is:

```
N = number of packets per OS clock cycle
tIRQ = OS clock cycle time
while (1)
   {
   for(i=1 to N)
      transmit(packet)
   tnow=read(hardware clock)
   sleep(tIRQ-tnow) /* Process is put to sleep.
                    CPU load 0% */
   }
```

The number of packets per cycle can be calculated easily if we take into account that the video data rate should be met by both transmission modes. For the accurate mode such data rate simply equals the specified rate $1/t_0$ while for the bursty mode the data rate equals N/t_{IRQ}. Since both have to be equal we have $N = t_{IRQ}/t_0$. Figure 2 shows the packet transmission dynamics in both accurate and bursty mode.

It seems appealing at this point to simply increase OS clock resolution in order to minimize CPU load while meeting transmission clock requirements. However, a number of empirical studies clearly indicate that increasing OS clock granularity results in a severe performance penalty due to CPU thrashing [3]. Since the kernel wakes up once per OS clock cycle processes are moved in and out from CPU, resulting in a considerable context switching overhead. Since having OS clock resolution in the network clock timescale is unfeasible we note that bursty transmission becomes necessary. In the next section, we perform extensive evaluation of bursty mode impact on CPU performance.

4.1. Analysis of CPU impact with bursty transmission mode

Recall that a video server in bursty transmission mode enters CPU once per OS clock cycle in order to transmit a number of packets (N) according to the video rate. The performance analysis presented in this section yields the following results: CPU load is minimally affected by the context switch produced by the video server once per OS cycle *and not at all by the number of packets transmitted by cycle*. Therefore, the bursty mode can be used for a wide range of video rates with minimal impact on CPU performance.

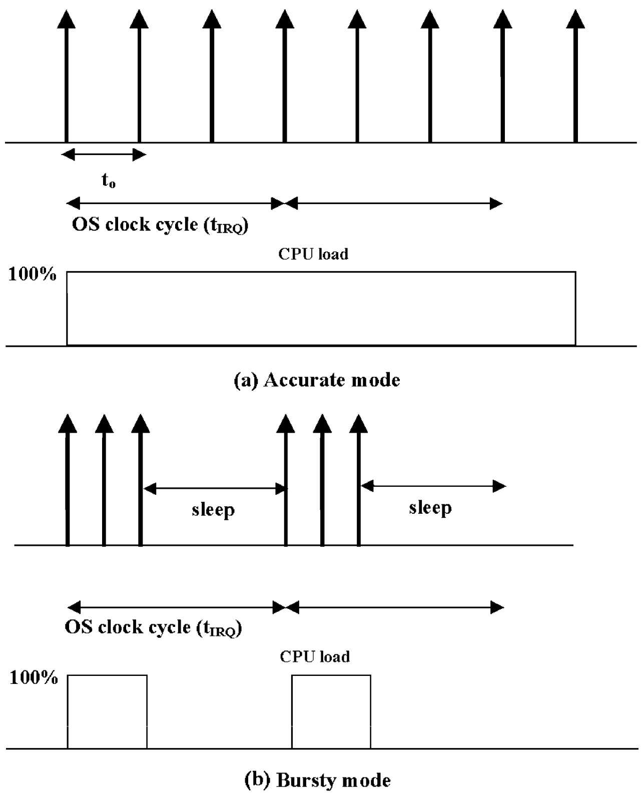


Figure 2. Accurate (a) and bursty (b) transmission modes.

We set up a simple video server running on a Linux 450 Mhz Pentium III machine attached to a 100 Mbps Ethernet, in which the number of packets transmitted per burst (N) is a parameter. We analyze CPU impact versus number of packets per burst. The results are shown in figure 3 which presents the number of context switches per OS cycle versus number of packets (MTU size) transmitted per burst. We note that the number of context switches per OS cycle is approximately 0.5 if the system is idle. Thus, for small size bursts only one context switch per OS cycle is measured. However, we note an increase to two and three context switches per OS cycle as we increase the number of packets heavily.

In order to explain figure 3 we first need to provide some insight into the dynamics of packet transmission in a general-purpose kernel. We note that after the execution of the *transmit(packet)* system call transmission does not take place immediately. Actually packets are *queued for transmission* in RAM space. Then, the network interface card (NIC) perform direct memory access to download packets to NIC hardware memory and packets are finally delivered to the network. Such direct memory access download has no impact on CPU performance. Since *transmit(packet)* is called N times per cycle

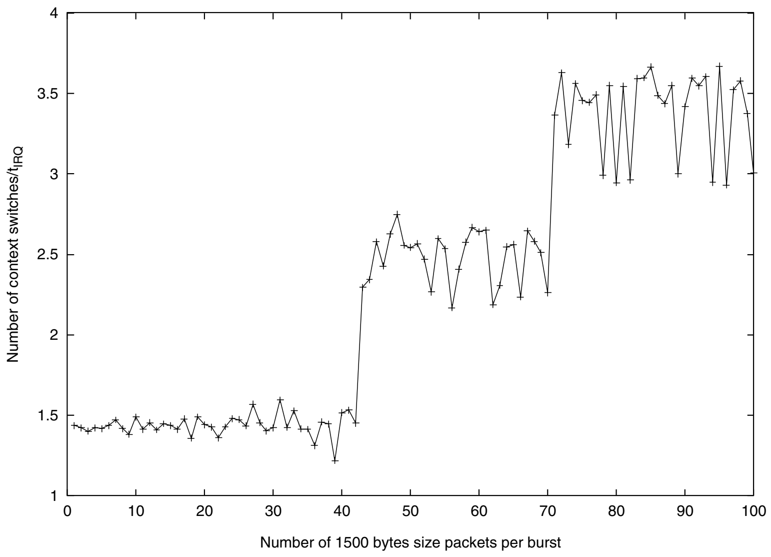


Figure 3. Number of context switches per OS cycle versus packets per burst.

according to the bursty mode algorithm packets are queued in memory so rapidly that the NIC is unable to cope with such rate. Consequently, overflow may occur, since the allocated memory space for packet transmission is limited. Before buffer overflow actually happens, thus leading to packet loss *prior to transmission*, the operating system interrupts the video server process in order to let the NIC driver release packets from the transmission queue. Such interrupt leads to an increase of one context switch as noted in figure 3.

In order to verify such hypothesis we perform the same experiment with different transmission buffer sizes. Fortunately, transmission buffer size is a tunable OS parameter. [2] The results are shown in figure 4, which presents number of context switches per OS cycle versus number of bytes per burst for different transmission buffer sizes. We clearly observe that the number of context switches decreases dramatically as the transmission buffer size increases. Most interestingly, we observe an increase in one context switch precisely in multiples of the buffer size, indicating that context switches are due to buffer overflow (while the first increase takes place exactly at the socket buffer size, further increases may happen at smaller steps because the OS unblocks the sending process when the buffer fill level has significantly dropped).

Finally, CPU impact versus number of packets per burst is presented in figure 5. Our video server is delivering a lengthy video clip in order to achieve enough significance for the CPU time and total execution time measurements, which are performed at different rates (packets per burst). We note that the CPU impact is clearly negligible since CPU utilization factor is less than 1% in all cases.

The results presented in this section show that bursty mode allows for high video rates with minimal impact on CPU. Indeed, video rates in the order of tens of Mbps can be provided

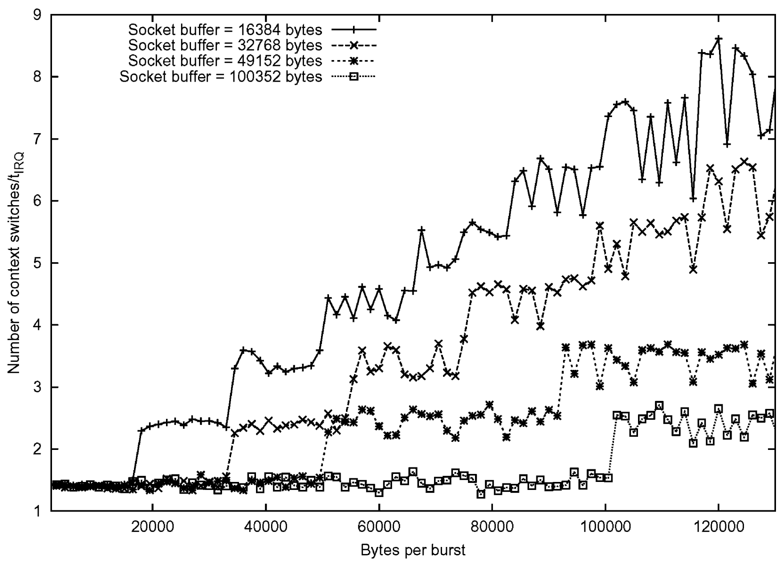


Figure 4. Number of context switches per OS cycle versus bytes per burst.

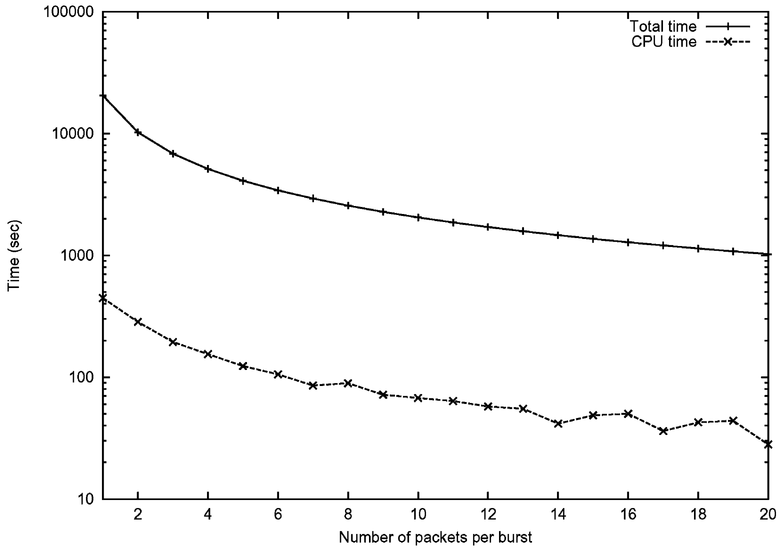


Figure 5. CPU load versus packets per burst.

through adequate tuning of OS parameters (transmission buffer size). In the client side, the jitter introduced by bursty transmission can be removed by means of playback buffering. The analysis presented in this section provides a clear picture of video transmission dynamics in a general-purpose PC. In the following section we turn our attention to the network dynamics, focusing on the specific case of an Ethernet.

5. Impact of bursty and accurate mode on network performance

In order to evaluate the strengths and drawbacks of both bursty and accurate mode from the network standpoint we set up an experimental scenario consisting of: (i) video server and (ii) video client, both implemented on a general purpose Linux PC, (iii) applications servers and (iv) application clients. On the other hand, a network monitoring tool (sniffer) is in charge of taking traffic traces. Experiments are performed in a 10 Mbps Ethernet since sniffer resolution does not allow for higher speeds. The application servers and clients serve to the purpose of assessing the impact of video transmission over a number of generic Internet applications. We distinguish between two types of reference applications:

- Transactional applications: like WWW, FTP, email or distributed file system transactions, that use TCP as a transport protocol. Such applications are normally characterized by a download from the server to the client, and usually produce relatively big size packets. The quality of service is measured by the transaction latency. In our experiments, the reference TCP application is sending 100 times the same 10 Kbytes size file from server to client. A number of 75 experiments are performed in order to achieve an acceptable confidence interval. We note that the 10 Kbytes are segmented into smaller size packets by lower protocol levels in order not to exceed the Ethernet MTU which is 1514 bytes (plus 4 preamble bytes). In any case, packet sizes are not under the control of the application but the TCP protocol agent.
- Interactive applications: like Voice over IP, that uses UDP as a transport protocol. Such applications are delay-sensitive and usually produce small packets. The quality of service is measured by the packet delay. In our case study, the reference UDP application is based on the *echo* service [9]. A 100 bytes packet is sent from the client to the server which in turn sends another packet immediately after upon server reply. By doing so, we emulate an interactive service. The application performs 100 packet transmissions and the whole experiment is repeated 75 times to achieve an acceptable confidence interval. Average measurements are taken in order to compensate for possible inaccuracies in the sniffer clock resolution.

Figure 6 presents the experimental setup with both video server and client and application servers and clients, together with the network monitoring tool.

The experiment dynamics are as follows: the video server is run in accurate and bursty mode, while the reference TCP and UDP applications run concurrently in the application servers. We take into account the following network parameters per experiment:

- Video packet sizes: small packet sizes make packet transmissions more frequent while on the other hand the buffer requirements at the client will be lower and also the protocol efficiency.
- Video rate: clearly, a growing share of Ethernet bandwidth is consumed as the video rate increases.

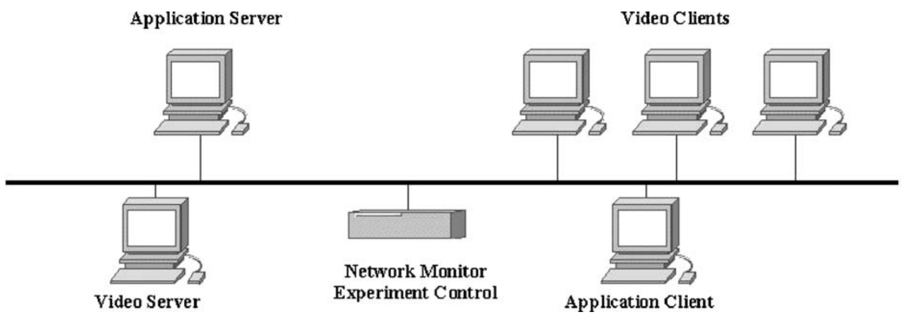


Figure 6. Experimental set-up.

As expected, CPU load reaches 100% in accurate mode while bursty mode allows for video transmission with CPU load under 5%. Furthermore, CPU load is unaffected by number of packets per OS cycle, i.e. video rate.

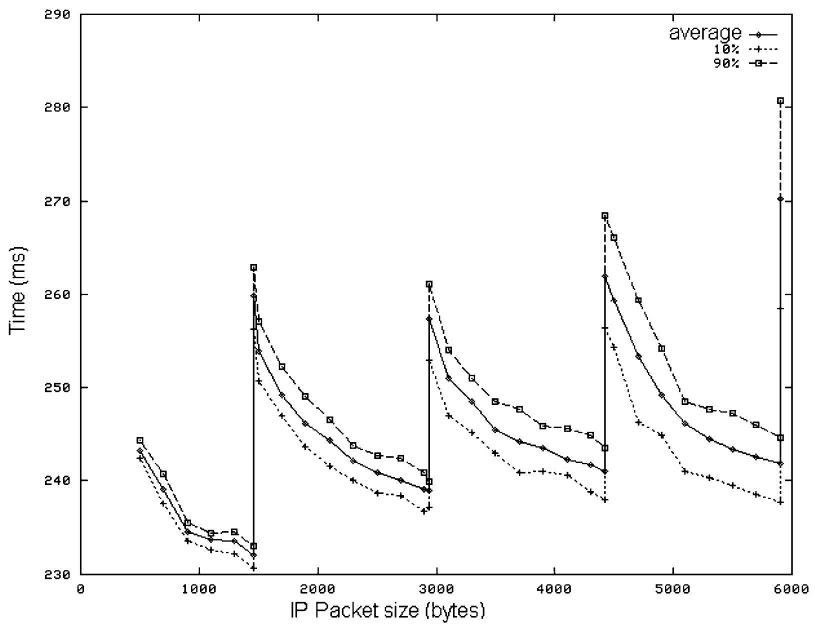
5.1. Impact of video service on UDP interactive applications

Figure 7 shows the average and 10–90% percentile UDP packet delay versus video packet size with video rates of 1.5 Mbps and a video server running in accurate and bursty mode. First, we observe an striking dependence with video packet size, being the delay minimums placed in packet sizes which are exact multiples of the Ethernet MTU, while peaks correspond to packet sizes slightly larger than the MTU size. This is simply because there is no fragmentation when packet sizes correspond to the MTU size. If the packet size is just above 1514 bytes segmentation into two packets becomes necessary, the second one with a very short payload.

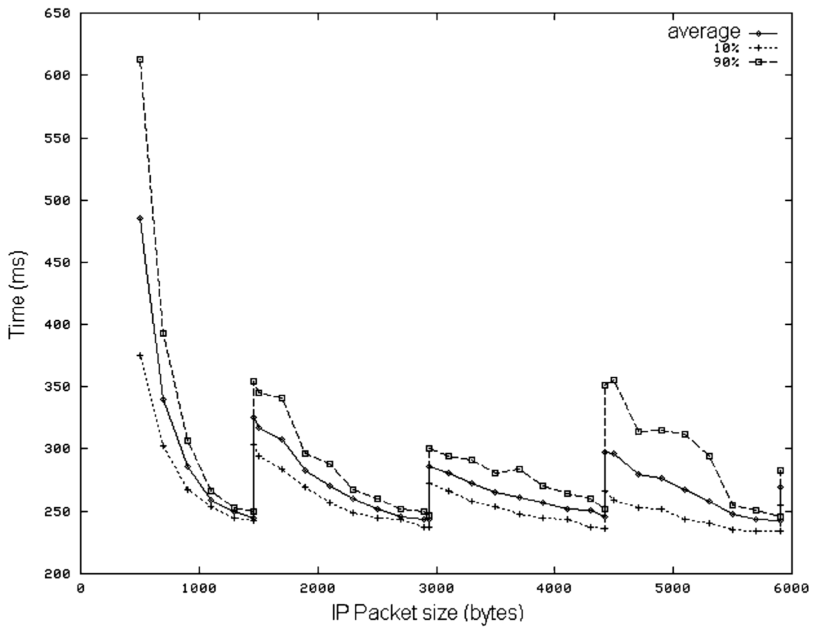
Secondly, we observe that in the accurate mode both delay and jitter are smaller. However, figure 8 shows the impact in the CPU load for both accurate and bursty mode following the event chart in Table 1. The results show that while delay and jitter are both smaller in the accurate mode CPU load increases significantly. This trade-off is due to the mismatch

Table 1. Video server event chart.

Time	Event
15 s	Server on
20 s	Video rate 1.5 Mbps
38 s	Video rate 3 Mbps
62 s	Video rate 4.5 Mbps
84 s	Video rate 3 Mbps
102 s	Video rate 1.5 Mbps
152 s	End of video transmission
155 s	Server off

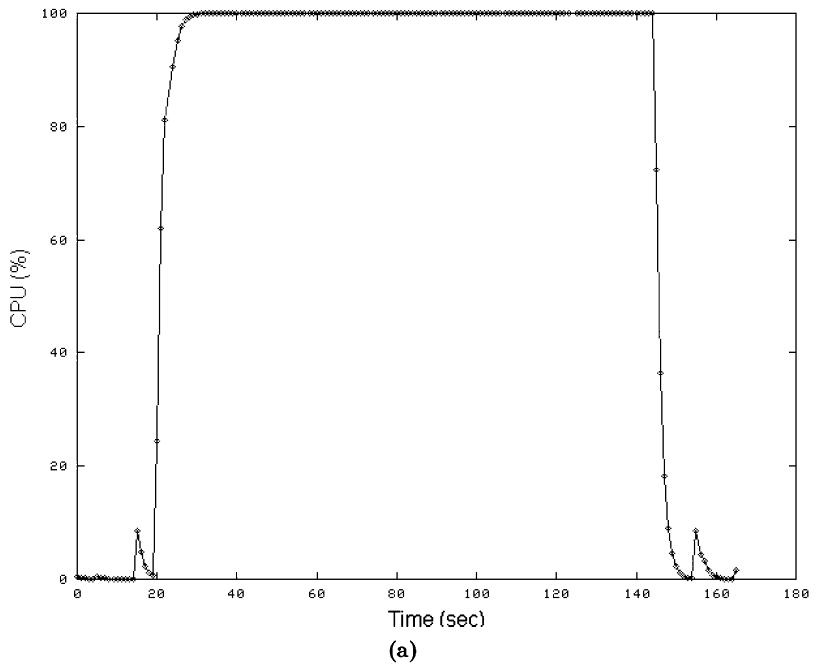


(a)



(b)

Figure 7. UDP application delay, three video clients and (a) accurate mode, (b) bursty mode video server.



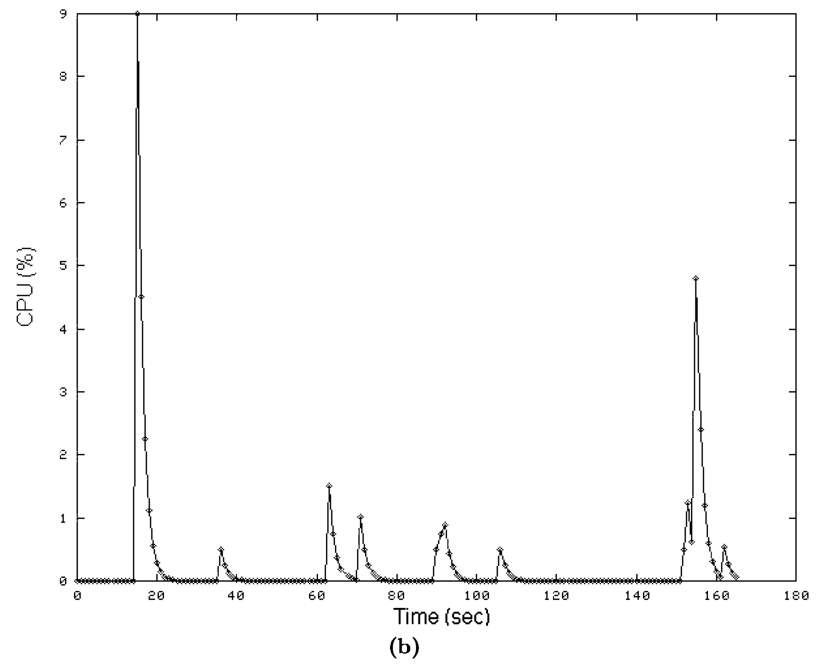


Figure 8. CPU load with 1514 bytes packet size (a) accurate mode, and (b) bursty mode.

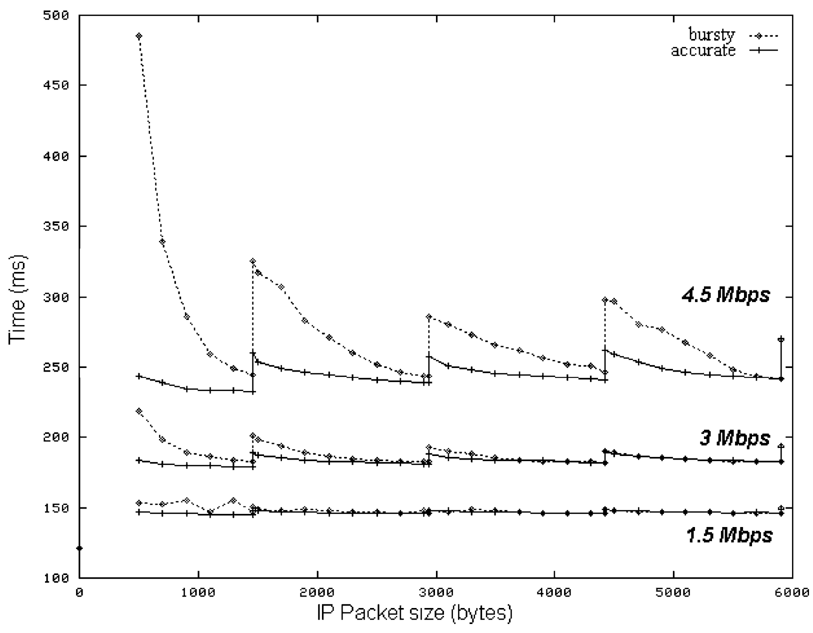


Figure 9. Average UDP application delay with increasing video rate.

between the OS clock and network clock, as noted in the previous section. On the other hand, while CPU load remains very low in bursty mode we note a number of peaks than can be explained with the event chart in Table 1. We run the video server at 1.5 Mbps and increase video rate to 3 Mbps and 4.5 Mbps at the time instants that precisely correspond to CPU load peaks. In steady-state operation we note that CPU load is independent of video rate, as noted in the previous section.

Figure 9 shows the average delay curves for both bursty and accurate mode but with variable video rate (1.5, 3 and 4.5 Mbps). As expected, the more video rate the more delay, since network load and collisions increase. On the other hand we also note the video packet size effect, which is more significant as the rate increases.

5.2. Impact of video service on TCP transactional applications

The TCP application results are quite similar, as shown in figure 10, which presents TCP transaction latency versus packet size for 1.5, 3 and 4.5 Mbps video rates. However, we note that the TCP case is not as homogeneous as the UDP case. While the curve shape is nearly the same we observe several irregularities in the graphs. Again, the video server in accurate mode gives better results than the bursty mode, especially as the video rate increases.

The curves are more irregular than in the UDP case due to the dynamics of the TCP protocol. The occurrence of repeated collisions makes the congestion control mechanism react, so that each particular TCP connection has a different and highly unpredictable

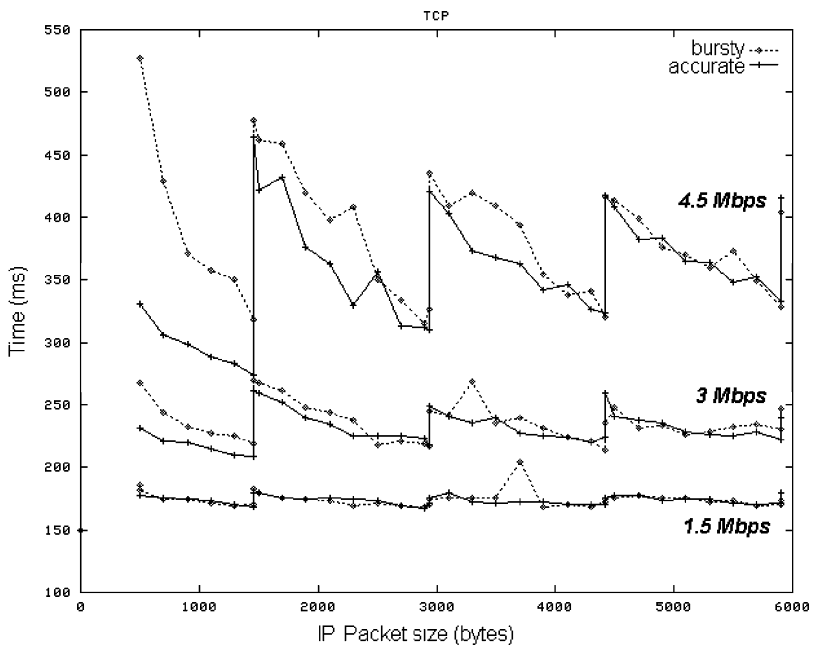
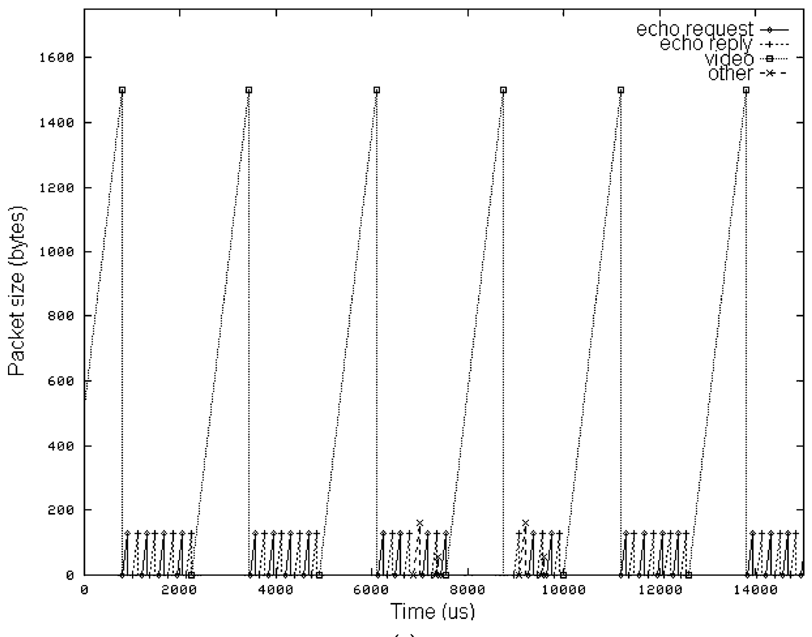


Figure 10. TCP application delay (average, 10–90% percentile) for different video rates.

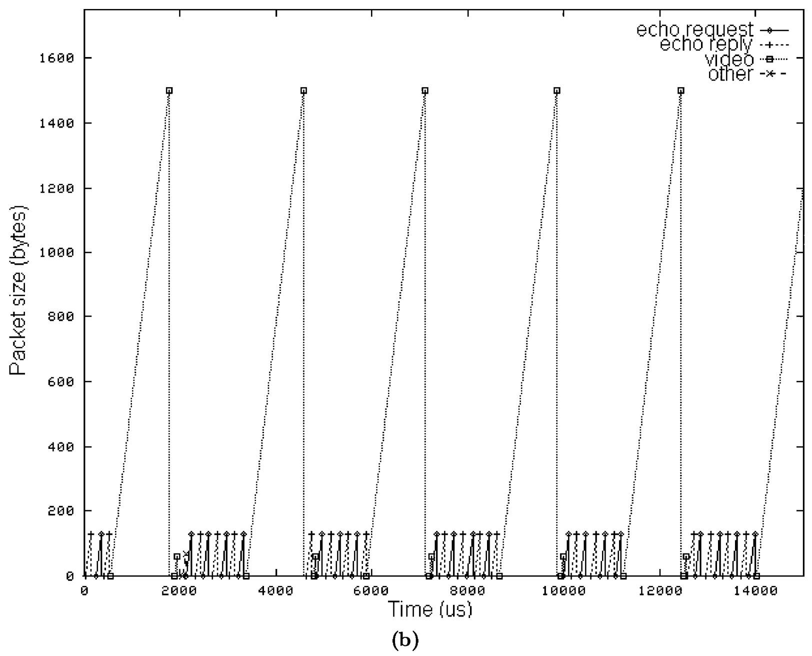
transmission pattern. Even though the shape of the curves is the same, with minimums located at multiples of the MTU size, we note a strong impact of the bursty mode in comparison to the UDP case. Considering IP packet sizes of 1,500 bytes, which provide minimum delay, there is a 70 ms difference between bursty and accurate mode.

5.3. Discussion

In order to analyze the above results we plot the time series of UDP packet arrivals in the Ethernet in figure 11 . A traffic sniffer performs high resolution captures. Packet arrivals in such captures are presented as triangles, where the first vertex position in the time axis is the time instant in which the first bit of the packet is put on the wire and the second vertex represents the instant in which the last bit of the packet leaves the wire. The triangle height represents the packet size, which is necessarily below 1514 bytes (Ethernet MTU). Figure 11 shows the packet arrival time series for the interactive applications experiments (UDP), being the server in accurate mode. From figure 11, we observe large packets corresponding to the accurate video server and small packets corresponding to the UDP application. Figure 11(a) shows video packet sizes which are close to the maximum size, 1514 bytes and figure 11(b) shows the segmentation effect when the IP video packet size is slightly larger than the Ethernet MTU, thus producing a second small packet right after the first large packet.



(a)



(b)

Figure 11. UDP application and video packets for an accurate video server with packet size of (a) 1514 bytes and (b) 1520 bytes.

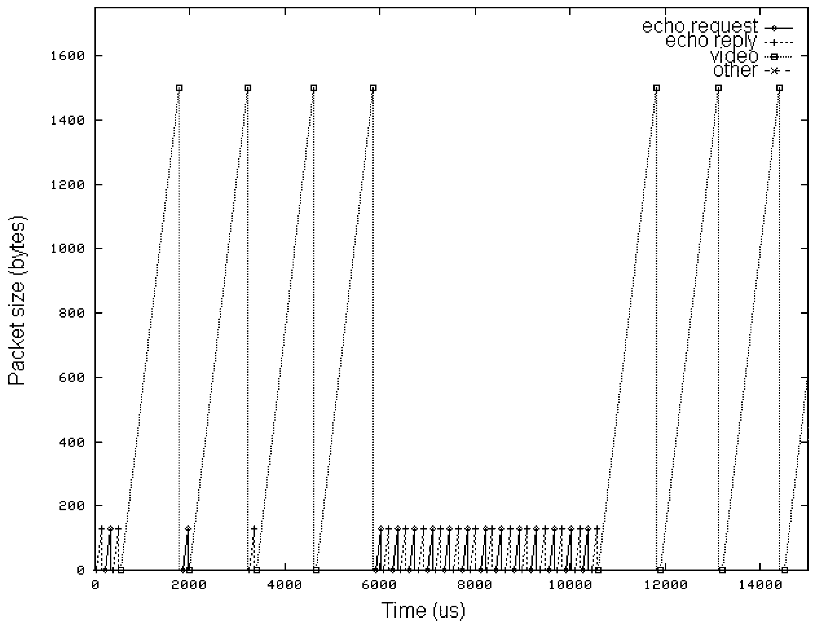
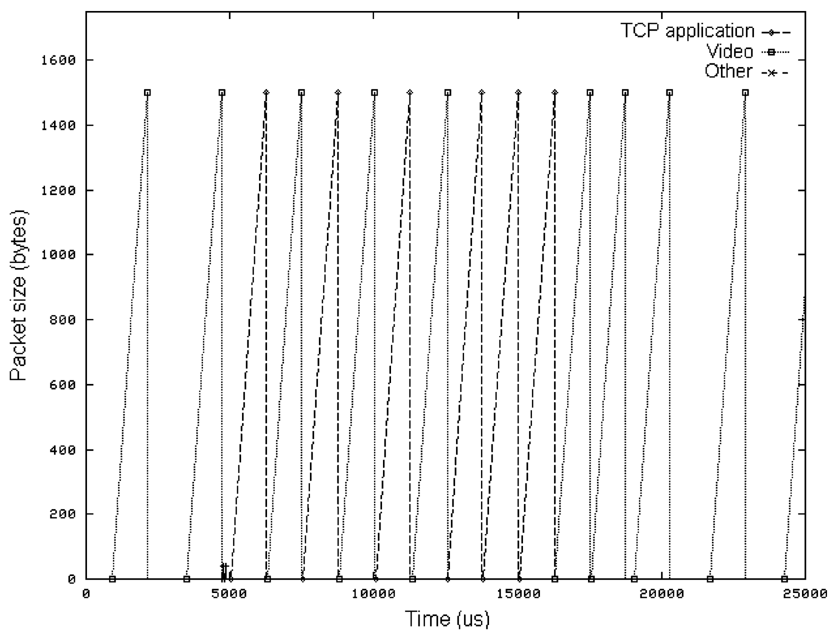


Figure 12. UDP application and video packets for a bursty video server with packet size of 1514 bytes.

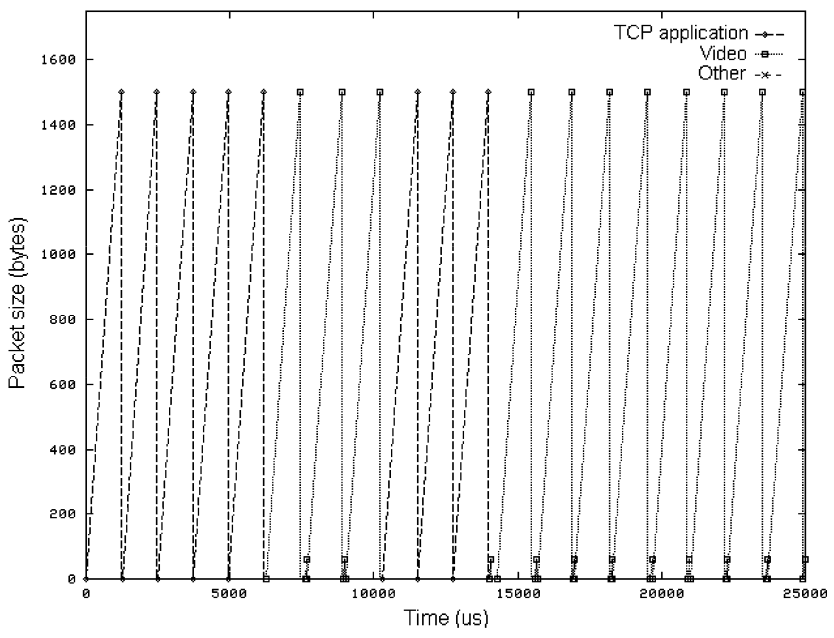
Both figures show that the UDP application tends to occupy the gap between video packets. But in 11(b), there is a high chance of collision due to the second small video packet resulting from segmentation in the video server. As a result, we observe the rather striking performance dependence with packet size for the UDP application, shown in figures 7 and 9 in the vicinity of the MTU size.

In figure 12, we show the packet transmission dynamics for the bursty video server and UDP interactive application. We note that the burstiness generated by the video server affects performance, since the server takes over the network capacity, leading to a capture effect.

Finally, figure 13 shows traces obtained with an accurate server and transactional application (TCP), showing data packets appearing between video server packets and small TCP ACKs. We observe three TCP successive packets that make the video server accumulate packets that will be sent later. Thus, as soon as the video server is able to transmit, it will be sending a large burst of outstanding packets. We note that the TCP socket layer does not allow the application to determine the time instant a packet is put on the wire, nor the size, since a pure stream service is provided. The maximum packet size will be sought by the TCP in order to maximize efficiency, as shown in figures 13(a) and 13(b) which presents packet sizes in the vicinity of the Ethernet MTU. In figure 13(b) the appearance of a second small video packet fragment whose size is bigger than the MTU increases the chances of collision. Consequently, the TCP is significantly affected by the video transmission for a twofold reason: first the TCP transmission is more bursty due to the window protocol



(a)



(b)

Figure 13. TCP application and video packets for an accurate video server with packet size of (a) 1514 bytes and (b) 1520 bytes.

dynamics. Secondly, the packet sizes are normally bigger and are not under the application control.

5.4. Enhancing general-purpose systems with packet video broadcasting capabilities

The previous sections show that an accurate transmission mode is crucial for network performance, since the impact of video transmission on the rest of TCP/IP application is minimized. However, the impact on CPU load is significant since accurate transmission can only be performed through active waiting. Thus, there is a fundamental trade-off between CPU impact and network impact. As a first approximation, the OS clock granularity may be increased, but at the cost of inefficiency due to the larger context switching rate-CPU thrashing [4].

Concerning the local area network, resource reservation capabilities are desirable, in order for the video traffic not to interfere with the rest of applications. Such resource reservation features may be provided by the MAC protocol. For instance, the Fair Dual Distributed Queue (FDDQ) [12] is a distributed protocol that provides resource reservation over the Ethernet. In the event of a collision a number of reservation minislots are provided in order for real-time stations to issue bandwidth requests. Nevertheless, the suitability of resource reservation protocols in the LAN environment can be questioned since the most part of the traffic is inherently bursty and requires best effort service with low access times. Precisely, we have shown that a random access MAC (CSMA/CD) is compatible with video broadcasting if accurate transmission is performed (see Section 2).

In order to meet both the network and CPU constraints we suggest the use of a separate processor in the NIC, so that the main CPU is offloaded from the task of transmission timing. The NIC CPU acts as a transmission clock for the video traffic, thus releasing the mother CPU from this task. A number of commercial ATM network interface cards incorporate a processor in order to perform traffic shaping and, specially, AAL segmentation and reassembly. Slight modifications at the BSD socket layer and packet driver would also be required, in order to implement "accurate transmission sockets". The socket library implementation would then mark the packets arriving from such sockets as demanding accurate transmission, and, consequently, the packets would be delivered with deterministic packet interarrival times by the NIC. Furthermore, packets could be released from the kernel to the NIC in bursty mode, since transmission timing is not under the OS responsibility anymore. At each OS cycle, the socket buffer could be transferred to the NIC buffer, in order for the NIC processor to take care of transmission timing. The fact that the number of packets transferred per OS cycle is not a limiting factor (see Section 4.1, figure 5) ensures low CPU load and good video transmission capabilities at high rates.

Figure 14 shows a block diagram of the proposed solution. A modification at the socket layer is necessary in order to distinguish the packets that require accurate transmission mode from those that require best-effort service. As a possible implementation alternative, a new socket type could be provided. Current UDP applications use the *socket*[3] system call in order to obtain an I/O descriptor for kernel communication services. For UDP applications (best effort), the socket type is SOCK_DGRAM. For video applications, a new socket type (SOCK_TIMING) would be required in order to store the video packets in separate buffers

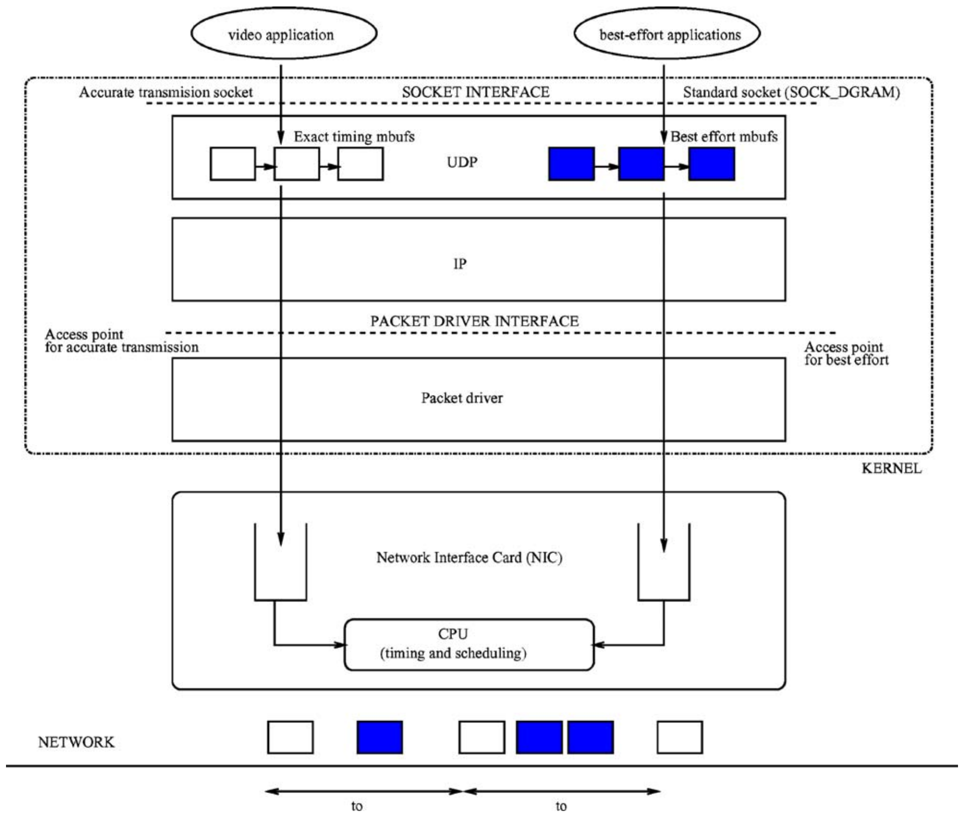

video application
best-effort applications
Accurate transmision socket
SOCKET INTERFACE
Standard socket (SOCK_DGRAM)
Exact timing mbufs
UDP
Best effort mbufs
IP
PACKET DRIVER INTERFACE
Access point for accurate transmission
Access point for best effort
Packet driver
KERNEL
Network Interface Card (NIC)
CPU (timing and scheduling)
NETWORK
to
to

Figure 14. Block diagram of the proposed solution using dedicated CPU in NIC.

(*mbufs*) at the kernel UDP/IP agent. Then, video packets could be relayed to the NIC through the packet driver access point for accurate transmission service. Note that nearly no changes are required at the UDP/IP layers, since video and best-effort packets receive the same treatment, except for the fact that separate buffers are allocated for each type of traffic.

On the other hand, the packet driver should provide two service classes: “accurate transmission service” for video packets and “best effort service” for the rest of UDP and TCP traffic. Other than separate buffering for both services no additional changes would be required. The NIC service interrupt routine would be in charge of downloading the buffer contents to the NIC (once per OS cycle), as happens in current PCs. Both video packets and best effort packets would be sent in bursts to the NIC, at nearly no CPU cost. Then, specific hardware, either a CPU or a Programmable Logic Device (PLD), would be devoted to accurate transmission of video packets. On the other hand, such dedicated hardware would also be required to interleave best-effort packets in the constant rate video stream. The analysis presented in this paper shows that an accurate transmission mode allows to optimize network usage since the rest of applications delivering traffic to the same LAN suffer a lesser impact. However, such accurate transmission cannot be achieved if the OS is not released from the burden of transmission timing, due to the mismatch between OS clock and network clock granularity, that leads to active waiting. We believe that the latter problem can be circumvented with the use of the protocol stack and NIC architecture shown in figure 14. To the best of our knowledge, neither commercial NICs nor research prototypes have been designed with such an architecture, and a significant benefit could be obtained.

6. Conclusions

As a conclusion, our findings show that an accurate transmission mode produces less burstiness in the Ethernet, that translates in a reduced impact for the rest of applications. However, the CPU load increases significantly since the only way to cope with the network timing is through active waiting. A bursty transmission mode permits low CPU utilization while the impact in the network is not that significant. On the other hand, we note a strong performance dependence with packet size. In order to maintain the video rate it becomes necessary to control not only the timing but the packet size so that the rest of applications are not severely affected. Interestingly, we note that packet sizes equal to Ethernet MTU multiples not only minimize the impact over the rest of applications but also maximize bandwidth efficiency. Furthermore, TCP applications will suffer a more severe impact, since the traffic burstiness and packet size are not under the application control but under TCP control, which will produce bursts due to the window flow and congestion control mechanisms and large packet sizes in order to maximize efficiency.

In order to transform future PCs into real multimedia terminals the design of the general-purpose operating system should incorporate video and audio transmission capabilities. The primary concern is to provide agile process swapping so that the timing requirements imposed by the network are not realized through active waiting. While we are witnessing an extraordinary increase in CPU processing power the network bandwidth is growing at an even faster pace. Our findings show that the adaptation between OS and network is of fundamental importance to facilitate the deployment of multimedia applications. A departure from the traditional protocol engineering paradigm, that considers the network as an isolated subsystem from the host, has an starting point in the present work and is the subject of our future work.

Notes

1. Additionally, such transitions can be triggered by hardware or software interrupts.
2. The BSD socket library provides the *setsockopt* function to do so.
3. int socket(int family, int type, int protocol).

References

1. J. Bolot and T. Turletti, "Experience with control mechanisms for packet video in the internet," ACM Computer Communication Review, 1997.

2. I. Dalgic, W. Cien, and F. Tobagi, “Evaluation of 10Base-T and 100Base-T Ethernets carrying video, audio and data traffic,” in Proceedings of IEEE INFOCOM’94, Toronto, Canada, 1994.
3. B. Goodheart and J. Cox, The Magic Garden Explained: The Internals of UNIX System V Release 4, An Open System Design, Prentice Hall, 1994.
4. S. Gringeri, K. Shuaib, and R. Egorov, “Traffic shaping, bandwidth allocation, and quality assessment for MPEG video distribution over broadband networks,” IEEE Network, 1998.
5. S. Guota and C. Williamson, “An experimental study of video traffic on an Ethernet local area network,” in Proceedings of GLOBECOM’94, San Francisco, CA, 1994.
6. D. Meliksetin, F. Feng-Kuo Yu, and C.R. Chen, “Methodologies for designing video servers,” IEEE Transactions on Multimedia, Vol. 2, No. (1), 2000.
7. M. Molle and K. Christensen, “The effects of controlling capture on multimedia traffic for shared Ethernet systems,” Journal of Telecommunications Systems, Vol. 9, Nos. (3/4), 1998.
8. J. Ni, T. Yang, and D. Tsang, “Source modelling, queueing analysis, and bandwidth allocation for VBR MPEG-2 video traffic in ATM networks,” IEE Proceedings in Communications, Vol. 143, No. (4), 1996.
9. J. Postel, “Echo protocol,” Internet Standard STD0020-RFC862, 1983.
10. K.K. Ramakrishnan and H. Yang, “The Ethernet capture effect: Analysis and solution,” in Proceedings of 19th Conference on Local Computer Networks, Minneapolis Minn, 1994, pp. 228–240.
11. L. Vidaller and J. Aracil, “The ETSIT teleeducation system,” in Proceedings of DELTA Telematics Conference 94, Dusseldorf, Germany, 1994.
12. B. Whetten, S. Steinberg, and D. Ferrari, “The packet starvation effect in CSMA/CD LANs and solution,” in Proceedings of IEEE Local Computer Networks, Minneapolis, MN, 1994.

Eduardo Magaña received his M.Sc. and Ph.D. degrees in Telecommunications Engineering from Public University of Navarra, Pamplona, Spain, in 1998 and 2001 respectively. He is an assistant lecturer at Public University of Navarra. During 2002 he is a postdoctoral visiting research fellow at the Department of Electrical Engineering and Computer Science, University of California, Berkeley. His main research interests are network monitoring, multimedia services and wireless networks.

Javier Aracil received the M.Sc. and Ph.D. degrees (Honors) from Technical University of Madrid in 1993 and 1995, both in Telecommunications Engineering. In 1995 he was awarded with a Fulbright scholarship and was appointed as a Postdoctoral Researcher of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. In 1998 he was a research scholar at the Center for Advanced Telecommunications, Systems and Services of The University of Texas at Dallas and he is currently a tenured Associate Professor at Public University of Navarra, Spain. His research interest are in Internet services analysis and performance evaluation of communication networks. Dr. Aracil is a member of the editorial board of SPIE/Kluwer Optical Networks Magazine.

Jesús Villadangos is an associate professor at the Public University of Navarra (UPNA) in Spain. He receives his B.S. degree in physics from the Universidad del Pas Vasco and his Ph.D. in communications engineering from Public University of Navarra in 1999. He has been a research assistant at the Universitt Hannover and Mnster in Germany. Since 2000 he has been jointly in Telematics Engineering at the UPNA. His research interests include distributed algorithms, adaptive network control and performance evaluation of communication networks and distributed algorithms.