♦ Latency in Cloud-Based Interactive Streaming Content

Ron Sharp

Personal cloud computing will offer individuals and small businesses unlimited computing and network resources with no restrictions of hardware, operating systems, and access networks and without the hassle and cost of regular maintenance and security. However, moving these applications into the cloud can add a good bit of latency between an action (e.g., dragging a mouse) and the expected result (highlighted text). Our initial investigation found that user-acceptable latency depends on the application, but the most stringent games demand a roundtrip latency of less than 100 ms. Examining the entire action-response path, we identified the major contributors to latency and what can be done to minimize it for personal computer (PC) applications executed in the cloud. In this paper, we provide target latency values for each stage and suggestions on how to achieve this target. Special focus is given to video encoding since current encoders do not sufficiently support multiple streams and the low latency required for a cost-effective personal cloud computing service.

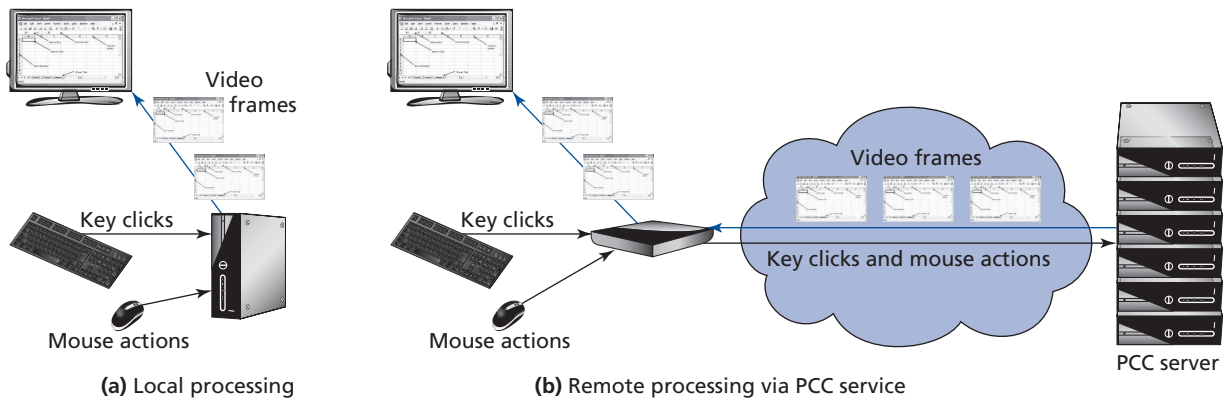
Introduction

Interactive content is video (or other media) which is dynamically influenced by user input. For this paper, we will focus on interactive video for a personal cloud computing (PCC) service. **Figure 1** depicts this service. Figure 1a shows a typical personal computer (PC) setup with all application processing performed locally. Figure 1b shows the PCC service with all input being sent over the network to a remote server and the resulting video being sent back to the user's screen. It is important for the discussion to understand that all computer-generated content viewed on a computer monitor, television (TV), or mobile device is in effect a video running at 25 to 60 frames per second (fps) or more as depicted in Figure 1a. Each frame of the video (one screen image) is created by a computer application, often with help from a graphics coprocessor. The primary difference between this video and a television show or a streamed Netflix* movie is that each frame of the video from the computer is created a fraction of a second prior to viewing. The other important difference is that the content displayed is dynamically influenced by the consumer in real time. For some applications, such as reading email, the frame content does not change often but the frames are still redisplayed (refreshed) many times a second.

Bell Labs Technical Journal 17(2), 67–80 (2012) © 2012 Alcatel-Lucent. Published by Wiley Periodicals, Inc.
Published online in Wiley Online Library (wileyonlinelibrary.com) • DOI: 10.1002/bltj.21545

Panel 1. Abbreviations, Acronyms, and Terms

3D—Three dimensional
3G—Third generation
4G—Fourth generation
ASP—Application service provider
CPU—Control processor unit
CRT—Cathode ray tube
DSL—Digital subscriber line
FPGA—Field programmable gate array
FPS—Frames per second
GDR—Gradual decoder refresh
GPU—Graphics processor unit
GSM—Global System for Mobile Communications
GTA—Grand Theft Auto
HDTV—High-definition television
HSPA—High Speed Packet Access
HTML—Hypertext Markup Language
IP—Internet Protocol
ISP—Internet service provider
ITU—International Telecommunication Union
LCD—Liquid crystal display
LTE—Long Term Evolution
NAL—Network abstraction layer
PC—Personal computer
PCC—Personal cloud computing
PCIe—Peripheral component interconnect express
PON—Passive optical network
PS2—PlayStation 2
RTP—Real Time Transport Protocol
STB—Set-top box
TCP—Transmission Control Protocol
TV—Television
USB—Universal serial buss



(a) Local processing

(b) Remote processing via PCC service

PCC—Personal cloud computing

Figure 1.
Interactive video for a personal cloud computing service.

With interactive content there is always some gap between a user action and the resulting change as perceived by the user. For some functions, such as moving the cursor with the mouse, we do not notice this delay. However, time is required to detect mouse movement, calculate the new position on the screen for the cursor, determine appropriate changes to the next frame, modify the frame, wait for next frame time, and send the frame to the monitor. When all of this occurs on your local computer it takes less then a twentieth of a second (50 ms) and you cannot perceive the delay. However, when streaming interactive video from the cloud, the processor is moved far away from the user's keyboard, mouse, and monitor. This introduces many additional requirements and steps which add to the overall latency.

Let's define what we mean by latency in this environment. Let's assume an application (e.g., game or

photo editor) is running on a server many miles away (see Figure 1b). All keyboard and mouse input are sent over the network to an application running on a remote server. The video frames generated by the application on the remote server are diverted from going to the host's video monitor and sent back over the network to a simple client box which forwards the frames to a connected display (monitor, TV, or handheld). Latency represents the time from a local user event (e.g., mouse move) to the user seeing the response on the screen (cursor move). **Figure 2** shows the steps which impact latency for interactive content created in the cloud. A short description of each of these steps is included in **Table I**.

Steps 2 through 4, and steps 6 through 11 in Table I were added as a result of moving this application to the cloud. The focus of our research is to minimize the added latency from these steps. In this paper we will provide a detailed description of the complete round trip, and explain how each step contributes to the total latency and what has been done or can be done to reduce its impact. We will go into additional detail in the area of compression, which can add greatly to this latency and which has been the focus of much of our work.

Previous Work

Hypertext Markup Language (HTML) web pages serve as the primary interface mechanism to the web. Web servers do not create each video frame but rather send instructions to the client on how to build and display each page. Much work has been done on the latency between a mouse click and the resulting web page being displayed due to the financial impact to web vendors. Google determined that a search results page that was delayed by more then 500 ms (half a second) resulted in 20 percent less traffic. Amazon found that every 100 ms of latency cost the company one percent in sales [7]. Other research has studied latency in multi-player games. In these games, the video frames are created locally, but user input is sent to a remote server and instructions on their impact are returned. Thus, high latency could result in shooting at a target which is no longer there. Both [3] and [5] discuss the impact of network latency on multiplayer games. The consensus was that a user's tolerance to latency can vary greatly by the type of game and also the extent to which the developers worked to hide any latency (e.g., shotguns tolerate latency much better then sniper rifles). Other work has been done on latency in video conferencing. It is a common

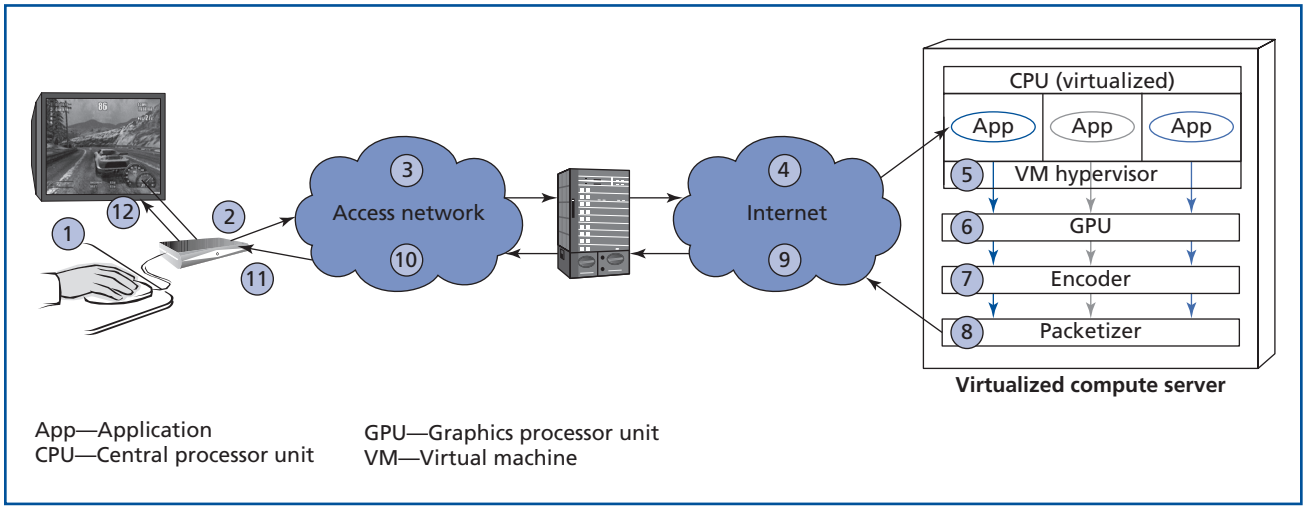


Figure 2.
Round trip latency path.

Table I. Latency contributors.

#	Steps	Added by cloud	Latency low	Latency typical	Latency high
1	**User action and detection**: Depresses mouse key (or tap a touchscreen) and event sent to local device (STB, PC, TV, Tablet). Low assumes optimized user device.		2	4	8
2	**Event packetization**: Package event notification in a packet and send. Computer cycles, negligible impact.	X	0	0	1
3	**Event packet to ISP**: Travels over access network to Internet service provider	X	2	4	8
4	**Event packet to ASP**: Travels over Internet to application service provider.	X	2	6	20
5	**Application processing**: Application receives packet and makes changes (working with GPU) to next video frame. Includes rendering time. High assumes 30 fps.		40	40	80
6	**Frame transfer to encoder**: Frame is transferred from GPU to encoder. Low assumes minimal CPU intervention and frame pushed directly to encoder.	X	3	18	32
7	**Encoding**: Frame is encoded (compressed). Low is time required for first segment to be encoded.	X	4	16	30
8	**Frame packetization**: Encoded frame is broken into packets and sent out. Higher numbers assume queuing.	X	1	12	16
9	**Frame packets to ISP**: Travel over ASP network and Internet to service provider.	X	2	8	35
10	**Frame packets to user**: Travel over access network to user. Low assumes a fast (>30 MB/s) access network.	X	8	20	32
11	**Decode frame**: Local processor reconstructs and decodes (decompresses) frame	X	8	16	32
12	**Display frame**: Frame is sent to user display. Low assumes 0 processing by display.		8	16	62
	Total		80	160	356
	Total for ms added by cloud	X	30	100	206

Note: Latency times in milliseconds (1/1000 second).

ASP—Application service provider
CPU—Control processor unit
fps—Frames per second
GPU—Graphics processor unit
ISP—Internet service provider
PC—Personal computer
STB—Set-top box
TV—Television

occurrence today for the audio and video in a video conference to be out of sync. This is primarily due to latency introduced by the extra processing required by the video [6]. Very little work has been done in the area of latency on streaming interactive video content in which the remote server creates and transmits each video frame. The primary reason for this is that until recently few homes could support the bandwidth required for this content. An emerging leader in the area of offering a streaming, interactive video

service is OnLive* [12]. It is a startup offering online gaming via streaming content. Throughout this paper, I will cite examples of their efforts. OnLive has obviously done considerable research in this area over the years but has published very little, presumably to protect their trade secrets.

Acceptable Latency

How much latency will a user tolerate? Not surprisingly, the answer depends on the service and action. For voice applications, the International Telecommunication Union (ITU) G.114 standard [10] recommends a maximum "one-way" latency of 150 ms. Video conferencing uses the same value for maximum latency. However, it is recommended that the voice and video should not be separated by more than 75 ms [6] to avoid a detectable lip sync issue. For personal cloud computing (PCC), the tolerable amount of latency can vary greatly by the application. For many office applications, such as word processing and email, the latency (round trip) can be as high as 1/3 second (333 ms) without a user noticing the extra lag. A notable exception to this is mouse cursor movement which our tests have found distracting above 100 ms. When the application is a game, latency requirements are even more stringent. Some hardcore gamers will say it must be less than 50 ms. However, most agree the desired target is between 50 ms and 100 ms for interactive games. It is interesting to note that many console games (e.g., PlayStation*, Xbox*, Wii*) exceed this target anywhere from 51 ms to a high of 166 ms for Grand Theft Auto (GTA*) [18]. Some of the reason for this added delay in console games is due to the expanding market for high-definition televisions (HDTVs) over those with cathode ray tubes (CRTs). This is discussed further in the "Latency Contributors" section.

OnLive Gaming Service

OnLive offers PC gaming from the cloud. From a PC or Mac*, TV (with set-top box), or tablet/cellphone you can play high-end PC games. OnLive had publicly stated a target of 80 ms for latency. Considering that half of that number can be from the game itself (steps 1, 5, and 12 in Table I) this is a very challenging goal to reach. OnLive plans to place five data centers around the country to service all of the United States (U.S.), with each data center sited within a 1,000 mile radius of the others. Currently they have centers in Virginia and California. I tested the OnLive service from locations which were 25 miles and 250 miles from the OnLive data center in Virginia. Results are shown in **Table II**. Playing a first person shooter game, I measured latency of 150 ms and above. This is considerably above OnLive's target as well as most hard-core gamers' "stated" threshold. However, OnLive's business seems to be succeeding. Reviews of the service are mostly positive. The reviewers all mention the noticeable latency but they feel the average gamer can compensate and the service is worth it. However, many hard core gamers give it poor reviews due to latency and swear they would never pay for the service. The most common complaint is the degraded quality due to compression and decompression of the video stream.

I found the service very acceptable for most games. Video quality was lower than with the PC version and the latency was only somewhat noticeable. The only latency issue I encountered was with games where fast and accurate mouse movement was required, such as a trivia game which required using the mouse to select the proper answer as quickly as possible. I surveyed a few avid gamers (age 18–25) who noted that they could detect the slower response times but felt the games were playable and it is a service they would consider. Their primary concern was the loss of video quality. Video quality from encoding is highly dependent on available bandwidth so we can expect video quality to improve as access speeds increase. Digital Foundry performed a detailed review of the OnLive service in the United Kingdom (U.K.), and provided test results for latency and evaluations of video quality and game performance [11].

Latency Contributors

Table I provides the list of steps for the total round trip for cloud-based interactive video content. With each is an estimate of the expected time in milliseconds for minimum, typical, and high latency. Some of the latency values shown are measured and some are estimated using data from other studies. The *minimum* (low latency) column is highly optimized and assumes

Table II. Test results for the Online† service.

Location	Murray Hill, New Jersey	Greenbelt, Maryland
Miles to OnLive data center	~200	~10
Access service	Comcast	FIOS‡
Access type	Cable	Fiber optic
Speed – downstream	23 Mb/s	42 Mb/s
Ping time to OnLive server network	21 ms	6 ms
Traceroute hops to OnLive network	14	7
OnLive downstream video speeds	4–6 Mb/s	4–6 Mb/s
OnLive frame rate	40–60 fps	40–60 fps
Average latency using keyboard	175 ms	160 ms
Average latency using mouse click	160 ms	140 ms
Average latency using mouse movement	155 ms	140 ms

fps—Frames per second

†Registered trademark of OnLive, Inc.
‡Registered trademark of Verizon Trademark Services LLC.

all recommendations described in this paper were implemented. This is the column we should use as our target for each segment. The *typical* column is what can be expected with today's technology, Internet, and access networks and servers tuned for interactive content. The *high* column shows the latency with no tuning for interactive content and poor choice for user equipment. The sections below describe each step in more detail and provide suggestions on how to reduce the latency for the step.

User Action and Detection

This step is the same for local or cloud processing. The local device, whether PC, set-top box, or tablet, must detect an event from a peripheral such as a keyboard, mouse, or touchscreen. The latency can vary greatly depending on the device and the protocol used to connect to it. For example, the standard Microsoft Windows* universal service bus (USB) port polls at 125 Hz so it can take up to 8 ms to detect a mouse movement. On average, it will take 4 ms. PlayStation 2 (PS2*) ports scan at varying rates but most are set at 100 Hz so it is a little slower than a USB device. Both of these numbers were verified with lab tests. Wireless (Bluetooth*) peripherals can add a good bit of extra lag. Bluetooth can add up to 20 ms or more. This was confirmed in our lab using a Bluetooth keyboard and mouse. Most local processing can afford this extra latency but cloud applications cannot.

In another test, I used a wireless performance mouse from Logitech (M705). It uses a wireless protocol in the 2.4 GHz range. The result was 125 Hz, matching the USB port into which its receiver was plugged. I performed another test on a touchpad for a laptop computer and obtained a result of 80 Hz (an average of 6.25 ms delay). Clearly a USB mouse or keyboard is the preferred device, however even they still used precious milliseconds.

One method used by gamers to lower this latency is to overclock the USB or PS2 port [13]. You can, in theory, overclock the USB port to 1000 Hz reducing the added latency to 1 ms. Of course, your mouse (and keyboard) must be fast enough to take advantage of the increase in speed. OnLive decided to go with the IEEE 802.15.4 wireless standard [9] for their microconsole (TV set-top box (STB)). They report that

it adds only 800 μs (for one controller) and a maximum of 2 ms for four controllers [8].

Event Packetization

The event details (mouse button or keyboard key depressions) are enclosed in a small Internet Protocol (IP) packet. This is handled by the processor on the PC, set-top box, or mobile device and adds negligible latency.

Event Packet to ISP

The small event packet is sent up the access network to the Internet service provider (ISP). The amount of time for this will vary greatly with the access technology. In a passive optical network (PON) (fiber), the latency can be as low as 1 ms with an optimized scheduling algorithm. For cable, we measured average latencies of 4 ms in our lab using Comcast cable. For digital subscriber line (DSL), the numbers are higher with a minimum of 5 ms to a typical of 15 ms. The numbers for PON and DSL were obtained from experts within Bell Labs. In wireless networks, latency can be even higher [14]. For example, in a Global System for Mobile Communications (GSM) network, it can be 120 ms, but with newer technologies, latency is decreasing. For High Speed Packet Access (HSPA) it is 40 ms, and for Long Term Evolution (LTE) latency can be reduced to 20 ms. A lab within Alcatel-Lucent has reported latencies below 15 ms for packets going from a mobile device to an LTE base station node. For both wireline and wireless technologies the latency can be reduced with better scheduling algorithms and minimizing hops within the ISP's network. A traceroute test with a Comcast cable service revealed the packet traversed eight routers in the Comcast network before exiting to the Internet or to the application service provider (ASP) network.

Event Packet to ASP

The packet now must travel over the Internet to the ASP. The cost here is speed of light and processing time at each router it encounters. The speed of light through a fiber [4] creates a delay of approximately 4 ms for each 500 miles traveled. The latency added at a router varies considerably from under 50 μs to over a millisecond. The variance is caused by the amount of time the packet must wait in a queue behind other packets going out a particular network port. If there is no congestion, the latency added by large Internet routers is small. However, a packet may have to traverse many routers before reaching its destination.

During the test of the OnLive service from our lab, packets traversed 15 routers (30 total for round trip). Using the "ping" tool, I measured a total round trip time of only 16 ms. Pinging the nearest router showed that half of that 16 ms was for the access network (i.e., event packet to ISP). Half of the remaining 8 ms was speed of light cost. So each router contributed less than 1/8 of a millisecond (4 ms/30). Of course a few of the ping measurements showed much higher numbers (up to 78 ms), presumably due to congestion. If these numbers were to increase or become more common, the network cost would be much higher. The Internet and most access networks provide no latency guarantees. The current strategy is to add more bandwidth capability rather than quality of service controls.

ISPs often set up peering agreements with one another so traffic travels directly between their two networks. This will reduce latency. In addition, an ASP will often negotiate with its ISP to get preferred treatment for packets going to or from the network. OnLive went a step further and connects its data center to several ISPs. When a connection is established, it tests each ISP link to see which offers the better connection (for latency and bandwidth) back to the user. OnLive plans to set up only five data centers to service the entire U.S. The company calculates that a data center can afford to be up to 1,000 miles from the user. Just accounting for the speed of light within a fiber, 1,000 miles will add 16 ms (round trip) to the latency.

The optimum solution to reduce latency for this and the previous step (event packet to ISP) is to place the application servers in the ISP network, close to the access point of the user. This eliminates all of the latency for this step. In addition, it eliminates the latency for travel within the ISP network to the Internet in the previous step.

Application Processing

Applications are generally input-oriented. When input arrives, the application makes the appropriate change to the screen in the next video frame. An

```
while(player alive)
      check for user input
      calulate change to user or objects
      move characters based on AI
      resolve collisions
      draw graphics
end while
```

Figure 3.
Pseudo code of a basic "game loop."

example would be to type into a word processor. When there is no input, the screen remains the same. These applications are said to be "event driven." For games and other similar applications, the screen may need to change based on user events or it may need to change just due to time. Therefore most games will invoke a "game loop." Pseudo code from a basic game loop is shown in **Figure 3**.

Let's say the application is an interactive game. The software will receive the event packet which might be a move of the onscreen character. It will calculate the required change to the displayed scene and communicate these desired changes to the graphics processor unit (GPU). The application must then wait (0 ms to 32 ms) for the next frame time slot before releasing the frame. The GPU will then fill in all of the changes to the three-dimensional (3D) landscape.

Several factors will affect the latency at this point. Let's assume the game operates at 30 fps, which implies that each frame time is 32 ms in length. The game has 32 ms for the loop to check for new input, calculate changes, and determine the appropriate changes to the screen. On average we can expect the packet will arrive 16 ms into this loop. It will then wait 16 ms until it is recognized by the application and another 32 ms to process for a total time of 48 ms. After releasing the frame, the GPU will take another frame time to make the changes. The total for the application and GPU is 2.5 × frame time or 80 ms for a game running at 30 fps. To reduce this cost, most games run at 60 fps (16 ms frame time) and thus the delay is only 40 ms (2.5 1 × 6 ms). More detail is provided on the fps penalty in a later section.

There is not much we can do to reduce application latency. Providing a more powerful control processor unit (CPU) and GPU can make a game run faster, but the average latency will always be 2.5 × 1/frame rate. And running the game at a faster fps will dramatically increase the required bandwidth. An idea proposed by the author is to run the game at 30 fps and increase to 60 fps only for a very short period after a user event such as a mouse click.

Frame Transfer to Encoder

The created frame resides in the local memory of the GPU peripheral component interconnect express (PCIe) card. It must be transmitted over the PCIe bus to the encoder. With the high speed of PCIe (16–64 Gb/s) you would expect delay to be minimal but the frame is very large at this point. A single 720P frame is 30 Mbits. A 1080P frame is twice that. If the CPU has to be employed to transfer this frame, the latency as well as the added burden to the CPU can be quite high. Our team has developed a way to transfer the frame directly with minimal CPU intervention. Obtaining even lower latency numbers would require removing the PCIe bus, placing the GPU and encoder on the same line card, and using a higher speed connection or shared memory. Even better is to perform some or all of the encoding in the GPU. The initial stage of encoding (chroma subsampling) can be performed in the GPU and this can reduce the required bandwidth by up to 50 percent.

The other issue is caused by multiple streams sharing the same encoder card. This can cause a resource contention problem, adding more latency while a frame waits its turn. See the section on "Resource Contention Penalty" for more discussion.

Encoding

Encoding is a very compute intensive step which compresses a 720P stream from 1 Gb/s to less than 5 Mb/s (a 200x reduction). The primary method used in most video compression is referred to as "prediction" and sends only the differences from a previous frame. This requires a good bit of search in the previous frames to find duplicate (or similar) blocks of pixels. Most real time encoders process at frame rate 30 fps which means it requires a full 32 ms to complete the encoding. However, this adds directly to the latency so

a faster encoder is required. We are looking at encoders that can encode a frame within 5 to 10 ms as well as supporting many streams simultaneously. Good encoder algorithms such as ITU H.264 also use "B-frames" to improve video quality and reduce bandwidth. With B-frames (or B slices) [15], the encoder will search past and future frames for matches. This requires a reordering of frames and more buffering at the server and client, resulting in additional latency of one or two frames. For this reason, this, and other latency-insensitive encoder options are typically not invoked for real time encoding. Within the H.264 standard, real time encoders typically only implement the "baseline profile." In addition, they are not able to spend as much time searching for the best match during prediction. These limitations result either in an increase in the bandwidth required to transmit the stream, or a reduction in video quality. Highly efficient H.264 encoders can reduce a 720P stream to 3 Mb/s. To maintain the same level of quality, a low latency encoder will require roughly twice the bandwidth of a non-interactive video stream. We have spent some time looking into encoder efficiency and a special section on that subject is provided later in this document.

Frame Packetization

After the frame is encoded, it must be broken up into packets and sent out. Since the frame is much smaller, less bandwidth is required for the transfer over the PCIe, however even this small penalty can be reduced somewhat by placing the packetizer and network interface on the same card as the encoder. The actual packetization requires minimal time. However, video streams must stay within a certain bit rate to ensure they do not overrun access network bandwidths as well as the buffers on the client side. One solution is to buffer the packets to achieve the proper bit rate but this would impact latency. The typical solution is to lower the quality of the video during encoding (increase quantification/noise, lower resolution and/or frame rate) to achieve a constant bit rate. This allows the stream to stay within required bit rates without adding additional latency. As needed, a frame can also be skipped (dropped) to keep a stream within bandwidth requirements. A skipped frame will result in the previous frame being displayed twice on the client. The impact to the viewer is negligible, assuming the dropped frame was not referenced by the encoder for decoding another frame. Buffering can be reduced if packetization and the transmission of packets to the network occurs as each piece of a frame is completed by the encoder. This parallelization is discussed further in the "Frame Pipelining" section.

Frame Packets to ISP

Even after encoding, a typical 720P frame is 20K to 30K bytes (before the network abstraction layer (NAL) and Real Time Transport Protocol (RTP) headers), resulting in 15 to 30 IP packets. These packets are much larger and will take longer to send than the event packet. Except for this difference, the description and possible improvements described in "Event Packet to ASP" also apply here. A key element here and for the access network is to reduce jitter between frames. This is where a network element may delay one frame more than others causing two or more frames to arrive very closely spaced. Decoders are forced to buffer several frames to absorb this frame jitter. If the network can maintain the frame separation, the decoder can remove these buffers, which eliminates much of the latency.

Frame Packets to User

These 15 to 30 packets must now traverse the access network. Many access networks are optimized for traffic *to* the user rather than *from* the user, and offer higher speed and lower latency for downstream traffic. However, the speeds will not be nearly as fast as those in the Internet. The server may be able to send out the packets comprising a single frame in less than 5 ms from first to last using its fast network connection. When these packets hit the slower access network they will be buffered and spaced-out. A 25 Kbyte frame will take a minimum of 20 ms to traverse a 10 Mb/s cable link. Quality is also an issue here. With so many packets, there is the chance for dropped or modified packets, particularly with wireless connections. H.264 decoding relies on information from previous packets, so one lost or corrupted packet can affect the quality of many frames. Retransmit with Transmission Control Protocol (TCP) or at the

wireless node is not feasible due to the added latency. There are mechanisms to reduce this impact such as using slices [15] where an error only affects one piece of the frame rather then the whole frame. Another is using gradual decoder refresh (GDR) [17] as discussed later in the "Encoder Efficiency" section.

As with the "Event Packet to ISP" step there is much work that can be done in scheduling these real time video packets over the access network to decrease latency. Shared access networks such as fiber and cable actually provide 100 Mb/s to 2.5 Gb/s to the home. The scheduler provides a share to each customer, resulting in transmission speeds of 10 Mb to 100 Mb for each home. However, the scheduler could allow a frame's worth of packets to momentarily use a larger portion of the network, which would reduce or eliminate buffering. In addition, during congestion, a smart queue manager at the access headend could drop all packets for a frame rather than only one packet, since the dropped packet makes the rest of the packets useless. In addition, forward error control can be used to protect all or some of the more critical frames.

Decode Frame

H.264 is optimized such that most of the work is done during encode so decode requires less time and hardware. The primary challenge here is how the decoder was designed. A hardware decoder can decode a 720P frame in less than 4 ms. However, since most streaming video is buffered, it is logical to assume the power of the decoder was set to the maximum fps it must support. For 30 fps, that is 32 ms. At 60 fps, decoding must occur in under 16 ms. We have less control here since in many applications the service provider will not want to replace the set-top box. As discussed later in the "Frame Pipelining" section, the decoder can reduce the impact of a slower access network by starting the decode as soon as the first slice of a frame arrives. This is particularly true for wireless devices. In the low latency column in Table I, we assumed a latency-optimized decoder. However, if we use an existing set-top box, this figure may not be practical. OnLive provides its own set-top box, which provides latency-optimized decoding.

Display Frame

The primary concern here is that it is typical to buffer frames before starting the video to better tolerate jitter (variation in the time between packet arrivals). This buffering could add several seconds of latency. For non-interactive video this is not a problem. For interactive video we need to turn off buffering on the client device. This places an added burden on the server and network to supply these packets with very little jitter so that the video is not frozen or jerky. Use of gradual decoder refresh is important here to reduce the impact of I-frames to the frame buffer.

Even with no buffering, the incoming frame must wait for the next screen refresh, which even at 60 fps is every 16 ms. So on average it must wait 8 ms. The frame could be displayed immediately, but that would cause screen tearing. There is also delay added by the monitor itself. CRTs are best since they do not process the image before displaying it. Liquid crystal display (LCD) computer monitors are good, but some still can add significant delay [16]. With transmission to HDTV, there can be added delay to process the frame prior to display, and with some screens the delay can be as high as 50 ms [2] to enhance or resize the picture to the TV's natural resolution. This can be a real problem with HDTVs. Some have a "game mode" which reduces delay, but it can still approach 10 ms to 35 ms. The best solution (besides TV makers fixing their problem) is to output video to the screen at its native resolution. For most HDTVs that is 1920x1080, which requires double the bandwidth of 720P. Another solution would be to up-convert the resolution at the set-top box prior to sending it to the TV. This is what OnLive does with the console they offer for using their service with a TV.

Frame Pipelining

We have described a pipeline of tasks for the downstream processing and transmission of each frame:

- Rendering,
- PCIe transfer,
- Encoding,
- Network transmission, and
- Decoding.

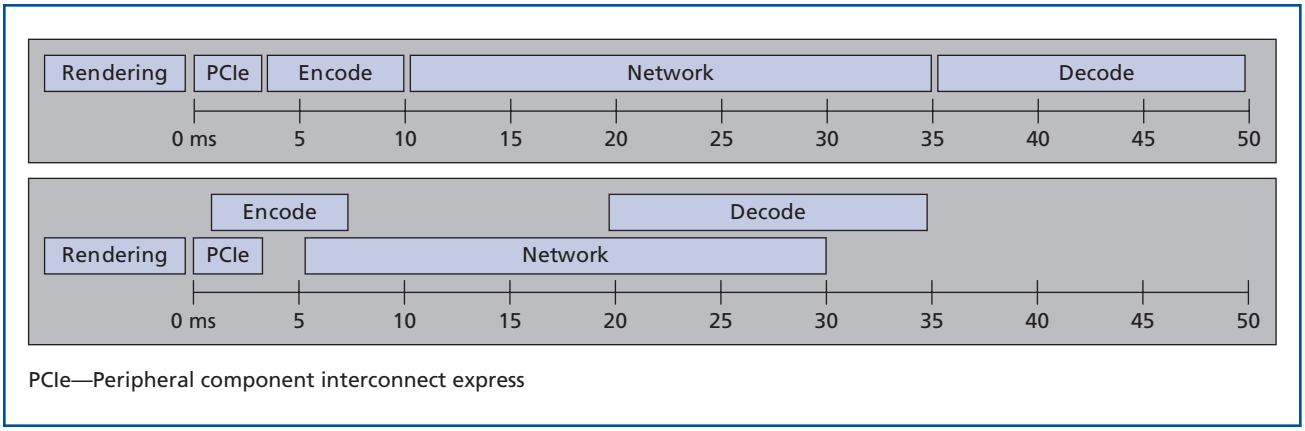


Figure 4.
Frame pipeline with and without parallelization.

Each requires precious microseconds to process a frame. However, each can break up the frames into segments (slices or packets) for processing. If each stage waited for the entire frame to arrive before starting, the latency incurred would be the sum of the total processing time for each stage, as shown in **Figure 4**. (The millisecond values in this figure are chosen just for demonstration.) However, if each stage started as soon as the first segment arrived, the latency penalty would be the total processing time for just the first segment, plus the time required for the last stage processing the rest of the packet. Note that we show no overlap for the rendering stage. GPUs finish segments of the frame in random order, so it is not possible to pass a segment of the frame to the PCIe until the whole frame has been rendered.

Pipelining also has the advantage of reducing the latency requirement for stages which are not in the critical path. For example, in Figure 3 there would be little impact to the latency if the length of the PCIe or the encoder stages were increased. The key is how fast the first segment can be processed at each stage. The network typically always represents the slowest stage, and thus hides much of the delay from the other stages.

Encoder Efficiency

In our work we have given encoding special emphasis. The primary reason is the expense of the encoding stage due to the heavy processing required. With the use of field programmable gate arrays (FPGAs) or custom multiprocessor chips, we can reduce that cost and support many video streams with a single encoder card. An important factor in these designs is latency. To maintain 30 fps, a real time (non-interactive) video encoder needs only to encode a frame in 32 ms. Of course we cannot afford a level of latency representing nearly one-third of our total target budget. There are several methods we use to reduce this latency. First, as described earlier, we turn off H.264 options which add to latency such as B-frames, look-ahead features, and extended prediction search. This lowers the effectiveness of the compression, but it is required for low latency encoding. Next, we use the H.264 option of splitting the frame into independent slices allowing us to spread the processing over several processing cores to work in parallel. In addition this allows the packetization stage to start as soon as the first slice is encoded, enabling the frame pipelining described in the previous section.

A potentially significant source of latency is a multi-frame frame buffer installed on at the decoder side to absorb jitter and large frames. This alone can add 100 ms or more to the latency. We want to eliminate this buffer. There are two key enhancements required to do so. First, the networks between the application provider and the end user must ensure the orderly flow of frames. Though the "bunching"

of packets making up a frame is not a problem, each frame in total should take approximately the same amount of time to traverse the networks. Since a real time encoder will separate frames by a fixed amount of time, the network needs only to send these frames to the client as quickly as possible with minimal or no buffering to prevent jitter.

The second requirement is for the encoder to ensure all frames are approximately the same size. By adjusting the quantization factor, most frames can be kept to the same size. However, an "I-frame" must be sent every so often (approximately every 5 frames to 100 frames) to prevent error propagation. An I-frame does not use other frames for prediction so its compression is much less efficient, which results in a much larger encoded frame. To avoid this, we can use gradual decoder refresh [17]. With this H.264 option, an I-frame is sent only at the start of the stream, and a portion of each successive frame is encoded without referencing other frames. In this way, GDR spreads the cost of I-frame functionality over many frames. It turns out that this method also increases resiliency to packet drops in the network over the I-frame method.

Resource Contention Penalty

One of the primary reasons to move applications to the cloud is the cost savings due to sharing of resources such as the CPU, GPU, and encoder. Games on your local PC or smartphone typically have exclusive use of the local resources when they are running. On a cloud server, we need to allow multiple customers to share the CPU, GPU, and encoder to spread the cost. However there can be an impact to latency. Applications or frames that are ready for a resource may have to wait until it is available. For example, if two frames from different applications are ready at the same time and delivered to the same encoder, the second frame will have to wait the time it takes to encode the first frame. The minimum latency times shown in Table I assume there is very little contention for resources.

Fortunately, events are not random and have a tendency to line up to some degree. For example, an application may have to wait for the CPU to send its first frame to the GPU. Its frame sequence will not line up with the first application, and thus the resulting frames will not be delivered to the encoder at the same time. However, this is a new area and much more work is required to understand the true queuing relationship. Some type of stream scheduling may be required to reduce the impact of resource contention. We can also reduce the latency of the encoder and/or enable it to process multiple frames in parallel. In addition, we can restrict access to the CPU and GPU to smaller segments of time to reduce penalties for resource request collisions.

Frames Per Second Tradeoff

There is a tradeoff between latency and throughput. As discussed, there are several places in the path where a frame must wait for the current frame to complete. This was mentioned in the "Applications Processing" and "Display Frame" sections. In addition, a higher frame rate can cause more resource contention situations. The standard rate for televisions in the U.S. is 30 fps (in Europe it is 25 fps), but the standard for computers is 60 fps and many operate at 72 fps. Some gamers like to set their fps rate even higher. At 30 fps, a frame is received and displayed every 32 ms, and at 60 fps it is of course twice as fast, at 16 ms. At a minimum, going from 30 fps to 60 fps will decrease latency by 16 ms on average, but in reality the gain will be greater. However, the cost of going to 60 fps is high, nearly doubling the required bandwidth. For wireless links this is not an option. Presumably to lower the latency, the OnLive service tries to achieve 60 fps and only lowers the frame rate if the game cannot do any better, or if the link bandwidth cannot handle it.

Mobile Considerations

More and more customers are using their mobile handsets and tablets as their primary device for communication, information, and entertainment. Providing interactive video over third generation (3G) and fourth generation (4G) networks presents an interesting set of opportunities and challenges. All of the factors contributing to latency discussed in this document pertain to the mobile architecture as well.

There are a few important differences. First, wireless technology offers lower bandwidth and greater error rate. Slower rates will add latency transmitting the frames. Encoding needs to be much more effective to reduce packet size.

To further reduce bandwidth, the frame rate will likely be reduced to between 20 fps and 25 fps. As noted in the previous section, this will increase latency as well. Errors will reduce quality. Methods should be employed to reduce their impact. Fortunately mobile customers have a history of being willing to trade quality for new capabilities.

LTE is a leading contender for the 4G crown. In an LTE network, packets from a mobile device must go from the base station node via a backhaul network to a set of gateways before they can access the ISP's data network or go on to the Internet. These gateways can often be a good distance away, increasing the speed of light penalty as well as any delay added by intervening routers. Placing content servers near these gateways will help to counter this cost.

In a home, it is tempting to do as OnLive has done and create a custom set-top box. However, for a mobile device we need to work with the available devices and operating systems. We must use the hardware decoders supported in these devices, which are typically H.264. The challenges discussed in the decoder section apply here and are made more difficult due to limited control over these devices. With faster processors in mobile devices, we may be able to decode via software, but there is also a possibility this may be no faster at all and it will consume more power. Finally, we could add some software to help other unique obstacles such as error tolerance.

The additional challenges imposed by mobile devices are greatly offset by the benefits users will receive from a personal cloud computing service which will provide near-limitless computing, storage, content, and networking capabilities not possible from their mobile device. Latency estimates from laboratory tests of LTE networks provide good promise that LTE will be able to support interactive video content. Advancements such as Alcatel-Lucent's lightRadio™ [1] may well provide the additional bandwidth required to support this new service at an affordable rate.

Conclusion

In this paper we looked at the elements which impact latency for interactive video. We have a goal of less than 100 ms of latency for the round trip from the event to seeing a response on the screen. Many factors account for this latency, some of which we can control and others we cannot. Much work is needed to reduce and maintain latency in the areas over which we have control. We can learn from OnLive, which has implemented a solution and has already hit most of these barriers. Due to Alcatel-Lucent's position with service providers, there are unique solutions we can implement that are not available to OnLive. One example is to move the interactive content server into the ISP network and closer to the customer, thus removing the latency caused by the Internet, the speed of light distance, as well as latency in the ISP's network. Other work can be done on the access side to improve packet scheduling in both directions. The initial step should be to build an interactive content service node and network, and implement these enhancements directly.

Acknowledgements

The author would like to thank the Bell Labs Cygnet team for their assistance with this work as well as this paper. I would also like to thank Ken Sharp and Elias Ayrey for their user experience testing support and contributions.

*Trademarks

Bluetooth is a registered trademark of Bluetooth SIG, Inc.
GTA is a registered trademark of Take-Two Interactive Software, Inc.
Mac is a registered trademark of Apple, Inc.
Netflix is a registered trademark of Netflix, Inc.
OnLive is registered trademark of OnLive, Inc.
PlayStation and PS2 are registered trademarks of Kabushiki Kaisha Sony Computer Entertainment TA Sony Computer Entertainment Inc.
Wii is a registered trademark of Nintendo of America, Inc.
Windows and Xbox are registered trademarks of Microsoft Corporation.

References

[1] Alcatel-Lucent, "lightRadio: Evolve Your Wireless Broadband Network for the New Generation of Applications and Users," <www.alcatel-lucent.com/features/light_radio/>.

[2] G. Block, "HDTV-Gaming-Lag: An Epidemic Exposed," IGN, June 12, 2006, <http://www.ign.com/articles/2006/06/12/hdtv-gaming-lag-an-epidemic-exposed>.

[3] M. Claypool and K. Claypool, "Latency Can Kill: Precision and Deadline in Online Games," Proc. 1st Annual ACM Conf. on Multimedia Syst. (MMSys '10) (Phoenix, AZ, 2010), pp. 215–222.

[4] J. Crisp and B. Elliott, Introduction to Fiber Optics, 3rd ed., Newnes, Amsterdam, Boston, 2005.

[5] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of Factors Affecting Players' Performance and Perception in Multiplayer Games," Proc. 4th ACM SIGCOMM Workshop on Network and Syst. Support for Games (NetGames '05) (Hawthorne, NY, 2005).

[6] S. Firestone, T. Ramalingam, and S. Fry, Voice and Video Conferencing Fundamentals: Design, Develop, Select, Deploy, and Support Advanced IP-Based Audio and Video Conferencing Systems, Cisco Press, Indianapolis, IN, 2007.

[7] T. Hoff, "Latency Is Everywhere and It Costs You Sales—How to Crush It," High Scalability, July 25, 2009, <http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it>.

[8] S. Hollister, "OnLive MicroConsole Official at $99, We Go Hands-On and Bombard You with Details," Engadget, Nov. 18, 2010, <http://www.engadget.com/2010/11/18/onlive-microconsole-official-at-99-we-go-hands-on/>.

[9] Institute of Electrical and Electronics Engineers, "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," IEEE 802.15.4, 2006, <http://www.ieee.org>.

[10] International Telecommunication Union, Telecommunication Standardization Sector, "One-Way Transmission Time," ITU-T Rec. G.114, May 2003, <http://www.itu.int>.

[11] R. Leadbetter, "Digital Foundry vs. OnLive UK," Eurogamer, Oct. 1, 2011, <http://www.eurogamer.net/articles/digitalfoundry-vs-onlive-uk>.

[12] OnLive, <http://www.onlive.com>.

[13] Pure E-Sports, "Mouse Optimization Guide," Jan. 11, 2007, <http://purehighspeed.com/forum/showthread.php?t=83>.

[14] M. Ricknäs, "LTE Will Improve Latency As Well As Speed," Techworld, Mar. 3, 2010, <http://news.techworld.com/mobile-wireless/3214090/lte-will-improve-latency-as-well-as-speed/>.

[15] R. Schäfer, T. Wiegand, and H. Schwarz, "The Emerging H.264/AVC Standard," EBU Tech. Rev., 293 (2003), <http://tech.ebu.ch/docs/techreview/trev_293-schaefer.pdf>.

[16] T. Thiemann, "An Investigation of the Test Process Used to Date for Determining the Response Time of an LCD Monitor, Known as Input Lag," PRAD, Aug. 8, 2009, <http://www.prad.de/en/monitore/specials/inputlag/inputlag.html>.

[17] Y.-K. Wang and M. M. Hannuksela, "Gradual Decoder Refresh Using Isolated Regions," Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG (ISO/IEC JTC1/SC29/WG11 and ITU/T SG16 Q.6), Doc. JVT-C074, May 2002, <wftp3.itu.int/av-arch/jvt-site/2002_05_Fairfax/JVT-C074.doc>.

[18] M. West, "Measuring Responsiveness in Video Games," Gamasutra, July 16, 2008, <http://www.gamasutra.com/view/feature/3725/measuring_responsiveness_in_video_.php?page=3>.

(Manuscript approved March 2012)

RON SHARP is a distinguished member of technical staff in the Bell Labs' Access Solutions department in Murray Hill, New Jersey. He has worked at Bell Labs for over 25 years, and has published many papers on network security and high speed networking, including four for this journal. He authored a book on Internet firewalls and holds five patents, with six more currently pending. He is a retired U.S. Air Force Officer. Mr. Sharp earned a B.S. degree in computer science and systems design from the University of Arkansas, Fayetteville, and an M.S. from the University of Texas. He was one of the original three engineers who created the Alcatel-Lucent VPN Firewall Brick®. His current research work is in the area of cloud computing with emphasis in video encoding and embedded multiprocessors. ◆