Latency

and the Quest for Interactivity

Stuart Cheshire*

cheshire@cs.stanford.edu

http://www.stuartcheshire.org/

November 1996

Abstract

Many kinds of current network hardware, modems especially, have latencies many times worse than they need to be. There's no strong technical reason why the latencies are so bad; the root cause is one of perception. The buying public, and modem manufacturers, concentrate solely on one parameter — throughput — so not surprisingly latency gets ignored and as a consequence suffers badly.

Poor latency, not limited throughput, is the factor that is hindering the development of a whole class of networked application software — interactive games, conferencing software, collaborative work software and all forms of interactive multi-user software.

"The entire focus of the industry is on bandwidth, but the true killer is latency."

Professor M. Satyanarayanan,
keynote address to ACM Mobicom '96.

Introduction

Interaction is fundamental to human experience.

Whether we're dealing with people or objects, interaction is essential. We perform some action, and when we observe how the person or object reacts we modify our behaviour accordingly. Sometimes the reaction needs to be very quick, like when we're having a conversation or fighting with swords, and sometimes a slow reaction is adequate, such as carrying out a lengthy postal correspondence with a distant relative, but it is rare that we perform an action without wanting some kind of feedback by which to observe the results of that action.

We would like to be able to use computers and computer networks to facilitate not only slow human interactions like written correspondence, but also rapid interactions as well.

This leads us to two questions:

Firstly, how fast would response times need to be to provide us with the full gamut of human interactive experience? The answer to this question tells us the target interactive response time that we should be aiming for.

Secondly, can computer networks achieve that target response time? Is there some physical limit that prevents us from reaching the required performance? As we will see, the speed of light does become a serious limitation when we are communicating with people on the other side of the planet. The good news is that most of the time we are communicating with people much closer than that, and most of the problems with current computer networks are things that are within our power to fix (unlike the speed of light, which we are unlikely to change).

Many components of today's computer systems — modems especially — are nowhere near to the limits imposed by the speed of light and offer huge opportunities for quite dramatic improvements. The rewards are there for the taking, for those prepared to meet the challenge.

*This paper was commissioned by Volpe Welty Asset Management, L.L.C.

Section 1: Limits of Human Perception

Our first question was, "How fast would response times need to be to provide us with the full gamut of human interactive experience?"

A number of parameters of human perception that might help us answer this question have been studied extensively.

Audio

It is known that typical human hearing cannot detect frequencies much above 20kHz. Shannon's sampling theorem says that a sampling rate of 40 000 samples per second can record frequencies up to 20kHz. Compact Discs are recorded at 44 000 samples per second and Digital Audio Tape is recorded at 48 000 samples per second. The decision to use 48 000 samples per second for DAT was made not because the quality would be higher — human beings are unable to tell the difference — but simply because they wished to use a rate that was significantly different to CDs to prevent music pirates from being able to easily make direct binary copies from CD to DAT. There is no point doing computer audio for human listeners at rates higher than 48 000 samples per second.

Video

Similar parameters are known for the human visual system. The threshold of motion fusion — the point where a rapid sequence of still images creates the illusion of movement — is about 8-12 frames per second. Typical Saturday morning television cartoons are made at this frame rate. The threshold of *smooth* motion is about 24-30 frames per second. Cinema films and television pictures run at this frame rate. Interestingly, the threshold of flicker perception is higher — about 50-75 flashes per second. Even though 30 images per second create a convincing illusion of smooth motion, a light flashing 30 times per second creates visible flicker. That's why NTSC television has 30 *frames* per second, but uses a technique called interlacing to divide every frame into a odd and even *field*, to produce 60 flashes of light per second so that the flicker is not visible. PAL television operates at 25 interlaced frames per second giving 50 fields per second. Cinema movies are filmed at 24 frames per second, but high quality projection equipment displays those 24 frames per second using two flashes of light per frame, to produce 48 flashes per second. Some people are able to perceive a slight flicker at that rate. High quality computer monitors operate at 75 (non-interlaced) frames per second, which is high enough that no one can see the flicker. It is surprising, I think, to find that all the world's display systems fall into such a relatively narrow range.

Interactivity

Given that other human perception parameters are known so accurately, you might think that the threshold of interactive response would be another well-known, widely studied parameter. Unfortunately it is not. For a number of years I've been interested in this question and I've asked a number of experts in the field of computer graphics and virtual reality, but so far none has been able to give me an answer. There seems to be a widespread belief that whatever the threshold of interactive response is, it is definitely not less than 15ms. I fear this number is motivated by wishful thinking rather than by any scientific evidence. 15ms is the time it takes to draw a single frame on a 66Hz monitor. Should it turn out that the threshold of interactive response is less than 15ms, it would be dire news for the entire virtual reality community. That would mean that there would be no way to build a convincing immersive virtual reality system using today's monitor technology, because today's monitor technology simply cannot display an image in under 15ms. If we find that convincing immersive virtual reality requires custom monitor hardware that runs at say, 200 frames per second, then it is going to make effective virtual reality systems prohibitively expensive.

Given that the one universal truth of all current head-mounted virtual reality systems is that they rapidly induce nausea in the viewer, I'm not yet prepared to rule out the possibility that threshold of interactive visual response might turn out to be less than 15ms.

However, this news is not as hopeless as it may sound. There may be more than one threshold of interactive response, for different kinds of interactivity. The eyes are very closely coupled with the brain, both neurologically and in terms of physical location. Helping to maintain balance is one of the more demanding tasks the visual system is required to perform. It allows us to perform the remarkable feat of being able to run over rough terrain without falling down. It is to be expected that the mental processes that coordinate head movements with signals from the inner ear and from the eyes to achieve this would require very fast pathways from the eyes and the ears.

It is almost certain that the response time required for convincing hand/eye coordination is much less. For one thing, the hands are a lot further from the brain that the eyes are, and it takes signals considerably longer to travel that distance through the nerve fibres. One ramification of this is that although it may be difficult to achieve wide-area network response times under 15ms, that may not be necessary. The changes in the visual display that are needed in response to a movement of the head can be computed locally without needing any communication over the network. Interactions with the virtual world, such as picking up an object from a table, may need communication over the network, but such communication will probably not need the very fast response time that head-tracking interactions do.

Computing responses locally is not a new idea. Even the human body does it. If we touch a hot object we react by quickly withdrawing the limb in question. That flinch reflex is controlled by nerves in the spinal column, not the brain, because sending the pain signal all the way to the brain and waiting for a response would take too long. Sometimes the flinch reflex is the wrong reaction, but generally our reflexes protect us more than they hurt us. One of the challenges of using local computation is to try to minimize both (a) the number of cases where the wrong response is computed, and (b) the adverse effects that occur if that does happen.

100ms

As another data point, the rule of thumb used by the telephone industry is that the round-trip delay over a telephone call should be less than 100ms. This is because if the delay is much more than 100ms, the normal unconscious etiquette of human conversation breaks down. Participants think they hear a slight pause in the conversation and take that as their cue to begin to speak, but by the time their words arrive at the other end the other speaker has already begun the next sentence and feels that he or she is being rudely interrupted. When telephone calls go over satellite links the round-trip delay is typically about 250ms, and conversations become very stilted, full of awkward silences and accidental interruptions.

For these reasons I'm going to suggest 100ms as a target round-trip time for general network interaction. Some interactions may require a faster response time than this, but these interactions should hopefully be ones that can be computed locally without resorting to network communication. Some styles of interaction, like playing a game of chess, may work effectively with delays longer than 100ms, but we don't want to limit ourselves to only being able to support these limited kinds of interaction.

There are also 'latency hiding' techniques that can help create the illusion of the latency being lower than it really is, but these techniques don't provide true interactivity; they only provide the illusion of interactivity. The illusion of interactivity may be better than no interactivity at all, but it is not as good as the real thing

Consider a computer sword fighting game. You make a thrust with your sword, your opponent sees the thrust and parries, you see his parry and adjust your attack accordingly. For this kind of tightly coupled interaction to work, your thrust has got to be communicated over the network to your opponent and drawn on his screen, and his response to what he sees on the screen has to be communicated over the network back to you and drawn on your screen before you can see it and respond. An example of a latency-hiding technique in this case would be to have your own computer predict reactions locally. When you make a thrust with your sword, your computer predicts that your opponent will parry in a certain way and updates your screen appropriately, without waiting for the real response to actually come back over the network. In this case you are not really interacting with your opponent; you are interacting with your computer, but you have the *illusion* that you are interacting with your opponent. Very good predictive software could make it a very convincing illusion, but it is still an illusion. If your opponent cleverly responds to your thrust in a different, unexpected way, then you're going to find that what's happening on your screen is not the same as what's happening on his. Your computer's locally predicted response was wrong in this case, and your computer now has the problem of finding some way to resolve the inconsistency without you noticing. If, on your screen you sliced him through the heart, but on his screen he deflected the blow, it is going to be pretty difficult for the computer to resolve that without the players noticing.

Latency hiding techniques are valuable, but they are no substitute for the real thing. Our target should be to get true interactivity with a response time of 100ms or less. Since we know it takes the electron beam of the monitor at least 15ms to draw the image on the screen's phosphor at each end, that accounts for 30ms of the time delay, leaving only 70ms[1] for network communication, or 35ms each way (and that's optimistically assuming that the software overhead is negligible). A 70ms round-trip delay is no easy goal to aim for, but it is important to know the challenge that lies ahead.

[1] In Wired 4.10, page 133 (October 1996) James Martin also quoted the same target round-trip delay: 70ms. (Unfortunately the rest of his logic was flawed. He forgot that to determine response time you need to measure the round-trip delay, not the one-way delay, and he used the speed of light in vacuum for his calculation, not the speed of light in glass fibre, which is typically 33% less. This made his final result out by a factor of three, but his initial value of 70ms was correct.)

Some people claim to have written (and patented) software than can invisibly mask the effects of network round-trip delays of one second or even more, and they are actually glad that modems have poor latency because they think it gives value to their patents. This belief reflects a failure to understand the basic nature of the problem. Let me illustrate with another simple example. Say I toss a ball across the room to you in our simulated computer world. I need to see on my screen whether you (a) caught the ball, (b) tried to catch the ball but failed, or (c) ignored it and let it fall to the floor. To achieve this, my computer needs to tell yours that I threw the ball, yours needs to display that on its screen, you need to decide to react (or not, as the case may be), and your computer needs to tell mine how you responded. Only then can my computer update its screen to display the appropriate outcome. Unfortunately if I toss a ball across the room to you in the real world, you will have caught it (or not) half a second later. If our network round-trip delay is one second, then there is no way that our computer simulated world can ever reproduce this behaviour. There is no way for me to find out how you actually responded within the half second time interval necessary to display the result on my screen. My computer could predict how it expects you to respond, but if it predicts wrongly, the illusion of reality in our artificial world will be shattered. Another obvious 'solution' is to insert a one-second delay between me pressing the control to throw the ball and my character in the world actually throwing the ball. This would allow your computer the one second it needs to update your screen and communicate your response back to me in time to draw it on my screen, but it hasn't actually solved the true problem. This 'solution' does not facilitate our desired interaction — me pressing the control to throw a ball to you and seeing half a second later whether or not you caught it — it simply prohibits any interaction that rapid. Remember in the introduction I stated that our goal is to support the *full gamut* of human interactive experience, not to prohibit the interactions that are difficult to support

Latency hiding techniques are valuable, but as explained before they are a necessary evil, not a desirable alternative to true low-latency communication.

Finally, you'll notice that so far I've not made any mention of the *amount* of data we have to send, only how long it should take. The amount is secondary. The primary concern is the time it takes. Bandwidth is secondary. Latency is the critical factor. This is odd, don't you think? Currently, modem makers care only about throughput, not at all about latency. The throughput of a modem is written in huge type across the front of every box, but the latency is nowhere to be seen. This is a theme we will return to.

Section 2: Limits of the Physical Universe

Our second question was, "Can computer networks ever achieve our target response time of 70ms?"

We're not used to worrying about the speed of light being a problem in everyday life but in this case it can be.

The speed of light in vacuum is 300 000 km/sec and the speed of light in typical glass fibre is roughly 33% less, about 200 000 km/sec. How long would it take to send a signal to someone on the far side of the planet and get a response back?

Speed of light in vacuum	= 300 000 km/sec
Speed of light in glass fibre	= 200 000 km/sec
Circumference of earth	= 40 000 km
Total round-trip delay to far side of the planet and back	= 40 000/200 000 = 200ms

So, sadly, the best response time we can hope to get to someone on the other side of the planet is 200ms, a long way short of our target for interactivity. Interactions with people on the far side of the planet are going to need the benefit of latency-hiding techniques, and true tightly-coupled interactivity will not be possible. We will probably never be able to have a truly interactive virtual sword fight with someone on the far side of the planet.

However, we're not usually communicating with people on the far side of the planet. Most of the time we're communicating with people much closer to home. Might we be able to get true interactivity with people closer to us? Could we do it within the United States?

Distance from Stanford to Boston	= 4400km
One-way delay to Boston	= 4400/200 000 = 22ms
Round-trip to Boston	= 44ms

This result is promising. Within the United States we should be able to achieve true interactivity. The speed of light consumes 44ms of our 70ms time budget, leaving us with 26ms for software overhead in the network, the endpoints, and the graphics systems. Not an easy challenge, but at least it's nice to know it's possible.

Interestingly, the current Internet is already doing quite well in this respect. If I ping a well-connected host at

MIT from a well-connected host at Stanford, the measured round-trip delay is about 79ms:

```
> ping -c 1 lcs.mit.edu
PING lcs.mit.edu (18.26.0.36): 56 data bytes
64 bytes from 18.26.0.36: icmp_seq=0 ttl=239
time=78.6 ms
```

The Internet backbone is already operating within a factor of two of the theoretical best possible round-trip delay. This is very reassuring. Over the next few years we can expect the latency of the Internet backbone to drop steadily towards the theoretical minimum. Coast-to-coast interactivity is within our grasp.

Section 3: Current Computer Hardware

Modems

Those of us not lucky enough to have a direct Internet connection have to dial in to an ISP using a modem. Modems have much lower bandwidth than Ethernet, but that's not a problem for most interactive applications like games. What matters is the time delay — the latency — not the bandwidth. Interactive applications often don't need to send very much data, but the little bits they do send should be sent quickly, to create a quality interactive experience.

Unfortunately, current modems fail very badly in this respect. Say I'm 10km from my ISP (Internet service provider). What is the speed of light limit over this distance?

Speed of light in glass fibre	= 200 000 km/sec
Round-trip distance	= 20 km
Speed of light round-trip delay	= 20/200 000 = 0.1ms
Actual round-trip delay using a typical current modem	= 260ms

Current modems fall a long way short of the physical limits imposed by the speed of light. What is going wrong here?

Let us analyse what is happening. The time delay for a single packet one-way transmission has two components:

1. Queuing delay in the network

2. Transmission time

The queuing delay in the network occurs when network links or routers are congested, and packets have to be queued. With techniques like packet prioritization and Weighted Fair Queueing [Dem89] [Par92] the queuing delay can be reduced effectively to zero. Bulk file transfer traffic may be delayed by queues, but because interactive traffic always goes to the front of the queue, from its point of view it is as if there were no queue there at all.

Also, at the user's own modem, unless the user is doing something else at the same time as playing the game, there should be no queueing at all.

That leaves us with transmission time to consider. At each hop, the transmission time has two components:

1. Per-byte transmission time

2. Fixed overhead

Per-byte transmission time is easy to calculate, since it typically depends only on the raw transmission rate. The fixed overhead comes from sources like software overhead, speed of light delay, etc.

For a wide area link, such as a 1000km T3 line at 45Mb/sec, the per-byte component of the transmission time is 0.178μs per byte, or 17.8μs for a 100 byte packet. This is eclipsed by the overhead due to the speed of light, which at 5ms is much larger than any of the other components. So for wide area links the per-byte component of the transmission time is negligible compared to the speed of light delay.

For modems, it is a different story. The distance is typically not very far, so speed of light delay should be negligible. The data rate is low, so it takes a long time to send each byte. The per-byte transmission time should account for most of the time taken to send the packet. To send 100 bytes over a 28.8 modem should take:

100 bytes * 8 bits per byte / 28800 bits per second
= 28ms

That means the round-trip should be 56ms. In fact it's often more like 260ms. What's going on?

There are two other factors contributing to the time here:

1. Modems are often connected via serial ports.

Many modem users assume that if they connect their 28.8 modem to their serial port at 38.4 kbit/sec they won't limit their performance, because 38.4 > 28.8. It's true that the serial port won't limit their throughput, but it will add delay, and delay, once added, never goes away:

100 bytes * 10 bits per byte / 38400 bits per second
= 26ms for the computer to send 100 bytes down serial port to the modem

2. Modems try to group data into blocks.

The modem will wait for about 50ms to see if more data is coming that it could add to the block, before it actually starts to send the data it already has.

Let's see what the total time is now:

26ms (100 bytes down serial port to modem)

50ms (modem's waiting time)

28ms (actual transmission time over telephone line)

26ms (100 bytes up serial port at receiving end)

Total time = 130ms each way, or 260ms round-trip.

To make things worse, if both players are connected to their respective ISPs by modem, then the total player-to-player round-trip delay is 520ms, which is clearly hopeless for any kind of tightly-coupled interactivity, and this is reflected in the state of today's networked computer games.

Can we do anything to improve this?

One thing to notice is that the 38400baud serial connection between the computer and the modem, which many people don't think of as being the bottleneck, turns out to be responsible for 52ms of the delay. It's the biggest single contributor to delay — almost twice as much as the actual communication over the telephone line. What can we do about this? If you can connect the modems at both ends at 115200baud instead of 38400baud, the serial port delay can be reduced to 9ms at each end. Better, still if you can use an internal modem on a plug-in card instead of one connected through a serial port, the serial port delay can be eliminated entirely, leaving us with a round-trip delay of only 156ms.

Having eliminated the serial port delay, the next biggest contributor to delay is the fixed 50ms overhead. Why is there a fixed 50ms overhead at all? The reason is that modern modems offer lots of 'features'. They do compression and automatic error correction. To get effective compression and error correction they have to work on blocks of data, not individual characters, and that means that as characters arrive down the serial port they have to corral them into a buffer until they have a block big enough to work on efficiently. While the characters are being accumulated in the buffer, they're not being sent over the phone line. 100 bytes is not really enough for the modem to work on effectively, so it would like a bigger block. After you have sent the 100 bytes to the modem, it waits to see if more characters are coming. After some time — about 50ms — it realizes that no more characters are coming, so it had better compress and ship what it has. That 50ms the modem spends watching an idle serial port hoping for more data is wasted time. The time is lost. The packet has now been delayed by an unnecessary 50ms, and there's no way to 'un-delay' it.

The reason for this design decision is that modems were originally designed with remote terminal access in mind. They were designed to take the characters that a human types and group them together in little blocks to send. They were designed to take the characters that the mainframe at the other end printed out, and group them together in little blocks to send. The only indication the modem had that the user had finished typing, or that the mainframe had finished printing what it was going to print, was when the data stream paused. No one was going to tell the modem that the data had finished and no more would be coming for a while. It had to guess.

This is no longer the case. Most people these days are using modems to connect to the Internet, not old mainframes. Internet traffic is made up of discrete packets, not a continuous stream of characters, but the modem doesn't know that. It still thinks it is sending a continuous stream of characters (from someone who types really fast).

There's a really simple fix for this problem. We could simply make modems be aware that they are sending Internet packets. When a modem sees the SLIP (Serial Line IP) End-Of-Packet character (0xC0), it should realize that the packet is now complete, and immediately begin work on compressing and sending the block of data it has, without waiting for a 50ms pause. This simple fix would eliminate the 50ms fixed overhead, and should allow us to achieve a 56ms round-trip delay over a modem SLIP connection — almost five times better than what typical modems achieve today.

It should be realized that this solution fixes the symptom, not the underlying root cause of the problem. The root cause of the problem is that the modem is not aware of what is going on. It doesn't know that a DNS (Domain Name Service) packet is a very urgent packet, because the computer is stuck waiting, unable to proceed until it receives the answer, whereas a TCP acknowledgment packet is less urgent, and delaying it a little won't cause any great problem. The same thing applies to compression. The modem will try to compress image data that has already been compressed with JPEG, and video data that has already been compressed with MPEG. All of it's futile efforts to help result in no benefit, and just add crippling latency to the transmission time. The modem thinks it's helping by trying to compress the data. I'm reminded of the line from the film *Total Recall*: "Don't think. I don't give you enough information to think."

The correct solution to this problem is that the modulation/demodulation functions of the modem hardware should be more closely tied to the host computers CPU and its networking software, which does know what is going on. It knows which data might benefit from compression, and which will not. It knows which data is urgent, and which data is not.

Ironically, Apple could do this today with the hardware it already has. The Apple Geoport telecom adapter, which has suffered so much criticism, may offer an answer to this problem. The Apple Geoport telecom adapter connects your computer to a telephone line, but it's not a modem. All of the functions of a modem are performed by software running on the Mac. The main reason for all the criticism is that running this extra software takes up memory slows down the Mac, but it could also offer an advantage that no external modem could ever match. Because when you use the Geoport adapter the modem software is running on the same CPU as your TCP/IP software and your Web browser, it could know exactly what you are doing. When your Web browser sends a TCP packet, there's no need for the Geoport modem software to mimic the behaviour of current modems. It could take that packet, encode it, and start sending it over the telephone line immediately, with almost zero latency.

Sending 36 bytes of data, a typical game-sized packet, over an Apple Geoport telecom adapter running at 28.8kb/s could take as little as 10ms, making it as fast as ISDN, and ten times faster than the current best modem you can buy. For less than the price of a typical modem the Geoport telecom adapter would give you Web browsing performance close to that of ISDN. Even better, all the people who already own Apple Geoport telecom adapters wouldn't need to buy anything at all — they'd just need a software upgrade. So far Apple has shown no interest in pursuing this opportunity, which is good news for everyone else in the industry.

I should clarify that I'm not saying that having a modem do compression *never* helps. In the case where the host software at the endpoints is not very smart, and doesn't compress its data appropriately, then the modem's own compression can compensate somewhat for that deficiency and improve throughput. The point is that modem compression only helps dumb software, and it actually hurts smart software by adding extra delay. For someone planning to write dumb software this is no problem. For anyone planning to write smart software this should be a big cause for concern.

Service Models

Having considered modem-induced delays, we should also consider delays within the network itself.

The appropriate service model for a worldwide communications network is a hot topic of debate. On the one side, the computer community favours a datagram model where individual packets are addressed and delivered like letters in the US mail. On the other side, the telephony community favours a virtual-circuit model where predictable connections are set up, used, and then torn down, like telephone calls.

This difference of opinion can be summarised thus:

> The computer community says, "Give me your packet, and I'll do my best to deliver it."
>
> The telephony community says, "Tell me in advance what your bandwidth requirement is, and I'll set you up a guaranteed virtual-circuit if I can, or give you a busy signal if I can't."

The big debate basically comes down to the question of whether all types of communication can be described in terms of a particular 'bandwidth requirement' (or indeed whether *any* types of communication aside from voice telephone calls can be characterized that way). Do ATM-style virtual circuit connections with bandwidth reservations have anything useful to offer for traffic other than voice data?

A sound wave is a continuous phenomenon, like the flow of water through a pipe. Computers however like to deal with discrete events, like key strokes and mouse clicks, so to deal with sound computers (and the digital telephone network) break the continuous wave form into discrete units by sampling it with a analogue-to-digital converter. These samples are taken very rapidly, at 8000 times per second for low-quality audio up to 48000 times per second for high quality audio. Although this is a finite number of discrete samples per second, there enough of them that for the purpose of networking we can treat audio data as if it were a continuous fluid stream.

This makes audio data very predictable. It begins, it continues for some duration with some fixed data rate, and then it ends. For the duration of the sound, it behaves like a continuous flow, like water in a pipe. To play sound we have to deliver the sampled values to the sound hardware at very precise intervals. For telephone quality audio there are 8000 samples per second, so we have to deliver one sample every 125μsec. That's a fairly exacting real-time requirement, but each sample is a small piece of data — typically 8 or 16 bits.

In contrast video data is quite different. Each sample (frame) is much bigger, often several kilobytes, but there are only 30 of them per second, or less. What's more, if a single occasional video frame is lost or delayed, the screen can continue showing the previous frame and the viewer will most likely not notice. In contrast, if we were to lose $^1/_{30}$ of second of audio data, the listener would most definitely hear a disturbing click. In addition, compressed video frames are not all the same size. This means that video does not really have a fixed bandwidth requirement like audio does, and it's not even that sensitive to occasional losses. What it does need, like audio, is low latency delivery.

Lots of applications, not just voice and video, require good interactivity and hence low latency. The key point is that while these applications need low latency, they don't need a *continuous* stream of low latency packets the way voice does. If my tank in Bolo[2] is traveling North, with a certain speed, turning at a certain rate, etc., then all the other machines playing the game can continue to extrapolate the smooth movement of the tank without needing to see a constant flow of packets. I don't need to send another packet until I *change* my speed, or rate or turn, etc. That smooth extrapolation of movement is called dead-reckoning. Dead-reckoning works fine for displaying convincing-looking smooth motion until the vehicle hits or otherwise interacts with one of the other participants. It is how to solve those interactions that is the problem. A vehicle just floating along, minding it's own business, not interacting with anything is easy, and dead-reckoning is a good way to get smooth motion without excessive network traffic. The life of a vehicle is typically long periods of uninteresting movement, punctuated by brief episodes of interaction. It is to handle those brief episodes of interaction that we need to send occasional low-latency packets.

This is why reservations don't buy us what we need. A reservation reserves a continuous uniform data flow at a certain rate. What interactive software needs is not a continuous uniform data flow, but low latency for the occasional intermittent packets it does send. Only voice needs a continuous uniform data flow. We need networks that can deliver us low-latency without requiring us to reserve (and pay for) a continuous bandwidth stream for our occasional intermittent packets.

To get good latency what we need is prioritization of network packets, not virtual-circuits with reserved bandwidth. We need to be able to mark packets (say with bits in the header) to indicate their level of urgency. Of course, network service providers may decide to charge more for sending priority packets. Users who wish to pay more for better quality service will still be able to. The difference is that the better quality service will in the form of high priority packets not virtual circuits, reservations and guarantees.

Some computer network game companies are already using techniques like this. Rather than using bits in the header to indicate packet priority, they make deals with Internet service providers that packets from their IP addresses should be given top priority and sent out first. This makes no detectable difference to the ISP or its other customers because video games produce so few packets compared to other uses like Web browsing, but it makes a big difference to the quality of interaction that that game players experience. Now, if only they weren't using such slow modems…

[2] Multi-player networked tank battle game for the Apple Macintosh

Section 4: Conclusions

To improve the quality of computer network interaction, we need to do two things:

1. We need to aggressively eliminate all causes of unnecessary latency in our computer hardware and software.
2. For interactions with the other side of the planet, we can never beat the speed of light, so we need to develop latency-hiding techniques that give us the illusion of interactivity when true interactivity is not possible.

As long as customers think that what they want is more throughput, and they don't care about latency, modem makers will continue to make design decisions that trade off worse latency for better throughput.

Modems are not the only problem here. In the near future we can expect to see a big growth in areas such as ISDN, Cable TV modems, ADSL modems and even Wireless 'modems', all offering increases in bandwidth. If we don't also concentrate on improved latency, we're not going to get it.

One first step in this process would be for the industry to adopt a modem latency rating scheme. TEN, MPath, Catapult, Sandcastle and the other network gaming companies could collaborate to set this up. Modems that can achieve a round-trip delay below say, 100ms, could be authorized to place a sticker on the box saying "Gameplay approved" and the gaming companies could encourage their customers to buy only modems that are "Gameplay approved".

If we don't start caring about latency, we're going to find ourselves in a marketplace offering nothing but appallingly useless hardware.

References

[Dem89] A. Demers, S. Keshav and S. Shenkar, Analysis and Simulation of a Fair Queueing Algorithm, Proceeding of ACM SIGCOMM '89, Zurich, Switzerland, pp. 3-12, August 1989.

[Par92] A. Parekh, A Generalized Processor Sharing Approach to Flow Control - The Single Node Case, Proceedings of IEEE INFOCOM '92, San Francisco, March 1992.