Fighting Physics: A Tough Battle

Thinking of doing IPC over the long haul?
Think again. The laws of physics say you're hosed.

Over the past several years, SaaS (software as a service) has become an attractive option for companies looking to save money and simplify their computing infrastructures. SaaS is an interesting group of techniques for moving computing from the desktop to the cloud; however, as it grows in popularity, engineers should be aware of some of the fundamental limitations they face when developing these kinds of distributed applications—in particular, the finite speed of light.

Jonathan M. Smith, University of Pennsylvania

Fighting Physics: A Tough Battle

Consider a company that wants to build a distributed application that does IPC (interprocess communication) over the long haul. The obvious advice is “just say no”—don’t do it. If you’re going far outside your local networking environment, the physics of distance and the speed of light, combined with the delays that come from the Internet’s routing infrastructure, tell us that it will be much too slow. These concepts are not generally understood, however, and even when they are, they’re sometimes forgotten.

So, exactly what are the basic principles related to speed of light and network hops that all software developers need to acquaint themselves with? This article answers that question by first working out some quantitative preliminaries with an example, moving on to the networking implications, and then covering applications. Finally, it provides some rules of thumb to keep in mind as applications and architectures evolve in reaction to new network capabilities and unchanging physics.

PRELIMINARIES: THE PHYSICS

The speed of light in a vacuum is exactly 299,792,458 meters/second.¹ This is as fast as you can move a bit of data, and according to our current understanding of physics, it is a fundamental constraint of the universe we live in. In fiber, the speed of light is 2.14×10^8 meters/second or about 70 percent of the speed of light in a vacuum. If a fiber were stretched in a straight line from New York to San Francisco, it would be about 4,125 kilometers long, and it would take about 19 (4,125 ÷ 214) milliseconds for light to make the one-way trip. Assuming an 8,250-km length of fiber was used, you can just double this time to get an estimate for minimum round-trip time.

At first glance, 19 ms might seem like a short time, certainly on a human scale. As computer scientists, however, we are usually concerned with a different time scale: that of the computer. Here we can calculate the 19 ms

in terms of instructions, the fundamental units of work for computers. As an example, we can use a 2003-vintage single-core machine: the Intel Pentium 4 Extreme Edition, which at a 3.2-GHz clock rate was rated at 9,726 million instructions per second: $9,726 \div 0.019$ is 184 million instructions—sufficient, for example, to search through or sort millions of names.

It is always important to keep in mind that the purpose of computer networking is to interconnect computers, and that computers operate on very short timescales. Also, a single human operation sometimes translates to many computer operations (i.e., round-trips). For example, opening a single Web page usually requires many round-trips, even if you are getting only a single large object (e.g., a large picture), as is discussed further later in this article.

PROPAGATION, BANDWIDTH, LATENCIES, AND HOPS

The traversal of the fiber loop between New York and San Francisco presumes a data-transfer unit of a single encoded binary digit of information. The lower bound for that traversal would be 219, or 38 ms (or 368 million instructions). The time for this bit to travel from its source to its destination and back again is called its *propagation delay*.

Propagation delay is important, but compared with the much more common metric of bandwidth—measured in bits per second—it is rarely quoted as a figure of merit. At least partially, this is because the observed propagation delay depends on context, whereas bandwidth (say of a fiber-optic transmission system) can be measured in isolation. Bandwidth can also be increased through engineering (for example, through encoding schemes for transmission systems that encode multiple bits per symbol) and thus is more attractive as a figure of merit to those who build transmission systems. Finally, bandwidth is a measure of work, which is attractive to purchasers.

Bandwidth can also affect latency, which is distinct, in my view, from propagation delay; the propagation delay

is a metric for the first bit, while latency is a metric for the entire data unit, which may contain more than one bit. In general:

$$\text{latency} = \text{propagation delay} + \frac{\text{data unit size}}{\text{bandwidth}}$$

What this basically says is that the propagation delay is only part of the picture and that bandwidth affects performance as well. A look at the impact of the bandwidth in an example system shows why propagation delay is so important. Consider a 10-Gbps transmission system and a 1,250-byte (or equivalently, 10 Kbit, chosen both to reflect a reasonable maximum transmission unit with Ethernets and to make arithmetic easier!) data unit. The propagation time for the first bit in the NY/SF loop is 38 ms, and the last bit arrives a microsecond (10K/10G) later, making the total latency 38.001 ms. The majority of the latency is propagation delay. An interesting arithmetic exercise is to compute the distance at which a transmission system's latency is double the propagation delay. For a 10-Gbps transmission system and 10-Kbit data-unit size, this is about 214 meters, or a few city blocks. For smaller data units or longer distances, propagation delay is the majority of the latency. (John Shaffer and I offer more detail on propagation delay versus latency in a previous paper.⁴)

It is instructive to take a few measurements to see what is what. Using the ping utility to send ICMP ECHO packets, I measured the round-trip latency between the University of Pennsylvania (klondike.cis.upenn.edu) and

Stanford University (cs.stanford.edu), two well-connected sites, as being about 87.5 ms. Rounding this to 88 ms and subtracting the fiber propagation time of 38 ms leaves a 50-ms difference. (Note that the New York–San Francisco numbers are assumed to be roughly equivalent to those from Philadelphia to Palo Alto.) Since these data units are only about 500 bits long, bandwidth between Penn and Stanford would have to be pretty bad (500 bits in 50 ms would be about 10 Kbps!) to explain the delay. So what could be the cause?

There are at least a couple of possible factors, both of which can be explained with the notion of *hops*. To understand hops, it helps to understand how a network differs from our 8,250-km loop of fiber. A real network is constructed of many interconnected pieces—for example, local area networks and wide area networks. Figure 1 represents a real physical network topology, with many types of networks and multiple devices. Hosts are labeled with H, routers with R, and network types are shown to be multiple in nature.

Many different packet formats and data units are in use, and the genius of the Internet is that it has a solution to make them all work together. This *interoperability layer* consists of a packet format and an address that are interpreted by devices called IP routers. The subnets interconnecting the routers can use whatever technology they choose as long as they can carry encapsulated IP packets between routers. Each router-router path is called a *hop*. As before with ping, it is instructive to obtain a measurement, which I obtained using traceroute between the two hosts I used before. There are 17 hops reported. Our analysis of an unobstructed fiber did not account for these routers, nor for the possibility that packets did not travel “as the crow flies” between the source and destination. The total propagation delay through this network, then, is equal to the sum of the propagation time across each subnet, plus the time required to pass through the routers.

Modern routers such as the Cisco CRS-1 exhibit average latencies of about 100 microseconds.² Our Philadelphia–Palo Alto example would include roughly 30 of them in the round-trip path, making the total router latency about 3 ms. The other causes of delay are more difficult to measure. Routers attempt to optimize a path between two points, but that may be difficult, so in addition to the delay through the routers we can expect a certain delay caused by path selections that deviate from a straight line. Other possible sources of delay include slower routers and other intervening appliances (such as firewalls), queuing (to wait for access to a busy shared

Fighting Physics: A Tough Battle

link), poor route selection, and slow links. Nonetheless, it is impressive that the IP routing infrastructure is only about a factor of two “slower” than the speed of light in fiber: 88 ms versus 38 ms.

This observation of the difference between pencil and paper and measured results leads to the definition of the *throughput* of a system, which is how many bits per second you can send after taking into account all the real-world limitations—propagation delays, bandwidth, latency, and hops.

INTERPROCESS COMMUNICATION AND PROTOCOLS

In a distributed system, processes that need to communicate do so via one or more schemes for IPC.³ Example schemes include messages, reliable streams, and remote procedure calls. It is easiest to think of IPC in terms of messages, sometimes called ADUs (application data units), as they are the building blocks on which other IPC mechanisms, including reliable bytestreams, are built. Messages may require multiple IP packets. The socket API is one example of a way in which message and reliable bytestream services can be accessed. It resembles input/output, supporting a read/write style of interface. The impact of the IPC software on a single message’s latency is typically low; ping measurements of a local loopback interface on `klondike.cis.upenn.edu` show times of about

20 microseconds of latency. The largest cause of propagation delays in IPC is protocols.

Protocols are rules for communicating intended to provide desired properties, such as high application throughput, reliability, or ordering. Reliable message delivery is a common application requirement and usually requires confirmation from the receiver to the sender, thus implying a round-trip. Communications requiring more data than a single packet must use multiple packets, implying multiple round-trip times. To see the impact of the physics on a naïve protocol, imagine an IPC system that uses 10-Kbit packets and must move 100 Kbits (10 packets’ worth of data) across the U.S., which as we have seen (for a single transcontinental piece of fiber) should require about 19 ms. If a new packet is sent only when a previous one has been acknowledged, one packet will be sent every 38 ms, and the communication will require 380 ms, or almost one half second, independent of the bandwidth of the network. Yet, it’s clear that with a high-throughput network, one could have sent all 10 of the packets in a row and waited for a confirmation that all 10 arrived, and this could be done in 38 ms.

This example along with figure 2 illustrates what is often called the *bandwidth-delay product*, which is a measure of the capacity of a path in bits between a source and a destination. Figure 2 shows that there may be usable capacity that is not being used, illustrated here by the spaces between packets. If the network were fully utilized, then all of the capacity would be fully occupied by packets in flight. When the network is fully occupied with packets, a bandwidth-delay product of bits will be in flight between a source and destination. The challenge is estimating the available capacity at any given time, as network dynamics could make this estimate highly variable. If we overestimate the capacity, too many packets will be pushed into the network, resulting in congestion. If we underestimate the capacity, too few packets will be in flight and performance will suffer.

Optimizing protocols to the available bandwidth-delay product has been a long-standing problem of interest to

the networking community, resulting in many algorithms for flow control and congestion control. TCP/IP, for example, uses acknowledgments from the receiver to pace the sender, opening and closing a window of unacknowledged packets that is a measure of the bandwidth-delay product. If a packet loss occurs, TCP/IP assumes it is congestion and closes the window. Otherwise, it continues trying to open the window to discover new bandwidth as it becomes available.

Figure 3 shows how TCP/IP attempts to discover the correct window size for a path through the network. The line indicates what is available, and significantly, this changes with time, as competing connections come and go, and capacities change with route changes. When new capacity becomes available, the protocol tries to discover it by pushing more packets into the network until losses indicate that too much capacity is used; in that case the protocol quickly reduces the window size to protect the network from overuse. Over time, the “sawtooth” reflected in this figure results as the algorithm attempts to learn the network capacity.

A major “physics” challenge for TCP/IP is that it is learning on a round-trip timescale and is thus affected by distance. Some new approaches based on periodic router estimates of available capacity are not subject to round-trip time variation and may be better in achieving high throughputs with high bandwidth-delay paths.

IMPLICATIONS FOR DISTRIBUTED SYSTEMS

Many modern distributed systems are built as if all network locations are roughly equivalent. As we have seen, even if there is connectivity, delay can affect

some applications and protocols more than others. In a request/response type of IPC, such as a remote procedure call, remote copies of data can greatly delay application execution, since the procedure call is blocked waiting on the response. Early Web applications were slow because the original HTTP opened a new TCP/IP connection for each fetched object, meaning that the new connection’s estimate of the bandwidth-delay was almost always an underestimate. Newer HTTPs exhibit persistent learning of bandwidth-delay estimates and perform much better.

The implication for distributed systems is that one size does not fit all. For example, use of a centralized data store will create large numbers of hosts that cannot possibly perform well if they are distant from the data store. In some cases, where replicas of data or services are viable, data can be cached and made local to applications. This, for example, is the logical role of a Web-caching system. In other cases, however, such as stock exchanges, the data is live and latency characteristics in such circumstances have significant financial implications, so caching is not effective for applications such as computerized trading. While in principle, distributed systems might be built that take this latency into account, in practice, it has proven easier to move the processing close to the market.

RULES OF THUMB TO HOLD YOUR OWN WITH PHYSICS

Here are a few suggestions that may help software developers adapt to the laws of physics.

Bandwidth helps latency, but not propagation delay.

If a distributed application can move fewer, larger messages, then this can help the application as the total cost in delay is reduced since fewer round-trip delays are introduced. The effects of bandwidth are quickly lost for large distances and small data objects. Noise can also be a big issue for increasingly more common wireless links, where shorter packets suffer a lower per-packet risk of bit errors. The lesson for the application software designer is to think carefully about a design’s assumptions about latency. Assume large latencies, make it work under those circumstances, and take advantage of lower latencies when they are available. For example, use a Web-embedded caching scheme to ensure the application is responsive when latencies are long, but no cache when it’s not necessary.

Spend available resources (such as throughput and storage capacity) to save precious ones, such as response time. This may be the most important of these rules. An example is the use of caches, including preemptive caching of data. In principle, caches can be replicated local to

Fighting Physics: A Tough Battle

applications, causing some cost in storage and throughput (to maintain the cache) to be incurred. In practice, this is almost always a good bet when replicas can be made, because growth in storage capacities and network throughputs appears to be increasing on a steady exponential. Prefilling the cache with data likely to be used means that some capacity will be wasted (what is fetched but not needed) but that the effects of some delays will be mitigated when predictions of what is needed are good.

Think relentlessly about the architecture of the distributed application. One key observation is that a distributed system can be distributed based on function. To return to the design of a system with a live data store (such as a stock market), we might place the program trading of stocks near the relevant exchanges, while placing the user interaction functionality, account management, compliance logging, etc. remotely in less exchange-local real estate. Part of such a functional decomposition exercise is identifying where latency makes a difference and where the delay must be addressed directly rather than via caching techniques.

Where possible, adapt to varying latencies. The example of protocols maximizing throughput by adapting to bandwidth-delay capacities shows how a wide range of latencies can be accommodated. For distributed applications, this might be accomplished by dynamically relocating elements of a system (e.g., via process migration or remote evaluation).

None of these suggestions will allow you to overcome physics, although prefetching in the best of circumstances might provide this illusion. With careful design, however, responsive distributed applications can be architected and implemented to operate over long distances.

SUMMARY

Propagation delay is an important physical limit. This measure is often given short shrift in system design as application architectures evolve, but it may have more performance impact on real distributed applications than

bandwidth, the most commonly used figure of merit for networks. Modern distributed applications require adherence to some rules of thumb to maintain their responsiveness over a wide range of propagation delays. **Q**

ACKNOWLEDGMENTS

Comments from the ACM *Queue* editorial board, particularly Eric Allman, and from Craig Partridge greatly improved this article.

REFERENCES

1. Mohr, P. J., Taylor, B. N. 2005. CODATA recommended values of the fundamental physical constants. *Reviews of Modern Physics* 77(1): 1-107.
2. 40-gig router test results. 2004. Light Reading; http://www.lightreading.com/document.asp?doc_id=63606&page_number=4&image_number=9.
3. Partridge, C. 1994. *Gigabit Networking*. Addison-Wesley Professional.
4. Shaffer, J. H., Smith, J. M. 1996. A new look at bandwidth latency tradeoffs. University of Pennsylvania, CIS TR MS-CIS-96-10; http://repository.upenn.edu/cgi/viewcontent.cgi?article=1192&context=cis_reports.

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

JONATHAN M. SMITH is the Olga and Alberico Pompa Professor of Engineering and Applied Science and a professor of computer and information science at the University of Pennsylvania. He served as a program manager at DARPA from 2004 to 2006 and was awarded the OSD (Office of the Secretary of Defense) Medal for Exceptional Public Service in 2006. He is an IEEE Fellow. His current research interests range from programmable network infrastructures and cognitive radios to disinformation theory and architectures for computer-augmented immune response.

© 2009 ACM 1542-7730/09/0200 \$5.00

Association for
Computing Machinery

Advancing Computing as a Science & Profession

You've come a long way. Share what you've learned.

ACM has partnered with MentorNet, the award-winning nonprofit e-mentoring network in engineering, science and mathematics. MentorNet's award-winning **One-on-One Mentoring Programs** pair ACM student members with mentors from industry, government, higher education, and other sectors.

- Communicate by email about career goals, course work, and many other topics.
- Spend just **20 minutes a week** - and make a huge difference in a student's life.
- Take part in a lively online community of professionals and students all over the world.

Make a difference to a student in your field.

Sign up today at: www.mentornet.net

Find out more at: www.acm.org/mentornet

MentorNet's sponsors include 3M Foundation, ACM, Alcoa Foundation, Agilent Technologies, Amylin Pharmaceuticals, Bechtel Group Foundation, Cisco Systems, Hewlett-Packard Company, IBM Corporation, Intel Foundation, Lockheed Martin Space Systems, National Science Foundation, Naval Research Laboratory, NVIDIA, Sandia National Laboratories, Schlumberger, S.D. Bechtel, Jr. Foundation, Texas Instruments, and The Henry Luce Foundation.