Realtime publishers

The Definitive Guide to Cloud Acceleration

Dan Sullivan

sponsored by

CDNetworks

Copyright Statement

Chapter 3: Why the Internet Can Be the Root Cause of Bottlenecks

Developers, software designers, and architects can spend a significant amount of time and effort tuning their applications and infrastructure and their networks can still suffer from performance problems. This reality does not necessarily mean these IT professionals are bad at what they do; it can mean that the problem lies outside their area of control. One of the biggest potential bottlenecks software developers have to contend with is the Internet itself.

The term *Internet* is an accurate description of the resource we have come to rely on. It is literally an interconnected network of networks. There is no single, monolithic network encompassing the globe. There is no single organizational entity that monitors, maintains, and expands the Internet. (Although there are bodies that set technical standards and administrative organizations that manage functions such as domain naming, these bodies do not control the infrastructure that constitutes the Internet).

Two Fundamental Characteristics of the Internet that Influence Performance

Two fundamental characteristics of the Internet are central to understanding how the Internet can be a bottleneck to Web application performance. The networks that constitute the Internet are interoperable. That is, network traffic on one network can move across another network without changing or translating data from one representation to another. The reason is that networked devices on the Internet run a shared set of protocols.

Technical Characteristic: Protocols

Consider how a customer working from her office in Singapore can work with an application running on a cluster of servers in a data center in the eastern United States. The customer uses a browser to navigate to a Web application and creates transactions using that application. The data that is entered, the clicks used to navigate the application, and the Web pages that are generated in response are all organized, packaged, and transmitted halfway around the globe using the same standard protocols.

The data that needs to be sent from the customer in Singapore to the data center in the United States is packaged into Transmission Control Protocol (TCP) packets. TCP is one of many Internet protocols, but it is one of the most important from an application perspective. TCP guarantees delivery of data packets in the same order in which they were sent. The importance of this feature is obvious.

Imagine if the data entered for a transaction sent halfway across the globe was broken into pieces, transmitted, and arrived at the destination in the wrong order. When the packets were reassembled at the destination, you might find that the order quantity was swapped with the customer's shipping address. TCP prevents this kind of problem by implementing controls that ensure packets are properly ordered when they are reassembled at their destination.

Other protocols are required as well to get data from one network device to another. Data that is transmitted over long distances can move across multiple paths. For example, data arriving from the west coast of the United States could be sent to an east cost data center over paths across the northern or southern United States or could be sent across a less straightforward route (at least from a geographical perspective). Routing protocols are used to determine how to send packets of data from one point in the Internet to the next.

Organizational Characteristic: Peering

Theoretically, routing protocols will transmit packets along the most efficient path between two points. Here is one of those situations in which theory and practice diverge. The most efficient route, in terms of the time required to transmit a packet or the distance between points on the network, may not be the route that is actually taken. The reason is that the most efficient route may follow paths that use networks from several network providers.

Some network providers have agreements to carry the traffic of the other. This arrangement, known as peering, is the second important factor that can influence Web application performance. Assume for a moment that you need to route a packet from a device on Network A to a device on Network D. The most efficient path from the source to the destination device is over network B; however, Network A and Network B do not have a peering agreement. Network A does have a peering agreement with Network C, which also links to Network D, so the packet from Network A is routed to Network D over Network C (see Figure 3.1).

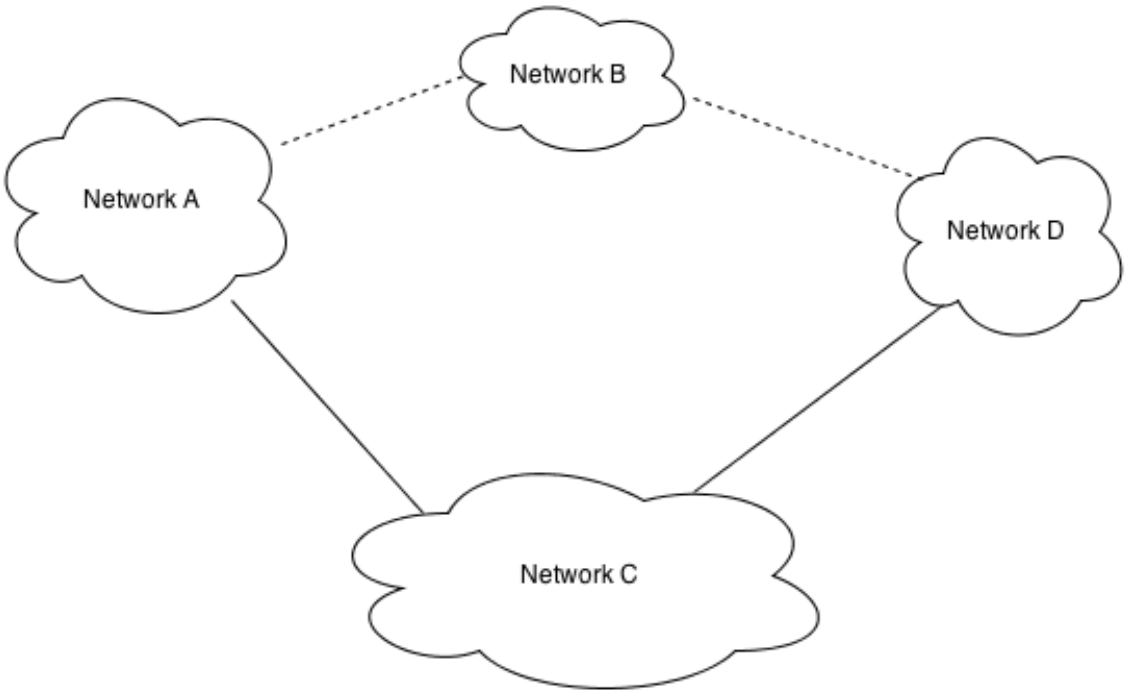

Figure 3.1: The most efficient route, from a technical perspective, may not be taken. Organizational and business relationships between network providers influence the path taken between source and destination devices.

Peering is a business arrangement. Network providers obviously want to provide their customers with access to any Internet-addressable device. They would not be much of an Internet service provider (ISP) if they could not do that. At the same time, they cannot deploy their own network infrastructure across the globe in order to access every Internet-addressable device. The only practical method to ensure complete access to all devices on the Internet is to make arrangements with other network providers.

In some cases, a peering relationship can be mutually beneficial. For example, if a country had two major network providers and each provided services to 50% of Internet users in the country, both would benefit from the ability to route traffic to the other provider's network. (Assuming, of course, the characteristics of traffic on both networks is roughly equal). Such is not always the case.

Some network providers have created large advanced networks with substantial geographic reach. These network providers can make arrangements with each other to allow the transit of each other's traffic free of charge. Sometimes the cross-network routing arrangement is not equally beneficial. For example, a small network provider may have a large portion of its traffic sent outside the network while a large provider has only a small amount of data routed to smaller providers. This kind of asymmetry undermines the logic for free exchange of traffic.

When there is significant asymmetry between providers, the smaller provider will likely have to pay some kind of fee, known as a transit charge, to the larger provider. Obviously, one of the ways these smaller providers control cost is by limiting as much as possible the amount they have to pay in transit fees. Let's go back to the example with network providers A, B, C and D.

The most efficient route from a device on Network A to a device on Network D is through Network B. Network B charges 50% more in transit charges than Network C. Network A is a midsize network provider and cannot secure peering agreements without paying fees to some of the larger network providers. From a technical perspective, Network A could route its traffic to Network D over Network B; from a business perspective, that option is cost prohibitive. As a result, a customer on Network A sending and receiving traffic from Network D depend on sub-optimal routing.

Network Provider Terminology

Network providers come in a wide range of sizes and implement a variety of business relationships with each other. One way to coarsely group network providers is according to a commonly-used three-tier scheme:

- **Tier 1 providers**: Network providers that can reach all other networks on the Internet without paying transit charges are known as Tier 1 providers.
- **Tier 2 providers:** Tier 2 providers have some fee-free peering agreements but also pay transit fees to other network providers.
- **Tier 3 providers**: Tier 3 providers pay transit fees to other network providers in order to route traffic to all Internet-addressable devices.

These definitions are not hard and fast. Users do not always know the business arrangements between network providers. Fee-free agreements may include other conditions that an accountant might consider a form of transfer of revenue, but this categorization scheme ignores those distinctions. This scheme is simply a rough way to divide Internet providers into large, medium, and small.

Protocols and peering agreements are two fundamental characteristics of the Internet that influence Web application performance. The rest of this chapter will delve into more detail about both protocols and peering agreements to understand how they affect application performance and what you can do to improve it. Before moving on to the details of protocols and peering, let's establish a few measurements that are important for quantifying network performance.

Measuring Performance

If your customers are complaining about poor performance or abandoning shopping carts at high rates, application performance might be a problem. Many factors influence the overall speed at which your application processes transactions:

- Algorithms used to process data
- Implementation of algorithms in code
- Database design and query tuning
- Application and server configuration
- Network throughput
- Latency
- Packet loss

Software developers and database administrators can address the first four potential issues in an application test environment. It is a fairly straightforward process to test parts of large systems when you have access to source code and documentation. You can, for example, time various modules to help identify which modules are taking the most time to execute. Solving the problem once it is identified is not always trivial but at least you have a sense of where the problem lies.

The last three issues in the list are related to network performance. Diagnosing problems on the network is somewhat different than debugging and analyzing code. Your goal is not to use your favorite integrated development environment to analyze code implementing TCP on your server but instead to understand why network operations take as long as they do.

Like software development, analyzing network problems is partly a matter of dividing and conquering. The goal is to find the bottleneck (or bottlenecks) on the network between your servers and client devices. Is the problem at a router in your data center? Is your ISP meeting agreed-upon service levels with regard to your network traffic? Does the problem lie with networking infrastructure closer to the client devices? To find out where network problems occur, it helps to have standard measures that allow you to quantify network performance.

Customers or other end users might describe problems with application performance in qualitative terms such as “slow” and “takes too long.” Sometimes they might even experience errors and can provide qualitative error messages, such as “the connection timed out.” These kinds of descriptions are symptoms of an underlying problem but they lack specificity and precision.

This discussion will focus on three quantitative measures when considering network performance:

- Throughput
- Latency
- Packet loss

These measures describe characteristics of networks that can influence the overall performance of your applications.

Throughput

Throughput is a measure of data packets that are transmitted over a network. It is usually described in bits per second. We often describe network throughput in terms of a single quantity, such a 100Mbits per second. In practice, network throughput can vary over time with changes in network conditions.

A number of factors can influence throughput. Physical characteristics such as the noise on the line can interfere with signals and reduce throughput. The speed at which network equipment can process data packets can also impact throughput. The Internet Protocol (IP) packets, for example, are comprised of header data and payloads. The network devices use data in the header to determine how to process a packet. Throughput of the network will depend, to some degree, on how fast network devices can process this header data.

We can draw an analogy between network throughput and traffic on a highway. A single lane highway may allow only one car at a time at any point on the highway, a two lane highway can have two cars at a time, and so one. By adding more lanes, you can increase the number of cars that can travel over the highway.

When cars travel at the top speed on the network and there are cars in every lane, you can reach peak throughput of the highway. Similarly, with networks, you can reach peak throughput when you fully utilize the network's carrying capacity. Peak throughput cannot always be maintained, though. In the highway analogy, a car accident or slow moving car can reduce the speed of other drivers. In the case of the network, signal degradation or delays processing data packets can reduce throughput below peak throughput. As this discussion is primarily concerned with Web application performance, it is helpful to work with average or sustained throughput over a period of time rather then peak throughput.

Latency

In addition to throughput, which measures the amount of data you can transmit over a particular period of time, you should consider the time required to send a packet from a source system to a destination system. Latency is a measure of the amount of time required to send a packet to another device and back to the sending device. Latency can also be measured in terms of a one-way trip from a source to the destination, but, for this discussion, latency will refer to round-trip latency unless otherwise noted.

Returning to the highway driving analogy, you can think of latency as the time required to make a round-trip from one point on the highway to another point and then return to the starting point. Obviously, the speed at which the car moves will determine the time required for the round-trip, but other factors can influence latency. If there is congestion on the network, as with highways, latency can increase.

If a car needs to leave fast-moving traffic on a highway to transit a road with slower maximum speeds, the latency will increase over what would have been the latency had the car stayed on the highway. This situation is analogous to a data packet that is transmitted over a high-speed network but then routed over a slower network en route to its destination device.

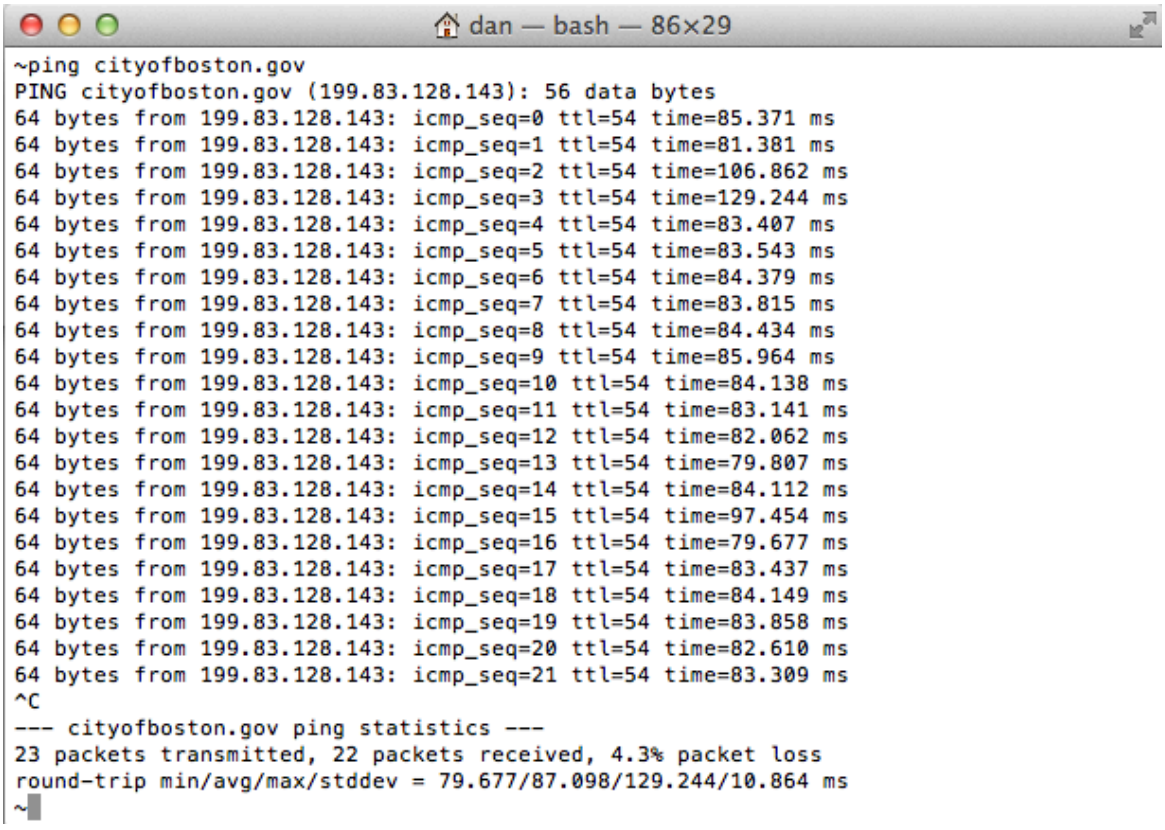

Figure 3.2: Ping can be used to measure latency between two devices. This example highlights the latency of packets sent from the west coast of the United States (Oregon) to a server on the east coast (Delaware), averaging 87.1ms and ranging from 79.68ms to 129.24ms.

Just to clarify, latency is the time needed to send a packet of data on a round-trip between two networked devices. This idea should not be confused with the time required to complete a transaction on a Web application. Network latency is one factor in the time required to complete a Web transaction; the time required to complete a transaction also includes the time needed by database servers, applications servers, and other components in the application stack. In addition to throughput and latency, you should also consider the rate of packet loss on a network.

Packet Loss

Packet loss occurs when a device sends a packet that is never received by the target device. This loss is easily detected by TCP. Once a TCP connection is established between two devices, the devices allocate memory as a buffer to receive and hold data packets. The buffer is needed because packets may arrive out of order. For example, packet 1 may be routed over a network that suddenly experiences a spike in congestion while packet 2 is routed over another network. When packet 2 arrives at the destination, it is held in the buffer until packet 1 arrives. (There are other conditions that determine how the buffer is used and when it is cleared, but those conditions can be safely ignored in this illustration.) If packet 1 does not arrive within a reasonable period of time, the packet is considered lost.

Lost packets have to be retransmitted, which consumes additional network bandwidth. In addition to transmitting the packet twice, there is additional traffic between the two devices to coordinate the retransmission. The receiving device also waits the period of time specified in the TCP configuration before determining a packet has been lost. All of these factors combine to increase the time needed to send a message from the source to the destination.

Packet loss is one of the factors that decreases throughput. Packet loss can be caused by signal degradation if the single must travel long distances or is subject to physical interference. Congestion on a network device can also cause packet loss. Routers, for example, have only so much memory to buffer data arriving at the router. When the traffic arriving at the device exceeds the capacity of the device to buffer and process the data, packets may be lost.

In some cases, packet loss does not cause retransmission of packets. Some applications can make use of the User Datagram Protocol (UDP), which unlike TCP, does not guarantee delivery or delivery in order. This setup obviously will not meet the needs of transaction processing systems, but there are use cases where UDP is appropriate.

When transmitted data has a limited useful life and is replaced with newly generated data, then UDP is appropriate. For example, if you were to synchronize clocks on multiple servers by sending a timestamp from a time server, you could lose a single timestamp without adversely affecting the other systems. A new datagram with a new timestamp would be generated soon after the first datagram was lost. Transmitting video or audio over UDP makes sense as well. If many packets are lost, the quality of the image or sound may degrade but the benefits of lower overhead compared with TCP make UDP a viable alternative.

From this discussion, you can see that there are at least three measures to consider when analyzing the impact of the network on Web applications. Throughput is a measure of the number of packets, and therefore the number of successfully delivered bits, sent between devices. Latency is a measure of time required for a round-trip transmission between devices, while packet loss is a measure of the amount of data that is lost on a line. Applications that use TCP (most transactional business applications) not only have to retransmit lost packets but also incur additional overhead to coordinate the retransmission. Next, let's consider how protocols, especially Hypertext Transfer Protocol (HTTP) and TCP, can contribute to network bottleneck issues.

Protocol Issues

The Internet utilizes a substantial number of protocols to implement the many services we all use. Some of these operate largely behind the scenes without demanding much attention from software developers who create Web applications. The Border Gateway Protocol (BGP), for instance, is needed to route traffic between independent domains and to exchange information about network routes. The Domain Name Service (DNS) is probably more widely recognized as the service that maps from human readable domain names (for example, www.example.com) to IP addresses. As you consider Web application performance issues, you should consider how these services can adversely impact performance. DNS, for example, has been a significant contributor to adverse performance. However, it does help to understand some of the implementation details around HTTP and TCP. The way you use HTTP and configure TCP can have noticeable impact on Web application performance.

Web Application Performance and HTTP

HTTP was created about 20 years ago as a means of sharing documents at the physics research center CERN. From its earliest uses, HTTP has been used to create feature-rich documents, and later, applications. The first browser supported both text and images (see Figure 3.3).

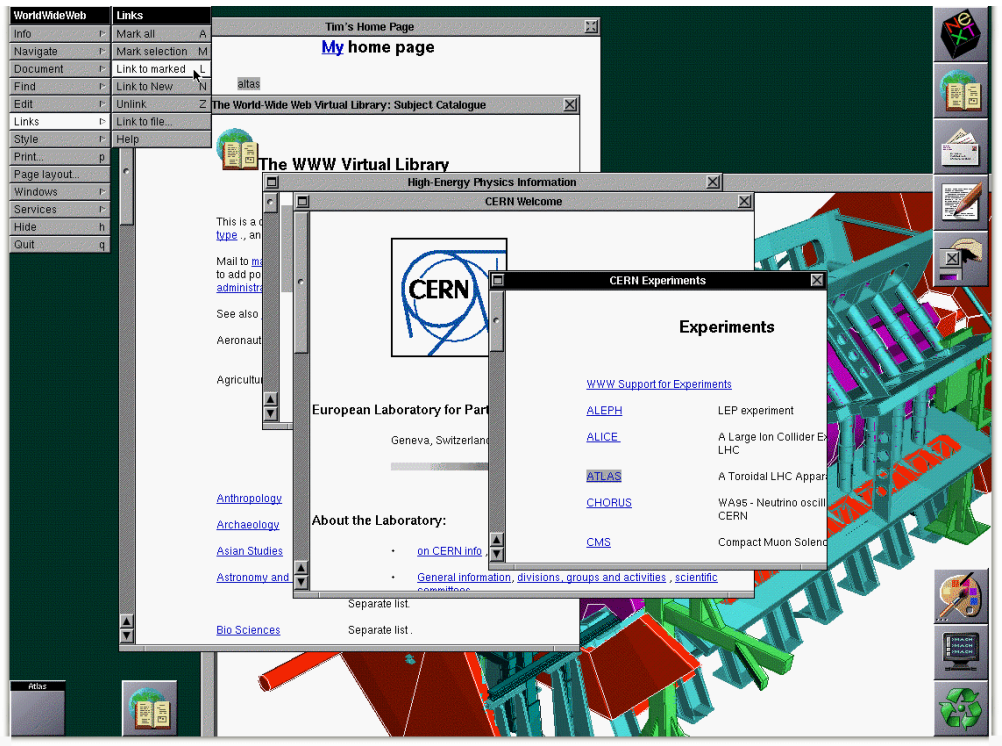

Figure 3.3: The original Web browser running on the NeXT computer at CERN, the birthplace of the Web (Source: http://info.cern.ch/).

In many ways, the fundamentals of transmitting Web content has not changed much from the early days of HTTP adoption. The way we use HTTP and generate content has certainly changed with new development technologies, advances in HTML, and the ability to deploy functional applications in a browser. The fact that we can use the same protocol (with some modifications over the years) to implement today's applications attests to the utility of HTTP. There are, unfortunately, limitations of HTTP that can still create performance problems for developers and Web designers.

Multiple Objects Per Web Page

Navigating the Web is a simple matter and masks much of the complexity behind rendering a typical Web page. A quick survey of several major retailer Web home pages found pages with 40 to more than 100 images rendered on each page. Each of these images must be downloaded over a TCP connection between the Web server and the client device. In addition, Web pages often provide search services or other application-like features on their Web sites. Some of these are services provided by third parties that require connections to those third parties' servers.

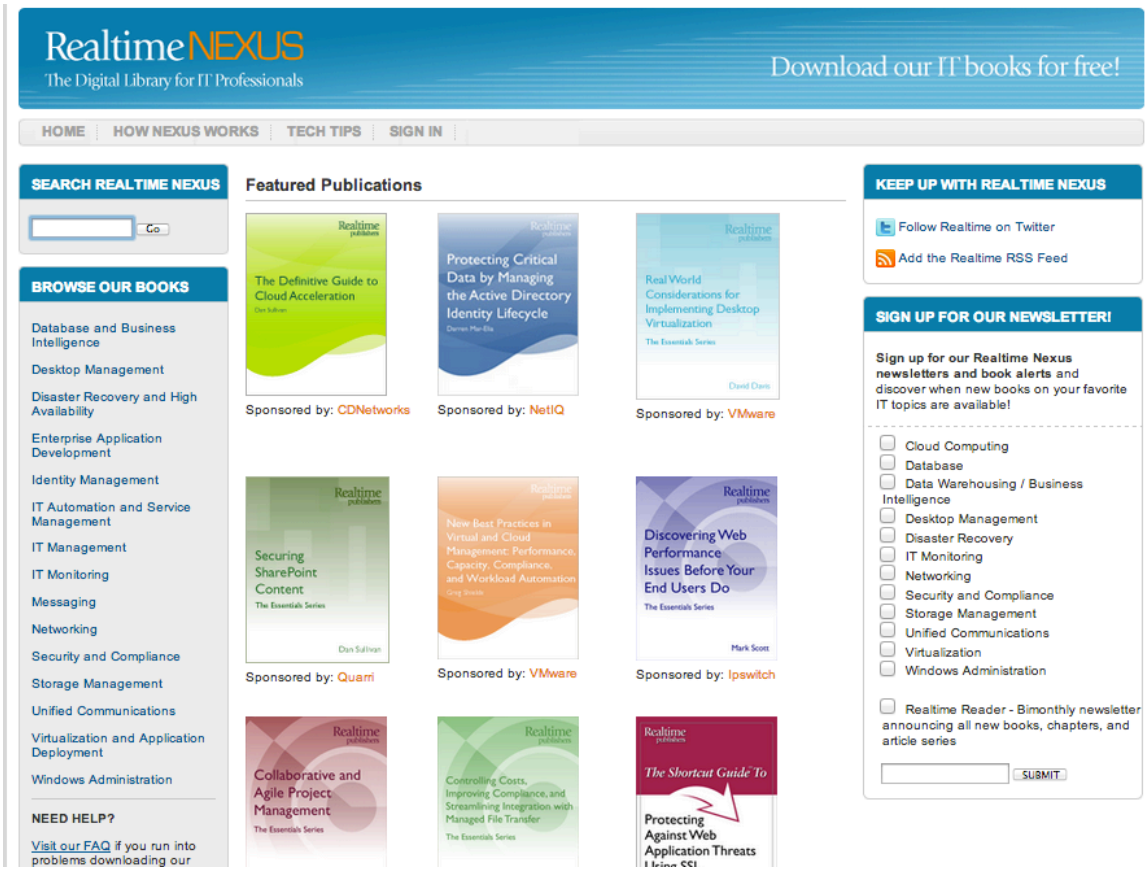

Figure 3.4: Web pages and applications include multiple components, ranging from text and images to application services. Each requires a connection to download the component from its server to the client device.

The fact that multiple connections are needed to download all the components of a Web page helps to draw attention to the process of establishing a connection and the overhead that entails.

TCP and Implications for Web Application Performance

As noted earlier, TCP has many features that support the development of Web applications and documents. Guaranteed delivery of packets in the order they were sent is a basic but essential building block for the Web. Ensuring this level of delivery comes with a cost, though. There is some amount of overhead associated with TCP. The overhead starts when a new connection is established.

TCP Handshake

Before devices can exchange packets using TCP, the two devices must first establish a connection. This arrangement is essentially an initial agreement to exchange data and an acknowledgement that both devices are reachable. This exchange is known as the TCP handshake.

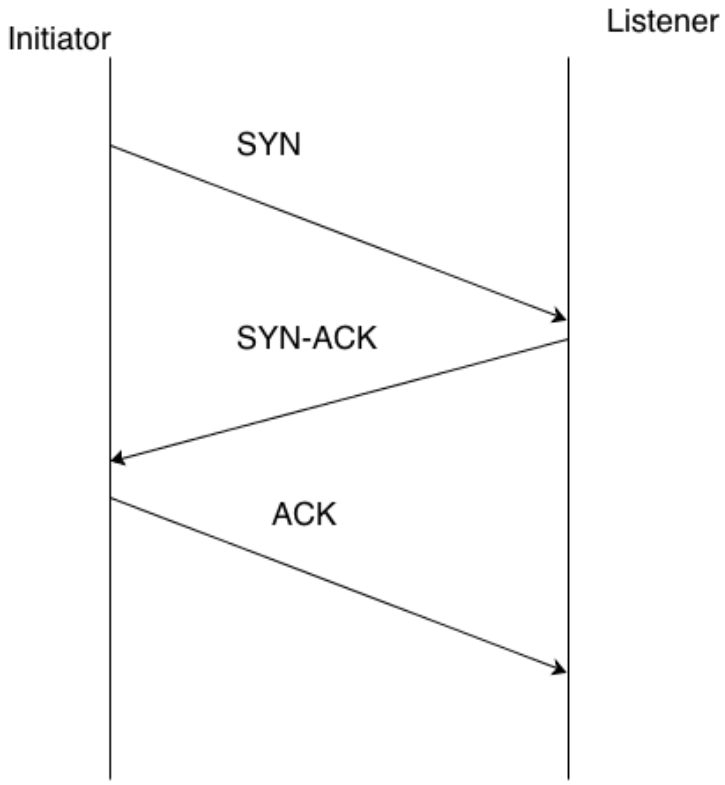

Figure 3.5: The TCP handshake protocol requires an initial exchange of synchronizing information before data is transmitted.

The TCP handshake is a three-step process that requires the exchange of packets to acknowledge different phases of the process known as:

- SYN
- SYN-ACK
- ACK

Each of these phases must occur before the next phase in the list can occur; if all phases complete successfully, a transfer of data can begin.

The TCP handshake begins with one device initiating a connection to another device. For example, when a customer uses a browser to download a Web page from a business' Web site, a connection is created to download the initial HTML for the page. (There will probably be other connections as well but this discussion will describe this process for a single connection.)

The first step of the process entails the initiating device sending a SYN packet to the listening device; for example, a laptop sending a SYN message to a Web server to download a Web page. Part of the SYN packet contains a random number representing a sequence number, which is needed in later steps of the process. After the SYN message is sent, the initiator is said to be in SYN-SENT state and it is waiting for a reply. Prior to receiving the SYN packet, the listening device is said to be in the LISTEN state.

Once the listening device—for example, the server—receives the SYN packet, it responds by replying with a SYN-ACK packet. This packet includes the sequence number sent in the SYN packet incremented by 1. The listening device generates another random number that is the second sequence number, which is included in the SYN-ACK packet. The listening device is then in the SYN-RECEIVED state.

The third and final set occurs when the initiating device responds to the SYN-ACK packet with a final ACK packet. This packet includes the first sequence number incremented by 1 as well as the second sequence number incremented by 1. After this point, both the initiator and the listening devices are in the ESTABLISHED state and data transfer can begin.

The connection can persist as long as needed. When the connection is no longer needed, a 3-step termination process can close down the connection. The termination process uses a similar 3-step process with FIN, FIN-ACK, and ACK packet exchanges.

Reliable and Ordered Transfer of Data

Once a connection is established with the TCP handshake, data exchange begins. As with the SYN, SYN-ACK, and ACK messages, a sequence number is included with data packets. Sequence numbers are used to ensure that data is reconstituted in the same order it was sent.

This inclusion is particularly important when the packets may be routed over different paths and could arrive at the destination in a different order from which they are sent. For example, if someone is loading a large file to a cloud storage service thousands of miles away, the data may take paths over two or more network service providers. Sequence numbers are used to determine the order in which packets were sent and, in some cases, determine whether packets are lost.

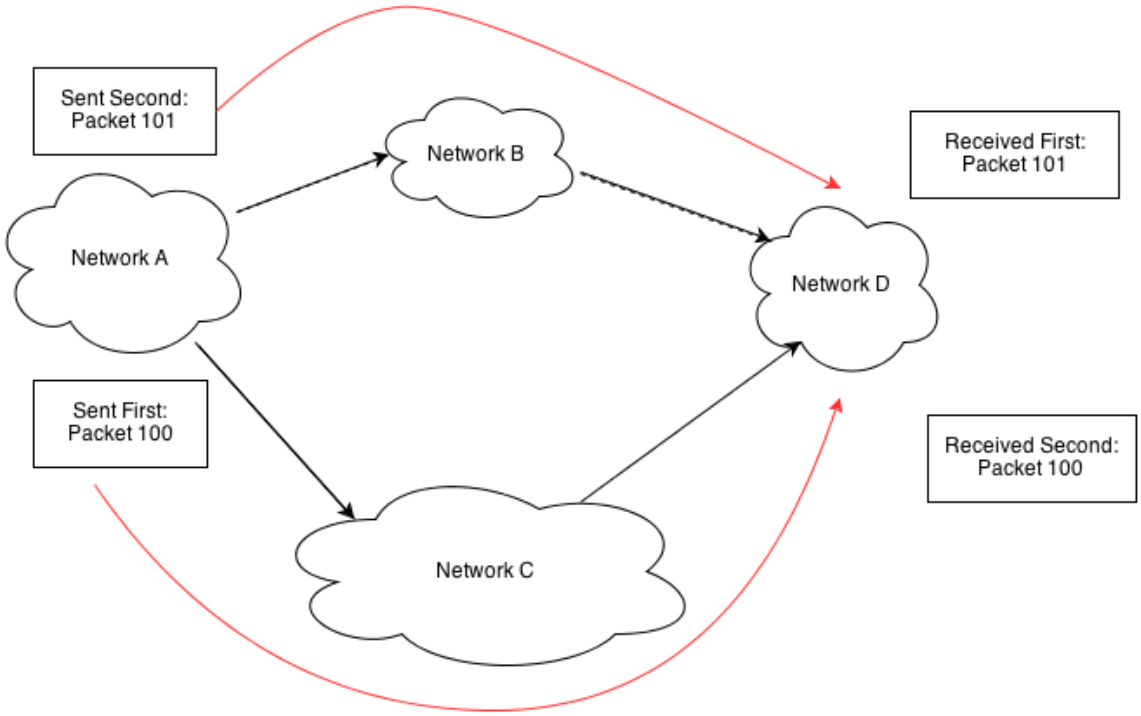

Figure 3.6: TCP packets transmitted in the same connection may use different paths to reach the destination device. Sequence numbers in each packet are used to reconstruct the data stream in the same order it was sent.

Consider a situation in which a device receives a series of packets with numbers 100, 101, 102, and 104. Packet 103 has not been received. This situation might simply be a case of the situation depicted in Figure 3.6. Some of the packets used a path that allowed for faster delivery than did packet 103. In that case, it might be just a matter of a small amount of time before packet 103 arrives.

When all the packets in a range of packets are received, the receiving device transmits an acknowledgment to the sender. Senders can use these acknowledgments to determine whether packets are lost.

TCP requires that you decide how long you are willing to wait for a packet to arrive. When packets are traveling over high-latency networks, such as satellite networks, you might want to have long waiting periods. There would be no advantage to asking the sending device to retransmit a packet that has not had sufficient time to reach the destination device. In fact, that could create additional, unnecessary traffic that could consume bandwidth and increase congestion on the network.

At the same time, you do not want to wait too long for packets to arrive. If packets are dropped, they will never arrive. For example, a packet may have been dropped at a router that was experiencing congestion and unable to process all the traffic arriving at the router. In other cases, the transmission signal may have degraded to the point that the packet was corrupted.

When packets are actually dropped, it is better to infer the need to retransmit sooner rather than later. The destination device will buffer incoming data waiting for missing packets to arrive. Retransmitting requires sending messages from the destination to the sender and then from the sender to the destination device, so the wait for the retransmitted packet will be the time of a round-trip between the two devices.

This discussion has delved into design and configuration details of TCP, but it is important to remember this discussion is motivated by Web application performance. TCP design issues are important and interesting, but unless you are a network designer or computer scientist studying protocols, the primary reason for reviewing TCP is to help improve the performance of your Web applications.

Let's summarize how Web application performance is related to TCP. Here is a set of events triggered by a Web application user requesting a Web page:

- The Web application generates Web pages that are downloaded to client devices.
- Web pages are composed of multiple elements, including layout code, scripts, images, audio files, etc. Each component that is downloaded separately requires a TCP connection.
- TCP connections are initiated with a three-step TCP handshake before data transmission can begin.
- Once data transmission begins, the connection remains open until a termination process is complete.
- While the connection between two devices is open, TCP packets are transmitted between the devices.
- TCP buffers incoming packets as needed to ensure a data stream is reconstructed in the order in which it was sent
- Listening devices wait for packets to arrive and then send acknowledgments to the sending device.
- If the sending device does not receive an acknowledgement for the delivery of packets within a predefined time period, the missing packets are retransmitted.

You can see from this list of events that a single Web page can lead to the need for multiple connections, that multiple connections have to manage multiple packets, and that packets may be routed over slow, noisy, or congested networks. Packets routed over those poor quality networks can be lost requiring retransmission. Therefore, to improve the performance of your Web application, you might need to improve the performance of TCP transmissions.

Improving Protocol Performance

There are a number of ways to improve or avoid common network performance problems. This discussion will focus on three:

- Application and data replication
- Maintaining a pool of connections
- Optimizing TCP traffic

These are complementary strategies that can all be applied to the same Web application to help improve performance.

Application and Data Replication

Physics has the final say in how fast data can travel over a network. No matter what you do to improve signal quality, correct for errors, and avoid network congestion, the distance between devices will limit data transmission, which in turn can limit application performance. You cannot tune or engineer your way out of this problem, but you can avoid it (or at least minimize it).

One way to reduce the distance between your Web applications and your users is to replicate your data and your applications over geographically distributed data centers. Application users would be routed to the closest replication site when they accessed the application or data. This approach not only reduces latency inherent in long distance transmission but also can help avoid congestion on distant networks and limit exposure to the disadvantages of peering relationships between ISPs (more on that in a minute).

Consider the latency of an application hosted in North America and serving a global customer base. Within North America, customers might experience around a 40ms round-trip time. The same type of user in Europe might experience a 75ms round-trip time, while customers in East Asia might experience a 120ms round-trip time. If that same application were replicated in Europe and Asia, customers in those regions could expect to see round-trip latency to drop significantly.

Just hosting an application in different regions will not necessarily improve application performance. Geographically close locations can still contend with high latencies and packet loss rates if the quality of network between the locations is poor.

Maintaining Pools of Connections

In addition to working around the limits of the network, you can tune devices to improve performance. You have seen that a single Web page can require multiple connections, for example, to download multiple image files. The process of creating a connection takes some amount of time and when a large number of connections are used between a client and a server, that time can add up to a substantial amount. One way to reduce that time is to maintain a pool of connections. Rather than destroy a connection when it is no longer needed, you can maintain the data structures associated with the connection and reuse them when another component needs to be transferred. This kind of recycling of connections reduces the overhead of creating and destroying connections and therefore reduces the time that passes between the initiation of a request for a connection and the time data is transmitted between the devices.

Optimizing TCP Traffic

TCP implements a well-defined set of operations and specifies particular sequences of events for those operations, but there is still room for tuning the protocol. TCP depends on a number of parameters and optional features that can be used to improve throughput and overall performance of Web applications.

There are several characteristics of TCP connections that can be configured to improve performance including:

- Adjusting the size of buffers receiving data—A 64K buffer may be sufficient for some applications but long round-trip times may warrant larger buffers.
- Adjusting parameters that determine when to retransmit a packet—TCP has a parameter that determines how much data a destination device will receive before it needs to acknowledge those packets. If this parameter is set too low, you might find that you are not using all the bandwidth available to you because the TCP clients are waiting for an acknowledgement before sending more data.
- Using selective acknowledgements—TCP was originally designed to use a cumulative packet loss acknowledgement scheme, which is not efficient on networks with high packet loss rates. It can limit a sender to receiving information about a single packet in the span of one round-trip time; in some cases, the sender may retransmit more than necessary rather than wait for a series of round-trip time periods to determine whether all packets were received. An alternative method, known as selective acknowledgments, allows a receiving device to acknowledge all received packets so that the sender can determine which packets need to be resent.

The TCP stack provided with some operating systems (OSs) might implement some of these optimizations. Specialized tools are available for transmitting large files over specially configured connection to improve performance. Network service providers may also implement TCP optimizations between their data centers to improve performance.

Web application performance is influenced by the HTTP and TCP protocols. The way components of a Web page are downloaded, the overhead of establishing connections, and the way packets are reliably delivered can all impact Web application performance. Developers, designers, and system architects do have options though. Application and data replication can overcome the inherent limitations of long distance transmissions. Connection pooling and other device-specific optimizations can help reduce overhead at the endpoints. Even TCP can be optimized to improve network performance.

Now that we have covered some of the lower level details about how the Internet can impact Web application performance, it is time to turn our attention to higher-level constructs: network providers.

Peering: Linking Networks

We noted earlier that the Internet is a collection of networks created and maintained by network providers around the globe. Some network providers have access to all other networks without incurring charges (Tier 1 providers); some have access without charges to some but not other networks (Tier 2 providers); and others have to pay to access other networks (Tier 3 providers). The relationship between providers that you might have never heard of can have an impact on your Web application performance.

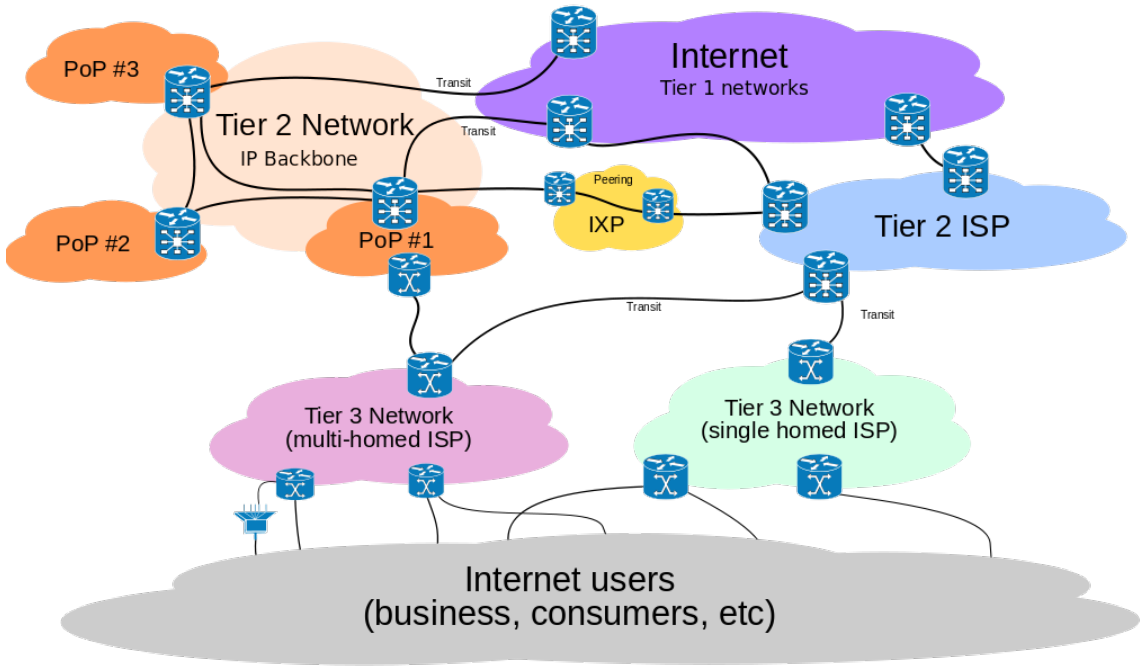

Figure 3.7: Peering creates a set of relationships between ISPs. These relationships determine at a high level how data is transmitted across the Internet (Source: By User:Ludovic.ferre (Internet Connectivity Distribution&Core.svg) [CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons).

Internet Exchange Points

As the number of network providers increases, direct connections between providers becomes problematic. If there were only two Internet providers, a single connection would link both. If there were four network providers directly connected to each other, then you would need six connections between them. The number of direct connections grows rapidly: 10 networks would require 45 direct connections to be fully interconnect and 100 networks would require 4950. (The formula to calculate this is [n*(n-1)]/2 where n is the number of network providers).

Instead of implementing point-to-point connections through third-party networks, the Internet was developed with exchange points where multiple ISPs can connect to each other at a single location. These points are known as Internet exchange points (IXPs). The IXP implements the physical infrastructure required to connect networks, and BGP is used to logically manage the exchange of data across networks. Network providers with peering agreements will accept traffic from their peering partners but not from other providers, and they enforce that policy with BGP.

Performance Issues Related to Peering

Peering is perhaps the most pragmatic, efficient way to implement the Internet, but even with that recognition, there are limitations to this scheme. These limitations can adversely affect your Web application performance.

Potential for Sub-Optimal Routing

Routing between network providers is governed by agreements between those providers. The optimal route between two points may be across a path that includes ISPs that do not have peering agreements. (This discussion includes agreements to pay transit fees for access to a network; this may not be part of the strict definition of peering but we'll continue with this slightly relaxed definition.)

If your application is hosted on servers in one continent and you are servicing customers around the globe, you will almost certainly be transmitting your data across multiple providers. Is the route that data takes the most optimal route? In some cases, it probably is and in some cases, it might not be. If your data or applications are replicated in multiple sites with a hosting provider with appropriate peering agreements, you might avoid some of the performance issues stemming from peering arrangements.

For application designers and network engineers, peering agreements are only slightly more forgiving than the laws of physics. They dictate the limits of what is possible with regards to routing on a global scale.

Congestion

Traffic congestion is a risk in any network. When a large number of networks come together in a single location, that risk grows with the amount of traffic relative to the infrastructure in place to receive that traffic. To avoid the risk of congestion, you could reduce the amount of traffic on the Internet, but that is unlikely. In fact, you are more likely to face contention for available bandwidth as more and more devices become network-enabled. (See, for example, the McKinsey & Company report entitled "The Internet of Things.")

Once again, you encounter a situation that you cannot eliminate but you might be able to avoid. By replicating data and applications, you can avoid having to transmit data over congested paths on the Internet. When replication is not an option, and even in some cases where it is, you can still improve performance using various TCP optimization and end point optimizations such as connection pooling.

Summary

In spite of all the time and effort you might put into tuning your Web application, the Internet can become a bottleneck for your Web application performance. Both the HTTP and TCP protocols have an impact on throughput. Network infrastructure and configuration dictates latency. Aspects such as noise and congestion factor into packet loss rates. These are difficult issues to contend with especially when you deploy an application used on a global scale. Fortunately, there are strategies for dealing with these performance issues, which the next chapter will address.