13.15 Acknowledgements And Retransmission

Because TCP sends data in variable length segments and because retransmitted segments can include more data than the original, acknowledgements cannot easily refer to datagrams or segments. Instead, they refer to a position in the stream using the stream sequence numbers. The receiver collects data octets from arriving segments and reconstructs an exact copy of the stream being sent. Because segments travel in IP datagrams, they can be lost or delivered out of order; the receiver uses the sequence numbers to reorder segments. At any time, the receiver will have reconstructed zero or more octets contiguously from the beginning of the stream, but may have additional pieces of the stream from datagrams that arrived out of order. The receiver always acknowledges the longest contiguous prefix of the stream that has been received correctly. Each acknowledgement specifies a sequence value one greater than the highest octet position in the contiguous prefix it received. Thus, the sender receives continuous feedback from the receiver as it progresses through the stream. We can summarize this important idea:

> *A TCP acknowledgement specifies the sequence number of the next octet that the receiver expects to receive.*

The TCP acknowledgement scheme is called *cumulative* because it reports how much of the stream has accumulated. Cumulative acknowledgements have both advantages and disadvantages. One advantage is that acknowledgements are both easy to generate and unambiguous. Another advantage is that lost acknowledgements do not necessarily force retransmission. A major disadvantage is that the sender does not receive information about all successful transmissions, but only about a single position in the stream that has been received.

To understand why lack of information about all successful transmissions makes cumulative acknowledgements less efficient, think of a window that spans *5000* octets starting at position *101* in the stream, and suppose the sender has transmitted all data in the window by sending five segments. Suppose further that the first segment is lost, but all others arrive intact. As each segment arrives, the receiver sends an acknowledgement, but each acknowledgement specifies octet *101*, the next highest contiguous octet it expects to receive. There is no way for the receiver to tell the sender that most of the data for the current window has arrived.

When a timeout occurs at the sender's side, the sender must choose between two potentially inefficient schemes. It may choose to retransmit one segment or all five segments. In this case retransmitting all five segments is inefficient. When the first segment arrives, the receiver will have all the data in the window, and will acknowledge *5101*. If the sender follows the accepted standard and retransmits only the first unacknowledged segment, it must wait for the acknowledgement before it can decide what and how much to send. Thus, it reverts to a simple positive acknowledgement protocol and may lose the advantages of having a large window.

13.16 Timeout And Retransmission

One of the most important and complex ideas in TCP is embedded in the way it handles timeout and retransmission. Like other reliable protocols, TCP expects the destination to send acknowledgements whenever it successfully receives new octets from the data stream. Every time it sends a segment, TCP starts a timer and waits for an acknowledgement. If the timer expires before data in the segment has been acknowledged, TCP assumes that the segment was lost or corrupted and retransmits it.

To understand why the TCP retransmission algorithm differs from the algorithm used in many network protocols, we need to remember that TCP is intended for use in an internet environment. In an internet, a segment traveling between a pair of machines may traverse a single, low-delay network (e.g., a high-speed LAN), or it may travel across multiple intermediate networks through multiple routers. Thus, it is impossible to know *a priori* how quickly acknowledgements will return to the source. Furthermore, the delay at each router depends on traffic, so the total time required for a segment to travel to the destination and an acknowledgement to return to the source varies dramatically from one instant to another. Figure 13.10, which shows measurements of round trip times across the global Internet for 100 consecutive packets, illustrates the problem. TCP software must accommodate both the vast differences in the time required to reach various destinations and the changes in time required to reach a given destination as traffic load varies.

TCP accommodates varying internet delays by using an *adaptive retransmission algorithm*. In essence, TCP monitors the performance of each connection and deduces reasonable values for timeouts. As the performance of a connection changes, TCP revises its timeout value (i.e., it adapts to the change).

To collect the data needed for an adaptive algorithm, TCP records the time at which each segment is sent and the time at which an acknowledgement arrives for the data in that segment. From the two times, TCP computes an elapsed time known as a *sample round trip time* or *round trip sample*. Whenever it obtains a new round trip sample, TCP adjusts its notion of the average round trip time for the connection. Usually, TCP software stores the estimated round trip time, *RTT*, as a weighted average and uses new round trip samples to change the average slowly. For example, when computing a new weighted average, one early averaging technique used a constant weighting factor, α, where $0 \leq \alpha < 1$, to weight the old average against the latest round trip sample:

$$\text{RTT} = (\alpha * \text{Old\_RTT}) + ((1-\alpha) * \text{New\_Round\_Trip\_Sample})$$

Choosing a value for α close to *1* makes the weighted average immune to changes that last a short time (e.g., a single segment that encounters long delay). Choosing a value for α close to *0* makes the weighted average respond to changes in delay very quickly.

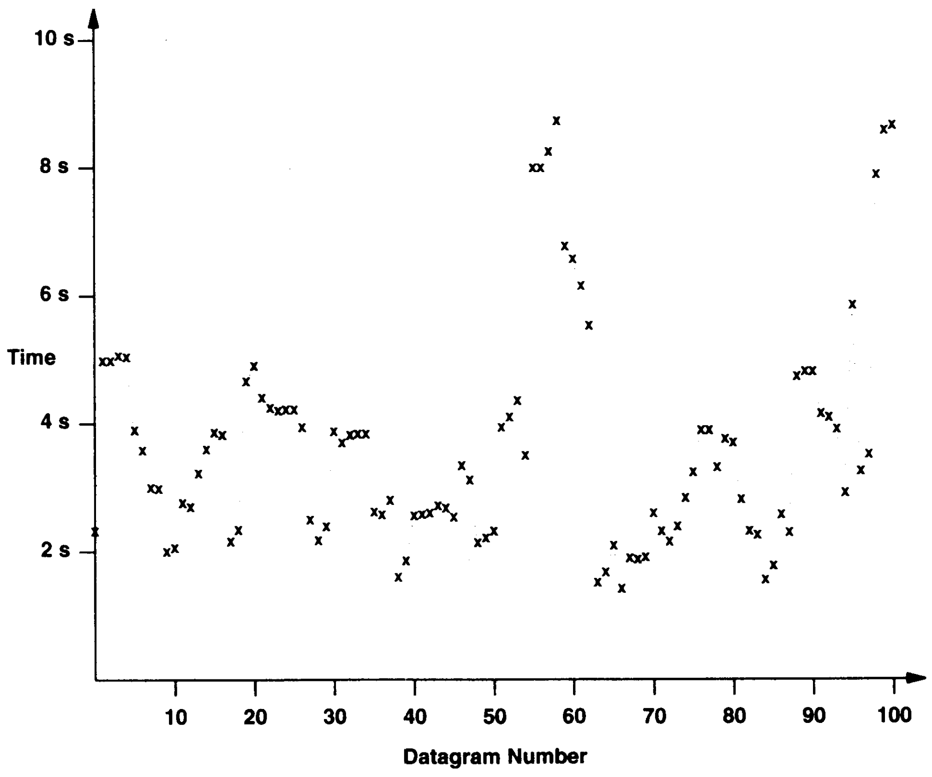


Figure 13.10 A plot of Internet round trip times as measured for 100 successive IP datagrams. Although the Internet now operates with much lower delay, the delays still vary over time.

When it sends a packet, TCP computes a timeout value as a function of the current round trip estimate. Early implementations of TCP used a constant weighting factor, β ($\beta > 1$), and made the timeout greater than the current round trip estimate:

$$\text{Timeout} = \beta * \text{RTT}$$

Choosing a value for β can be difficult. On one hand, to detect packet loss quickly, the timeout value should be close to the current round trip time (i.e., β should be close to *1*). Detecting packet loss quickly improves throughput because TCP will not wait an unnecessarily long time before retransmitting. On the other hand, if $\beta = 1$, TCP is overly eager — any small delay will cause an unnecessary retransmission, which wastes network bandwidth. The original specification recommended setting $\beta = 2$; more recent work described below has produced better techniques for adjusting timeout.

We can summarize the ideas presented so far:

> *To accommodate the varying delays encountered in an internet environment, TCP uses an adaptive retransmission algorithm that monitors delays on each connection and adjusts its timeout parameter accordingly.*

13.17 Accurate Measurement Of Round Trip Samples

In theory, measuring a round trip sample is trivial — it consists of subtracting the time at which the segment is sent from the time at which the acknowledgement arrives. However, complications arise because TCP uses a cumulative acknowledgement scheme in which an acknowledgement refers to data received, and not to the instance of a specific datagram that carried the data. Consider a retransmission. TCP forms a segment, places it in a datagram and sends it, the timer expires, and TCP sends the segment again in a second datagram. Because both datagrams carry exactly the same data, the sender has no way of knowing whether an acknowledgement corresponds to the original or retransmitted datagram. This phenomenon has been called *acknowledgement ambiguity*, and TCP acknowledgements are said to be *ambiguous*.

Should TCP assume acknowledgements belong with the earliest (i.e., original) transmission or the latest (i.e., the most recent retransmission)? Surprisingly, neither assumption works. Associating the acknowledgement with the original transmission can make the estimated round trip time grow without bound in cases where an internet loses datagrams†. If an acknowledgement arrives after one or more retransmissions, TCP will measure the round trip sample from the original transmission, and compute a new RTT using the excessively long sample. Thus, RTT will grow slightly. The next time TCP sends a segment, the larger RTT will result in slightly longer timeouts, so if an acknowledgement arrives after one or more retransmissions, the next sample round trip time will be even larger, and so on.

Associating the acknowledgement with the most recent retransmission can also fail. Consider what happens when the end-to-end delay suddenly increases. When TCP sends a segment, it uses the old round trip estimate to compute a timeout, which is now too small. The segment arrives and an acknowledgement starts back, but the increase in delay means the timer expires before the acknowledgement arrives, and TCP retransmits the segment. Shortly after TCP retransmits, the first acknowledgement arrives and is associated with the retransmission. The round trip sample will be much too small and will result in a slight decrease of the estimated round trip time, RTT. Unfortunately, lowering the estimated round trip time guarantees that TCP will set the timeout too small for the next segment. Ultimately, the estimated round trip time can stabilize at a value, *T*, such that the correct round trip time is slightly longer than some multiple of *T*. Implementations of TCP that associate acknowledgements with the most recent retransmission have been observed in a stable state with RTT slightly less than one-half of the correct value (i.e., TCP sends each segment exactly twice even though no loss occurs).

†The estimate can only grow arbitrarily large if every segment is lost at least once.

13.18 Karn's Algorithm And Timer Backoff

If the original transmission and the most recent transmission both fail to provide accurate round trip times, what should TCP do? The accepted answer is simple: TCP should not update the round trip estimate for retransmitted segments. This idea, known as *Karn's Algorithm*, avoids the problem of ambiguous acknowledgements altogether by only adjusting the estimated round trip for unambiguous acknowledgements (acknowledgements that arrive for segments that have only been transmitted once).

Of course, a simplistic implementation of Karn's algorithm, one that merely ignores times from retransmitted segments, can lead to failure as well. Consider what happens when TCP sends a segment after a sharp increase in delay. TCP computes a timeout using the existing round trip estimate. The timeout will be too small for the new delay and will force retransmission. If TCP ignores acknowledgements from retransmitted segments, it will never update the estimate and the cycle will continue.

To accommodate such failures, Karn's algorithm requires the sender to combine retransmission timeouts with a *timer backoff* strategy. The backoff technique computes an initial timeout using a formula like the one shown above. However, if the timer expires and causes a retransmission, TCP increases the timeout. In fact, each time it must retransmit a segment, TCP increases the timeout (to keep timeouts from becoming ridiculously long, most implementations limit increases to an upper bound that is larger than the delay along any path in the internet).

Implementations use a variety of techniques to compute backoff. Most choose a multiplicative factor, γ, and set the new value to:

$$\text{new\_timeout} = \gamma * \text{timeout}$$

Typically, γ is 2. (It has been argued that values of γ less than 2 lead to instabilities.) Other implementations use a table of multiplicative factors, allowing arbitrary backoff at each step†.

Karn's algorithm combines the backoff technique with round trip estimation to solve the problem of never increasing round trip estimates:

> *Karn's algorithm: When computing the round trip estimate, ignore samples that correspond to retransmitted segments, but use a backoff strategy, and retain the timeout value from a retransmitted packet for subsequent packets until a valid sample is obtained.*

Generally speaking, when an internet misbehaves, Karn's algorithm separates computation of the timeout value from the current round trip estimate. It uses the round trip estimate to compute an initial timeout value, but then backs off the timeout on each retransmission until it can successfully transfer a segment. When it sends subsequent segments, it retains the timeout value that results from backoff. Finally, when an acknowledgement arrives corresponding to a segment that did not require retransmission,

†Berkeley UNIX is the most notable system that uses a table of factors, but current values in the table are equivalent to using $\gamma=2$.

TCP recomputes the round trip estimate and resets the timeout accordingly. Experience shows that Karn's algorithm works well even in networks with high packet loss†.

13.19 Responding To High Variance In Delay

Research into round trip estimation has shown that the computations described above do not adapt to a wide range of variation in delay. Queueing theory suggests that the variation in round trip time, σ, varies proportional to $1/(1\text{-}L)$, where L is the current network load, $0 \le L \le 1$. If an internet is running at 50% of capacity, we expect the round trip delay to vary by a factor of $\pm 2\sigma$, or *4*. When the load reaches 80%, we expect a variation of *10*. The original TCP standard specified the technique for estimating round trip time that we described earlier. Using that technique and limiting β to the suggested value of *2* means the round trip estimation can adapt to loads of at most 30%.

The 1989 specification for TCP requires implementations to estimate both the average round trip time and the variance, and to use the estimated variance in place of the constant β. As a result, new implementations of TCP can adapt to a wider range of variation in delay and yield substantially higher throughput. Fortunately, the approximations require little computation; extremely efficient programs can be derived from the following simple equations:

$$\text{DIFF} = \text{SAMPLE} - \text{Old\_RTT}$$

$$\text{Smoothed\_RTT} = \text{Old\_RTT} + \delta * \text{DIFF}$$

$$\text{DEV} = \text{Old\_DEV} + \rho\,(|\text{DIFF}| - \text{Old\_DEV})$$

$$\text{Timeout} = \text{Smoothed\_RTT} + \eta * \text{DEV}$$

where *DEV* is the estimated mean deviation, δ is a fraction between *0* and *1* that controls how quickly the new sample affects the weighted average, ρ is a fraction between *0* and *1* that controls how quickly the new sample affects the mean deviation, and η is a factor that controls how much the deviation affects the round trip timeout. To make the computation efficient, TCP chooses δ and ρ to each be an inverse of a power of *2*, scales the computation by 2^n for an appropriate n, and uses integer arithmetic. Research suggests values of $\delta = 1/2^3$, $\rho = 1/2^2$, and $n = 3$ will work well. The original value for η in 4.3BSD UNIX was *2*; it was changed to *4* in 4.4 BSD UNIX.

Figure 13.11 uses a set of randomly generated values to illustrate how the computed timeout changes as the roundtrip time varies. Although the roundtrip times are artificial, they follow a pattern observed in practice: successive packets show small variations in delay as the overall average rises or falls.

†Phil Karn is an amateur radio enthusiast who developed this algorithm to allow TCP communication across a high-loss packet radio connection.

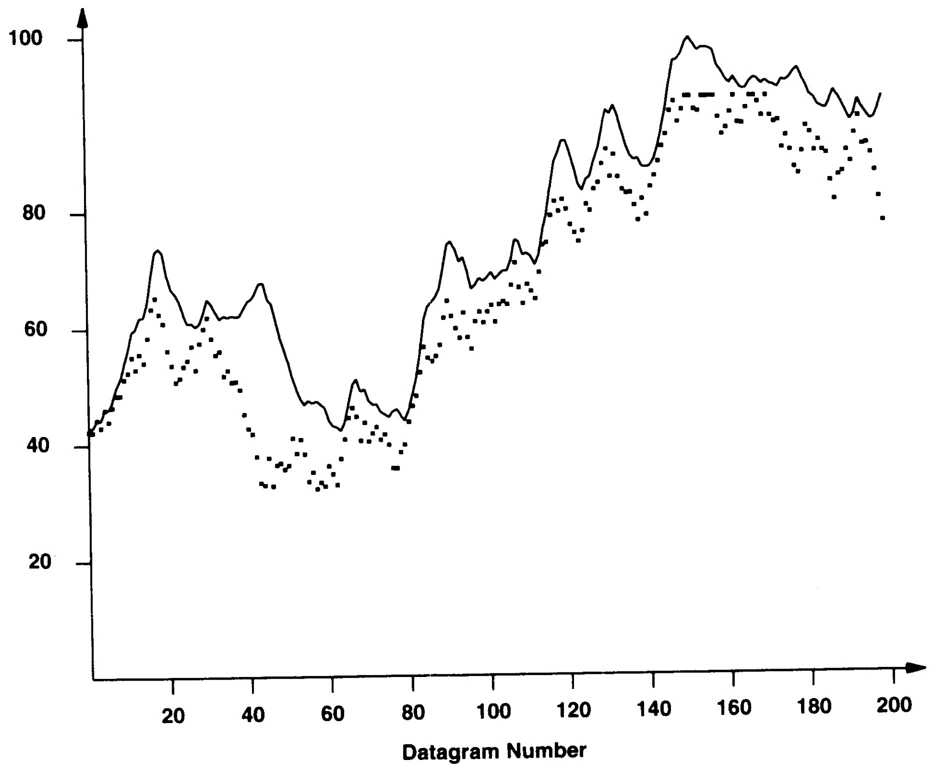


Figure 13.11 A set of 200 (randomly generated) roundtrip times shown as dots, and the TCP retransmission timer shown as a solid line. The timeout increases when delay varies.

Note that frequent change in the roundtrip time, including a cycle of increase and decrease, can produce an increase in the retransmission timer. Furthermore, although the timer tends to increase quickly when delay rises, it does not decrease as rapidly when delay falls.

Figure 13.12 uses the data points from Figure 13.10 to show how TCP responds to the extreme case of variance in delay. Recall that the goal is to have the retransmission timer estimate the actual roundtrip time as closely as possible without underestimating. The figure shows that although the timer responds quickly, it can underestimate. For example, between the two successive datagrams marked with arrows, the delay doubles from less than 4 seconds to more than 8. More important, the abrupt change follows a period of relative stability in which the variation in delay is small, making it impossible for any algorithm to anticipate the change. In the case of the TCP algorithm, because the timeout (approximately 5) substantially underestimates the large delay, an unnecessary retransmission occurs. However, the estimate responds quickly to the increase in delay, meaning that successive packets arrive without retransmission.

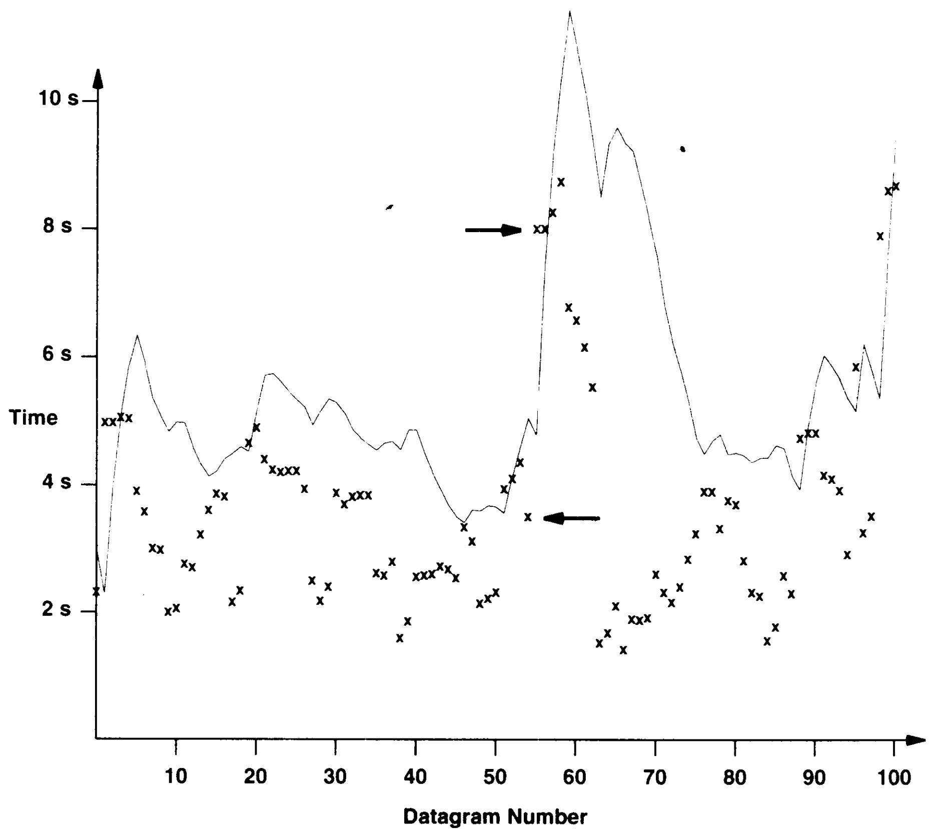


Figure 13.12 The TCP retransmission timer for the data from Figure 13.10. Arrows mark two successive datagrams where the delay doubles.

13.20 Response To Congestion

It may seem that TCP software could be designed by considering the interaction between the two endpoints of a connection and the communication delays between those endpoints. In practice, however, TCP must also react to *congestion* in the internet. Congestion is a condition of severe delay caused by an overload of datagrams at one or more switching points (e.g., at routers). When congestion occurs, delays increase and the router begins to enqueue datagrams until it can route them. We must remember that each router has finite storage capacity and that datagrams compete for that storage (i.e., in a datagram based internet, there is no preallocation of resources to individual TCP connections). In the worst case, the total number of datagrams arriving at the congested router grows until the router reaches capacity and starts to drop datagrams.

Endpoints do not usually know the details of where congestion has occurred or why. To them, congestion simply means increased delay. Unfortunately, most transport protocols use timeout and retransmission, so they respond to increased delay by retransmitting datagrams. Retransmissions aggravate congestion instead of alleviating it. If unchecked, the increased traffic will produce increased delay, leading to increased traffic, and so on, until the network becomes useless. The condition is known as *congestion collapse.*

To avoid congestion collapse, TCP must reduce transmission rates when congestion occurs. Routers watch queue lengths and use techniques like ICMP source quench to inform hosts that congestion has occurred†, but transport protocols like TCP can help avoid congestion by reducing transmission rates automatically whenever delays occur. Of course, algorithms to avoid congestion must be constructed carefully because even under normal operating conditions an internet will exhibit wide variation in round trip delays.

To avoid congestion, the TCP standard now recommends using two techniques: *slow-start* and *multiplicative decrease.* They are related and can be implemented easily. We said that for each connection, TCP must remember the size of the receiver's window (i.e., the buffer size advertised in acknowledgements). To control congestion TCP maintains a second limit, called the *congestion window limit* or *congestion window,* that it uses to restrict data flow to less than the receiver's buffer size when congestion occurs. That is, at any time, TCP acts as if the window size is:

Allowed_window = min(receiver_advertisement, congestion_window)

In the steady state on a non-congested connection, the congestion window is the same size as the receiver's window. Reducing the congestion window reduces the traffic TCP will inject into the connection. To estimate congestion window size, TCP assumes that most datagram loss comes from congestion and uses the following strategy:

> *Multiplicative Decrease Congestion Avoidance: Upon loss of a segment, reduce the congestion window by half (down to a minimum of at least one segment). For those segments that remain in the allowed window, backoff the retransmission timer exponentially.*

Because TCP reduces the congestion window by half for *every* loss, it decreases the window exponentially if loss continues. In other words, if congestion is likely, TCP reduces the volume of traffic exponentially and the rate of retransmission exponentially. If loss continues, TCP eventually limits transmission to a single datagram and continues to double timeout values before retransmitting. The idea is to provide quick and significant traffic reduction to allow routers enough time to clear the datagrams already in their queues.

How can TCP recover when congestion ends? You might suspect that TCP should reverse the multiplicative decrease and double the congestion window when traffic begins to flow again. However, doing so produces an unstable system that oscillates wild-

†In a congested network, queue lengths grow exponentially for a significant time.

ly between no traffic and congestion. Instead, TCP uses a technique called *slow-start*† to scale up transmission:

> *Slow-Start (Additive) Recovery: Whenever starting traffic on a new connection or increasing traffic after a period of congestion, start the congestion window at the size of a single segment and increase the congestion window by one segment each time an acknowledgement arrives.*

Slow-start avoids swamping the internet with additional traffic immediately after congestion clears or when new connections suddenly start.

The term *slow-start* may be a misnomer because under ideal conditions, the start is not very slow. TCP initializes the congestion window to *1*, sends an initial segment, and waits. When the acknowledgement arrives, it increases the congestion window to *2*, sends two segments, and waits. When the two acknowledgements arrive they each increase the congestion window by *1*, so TCP can send *4* segments. Acknowledgements for those will increase the congestion window to *8*. Within four round-trip times, TCP can send *16* segments, often enough to reach the receiver's window limit. Even for extremely large windows, it takes only $\log_2 N$ round trips before TCP can send *N* segments.

To avoid increasing the window size too quickly and causing additional congestion, TCP adds one additional restriction. Once the congestion window reaches one half of its original size before congestion, TCP enters a *congestion avoidance* phase and slows down the rate of increment. During congestion avoidance, it increases the congestion window by *1* only if all segments in the window have been acknowledged.

Taken together, slow-start increase, multiplicative decrease, congestion avoidance, measurement of variation, and exponential timer backoff improve the performance of TCP dramatically without adding any significant computational overhead to the protocol software. Versions of TCP that use these techniques have improved the performance of previous versions by factors of *2* to *10*.

13.21 Congestion, Tail Drop, And TCP

We said that communication protocols are divided into layers to make it possible for designers to focus on a single problem at a time. The separation of functionality into layers is both necessary and useful — it means that one layer can be changed without affecting other layers, but it means that layers operate in isolation. For example, because it operates end-to-end, TCP remains unchanged when the path between the endpoints changes (e.g., routes change or additional networks routers are added). However, the isolation of layers restricts inter-layer communication. In particular, although TCP on the original source interacts with TCP on the ultimate destination, it cannot interact with lower layer elements along the path. Thus, neither the sending nor receiving

†The term *slow-start* is attributed to John Nagle; the technique was originally called *soft-start*.

TCP receives reports about conditions in the network, nor does either end inform lower layers along the path before transferring data.

Researchers have observed that the lack of communication between layers means that the choice of policy or implementation at one layer can have a dramatic effect on the performance of higher layers. In the case of TCP, policies that routers use to handle datagrams can have a significant effect on both the performance of a single TCP connection and the aggregate throughput of all connections. For example, if a router delays some datagrams more than others†, TCP will back off its retransmission timer. If the delay exceeds the retransmission timeout, TCP will assume congestion has occurred. Thus, although each layer is defined independently, researchers try to devise mechanisms and implementations that work well with protocols in other layers.

The most important interaction between IP implementation policies and TCP occurs when a router becomes overrun and drops datagrams. Because a router places each incoming datagram in a queue in memory until it can be processed, the policy focuses on queue management. When datagrams arrive faster than they can be forwarded, the queue grows; when datagrams arrive slower than they can be forwarded, the queue shrinks. However, because memory is finite, the queue cannot grow without bound. Early router software used a *tail-drop* policy to manage queue overflow:

> *Tail-Drop Policy For Routers: if the input queue is filled when a datagram arrives, discard the datagram.*

The name *tail-drop* arises from the effect of the policy on an arriving sequence of datagrams. Once the queue fills, the router begins discarding all additional datagrams. That is, the router discards the "tail" of the sequence.

Tail-drop has an interesting effect on TCP. In the simple case where datagrams traveling through a router carry segments from a single TCP connection, the loss causes TCP to enter slow-start, which reduces throughput until TCP begins receiving ACKs and increases the congestion window. A more severe problem can occur, however, when the datagrams traveling through a router carry segments from many TCP connections because tail-drop can cause global synchronization. To see why, observe that datagrams are typically multiplexed, with successive datagrams each coming from a different source. Thus, a tail-drop policy makes it likely that the router will discard one segment from *N* connections rather than *N* segments from one connection. The simultaneous loss causes all *N* instances of TCP to enter slow-start at the same time.

13.22 Random Early Discard (RED)

How can a router avoid global synchronization? The answer lies in a clever scheme that avoids tail-drop whenever possible. Known as *Random Early Discard*, *Random Early Drop*, or *Random Early Detection*, the scheme is more frequently referred to by its acronym, *RED*. A router that implements RED uses two threshold

†Technically, variance in delay is referred to as *jitter*.

values to mark positions in the queue: T_{min} and T_{max}. The general operation of RED can be described by three rules that determine the disposition of each arriving datagram:

- If the queue currently contains fewer than T_{min} datagrams, add the new datagram to the queue.
- If the queue contains more than T_{max} datagrams, discard the new datagram.
- If the queue contains between T_{min} and T_{max} datagrams, randomly discard the datagram according to a probability, p.

The randomness of RED means that instead of waiting until the queue overflows and then driving many TCP connections into slow-start, a router slowly and randomly drops datagrams as congestion increases. We can summarize:

> *RED Policy For Routers: if the input queue is full when a datagram arrives, discard the datagram; if the input queue is not full but the size exceeds a minimum threshold, avoid synchronization by discarding the datagram with probability* p.

The key to making RED work well lies in the choice of the thresholds T_{min} and T_{max}, and the discard probability p. T_{min} must be large enough to ensure that the output link has high utilization. Furthermore, because RED operates like tail-drop when the queue size exceeds T_{max}, the value must be greater than T_{min} by more than the typical increase in queue size during one TCP round trip time (e.g., set T_{max} at least twice as large as T_{min}). Otherwise, RED can cause the same global oscillations as tail-drop.

Computation of the discard probability, p, is the most complex aspect of RED. Instead of using a constant, a new value of p is computed for each datagram; the value depends on the relationship between the current queue size and the thresholds. To understand the scheme, observe that all RED processing can be viewed probabilistically. When the queue size is less than T_{min}, RED does not discard any datagrams, making the discard probability *0*. Similarly, when the queue size is greater than T_{max}, RED discards all datagrams, making the discard probability *1*. For intermediate values of queue size, (i.e., those between T_{min} and T_{max}), the probability can vary from *0* to *1* linearly.

Although the linear scheme forms the basis of RED's probability computation, a change must be made to avoid overreacting. The need for the change arises because network traffic is bursty, which results in rapid fluctuations of a router's queue. If RED used a simplistic linear scheme, later datagrams in each burst would be assigned high probability of being dropped (because they arrive when the queue has more entries). However, a router should not drop datagrams unnecessarily because doing so has a negative impact on TCP throughput. Thus, if a burst is short, it is unwise to drop datagrams because the queue will never overflow. Of course, RED cannot postpone discard indefinitely because a long-term burst will overflow the queue, resulting in a tail-drop policy which has the potential to cause global synchronization problems.

How can RED assign a higher discard probability as the queue fills without discarding datagrams from each burst? The answer lies in a technique borrowed from TCP: instead of using the actual queue size at any instant, RED computes a weighted average queue size, *avg*, and uses the average size to determine the probability. The value of *avg* is an exponential weighted average, updated each time a datagram arrives according to the equation:

$$\text{avg} = (1 - \gamma) * \text{Old\_avg} + \gamma * \text{Current\_queue\_size}$$

where γ denotes a value between *0* and *1*. If γ is small enough, the average will track long term trends, but will remain immune to short bursts†

In addition to equations that determine γ, RED contains other details that we have glossed over. For example, RED computations can be made extremely efficient by choosing constants as powers of two and using integer arithmetic. Another important detail concerns the measurement of queue size, which affects both the RED computation and its overall effect on TCP. In particular, because the time required to forward a datagram is proportional to its size, it makes sense to measure the queue in octets rather than in datagrams; doing so requires only minor changes to the equations for p and γ. Measuring queue size in octets affects the type of traffic dropped because it makes the discard probability proportional to the amount of data a sender puts in the stream rather than the number of segments. Small datagrams (e.g., those that carry remote login traffic or requests to servers) have lower probability of being dropped than large datagrams (e.g., those that carry file transfer traffic). One positive consequence of using size is that when acknowledgements travel over a congested path, they have a lower probability of being dropped. As a result, if a (large) data segment does arrive, the sending TCP will receive the ACK and will avoid unnecessary retransmission.

Both analysis and simulations show that RED works well. It handles congestion, avoids the synchronization that results from tail drop, and allows short bursts without dropping datagrams unnecessarily. The IETF now recommends that routers implement RED.

13.23 Establishing A TCP Connection

To establish a connection, TCP uses a *three-way handshake*. In the simplest case, the handshake proceeds as Figure 13.13 shows.

†An example value suggested for γ is .002.

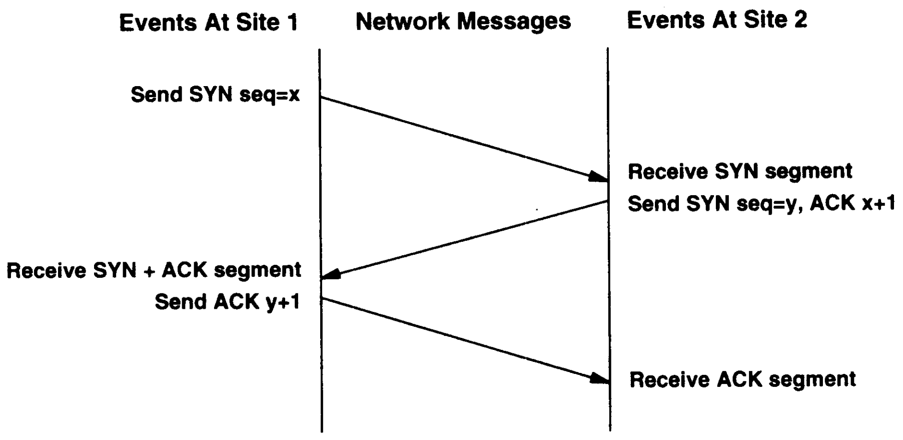


Figure 13.13 The sequence of messages in a three-way handshake. Time proceeds down the page; diagonal lines represent segments sent between sites. SYN segments carry initial sequence number information.

The first segment of a handshake can be identified because it has the SYN† bit set in the code field. The second message has both the SYN bit and ACK bits set, indicating that it acknowledges the first SYN segment as well as continuing the handshake. The final handshake message is only an acknowledgement and is merely used to inform the destination that both sides agree that a connection has been established.

Usually, the TCP software on one machine waits passively for the handshake, and the TCP software on another machine initiates it. However, the handshake is carefully designed to work even if both machines attempt to initiate a connection simultaneously. Thus, a connection can be established from either end or from both ends simultaneously. Once the connection has been established, data can flow in both directions equally well. There is no master or slave.

The three-way handshake is both necessary and sufficient for correct synchronization between the two ends of the connection. To understand why, remember that TCP builds on an unreliable packet delivery service, so messages can be lost, delayed, duplicated, or delivered out of order. Thus, the protocol must use a timeout mechanism and retransmit lost requests. Trouble arises if retransmitted and original requests arrive while the connection is being established, or if retransmitted requests are delayed until after a connection has been established, used, and terminated. A three-way handshake (plus the rule that TCP ignores additional requests for connection after a connection has been established) solves these problems.

†SYN stands for *synchronization*; it is pronounced "sin."

13.24 Initial Sequence Numbers

The three-way handshake accomplishes two important functions. It guarantees that both sides are ready to transfer data (and that they know they are both ready), and it allows both sides to agree on initial sequence numbers. Sequence numbers are sent and acknowledged during the handshake. Each machine must choose an initial sequence number at random that it will use to identify bytes in the stream it is sending. Sequence numbers cannot always start at the same value. In particular, TCP cannot merely choose sequence *1* every time it creates a connection (one of the exercises examines problems that can arise if it does). Of course, it is important that both sides agree on an initial number, so octet numbers used in acknowledgements agree with those used in data segments.

To see how machines can agree on sequence numbers for two streams after only three messages, recall that each segment contains both a sequence number field and an acknowledgement field. The machine that initiates a handshake, call it *A*, passes its initial sequence number, *x*, in the sequence field of the first SYN segment in the three-way handshake. The second machine, *B*, receives the SYN, records the sequence number, and replies by sending its initial sequence number in the sequence field as well as an acknowledgement that specifies *B* expects octet $x+1$. In the final message of the handshake, *A* "acknowledges" receiving from *B* all octets through *y*. In all cases, acknowledgements follow the convention of using the number of the *next* octet expected.

We have described how TCP usually carries out the three-way handshake by exchanging segments that contain a minimum amount of information. Because of the protocol design, it is possible to send data along with the initial sequence numbers in the handshake segments. In such cases, the TCP software must hold the data until the handshake completes. Once a connection has been established, the TCP software can release data being held and deliver it to a waiting application program quickly. The reader is referred to the protocol specification for the details.

13.25 Closing a TCP Connection

Two programs that use TCP to communicate can terminate the conversation gracefully using the *close* operation. Internally, TCP uses a modified three-way handshake to close connections. Recall that TCP connections are full duplex and that we view them as containing two independent stream transfers, one going in each direction. When an application program tells TCP that it has no more data to send, TCP will close the connection *in one direction*. To close its half of a connection, the sending TCP finishes transmitting the remaining data, waits for the receiver to acknowledge it, and then sends a segment with the FIN bit set. The receiving TCP acknowledges the FIN segment and informs the application program on its end that no more data is available (e.g., using the operating system's end-of-file mechanism).

Once a connection has been closed in a given direction, TCP refuses to accept more data for that direction. Meanwhile, data can continue to flow in the opposite direction until the sender closes it. Of course, acknowledgements continue to flow back to the sender even after a connection has been closed. When both directions have been closed, the TCP software at each endpoint deletes its record of the connection.

The details of closing a connection are even more subtle than suggested above because TCP uses a modified three-way handshake to close a connection. Figure 13.14 illustrates the procedure.

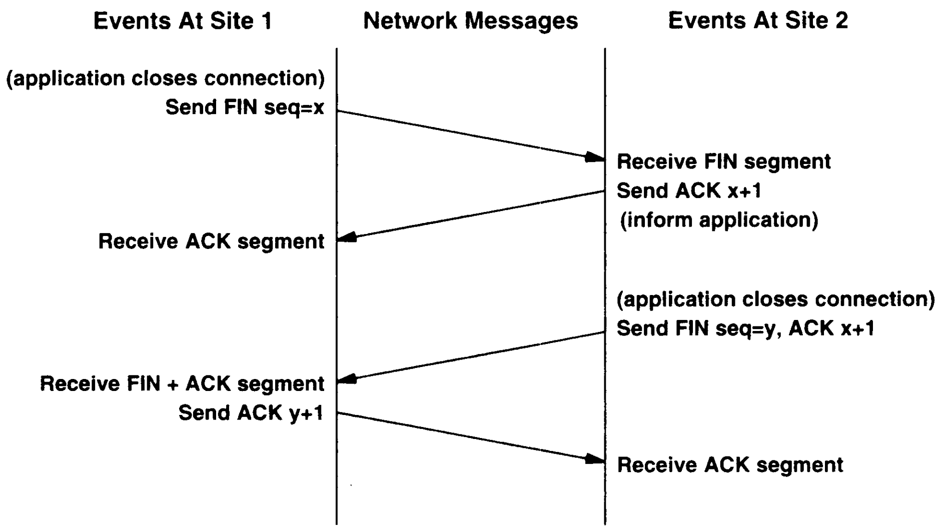


Figure 13.14 The modified three-way handshake used to close connections. The site that receives the first FIN segment acknowledges it immediately and then delays before sending the second FIN segment.

The difference between three-way handshakes used to establish and break connections occurs after a machine receives the initial FIN segment. Instead of generating a second FIN segment immediately, TCP sends an acknowledgement and then informs the application of the request to shut down. Informing the application program of the request and obtaining a response may take considerable time (e.g., it may involve human interaction). The acknowledgement prevents retransmission of the initial FIN segment during the wait. Finally, when the application program instructs TCP to shut down the connection completely, TCP sends the second FIN segment and the original site replies with the third message, an ACK.

13.26 TCP Connection Reset

Normally, an application program uses the close operation to shut down a connection when it finishes using it. Thus, closing connections is considered a normal part of use, analogous to closing files. Sometimes abnormal conditions arise that force an application program or the network software to break a connection. TCP provides a reset facility for such abnormal disconnections.

To reset a connection, one side initiates termination by sending a segment with the RST bit in the *CODE* field set. The other side responds to a reset segment immediately by aborting the connection. TCP also informs the application program that a reset occurred. A reset is an instantaneous abort that means that transfer in both directions ceases immediately, and resources such as buffers are released.

13.27 TCP State Machine

Like most protocols, the operation of TCP can best be explained with a theoretical model called a *finite state machine*. Figure 13.15 shows the TCP finite state machine, with circles representing states and arrows representing transitions between them. The label on each transition shows what TCP receives to cause the transition and what it sends in response. For example, the TCP software at each endpoint begins in the *CLOSED* state. Application programs must issue either a *passive open* command (to wait for a connection from another machine), or an *active open* command (to initiate a connection). An active open command forces a transition from the *CLOSED* state to the *SYN SENT* state. When TCP follows the transition, it emits a SYN segment. When the other end returns a segment that contains a SYN plus ACK, TCP moves to the *ESTABLISHED* state and begins data transfer.

The *TIMED WAIT* state reveals how TCP handles some of the problems incurred with unreliable delivery. TCP keeps a notion of *maximum segment lifetime* (*MSL*), the maximum time an old segment can remain alive in an internet. To avoid having segments from a previous connection interfere with a current one, TCP moves to the *TIMED WAIT* state after closing a connection. It remains in that state for twice the maximum segment lifetime before deleting its record of the connection. If any duplicate segments happen to arrive for the connection during the timeout interval, TCP will reject them. However, to handle cases where the last acknowledgement was lost, TCP acknowledges valid segments and restarts the timer. Because the timer allows TCP to distinguish old connections from new ones, it prevents TCP from responding with a *RST* (reset) if the other end retransmits a *FIN* request.

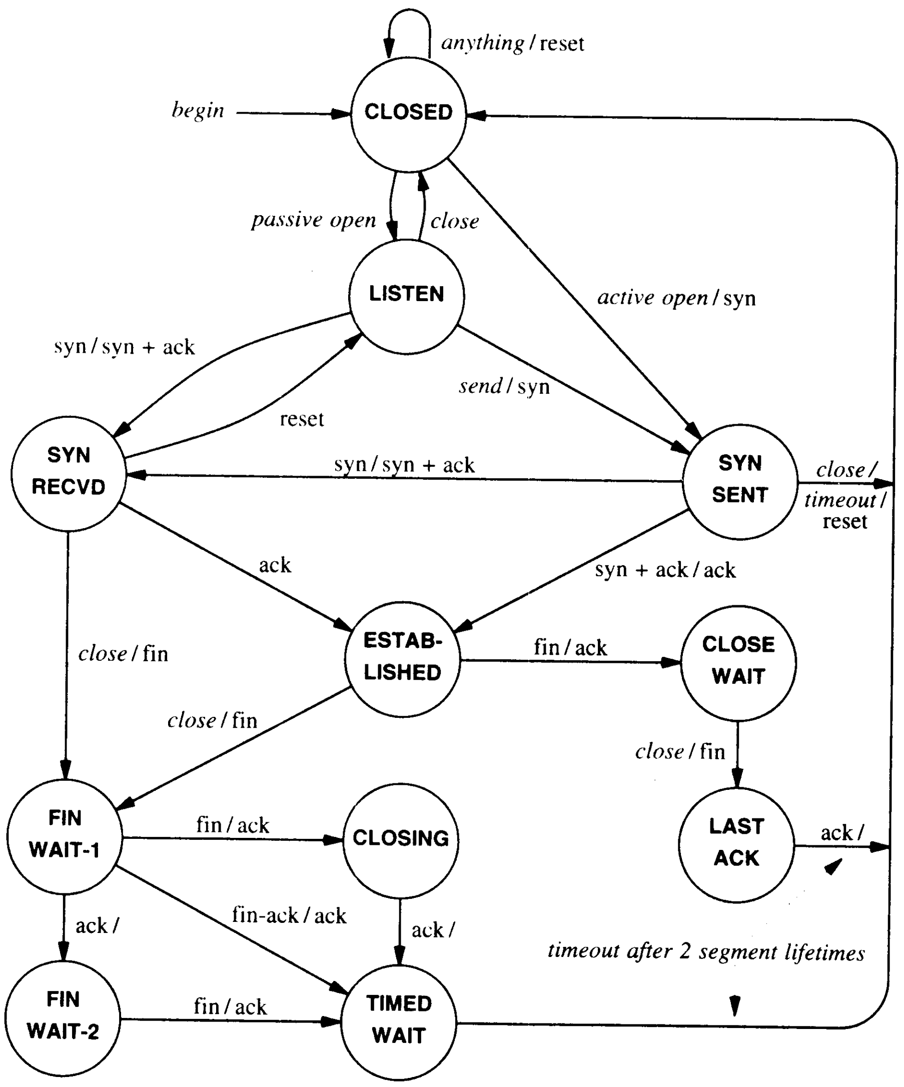


Figure 13.15 The TCP finite state machine. Each endpoint begins in the *closed* state. Labels on transitions show the input that caused the transition followed by the output if any.

13.28 Forcing Data Delivery

We have said that TCP is free to divide the stream of data into segments for transmission without regard to the size of transfer that application programs use. The chief advantage of allowing TCP to choose a division is efficiency. It can accumulate enough octets in a buffer to make segments reasonably long, reducing the high overhead that occurs when segments contain only a few data octets.

Although buffering improves network throughput, it can interfere with some applications. Consider using a TCP connection to pass characters from an interactive terminal to a remote machine. The user expects instant response to each keystroke. If the sending TCP buffers the data, response may be delayed, perhaps for hundreds of keystrokes. Similarly, because the receiving TCP may buffer data before making it available to the application program on its end, forcing the sender to transmit data may not be sufficient to guarantee delivery.

To accommodate interactive users, TCP provides a *push* operation that an application program can use to force delivery of octets currently in the stream without waiting for the buffer to fill. The push operation does more than force TCP to send a segment. It also requests TCP to set the *PSH* bit in the segment code field, so the data will be delivered to the application program on the receiving end. Thus, when sending data from an interactive terminal, the application uses the push function after each keystroke. Similarly, application programs can force output to be sent and displayed on the terminal promptly by calling the push function after writing a character or line.

13.29 Reserved TCP Port Numbers

Like UDP, TCP combines static and dynamic port binding, using a set of *well-known port assignments* for commonly invoked programs (e.g., electronic mail), but leaving most port numbers available for the operating system to allocate as programs need them. Although the standard originally reserved port numbers less than *256* for use as well-known ports, numbers over 1024 have now been assigned. Figure 13.16 lists some of the currently assigned TCP ports. It should be pointed out that although TCP and UDP port numbers are independent, the designers have chosen to use the same integer port numbers for any service that is accessible from both UDP and TCP. For example, a domain name server can be accessed either with TCP or with UDP. In either protocol, port number *53* has been reserved for servers in the domain name system.

13.30 TCP Performance

As we have seen, TCP is a complex protocol that handles communication over a wide variety of underlying network technologies. Many people assume that because TCP tackles a much more complex task than other transport protocols, the code must be cumbersome and inefficient. Surprisingly, the generality we discussed does not seem to hinder TCP performance. Experiments at Berkeley have shown that the same TCP that operates efficiently over the global Internet can deliver 8 Mbps of sustained throughput of user data between two workstations on a 10 Mbps Ethernet†. At Cray Research, Inc., researchers have demonstrated TCP throughput approaching a gigabit per second.

Decimal	Keyword	UNIX Keyword	Description
0			Reserved
1	TCPMUX	-	TCP Multiplexor
7	ECHO	echo	Echo
9	DISCARD	discard	Discard
11	USERS	systat	Active Users
13	DAYTIME	daytime	Daytime
15	-	netstat	Network status program
17	QUOTE	qotd	Quote of the Day
19	CHARGEN	chargen	Character Generator
20	FTP-DATA	ftp-data	File Transfer Protocol (data)
21	FTP	ftp	File Transfer Protocol
22	SSH	ssh	Secure Shell
23	TELNET	telnet	Terminal Connection
25	SMTP	smtp	Simple Mail Transport Protocol
37	TIME	time	Time
43	NICNAME	whois	Who Is
53	DOMAIN	nameserver	Domain Name Server
67	BOOTPS	bootps	BOOTP or DHCP Server
77	-	rje	any private RJE service
79	FINGER	finger	Finger
80	WWW	www	World Wide Web Server
88	KERBEROS	kerberos	Kerberos Security Service
95	SUPDUP	supdup	SUPDUP Protocol
101	HOSTNAME	hostnames	NIC Host Name Server
102	ISO-TSAP	iso-tsap	ISO-TSAP
103	X400	x400	X.400 Mail Service
104	X400-SND	x400-snd	X.400 Mail Sending
110	POP3	pop3	Post Office Protocol Vers. 3
111	SUNRPC	sunrpc	SUN Remote Procedure Call
113	AUTH	auth	Authentication Service
117	UUCP-PATH	uucp-path	UUCP Path Service
119	NNTP	nntp	USENET News Transfer Protocol
123	NTP	ntp	Network Time Protocol
139	NETBIOS-SSN	-	NETBIOS Session Service
161	SNMP	snmp	Simple Network Management Protocol

Figure 13.16 Examples of currently assigned TCP port numbers. To the extent possible, protocols like UDP use the same numbers.

†Ethernet, IP, and TCP headers and the required inter-packet gap account for the remaining bandwidth.

13.31 Silly Window Syndrome And Small Packets

Researchers who helped developed TCP observed a serious performance problem that can result when the sending and receiving applications operate at different speeds. To understand the problem, remember that TCP buffers incoming data, and consider what can happen if a receiving application chooses to read incoming data one octet at a time. When a connection is first established, the receiving TCP allocates a buffer of *K* bytes, and uses the *WINDOW* field in acknowledgement segments to advertise the available buffer size to the sender. If the sending application generates data quickly, the sending TCP will transmit segments with data for the entire window. Eventually, the sender will receive an acknowledgement that specifies the entire window has been filled, and no additional space remains in the receiver's buffer.

When the receiving application reads an octet of data from a full buffer, one octet of space becomes available. We said that when space becomes available in its buffer, TCP on the receiving machine generates an acknowledgement that uses the *WINDOW* field to inform the sender. In the example, the receiver will advertise a window of *1* octet. When it learns that space is available, the sending TCP responds by transmitting a segment that contains one octet of data.

Although single-octet window advertisements work correctly to keep the receiver's buffer filled, they result in a series of small data segments. The sending TCP must compose a segment that contains one octet of data, place the segment in an IP datagram, and transmit the result. When the receiving application reads another octet, TCP generates another acknowledgement, which causes the sender to transmit another segment that contains one octet of data. The resulting interaction can reach a steady state in which TCP sends a separate segment for each octet of data.

Transferring small segments consumes unnecessary network bandwidth and introduces unnecessary computational overhead. The transmission of small segments consumes unnecessary network bandwidth because each datagram carries only one octet of data; the ratio of header to data is large. Computational overhead arises because TCP on both the sending and receiving computers must process each segment. The sending TCP software must allocate buffer space, form a segment header, and compute a checksum for the segment. Similarly, IP software on the sending machine must encapsulate the segment in a datagram, compute a header checksum, route the datagram, and transfer it to the appropriate network interface. On the receiving machine, IP must verify the IP header checksum and pass the segment to TCP. TCP must verify the segment checksum, examine the sequence number, extract the data, and place it in a buffer.

Although we have described how small segments result when a receiver advertises a small available window, a sender can also cause each segment to contain a small amount of data. For example, imagine a TCP implementation that aggressively sends data whenever it is available, and consider what happens if a sending application generates data one octet at a time. After the application generates an octet of data, TCP creates and transmits a segment. TCP can also send a small segment if an application generates data in fixed-sized blocks of *B* octets, and the sending TCP extracts data from the buffer in maximum segment sized blocks, M, where $M \neq B$, because the last block in a buffer can be small.

Known as *silly window syndrome* (*SWS*), the problem plagued early TCP implementations. To summarize,

> *Early TCP implementations exhibited a problem known as* silly window syndrome *in which each acknowledgement advertises a small amount of space available and each segment carries a small amount of data.*

13.32 Avoiding Silly Window Syndrome

TCP specifications now include heuristics that prevent silly window syndrome. A heuristic used on the sending machine avoids transmitting a small amount of data in each segment. Another heuristic used on the receiving machine avoids sending small increments in window advertisements that can trigger small data packets. Although the heuristics work well together, having both the sender and receiver avoid silly window helps ensure good performance in the case that one end of a connection fails to correctly implement silly window avoidance.

In practice, TCP software must contain both sender and receiver silly window avoidance code. To understand why, recall that a TCP connection is full duplex — data can flow in either direction. Thus, an implementation of TCP includes code to send data as well as code to receive it.

13.32.1 Receive-Side Silly Window Avoidance

The heuristic a receiver uses to avoid silly window is straightforward and easiest to understand. In general, a receiver maintains an internal record of the currently available window, but delays advertising an increase in window size to the sender until the window can advance a significant amount. The definition of "significant" depends on the receiver's buffer size and the maximum segment size. TCP defines it to be the minimum of one half of the receiver's buffer or the number of data octets in a maximum-sized segment.

Receive-side silly window prevents small window advertisements in the case where a receiving application extracts data octets slowly. For example, when a receiver's buffer fills completely, it sends an acknowledgement that contains a zero window advertisement. As the receiving application extracts octets from the buffer, the receiving TCP computes the newly available space in the buffer. Instead of sending a window advertisement immediately, however, the receiver waits until the available space reaches one half of the total buffer size or a maximum sized segment. Thus, the sender always receives large increments in the current window, allowing it to transfer large segments. The heuristic can be summarized as follows.

Receive-Side Silly Window Avoidance: Before sending an updated window advertisement after advertising a zero window, wait for space to become available that is either at least 50% of the total buffer size or equal to a maximum sized segment.

13.32.2 Delayed Acknowledgements

Two approaches have been taken to implement silly window avoidance on the receive side. In the first approach, TCP acknowledges each segment that arrives, but does not advertise an increase in its window until the window reaches the limits specified by the silly window avoidance heuristic. In the second approach, TCP delays sending an acknowledgement when silly window avoidance specifies that the window is not sufficiently large to advertise. The standards recommend delaying acknowledgements.

Delayed acknowledgements have both advantages and disadvantages. The chief advantage arises because delayed acknowledgements can decrease traffic and thereby increase throughput. For example, if additional data arrives during the delay period, a single acknowledgement will acknowledge all data received. If the receiving application generates a response immediately after data arrives (e.g., character echo in a remote login session), a short delay may permit the acknowledgement to piggyback on a data segment. Furthermore, TCP cannot move its window until the receiving application extracts data from the buffer. In cases where the receiving application reads data as soon as it arrives, a short delay allows TCP to send a single segment that acknowledges the data and advertises an updated window. Without delayed acknowledgements, TCP will acknowledge the arrival of data immediately, and later send an additional acknowledgement to update the window size.

The disadvantages of delayed acknowledgements should be clear. Most important, if a receiver delays acknowledgements too long, the sending TCP will retransmit the segment. Unnecessary retransmissions lower throughput because they waste network bandwidth. In addition, retransmissions require computational overhead on the sending and receiving machines. Furthermore, TCP uses the arrival of acknowledgements to estimate round trip times; delaying acknowledgements can confuse the estimate and make retransmission times too long.

To avoid potential problems, the TCP standards place a limit on the time TCP delays an acknowledgement. Implementations cannot delay an acknowledgement for more than 500 milliseconds. Furthermore, to guarantee that TCP receives a sufficient number of round trip estimates, the standard recommends that a receiver should acknowledge at least every other data segment.

13.32.3 Send-Side Silly Window Avoidance

The heuristic a sending TCP uses to avoid silly window syndrome is both surprising and elegant. Recall that the goal is to avoid sending small segments. Also recall that a sending application can generate data in arbitrarily small blocks (e.g., one octet at a time). Thus, to achieve the goal, a sending TCP must allow the sending application to make multiple calls to *write*, and must collect the data transferred in each call before transmitting it in a single, large segment. That is, a sending TCP must delay sending a segment until it can accumulate a reasonable amount of data. The technique is known as *clumping*.

The question arises, "How long should TCP wait before transmitting data?" On one hand, if TCP waits too long, the application experiences large delays. More important, TCP cannot know whether to wait because it cannot know whether the application will generate more data in the near future. On the other hand, if TCP does not wait long enough, segments will be small and throughput will be low.

Protocols designed prior to TCP confronted the same problem and used techniques to clump data into larger packets. For example, to achieve efficient transfer across a network, early remote terminal protocols delayed transmitting each keystroke for a few hundred milliseconds to determine whether the user would continue to press keys. Because TCP is designed to be general, however, it can be used by a diverse set of applications. Characters may travel across a TCP connection because a user is typing on a keyboard or because a program is transferring a file. A fixed delay is not optimal for all applications.

Like the algorithm TCP uses for retransmission and the slow-start algorithm used to avoid congestion, the technique a sending TCP uses to avoid sending small packets is adaptive — the delay depends on the current performance of the internet. Like slow-start, send-side silly window avoidance is called *self clocking* because it does not compute delays. Instead, TCP uses the arrival of an acknowledgement to trigger the transmission of additional packets. The heuristic can be summarized:

> *Send-Side Silly Window Avoidance: When a sending application generates additional data to be sent over a connection for which previous data has been transmitted but not acknowledged, place the new data in the output buffer as usual, but do not send additional segments until there is sufficient data to fill a maximum-sized segment. If still waiting to send when an acknowledgement arrives, send all data that has accumulated in the buffer. Apply the rule even when the user requests a* push *operation.*

If an application generates data one octet at a time, TCP will send the first octet immediately. However, until the ACK arrives, TCP will accumulate additional octets in its buffer. Thus, if the application is reasonably fast compared to the network (i.e., a file transfer), successive segments will each contain many octets. If the application is slow compared to the network (e.g., a user typing on a keyboard), small segments will be sent without long delay.

Known as the *Nagle algorithm* after its inventor, the technique is especially elegant because it requires little computational overhead. A host does not need to keep separate timers for each connection, nor does the host need to examine a clock when an application generates data. More important, although the technique adapts to arbitrary combinations of network delay, maximum segment size, and application speed, it does not lower throughput in conventional cases.

To understand why throughput remains high for conventional communication, observe that applications optimized for high throughput do not generate data one octet at a time (doing so would incur unnecessary operating system overhead). Instead, such applications write large blocks of data with each call. Thus, the outgoing TCP buffer begins with sufficient data for at least one maximum size segment. Furthermore, because the application produces data faster than TCP can transfer data, the sending buffer remains nearly full, and TCP does not delay transmission. As a result, TCP continues to send segments at whatever rate the internet can tolerate, while the application continues to fill the buffer. To summarize:

> *TCP now requires the sender and receiver to implement heuristics that avoid the silly window syndrome. A receiver avoids advertising a small window, and a sender uses an adaptive scheme to delay transmission so it can clump data into large segments.*

13.33 Summary

The Transmission Control Protocol, TCP, defines a key service provided by an internet, namely, reliable stream delivery. TCP provides a full duplex connection between two machines, allowing them to exchange large volumes of data efficiently.

Because it uses a sliding window protocol, TCP can make efficient use of a network. Because it makes few assumptions about the underlying delivery system, TCP is flexible enough to operate over a large variety of delivery systems. Because it provides flow control, TCP allows systems of widely varying speeds to communicate.

The basic unit of transfer used by TCP is a segment. Segments are used to pass data or control information (e.g., to allow TCP software on two machines to establish connections or break them). The segment format permits a machine to piggyback acknowledgements for data flowing in one direction by including them in the segment headers of data flowing in the opposite direction.

TCP implements flow control by having the receiver advertise the amount of data it is willing to accept. It also supports out-of-band messages using an urgent data facility and forces delivery using a push mechanism.

The current TCP standard specifies exponential backoff for retransmission timers and congestion avoidance algorithms like slow-start, multiplicative decrease, and additive increase. In addition, TCP uses heuristics to avoid transferring small packets. Finally, the IETF recommends that routers use RED instead of tail-drop because doing so avoids TCP synchronization and improves throughput.

FOR FURTHER STUDY

The standard for TCP can be found in Postel [RFC 793]; Braden [RFC 1122] contains an update that clarifies several points. Clark [RFC 813] describes TCP window management, Clark [RFC 816] describes fault isolation and recovery, and Postel [RFC 879] reports on TCP maximum segment sizes. Nagle [RFC 896] comments on congestion in TCP/IP networks and explains the effect of self clocking for send-side silly window avoidance. Karn and Partridge [1987] discusses estimation of round-trip times, and presents Karn's algorithm. Jacobson [1988] gives the congestion control algorithms that are now a required part of the standard. Floyd and Jacobson [1993] presents the RED scheme, and Clark and Fang [1998] discusses an allocation framework that uses RED. Tomlinson [1975] considers the three-way handshake in more detail. Mills [RFC 889] reports measurements of Internet round-trip delays. Jain [1986] describes timer-based congestion control in a sliding window environment. Borman [April 1989] summarizes experiments with high-speed TCP on Cray computers.

EXERCISES

13.1 TCP uses a finite field to contain stream sequence numbers. Study the protocol specification to find out how it allows an arbitrary length stream to pass from one machine to another.

13.2 The text notes that one of the TCP options permits a receiver to specify the maximum segment size it is willing to accept. Why does TCP support an option to specify maximum segment size when it also has a window advertisement mechanism?

13.3 Under what conditions of delay, bandwidth, load, and packet loss will TCP retransmit significant volumes of data unnecessarily?

13.4 Lost TCP acknowledgements do not necessarily force retransmissions. Explain why.

13.5 Experiment with local machines to determine how TCP handles machine restart. Establish a connection (e.g., a remote login) and leave it idle. Wait for the destination machine to crash and restart, and then force the local machine to send a TCP segment (e.g., by typing characters to the remote login).

13.6 Imagine an implementation of TCP that discards segments that arrive out of order, even if they fall in the current window. That is, the imagined version only accepts segments that extend the byte stream it has already received. Does it work? How does it compare to a standard TCP implementation?

13.7 Consider computation of a TCP checksum. Assume that although the checksum field in the segment has *not* been set to zero, the result of computing the checksum *is* zero. What can you conclude?

13.8 What are the arguments for and against automatically closing idle connections?

13.9 If two application programs use TCP to send data but only send one character per segment (e.g., by using the PUSH operation), what is the maximum percent of the network bandwidth they will have for their data?

13.10 Suppose an implementation of TCP uses initial sequence number *1* when it creates a connection. Explain how a system crash and restart can confuse a remote system into believing that the old connection remained open.

13.11 Look at the round-trip time estimation algorithm suggested in the ISO TP-4 protocol specification and compare it to the TCP algorithm discussed in this chapter. Which would you prefer to use?

13.12 Find out how implementations of TCP must solve the *overlapping segment problem*. The problem arises because the receiver must accept only one copy of all bytes from the data stream even if the sender transmits two segments that partially overlap one another (e.g., the first segment carries bytes 100 through 200 and the second carries bytes 150 through 250).

13.13 Trace the TCP finite state machine transitions for two sites that execute a passive and active open and step through the three-way handshake.

13.14 Read the TCP specification to find out the exact conditions under which TCP can make the transition from *FIN WAIT-1* to *TIMED WAIT*.

13.15 Trace the TCP state transitions for two machines that agree to close a connection gracefully.

13.16 Assume TCP is sending segments using a maximum window size (64 Kbytes) on a channel that has infinite bandwidth and an average roundtrip time of 20 milliseconds. What is the maximum throughput? How does throughput change if the roundtrip time increases to 40 milliseconds (while bandwidth remains infinite)?

13.17 As the previous exercise illustrates, higher throughput can be achieved with larger windows. One of the drawbacks of the TCP segment format is the size of the field devoted to window advertisement. How can TCP be extended to allow larger windows without changing the segment format?

13.18 Can you derive an equation that expresses the maximum possible TCP throughput as a function of the network bandwidth, the network delay, and the time to process a segment and generate an acknowledgement. Hint: consider the previous exercise.

13.19 Describe (abnormal) circumstances that can leave one end of a connection in state *FIN WAIT-2* indefinitely (hint: think of datagram loss and system crashes).

13.20 Show that when a router implements RED, the probability a packet will be discarded from a particular TCP connection is proportional to the percentage of traffic that the connection generates.

14

Routing: Cores, Peers, And Algorithms

14.1 Introduction

Previous chapters concentrate on the network level services TCP/IP offers and the details of the protocols in hosts and routers that provide those services. In the discussion, we assumed that routers always contain correct routes, and we observed that routers can ask directly connected hosts to change routes with the ICMP redirect mechanism.

This chapter considers two broad questions: ''What values should routing tables contain?'' and ''How can those values be obtained?'' To answer the first question, we will consider the relationship between internet architecture and routing. In particular, we will discuss internets structured around a backbone and those composed of multiple peer networks, and consider the consequences for routing. While many of our examples are drawn from the global Internet, the ideas apply equally well to smaller corporate internets. To answer the second question, we will consider the two basic types of route propagation algorithms and see how each supplies routing information automatically.

We begin by discussing routing in general. Later sections concentrate on internet architecture and describe the algorithms routers use to exchange routing information. Chapters 15 and 16 continue to expand our discussion of routing. They explore protocols that routers owned by two independent administrative groups use to exchange information, and protocols that a single group uses among all its routers.

14.2 The Origin Of Routing Tables

Recall from Chapter 3 that IP routers provide active interconnections among networks. Each router attaches to two or more physical networks and forwards IP datagrams among them, accepting datagrams that arrive over one network interface, and routing them out over another interface. Except for destinations on directly attached networks, hosts pass all IP traffic to routers which forward datagrams on toward their final destinations. A datagram travels from router to router until it reaches a router that attaches directly to the same network as the final destination. Thus, the router system forms the architectural basis of an internet and handles all traffic except for direct delivery from one host to another.

Chapter 8 describes the IP routing algorithm that hosts and routers follow to forward datagrams, and shows how the algorithm uses a table to make routing decisions. Each entry in the routing table specifies the network portion of a destination address and gives the address of the next machine along a path used to reach that network. Like hosts, routers directly deliver datagrams to destinations on networks to which the router attaches.

Although we have seen the basics of datagram forwarding, we have not said how hosts or routers obtain the information for their routing tables. The issue has two aspects: *what* values should be placed in the tables, and *how* routers obtain those values. Both choices depend on the architectural complexity and size of the internet as well as administrative policies.

In general, establishing routes involves initialization and update. Each router must establish an initial set of routes when it starts, and it must update the table as routes change (e.g., when a network interface fails). Initialization depends on the operating system. In some systems, the router reads an initial routing table from secondary storage at startup, keeping it resident in main memory. In others, the operating system begins with an empty table which must be filled in by executing explicit commands (e.g., commands found in a startup command script). Finally, some operating systems start by deducing an initial set of routes from the set of addresses for the local networks to which the machine attaches and contacting a neighboring machine to ask for additional routes.

Once an initial routing table has been built, a router must accommodate changes in routes. In small, slowly changing internets, managers can establish and modify routes by hand. In large, rapidly changing environments, however, manual update is impossibly slow and prone to human errors. Automated methods are needed.

Before we can understand the automatic routing table update protocols used in IP routers, we need to review several underlying ideas. The next sections do so, providing the necessary conceptual foundation for routing. Later sections discuss internet architecture and the protocols routers use to exchange routing information.

14.3 Routing With Partial Information

The principal difference between routers and typical hosts is that hosts usually know little about the structure of the internet to which they connect. Hosts do not have complete knowledge of all possible destination addresses, or even of all possible destination networks. In fact, many hosts have only two routes in their routing table: a route for the local network and a default route for a nearby router. The host sends all nonlocal datagrams to the local router for delivery. The point is that:

> *A host can route datagrams successfully even if it only has partial routing information because it can rely on a router.*

Can routers also route datagrams with only partial information? Yes, but only under certain circumstances. To understand the criteria, imagine an internet to be a foreign country crisscrossed with dirt roads that have directional signs posted at intersections. Imagine that you have no map, cannot ask directions because you cannot speak the local language, have no ideas about visible landmarks, but you need to travel to a village named *Sussex*. You leave on your journey, following the only road out of town and begin to look for directional signs. The first sign reads:

Norfolk to the left; Hammond to the right; others straight ahead.†

Because the destination you seek is not listed explicitly, you continue straight ahead. In routing jargon, we say you follow a *default route*. After several more signs, you finally find one that reads:

Essex to the left; Sussex to the right; others straight ahead.

You turn to the right, follow several more signs, and emerge on a road that leads to Sussex.

Our imagined travel is analogous to a datagram traversing an internet, and the road signs are analogous to routing tables in routers along the path. Without a map or other navigational aids, travel is completely dependent on road signs, just as datagram routing in an internet depends entirely on routing tables. Clearly, it is possible to navigate even though each road sign contains only partial information.

A central question concerns correctness. As a traveler, you might ask, "How can I be sure that following the signs will lead to my final destination?" You also might ask, "How can I be sure that following the signs will lead me to my destination along a shortest path?" These questions may seem especially troublesome if you pass many signs without finding your destination listed explicitly. Of course, the answers depend on the topology of the road system and the contents of the signs, but the fundamental idea is that when taken as a whole, the information on the signs should be both consistent and complete. Looking at this another way, we see that it is not necessary for each intersection to have a sign for every destination. The signs can list default paths as

†Fortunately, signs are printed in a language you can read.

long as all explicit signs point along a shortest path, and the turns for shortest paths to all destinations are marked. A few examples will explain some ways that consistency can be achieved.

At one extreme, consider a simple star-shaped topology of roads in which each village has exactly one road leading to it, and all those roads meet at a central point. To guarantee consistency, the sign at the central intersection must contain information about all possible destinations. At the other extreme, imagine an arbitrary set of roads with signs at all intersections listing all possible destinations. To guarantee consistency, it must be true that at any intersection if the sign for destination *D* points to road *R*, no road other than *R* leads to a shorter path to *D*.

Neither of these architectural extremes works well for an internet router system. On one hand, the central intersection approach fails because no machine is fast enough to serve as a central switch through which all traffic passes. On the other hand, having information about all possible destinations in all routers is impractical because it requires propagating large volumes of information whenever a change occurs or whenever administrators need to check consistency. Thus, we seek a solution that allows groups to manage local routers autonomously, adding new network interconnections and routes without changing distant routers.

To help explain some of the architecture described later, consider a third topology in which half the cities lie in the eastern part of the country and half lie in the western part. Suppose a single bridge spans the river that separates east from west. Assume that people living in the eastern part do not like westerners, so they are willing to allow road signs that list destinations in the east but none in the west. Assume that people living in the west do the opposite. Routing will be consistent if every road sign in the east lists all eastern destinations explicitly and points the default path to the bridge, while every road sign in the west lists all western destinations explicitly and points the default path to the bridge.

14.4 Original Internet Architecture And Cores

Much of our knowledge of routing and route propagation protocols has been derived from experience with the global Internet. When TCP/IP was first developed, participating research sites were connected to the ARPANET, which served as the Internet backbone. During initial experiments, each site managed routing tables and installed routes to other destinations by hand. As the fledgling Internet began to grow, it became apparent that manual maintenance of routes was impractical; automated mechanisms were needed.

The Internet designers selected a router architecture that consisted of a small, central set of routers that kept complete information about all possible destinations, and a larger set of outlying routers that kept partial information. In terms of our analogy, it is like designating a small set of centrally located intersections to have signs that list all destinations, and allowing the outlying intersections to list only local destinations. As long as the default route at each outlying intersection points to one of the central intersections, travelers will eventually reach their destination. The advantage of using partial information in outlying routers is that it permits local administrators to manage local structural changes without affecting other parts of the Internet. The disadvantage is that it introduces the potential for inconsistency. In the worst case, an error in an outlying router can make distant routes unreachable.

We can summarize these ideas:

> *The routing table in a given router contains partial information about possible destinations. Routing that uses partial information allows sites autonomy in making local routing changes, but introduces the possibility of inconsistencies that may make some destinations unreachable from some sources.*

Inconsistencies among routing tables usually arise from errors in the algorithms that compute routing tables, incorrect data supplied to those algorithms, or from errors that occur while transmitting the results to other routers. Protocol designers look for ways to limit the impact of errors, with the objective being to keep all routes consistent at all times. If routes become inconsistent for some reason, the routing protocols should be robust enough to detect and correct the errors quickly. Most important, the protocols should be designed to constrain the effect of errors.

14.5 Core Routers

Loosely speaking, early Internet routers could be partitioned into two groups, a small set of *core routers* controlled by the Internet Network Operations Center (INOC), and a larger set of *noncore routers*† controlled by individual groups. The core system was designed to provide reliable, consistent, authoritative routes for all possible destinations; it was the glue that held the Internet together and made universal interconnection possible. By fiat, each site assigned an Internet network address had to arrange to advertise that address to the core system. The core routers communicated among themselves, so they could guarantee that the information they shared was consistent. Because a central authority monitored and controlled the core routers, they were highly reliable.

To fully understand the core router system, it is necessary to recall that the Internet evolved with a wide-area network, the ARPANET, already in place. When the Internet experiments began, designers thought of the ARPANET as a main backbone on which to build. Thus, a large part of the motivation for the core router system came from the desire to connect local networks to the ARPANET. Figure 14.1 illustrates the idea.

†The terms *stub* and *nonrouting* have also been used in place of *noncore*.

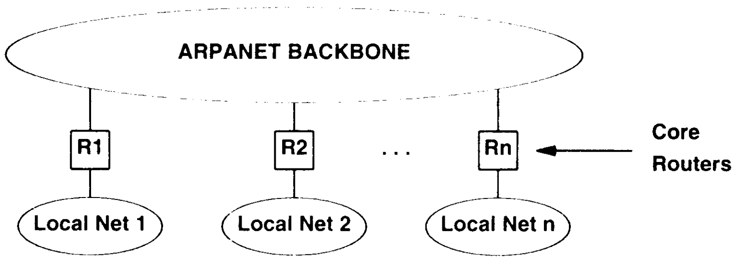


Figure 14.1 The early Internet core router system viewed as a set of routers that connect local area networks to the ARPANET. Hosts on the local networks pass all nonlocal traffic to the closest core router.

To understand why such an architecture does not lend itself to routing with partial information, suppose that a large internet consists entirely of local area networks, each attached to a backbone network through a router. Also imagine that some of the routers rely on default routes. Now consider the path a datagram follows. At the source site, the local router checks to see if it has an explicit route to the destination and, if not, sends the datagram along the path specified by its default route. All datagrams for which the router has no route follow the same default path regardless of their ultimate destination. The next router along the path diverts datagrams for which it has an explicit route, and sends the rest along its default route. To ensure global consistency, the chain of default routes must reach every router in a giant cycle as Figure 14.2 shows. Thus, the architecture requires all local sites to coordinate their default routes. In addition, depending on default routes can be inefficient even when it is consistent. As Figure 14.2 shows, in the worst case a datagram will pass through all *n* routers as it travels from source to destination instead of going directly across the backbone.

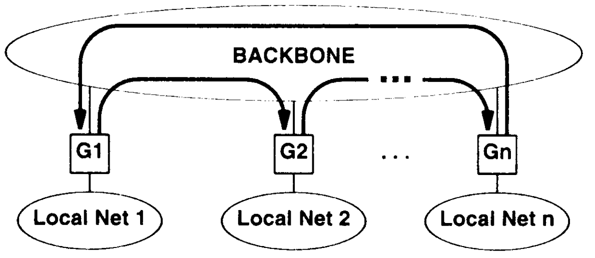


Figure 14.2 A set of routers connected to a backbone network with default routes shown. Routing is inefficient even though it is consistent.

To avoid the inefficiencies default routes cause, Internet designers arranged for all core routers to exchange routing information so that each would have complete information about optimal routes to all possible destinations. Because each core router knew routes to all possible destinations, it did not need a default route. If the destination address on a datagram was not in a core router's routing table, the router would generate an ICMP destination unreachable message and drop the datagram. In essence, the core design avoided inefficiency by eliminating default routes.

Figure 14.3 depicts the conceptual basis of a core routing architecture. The figure shows a central core system consisting of one or more core routers, and a set of outlying routers at local sites. Outlying routers keep information about local destinations and use a default route that sends datagrams destined for other sites to the core.

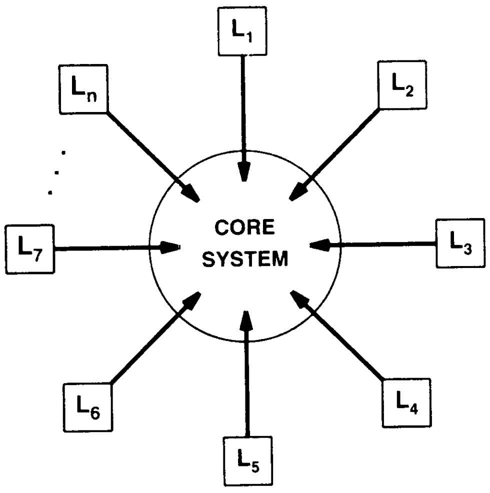


Figure 14.3 The routing architecture of a simplistic core system showing default routes. Core routers do not use default routes; outlying routers, labeled L_i, each have a default route that points to the core.

Although the simplistic core architecture illustrated in Figure 14.3 is easy to understand, it became impractical for three reasons. First, the Internet outgrew a single, centrally managed long-haul backbone. The topology became complex and the protocols needed to maintain consistency among core routers became nontrivial. Second, not every site could have a core router connected to the backbone, so additional routing structure and protocols were needed. Third, because core routers all interacted to ensure consistent routing information, the core architecture did not scale to arbitrary size. We will return to this last problem in Chapter 15 after we examine the protocols that the core system used to exchange routing information.

14.6 Beyond The Core Architecture To Peer Backbones

The introduction of the NSFNET backbone into the Internet added new complexity to the routing structure. From the core system point of view, the connection to NSFNET was initially no different than the connection to any other site. NSFNET attached to the ARPANET backbone through a single router in Pittsburgh. The core had explicit routes to all destinations in NSFNET. Routers inside NSFNET knew about local destinations and used a default route to send all non-NSFNET traffic to the core via the Pittsburgh router.

As NSFNET grew to become a major part of the Internet, it became apparent that the core routing architecture would not suffice. The most important conceptual change occurred when multiple connections were added between the ARPANET and NSFNET backbones. We say that the two became *peer backbone networks* or simply *peers*. Figure 14.4 illustrates the resulting peer topology.

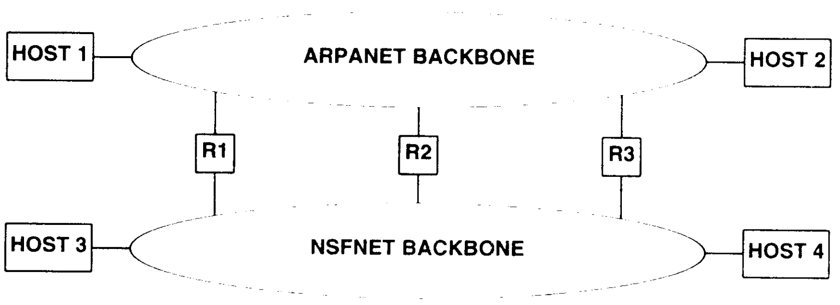


Figure 14.4 An example of peer backbones interconnected through multiple routers. The diagram illustrates the architecture of the Internet in 1989. In later generations, parallel backbones were each owned by an ISP.

To understand the difficulties of IP routing among peer backbones, consider routes from host *3* to host *2* in Figure 14.4. Assume for the moment that the figure shows geographic orientation, so host *3* is on the West Coast attached to the NSFNET backbone while host *2* is on the East Coast attached to the ARPANET backbone. When establishing routes between hosts *3* and *2*, the managers must decide whether to (a) route the traffic from host *3* through the West Coast router, *R1*, and then across the ARPANET backbone, or (b) route the traffic from host *3* across the NSFNET backbone, through the Midwest router, *R2*, and then across the ARPANET backbone to host *2*, or (c) route the traffic across the NSFNET backbone, through the East Coast router, *R3*, and then to host *2*. A more circuitous route is possible as well: traffic could flow from host *3* through the West Coast router, across the ARPANET backbone to the Midwest router, back onto the NSFNET backbone to the East Coast router, and finally across the

ARPANET backbone to host *2*. Such a route may or may not be advisable, depending on the policies for network use and the capacity of various routers and backbones.

For most peer backbone configurations, traffic between a pair of geographically close hosts should take a shortest path, independent of the routes chosen for cross-country traffic. For example, traffic from host *3* to host *1* should flow through the West Coast router because it minimizes distance on both backbones.

All these statements sound simple enough, but they are complex to implement for two reasons. First, although the standard IP routing algorithm uses the network portion of an IP address to choose a route, optimal routing in a peer backbone architecture requires individual routes for individual hosts. For our example above, the routing table in host *3* needs different routes for host *1* and host *2*, even though both hosts *1* and *2* attach to the ARPANET backbone. Second, managers of the two backbones must agree to keep routes consistent among all routers or *routing loops* can develop (a routing loop occurs when routes in a set of routers point in a circle).

It is important to distinguish network topology from routing architecture. It is possible, for example, to have a single core system that spans multiple backbone networks. The core machines can be programmed to hide the underlying architectural details and to compute shortest routes among themselves. It is not possible, however, to partition the core system into subsets that each keep partial information without losing functionality. Figure 14.5 illustrates the problem.

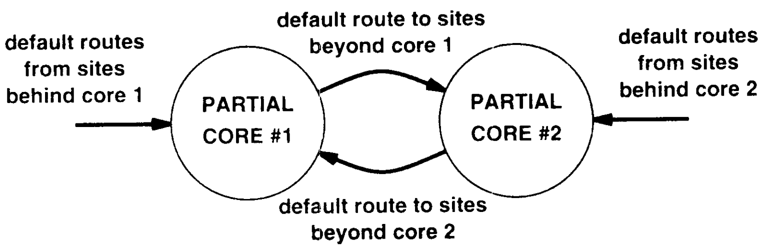


Figure 14.5 An attempt to partition a core routing architecture into two sets of routers that keep partial information and use default routes. Such an architecture results in a routing loop for datagrams that have an illegal (nonexistent) destination.

As the figure shows, outlying routers have default routes to one side of the partitioned core. Each side of the partition has information about destinations on its side of the world and a default route for information on the other side of the world. In such an architecture, any datagram sent to an illegal address will cycle between the two partitions in a routing loop until its time to live counter reaches zero.

We can summarize as follows:

A core routing architecture assumes a centralized set of routers serves as the repository of information about all possible destinations in an internet. Core systems work best for internets that have a single, centrally managed backbone. Expanding the topology to multiple backbones makes routing complex; attempting to partition the core architecture so that all routers use default routes introduces potential routing loops.

14.7 Automatic Route Propagation

We said that the original Internet core system avoided default routes because it propagated complete information about all possible destinations to every core router. Many corporate internets now use a similar scheme — routers in the corporation run programs that communicate routing information. The next sections discuss two basic types of algorithms that compute and propagate routing information, and use the original core routing protocol to illustrate one of the algorithms. A later section describes a protocol that uses the other type of algorithm.

It may seem that automatic route propagation mechanisms are not needed, especially on small internets. However, internets are not static. Connections fail and are later replaced. Networks can become overloaded at one moment and underutilized at the next. The purpose of routing propagation mechanisms is not merely to find a set of routes, but to continually update the information. Humans simply cannot respond to changes fast enough; computer programs must be used. Thus, when we think about route propagation, it is important to consider the dynamic behavior of protocols and algorithms.

14.8 Distance Vector (Bellman-Ford) Routing

The term *distance-vector*† refers to a class of algorithms routers use to propagate routing information. The idea behind distance-vector algorithms is quite simple. The router keeps a list of all known routes in a table. When it boots, a router initializes its routing table to contain an entry for each directly connected network. Each entry in the table identifies a destination network and gives the distance to that network, usually measured in hops (which will be defined more precisely later). For example, Figure 14.6 shows the initial contents of the table on a router that attaches to two networks.

†The terms *vector-distance*, *Ford-Fulkerson*, *Bellman-Ford*, and *Bellman* are synonymous with *distance-vector*; the last two are taken from the names of researchers who published the idea.

Destination	Distance	Route
Net 1	0	direct
Net 2	0	direct

Figure 14.6 An initial distance-vector routing table with an entry for each directly connected network. Each entry contains the IP address of a network and an integer distance to that network.

Periodically, each router sends a copy of its routing table to any other router it can reach directly. When a report arrives at router *K* from router *J*, *K* examines the set of destinations reported and the distance to each. If *J* knows a shorter way to reach a destination, or if *J* lists a destination that *K* does not have in its table, or if *K* currently routes to a destination through *J* and *J*'s distance to that destination changes, *K* replaces its table entry. For example, Figure 14.7 shows an existing table in a router, *K*, and an update message from another router, *J*.

Destination	Distance	Route
Net 1	0	direct
Net 2	0	direct
Net 4	8	Router L
Net 17	5	Router M
Net 24	6	Router J
Net 30	2	Router Q
Net 42	2	Router J

(a)

	Destination	Distance
	Net 1	2
→	Net 4	3
	Net 17	6
→	Net 21	4
	Net 24	5
	Net 30	10
→	Net 42	3

(b)

Figure 14.7 (a) An existing route table for a router *K*, and (b) an incoming routing update message from router *J*. The marked entries will be used to update existing entries or add new entries to *K*'s table.

Note that if *J* reports distance *N*, an updated entry in *K* will have distance *N+1* (the distance to reach the destination from *J* plus the distance to reach *J*). Of course, the routing table entries contain a third column that specifies a next hop. The next hop entry in each initial route is marked *direct delivery*. When router *K* adds or updates an entry in response to a message from router *J*, it assigns router *J* as the next hop for that entry.

The term *distance-vector* comes from the information sent in the periodic messages. A message contains a list of pairs *(V, D)*, where *V* identifies a destination (called the *vector*), and *D* is the distance to that destination. Note that distance-vector algorithms report routes in the first person (i.e., we think of a router advertising, "I can reach destination *V* at distance *D*''). In such a design, all routers must participate in the distance-vector exchange for the routes to be efficient and consistent.

Although distance-vector algorithms are easy to implement, they have disadvantages. In a completely static environment, distance-vector algorithms propagate routes to all destinations. When routes change rapidly, however, the computations may not stabilize. When a route changes (i.e, a new connection appears or an old one fails), the information propagates slowly from one router to another. Meanwhile, some routers may have incorrect routing information.

For now, we will examine a simple protocol that uses the distance-vector algorithm without discussing all the shortcomings. Chapter 16 completes the discussion by showing another distance-vector protocol, the problems that can arise, and the heuristics used to solve the most serious of them.

14.9 Gateway-To-Gateway Protocol (GGP)

The original core routers used a distance-vector protocol known as the *Gateway-to-Gateway Protocol*† *(GGP)* to exchange routing information. Although GGP only handled classful routes and is no longer part of the TCP/IP standards‡, it does provide a concrete example of distance-vector routing. GGP was designed to travel in IP datagrams similar to UDP datagrams or TCP segments. Each GGP message has a fixed format header that identifies the message type and the format of the remaining fields. Because only core routers participated in GGP, and because core routers were controlled by a central authority, other routers could not interfere with the exchange.

The original core system was arranged to permit new core routers to be added without modifying existing routers. When a new router was added to the core system, it was assigned one or more core *neighbors* with which it communicated. The neighbors, members of the core, already propagated routing information among themselves. Thus, the new router only needed to inform its neighbors about networks it could reach; they updated their routing tables and propagated this new information further.

GGP is a true distance-vector protocol. The information routers exchange with GGP consists of a set of pairs, *(N, D)*, where *N* is an IP network address, and *D* is a distance measured in *hops*. We say that a router using GGP *advertises* the networks it can reach and its cost for reaching them.

GGP measures distance in *router hops*, where a router is defined to be zero hops from directly connected networks, one hop from networks that are reachable through one other router, and so on. Thus, the *number of hops* or the *hop count* along a path from a given source to a given destination refers to the number of routers that a datagram encounters along that path. It should be obvious that using hop counts to calculate shortest paths does not always produce desirable results. For example, a path with hop count *3* that crosses three LANs may be substantially faster than a path with hop count *2* that crosses two slow speed serial lines. Many routers use artificially high hop counts for routes across slow networks.

†Recall that although vendors adopted the term *IP router*, scientists originally used the term *IP gateway*.

‡The IETF has declared GGP *historic*, which means that it is no longer recommended for use with TCP/IP.

14.10 Distance Factoring

Like most routing protocols, GGP uses multiple message *types*, each with its own format and purpose. A field in the message header contains a code that identifies the specific message type; a receiver uses the code to decide how to process the message. For example, before two routers can exchange routing information, they must establish communication, and some message types are used for that purpose. The most fundamental message type in GGP is also fundamental to any distance-vector protocol: a routing update which is used to exchange routing information.

Conceptually, a routing update contains a list of pairs, where each entry contains an IP network address and the distance to that network. In practice, however, many routing protocols rearrange the information to keep messages small. In particular, observe that few architectures consist of a linear arrangement of networks and routers. Instead, most are hierarchical, with multiple routers attached to each network. Consequently, most distance values in an update are small numbers, and the same values tend to be repeated frequently. To reduce message size, routing protocols often use a technique that was pioneered in GGP. Known as *distance factoring*, the technique avoids sending copies of the same distance number. Instead, the list of pairs is sorted by distance, each distance value is represented once, and the networks reachable at that distance follow. The next chapter shows how other routing protocols factor information.

14.11 Reliability And Routing Protocols

Most routing protocols use connectionless transport. For example, GGP encapsulates messages directly in IP datagrams; modern routing protocols usually encapsulate in UDP†. Both IP and UDP offer the same semantics: messages can be lost, delayed, duplicated, corrupted, or delivered out of order. Thus, a routing protocol that uses them must compensate for failures.

Routing protocols use several techniques to handle delivery problems. Checksums are used to handle corruption. Loss is either handled by *soft state*‡ or through acknowledgement and retransmission. For example, GGP uses an extended acknowledgement scheme in which a receiver can return either a positive or negative acknowledgement.

To handle delivery out of order and the corresponding reply that occurs when an old message arrives, routing protocols often used *sequence numbers*. In GGP, for example, each side chooses an initial number to use for sequencing when communication begins. The other side must then acknowledge the sequence number. After the initial exchange, each message contains the next number in the sequence, which allows the receiver to know whether the message arrived in order. In a later chapter, we will see an example of a routing protocol that uses soft state information.

†There are exceptions — the next chapter discusses a protocol that uses TCP.

‡Recall that soft state relies on timeouts to remove old information rather than waiting for a message from the source.

14.12 Link-State (SPF) Routing

The main disadvantage of the distance-vector algorithm is that it does not scale well. Besides the problem of slow response to change mentioned earlier, the algorithm requires the exchange of large messages. Because each routing update contains an entry for every possible network, message size is proportional to the total number of networks in an internet. Furthermore, because a distance-vector protocol requires every router to participate, the volume of information exchanged can be enormous.

The primary alternative to distance-vector algorithms is a class of algorithms known as *link state*, *link status*, or *Shortest Path First*† (*SPF*). The SPF algorithm requires each participating router to have complete topology information. The easiest way to think of the topology information is to imagine that every router has a map that shows all other routers and the networks to which they connect. In abstract terms, the routers correspond to nodes in a graph and networks that connect routers correspond to edges. There is an edge (link) between two nodes if and only if the corresponding routers can communicate directly.

Instead of sending messages that contain lists of destinations, a router participating in an SPF algorithm performs two tasks. First, it actively tests the status of all neighbor routers. In terms of the graph, two routers are neighbors if they share a link; in network terms, two neighbors connect to a common network. Second, it periodically propagates the link status information to all other routers.

To test the status of a directly connected neighbor, a router periodically exchanges short messages that ask whether the neighbor is alive and reachable. If the neighbor replies, the link between them is said to be *up*. Otherwise, the link is said to be *down*‡. To inform all other routers, each router periodically broadcasts a message that lists the status (state) of each of its links. A status message does not specify routes — it simply reports whether communication is possible between pairs of routers. Protocol software in the routers arranges to deliver a copy of each link status message to all participating routers (if the underlying networks do not support broadcast, delivery is done by forwarding individual copies of the message point-to-point).

Whenever a link status message arrives, a router uses the information to update its map of the internet, by marking links up or down. Whenever link status changes, the router recomputes routes by applying the well-known *Dijkstra shortest path algorithm* to the resulting graph. Dijkstra's algorithm computes the shortest paths to all destinations from a single source.

One of the chief advantages of SPF algorithms is that each router computes routes independently using the same original status data; they do not depend on the computation of intermediate machines. Because link status messages propagate unchanged, it is easy to debug problems. Because routers perform the route computation locally, it is guaranteed to converge. Finally, because link status messages only carry information about the direct connections from a single router, the size does not depend on the number of networks in the internet. Thus, SPF algorithms scale better than distance-vector algorithms.

†The name "shortest path first" is a misnomer because all routing algorithms seek shortest paths.

‡In practice, to prevent oscillations between the up and down states, most protocols use a *k-out-of-n rule* to test liveness, meaning that the link remains up until a significant percentage of requests have no reply, and then it remains down until a significant percentage of messages receive a reply.

14.13 Summary

To ensure that all networks remain reachable with high reliability, an internet must provide globally consistent routing. Hosts and most routers contain only partial routing information; they depend on default routes to send datagrams to distant destinations. Originally, the global Internet solved the routing problem by using a core router architecture in which a set of core routers each contained complete information about all networks. Routers in the original Internet core system exchanged routing information periodically, meaning that once a single core router learned about a route, all core routers learned about it. To prevent routing loops, core routers were forbidden from using default routes.

A single, centrally managed core system works well for an internet architecture built on a single backbone network. However, a core architecture does not suffice for an internet that consists of a set of separately managed peer backbones that interconnect at multiple places.

When routers exchange routing information they use one of two basic algorithms, distance-vector or SPF. A distance-vector protocol, GGP, was originally used to propagate routing update information throughout the Internet core.

The chief disadvantage of distance-vector algorithms is that they perform a distributed shortest path computation that may not converge if the status of network connections changes continually. Another disadvantage is that routing update messages grow large as the number of networks increases.

The use of SPF routing predates the Internet. One of the earliest examples of an SPF protocol comes from the ARPANET, which used a routing protocol internally to establish and maintain routes among packet switches. The ARPANET algorithm was used for a decade.

FOR FURTHER STUDY

The definition of the core router system and GGP protocol in this chapter comes from Hinden and Sheltzer [RFC 823]. Braden and Postel [RFC 1812] contains further specifications for Internet routers. Almquist [RFC 1716] summarizes later discussions. Braun [RFC 1093] and Rekhter [RFC 1092] discuss routing in the NSFNET backbone. Clark [RFC 1102] and Braun [RFC 1104] both discuss policy-based routing. The next two chapters present protocols used for propagating routing information between separate sites and within a single site.

EXERCISES

14.1 Suppose a router discovers it is about to route an IP datagram back over the same network interface on which the datagram arrived. What should it do? Why?

14.2 After reading RFC 823 and RFC 1812, explain what an Internet core router (i.e., one with complete routing information) should do in the situation described in the previous question.

14.3 How can routers in a core system use default routes to send all illegal datagrams to a specific machine?

14.4 Imagine students experimenting with a router that attaches a local area network to an internet that has a core routing system. The students want to advertise their network to a core router, but if they accidentally advertise zero length routes to arbitrary networks, traffic from the internet will be diverted to their router incorrectly. How can a core protect itself from illegal data while still accepting updates from such "untrusted" routers?

14.5 Which ICMP messages does a router generate?

14.6 Assume a router is using unreliable transport for delivery. How can the router determine whether a designated neighbor is "up" or "down"? (Hint: consult RFC 823 to find out how the original core system solved the problem.)

14.7 Suppose two routers each advertise the same cost, *k*, to reach a given network, *N*. Describe the circumstances under which routing through one of them may take fewer total hops than routing through the other one.

14.8 How does a router know whether an incoming datagram carries a GGP message? An OSPF message?

14.9 Consider the distance-vector update shown in Figure 14.7 carefully. For each item updated in the table, give the reason why the router will perform the update.

14.10 Consider the use of sequence numbers to ensure that two routers do not become confused when datagrams are duplicated, delayed, or delivered out of order. How should initial sequence numbers be selected? Why?

15

Routing: Exterior Gateway Protocols And Autonomous Systems (BGP)

15.1 Introduction

The previous chapter introduces the idea of route propagation and examines one protocol routers use to exchange routing information. This chapter extends our understanding of internet routing architectures. It discusses the concept of autonomous systems, and shows a protocol that a group of networks and routers operating under one administrative authority uses to propagate routing information about its networks to other groups.

15.2 Adding Complexity To The Architectural Model

The original core routing system evolved at a time when the Internet had a single wide area backbone as the previous chapter describes. Consequently, part of the motivation for a core architecture was to provide connections between a network at each site and the backbone. If an internet consists of only a single backbone plus a set of attached local area networks, the core approach propagates all necessary routing information correctly. Because all routers attach to the wide area backbone network, they can exchange all necessary routing information directly. Each router knows the single local network to which it attaches, and propagates that information to the other routers. Each router learns about other destination networks from other routers.

It may seem that it would be possible to extend the core architecture to an arbitrary size internet merely by adding more sites, each with a router connecting to the backbone. Unfortunately, the scheme does not scale — having all routers participate in a single routing protocol only suffices for trivial size internets. There are three reasons. First, even if each site consists of a single network, the scheme cannot accommodate an arbitrary number of sites because each additional router generates routing traffic. If a large set of routers attempt to communicate, the total bandwidth becomes overwhelming. Second, the scheme cannot accommodate multiple routers and networks at a given site because only those routers that connect directly to the backbone network can communicate directly. Third, in a large internet, the networks and routers are not all managed by a single entity, nor are shortest paths always used. Instead, because networks are owned and managed by independent groups, the groups may choose policies that differ. A routing architecture must provide a way for each group to independently control routing and access.

The consequences of limiting router interaction are significant. The idea provides the motivation for much of the routing architecture used in the global Internet, and explains some of the mechanisms we will study. To summarize this important principle:

> *Although it is desirable for routers to exchange routing information, it is impractical for all routers in an arbitrarily large internet to participate in a single routing update protocol.*

15.3 Determining A Practical Limit On Group Size

The above statement leaves many questions open. For example, what size internet is considered "large"? If only a limited set of routers can participate in an exchange of routing information, what happens to routers that are excluded? Do they function correctly? Can a router that is not participating ever forward a datagram to a router that is participating? Can a participating router forward a datagram to a non-participating router?

The answer to the question of size involves understanding the algorithm being used and the capacity of the network that connects the routers as well as the details of the routing protocol. There are two issues: delay and overhead. Delay is easy to understand. For example, consider the maximum delay until all routers are informed about a change when they use a distance-vector protocol. Each router must receive the new information, update its routing table, and then forward the information to its neighbors. In an internet with N routers arranged in a linear topology, N steps are required. Thus, N must be limited to guarantee rapid distribution of information.

The issue of overhead is also easy to understand. Because each router that participates in a routing protocol must send messages, a larger set of participating routers means more routing traffic. Furthermore, if routing messages contain a list of possible destinations, the size of each message grows as the number of routers and networks increase. To ensure that routing traffic remains a small percentage of the total traffic on the underlying networks, the size of routing messages must be limited.

In fact, most network managers do not have sufficient information required to perform detailed analysis of the delay or overhead. Instead, they follow a simple heuristic guideline:

> *It is safe to allow up to a dozen routers to participate in a single routing information protocol across a wide area network; approximately five times as many can safely participate across a set of local area networks.*

Of course, the rule only gives general advice and there are many exceptions. For example, if the underlying networks have especially low delay and high capacity, the number of participating routers can be larger. Similarly, if the underlying networks have unusually low capacity or a high amount of traffic, the number of participating routers must be smaller to avoid overloading the networks with routing traffic.

Because an internet is not static, it can be difficult to estimate how much traffic routing protocols will generate or what percentage of the underlying bandwidth the routing traffic will consume. For example, as the number of hosts on a network grows over time, increases in the traffic generated consume more of the network capacity. In addition, increased traffic can arise from new applications. Therefore, network managers cannot rely solely on the guideline above when choosing a routing architecture. Instead, they usually implement a *traffic monitoring* scheme. In essence, a traffic monitor listens passively to a network and records statistics about the traffic. In particular, a monitor can compute both the network utilization (i.e., percentage of the underlying bandwidth being used) and the percentage of packets carrying routing protocol messages. A manager can observe traffic trends by taking measurements over long periods (e.g., weeks or months), and can use the output to determine whether too many routers are participating in a single routing protocol.

15.4 A Fundamental Idea: Extra Hops

Although the number of routers that participate in a single routing protocol must be limited, doing so has an important consequence because it means that some routers will be outside the group. It might seem that an "outsider" could merely make a member of the group a default. In the early Internet, the core system did indeed function as a central routing mechanism to which noncore routers sent datagrams for delivery. However, researchers learned an important lesson: if a router outside of a group uses a member of the group as a default route, routing will be suboptimal. More important, one does not need a large number of routers or a wide area network — the problem can occur whenever a nonparticipating router uses a participating router for delivery. To see why, consider the example in Figure 15.1.

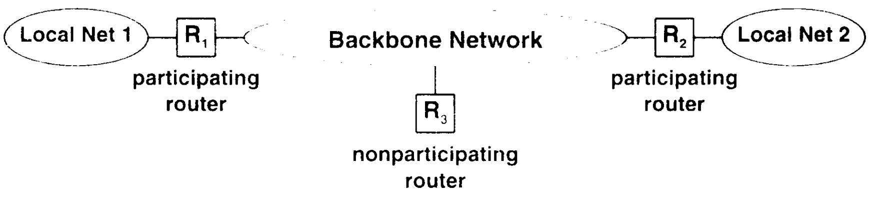


Figure 15.1 An architecture that can cause the extra hop problem. Nonoptimal routing occurs when a nonparticipating router connected to the backbone has a default route to a participating router.

In the figure, routers R_1 and R_2 connect to local area networks *1* and *2*, respectively. Because they participate in a routing protocol, they both know how to reach both networks. Suppose nonparticipating router R_3 chooses one of the participating routers, say R_1, as a default. That is, R_3 sends R_1 all datagrams destined for networks to which it has no direct connection. In particular, R_3 sends datagrams destined for network *2* across the backbone to its chosen participating router, R_1, which must then forward them back across the backbone to router R_2. The optimal route, of course, requires R_3 to transmit datagrams destined for network *2* directly to R_2. Notice that the choice of participating router makes no difference. Only destinations that lie beyond the chosen router have optimal routes; all paths that go through other backbone routers require the datagram to make a second, unnecessary trip across the backbone network. Also notice that the participating routers cannot use ICMP redirect messages to inform R_3 that it has nonoptimal routes because ICMP redirect messages can only be sent to the original source and not to intermediate routers.

We call the routing anomaly illustrated in Figure 15.1 the *extra hop problem*. The problem is insidious because everything appears to work correctly — datagrams do reach their destination. However, because routing is not optimal, the system is extremely inefficient. Each datagram that takes an extra hop consumes resources on the intermediate router as well as twice as much backbone bandwidth as it should. Solving the problem requires us to change our view of architecture:

> *Treating a group of routers that participate in a routing update protocol as a default delivery system can introduce an extra hop for datagram traffic; a mechanism is needed that allows nonparticipating routers to learn routes from participating routers so they can choose optimal routes.*

15.5 Hidden Networks

Before we examine mechanisms that allow a router outside a group to learn routes, we need to understand another aspect of routing: hidden networks (i.e. networks that are concealed from the view of routers in a group). Figure 15.2 shows an example that will illustrate the concept.

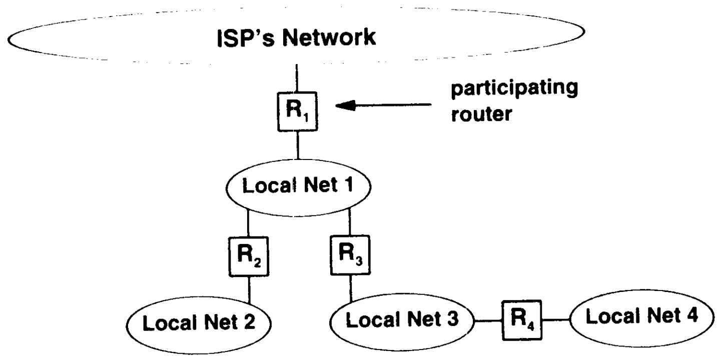


Figure 15.2 An example of multiple networks and routers with a single backbone connection. A mechanism is needed to pass reachability information about additional local networks into the core system.

In the figure, a site has multiple local area networks with multiple routers connecting them. Suppose the site has just installed local network *4* and has obtained an Internet address for it (for now, imagine that the site obtained the address from another ISP). Also assume that the routers R_2, R_3, and R_4 each have correct routes for all four of the site's local networks as well as a default route that passes other traffic to the ISP's router, R_1. Hosts directly attached to local network *4* can communicate with one another, and any computer on that network can route packets out to other Internet sites. However, because router R_1 attaches only to local network *1*, it does not know about local network *4*. We say that, from the viewpoint of the ISP's routing system, local network *4* is *hidden* behind local network *1*. The important point is:

Because an individual organization can have an arbitrarily complex set of networks interconnected by routers, no router from another organization can attach directly to all networks. A mechanism is needed that allows nonparticipating routers to inform the other group about hidden networks.

We now understand a fundamental aspect of routing: information must flow in two directions. Routing information must flow from a group of participating routers to a nonparticipating router, and a nonparticipating router must pass information about hidden networks to the group. Ideally, a single mechanism should solve both problems. Building such a mechanism can be tricky. The subtle issues are responsibility and capability. Exactly where does responsibility for informing the group reside? If we decide that one of the nonparticipating routers should inform the group, which one is capable of doing it? Look again at the example. Router R_4 is the router most closely associated with local network *4*, but it lies *2* hops away from the nearest core router. Thus, R_4 must depend on router R_3 to route packets to network *4*. The point is that R_4 knows about local network *4* but cannot pass datagrams directly from R_1. Router R_3 lies one hop from the core and can guarantee to pass packets, but it does not directly attach to local network *4*. So, it seems incorrect to grant R_3 responsibility for advertising network *4*. Solving this dilemma will require us to introduce a new concept. The next sections discuss the concept and a protocol that implements it.

15.6 Autonomous System Concept

The puzzle over which router should communicate information to the group arises because we have only considered the mechanics of an internet routing architecture and not the administrative issues. Interconnections, like those in the example of Figure 15.2, that arise because an internet has a complex structure, should not be thought of as multiple independent networks connected to an internet. Instead, the architecture should be thought of as a single organization that has multiple networks under its control. Because the networks and routers fall under a single administrative authority, that authority can guarantee that internal routes remain consistent and viable. Furthermore, the administrative authority can choose one of its routers to serve as the machine that will apprise the outside world of networks within the organization. In the example from Figure 15.2, because routers R_2, R_3, and R_4 fall under control of one administrative authority, that authority can arrange to have R_3 advertise networks *2*, *3*, and *4* (R_1 already knows about network *1* because it has a direct connection to it).

For purposes of routing, a group of networks and routers controlled by a single administrative authority is called an *autonomous system* (*AS*). Routers within an autonomous system are free to choose their own mechanisms for discovering, propagating, validating, and checking the consistency of routes. Note that, under this definition, the original Internet core routers formed an autonomous system. Each change in routing protocols within the core autonomous system was made without affecting the routers in other autonomous systems. In the previous chapter, we said that the original Internet core system used GGP to communicate, and a later generation used SPREAD. Eventually, ISPs evolved their own backbone networks that use more recent protocols. The next chapter reviews some of the protocols that autonomous systems use internally to propagate routing information.

15.7 From A Core To Independent Autonomous Systems

Conceptually, the autonomous system idea was a straightforward and natural generalization of the original Internet architecture, depicted by Figure 15.2, with autonomous systems replacing local area networks. Figure 15.3 illustrates the idea.

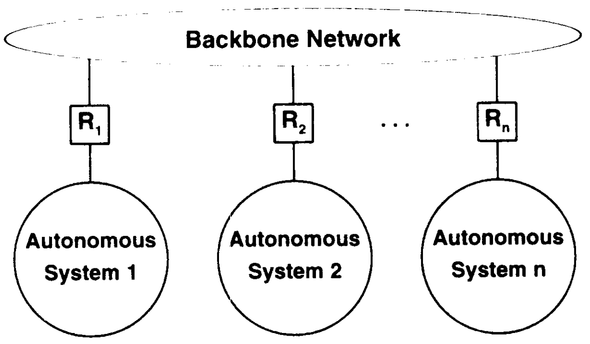


Figure 15.3 Architecture of an internet with autonomous systems at backbone sites. Each autonomous system consists of multiple networks and routers under a single administrative authority.

To make networks that are hidden inside autonomous systems reachable throughout the Internet, each autonomous system must advertise its networks to other autonomous systems. An advertisement can be sent to any autonomous system. In a centralized, core architecture, however, it is crucial that each autonomous system propagate information to one of the routers in the core autonomous system.

It may seem that our definition of an autonomous system is vague, but in practice the boundaries between autonomous systems must be precise to allow automated algorithms to make routing decisions. For example, an autonomous system owned by a corporation may choose not to route packets through an autonomous system owned by another even though they connect directly. To make it possible for automated routing algorithms to distinguish among autonomous systems, each is assigned an *autonomous system number* by the central authority that is charged with assigning all Internet network addresses. When routers in two autonomous systems exchange routing information, the protocol arranges for messages to carry the autonomous system number of the system each router represents.

We can summarize the ideas:

A large TCP/IP internet has additional structure to accommodate administrative boundaries: each collection of networks and routers managed by one administrative authority is considered to be a single autonomous system that is free to choose an internal routing architecture and protocols.

We said that an autonomous system needs to collect information about all its networks and designate one or more routers to pass the information to other autonomous systems. The next sections presents the details of a protocol routers use to advertise network reachability. Later sections return to architectural questions to discuss an important restriction the autonomous system architecture imposes on routing.

15.8 An Exterior Gateway Protocol

Computer scientists use the term *Exterior Gateway Protocol* (*EGP*)† to denote any protocol used to pass routing information between two autonomous systems. Currently a single exterior protocol is used in most TCP/IP internets. Known as the *Border Gateway Protocol* (*BGP*), it has evolved through four (quite different) versions. Each version is numbered, which gives rise to the formal name of the current version: *BGP-4*. Throughout this text, the term *BGP* will refer to *BGP-4*.

When a pair of autonomous systems agree to exchange routing information, each must designate a router‡ that will speak BGP on its behalf; the two routers are said to become *BGP peers* of one another. Because a router speaking BGP must communicate with a peer in another autonomous system, it makes sense to select a machine that is near the "edge" of the autonomous system. Hence, BGP terminology calls the machine a *border gateway* or *border router*. Figure 15.4 illustrates the idea.

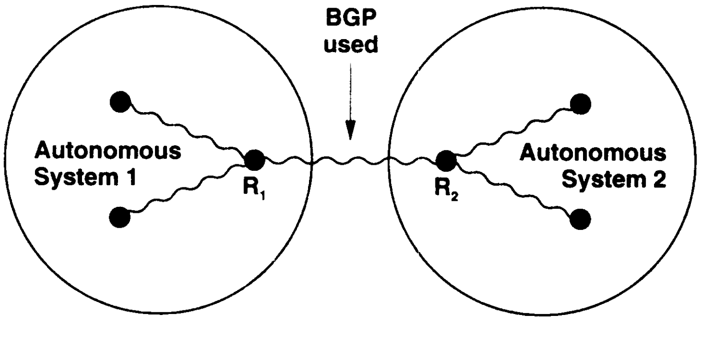


Figure 15.4 Conceptual illustration of two routers, R_1 and R_2, using BGP to advertise networks in their autonomous systems after collecting the information from other routers internally. An organization using BGP usually chooses a router that is close to the outer "edge" of the autonomous system.

In the figure, router R_1 gathers information about networks in autonomous system *1* and reports that information to router R_2 using BGP, while router R_2 reports information from autonomous system *2*.

†Originally, the term *EGP* referred to a specific protocol that was used for communication with the Internet core; the name was coined when the term *gateway* was used instead of *router*.

‡Although the protocol allows an arbitrary computer to be used, most autonomous systems run BGP on a router; all the examples in this text will assume BGP is running on a router.

15.9 BGP Characteristics

BGP is unusual in several ways. Most important, BGP is neither a pure distance-vector protocol nor a pure link state protocol. It can be characterized by the following:

Inter-Autonomous System Communication. Because BGP is designed as an exterior gateway protocol, its primary role is to allow one autonomous system to communicate with another.

Coordination Among Multiple BGP Speakers. If an autonomous system has multiple routers each communicating with a peer in an outside autonomous system, BGP can be used to coordinate among routers in the set to guarantee that they all propagate consistent information.

Propagation Of Reachability Information. BGP allows an autonomous system to advertise destinations that are reachable either in or through it, and to learn such information from another autonomous system.

Next-Hop Paradigm. Like distance-vector routing protocols, BGP supplies *next hop* information for each destination.

Policy Support. Unlike most distance-vector protocols that advertise exactly the routes in the local routing table, BGP can implement policies that the local administrator chooses. In particular, a router running BGP can be configured to distinguish between the set of destinations reachable by computers inside its autonomous system and the set of destinations advertised to other autonomous systems.

Reliable Transport. BGP is unusual among protocols that pass routing information because it assumes reliable transport. Thus, BGP uses TCP for all communication.

Path Information. In addition to specifying destinations that can be reached and a next hop for each, BGP advertisements include path information that allows a receiver to learn a series of autonomous systems along a path to the destination.

Incremental Updates. To conserve network bandwidth, BGP does not pass full information in each update message. Instead, full information is exchanged once, and then successive messages carry incremental changes called *deltas.*

Support For Classless Addressing. BGP supports CIDR addresses. That is, rather than expecting addresses to be self-identifying, the protocol provides a way to send a mask along with each address.

Route Aggregation. BGP conserves network bandwidth by allowing a sender to aggregate route information and send a single entry to represent multiple, related destinations.

Authentication. BGP allows a receiver to authenticate messages (i.e., verify the identity of a sender).

15.10 BGP Functionality And Message Types

BGP peers perform three basic functions. The first function consists of initial peer acquisition and authentication. The two peers establish a TCP connection and perform a message exchange that guarantees both sides have agreed to communicate. The second function forms the primary focus of the protocol — each side sends positive or negative reachability information. That is, a sender can advertise that one or more destinations are reachable by giving a next hop for each, or the sender can declare that one or more previously advertised destinations are no longer reachable. The third function provides ongoing verification that the peers and the network connections between them are functioning correctly.

To handle the three functions described above, BGP defines four basic message types. Figure 15.5 contains a summary.

Type Code	Message Type	Description
1	OPEN	Initialize communication
2	UPDATE	Advertise or withdraw routes
3	NOTIFICATION	Response to an incorrect message
4	KEEPALIVE	Actively test peer connectivity

Figure 15.5 The four basic message types in BGP-4.

15.11 BGP Message Header

Each BGP message begins with a fixed header that identifies the message type. Figure 15.6 illustrates the header format.

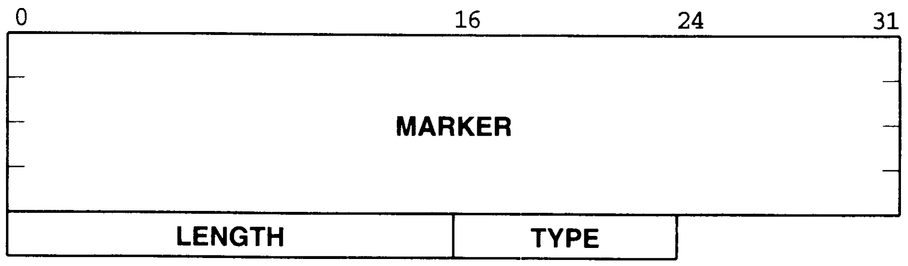


Figure 15.6 The format of the header that precedes every BGP message.

The 16-octet *MARKER* field contains a value that both sides agree to use to mark the beginning of a message. The 2-octet *LENGTH* field specifies the total message length measured in octets. The minimum message size is *19* octets (for a message type that has no data following the header), and the maximum allowable length is *4096* octets. Finally, the 1-octet *TYPE* field contains one of the four values for the message type listed in Figure 15.5.

The *MARKER* may seem unusual. In the initial message, the marker consists of all 1s; if the peers agree to use an authentication mechanism, the marker can contain authentication information. In any case, both sides must agree on the value so it can be used for *synchronization*. To understand why synchronization is necessary, recall that all BGP messages are exchanged across a stream transport (i.e., TCP), which does not identify the boundary between one message and the next. In such an environment, a simple error on either side can have dramatic consequences. In particular, if either the sender or receiver miscounts the octets in a message, a *synchronization error* will occur. More important, because the transport protocol does not specify message boundaries, the transport protocol will not alert the receiver to the error. Thus, to ensure that the sender and receiver remain synchronized, BGP places a well-known sequence at the beginning of each message, and requires a receiver to verify that the value is intact before processing the message.

15.12 BGP OPEN Message

As soon as two BGP peers establish a TCP connection, they each send an *OPEN* message to declare their autonomous system number and establish other operating parameters. In addition to the standard header, an *OPEN* message contains a value for a *hold timer* that is used to specify the maximum number of seconds which may elapse between the receipt of two successive messages. Figure 15.7 illustrates the format.

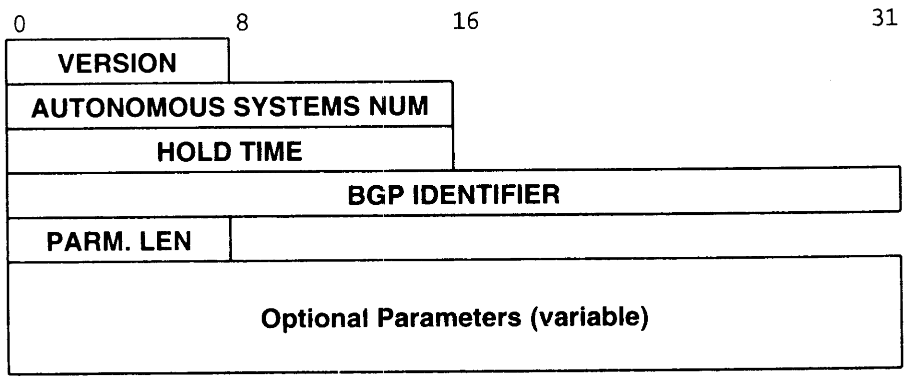


Figure 15.7 The format of the BGP OPEN message that is sent at startup. These octets follow the standard message header.

Most fields are straightforward. The *VERSION* field identifies the protocol version used (this format is for version 4). Recall that each autonomous system is assigned a unique number. Field *AUTONOMOUS SYSTEMS NUM* gives the autonomous system number of the sender's system. The *HOLD TIME* field specifies a maximum time that the receiver should wait for a message from the sender. The receiver is required to implement a timer using this value. The timer is reset each time a message arrives; if the timer expires, the receiver assumes the sender is no longer available (and stops forwarding datagrams along routes learned from the sender).

Field *BGP IDENTIFIER* contains a 32-bit integer value that uniquely identifies the sender. If a machine has multiple peers (e.g., perhaps in multiple autonomous systems), the machine must use the same identifier in all communication. The protocol specifies that the identifier is an IP address. Thus, a router must choose one of its IP addresses to use with all BGP peers.

The last field of an *OPEN* message is optional. If present, field *PARM. LEN* specifies the length measured in octets, and the field labeled *Optional Parameters* contains a list of parameters. It has been labeled *variable* to indicate that the size varies from message to message. When parameters are present, each parameter in the list is preceded by a 2-octet header, with the first octet specifying the type of the parameter, and the second octet specifying the length. If there are no parameters, the value of *PARM. LEN* is zero and the message ends with no further data.

Only one parameter type is defined in the original standard: type *1* is reserved for authentication. The authentication parameter begins with a header that identifies the type of authentication followed by data appropriate for the type. The motivation for making authentication a parameter arises from a desire to allow BGP peers to choose an authentication mechanism without making the choice part of the BGP standard.

When it accepts an incoming *OPEN* message, a machine speaking BGP responds by sending a *KEEPALIVE* message (discussed below). Each side must send an *OPEN* and receive a *KEEPALIVE* message before they can exchange routing information. Thus, a *KEEPALIVE* message functions as the acknowledgement for an *OPEN*.

15.13 BGP UPDATE Message

Once BGP peers have created a TCP connection, sent *OPEN* messages, and acknowledged them, the peers use *UPDATE* messages to advertise new destinations that are reachable or to withdraw previous advertisements when a destination has become unreachable. Figure 15.8 illustrates the format of *UPDATE* messages.

As the figure shows, each *UPDATE* message is divided into two parts: the first lists previously advertised destinations that are being withdrawn, and the second specifies new destinations being advertised. As usual, fields labeled *variable* do not have a fixed size; if the information is not needed for a particular *UPDATE*, the field can be omitted from the message. Field *WITHDRAWN LEN* is a 2-octet field that specifies the size of the *Withdrawn Destinations* field that follows. If no destinations are being withdrawn, *WITHDRAWN LEN* contains zero. Similarly, the *PATH LEN* field specifies the size of the *Path Attributes* that are associated with new destinations being advertised. If there are no new destinations, the *PATH LEN* field contains zero.

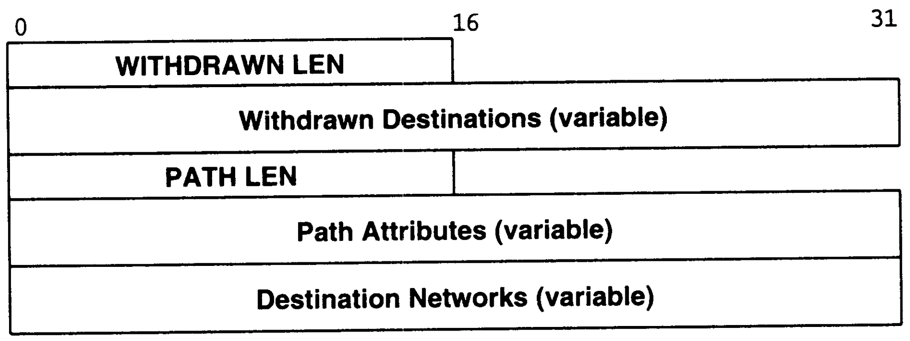

0
16
31
WITHDRAWN LEN
Withdrawn Destinations (variable)
PATH LEN
Path Attributes (variable)
Destination Networks (variable)

Figure 15.8 BGP UPDATE message format in which variable size areas of the message may be omitted. These octets follow the standard message header.

15.14 Compressed Mask-Address Pairs

Both the *Withdrawn Destinations* and the *Destination Networks* fields contain a list of IP network addresses. To accommodate classless addressing, BGP must send an address mask with each IP address. Instead of sending an address and a mask as separate 32-bit quantities, however, BGP uses a compressed representation to reduce message size. Figure 15.9 illustrates the format:

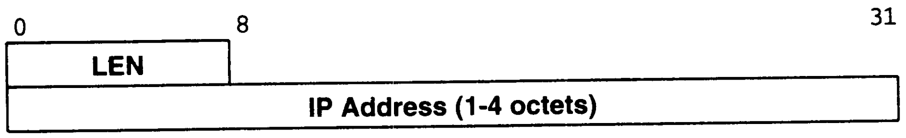


Figure 15.9 The compressed format BGP uses to store a destination address and the associated mask.

The figure shows that BGP does not actually send a bit mask. Instead, it encodes information about the mask into a single octet that precedes each address. The mask octet contains a binary integer that specifies the number of bits in the mask (mask bits are assumed to be contiguous). The address that follows the mask octet is also compressed — only those octets covered by the mask are included. Thus, only one address octet follows a mask value of *8* or less, two follow a mask value of *9* to *16*, three follow a mask value of *17* to *24*, and four follow a mask value of *25* to *32*. Interestingly, the standard also allows a mask octet to contain zero (in which case no address octets follow it). A zero length is useful because it corresponds to a default route.

15.15 BGP Path Attributes

We said that BGP is not a pure distance-vector protocol because it advertises more than a next hop. The additional information is contained in the *Path Attributes* field of an update message. A sender can use the path attributes to specify: a next hop for the advertised destinations, a list of autonomous systems along the path to the destinations, and whether the path information was learned from another autonomous system or derived from within the sender's autonomous system.

It is important to note that the path attributes are factored to reduce the size of the UPDATE message, meaning that the attributes apply to all destinations advertised in the message. Thus, if different attributes apply to some destinations, they must be advertised in a separate UPDATE message.

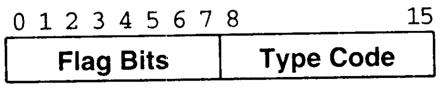
Path attributes are important in BGP for three reasons. First, path information allows a receiver to check for routing loops. The sender can specify an exact path through all autonomous systems to the destination. If any autonomous system appears more than once on the list, there must be a routing loop. Second, path information allows a receiver to implement policy constraints. For example, a receiver can examine paths to verify that they do not pass through untrusted autonomous systems (e.g., a competitor's autonomous system). Third, path information allows a receiver to know the source of all routes. In addition to allowing a sender to specify whether the information came from inside its autonomous system or from another system, the path attributes allow the sender to declare whether the information was collected with an exterior gateway protocol such as BGP or an interior gateway protocol†. Thus, each receiver can decide whether to accept or reject routes that originate in autonomous systems beyond the peer's.

Conceptually, the *Path Attributes* field contains a list of items, where each item consists of a triple:

(type, length, value)

Instead of fixed-size fields, the designers chose a flexible encoding scheme that minimizes the space each item occupies. As specified in Figure 15.10, the type information always requires two octets, but other fields vary in size.

†The next chapter describes interior gateway protocols.



Flag Bits	Description
0	0 for required attribute, 1 if optional
1	1 for transitive, 0 for nontransitive
2	0 for complete, 1 for partial
3	0 if length field is one octet; 1 if two
5-7	unused (must be zero)

Figure 15.10 Bits of the 2-octet type field that appears before each BGP attribute path item and the meaning of each.

For each item in the *Path Attributes* list, a length field follows the 2-octet type field, and may be either one or two octets long. As the figure shows, flag bit *3* specifies the size of the length field. A receiver uses the type field to determine the size of the length field, and then uses the contents of the length field to determine the size of the value field.

Each item in the *Path attributes* field can have one of seven possible type codes. Figure 15.11 summarizes the possibilities.

Type Code	Meaning
1	Specify origin of the path information
2	List of autonomous systems on path to destination
3	Next hop to use for destination
4	Discriminator used for multiple AS exit points
5	Preference used within an autonomous system
6	Indication that routes have been aggregated
7	ID of autonomous system that aggregated routes

Figure 15.11 The BGP attribute type codes and the meaning of each.

15.16 BGP KEEPALIVE Message

Two BGP peers periodically exchange *KEEPALIVE* messages to test network connectivity and to verify that both peers continue to function. A *KEEPALIVE* message consists of the standard message header with no additional data. Thus, the total message size is *19* octets (the minimum BGP message size).

There are two reasons why BGP uses keepalive messages. First, periodic message exchange is needed because BGP uses TCP for transport, and TCP does not include a mechanism to continually test whether a connection endpoint is reachable. However, TCP does report an error to an application if it cannot deliver data the application sends. Thus, as long as both sides periodically send a keepalive, they will know if the TCP connection fails. Second, keepalives conserve bandwidth compared to other messages. Many early routing protocols used periodic exchange of routing information to test connectivity. However, because routing information changes infrequently, the message content seldom changes. Furthermore, because routing messages are usually large, resending the same message wastes network bandwidth needlessly. To avoid the inefficiency, BGP separates the functionality of route update from connectivity testing, allowing BGP to send small *KEEPALIVE* messages frequently, and reserving larger *UPDATE* messages for situations when reachability information changes.

Recall that a BGP speaker specifies a *hold timer* when it opens a connection; the hold timer specifies a maximum time that BGP is to wait without receiving a message. As a special case, the hold timer can be zero to specify that no *KEEPALIVE* messages are used. If the hold timer is greater than zero, the standard recommends setting the *KEEPALIVE* interval to one third of the hold timer. In no case can a BGP speaker make the *KEEPALIVE* interval less than one second (which agrees with the requirement that a nonzero hold timer cannot be less than three seconds).

15.17 Information From The Receiver's Perspective

Unlike most protocols that propagate routing information, an Exterior Gateway Protocol does not merely report the set of destinations it can reach. Instead, exterior protocols must provide information that is correct from the outsider's perspective. There are two issues: policies and optimal routes. The policy issue is obvious: a router inside an autonomous system may be allowed to reach a given destination, while outsiders are prohibited from reaching the same destination. The routing issue means that a router must advertise a next hop that is optimal from the outsider's perspective. Figure 15.12 illustrates the idea.

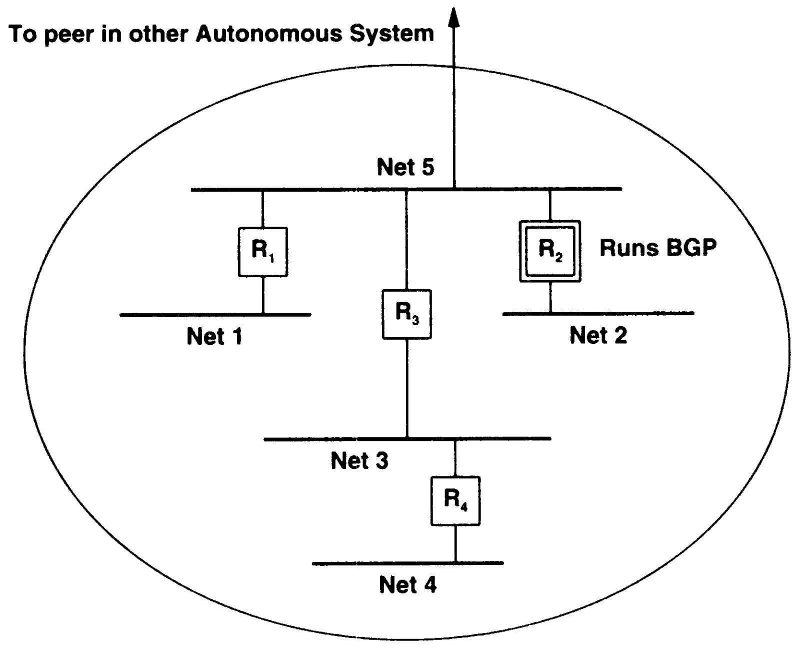


Figure 15.12 Example of an autonomous system. Router R_2 runs BGP and reports information from the outsider's perspective, not from its own routing table.

In the figure, router R_2 has been designated to speak BGP on behalf of the autonomous system. It must report reachability to networks *1* through *4*. However, when giving a next hop, it reports network *1* as reachable through router R_1, networks *3* and *4* as reachable through router R_3, and network *2* as reachable through R_2.

15.18 The Key Restriction Of Exterior Gateway Protocols

We have already seen that because exterior protocols follow policy restrictions, the networks they advertise may be a subset of the networks they can reach. However, there is a more fundamental limitation imposed on exterior routing:

An exterior gateway protocol does not communicate or interpret distance metrics, even if metrics are available.

Protocols like BGP do allow a speaker to declare that a destination has become unreachable or to give a list of autonomous systems on the path to the destination, but they cannot transmit or compare the cost of two routes unless the routes come from within the same autonomous system. In essence, BGP can only specify whether a path exists to a given destination; it cannot transmit or compute the shorter of two paths.

We can see now why BGP is careful to label the origin of information it sends. The essential observation is this: when a router receives advertisements for a given destination from peers in two different autonomous systems, it cannot compare the costs. Thus, advertising reachability with BGP is equivalent to saying, "My autonomous system provides a path to this network." There is no way for the router to say, "My autonomous system provides a better path to this network than another autonomous system."

Looking at interpretation of distances allows us to realize that BGP cannot be used as a routing algorithm. In particular, even if a router learns about two paths to the same network, it cannot know which path is shorter because it cannot know the cost of routes across intermediate autonomous systems. For example, consider a router that uses BGP to communicate with two peers in autonomous systems *p* and *f*. If the peer in autonomous system *p* advertises a path to a given destination through autonomous systems *p*, *q*, and *r*, and the peer in *f* advertises a path to the same destination through autonomous systems *f* and *g*, the receiver has no way of comparing the lengths of the two paths. The path through three autonomous systems might involve one local area network in each system, while the path through two autonomous systems might require several hops in each. Because a receiver does not obtain full routing information, it cannot compare.

Because it does not include a distance metric, an autonomous system must be careful to advertise only routes that traffic should follow. Technically, we say that an Exterior Gateway Protocol is a *reachability protocol* rather than a routing protocol. We can summarize:

> *Because an Exterior Gateway Protocol like BGP only propagates reachability information, a receiver can implement policy constraints, but cannot choose a least cost route. A sender must only advertise paths that traffic should follow.*

The key point here is that any internet which uses BGP to provide exterior routing information must either rely on policies or assume that each autonomous system crossing is equally expensive. Although it may seem innocuous, the restriction has some surprising consequences:

1. Although BGP can advertise multiple paths to a given network, it does not provide for the simultaneous use of multiple paths. That is, at any given instant, all traffic routed from a computer in one autonomous system to a network in another will traverse one path, even if multiple physical connections are present. Also note that an outside autonomous system will only use one return path even if the

source system divides outgoing traffic among two or more paths. As a result, delay and throughput between a pair of machines can be asymmetric, making an internet difficult to monitor or debug.

2. BGP does not support load sharing on routers between arbitrary autonomous systems. If two autonomous systems have multiple routers connecting them, one would like to balance the traffic equally among all routers. BGP allows autonomous systems to divide the load by network (e.g., to partition themselves into multiple subsets and have multiple routers advertise partitions), but it does not support more general load sharing.

3. As a special case of point 2, BGP alone is inadequate for optimal routing in an architecture that has two or more wide area networks interconnected at multiple points. Instead, managers must manually configure which networks are advertised by each exterior router.

4. To have rationalized routing, all autonomous systems in an internet must agree on a consistent scheme for advertising reachability. That is, BGP alone will not guarantee global consistency.

15.19 The Internet Routing Arbiter System

For an internet to operate correctly, routing information must be globally consistent. Individual protocols such as BGP that handle the exchange between a pair of routers, do not guarantee global consistency. Thus, a mechanism is needed to rationalize routing information globally. In the original Internet routing architecture, the core system guaranteed globally consistent routing information because at any time the core had exactly one path to each destination. When the core system was removed, a new mechanism was created to rationalize routing information.

Known as the *routing arbiter* (*RA*) system, the new mechanism consists of a replicated, authenticated database of reachability information. Updates to the database are *authenticated* to prevent an arbitrary router from advertising a path to a given destination. In general, only an autonomous system that owns a given network is allowed to advertise reachability. The need for such authentication became obvious in the early core system, which allowed any router to advertise reachability to any network. On several occasions, routing errors occurred when a router inadvertently advertised incorrect reachability information. The core accepted the information and changed routes, causing some networks to become unreachable.

To understand how other routers access the routing arbiter database, consider the current Internet architecture. We said that major ISPs interconnect at Network Access Points (NAPs). Thus, in terms of routing, a NAP represents the boundary between multiple autonomous systems. Although it would be possible to use BGP among each pair of ISPs at the NAP, doing so is both inefficient and prone to inconsistencies. Instead, each NAP has a computer called a *route server* (*RS*) that maintains a copy of the routing arbiter database and runs BGP. Each ISP designates one of its routers near a NAP to be a BGP border router. The designated border router maintains a connection to the route server over which it uses BGP. The ISP advertises reachability to its networks and the networks of its customers, and learns about networks in other ISPs.

One of the chief advantages of using BGP for route server access lies in its ability to carry negative information as well as positive information. When a destination becomes unreachable, the ISP informs the route server, which then makes the information available to other ISPs. Spreading negative information reduces unnecessary traffic because datagrams to unreachable destinations can be discarded before they transit from one ISP to another†.

15.20 BGP NOTIFICATION Message

In addition to the OPEN and UPDATE message types described above, BGP supports a *NOTIFICATION* message type used for control or when an error occurs. Errors are permanent — once it detects a problem, BGP sends a notification message and then closes the TCP connection. Figure 15.13 illustrates the message format.

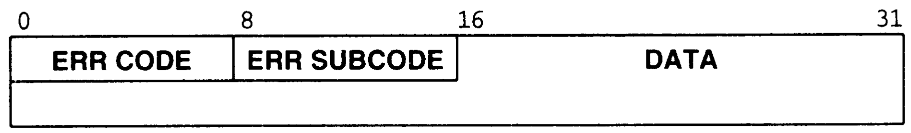


Figure 15.13 BGP NOTIFICATION message format. These octets follow the standard message header.

The 8-bit field labeled *ERR CODE* specifies one of the possible reasons listed in Figure 15.14.

ERR CODE	Meaning
1	Error in message header
2	Error in OPEN message
3	Error in UPDATE message
4	Hold timer expired
5	Finite state machine error
6	Cease (terminate connection)

Figure 15.14 The possible values of the *ERR CODE* field in a BGP NOTIFICATION message.

†Like the core system it replaced, the routing arbiter system does not include default routes. As a consequence, it is sometimes called a *default-free zone*.

For each possible *ERR CODE*, the *ERR SUBCODE* field contains a further explanation. Figure 15.15 lists the possible values.

Subcodes For Message Header Errors	
1	Connection not synchronized
2	Incorrect message length
3	Incorrect message type
Subcodes For OPEN Message Errors	
1	Version number unsupported
2	Peer AS invalid
3	BGP identifier invalid
4	Unsupported optional parameter
5	Authentication failure
6	Hold time unacceptable
Subcodes For UPDATE Message Errors	
1	Attribute list malformed
2	Unrecognized attribute
3	Missing attribute
4	Attribute flags error
5	Attribute length error
6	Invalid ORIGIN attribute
7	AS routing loop
8	Next hop invalid
9	Error in optional attribute
10	Invalid network field
11	Malformed AS path

Figure 15.15 The meaning of the *ERR SUBCODE* field in a BGP NOTIFICATION message.

15.21 Decentralization Of Internet Architecture

Two important architecture questions remain unanswered. The first focuses on centralization: how can the Internet architecture be modified to remove dependence on a (centralized) router system? The second concerns levels of trust: can an internet architecture be expanded to allow closer cooperation (trust) between some autonomous systems than among others?

Removing all dependence on a central system and adding trust are not easy. Although TCP/IP architectures continue to evolve, centralized roots are evident in many protocols. Without some centralization, each ISP would need to exchange reachability information with all ISPs to which it attached. Consequently, the volume of routing traffic would be significantly higher than with a routing arbiter scheme. Finally, centralization fills an important role in rationalizing routes and guaranteeing trust — in addition to storing the reachability database, the routing arbiter system guarantees global consistency and provides a trusted source of information.

15.22 Summary

Routers must be partitioned into groups or the volume of routing traffic would be intolerable. The connected Internet is composed of a set of autonomous systems, where each autonomous system consists of routers and networks under one administrative authority. An autonomous system uses an Exterior Gateway Protocol to advertise routes to other autonomous systems. Specifically, an autonomous system must advertise reachability of its networks to another system before its networks are reachable from sources within the other system.

The Border Gateway Protocol, BGP, is the most widely used Exterior Gateway Protocol. We saw that BGP contains three message types that are used to initiate communication (OPEN), send reachability information (UPDATE) and report an error condition (NOTIFICATION). Each message starts with a standard header that includes (optional) authentication information. BGP uses TCP for communication, but has a keepalive mechanism to ensure that peers remain in communication.

In the global Internet, each ISP is assigned to a separate autonomous system, and the main boundary among autonomous systems occurs at NAPs, where multiple ISPs interconnect. Instead of requiring pairs of ISPs to use BGP to exchange routing information, each NAP includes a route server. An ISP uses BGP to communicate with the route server, both to advertise reachability to its networks and its customers' networks as well as to learn about networks in other ISPs.

FOR FURTHER STUDY

Background on early Internet routing can be found in [RFCs 827, 888, 904, and 975]. Rekhter and Li [RFC 1771] describes version 4 of the Border Gateway Protocol *(BGP-4)*. BGP has been through three substantial revisions; earlier versions appear in [RFCs 1163, 1267, and 1654]. Traina [RFC 1773] reports experience with BGP-4, and Traina [RFC 1774] analyzes the volume of routing traffic generated. Finally, Villamizar et. al. {RFC 2439] considers the problem of route flapping.

EXERCISES

15.1 If your site runs an Exterior Gateway Protocol such as BGP, how many routes does NSFNET advertise?

15.2 Some implementations of BGP use a "hold down" mechanism that causes the protocol to delay accepting an *OPEN* from a peer for a fixed time following the receipt of a *cease request* message from that neighbor. Find out what problem a hold down helps solve.

15.3 For the networks in Figure 15.2, which router(s) should run BGP? Why?

15.4 The formal specification of BGP includes a finite state machine that explains how BGP operates. Draw a diagram of the state machine and label transitions.

15.5 What happens if a router in an autonomous system sends BGP routing update messages to a router in another autonomous system, claiming to have reachability for every possible internet destination?

15.6 Can two autonomous systems establish a routing loop by sending BGP update messages to one another? Why or why not?

15.7 Should a router that uses BGP to advertise routes treat the set of routes advertised differently than the set of routes in the local routing table? For example, should a router ever advertise reachability if it has not installed a route to that network in its routing table? Why or why not? Hint: read the RFC.

15.8 With regard to the previous question, examine the BGP-4 specification carefully. Is it legal to advertise reachability to a destination that is not listed in the local routing table?

15.9 If you work for a large corporation, find out whether it includes more than one autonomous system. If so, how do they exchange routing information?

15.10 What is the chief advantage of dividing a large, multi-national corporation into multiple autonomous systems? What is the chief disadvantage?

15.11 Corporations *A* and *B* use BGP to exchange routing information. To keep computers in *B* from reaching machines on one of its networks, *N*, the network administrator at corporation *A* configures BGP to omit *N* from advertisements sent to *B*. Is network *N* secure? Why or why not?

15.12 Because BGP uses a reliable transport protocol, KEEPALIVE messages cannot be lost. Does it make sense to specify a keepalive interval as one-third of the hold timer value? Why or why not?

15.13 Consult the RFCs for details of the *Path Attributes* field. What is the minimum size of a BGP UPDATE message?

16

Routing: In An Autonomous System (RIP, OSPF, HELLO)

16.1 Introduction

The previous chapter introduces the autonomous system concept and examines BGP, an Exterior Gateway Protocol that a router uses to advertise networks within its system to other autonomous systems. This chapter completes our overview of internet routing by examining how a router in an autonomous system learns about other networks within its autonomous system.

16.2 Static Vs. Dynamic Interior Routes

Two routers within an autonomous system are said to be *interior* to one another. For example, two routers on a university campus are considered interior to one another as long as machines on the campus are collected into a single autonomous system.

How can routers in an autonomous system learn about networks within the autonomous system? In small, slowly changing internets, managers can establish and modify routes by hand. The administrator keeps a table of networks and updates the table whenever a new network is added to, or deleted from, the autonomous system. For example, consider the small corporate internet shown in Figure 16.1.

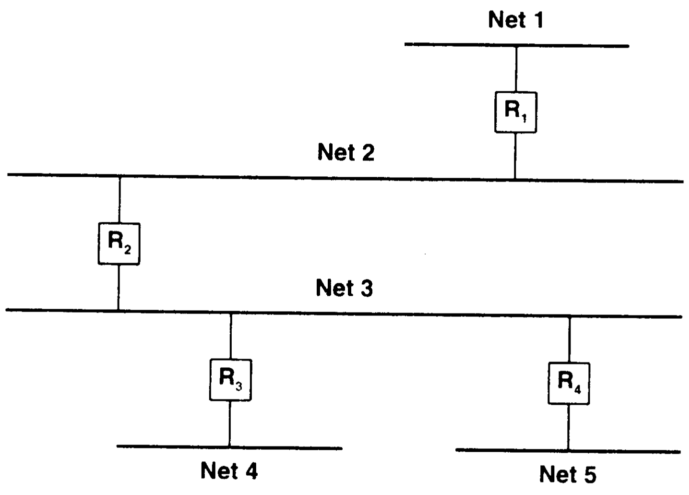


Figure 16.1 An example of a small internet consisting of 5 Ethernets and 4 routers at a single site. Only one possible route exists between any two hosts in this internet.

Routing for the internet in the figure is trivial because only one path exists between any two points. The manager can manually configure routes in all hosts and routers. If the internet changes (e.g., a new network is added), the manager must reconfigure the routes in all machines.

The disadvantages of a manual system are obvious: manual systems cannot accommodate rapid growth or rapid change. In large, rapidly changing environments like the global Internet, humans simply cannot respond to changes fast enough to handle problems; automated methods must be used. Automated methods can also help improve reliability and response to failure in small internets that have alternate routes. To see how, consider what happens if we add one additional router to the internet in Figure 16.1, producing the internet shown in Figure 16.2.

In internet architectures that have multiple physical paths, managers usually choose one to be the primary path. If the routers along the primary path fail, routes must be changed to send traffic along an alternate path. Changing routes manually is both time consuming and error-prone. Thus, even in small internets, an automated system should be used to change routes quickly and reliably.

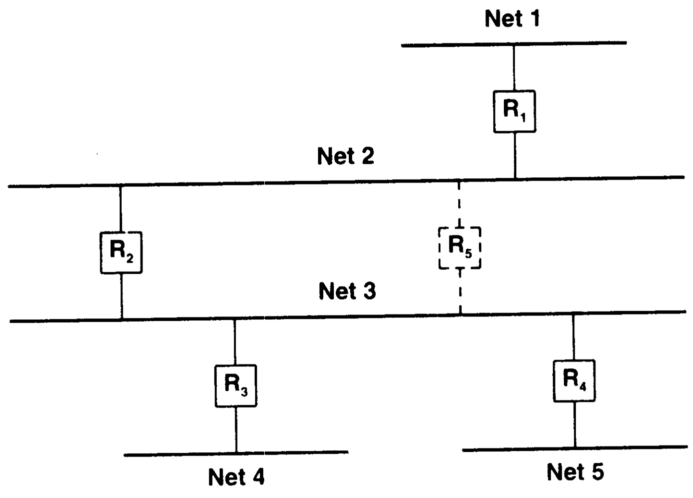


Figure 16.2 The addition of router R_5 introduces an alternate path between networks *2* and *3*. Routing software can quickly adapt to a failure and automatically switch routes to the alternate path.

To automate the task of keeping network reachability information accurate, interior routers usually communicate with one another, exchanging either network reachability data or network routing information from which reachability can be deduced. Once the reachability information for an entire autonomous system has been assembled, one of the routers in the system can advertise it to other autonomous systems using an Exterior Gateway Protocol.

Unlike exterior router communication, for which BGP provides a widely accepted standard, no single protocol has emerged for use within an autonomous system. Part of the reason for diversity comes from the varied topologies and technologies used in autonomous systems. Another part of the reason stems from the tradeoffs between simplicity and functionality — protocols that are easy to install and configure do not provide sophisticated functionality. As a result, a handful of protocols have become popular. Most small autonomous systems choose a single protocol, and use it exclusively to propagate routing information internally. Larger autonomous systems often choose a small set.

Because there is no single standard, we use the term *Interior Gateway Protocol* (*IGP*) as a generic description that refers to any algorithm that interior routers use when they exchange network reachability and routing information. For example, the last generation of core routers used a protocol named *SPREAD* as its Interior Gateway Protocol. Some autonomous systems use BGP as their IGP, although this seldom makes sense for small autonomous systems that span local area networks with broadcast capability.

Figure 16.3 illustrates two autonomous systems, each using an IGP to propagate routing information among its interior routers.

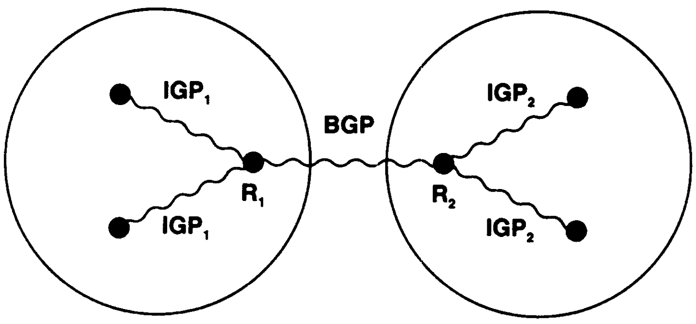


Figure 16.3 Conceptual view of two autonomous systems each using its own IGP internally, but using BGP to communicate between an exterior router and the other system.

In the figure, *IGP$_1$* refers to the interior router protocol used within autonomous system *1*, and *IGP$_2$* refers to the protocol used within autonomous system *2*. The figure also illustrates an important idea:

> *A single router may use two different routing protocols simultaneously, one for communication outside its autonomous system and another for communication within its autonomous system.*

In particular, routers that run BGP to advertise reachability usually also need to run an IGP to obtain information from within their autonomous system.

16.3 Routing Information Protocol (RIP)

16.3.1 History of RIP

One of the most widely used IGPs is the *Routing Information Protocol* (*RIP*), also known by the name of a program that implements it, *routed*†. The *routed* software was originally designed at the University of California at Berkeley to provide consistent routing and reachability information among machines on their local networks. It relies on physical network broadcast to make routing exchanges quickly. It was not designed to be used on large, wide area networks (although vendors now sell versions of RIP adapted for use on WANs).

Based on earlier internetworking research done at Xerox Corporation's Palo Alto Research Center (PARC), *routed* implements a protocol derived from the Xerox *NS Routing Information Protocol* (*RIP*), but generalizes it to cover multiple families of networks.

†The name comes from the UNIX convention of attaching "d" to the names of daemon processes; it is pronounced "route-d".

Despite minor improvements over its predecessors, the popularity of RIP as an IGP does not arise from its technical merits alone. Instead, it is the result of Berkeley distributing *routed* software along with their popular 4BSD UNIX systems. Thus, many TCP/IP sites adopted and installed *routed*, and started using RIP without even considering its technical merits or limitations. Once installed and running, it became the basis for local routing, and research groups adopted it for larger networks.

Perhaps the most startling fact about RIP is that it was built and widely adopted before a formal standard was written. Most implementations were derived from the Berkeley code, with interoperability among them limited by the programmer's understanding of undocumented details and subtleties. As new versions appeared, more problems arose. An RFC standard appeared in June 1988, and made it possible for vendors to ensure interoperability.

16.3.2 RIP Operation

The underlying RIP protocol is a straightforward implementation of distance-vector routing for local networks. It partitions participants into *active* and *passive* (i.e., *silent*) machines. Active participants advertise their routes to others; passive participants listen to RIP messages and use them to update their routing table, but do not advertise. Only a router can run RIP in active mode; a host must use passive mode.

A router running RIP in active mode broadcasts a routing update message every 30 seconds. The update contains information taken from the router's current routing database. Each update contains a set of pairs, where each pair contains an IP network address and an integer distance to that network. RIP uses a *hop count metric* to measure distances. In the RIP metric, a router is defined to be one hop from a directly connected network†, two hops from a network that is reachable through one other router, and so on. Thus, the *number of hops* or the *hop count* along a path from a given source to a given destination refers to the number of routers that a datagram encounters along that path. It should be obvious that using hop counts to calculate shortest paths does not always produce optimal results. For example, a path with hop count *3* that crosses three Ethernets may be substantially faster than a path with hop count *2* that crosses two satellite connections. To compensate for differences in technologies, many RIP implementations allow managers to configure artificially high hop counts when advertising connections to slow networks.

Both active and passive RIP participants listen to all broadcast messages, and update their tables according to the distance-vector algorithm described earlier. For example, in the internet of Figure 16.2, router R_1 will broadcast a message on network *2* that contains the pair $(1, 1)$, meaning that it can reach network *1* at cost *1*. Routers R_2 and R_5 will receive the broadcast and install a route to network *1* through R_1 (at cost *2*). Later, routers R_2 and R_5 will include the pair $(1, 2)$ when they broadcast their RIP messages on network *3*. Eventually, all routers and hosts will install a route to network *1*.

RIP specifies a few rules to improve performance and reliability. For example, once a router learns a route from another router, it must apply *hysteresis*, meaning that it does not replace the route with an equal cost route. In our example, if routers R_2 and

†Other routing protocols define a direct connection to be zero hops.

R_5 both advertise network *1* at cost *2*, routers R_3 and R_4 will install a route through the one that happens to advertise first. We can summarize:

> *To prevent oscillation among equal cost paths, RIP specifies that existing routes should be retained until a new route has strictly lower cost.*

What happens if the first router to advertise a route fails (e.g., if it crashes)? RIP specifies that all listeners must timeout routes they learn via RIP. When a router installs a route in its table, it starts a timer for that route. The timer must be restarted whenever the router receives another RIP message advertising the route. The route becomes invalid if 180 seconds pass without the route being advertised again.

RIP must handle three kinds of errors caused by the underlying algorithm. First, because the algorithm does not explicitly detect routing loops, RIP must either assume participants can be trusted or take precautions to prevent such loops. Second, to prevent instabilities RIP must use a low value for the maximum possible distance (RIP uses *16*). Thus, for internets in which legitimate hop counts approach *16*, managers must divide the internet into sections or use an alternative protocol. Third, the distance-vector algorithm used by RIP can create a *slow convergence* or *count to infinity* problem, in which inconsistencies arise because routing update messages propagate slowly across the network. Choosing a small infinity (*16*) helps limit slow convergence, but does not eliminate it.

Routing table inconsistency is not unique to RIP. It is a fundamental problem that occurs with any distance-vector protocol in which update messages carry only pairs of destination network and distance to that network. To understand the problem consider the set of routers shown in Figure 16.4. The figure depicts routes to network *1* for the internet shown in Figure 16.2.

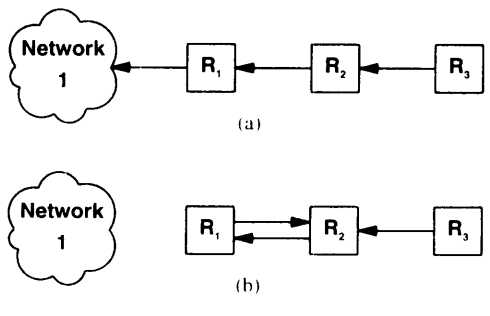


Figure 16.4 The slow convergence problem. In (a) three routers each have a route to network *1*. In (b) the connection to network *1* has vanished, but R_2 causes a loop by advertising it.

As Figure 16.4a shows, router R_1 has a direct connection to network *1*, so there is a route in its table with distance *1*, which will be included in its periodic broadcasts. Router R_2 has learned the route from R_1, installed the route in its routing table, and advertises the route at distance *2*. Finally, R_3 has learned the route from R_2 and advertises it at distance *3*.

Now suppose that R_1's connection to network *1* fails. R_1 will update its routing table immediately to make the distance *16* (infinity). In the next broadcast, R_1 will report the higher cost route. However, unless the protocol includes extra mechanisms to prevent it, some other router could broadcast its routes before R_1. In particular, suppose R_2 happens to advertise routes just after R_1*'s* connection fails. If so, R_1 will receive R_2's message and follow the usual distance-vector algorithm: it notices that R_2 has advertised a route to network *1* at lower cost, calculates that it now takes *3* hops to reach network *1* (*2* for R_2 to reach network *1* plus *1* to reach R_2*), and installs a new route with* R_2 listed as the next hop. Figure 16.4b depicts the result. At this point, if either R_1 or R_2 receives a datagram destined for network *1*, they will route the datagram back and forth until the datagram's time-to-live counter expires.

Subsequent RIP broadcasts by the two routers do not solve the problem quickly. In the next round of routing exchanges, R_1 broadcasts its routing table entries. When it learns that R_1's route to network *1* has distance *3*, R_2 calculates a new distance for its route, making it *4*. In the third round, R_1 receives a report from R_2 which includes the increased distance, and then increases the distance in its table to *5*. The two routers continue counting to RIP infinity.

16.3.3 Solving The Slow Convergence Problem

For the example in Figure 16.4, it is possible to solve the slow convergence problem by using a technique known as *split horizon update*. When using split horizon, a router does not propagate information about a route back over the same interface from which the route arrived. In the example, split horizon prevents router R_2 from advertising a route to network *1* back to router R_1, so if R_1 loses connectivity to network *1*, it must stop advertising a route. With split horizon, no routing loop appears in the example network. Instead, after a few rounds of routing updates, all routers will agree that the network is unreachable. However, the split horizon heuristic does not prevent routing loops in all possible topologies as one of the exercises suggests.

Another way to think of the slow convergence problem is in terms of information flow. If a router advertises a short route to some network, all receiving routers respond quickly to install that route. If a router stops advertising a route, the protocol must depend on a timeout mechanism before it considers the route unreachable. Once the timeout occurs, the router finds an alternative route and starts propagating that information. Unfortunately, a router cannot know if the alternate route depended on the route that just disappeared. Thus, negative information does not always propagate quickly. A short epigram captures the idea and explains the phenomenon:

Good news travels quickly; bad news travels slowly.

Another technique used to solve the slow convergence problem employs *hold down*. Hold down forces a participating router to ignore information about a network for a fixed period of time following receipt of a message that claims the network is unreachable. Typically, the hold down period is set to 60 seconds. The idea is to wait long enough to ensure that all machines receive the bad news and not mistakenly accept a message that is out of date. It should be noted that all machines participating in a RIP exchange need to use identical notions of hold down, or routing loops can occur. The disadvantage of a hold down technique is that if routing loops occur, they will be preserved for the duration of the hold down period. More important, the hold down technique preserves all incorrect routes during the hold down period, even when alternatives exist.

A final technique for solving the slow convergence problem is called *poison reverse*. Once a connection disappears, the router advertising the connection retains the entry for several update periods, and includes an infinite cost in its broadcasts. To make poison reverse most effective, it must be combined with *triggered updates*. Triggered updates force a router to send an immediate broadcast when receiving bad news, instead of waiting for the next periodic broadcast. By sending an update immediately, a router minimizes the time it is vulnerable to believing good news.

Unfortunately, while triggered updates, poison reverse, hold down, and split horizon techniques all solve some problems, they introduce others. For example, consider what happens with triggered updates when many routers share a common network. A single broadcast may change all their routing tables, triggering a new round of broadcasts. If the second round of broadcasts changes tables, it will trigger even more broadcasts. A broadcast avalanche can result†.

The use of broadcast, potential for routing loops, and use of hold down to prevent slow convergence can make RIP extremely inefficient in a wide area network. Broadcasting always takes substantial bandwidth. Even if no avalanche problems occur, having all machines broadcast periodically means that the traffic increases as the number of routers increases. The potential for routing loops can also be deadly when line capacity is limited. Once lines become saturated by looping packets, it may be difficult or impossible for routers to exchange the routing messages needed to break the loops. Also, in a wide area network, hold down periods are so long that the timers used by higher level protocols can expire and lead to broken connections. Despite these well-known problems, many groups continue to use RIP as an IGP in wide area networks.

16.3.4 RIP1 Message Format

RIP messages can be broadly classified into two types: routing information messages and messages used to request information. Both use the same format which consists of a fixed header followed by an optional list of network and distance pairs. Figure 16.5 shows the message format used with version *1* of the protocol, which is known as *RIP1*:

†To help avoid collisions on the underlying network, RIP requires each router to wait a small random time before sending a triggered update.

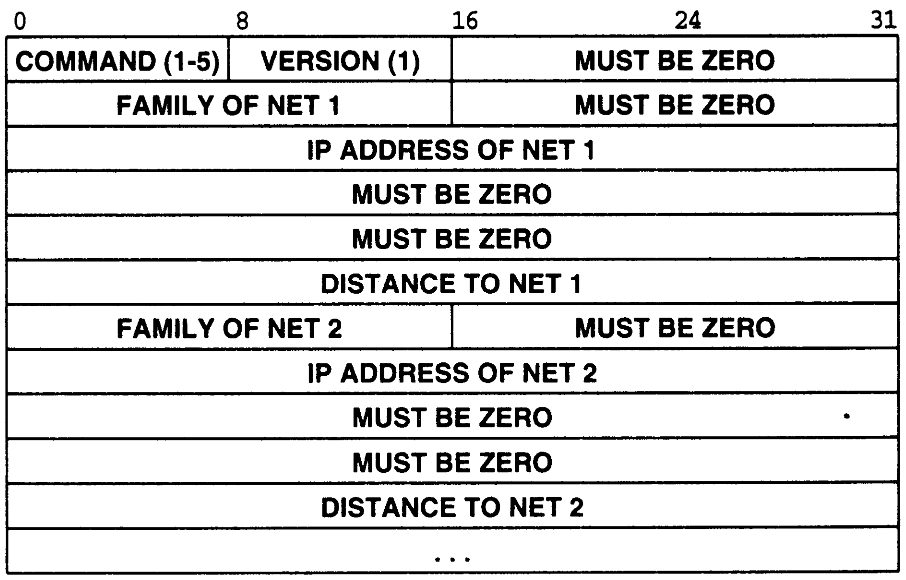


Figure 16.5 The format of a version 1 RIP message. After the 32-bit header, the message contains a sequence of pairs, where each pair consists of a network IP address and an integer distance to that network.

In the figure, field *COMMAND* specifies an operation according to the following table:

Command	Meaning
1	Request for partial or full routing information
2	Response containing network-distance pairs from sender's routing table
3	Turn on trace mode (obsolete)
4	Turn off trace mode (obsolete)
5	Reserved for Sun Microsystems internal use
9	Update Request (used with demand circuits)
10	Update Response (used with demand circuits)
11	Update Acknowledge (used with demand circuits)

A router or host can ask another router for routing information by sending a *request* command. Routers reply to requests using the *response* command. In most cases, however, routers broadcast unsolicited response messages periodically. Field *VERSION* contains the protocol version number (*1* in this case), and is used by the receiver to verify it will interpret the message correctly.

16.3.5 RIP1 Address Conventions

The generality of RIP is also evident in the way it transmits network addresses. The address format is not limited to use by TCP/IP; it can be used with multiple network protocol suites. As Figure 16.5 shows, each network address reported by RIP can have an address of up to 14 octets. Of course, IP addresses need only 4; RIP specifies that the remaining octets must be zero†. The field labeled *FAMILY OF NET i* identifies the protocol family under which the network address should be interpreted. RIP uses values assigned to address families under the 4BSD UNIX operating system (IP addresses are assigned value *2*).

In addition to normal IP addresses, RIP uses the convention that address *0.0.0.0* denotes a *default route*. RIP attaches a distance metric to every route it advertises, including default routes. Thus, it is possible to arrange for two routers to advertise a default route (e.g., a route to the rest of the internet) at different metrics, making one of them a primary path and the other a backup.

The final field of each entry in a RIP message, *DISTANCE TO NET i*, contains an integer count of the distance to the specified network. Distances are measured in router hops, but values are limited to the range *1* through *16*, with distance *16* used to signify infinity (i.e., no route exists).

16.3.6 RIP1 Route Interpretation And Aggregation

Because RIP was originally designed to be used with classful addresses, version 1 did not include any provision for a subnet mask. When subnet addressing was added to IP, version 1 of RIP was extended to permit routers to exchange subnetted addresses. However, because RIP1 update messages do not contain explicit mask information, an important restriction was added: a router can include host-specific or subnet-specific addresses in routing updates as long as all receivers can unambiguously interpret the addresses. In particular, subnet routes can only be included in updates sent across a network that is part of the subnetted prefix, and only if the subnet mask used with the network is the same as the subnet mask used with the address. In essence, the restriction means that RIP1 cannot be used to propagate variable-length subnet address or classless addresses. We can summarize:

> *Because it does not include explicit subnet information, RIP1 only permits a router to send subnet routes if receivers can unambiguously interpret the addresses according to the subnet mask they have available locally. As a consequence, RIP1 can only be used with classful or fixed-length subnet addresses.*

What happens when a router running RIP1 connects to one or more networks that are subnets of a prefix *N* as well as to one or more networks that are not part of *N*? The router must prepare different update messages for the two types of interfaces. Updates sent over the interfaces that are subnets of *N* can include subnet routes, but updates sent

†The designers chose to locate an IP address in the third through sixth octets of the address field to ensure 32-bit alignment.

over other interfaces cannot. Instead, when sending over other interfaces the router is required to *aggregate* the subnet information and advertise a single route to network *N*.

16.3.7 RIP2 Extensions

The restriction on address interpretation means that version 1 of RIP cannot be used to propagate either variable-length subnet addresses or the classless addresses used with CIDR. When version 2 of RIP (*RIP2*) was defined, the protocol was extended to include an explicit subnet mask along with each address. In addition, RIP2 updates include explicit next-hop information, which prevents routing loops and slow convergence. As a result, RIP2 offers significantly increased functionality as well as improved resistance to errors.

16.3.8 RIP2 Message Format

The message format used with RIP2 is an extension of the RIP1 format, with additional information occupying unused octets of the address field. In particular, each address includes an explicit next hop as well as an explicit subnet mask as Figure 16.6 illustrates.

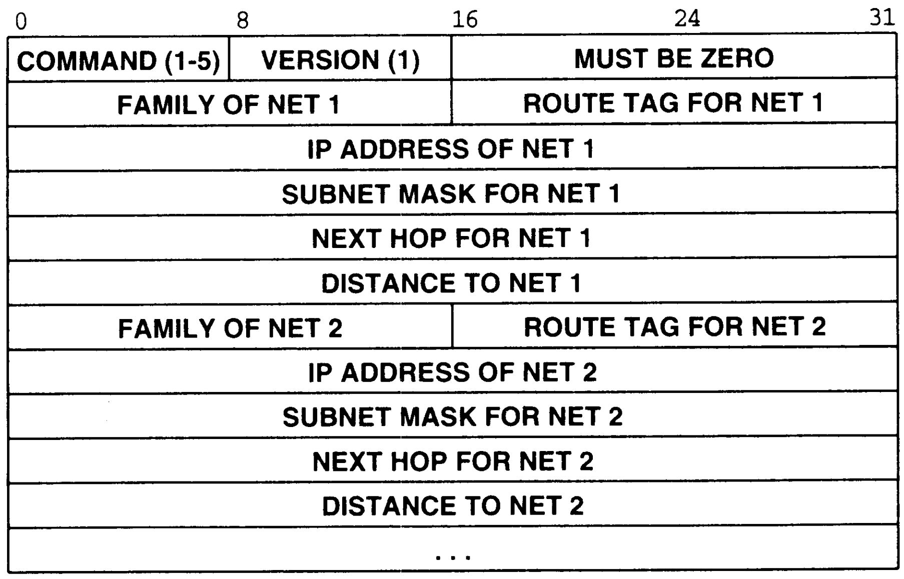


Figure 16.6 The format of a RIP2 message. In addition to pairs of a network IP address and an integer distance to that network, the message contains a subnet mask for each address and explicit next-hop information.

RIP2 also attaches a 16-bit *ROUTE TAG* field to each entry. A router must send the same tag it receives when it transmits the route. Thus, the tag provides a way to propagate additional information such as the origin of the route. In particular, if RIP2 learns a route from another autonomous system, it can use the *ROUTE TAG* to propagate the autonomous system's number.

Because the version number in RIP2 occupies the same octet as in RIP1, both versions of the protocols can be used on a given router simultaneously without interference. Before processing an incoming message, RIP software examines the version number.

16.3.9 Transmitting RIP Messages

RIP messages do not contain an explicit length field or an explicit count of entries. Instead, RIP assumes that the underlying delivery mechanism will tell the receiver the length of an incoming message. In particular, when used with TCP/IP, RIP messages rely on UDP to tell the receiver the message length. RIP operates on UDP port *520*. Although a RIP request can originate at other UDP ports, the destination UDP port for requests is always *520*, as is the source port from which RIP broadcast messages originate.

16.3.10 The Disadvantage Of RIP Hop Counts

Using RIP as an interior router protocol limits routing in two ways. First, RIP restricts routing to a hop-count metric. Second, because it uses a small value of hop count for infinity, RIP restricts the size of any internet using it. In particular, RIP restricts the *span* of an internet (i.e., the maximum distance across) to 16. That is, an internet using RIP can have at most 15 routers between any two hosts.

Note that the limit on network span is neither a limit on the total number of routers nor a limit on density. In fact, most campus networks have a small span even if they have many routers because the topology is arranged as a *hierarchy*. Consider, for example, a typical corporate intranet. Most use a hierarchy that consists of a high-speed backbone network with multiple routers each connecting the backbone to a workgroup, where each workgroup occupies a single LAN. Although the corporation can include dozens of workgroups, the span of the entire intranet is only 2. Even if each workgroup is extended to include a router that connects one or more additional LANs, the maximum span only increases to 4. Similarly, extending the hierarchy one more level only increases the span to 6. Thus, the limit that RIP imposes affects large autonomous systems or autonomous systems that do not have a hierarchical organization.

Even in the best cases, however, hop counts provide only a crude measure of network capacity or responsiveness. Thus, using hop counts does not always yield routes with least delay or highest capacity. Furthermore, computing routes on the basis of minimum hop counts has the severe disadvantage that it makes routing relatively static because routes cannot respond to changes in network load. The next sections consider an alternative metric, and explain why hop count metrics remain popular despite their limitations.

16.4 The Hello Protocol

The HELLO protocol provides an example of an IGP that uses a routing metric other than hop count. Although HELLO is now obsolete, it was significant in the history of the Internet because it was the IGP used among the original NSFNET backbone "fuzzball" routers†. HELLO is significant to us because it provides an example of a protocol that uses a metric of delay.

HELLO provides two functions: it synchronizes the clocks among a set of machines, and it allows each machine to compute shortest delay paths to destinations. Thus, HELLO messages carry timestamp information as well as routing information. The basic idea behind HELLO is simple: each machine participating in the HELLO exchange maintains a table of its best estimate of the clocks in neighboring machines. Before transmitting a packet, a machine adds its timestamp by copying the current clock value into the packet. When a packet arrives, the receiver computes an estimate of the current delay on the link by subtracting the timestamp on the incoming packet from the local estimate for the current clock in the neighbor. Periodically, machines poll their neighbors to reestablish estimates for clocks.

HELLO messages also allow participating machines to compute new routes. The protocol uses a modified distance-vector scheme that uses a metric of delay instead of hop count. Thus, each machine periodically sends its neighbors a table of destinations it can reach and an estimated delay for each. When a message arrives from machine *X*, the receiver examines each entry in the message and changes the next hop to *X* if the route through *X* is less expensive than the current route (i.e., any route where the delay to *X* plus the delay from *X* to the destination is less than the current delay to the destination).

16.5 Delay Metrics And Oscillation

It may seem that using delay as a routing metric would produce better routes than using a hop count. In fact, HELLO worked well in the early Internet backbone. However, there is an important reasons why delay is not used as a metric in most protocols: instability.

Even if two paths have identical characteristics, any protocol that changes routes quickly can become unstable. Instability arises because delay, unlike hop counts, is not fixed. Minor variations in delay measurements occur because of hardware clock drift, CPU load during measurement, or bit delays caused by link-level synchronization. Thus, if a routing protocol reacts quickly to slight differences in delay, it can produce a two-stage oscillation effect in which traffic switches back and forth between the alternate paths. In the first stage, the router finds the delay on path *I* slightly less and abruptly switches traffic onto it. In the next round, the router finds that path *B* has slightly less delay and switches traffic back.

To help avoid oscillation, protocols that use delay implement several heuristics. First, they employ the *hold down* technique discussed previously to prevent routes from

†The term *fuzzball* referred to a noncommercial router that consisted of specially-crafted protocol software running on a PDP11 computer.

changing rapidly. Second, instead of measuring as accurately as possible and comparing the values directly, the protocols round measurements to large multiples or implement a minimum *threshold* by ignoring differences less than the threshold. Third, instead of comparing each individual delay measurement, they keep a running *average* of recent values or alternatively apply a *K-out-of-N* rule that requires at least K of the most recent N delay measurements be less than the current delay before the route can be changed.

Even with heuristics, protocols that use delay can become unstable when comparing delays on paths that do not have identical characteristics. To undersand why, it is necessary to know that traffic can have a dramatic effect on delay. With no traffic, the network delay is simply the time required for the hardware to transfer bits from one point to another. As the traffic load imposed on the network increases, however, delays begin to rise because routers in the system need to enqueue packets that are waiting for transmission. If the load is even slightly more than 100% of the network capacity, the queue becomes unbounded, meaning that the effective delay becomes infinite. To summarize:

The effective delay across a network depends on traffic; as the load increases to 100% of the network capacity, delay grows rapidly.

Because delays are extremely sensitive to changes in load, protocols that use delay as a metric can easily fall into a *positive feedback cycle*. The cycle is triggered by a small external change in load (e.g., one computer injecting a burst of additional traffic). The increased traffic raises the delay, which causes the protocol to change routes. However, because a route change affects the load, it can produce an even larger change in delays, which means the protocol will again recompute routes. As a result, protocols that use delay must contain mechanisms to dampen oscillation.

We described heuristics that can solve simple cases of route oscillation when paths have identical throughput characteristics and the load is not excessive. The heuristics can become ineffective, however, when alternative paths have different delay and throughput characteristics. As an example consider the delay on two paths: one over a satellite and the other over a low capacity serial line (e.g., a 9600 baud serial line). In the first stage of the protocol when both paths are idle, the serial line will appear to have significantly lower delay than the satellite, and will be chosen for traffic. Because the serial line has low capacity, it will quickly become overloaded, and the delay will rise sharply. In the second stage, the delay on the serial line will be much greater than that of the satellite, so the protocol will switch traffic away from the overloaded path. Because the satellite path has large capacity, traffic which overloaded the serial line does not impose a significant load on the satellite, meaning that the delay on the satellite path does not change with traffic. In the next round, the delay on the unloaded serial line will once again appear to be much smaller than the delay on the satellite path. The protocol will reverse the routing, and the cycle will continue. Such oscillations do, in fact, occur in practice. As the example shows, they are difficult to manage because traffic which has little effect on one network can overload another.

16.6 Combining RIP, Hello, And BGP

We have already observed that a single router may use both an Interior Gateway Protocol to gather routing information within its autonomous system and an Exterior Gateway Protocol to advertise routes to other autonomous systems. In principle, it should be easy to construct a single piece of software that combines the two protocols, making it possible to gather routes and advertise them without human intervention. In practice, technical and political obstacles make doing so complex.

Technically, IGP protocols, like RIP and Hello, are routing protocols. A router uses such protocols to update its routing table based on information it acquires from other routers inside its autonomous system. Thus, *routed*, the UNIX program that implements RIP, advertises information from the local routing table and changes the local routing table when it receives updates. RIP trusts routers within the same autonomous system to pass correct data.

In contrast, exterior protocols such as BGP do not trust routers in other autonomous systems. Consequently, exterior protocols do not advertise all possible routes from the local routing table. Instead, such protocols keep a database of network reachability, and apply policy constraints when sending or receiving information. Ignoring such policy constraints can affect routing in a larger sense — some parts of the internet can be become unreachable. For example, if a router in an autonomous system that is running RIP happens to propagate a low-cost route to a network at Purdue University when it has no such route, other routers running RIP will accept and install the route. They will then pass Purdue traffic to the router that made the error. As a result, it may be impossible for hosts in that autonomous system to reach Purdue. The problem becomes more serious if Exterior Gateway Protocols do not implement policy constraints. For example, if a border router in the autonomous system uses BGP to propagate the illegal route to other autonomous systems, the network at Purdue may become unreachable from some parts of the internet.

16.7 Inter-Autonomous System Routing

We have seen that EGPs such as BGP allow one autonomous system to advertise reachability information to another. However, it would be useful to also provide *inter-autonomous system routing* in which routers choose least-cost paths. Doing so requires additional trust. Extending the notions of trust from a single autonomous system to multiple autonomous systems is complex. The simplest approach groups autonomous systems hierarchically. Imagine, for example, three autonomous systems in three separate academic departments on a large university campus. It is natural to group these three together because they share administrative ties. The motivation for hierarchical grouping comes primarily from the notion of trust. Routers within a group trust one another with a higher level of confidence than routers in separate groups.

Grouping autonomous systems requires extensions to routing protocols. When reporting distances, the values must be increased when passing across the boundary from one group to another. The technique, loosely called *metric transformation*, partitions distance values into three categories. For example, suppose routers within an autonomous system use distance values less than 128. We can make a rule that when passing distance information across an autonomous system boundary within a single group, the distances must be transformed into the range of 128 to 191. Finally, we can make a rule that when passing distance values across the boundary between two groups, the values must be transformed into the range of 192 to 254†. The effect of such transformations is obvious: for any given destination network, any path that lies entirely within the autonomous system is guaranteed to have lower cost than a path that strays outside the autonomous system. Furthermore, among all paths that stray outside the autonomous system, those that remain within the group have lower cost than those that cross group boundaries. The key advantage of metric transformations is that they allow each autonomous system to choose an IGP, yet make it possible for other systems to compare routing costs.

16.8 Gated: Inter-Autonomous System Communication

A mechanism has been created to provide an interface between autonomous systems. Known as *gated*‡, the mechanism understands multiple protocols (both IGPs and BGP), and ensures that policy constraints are honored. For example, *gated* can accept RIP messages and modify the local computer's routing table just like the *routed* program. It can also advertise routes from within its autonomous system using BGP. The rules gated follows allow a system administrator to specify exactly which networks *gated* may and may not advertise and how to report distances to those networks. Thus, although *gated* is not an IGP, it plays an important role in routing because it demonstrates that it is feasible to build an automated mechanism linking an IGP with BGP without sacrificing protection.

Gated performs another useful task by implementing metric transformations. Thus, it is possible and convenient to use gated between two autonomous systems as well as on the boundary between two groups of routers that each participate in an IGP.

16.9 The Open SPF Protocol (OSPF)

In Chapter 14, we said that a link state routing algorithm, which uses SPF to compute shortest paths, scales better than a distance-vector algorithm. To encourage the adoption of link state technology, a working group of the Internet Engineering Task Force has designed an interior gateway protocol that uses the link state algorithm. Called *Open SPF (OSPF)*, the new protocol tackles several ambitious goals.

• As the name implies, the specification is available in the published literature. Making it an open standard that anyone can implement without paying license fees has encouraged many vendors to support OSPF. Consequently, it has become a popular replacement for proprietary protocols.

†The term *autonomous confederation* has been used to describe a group of autonomous systems; boundaries of autonomous confederations correspond to transformations beyond 191.

‡The name *gated* is pronounced "gate d" from "gate daemon."

• OSPF includes *type of service routing*. Managers can install multiple routes to a given destination, one for each priority or type of service. When routing a datagram, a router running OSPF uses both the destination address and type of service field in an IP header to choose a route. OSPF is among the first TCP/IP protocols to offer type of service routing.

• OSPF provides *load balancing*. If a manager specifies multiple routes to a given destination at the same cost, OSPF distributes traffic over all routes equally. Again, OSPF is among the first open IGPs to offer load balancing; protocols like RIP compute a single route to each destination.

• To permit growth and make the networks at a site easier to manage, OSPF allows a site to partition its networks and routers into subsets called *areas*. Each area is self-contained; knowledge of an area's topology remains hidden from other areas. Thus, multiple groups within a given site can cooperate in the use of OSPF for routing even though each group retains the ability to change its internal network topology independently.

• The OSPF protocol specifies that all exchanges between routers can be *authenticated*. OSPF allows a variety of authentication schemes, and even allows one area to choose a different scheme than another area. The idea behind authentication is to guarantee that only trusted routers propagate routing information. To understand why this could be a problem, consider what can happen when using RIP1, which has no authentication. If a malicious person uses a personal computer to propagate RIP messages advertising low-cost routes, other routers and hosts running RIP will change their routes and start sending datagrams to the personal computer.

• OSPF includes support for host-specific, subnet-specific, and classless routes as well as classful network-specific routes. All types may be needed in a large internet.

• To accommodate multi-access networks like Ethernet, OSPF extends the SPF algorithm described in Chapter 14. We described the algorithm using a point-to-point graph and said that each router running SPF would periodically broadcast link status messages about each reachable neighbor. If K routers attach to an Ethernet, they will broadcast K^2 reachability messages. OSPF minimizes broadcasts by allowing a more complex graph topology in which each node represents either a router or a network. Consequently, OSPF allows every multi-access network to have a *designated gateway* (i.e., a *designated router*) that sends link status messages on behalf of all routers attached to the network; the messages report the status of all links from the network to routers attached to the network. OSPF also uses hardware broadcast capabilities, where they exist, to deliver link status messages.

• To permit maximum flexibility, OSPF allows managers to describe a virtual network topology that abstracts away from the details of physical connections. For example, a manager can configure a virtual link between two routers in the routing graph even if the physical connection between the two routers requires communication across a transit network.

• OSPF allows routers to exchange routing information learned from other (external) sites. Basically, one or more routers with connections to other sites learn information about those sites and include it when sending update messages. The message format distinguishes between information acquired from external sources and information acquired from routers interior to the site, so there is no ambiguity about the source or reliability of routes.

16.9.1 OSPF Message Format

Each OSPF message begins with a fixed, 24-octet header as Figure 16.7 shows:

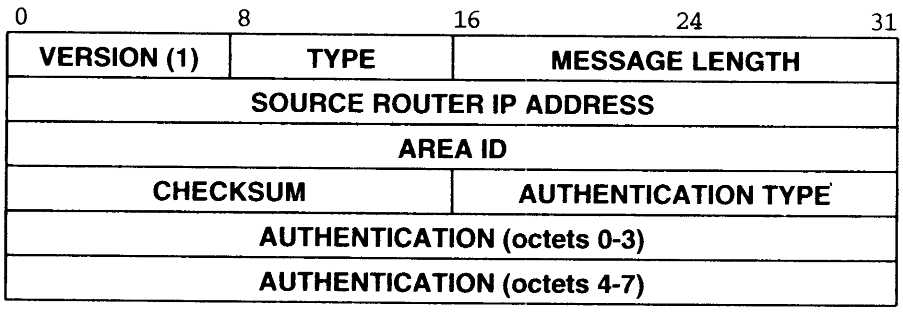


Figure 16.7 The fixed 24-octet OSPF message header.

Field *VERSION* specifies the version of the protocol. Field *TYPE* identifies the message type as one of:

Type	Meaning
1	Hello (used to test reachability)
2	Database description (topology)
3	Link status request
4	Link status update
5	Link status acknowledgement

The field labeled *SOURCE ROUTER IP ADDRESS* gives the address of the sender, and the field labeled *AREA ID* gives the 32-bit identification number for the area.

Because each message can include authentication, field *AUTHENTICATION TYPE* specifies which authentication scheme is used (currently, *0* means no authentication and *1* means a simple password is used).

16.9.2 OSPF Hello Message Format

OSPF sends *hello* messages on each link periodically to establish and test neighbor reachability. Figure 16.8 shows the format.

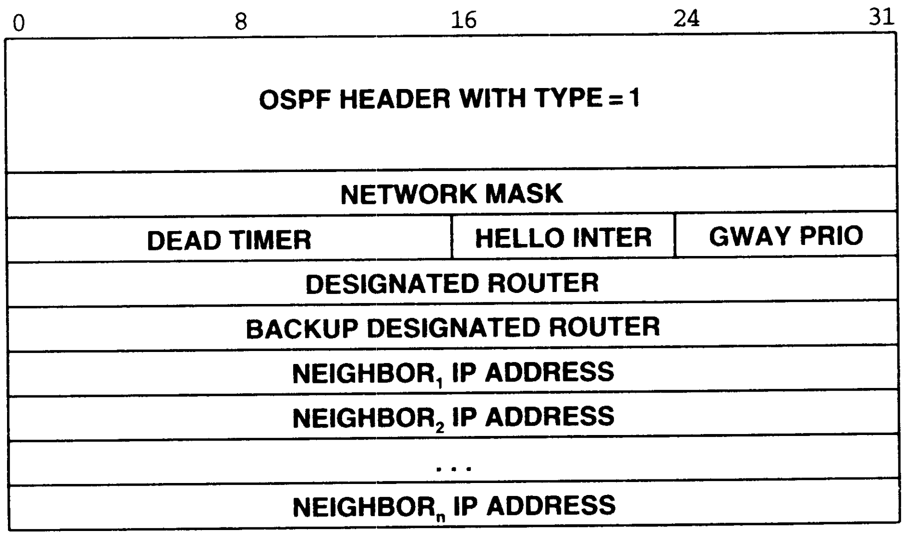


Figure 16.8 OSPF *hello* message format. A pair of neighbor routers exchanges these messages periodically to test reachability.

Field *NETWORK MASK* contains a mask for the network over which the message has been sent (see Chapter 10 for details about masks). Field *DEAD TIMER* gives a time in seconds after which a nonresponding neighbor is considered dead. Field *HELLO INTER* is the normal period, in seconds, between hello messages. Field *GWAY PRIO* is the integer priority of this router, and is used in selecting a backup designated router. The fields labeled *DESIGNATED ROUTER* and *BACKUP DESIGNATED ROUTER* contain IP addresses that give the sender's view of the designated router and backup designated router for the network over which the message is sent. Finally, fields labeled *NEIGHBOR$_i$ IP ADDRESS* give the IP addresses of all neighbors from which the sender has recently received hello messages.

16.9.3 OSPF Database Description Message Format

Routers exchange OSPF *database description* messages to initialize their network topology database. In the exchange, one router serves as a master, while the other is a slave. The slave acknowledges each database description message with a response. Figure 16.9 shows the format.

Because it can be large, the topology database may be divided into several messages using the *I* and *M* bits. Bit *I* is set to *1* in the initial message; bit *M* is set to *1* if additional messages follow. Bit *S* indicates whether a message was sent by a master (*1*) or by a slave (*0*). Field *DATABASE SEQUENCE NUMBER* numbers messages sequentially so the receiver can tell if one is missing. The initial message contains a random integer *R*; subsequent messages contain sequential integers starting at *R*.

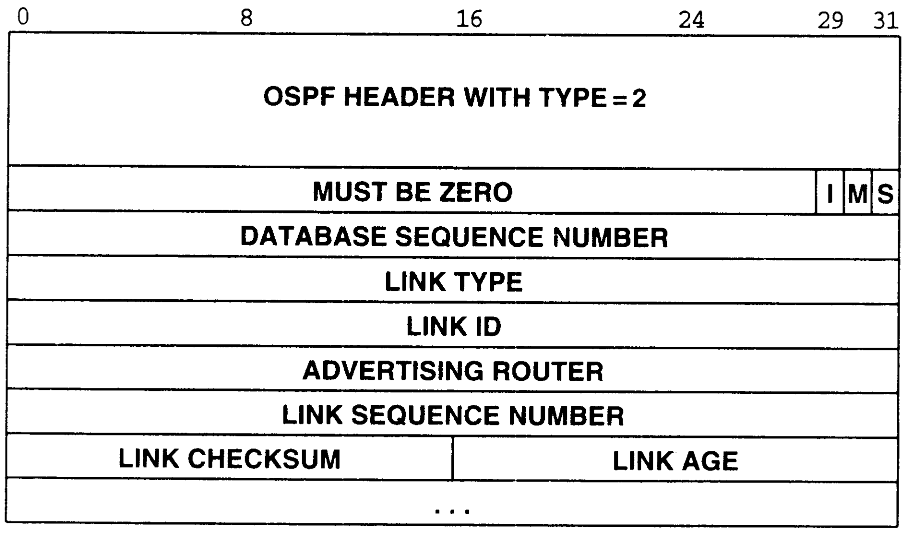


Figure 16.9 OSPF *database description* message format. The fields starting at *LINK TYPE* are repeated for each link being specified.

The fields from *LINK TYPE* through *LINK AGE* describe one link in the network topology; they are repeated for each link. The *LINK TYPE* describes a link according to the following table.

Link Type	Meaning
1	Router link
2	Network link
3	Summary link (IP network)
4	Summary link (link to border router)
5	External link (link to another site)

Field *LINK ID* gives an identification for the link (which can be the IP address of a router or a network, depending on the link type).

Field *ADVERTISING ROUTER* specifies the address of the router advertising this link, and *LINK SEQUENCE NUMBER* contains an integer generated by that router to ensure that messages are not missed or received out of order. Field *LINK CHECKSUM* provides further assurance that the link information has not been corrupted. Finally, field *LINK AGE* also helps order messages — it gives the time in seconds since the link was established.

16.9.4 OSPF Link Status Request Message Format

After exchanging database description messages with a neighbor, a router may discover that parts of its database are out of date. To request that the neighbor supply updated information, the router sends a *link status request* message. The message lists specific links as shown in Figure 16.10. The neighbor responds with the most current information it has about those links. The three fields shown are repeated for each link about which status is requested. More than one request message may be needed if the list of requests is long.

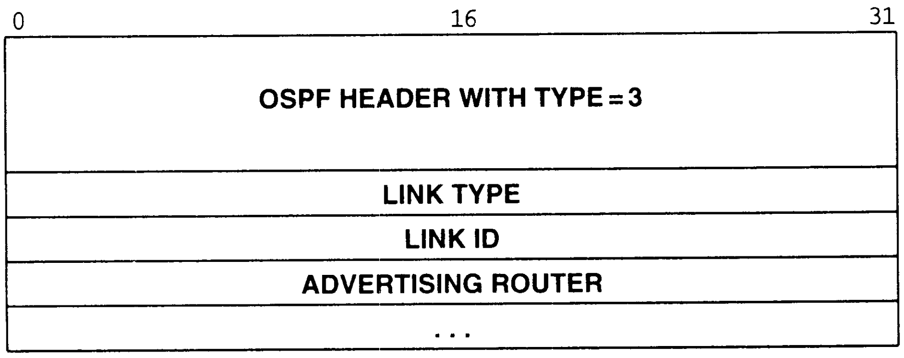


Figure 16.10 OSPF *link status request* message format. A router sends this message to a neighbor to request current information about a specific set of links.

16.9.5 OSPF Link Status Update Message Format

Routers broadcast the status of links with a *link status update* message. Each update consists of a list of advertisements, as Figure 16.11 shows.

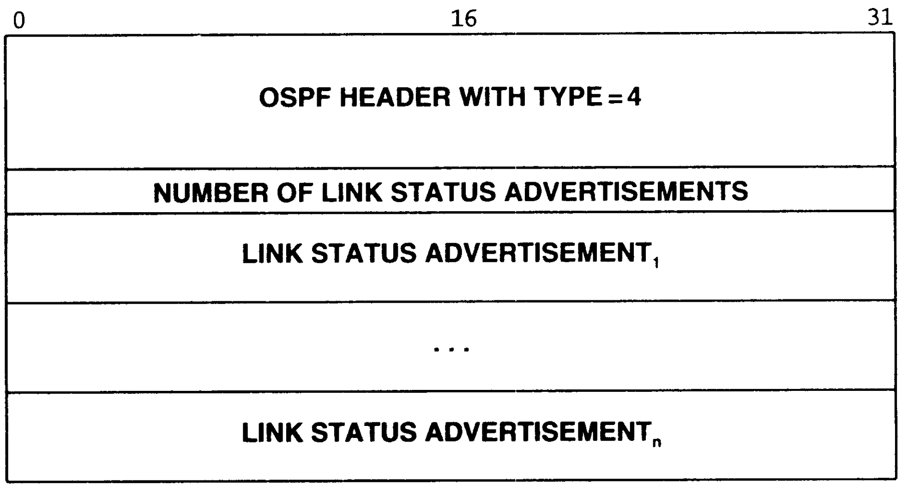

0
16
31
OSPF HEADER WITH TYPE = 4
NUMBER OF LINK STATUS ADVERTISEMENTS
LINK STATUS ADVERTISEMENT$_1$
. . .
LINK STATUS ADVERTISEMENT$_n$

Figure 16.11 OSPF *link status update* message format. A router sends such a message to broadcast information about its directly connected links to all other routers.

Each link status advertisement has a header format as shown in Figure 16.12. The values used in each field are the same as in the database description message.

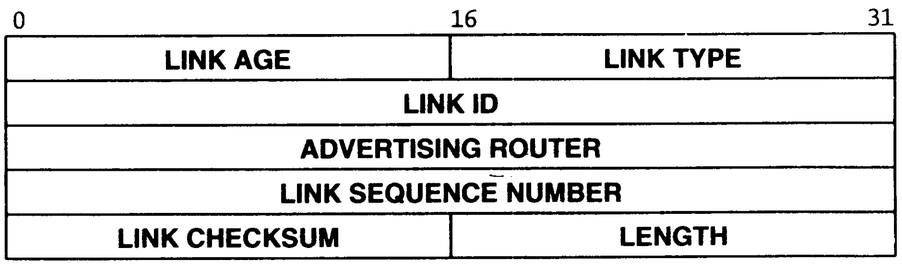


Figure 16.12 The format of the header used for all link status advertisements.

Following the link status header comes one of four possible formats to describe the links from a router to a given area, the links from a router to a specific network, the links from a router to the physical networks that comprise a single, subnetted IP network (see Chapter 10), or the links from a router to networks at other sites. In all cases, the *LINK TYPE* field in the link status header specifies which of the formats has been used. Thus, a router that receives a link status update message knows exactly which of the described destinations lie inside the site and which are external.

16.10 Routing With Partial Information

We began our discussion of internet router architecture and routing by discussing the concept of partial information. Hosts can route with only partial information because they rely on routers. It should be clear now that not all routers have complete information. Most autonomous systems have a single router that connects the autonomous system to other autonomous systems. For example, if the site connects to the global Internet, at least one router must have a connection that leads from the site to an ISP. Routers within the autonomous system know about destinations within that autonomous system, but they use a default route to send all other traffic to the ISP.

How to do routing with partial information becomes obvious if we examine a router's routing tables. Routers at the center of the Internet have a complete set of routes to all possible destinations that they learn from the routing arbiter system; such routers do not use default routing. In fact, if a destination network address does not appear in the routing arbiter database, only two possibilities exist: either the address is not a valid destination IP address, or the address is valid but currently unreachable (e.g., because routers or networks leading to that address have failed). Routers beyond those in ISPs at the center of the Internet do not usually have a complete set of routes; they rely on a default route to handle network addresses they do not understand.

Using default routes for most routers has two consequences. First, it means that local routing errors can go undetected. For example, if a machine in an autonomous system incorrectly routes a packet to an external autonomous system instead of to a local router, the external system will route it back (perhaps to a different entry point). Thus, connectivity may appear to be preserved even if routing is incorrect. The problem may not seem severe for small autonomous systems that have high speed local area networks, but in a wide area network, incorrect routes can be disastrous. Second, on the positive side, using default routes whenever possible means that the routing update messages exchanged by most routers will be much smaller than they would be if complete information had to be included.

16.11 Summary

Managers must choose how to pass routing information among the local routers within an autonomous system. Manual maintenance of routing information suffices only for small, slowly changing internets that have minimal interconnection; most require automated procedures that discover and update routes automatically. Two routers under the control of a single manager run an Interior Gateway Protocol, IGP, to exchange routing information.

An IGP implements either the distance-vector algorithm or the link state algorithm, which is known by the name Shortest Path First (SPF). We examined three specific IGPs: RIP, HELLO, and OSPF. RIP, a distance-vector protocol implemented by the UNIX program *routed*, is among the most popular. It uses split horizon, hold-down, and poison reverse techniques to help eliminate routing loops and the problem of counting to infinity. Although it is obsolete, Hello is interesting because it illustrates a distance-vector protocol that uses delay instead of hop counts as a distance metric. We discussed the disadvantages of delay as a routing metric, and pointed out that although heuristics can prevent instabilities from arising when paths have equal throughput characteristics, long-term instabilities arise when paths have different characteristics. Finally, OSPF is a protocol that implements the link status algorithm.

Also, we saw that the *gated* program provides an interface between an Interior Gateway Protocol like RIP and the Exterior Gateway Protocol, BGP, automating the process of gathering routes from within an autonomous system and advertising them to another autonomous system.

FOR FURTHER STUDY

Hedrick [RFC 1058] discusses algorithms for exchanging routing information in general and contains the standard specification for RIP1. Malkin [RFC 2453] gives the standard for RIP2. The HELLO protocol is documented in Mills [RFC 891]. Mills and Braun [1987] considers the problems of converting between delay and hop-count metrics. Moy [RFC 1583] contains the lengthy specification of OSPF as well as a discussion of the motivation behind it. Fedor [June 1988] describes *gated*.

EXERCISES

16.1 What network families does RIP support? Hint: read the networking section of the 4.3 BSD UNIX Programmer's Manual.

16.2 Consider a large autonomous system using an interior router protocol like HELLO that bases routes on delay. What difficulty does this autonomous system have if a subgroup decides to use RIP on its routers?

16.3 Within a RIP message, each IP address is aligned on a 32-bit boundary. Will such addresses be aligned on a 32-bit boundary if the IP datagram carrying the message starts on a 32-bit boundary?

16.4 An autonomous system can be as small as a single local area network or as large as multiple long haul networks. Why does the variation in size make it difficult to find a standard IGP?

16.5 Characterize the circumstances under which the split horizon technique will prevent slow convergence.

16.6 Consider an internet composed of many local area networks running RIP as an IGP. Find an example that shows how a routing loop can result even if the code uses "hold down" after receiving information that a network is unreachable.

16.7 Should a host ever run RIP in active mode? Why or why not?

16.8 Under what circumstances will a hop count metric produce better routes than a metric that uses delay?

16.9 Can you imagine a situation in which an autonomous system chooses *not* to advertise all its networks? Hint: think of a university.

16.10 In broad terms, we could say that RIP distributes the local routing table, while BGP distributes a table of networks and routers used to reach them (i.e., a router can send a BGP advertisement that does not exactly match items in its own routing table). What are the advantages of each approach?

16.11 Consider a function used to convert between delay and hop-count metrics. Can you find properties of such functions that are sufficient to prevent routing loops. Are your properties necessary as well? (Hint: look at Mills and Braun [1987].)

16.12 Are there circumstances under which an SPF protocol can form routing loops? Hint: think of best-effort delivery.

16.13 Build an application program that sends a request to a router running RIP and displays the routes returned.

16.14 Read the RIP specification carefully. Can routes reported in a response to a query differ from the routes reported by a routing update message? If so how?

16.15 Read the OSPF specification carefully. How can a manager use the virtual link facility?

16.16 OSPF allows managers to assign many of their own identifiers, possibly leading to duplication of values at multiple sites. Which identifier(s) may need to change if two sites running OSPF decide to merge?

16.17 Compare the version of OSPF available under 4BSD UNIX to the version of RIP for the same system. What are the differences in source code size? Object code size? Data storage size? What can you conclude?

16.18 Can you use ICMP redirect messages to pass routing information among *interior* routers? Why or why not?

16.19 Write a program that takes as input a description of your organization's internet, uses RIP queries to obtain routes from the routers, and reports any inconsistencies.

16.20 If your organization runs *gated*, obtain a copy of the configuration files and explain the meaning of each item.

17

Internet Multicasting

17.1 Introduction

Earlier chapters define the original IP classful addressing scheme and extensions such as subnetting and classless addressing. This chapter explores an additional feature of the IP addressing scheme that permits efficient multipoint delivery of datagrams. We begin with a brief review of the underlying hardware support. Later sections describe IP addressing for multipoint delivery and protocols that routers use to propagate the necessary routing information.

17.2 Hardware Broadcast

Many hardware technologies contain mechanisms to send packets to multiple destinations simultaneously (or nearly simultaneously). Chapter 2 reviews several technologies and discusses the most common form of multipoint delivery: *broadcasting*. Broadcast delivery means that the network delivers one copy of a packet to each destination. On bus technologies like Ethernet, broadcast delivery can be accomplished with a single packet transmission. On networks composed of switches with point-to-point connections, software must implement broadcasting by forwarding copies of the packet across individual connections until all switches have received a copy.

With most hardware technologies, a computer specifies broadcast delivery by sending a packet to a special, reserved destination address called the *broadcast address*. For example, Ethernet hardware addresses consist of 48-bit identifiers, with the all 1s address used to denote broadcast. Hardware on each machine recognizes the machine's hardware address as well as the broadcast address, and accepts incoming packets that have either address as their destination.

The chief disadvantage of broadcasting arises from its demand on resources — in addition to using network bandwidth, each broadcast consumes computational resources on all machines. For example, it would be possible to design an alternative internet protocol suite that used broadcast to deliver datagrams on a local network and relied on IP software to discard datagrams not intended for the local machine. However, such a scheme would be extremely inefficient because all computers on the network would receive and process every datagram, even though a machine would discard most of the datagrams that arrived. Thus, the designers of TCP/IP used unicast routing and address binding mechanisms like ARP to eliminate broadcast delivery.

17.3 Hardware Origins Of Multicast

Some hardware technologies support a second, less common form of multi-point delivery called *multicasting*. Unlike broadcasting, multicasting allows each system to choose whether it wants to participate in a given multicast. Typically, a hardware technology reserves a large set of addresses for use with multicast. When a group of machines want to communicate, they choose one particular *multicast address* to use for communication. After configuring their network interface hardware to recognize the selected multicast address, all machines in the group will receive a copy of any packet sent to that multicast address.

At a conceptual level, multicast addressing can be viewed as a generalization of all other address forms. For example, we can think of a conventional *unicast address* as a form of multicast addressing in which there is exactly one computer in the multicast group. Similarly, we can think of directed broadcast addressing as a form of multicasting in which all computers on a particular network are members of the multicast group. Other multicast addresses can correspond to arbitrary sets of machines.

Despite its apparent generality, multicasting cannot replace conventional forms because there is a fundamental difference in the underlying mechanisms that implement forwarding and delivery. Unicast and broadcast addresses identify a computer or a set of computers attached to one physical segment, so forwarding depends on the network topology. A multicast address identifies an arbitrary set of listeners, so the forwarding mechanism must propagate the packet to all segments. For example, consider two LAN segments connected by an adaptive bridge that has learned host addresses. If a host on segment *1* sends a unicast frame to another host on segment *1*, the bridge will not forward the frame to segment *2*. If a host uses a multicast address, however, the bridge will forward the frame. Thus, we can conclude:

> *Although it may help us to think of multicast addressing as a generalization that subsumes unicast and broadcast addresses, the underlying forwarding and delivery mechanisms can make multicast less efficient.*

17.4 Ethernet Multicast

Ethernet provides a good example of hardware multicasting. One-half of the Ethernet addresses are reserved for multicast — the low-order bit of the high-order octet distinguishes conventional unicast addresses (*0*) from multicast addresses (*1*). In dotted hexadecimal notation†, the multicast bit is given by:

$$01.00.00.00.00.00_{16}$$

When an Ethernet interface board is initialized, it begins accepting packets destined for either the computer's hardware address or the Ethernet broadcast address. However, device driver software can reconfigure the device to allow it to also recognize one or more multicast addresses. For example, suppose the driver configures the Ethernet multicast address:

$$01.5E.00.00.00.01_{16}$$

After the configuration, an interface will accept any packet sent to the computer's unicast address, the broadcast address, or that one multicast address (the hardware will continue to ignore packets sent to other multicast addresses). The next sections explain both how IP uses basic multicast hardware and the special meaning of the multicast address

17.5 IP Multicast

IP multicasting is the internet abstraction of hardware multicasting. It follows the paradigm of allowing transmission to a subset of host computers, but generalizes the concept to allow the subset to spread across arbitrary physical networks throughout the internet. In IP terminology, a given subset is known as a *multicast group*. IP multicasting has the following general characteristics:

- *Group address*. Each multicast group is a unique class *D* address. A few IP multicast addresses are permanently assigned by the Internet authority, and correspond to groups that always exist even if they have no current members. Other addresses are temporary, and are available for private use.
- *Number of groups*. IP provides addresses for up to 2^{28} simultaneous multicast groups. Thus, the number of groups is limited by practical constraints on routing table size rather than addressing.
- *Dynamic group membership*. A host can join or leave an IP multicast group at any time. Furthermore, a host may be a member of an arbitrary number of multicast groups.

†Dotted hexadecimal notation represents each octet as two hexadecimal digits with octets separated by periods; the subscript *16* can be omitted only when the context is unambiguous.

- *Use of hardware.* If the underlying network hardware supports multicast, IP uses hardware multicast to send IP multicast. If the hardware does not support multicast, IP uses broadcast or unicast to deliver IP multicast.
- *Inter-network forwarding.* Because members of an IP multicast group can attach to multiple physical networks, special *multicast routers* are required to forward IP multicast; the capability is usually added to conventional routers.
- *Delivery semantics.* IP multicast uses the same best-effort delivery semantics as other IP datagram delivery, meaning that multicast datagrams can be lost, delayed, duplicated, or delivered out of order.
- *Membership and transmission.* An arbitrary host may send datagrams to any multicast group; group membership is only used to determine whether the host receives datagrams sent to the group.

17.6 The Conceptual Pieces

Three conceptual pieces are required for a general purpose internet multicasting system:

1. A multicast addressing scheme
2. An effective notification and delivery mechanism
3. An efficient internetwork forwarding facility

Many goals, details, and constraints present challenges for an overall design. For example, in addition to providing sufficient addresses for many groups, the multicast *addressing scheme* must accommodate two conflicting goals: allow local autonomy in assigning addresses, while defining addresses that have meaning globally. Similarly, hosts need a *notification mechanism* to inform routers about multicast groups in which they are participating, and routers need a *delivery mechanism* to transfer multicast packets to hosts. Again there are two possibilities: we desire a system that makes effective use of hardware multicast when it is available, but also allows IP multicast delivery over networks that do not have hardware support for multicast. Finally a multicast *forwarding facility* presents the biggest design challenge of the three: our goal is a scheme that is both efficient and dynamic — it should route multicast packets along the shortest paths, should not send a copy of a datagram along a path if the path does not lead to a member of the group, and should allow hosts to join and leave groups at any time.

IP multicasting includes all three aspects. It defines IP multicast addressing, specifies how hosts send and receive multicast datagrams, and describes the protocol routers use to determine multicast group membership on a network. The remainder of the chapter considers each aspect in more detail, beginning with addressing.

17.7 IP Multicast Addresses

We said that IP multicast addresses are divided into two types: those that are permanently assigned, and those that are available for temporary use. Permanent addresses are called *well-known*; they are used for major services on the global Internet as well as for infrastructure maintenance (e.g., multicast routing protocols). Other multicast addresses correspond to ***transient multicast groups*** that are created when needed and discarded when the count of group members reaches zero.

Like hardware multicasting, IP multicasting uses the datagram's destination address to specify that a particular datagram must be delivered via multicast. IP reserves class *D* addresses for multicast; they have the form shown in Figure 17.1.

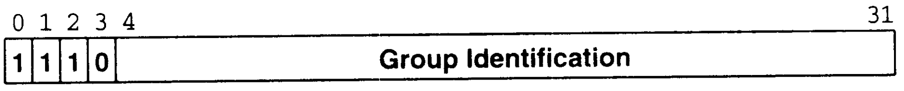


Figure 17.1 The format of class D IP addresses used for multicasting. Bits *4* through *31* identify a particular multicast group.

The first *4* bits contain *1110* and identify the address as a multicast. The remaining *28* bits specify a particular multicast group. There is no further structure in the group bits. In particular, the group field is not partitioned into bits that identify the origin or owner of the group, nor does it contain administrative information such as whether all members of the group are on one physical network.

When expressed in dotted decimal notation, multicast addresses range from

224.0.0.0 through 239.255.255.255

However, many parts of the address space have been assigned special meaning. For example, the lowest address, 224.0.0.0, is reserved; it cannot be assigned to any group. Furthermore, the remaining addresses up through 224.0.0.255 are devoted to multicast routing and group maintenance protocols; a router is prohibited from forwarding a datagram sent to any address in that range. Figure 17.2 shows a few examples of permanently assigned addresses.

Address	Meaning
224.0.0.0	Base Address (Reserved)
224.0.0.1	All Systems on this Subnet
224.0.0.2	All Routers on this Subnet
224.0.0.3	Unassigned
224.0.0.4	DVMRP Routers
224.0.0.5	OSPFIGP All Routers
224.0.0.6	OSPFIGP Designated Routers
224.0.0.7	ST Routers
224.0.0.8	ST Hosts
224.0.0.9	RIP2 Routers
224.0.0.10	IGRP Routers
224.0.0.11	Mobile-Agents
224.0.0.12	DHCP Server / Relay Agent
224.0.0.13	All PIM Routers
224.0.0.14	RSVP-Encapsulation
224.0.0.15	All-CBT-Routers
224.0.0.16	Designated-Sbm
224.0.0.17	All-Sbms
224.0.0.18	VRRP
224.0.0.19 through 224.0.0.255	Unassigned
224.0.1.21	DVMRP on MOSPF
224.0.1.84	Jini Announcement
224.0.1.85	Jini Request
239.192.0.0 through 239.251.255.255	Scope restricted to one organization
239.252.0.0 through 239.255.255.255	Scope restricted to one site

Figure 17.2 Examples of a few permanent IP multicast address assignments. Many other addresses have specific meanings.

We will see that two of the addresses in the figure are especially important to the multicast delivery mechanism. Address 224.0.0.1 is permanently assigned to the *all systems group*, and address 224.0.0.2 is permanently assigned to the *all routers group*. The *all systems* group includes all hosts and routers on a network that are participating in IP multicast, whereas the *all routers* group includes only the routers that are participating. In general, both of these groups are used for control protocols and not for the