

Routing and Flow Control

by

Rene L. Cruz

Dept. of Electrical & Computer Engineering

University of California, San Diego

La Jolla, CA 92093-0407

Introduction

There are a large number of issues relating to routing and flow control, and we cannot hope to discuss the subject comprehensively in this short article. Instead, we outline generic concepts and highlight some important issues from a high level perspective. We limit our discussion to the context of **packet switched networks**.

Routing and flow control are concerned with how information packets are transported from their source to their destination. Specifically, routing is concerned with where packets travel, i.e. the paths traveled by packets through the network. Flow control is concerned with when packets travel, i.e. the timing and rate of packet transmissions.

The behavior of routing and flow control can be characterized by a number of inter-related performance measures. These performance measures can be categorized into two groups: user metrics and network metrics. User metrics characterize performance from the perspective of a single user. Common user metrics are throughput (rate of information transfer), average delay, maximum delay, delay jitter (variations in delay), and packet loss probability. Network metrics characterize performance from the perspective of the network as a whole, and all users of it. Common network metrics are network throughput, number of users that can be supported, average delay (averaged over all users), and buffer requirements. Fairness issues may also arise from the perspective of the network.

The goals of routing and flow control are to balance these different performance measures. Optimization of user metrics are often at the expense of network metrics, and vice versa. For example, if the number of

users allowed to access the network is too large, the quality of service seen by some users might fall below acceptable levels.

In reality, it is often even difficult to optimize a single performance measure, due to the uncertainty of demand placed on the network. Statistical models can be used, but the appropriate parameters of the models are often unknown. Even with an accurate model of demand, the problem of predicting performance for a given routing or flow control strategy is often intractable. For this reason, routing and flow control algorithms are typically based on several simplifying assumptions, approximations, and heuristics.

Before we discuss routing and flow control in more detail, we begin by classifying different modes of operation common in packet switched networks.

Connection-oriented and connection-less protocols, services, and networks

In a connection-oriented protocol or service between two network terminals, the two terminals interact with one another prior to the transport of data, to set up a "connection" between them. The interaction prior to the exchange of data is required, for example, to identify the terminals to each other and to define the rules governing the exchange of data. For example, users of the telephone network engage in a connection-oriented protocol in the sense that a phone number must first be dialed, and typically the parties involved must identify themselves to each other, before the essential conversation can begin. The Transport Control Protocol (TCP) used on the Internet is an example of a connection-oriented protocol. Examples of connection oriented services are "rlogin" and "telnet," which can be built upon TCP. Connection-oriented protocols and services typically involve the exchange of data in both directions. In addition, connection-oriented protocols are frequently used to provide reliable transport of data packets between network endpoints, whereby the network endpoints continue to interact during the data exchange phase in order to manage retransmission of lost or errored packets.

In connection-less protocols or services between two network terminals, no interaction between the terminals takes place prior to the exchange of data. For example, when a letter is mailed over the postal

network, a connection-less protocol is used in the sense that the recipient of the letter need not interact with the sender ahead of time. An example of a connection-less protocol is the current Internet Protocol (IP) used to transport packets on the Internet. Electronic mail provides a connection-less service, in the same sense that the postal network provides a connection-less service. Because of the lack of interaction between terminals engaging in a connection-less protocol, retransmission of lost or errored packets is not possible, and hence connection-less protocols are not considered intrinsically reliable. On the other hand, connection-less protocols are often appropriate in environments where lost or errored packets are sufficiently rare.

A connection-oriented network is a network specifically designed to support connection-oriented protocols. When a connection request is made to a connection-oriented network, the network establishes a path between the source and destination terminal. The switching nodes are then configured to route packets from the connection along the established path. Each switching node is aware of the connections that pass through it and may reserve resources (e.g. buffers and bandwidth) on a per-connection basis. A packet switched network that is connection-oriented is also referred to as a **virtual circuit switched network**. Asynchronous Transfer Mode (ATM) networks are an example of virtual circuit switched networks.

A connection-less network is a network designed to support connection-less protocols. Each node of a connection-less network typically knows how to route packets to each possible destination, since there is no prior coordination between terminals before they communicate. Connection-less networks are sometimes referred to as **datagram networks**.

The terminology defined above is not standard, and indeed there does not appear to be widespread agreement on terminology. We have made a distinction between a connection-oriented protocol and a connection-oriented network, and a distinction between a connection-less protocol and a connection-less network. The reason for these distinctions is because of the many ways in which protocols and networks can be layered. In particular, connection oriented protocols can be built upon connection-less networks and

protocols, and connection-less protocols can be built upon connection-oriented networks and protocols. For example, TCP is in widespread use on end-hosts in the Internet, but the Internet itself is inherently a connection-less network which utilizes IP. Another example that has received attention recently is in using an ATM network (connection-oriented) as a lower layer transport mechanism to support the IP protocol between Internet nodes. This may be considered somewhat odd, since IP will in turn often support the connection-oriented TCP.

Packet switched networks can be further classified as lossy or loss-less according to whether or not network nodes will sometimes be forced to drop (destroy) packets without forwarding them to their intended destination. Connection-less networks are inherently lossy, since the lack of coordination of data sources creates the possibility that buffers inside the network will overflow. Connection-oriented networks can be designed to be loss-less. On the other hand, connection-oriented networks can also be lossy, depending on if and how flow control is exercised.

Routing in Datagram Networks

We will first confine our discussion routing to the context of connection-less networks. A common approach to routing in connection-less networks is for each node in the network to have enough intelligence to route packets to every possible destination in the network.

A common approach to route computation is shortest path routing. Each link in the network is assigned a weight, and the length of a given path is defined as the sum of the link weights along the path. The weight of a link can reflect the desirability of using that link and might be based on reliability or queueing delay estimates on the link, for example. However, another common choice for link weights is unity, and shortest paths in this case correspond to minimum hop paths.

We now briefly review some popular algorithms for computing shortest paths. Suppose there are n nodes in the network, labeled $1, 2, \dots, n$. The label (i, j) is assigned to a link that connects node i to node j . The links are assumed to be oriented, i.e. link (i, j) is distinct from (j, i) . We assume that if there is a link $(i,$

j) in the network, then there is also a link (j, i) in the network. We say that node i is a neighbor of node j if the link (i, j) exists. Let $N(i)$ be the set of nodes for which node i is a neighbor. Let w_{ij} denote the weight assigned to link (i, j) , and define $w_{ij} = +\infty$ if node i is not a neighbor of node j . We assume that $w_{ij} > 0$ for all i and j . Formally, a path from node i_1 to node i_k is a sequence of nodes (i_1, i_2, \dots, i_k) . The length of such a path is the sum of the weights of links between successive nodes, and the path is said to have $k - 1$ hops. We assume that the network is strongly connected, i.e. there exists a path between all ordered pairs of nodes with finite length.

Consider the problem of finding shortest (i.e. minimum length) paths from each node to a given destination node. Since each node in the network is a potential destination, this problem will be solved separately for each destination. Without loss of generality, assume that the destination node is node 1. The distance from a node i to the destination node is defined to be the length of the shortest path from node i to the destination node, and is denoted as D_i . Define $D_1 = 0$.

Consider a shortest path from a node i to the destination: $(i, j, k, \dots, 1)$. Note that the subpath which begins at node j , $(j, k, \dots, 1)$, must be a shortest path from node j to the destination, for otherwise there exists a path from node i to the destination which is shorter than $(i, j, k, \dots, 1)$. Thus, this subpath has length D_j , and $D_i = w_{ij} + D_j$. Furthermore, since D_i is the length of a shortest path, we have $D_i \leq w_{ik} + D_k$ for any k . We thus have $D_1 = 0$ and

$$D_i = \min\{ w_{ij} + D_j : 1 \leq j \leq n \}, \quad 1 < i \leq n$$

This is known as Bellman's equation, and forms the basis of an optimization technique known as dynamic programming. Let $\mathbf{D} = (D_1, D_2, \dots, D_n)$ denote the vector of shortest path lengths, and let f denote the vector valued mapping in Bellman's equation. With these definitions, Bellman's equation says that \mathbf{D} is a fixed point of the mapping f , i.e. $\mathbf{D} = f(\mathbf{D})$. It can be shown that under our assumption of positive link weights, there is only one solution \mathbf{x} to the equation $\mathbf{x} = f(\mathbf{x})$, i.e. \mathbf{D} is the unique solution to Bellman's equation.

The Bellman-Ford algorithm for shortest paths finds the solution to Bellman's equation iteratively. Specifically, let $\mathbf{D}^0 = (0, \infty, \infty, \dots, \infty)$. In the first iteration of the Bellman-Ford algorithm, $\mathbf{D}^1 = f(\mathbf{D}^0)$ is computed. In the m^{th} iteration, $\mathbf{D}^m = f(\mathbf{D}^{m-1})$ is computed. It can be shown by induction on m that \mathbf{D}^m is the vector of shortest path lengths, where the paths are constrained to have at most m hops. Since link weights are positive, any shortest path visits a node at most once, and hence any shortest path has at most $n-1$ hops. It thus follows that the Bellman-Ford algorithm converges within $n-1$ iterations. In other words we have $f(\mathbf{D}^{n-1}) = \mathbf{D}^{n-1} = \mathbf{D}$. The shortest paths can be generated from \mathbf{D} by examining the argument of the minimum in Bellman's equation. Specifically, for each i , let $\text{Next}(i) = j$, where j is an argument of the minimum in the equation for D_i , i.e. $D_i = w_{ij} + D_j$. With this definition, $\text{Next}(i)$ is clearly the first node along a shortest path from node i to the destination. The worst case time complexity of the Bellman-Ford algorithm is easily seen to be $O(n^3)$.

The Bellman-Ford shortest path algorithm is sometimes called a distance vector routing algorithm. The algorithm lends itself easily to a distributed, asynchronous implementation that forms the basis for many commonly used routing algorithms. In such an implementation, each node i is responsible for computing the distance from itself to the destination, D_i . Each node initially has an estimate of D_i . These initial estimates may be arbitrary, except the destination itself always knows the correct distance from itself to itself, i.e. node 1 knows that $D_1 = 0$. Each node i sends, or "advertises", to its neighbors the estimate of D_i that it currently has. This advertisement is either done periodically, or whenever the estimate changes. Whenever a node receives an estimate from a neighbor, it updates its own estimate in accordance with Bellman's equation. Specifically, if $D_i(t)$ is the estimate of D_i at time t and if $D_j(t - \tau_j)$ is the estimate of D_j from node j that was most recently advertised to node i , then $D_i(t) = \min \{w_{ij} + D_j(t - \tau_j) : j \in N(i)\}$.

Note that the minimum above can be taken over only nodes j which are neighbors of node i , since $w_{ij} = \infty$ if node i is not a neighbor of node j . It can be shown that each node's estimate converges to the actual distance to the destination under essentially arbitrary assumptions regarding propagation delays of advertisements, and delays associated with computing estimates [Bertsekas and Gallager, 1992].

In the above discussion we assumed that the link weights were constant. In practice, however, the link weights may change with time. For example, this may be as a result of link failures, or changing traffic characteristics on the links. If the link weights change slowly enough, the distributed asynchronous Bellman-Ford algorithm will track the shortest paths correctly as the link weights change. However, if the link weights change fast enough so that the algorithm doesn't have a chance to converge, problems may arise. For example, if link weights change quickly in response to changing traffic characteristics, it may be possible for packets to travel around in loops indefinitely. In such a scenario, the link weights oscillate as the traffic pattern on the links change due to oscillating estimates of shortest paths [Bertsekas and Gallager, 1992].

Another popular approach to finding shortest paths is for each node to independently compute the shortest paths. Each node i broadcasts the link weight w_{ij} for all outgoing links (i,j) to all other nodes. Each node then has complete information about the network and can compute shortest paths independently. Networks which use this approach are sometimes said to use link state routing. Each node could use the Bellman-Ford algorithm as described above to compute shortest paths. However, with link state routing it is common to use Dijkstra's algorithm for shortest paths, described below.

Dijkstra's algorithm for finding shortest paths first finds the closest node to the destination node (i.e. the node i other than the destination node with the smallest value of D_i), then the second closest node, then the third closest node, and so on. Let us assume again without loss of generality that node 1 is the destination node. Dijkstra's algorithm is summarized below:

Initialization : $m \leftarrow 1$, $P_m \leftarrow \{1\}$, $D_1 \leftarrow 0$.

For all i not in P_m :

$$X_i \leftarrow w_{i1} + D_1$$

$$\text{Next}(i) \leftarrow 1$$

Iteration :

$$i^* \leftarrow \arg \min \{ X_i : i \text{ not in } P_m \}$$

$$P_{m+1} \leftarrow P_m \cup \{i^*\}$$

$$D_{i^*} \leftarrow X_{i^*}$$

For all k not in P_{m+1} :

 If $X_k > w_{ki^*} + D_{i^*}$ then

$$X_k \leftarrow w_{ki^*} + D_{i^*}$$

$$\text{Next}(k) \leftarrow i^*$$

$$m \leftarrow m+1$$

If $m=n$ then terminate. Otherwise, execute iteration again.

The set P_m consist of the m closest nodes to the destination, including the destination. In other words, for each m we have $|P_m| = m$, and $D_i \geq D_j$ if j belongs to P_m and i does not belong to P_m . To see why Dijkstra's algorithm works, assume after executing the iteration m times that P_{m+1} is the set of m closest nodes, that $X_k = \min \{ w_{kj} + D_j : j \in P_{m+1} \}$, and that D_j has been computed correctly for all $j \in P_{m+1}$. Clearly this is true for $m=0$, and we now show by induction that it is true for all m . Consider the $(m+1)^{\text{st}}$ iteration. Suppose that node i^* is a node that could be added to P_{m+1} to yield P_{m+2} with the desired property, and that node k is an arbitrary node that cannot be added to P_{m+1} to yield P_{m+2} with the desired property. Note that $D_k > D_{i^*}$. By the induction hypothesis and the fact that all link weights are positive, it thus follows from Bellman's equation that $D_{i^*} = X_{i^*}$. Furthermore, we have $X_k \geq D_k > D_{i^*} = X_{i^*}$. Thus, in the $(m+1)^{\text{st}}$ iteration, node i^* is chosen correctly to form P_{m+2} and $D_{i^*} = X_{i^*}$. Furthermore, we see that X_k is updated correctly to preserve the induction hypothesis.

It is seen that the time complexity of Dijkstra's algorithm is $O(n^2)$, which is less than the worst case time complexity of $O(n^3)$ for the Bellman-Ford algorithm. The description of Dijkstra's algorithm above can also easily be adapted to address the computation of shortest paths from a given source node to all possible destinations, which makes it well suited for link state routing.

Deflection routing [Maxemchuk, 1987] has been proposed for high speed networks. With deflection routing, each node attempts to route packets along a shortest path. However, if a link becomes "congested," a node will intentionally send a packet to a node which is *not* along a shortest path. Such a packet is said to be "deflected." If the number of links incoming to a node is equal to the number of outgoing links, and the capacity of all links is the same, packet buffering can be essentially eliminated with deflection routing. It is possible for packets to circulate in loops with deflection routing. However, with a suitable priority structure, delay can be bounded. Deflection routing has also been proposed within high speed packet switches [Cruz and Tsai, 1996].

In another approach to routing, sometimes called optimal routing, the rate of traffic flow between each possible source-destination pair is estimated. Suppose that $r(w)$ is the estimated rate (bits/second) of traffic flow for a given source-destination pair w . A number of possible paths for routing traffic between each source destination pair are identified, say path 1 through P_w for source-destination pair w . The total flow $r(w)$ is allocated among all the P_w paths according to a vector $(f_1, f_2, \dots, f_{P_w})$ such that $r(w) = f_1 + f_2 + \dots + f_{P_w}$, and such that f_p flows on path p . Typically, bifurcation is allowed, so that f_p may be positive for more than one path p . The path flow allocation for all source-destination pairs is determined simultaneously, in order to minimize some cost function. Typically the cost function is of the form $\sum C_e(f_e)$, where the sum is taken over all links e in the network and f_e denotes the total flow over all paths which cross link e . A common choice for $C_e(f_e)$ is $C_e(f_e) = f_e / (C - f_e)$, where C is the capacity in bits/second of link e . This would correspond to the average delay on link e if the output queue for link e were an M/M/1 queue. Nonlinear optimization techniques can be used to optimize the path flow allocations with respect to the cost function. The disadvantage of the optimal routing approach is that it is difficult to estimate the rates $r(w)$ a priori. In addition, the choice of the cost function to minimize is somewhat arbitrary.

Typically in connection-less networks the network nodes will determine where to route a packet so that it gets forwarded to its destination. In such an environment, the source places the address of the destination

in a **header field** of each packet. Upon receiving a packet, a network node examines the destination address, and uses this to index into a lookup table to determine the outgoing link that the packet should be forwarded on.

In contrast, source routing describes the situation where the route that a packet travels is determined by the source. In this case, the source explicitly specifies what route the packet should take within a header field of the packet. Typically this specification consists of the list of node addresses along the path that the packet should travel. This eliminates the need for large lookup tables within each network node. For example, link state routing can be used, where the sources compute shortest paths.

Routing in Virtual Circuit Switched Networks

In connection-oriented networks, sources make connection requests to the network, transmit data to the destinations through the connection, and then terminate the connection. Since connection requests are distributed in time, a routing decision is typically made for each connection request as it is made. In other words, the sequence of connection requests that will be made is unknown, and a routing decision for a connection is not made until the connection request occurs.

Shortest path routing is still a viable approach in connection-oriented networks. The network nodes calculate shortest paths to all possible destinations. When a connection request is made, the connection request is forwarded along a shortest path to the destination, and the switching nodes along the path can determine if they have the resources (e.g. bandwidth, buffers) to support the connection request. If all network nodes determine that they can support the request, the network accepts the connection, link weights are updated to reflect the newly formed connection, and data for the connection is forwarded along the established path. If a network node along the shortest path is not able to support the connection request, the network would then reject the connection request.

It is possible for the network to attempt to route a connection request along another path if the first attempted path fails. However, this should be done carefully, since routing a connection on a path with

many hops may use up bandwidth that could be better used to support other connection requests. **Trunk reservation** techniques may be used to advantage in this regard.

Alternatively, the sources might be responsible for specifying a desired path in a connection request, which would be a form of source routing. Obviously in this case, the sources would need knowledge of the network topology in order to make intelligent choices for the routes they request.

In connection-oriented networks, the source need not place the destination address into each packet. Rather, it could put a virtual circuit identifier (VCI) in each packet, which labels the connection that the packet belongs to. This can result in a savings of bandwidth if there are a large number of potential destinations, and a comparatively smaller number of connections passing through each link. A given connection can have a different VCI assigned on every link along the path assigned to it. When a connection is set up through a node, the node assigns a VCI to the connection to be used on the outgoing link, which may be different from the VCI used for the connection on the incoming link. The node then establishes a binding between the VCI used on the incoming link, the VCI assigned on the outgoing link, as well as the appropriate output link itself. Upon receiving a packet, the network node reads the VCI used on the incoming link and uses this as an index into a lookup table. In the lookup table, the node determines the appropriate outgoing link to forward the packet on, as well as the new VCI to use. This is sometimes referred to as VCI translation.

Hierarchical Routing

In a network with a very large number of nodes, lookup tables for routing can get very large. It is possible to reduce the size of the lookup tables, as well as the complexity of route computation, by using a hierarchical routing strategy. One approach to a hierarchical routing strategy is as follows. A subset of the network nodes are identified as "backbone" nodes. A link connecting two backbones is called a backbone link. The set of backbone nodes and backbone links is called the backbone network. The non-backbone nodes are partitioned into clusters of nodes called domains. Each backbone node belongs to a distinct domain, and is called the "parent" of all nodes in the domain. The nodes within a domain are typically geographically close to one another. Routes between pairs of nodes in the same domain are constrained

not to go outside the domain. Routes between pairs of nodes in different domains are constrained to be a concatenation of a path completely within the source domain, a path completely within the backbone network, and a path completely within the destination domain. Routes are computed at two levels, the backbone level and the domain level. Route computations at the backbone level need only information about the backbone network and can proceed independently of route computations at the domain level. Conversely, route computations at the domain level need only information about the specific domain for which routes are computed. Because of this decoupling, the problem of route computation and packet forwarding can be considerably simplified in large networks.

With a hierarchical routing approach it is often convenient to have a hierarchical addressing strategy as well. For example, for the hierarchical routing strategy described above, a node address could be grouped into two parts, the domain address, and the address of the node within the domain. Packet forwarding mechanisms at the backbone level need only look at the domain part of the node address. Of course, it is possible to have hierarchical routing strategies with more than two levels as described above.

Flow Control in Datagram Networks

Since sources in a connection-less network do not interact with the network prior to the transmission of data, it is possible that the sources may overload the network, causing buffers to overflow and packets to be lost. The sources may become aware that their packets have been lost, and retransmit the lost packets as a result. In turn, it is possible that the retransmission of packets will cause more buffers to overflow, causing more retransmissions, and so on; The network may be caught in a "traffic jam" where the network throughput decreases to an unacceptable level. It is also possible that buffers may overflow at the destinations if the sources send data too fast. The objective of flow control in connection-less networks is to throttle the sources, but only as necessary, so that rate of packet loss due to buffer overflow is at an acceptably low level.

A number of schemes have been proposed for flow control in connection-less networks. One technique that has been proposed is that network nodes which experience congestion (e.g. buffer overflow) should send the network nodes causing the congestion special control packets, which in effect request that the

offending nodes decrease the rate at which packets are injected into the network. This has been implemented in the Internet, and the control packets are called source quench packets. A practical problem with this approach is the lack of accepted standards that specify under what conditions source quench packets are sent, and that specify exactly how network nodes should react to receiving a source quench packet.

We now focus our discussion on a common flow control technique used on connection-less networks, whereby the network carries traffic predominately from connection-oriented protocols, and window flow control is exercised within the connection oriented protocols.

Window flow control is a technique where a source may have to wait for acknowledgements of packets previously sent before sending new packets. Specifically, a "window size" W is assigned to a source, and the source can have at most W unacknowledged packets outstanding. Typically, window flow control is exercised end-to-end, and the acknowledgements are sent back to the source by the destination as soon as it is ready to receive another packet. By delaying acknowledgements, the destination can slow down the source to prevent buffer overflow within the destination.

Another motivation behind window flow control is that if the network becomes congested, packet delays will increase. This will slow down the delivery of acknowledgements back to the source, which will cause the source to decrease its rate of transmissions. The window size W is typically chosen large enough so that in the absence of congestion, the source will not be slowed down by waiting for acknowledgements to arrive.

Although it is possible to prevent buffer overflow by choosing the window sizes small enough and appropriately managing the creation of connections, this is typically not how datagram networks operate. In many cases, packet loss due to buffer overflow is possible, and window flow control is combined with an **ARQ protocol** to manage the retransmission of lost packets. In particular, the source will wait only a limited time for an acknowledgement to arrive. After this amount of time passes without an

acknowledgement arriving, the source will "timeout" and assume that the packet has been lost, causing one or more packets to be retransmitted. The amount of time that the source waits is called the timeout value.

As mentioned above, retransmissions should be carefully controlled to prevent the network throughput from becoming unacceptably low. We now provide an explanation of how this is commonly handled within the current Internet, specifically within the TCP protocol. The explanation is oversimplified, but should give a rough idea of how congestion control mechanisms have been built into the TCP protocol. This is the primary method of flow control within the current Internet.

The first idea [Jain, 1986] is that a timeout indicates congestion, and sources should wait for the congestion to clear and not add to it. In the TCP protocol [Jacobson, 1988], each timeout causes the timeout value to be doubled. This is called "exponential backoff," and is similar to what happens on Ethernet networks. In practice, so that the timeout values do not get unacceptably large, exponential backoff is stopped when the timeout value reaches a suitable threshold. After acknowledgements begin to arrive at the source again, the source can reset the timeout value appropriately.

In addition to exponential backoff, the TCP protocol dynamically adjusts the window size in response to congestion. In particular, a congestion window W' is defined. Initially W' is set to the nominal window size W which is appropriate for non-congestion conditions. After each timeout, W' is reduced by a factor of two, until $W' = 1$. This is called "multiplicative decrease."

The window size used by the TCP protocol is at most W' , and could be less. The idea is that after a period of congestion, a large number of sources could still retransmit a large number of packets and cause another period of congestion. To prevent this, each time a timeout occurs, the window size is reset to one. Hence, a source is then allowed to send one packet. If the source receives an acknowledgement of this packet, it increases the window size by one, and hence is then allowed to send two packets. In general, the source will increase the window size by one for each acknowledgement it receives, until the window size

reaches the congestion window size W' . This is called "slow start," and is also used for new connections. After the window size reaches W' , the window size is increased by one for every round trip delay, assuming no timeouts occur, until the original window size appropriate for non-congestion conditions is reached. This has been called "slower start."

Flow Control in Virtual Circuit Switched Networks

In connection-oriented networks, flow control, in a broad sense, can be exercised at two different levels. At a coarse level, when a connection request is made, the network may assess whether it has the resources to support the requested connection. If the network decides that it cannot support the requested connection, the request is blocked (rejected). This is commonly called admission control. A connection request may carry information such as the bandwidth of the connection that is required, which the network can use to assess its ability to carry the connection. Some sources do not formally require any specified bandwidth from the network, but would like as high a bandwidth as possible. Such sources are not acutely sensitive to delay; As an example, a file transfer might be associated with such a source. These types of sources have been recently named Available Bit Rate (ABR) sources. If all sources are ABR, it may be unnecessary for the network to employ admission control.

At a finer level, when a connection is made, flow control may be exercised at the packet level. In particular, the rate as well as the "burstiness" of sources may be controlled by the network. We focus our discussion on flow control entirely at the packet level. Proposals have also been made to exercise flow control at an intermediate level, on groups of packets called "bursts," but we do not discuss that here.

We first discuss flow control for ABR sources. Given that the bandwidth allocated to an ABR source is variable, the issue of fairness arises. There are many possible ways to define fairness. One popular definition of fairness is max-min fairness, which is defined as follows. Each link in the network has a given maximum total bandwidth that is allocated to ABR sources using that link. Each ABR source has an associated path through the network that is used to deliver data from the source to the associated destination. An allocation of bandwidths to the ABR sources is called max-min fair if the minimum

bandwidth that is allocated is as large as possible. Given that constraint, the next lowest bandwidth that is allocated is as large as possible, and so on.

A simple algorithm to compute the max-min fair allocation of bandwidths to the ABR sources is as follows. A link is said to be saturated if the sum of the bandwidths of all ABR sources using that link equals the total bandwidth available to ABR sources on that link. Each ABR source is initially assigned a bandwidth of zero. The bandwidth of all ABR sources is increased equally until a link becomes saturated. The bandwidth allocated to each ABR source which crosses a saturated link is then frozen. The bandwidth of all remaining ABR sources are then increased equally until another link becomes saturated, and the bandwidth of each ABR source which crosses a newly saturated link is then frozen. The algorithm repeats this process until the bandwidth of each ABR source is frozen.

One way to impose flow control on ABR sources that has been proposed is to use window flow control for each ABR connection at the hop level. That is, each ABR connection is subject to a window flow control algorithm for each hop along the path of the connection. Thus, each ABR connection is assigned a sequence of window sizes which specify the window size to be used at each hop along the path for the connection. Each node along the path of a connection reserves buffer space to hold a number of packets equal to the window size for the link incoming to that node. Packet loss is avoided by sending acknowledgements only after a packet has been forwarded to the next node along the path. This scheme results in "backpressure" that propagates towards the source in case of congestion. For example if congestion occurs on a link, it inhibits acknowledgements from being sent back on the upstream link, causing the buffer space allocated to the connection at the node feeding the upstream link to fill. This will cause that node to delay sending acknowledgements to the node upstream to that node, and so forth. If congestion persists long enough, the source will eventually be inhibited from transmitting packets. It has been shown that if each node transmits buffered packets from ABR connections that pass through it on a round robin basis, if window sizes are sufficiently large, and if each source always has data to send, then the resulting bandwidths that the ABR sources receive are max-min fair [Hahne, 1991]. This scheme is

also known as credit based flow control [Kung and Morris, 1995], and algorithms have been proposed for adaptively changing the window sizes in response to demand.

Another way to achieve max-min fairness (or fairness with respect to any criterion for allocated bandwidths) is to explicitly compute the bandwidth allocations and enforce the allocations at the entry points of the network. Explicit enforcement of bandwidth allocations is known as rate based flow control.

Bandwidth is defined in a time average sense, and this needs to be specified precisely in order to enforce a bandwidth allocation. Suppose packet sizes are fixed. One way to precisely define conformance to a bandwidth allocation of ρ packets per unit time is as follows. In any interval of time of duration x , a source can transmit at most $\rho x + \sigma$ packets. The packet stream departing the source is then said to be (σ, ρ) -smooth. The parameter σ is a constant positive integer and is a measure of the potential burstiness of the source. Over short intervals of time, the bandwidth of the source can be larger than ρ , but over sufficiently large (depending on the value σ) intervals of time, the bandwidth is essentially at most ρ . The source or network can insure conformance to this by using leaky bucket flow control [Turner, 1986], defined as follows.

A source is not allowed to transmit a packet unless it has a permit, and each transmitted packet by the source consumes one permit. As long as the number of permits is less than σ , the source receives new permits once every $(1/\rho)$ units of time. Thus, a source can accumulate up to σ permits. A packet that arrives when there are no permits can either be buffered by the source or discarded. Each ABR source could be assigned the parameters (σ, ρ) , where ρ is the allocated bandwidth of the source, and σ represents the allocated burstiness of the source, in some sense.

Non-ABR sources require a certain minimum bandwidth from the network for satisfactory operation, and may also require that latency be bounded. Examples are multimedia sources, interactive database sources, distributed computing applications, and real-time applications. Leaky bucket flow control can also be applied to non-ABR sources. The parameter ρ specifies the maximum average bandwidth of the source,

and the parameter σ measures the potential burstiness of the source, in some sense. If leaky bucket rate control is applied to all of the sources in the network, delay and buffering requirements can be controlled, and packet loss can be reduced or eliminated. Some recent analyses of this issue are presented in [Cruz, 1995], [Cruz, 1991a], [Cruz, 1991b]. We now briefly describe one important idea in these analyses.

A packet stream may become more bursty as it passes through network nodes. For example, suppose a packet stream arriving to a node is (σ, ρ) -smooth, and that each packet has a delay in the node of at most d seconds. Consider the packet stream as it departs the node. Over any interval of length x , say $[t, t+x]$, the packets that depart must have arrived to the switch in the interval $[t-d, t+x]$, since the delay is bounded by d . The number of these packets is bounded by $\rho(x+d) + \sigma = \rho x + (\sigma + \rho d)$, since the packet stream is (σ, ρ) -smooth as it enters the node. Hence, the packet stream as it departs the node is $(\sigma + \rho d, \rho)$ -smooth, and is potentially more bursty than the packet stream as it entered the node. Thus, as a packet stream passes through network nodes, it can get more bursty at each hop, and the buffering requirements will increase at each hop.

We note that buffer requirements at intermediate nodes can be reduced by using rate control (with buffering) to govern the transmission of packets within the network [Cruz, 1991b] [Golestani, 1991]. This limits the burstiness of packet flow within the network, at the expense of some increase in average delay. It is also possible to reduce delay jitter by using this approach.

Defining Terms

ARQ Protocol: ARQ is an acronym for "Automatic Repeat Request". An ARQ protocol insures reliable transport of packets between two points in a network. It involves labeling packets with sequence numbers and management of retransmission of lost or errored packets. It can be exercised on a link, or on an end-to-end basis.

Datagram Network: A packet switched network that treats each packet as a separate entity. Each packet, or datagram, is forwarded to its destination independently. The data sources do not coordinate with the network or the destinations prior to sending a datagram. Datagrams may be lost in the network, without notification of the source.

Header Field: A packet is generally divided into a "header" and a "payload". The payload carries the data being transported, while the header contains control information for routing, sequence numbers for ARQ, flow control, error detection, etc. The header is generally divided into fields, and a header field contains information specific to a type of control function.

Packet Switched Network: A network where the fundamental unit of information is a packet. To achieve economy, the communication links of a packet switched network are statistically shared. Buffering is required at network nodes to resolve packet contentions for communication links.

Virtual Circuit Switched Network: A packet switched network that organizes data packets according to the virtual circuit that they belong to. Packets from a given virtual circuit are routed along a fixed path, and each node along the path may reserve resources for the virtual circuit. A virtual circuit must be established before data can be forwarded along it.

Trunk Reservation: A technique used in telephone networks whereby some portion of a communications link is reserved for circuits (trunks) that make most efficient use of the link. A link may reject a circuit request because of trunk reservation, so that other circuits which make better use of a the link may later be accepted.

References

Bertsekas, D. and Gallager, R. 1992. *Data Networks*, 2nd ed. Prentice Hall, Inc., Englewood Cliffs, NJ.

Comer, D. 1991. *Internetworking with TCP/IP , Vol. 1: Principles, Protocols, and Architecture*. 2nd ed. Prentice Hall, Inc., Englewood Cliffs, NJ.

Cruz, R. L. 1995. Quality of Service Guarantees in Virtual Circuit Switched Networks. *IEEE J. Selected Areas in Comm.* , 13(6):1048-1056.

- Cruz, R. L. 1991a. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Trans. Inf. Th.* , 37(1): 114-131.
- Cruz, R. L. 1991b. A Calculus for Network Delay, Part II: Network Analysis. *IEEE Trans. Inf. Th.* , 37(1): 132-141.
- Cruz, R. L. and Tsai, J. T. 1996. COD: Alternative Architectures for High Speed Packet Switching. *IEEE/ACM Transactions on Networking*, 4(1):11-21.
- Hahne, E. 1991. Round-Robin Scheduling for Max-Min Fairness in Data Networks. *IEEE J. Selected Areas in Comm.*, 9(7): 1024-39.
- Jacobson, V. 1988. Congestion Avoidance and Control. *Proc. ACM SIGCOMM '88*, Stanford, CA, pp. 314-329.
- Jain, R. 1986. A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks. *IEEE J. Selected Areas in Comm.*, 4(7): 1162-1167.
- Golestani, S, J. 1991 Congestion-free Communication in High Speed Packet Networks. *IEEE Trans. Comm.*, 39(12):1802-12.
- Kung, H. T. and Morris, R. 1995. Credit-Based Flow Control for ATM Networks. *IEEE Network Mag.*, 9(2): 40-48.
- Maxemchuk, N. 1987. Routing in the Manhattan Street Network. *IEEE Trans..Comm.*, 35(5):503-512.
- Steenstrup, M. 1995. *Routing in Communication Networks*. Prentice Hall, Inc., Englewood Cliffs, NJ.
- Turner, J. S. 1986. New Directions in Communications. *IEEE Commun. Mag.*, 24(10):8-15.

For Further Information

Routing in Communication Networks, edited by Martha E. Steenstrup is a recent survey of the field of routing, including contributed chapters on routing in circuit switched networks, packet switched networks, high speed networks, and mobile networks.

Internetworking with TCP ,Vol. 1 by Douglas E. Comer provides an introduction to protocols used on the Internet, and provides pointers to information available on-line through the Internet.

Proceedings of the INFOCOM Conference are published annually by the Computer Society of the IEEE. These proceedings document some of the latest developments in communication networks.

The journal *IEEE/ACM Transactions on Networking* reports advances in the field of networks. For subscription information contact: IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway NJ, 08855-1331. Phone (800) 678-IEEE.