On Packet Switches with Infinite Storage

JOHN B. NAGLE

Abstract—**Most prior work on congestion in datagram systems focuses on buffer management. We find it illuminating to consider the case of a packet switch with infinite storage. Such a packet switch can never run out of buffers. It can, however, still become congested. The meaning of congestion in an infinite-storage system is explored. We demonstrate the unexpected result that a datagram network with infinite storage, first-in, first-out queueing, at least two packet switches, and a finite packet lifetime will, under overload, drop *all* packets. By attacking the problem of congestion for the infinite-storage case, we discover new solutions applicable to switches with finite storage.**

Introduction

PACKET switching was first introduced in an era when computer data storage was several orders of magnitude more expensive than it is today. Strenuous efforts were made in the early days to build packet switches with the absolute minimum of storage required for operation. The problem of congestion control was generally considered to be one of avoiding buffer exhaustion in the packet switches. We take a different view here. We choose to begin our analysis by assuming the availability of infinite memory. This forces us to look at congestion from a fresh perspective.

Pure datagram systems are especially prone to congestion problems. The blocking mechanisms provided by virtual circuit systems are absent. No fully effective solutions to congestion in pure datagram systems are known. Vinton Cerf's 1981 comment, "The state of the art is woefully weak,"[1] was unfortunately still valid in 1985. Most existing datagram systems behave badly under overload.

We will show that substantial progress can be made on the congestion control problem, even for pure datagram systems, once the problem is defined as determining the order of packet transmission rather than the allocation of buffer space.

A Packet Switch with Infinite Storage

Let us begin by describing a simple packet switch with infinite storage. A switch has incoming and outgoing links. Each link has a fixed data transfer rate. Not all links need have the same data rate. Packets arrive on incoming links and are immediately assigned an outgoing link by some routing mechanism not examined here. Each outgoing link has a queue. Packets are removed from that queue and sent on its outgoing link at the data rate for that link. Initially, we will assume that queues are managed in a first-in, first-out manner.

We assume that packets have a finite lifetime. In the DoD IP protocol, packets have a *time-to-live* field, which is the number of seconds remaining until the packet must be discarded as uninteresting. As the packet travels through the network, this field is decremented; if it becomes zero, the packet must be discarded. The initial value for this field is fixed; in the DoD IP protocol, this value is by default 15.

Paper approved by the Editor for Communication Switching of the IEEE Communications Society. Manuscript received January 27, 1986; revised April 22, 1986.

The author was with the Ford Aerospace and Communications Corporation, Palo Alto, CA 94303. He is now with the Center for Design Research, Stanford University, Stanford, CA 94305.

IEEE Log Number 8612595.

[1] *Protocols and Techniques for Data Communication Networks*, F. Kuo, Ed. Englewood Cliffs, NJ: Prentice-Hall, p. 18.

The time-to-live mechanism prevents queues from growing without bound; when the queues become sufficiently long, packets will time out before being sent. This places an upper bound on the total size of all queues; this bound is determined by the total data rate for all incoming links and the upper limit on the time-to-live.

However, this does not eliminate congestion. Let us see why.

Consider a simple node, with one incoming link and one outgoing link, as shown in Fig. 1. A number of hosts can generate packets for the node, which has more bandwidth into it than out of it. In the figure, we show the source host and the time-to-live value for each packet. Host A is the major generator of traffic, and is generating enough traffic to saturate the output link all by itself. The packets are queued first in, first out, and all hosts use the same initial time-to-live value. The packets are thus ordered by decreasing time-to-live.

Assume that the packet arrival rate at a node exceeds the departure rate. The queue length for the outgoing link will then grow until the transit time through the queue exceeds the time-to-live of the incoming packets. At this point, as the process serving the outgoing link removes packets from the queue, it will sometimes find a packet whose time-to-live field has been decremented to zero. In such a case, it will discard that packet and will try again with the next packet on the queue. Packets with nonzero time-to-live fields will be transmitted on the outgoing link.

The packets that do get transmitted have nonzero time-to-live values. But once the steady state under overload has been reached, these values will be small since the packet will have been on the queue for slightly less than the maximum time-to-live. In fact, if the departure rate is greater than one per time-to-live unit, the time-to-live of any forwarded packet will be exactly one. This follows from the observation that if more than one packet is sent per time-to-live unit, consecutive packets on the queue will have time-to-live values that differ by no more than 1. Thus, as the component of the packet switch that removes packets from the queue and either sends them or discards them as expired operates, it will either find packets with negative or zero time-to-live values (which it will discard) or packets with values of 1, which it will send.

So, clearly enough, at the next node of the packet-switching system, the arriving packets will all have time-to-live values of 1. Since we always decrement the time-to-live value by at least 1 in each node to guarantee that the time-to-live value decreases as the packet travels through the network, we will, in this case, decrement it to zero for each incoming packet and will then discard that packet.

We have thus shown that a datagram network with infinite storage, first-in, first-out queueing, and a finite packet lifetime will, under overload, drop *all* packets. This is a rather unexpected result. But it is quite real. It is not an artifact of the infinite-buffer assumption. The problem still occurs in networks with finite storage, but the effects are less clearly seen. Datagram networks are known to behave badly under overload, but analysis of this behavior has been lacking. In the infinite-buffer case, the analysis is quite simple, as we have shown, and we obtain considerable insight into the problem.

One would expect this phenomenon to have been discovered previously. But previous work on congestion control in

0090-6778/87/0400-0435$01.00

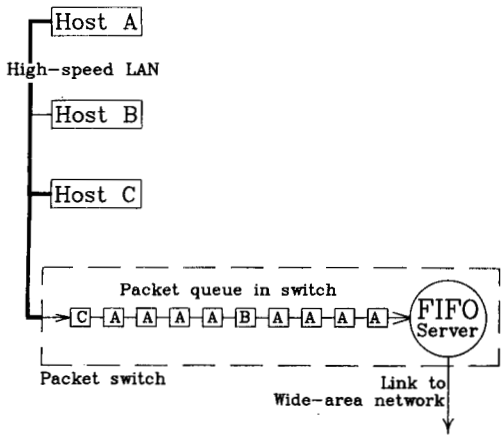

Fig. 1. First-in, first-out queueing.

packet-switching systems almost invariably focuses either on buffer management, link blocking, or in virtual circuit networks, call blocking. Analysis of the infinite-buffer case is apparently unique with this author. The fair queueing concept introduced below is, of course, well known in the operating systems world, but seems not to have been previously applied to packet-switched networks.

This result is directly applicable to networks with finite resources. The storage required to implement a switch that can never run out of buffers turns out to be quite reasonable. Let us consider a pure datagram switch for an ARPANET-like network. For the case of a packet switch with four 56 kbit links and an upper bound on the time-to-live of 15 s, the maximum buffer space that could ever be required is 420K bytes. A switch provided with this rather modest amount of memory need never drop a packet due to buffer exhaustion.

This problem is not just theoretical. We have demonstrated it experimentally on our own network, using a supermini with several megabytes of memory as a switch. We now have experimental evidence that the phenomenon described above occurs in practice. Our first experiment, using an Ethernet on one side of the switch and a 9600 baud line on the other, resulted in 916 IP datagrams queued in the switch at peak. However, we were applying the load over a TCP transport connection, and the transport connection timed out due to excessive roundtrip time before the queue reached the time to live limit, so we did not actually reach the stable state with the queue at the maximum length as predicted by our analysis above. It is interesting that we can force this condition from the position of a user application atop the transport layer (TCP), and this deserves further analysis.

Interaction with Transport Protocols

We have thus far assumed packet sources that emit packets at a fixed rate. This is a valid model for certain sources such as packet voice systems. Systems that use transport protocols of the ISO TP4 or DoD TCP class, however, ought to be better behaved. The key point is that transport protocols used in datagram systems should behave in such a way as to not overload the network, even where the network has no means of keeping them from doing so. This is quite possible. In a previous paper by this author,[2] the behavior of the TCP transport protocol over a congested network was explored. We have shown that a badly behaved transport protocol implementation can cause serious harm to a datagram network, and we discussed how such an implementation ought to behave. In that paper, we offered some specific guidance on how to implement a well-behaved TCP, and demonstrated that proper behavior could in some cases reduce network load by an order of magnitude. In summary, the conclusions of that paper are that a transport protocol, to be well behaved, should not have a retransmit time shorter than the current roundtrip time between the hosts involved, and that when informed by the network of congestion, the transport protocol should take steps to reduce the number of packets outstanding on the connection.

We reference our earlier work here to show that the network load imposed by a transport protocol is not necessarily fixed by the protocol specification. Some existing implementations of transport protocols are well behaved. Others are not. We have observed a wide variability among existing TCP implementations. We have reason to suspect that ISO TP4 implementations will be more uniform, given the greater rigidity of the specification, but we see enough open space in the TP4 standard to allow for considerable variability. We suspect that there will be marginal TP4 implementations, from a network viewpoint, just as there are marginal TCP implementations today. These implementations will typically work quite well until asked to operate over a heavily loaded network with significant delays. Then we find out which ones are well behaved.

Even if all hosts are moderately well behaved, there is potential for trouble. Each host can normally obtain more network bandwidth by transmitting more packets per unit time since the first-in, first-out strategy gives the most resources to the sender of the most packets. But collectively, as the hosts overload the network, total throughput drops. As shown above, throughput may drop to zero. Thus, the optimal strategy for each host is strongly suboptimal for the network as a whole.

Game-Theoretic Aspects of Network Congestion

This game-theory view of datagram networks leads us to a digression on the stability of multiplayer games. Systems in which the optimal strategy for each player is suboptimal for all players are known to tend towards the suboptimal state. The well-known prisoner's dilemma problem in game theory is an example of a system with this property. But a closer analog is the *tragedy of the commons* problem in economics. Where each individual can improve his own position by using more of a free resource, but the total amount of the resource degrades as the number of users increases, self-interest leads to overload of the resource and collapse. Historically, this analysis was applied to the use of common grazing lands; it also applies to such diverse resources as air quality and time-sharing systems. In general, experience indicates that many-player systems with this type of instability tend to get into serious trouble.

Solutions to the tragedy of the commons problem fall into three classes: cooperative, authoritarian, and market. Cooperative solutions, where everyone agrees to be well behaved, are adequate for small numbers of players, but tend to break down as the number of players increases. Authoritarian solutions are effective when behavior can be easily monitored, but tend to fail if the definition of good behavior is subtle. A market solution is possible only if the rules of the game can be changed so that the optimal strategy for players results in a situation that is optimal for all. Where this is possible, market solutions can be quite effective.

The above analysis is generally valid for human players. In the network case, we have the interesting situation that the player is a computer executing a preprogrammed strategy. But this alone does not ensure good behavior; the strategy in the computer may be programmed to optimize performance for that computer, regardless of network considerations. A similar situation exists with automatic redialing devices in telephony

[2] "Congestion control in IP/TCP internetworks," *ACM Comput. Commun. Rev.*, Oct. 1984.

where the user's equipment attempts to improve performance over an overloaded network by rapidly redialing failed calls. Since call-setup facilities are scarce resources on telephone systems, this can seriously impact the network; there are countries that have been forced to prohibit such devices (Brazil, for one). This solution by administrative fiat is sometimes effective and sometimes not, depending on the relative power of the administrative authority and the users.

As transport protocols become more commercialized and competing systems are available, we should expect to see attempts to tune the protocols in ways that may be optimal from the point of view of a single host, but suboptimal from the point of view of the entire network. We already see signs of this in the transport protocol implementation of one popular workstation manufacturer.

So, to return to our analysis of a pure datagram internetwork, an authoritarian solution would order all hosts to be "well behaved" by fiat; this might be difficult since the definition of a well-behaved host in terms of its externally observed behavior is subtle. A cooperative solution faces the same problem, along with the difficult additional problem of applying the requisite social pressures in a distributed system. A market solution requires that we make it pay to be well behaved. To do this, we will have to change the rules of the game.

Fairness in Packet-Switching Systems

We would like to protect the network from hosts that are not well behaved. More specifically, we would like, in the presence of both well-behaved and badly behaved hosts, to ensure that well-behaved hosts receive better service than badly behaved hosts. We have devised a means of achieving this.

Let us consider a network that consists of high-bandwidth pure-datagram local area networks without flow control (Ethernet and most IEEE 802.x datagram systems are of this class, whether based on carrier sensing or token passing), hosts connected to these local area networks, and an interconnected wide area network composed of packet switches and long-haul links. The wide area network may have internal flow control, but has no way of imposing mandatory flow control on the source hosts. The DoD Internet, Xerox Network Systems internetworks, and the networks derived from them fit this model.

If any host on a local area network generates packets routed to the wide area network at a rate greater than the wide area network can absorb them, congestion will result in the packet switch connecting the local and wide area networks. If the packet switches queue on a strictly first-in, first-out basis, the badly behaved host will interfere with the transmission of data by other, better behaved hosts.

We introduce the concept of fairness. We would like to make our packet switches fair; in other words, each source host should be able to obtain an equal fraction of the resources of each packet switch. We can do this by replacing the single first-in, first-out queue associated with each outgoing link with multiple queues, one for each source host in the entire network, as shown in Fig. 2.

The same conventions as for Fig. 1 apply, except that we service the multiple queues in a round-robin fashion, taking one packet from each nonempty queue in turn and transmitting the packets with positive time-to-live values on the associated outgoing link, while dropping the expired packets. Empty queues are skipped over and lose their turn. The queues are by source host, not by destination host, because we are trying to be fair to the sources, which have control over the generation of packets.

This mechanism is fair; outgoing link bandwidth is parceled out equally among source hosts. Each source host with packets queued in the switch for the specified outgoing link gets

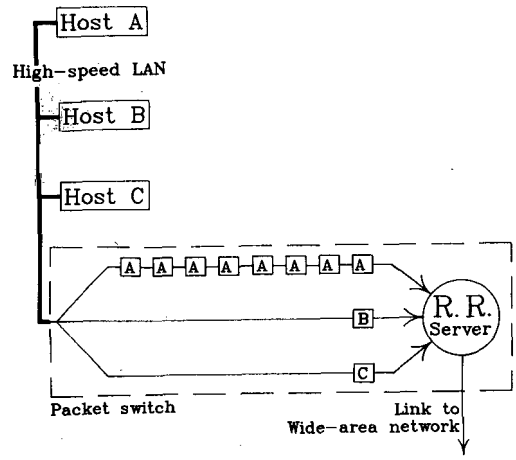

Fig. 2. Fair queueing.

exactly one packet sent on the outgoing link each time the round-robin algorithm cycles. So we have implemented a form of load balancing.

We have also improved the system from a game theory point of view. The optimal strategy for a given host is not longer to send as many packets as possible. The optimal strategy is now to send packets at a rate that keeps exactly one packet waiting to be sent in each packet switch since, in this way, the host will be serviced each time the round-robin algorithm cycles, and the host's packets will experience the minimum transit delay. This strategy is quite acceptable from the network's point of view since the length of each queue will, in general, be between 1 and 2.

The hosts need advisory information from the network to optimize their strategies. The existing Source Quench mechanism in DoD IP, while minimal, is sufficient to provide this. The packet switches should send a Source Quench message to a source host whenever the number of packets in the queue for that source host exceeds some small value, probably 2. If the hosts act to keep their traffic just below the point at which Source Quench messages are received, the network should run with mean queue lengths below 2 for each host.

Badly behaved hosts can send all the datagrams they want, but they will not thereby increase their share of the network resources. All that will happen is that packets from such hosts will experience long transit times through the network. A sufficiently badly behaved host can send enough datagrams to push its own transit times up to the time to live limit, in which case none of its datagrams will get through. This effect will happen sooner with fair queueing than with first-in, first-out queueing because the badly behaved host will only obtain a share of the bandwidth inversely proportional to the number of hosts using the packet switch at the moment. This is much less than the share it would have under the old system where more verbose hosts obtained more bandwidth. This provides a strong incentive for badly behaved hosts to improve their behavior.

It is worth noting that *malicious*, as opposed to merely badly behaved, hosts can overload the network by using many different source addresses in their datagrams, thereby impersonating a large number of different hosts and obtaining a larger share of the network bandwidth. This is an attack on the network; it is not likely to happen by accident. It is, thus, a network security problem, and will not be discussed further here.

Although we have made the packet switches fair, we have not thereby made the network as a whole fair. This is a weakness of our approach. The strategy outlined here is most applicable to a packet switch at a choke point in a network,

such as an entry node of a wide area network or an internetwork gateway. As a strategy applicable to an intermediate node of a large packet-switching network where the packets from many hosts at different locations pass through the switch, it is less applicable. The author does not claim that the approach described here is a complete solution to the problem of congestion in datagram networks. However, it presents a solution to a serious problem and a direction for future work on the general case.

Implementation

The problem of maintaining a separate queue for each source host for each outgoing link in each packet switch seems at first to add considerably to the complexity of the queueing mechanism in the packet switches. There is some complexity involved, but the manipulations are simpler than those required with, say, balanced binary trees. One simple implementation involves providing space for pointers as part of the header of each datagram buffer. The queue for each source host need only be singly linked, and the queue heads (which are the first buffer of each queue) need to be doubly linked so that we can delete an entire queue when it is empty. Thus, we need three pointers in each buffer. More elaborate strategies can be devised to speed up the process when the queues are long. But the additional complexity is probably not justified in practice.

Given a finite buffer supply, we may someday be faced with buffer exhaustion. In such a case, we should drop the packet at the end of the longest queue since it is the one that would be transmitted last. This, of course, is unfavorable to the host with the most datagrams in the network, which is in keeping with our goal of fairness.

Conclusion

By breaking away from packet-switching's historical fixation on buffer management, we have achieved some new insights into congestion control in datagram systems and developed solutions for some known problems in real systems. We hope that others, given this new insight, will go on to make some real progress on the general datagram congestion problem.

★

John B. Nagle received the B.S.E.E. degree from Case Western Reserve University, Cleveland, OH, in 1970 and the M.S.C.S. degree from Stanford University, Stanford, CA, in 1985.

At Ford Aerospace, he worked on secure operating systems, mathematical proof of correctness, and packet networks. As an independent inventor, he developed Autodesk Inc.'s CAD/camera product, and is presently affiliated with the Center for Design Research at Stanford.