M.I.T. Laboratory for Computer Science

Request for Comments No. 250
July 26, 1983

Congestion Control at Internet Gateway

by Lixia Zhang

1. Introduction

Congestion control has been a long-standing problem in packet-switched computer communication networks. A companion paper "A Survey on Congestion Control in Computer Networks" [9] briefly summarizes previous practical approaches to solving this problem, and explains, from the author's point of view, what the fundamental defects are in those approaches, and what a right direction seems to be for seeking new solutions.

As an application of those viewpoints, this paper proposes a congestion control algorithm for a specific environment - at gateways in an internetwork environment with DoD Internet Protocol (IP) implementation.

Section 2 gives the model of the problem environment. Section 3 introduces the current IP congestion control mechanism and a simulation result. From there we see that the current mechanism cannot effectively perform the congestion control without considerable degradation of performance. Hence designing a new congestion control algorithm for IP is necessary. Section 4 discusses some general considerations on the design of the new algorithm. Based on that, Section 5 describes an outline of the proposed algorithm. Section 6 is a short analysis of the new algorithm and the conclusion.

2. Modeling the Problem

We model an internet environment as a general store-and-forward computer communication network. In this "network", the "packet switching nodes" are gateways (GW) and "transmission lines" are local networks[1] connecting hosts to GWs and GWs to one another. In terms of the ISO protocol hierarchy terminology [1], IP is the network level protocol in this model. This abstract network has the following characteristics:

- ~ Because the transmission speeds of local area networks (LAN) and long haul networks usually differ by orders of magnitude, so possibly do the "line" speeds in our abstract network model. Here we see how easily a packet stream from a high bandwidth line can flood the GW when going out through a low bandwidth line if no proper control mechanism has been installed.
- ~ A local network connecting hosts to GWs or GWs to GWs may be a high bandwidth local area network, a long haul network, or satellite channels. Delays between GWs may also vary by orders of magnitude. Notice the difference between *line speed* and *delay*: the former is the rate at which outbound lines drain out packets, the latter is the period of time for a packet to travel from hosts to GWs (or GW to GW) or vice versa.
- ~ IP does not have an Acknowledgment/Retransmission mechanism for reliable packet delivery. If an outgoing packet is rejected by the local network, due to either transmission error or local network congestion, it will be thrown away.

We also make the following assumptions:

1. Each GW has a large, though finite, packet buffer, e.g. enough to hold 1000 packets, so that we will not consider the buffer shortage problem in the following proposed algorithm. This is a very reasonable assumption in terms of current microprocessor memory sizes[2], although most present GW implementations do not have such a large buffer space.
2. The network may use a dynamic routing algorithm, but there is only one single route

---

[1] In an internet environment, any homogeneous network, no matter whether it is within a single building or nation-wide, is considered as a *local network*.

[2] Assume the packet size is 500 bytes, then a pool for 1000 packets is 500 Kbytes large. MC68000 is claimed to be able to have up to 16 Mbytes memory, an order larger than our assumption.

between a pair of hosts at any instance of time[3].

3. The routing algorithm is relatively stable. That is, the route usually is unchanged during a bursty transmission, or at least a packet stream will not be frequently switched back and forth between different routes[3]. Currently the gateway-to-gateway routing uses a minimum-hop algorithm, so this assumption is valid as long as gateways do not fail too frequently, as is the case in reality.

3. IP Source Quench Algorithm and a Simulation Result

It is well known that congestion is unavoidable in computer networks, every network must have some sort of control mechanism to handle it. Currently IP uses a ***source quench*** mechanism, a part of the Internet Control Message Protocol (ICMP) [6] functions, for congestion control. It works as follows: A GW may discard incoming packets if it does not have the buffer space needed to queue the packets for output to the next network on the route to the destination. When it does so, it may send a source quench message to the source host for every packet it discards. This message is a request to the host to cut back its sending rate. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the GW. The host can then gradually increase the sending rate until it again receives source quench messages.

Although specified in the IP document, so far there are few implementations of this source quench mechanism. There is no report on their performance. In this section we analyze the algorithm first, then give a simulation result. From there we will argue why we do not think this algorithm is sufficient in congestion control, hence we need to design a new one.

---

[3] A bug here: in the case that there exist more than one route with the same minimum number of hops, the GW will route packets to them in round-robin. This fact is temporarily ignored here due to the following reasons:

- I think this algorithm is a subject to change. We can ask several questions on its performance: Should it keep round-robin routing even in the case that one of the minimum hop routes is congested? Should it count a long haul network as one hop, as it does for a high bandwidth local area network? Should it not consider the type-of-service of the user data packets when there exist more than one available route, but going through different networks with drastically different characteristics? Etc.
- There is an intrinsic relation between the routing and congestion control problems. A general belief is that we cannot solve one of the two completely without solving the other. At this early stage of investigating fundamental issues of congestion control, however, I try to simplify the problem by making some reasonable assumptions on routing, leaving exposing this relation and routing consideration for later.

3.1. Advantages and problems of IP source quench algorithm

We immediately see some advantages of this simple algorithm:

1. The two facts that GW can drop packets whenever necessary and that IP does not have acknowledgment/retransmission procedure together make the IP environment deadlock free, no matter how heavily the network is congested.

2. Because a GW sends a quench message to the source host for every dropped packet, it need not keep any traffic record.

3. It chokes the source hosts directly, trying to reduce the excess traffic coming into the network as soon as possible.

4. A host actively participates in this algorithm. It can always try to make higher throughput with no penalty except its own resources spent on possible retransmissions.

But there are also some questionable points about this algorithm:

1. It is too late to quench the source host after the GW runs out of buffer space. The quench message takes some period of delay to get back to the source host. All packets sent during that period might be discarded. The source host should be warned well before the GW starts dropping packets.

2. The algorithm does not specify how much a host should cut back its transmission rate upon receiving a quench message. Since the host has no knowledge about the congested GW, such as its capacity to drain out the congestion, and how many other packet streams are passing through at the same time, it cannot feasibly figure out a right cutoff rate on the fly.

3. By current specification, IP does not control the transmission rate itself. The source quench algorithm puts an implied implementation requirement on the higher layer at the host. It requires that each host have precise control of packet transmission rates to different destinations and have the ability to change them at will. But the current implementations of most higher level protocols built on top of IP do not have such control[4]. We cannot pass this requirement to the lower layer, the local network protocol, because that would make IP incompatible with the local networks that do not have such a function.

---

[4]For example, TCP, the mostly used IP based protocol, sends out packets at a rate determined by the end-to-end windowing scheme, a form of flow control that is dependent on the (unknown!) round-trip delay and not based on real time.

4. In the case that the GW CPU cycle is the bottleneck that causes the congestion, the input line(s) will have queued up incoming packets, filling up the buffer space. As a result, no more packets will be accepted. The GW may not be aware of dropping packets and will not send back quench messages. In another word, not every source host will be aware of the congestion even its packets have been dropped by some GW.
5. According to the specification, when there is no space available to forward an incoming packet, the GW should drop it and send back a quench message to the source host. But sending quench messages is resource consuming by itself. It takes both CPU time and buffer space, probably no cheaper than forwarding a packet, except that the reverse direction may or may not be congested. Consider a heavily congested GW, why does it necessarily quench the host for *every* dropped packet? If this is used just as a way to inform the source host that the GW is still congested, it is too expensive, and may not reflect the true situation of the GW since the quench message takes an unknown period of delay.

3.2. A simulation result of IP source quench algorithm

In his BS thesis Haldane Peterson studied the IP source quench algorithm and built a simulator to test the performance [4]. The following assumptions were made in the simulation:

1. The simplest network configuration was used, one host connected to one GW.
2. The transmission delay between the host and the GW was constant but unknown.
3. An infinite packet stream from the host was being sent to the GW.
4. The GW drained out packets at a constant rate. When its buffer was full, it sent out quench messages to the host with no affect on data transmission, i.e.*the expensive control cost was ignored.*
5. The host had precise control of the packet transmission rate. It tried several algorithms upon receiving a quench message from the GW, such as null algorithm[5], bang-bang

[5]The host completely ignores quench messages and keeps sending at the same rate no matter what happens.

speed control[6], event-based changes[7], time-dependent changes[8], event-slotted windows,[9] and some even more complicated ones.

The throughput and loss coefficient (i.e. the ratio of the number of quench packets the host received to the total number of packets the host sent) were both used to measure the performance. It turned out that the "toe-in-water" algorithm[10] gave better performance than the others (the throughput reached 99% of the gateway's capacity with the loss coefficient under 5%), because it was the best way for the host to tune to the proper transmission rate in order to match that of the GW. But a fatal drawback was that even in this extremely simple model, it was still expensive for the host to figure out the GW's capacity by such blank guessing and trial. In a real world situation of multiple host with random data transmissions and several networks with different speeds, it will not be possible at all, even without considering the cost.

Although this simple model of a single host and infinite packet stream transmission is not very close to actual network traffic pattern, we did learn a number of things from the simulator:

1. The inherent unknown communication delay between the host and the GW is a subtle problem, which easily causes traffic oscillation when the GW tries to control the source host sending rate.
2. Instead of letting source hosts guess their cut-off rate, the GW should decide transmission rate limits for them, quantitatively controlling the resource sharing among

---

[6]Upon receipt of a quench, the host stops entirely for some amount of time, after which it begins to send again at the same speed as before.

[7]Upon receipt of a quench, the host immediately multiplies its sending rate by some constant less than one. Every time it sends a packet, it multiplies its rate by some other constant greater than one.

[8]In this method, no action is taken at the time a quench occurs, except to note the fact that one did occur and to record the time T. Every time the host sends a packet, the new inter-packet delay time, d(n+1), is computed as

$$d(n+1) = d(n) + (k1 - k2(t - T))$$

where d(n) is the inter-packet delay computed at the last sending, t is the current time, and k1 and k2 are constants.

[9]Keeping a window on the recent past and counting the number of quenches in that window, and using this information with current sending rate to decide new sending speed.

[10]The name of this algorithm comes from a common practice of prudent swimmers. Before diving into new water, they will dip a toe in to test the temperature. The analogy here is in the speedup part of the algorithm. The host first tries a very short, very fast packet burst to the GW to see how full the queue is. If it is not too bad, then the host speeds up, otherwise holds its speed.

all users. One reason is that, as mentioned above, a single host does not have the knowledge of the total network traffic. Another more important reason is that the GW cannot safely rely on each host.

3. Therefore, it is necessary to have some sort of communication between hosts and GWs in order for them to work out the problem together. The quench message is one-way communication only and contains very vague information.

I feel that the basic defect of IP congestion control mechanism is that, for some reason, it did not intend to try hard to control the situation and to achieve high performance. People know that congestion control is needed but probably have not learned how to do it well.

4. Design Considerations on Proposed Congestion Control Algorithm

We concluded in [9] that the network congestion control is a distributed resource sharing problem, and the basic principle of this sharing is to enhance the network with packet traffic knowledge and to let the network properly regulate hosts' transmission rates. The implementation of this principle, however, may be different for different networks and protocols. Most considerations in this section are closely related to the special characteristics of IP.

4.1. General Principles

The following can be considered as general design principles for congestion control algorithms:

1. It should be quick to take effect.
2. The techniques should be preventive in nature, rather than being of disaster-detection-recovery type[11].
3. The control should be smooth; that is, it should be an incremental procedure rather than an "on/off" switching.
4. The overhead should be kept at a low level, preferably a constant level independent of the network load.
5. It should be fair, according to whatever the definition of fairness is given.

---

[11] In this sense, the name *congestion control* is sort of misleading. If we believe that a good congestion control mechanism is the one that prevents congestion, then *traffic control* might be a better name for the mechanism.

4.2. Design Philosophy

IP itself is not designed to be highly reliable. For simplicity and flexibility reasons, it is intentionally limited in scope to only provide functions necessary for delivering a packet of bits from a source to a destination over an internetwork. As part of IP, the purpose of ICMP is to provide feedback of problems in the communication environment and hence its use is optional. The congestion control algorithm proposed in this paper is planned to be part of ICMP. To be consistent with IP's goals, it should also be simple, optional, aiming at improving network performance and throughput, rather than being imperative to the correct functioning of the network.

Another consideration is that the networks connected by gateways may belong to different administrations that may or may not all have (or be willing to have) the congestion control mechanism installed. In such cases, the original IP functions should not be affected. However, the network will not have guaranteed performance for those hosts that do not support network congestion control.

4.3. Difficulties

We listed in [9] the four fundamental difficulties in congestion control, which are all caused by the geographical distribution of network resources:

1. Difficulties in allocating the distributed resources on dynamic demands;
2. Inherent communication delay between hosts and gateways;
3. Expensive control overhead, the control mechanism will consume critical network resources;
4. Difficulties in detecting and recovering from failure.

The dynamic demands and communication delay together make the general scheduling or reservation methods of resource sharing infeasible. There is no possible prediction of traffic load to do scheduling. There is no precise resource availability/unavailability information in making reservation: the resource might not be available at the time a reservation request is received by a GW but a little later, or vice versa, due to the load change or GWs failure/recovery. Because of communication delay, any reservation will also introduce delay in responding to user requests; the reserved resources will be wasted during the time the reservation is made and the time the resources start being used (this includes the time for the confirmation getting back to the user and the time

data packets arriving at the switching node). Thus the resources must be allocated dynamically based on the network's knowledge about its current load, which can only be abstracted from the traffic instant by instant.

If we let the network control the resource sharing, the communication delay also brings difficulties. It will cause the asynchrony and inconsistency problem between the network and the hosts when anything changes, for example, when a GW fails, or when it finds a host starts (or terminates) a bursty transmission, because the control updates will take time. It should be the control algorithm's responsibility to handle the situation properly during transient periods.

Both GWs and hosts may crash at any time. A host failure will not affect the network, while a GW failure may lose all its previous control information and leaves hosts with a low transmission rate limit for a long time. So on one hand, the network should have the control over its own resources; on the other hand, hosts must protect themselves from network malfunctioning due to GW failures.

Facing all these hard problems, the congestion control cannot possibly be done in a trivial way. But if we believe that nothing can work well in a shared world without control, then the control overhead is a necessary expense. We can only try to reduce the cost, but not to eliminate it.

4.4. Relation to local network congestion control and end-to-end flow control

Here we are designing a congestion control mechanism for a network level protocol like IP which has the goal of being implementable on top of a wide variety of local networks as well as supporting a wide variety of higher level protocols. The new algorithm should be consistent with this goal. It should not put more dependencies or restrictions than the original IP on the functionality of either higher or lower level protocols. In this paper, we only consider the problem down to the IP layer and temporarily ignore local network issues[12].

---

[12] If neglect the small percentage of the network traffic due to protocol peer messages at each layer, we see that all requests are from applications and eventually served by physical media. Restrictions to users requests due to finite physical resources come from bottom up. ISO layering structure gives a logical division of functions among layers. Since each layer uses functions offered by the layer below, its performance also directly depends on the lower layer. The interfaces between layers have been well defined in terms of functions, while the issues of performance dependency are mostly ignored. Congestion control is such a performance issue. After we learn how to add performance issue to the interface between IP and the higher layers, we expect to be able to deal with the lower level interfaces as well.

For example, we cannot assume that the higher level protocol has some fixed form of flow control. The trivial file transfer protocol TFTP [7] was built on top of IP (through UDP [5]). Due to TFTP's lock-step feature, any transmission rate restriction will be trivially met. Whereas another file transfer protocol, BLAST [8], is an example at the other extreme. Its goal is to achieve high transmission throughput. Thus it does not have conventional "window" flow control mechanism. It sends the whole file, no matter how long it is, at once, then checks over and retransmits those packets that were damaged or lost.

Therefore, the mechanism we are seeking should be able to handle the congestion by itself, and give a clear cut interface to the higher layer reflecting how well the network can serve the transmission requests. It should not rely on the flow control from the higher layer, nor imply any severe restrictions. The end-to-end control should be administered as far as possible by the users with the network intervention only when the service as a whole is endangered. It may require the IP module residing in each host be able to control transmission rate in some way. Under the limitation of the given physical resources of networks, it should try best to meet users' performance requests. When demand from the higher layer exceeds the limitation, it should be able to reject the excess part.

4.5. Local nature of congestion in IP environment

A distinct feature of IP is that it has no hop-by-hop acknowledgment/retransmission procedure. This makes congestion inherently a local phenomenon in nature, if we look at the problem only at the IP level. When congestion occurs at any GW, the pure effect is packets being dropped at that particular GW. The GW does not stop or slow down any incoming traffic, nor speed up or slow down its own outgoing traffic. The packet loss will eventually be discovered by end-to-end protocols and retransmission will be done if necessary. The real disaster of congestion manifests itself as degraded performance seen by network users, due to the delay and wasted resource of retransmission.

Since the congestion is local in nature in this particular environment of IP, it should be possible to solve the congestion problem locally, i.e. building the control mechanism on individual GWs and

without worrying about global situation[13]. Every GW arranges its own resources according to moment-to-moment traffic fluctuations that may be visible only at the local level.

4.6. Traffic pattern

There usually exists no magic as a universal solution to a problem that is not based on any assumptions, no matter what kind of bizarre the environment might be. When an algorithm is proposed, some assumptions must be made on the behavior of the problem to be solved, either explicitly claimed or implicitly assumed. The closer the assumption is to reality, the better the solution will be.

In solving the congestion control problem, we consider the network traffic as consisting of two classes: a large amount of single (or say low frequency) packets and bursty packet streams. The former kind of traffic may be composed by some service request/responses, remote interactive processings, TFTP transfer, etc.. They are either human-oriented or round-trip delay dependent, hence are not the main cause of network congestion. The packet streams are due to fast data transfers and will be the target of our congestion control.

Imagine the traffic pattern in the near future, there are two reasons that we probably should be expecting it to be more and more stream-like:

1. The desire to integrate all kinds of telecommunications, such as voice, video, as well as data, and all kind of applications, such as newspaper, information retrieval, etc., into a single network.
2. The tendency that cost effect (computation price decreases more rapidly than that of communication) and better performance requirement will gradually change the current way of interactive computer communications, such as remote login. More computation will be done locally, and computer communications will be in the form of large chunk of data. Consider the huge amount of short data packet of Telnet currently flowing in

---

[13] Be warned: by saying so we have made an assumption that the routing algorithm is responsible for network resources utilization, i.e. the routing will make full use of the network resources in a global viewpoint. But in any case, I don't think that the *global congestion*, whatever it means as many people mentioned, would ever occur in an IP environment.

networks[14], we are now actually trading communication cost for computation power. We expect this to change. Don't forget that Telnet was designed over ten years ago. At that time people perhaps did not foresee today's PC revolution.

4.7. Tradeoff considerations

Tradeoffs exist in doing any kind of control. We consider that only two kinds of resources are fundamental in the network, i.e. processing power and communication bandwidths, although many other people think the buffer space should also be counted. Our considerations are :

~ The most expensive resource in networks is the communication bandwidths that should be best used[15]. The control mechanism itself should not compose more than a small portion of line overhead.

~ Processing power at each packet switching node is finite, and the CPU-bottleneck is one of the potential causes of congestion. The congestion control mechanism will surely bring some CPU overhead. But even so, we are willing to trade a small amount of CPU time in doing control functions

* to prevent congestion from occurring in order to reduce the waste of both CPU time and bandwidths of the network as well as hosts resources resulted from retransmission;
* to make better coordination with the higher layer protocols; for example, to reduce the uncertainty of the end-to-end protocols as mentioned in [9]; and
* to offer a better service to users, since a controlled network will be more stable.

Packet buffer space at each gateway, as a necessary tool in the control scheme, is assumed to be large enough.

The next section gives the outline of a proposed algorithm based on these viewpoints.

---

[14] A big amount of Arpanet traffic is Telnet packets with single byte data. Sorry I lost my reference paper on Arpanet measurement.

[15] This is not true for high bandwidth local area networks, but because gws connect all kinds of networks, we should consider the problem in a more general situation.

5. Design of a New Congestion Control Algorithm for IP

To do congestion control, a GW need have two things: an efficient control algorithm, and the information required by the algorithm. We describe the algorithm first, then consider how to get the information.

5.1. Brief description of the algorithm

Assume a gateway knows all the information it needs, the control algorithm should be able to perform the following functions:

1. According to the GW's capability and traffic load, decide whether, when, and to whom to send control messages;
2. Compose the contents of the control messages, which give the source hosts the transmission capacity of the GW they may use.
3. Take into account the communication delay and be able to properly handle the transient situation during the short period the traffic load and the GW capacity do not match.
4. Be able to handle the case that some hosts do not responds to the control properly and keep the GW functioning well.
5. The algorithm itself should be robust to GW failures.

Basically, the GW tries hard to avoid dropping packets. Whenever it feels that traffic becomes exceeding its capacity, it sends control messages at an early stage to source hosts to request them to slow down; in the meantime, it uses its fairly large buffer pool to hold continuing flooding in packets, even though it may not be able to send them out at the rate they come in. But this is only a *temporary* situation. If all hosts cooperate properly, the traffic should be back to normal soon. If (some) hosts do not respond to GW's control messages, dropping packets is still the last thing the GW can resort to.

5.1.1. When to send control messages

We want to let the network control and manage its own resources. So control messages need be sent whenever a load change is observed and readjustment of the network resource sharing is needed, namely, when load increases or decreases.

When the load increases, the GW may send a quench message to source hosts in two cases:

~ The length of a single output queue is over a threshold, which reflects a line speed bottleneck.

~ The length of total input queue is over a threshold, which indicates a CPU bottleneck.

Since we desire the control algorithm to be preventive, control message should be sent to hosts well before the GW has accumulated too many packets and let hosts be aware of the GW's forwarding capacity as early as possible. Therefore we set multiple thresholds for measuring the traffic load, and in particular, choose the first threshold to be very low for the reasons below:

~ To warn hosts in advance.

~ Although we have large pool to buffer more packets, as soon as the GW detects that incoming packets *rate* exceeds the CPU processing rate or line output rate, it should start controlling the hosts rate immediately.

~ We must not forget that the control message takes a one-way delay to get back to the host, and the GW is responsible for those packets sent out before its control messages get there.

Because IP packets are not guaranteed for delivery, the control messages may be lost. The multi-threshold is also a solution to deal with this loss. When the load passes each threshold, control messages will be sent to hosts. The probability of losing all of them should be negligibly small. The messages sent at each threshold can also give hosts the transmission rate limits upon the most recent information of the GW load.

When the traffic load down passes each threshold, i.e. when the load decreases, the GW should recompute the new transmission capacity it can afford to each packet stream passing through and check its control record to see if it need send out new control messages to avoid under-utilization of its resources. Because there should still be fairly many packets waiting at the GW after a heavy load period, the GW will not become idle during the communication delay period if it has good measure on the load change and controls in time.

The control messages may also be sent upon requests from hosts, as explained below.

5.1.2. What's in the control message

We criticized that the current IP source quench message contains little usable information to let source hosts figure out what to do. We want to put in a control message

1. the destination address of the packet stream, which causes (or is likely to cause) congestion;
2. the GW address the control message is sent from.
3. the acceptable transmission rate to the above destination, expressed in terms of burst-period and burst-length pair (BP-BL), i.e. requesting the host not to send more than BL packets in BP period.

The destination address tells to whom the source host should slow down the transmission. The GW address may be useful to the hosts which can do source routing. They then can try other possible routes to get higher throughput if need to. The BP-BL pair is computed according to the GW's capacity and current load (e.g. assign its capacity equally among current users). The reason that the transmission rate is expressed in BP-BL rather than in strict rate of second/per-packet form is to give the host more flexibility in the way it handles packets. The GW tries to make BP as long as possible, and the host may like to send all BL packets at once and stop for BP time in order to save process switch time, or sends in some other way, as long as the average rate does not exceed BL/BP. This also has an impact of suggesting the host to send larger packets to achieve higher throughput, because the allowed throughput is equal to (BL X # bytes per packet)/BP.

Since IP does not have priority transmissions, let us temporarily assume that the fairness definition is equal sharing among all users. After take the consideration of those low frequency traffic (for example, allocate certain percent resources for it), the GW computes a fair share for all bursty packet streams and puts it as the parameters BP-BL in the control messages. A limit on a few high-speed traffics may preserve the traffic of a larger number of low-speed transmissions, i.e. making fewer users unhappy. The important point here is to *let the* GW *have the information of traffic and some way to control it.* Then the different fairness definitions become simply different ways of setting control parameters.[16]

---

[16] Take the favoring transit-over-input congestion control mechanism as an example, it is known to be unfair to local input packets. But if we drop local incoming and transit packets equally when congestion occurs, we also drop the *resources* spent on the transit packets already. The real problem is that the algorithm has no way to stop the transit traffic at the beginning. Many other fairness problems can also be solved after we build the traffic control functionality into networks.

5.1.3. On forwarding control messages

Control messages should be given special consideration. The delay or loss of a user data packet is unfortunate to that user, but it does not have any globally deleterious effect. The delay or loss of a control message, on the other hand, is directly related to the network performance and therefore all users may suffer. Currently all ICMP packets are treated equally as user data packets. I think this policy should be considered to change, because ICMP packets, such as routing updates, echoing, and the proposed congestion control messages, do carry more important information than general data packets. Therefore they should be handled differently. For example, the congestion control messages should not be dropped whenever possible, and they should be sent to hosts as soon as possible. This necessity can be explained by a real life analogy: everyone hears the traffic reporter in early morning radio broadcast, who gives information about traffic situation on all main streets to those hurrying drivers on their way to work. If we assume this reporter does not use the radio, but rather drives a car to distribute his information (forget how he gets it), then he may likely be stuck in a jam himself, just as anyone else, and his effect will be meaningless! Without priority in network transmission, the proposed congestion control would result in a similar situation.

5.1.4. If some hosts do not respond to control message

What should a GW do in this case? One way to punish those hosts that keep sending at high rate after a fairly long period the GW sent them control messages is to selectively put packets from those hosts to the end of the waiting queue if buffer space allows and drop them otherwise. Care must be taken in order not to let them affect packets from other hosts that do support the network congestion control.

5.2. Host reactions to the control message

Basically, I think the host should trust the gateway and set the transmission rate according to the control message if it receives one. That is its share of the network resources given by the GW, and the GW has the full control over it anyway. Fairness issue is the gateway policy and hosts cannot change it. Intuitively, the network should be willing to accept as much traffic as it can afford because that is the purpose it is there for.

The proposed congestion control requires that each host have control on its transmission rate to each destination host. One suggestion to do this is to build a transmission table in the IP module at each host. Each destination of packet transmissions will be the identifier for each entry in the table.

The host may record how many packets it sends out to different destinations during a certain time period, or may put nothing in the table until receiving control messages from the network. Upon receiving a control message from the GW, it should record the control limit in the table and reject further sending requests from the higher level protocols when it reaches the transmission limit. Or it may pass the limit directly to the higher protocols to ask them to send out packets accordingly. There are probably some implementation difficulties here. At IP level, GWs only see hosts, while at one level above, a host may have several protocols using the same IP module, and a protocol may again have several simultaneous *connections* to the same destination. Thus a host, especially a time-sharing mainframe, might face a problem of sharing given network resources among these protocols and connections. Someone will need to work out this sharing problem. But I do not expect it to be as hard as congestion control.[17] Let me repeat the thought one more time to make things clearer: it will not be very difficult to propose a control algorithm of the sharing for a host, but it does seem difficult to make an implementation, because we have no experience on how to do it, nor some concrete prediction about how good the performance can be.

An even better thing the host can do is to take into account the transmission delay upon receiving a control message. From the transmission table, it may compute the excess traffic it has sent out before receiving the message and stop for a proper period to let the GW drain out those excess packets. Ideally, this will help GWs avoid accumulating up too many packets and hosts will not suffer long delay.

Although we give the GW the full control on network traffic, a host still cannot rely on listening to the control information from GWs. A GW may fail and lose its previous control information, and this may result in locking some hosts at low rate for a long time. To prevent this situation, a host may ask the GWs, from which it received control messages before, for new transmission rate permission either periodically or when necessary.

---

[17] There are fundamental differences between the two:

~ This sharing control in the host does not suffer any delay.

~ The control overhead is independent of the shared network resources.

This is one of the reasons that I suggest to have separate control algorithms for the net and within a host. I also think that the network's algorithm should be well defined and known to all users, while the sharing policy in a host should be left to each host to decide.

Because a packet stream may travel through a series of GWs, a host may receive control messages from several GWs with different BP-BL values. In such case it should limit its transmission to the lowest rate.

5.3. Information Needed for This Algorithm

To implement the above algorithm, a GW need know

1. the capacities of its CPU and output lines for processing and transmitting data packets;
2. its current traffic load for the CPU and each outbound lines;
3. The source-destination host pairs of current packet streams, which will be used to send control messages when congestion occurs.

(1) is assumed to be constant and known (it may be the GW's real capacity decreased by a small percentage, which is used for handling ICMP packets). We use the packet queue lengths to measure (2), because

~ The queue lengths are easy to know;

~ it can be used to measure both lines and CPU loads: the length of total input queue for measuring CPU load, and the output queue length at each outbound line for line load.

The processing time for each data packet is independent of the packet length, so the unprocessed packet queue length is an accurate measure of CPU load (forget the byte swapping!). The line load, however, is packet length dependent, hence the number of packets waiting at a outbound line may not be an accurate measure of the line load. We expect that the average packet length does not vary drastically, and hence ignore this inaccuracy unless later on some evidence shows that it is not negligible.

(3) is much harder to get. From the traffic pattern assumed above, we need a table to record those packet streams that are contributing significant part of the current net traffic, and ignore other packets with low frequency, random occurrence. We imagine a device like a *high pass filter (HPF)* in electrical circuits that can filter out low frequency traffic, and build a packet stream table from the "HPF" output (see Figure 1). Whenever the GW is becoming overloaded, it will look up this stream table and send a control message to those source hosts on the table, requesting them to restrict the transmission rate by a specific limit in order to release the situation. When the load decreases, the GW can then reset the limit to a new value.

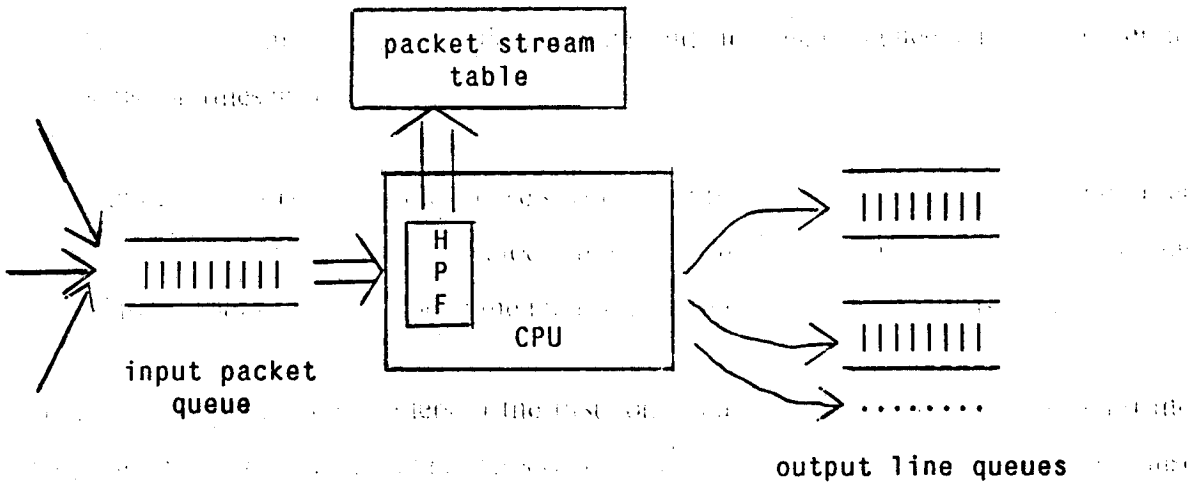

Figure 1: Outline of a GW structure.

5.3.1. How to build the HPF

We are actually looking for some tool that only counts the packets that arrive at the GW with the same source-destination addresses at relatively high frequencies, and then we record this information in the packet stream table. We also expect that the packets of a bursty transmission will arrive at the gateway closely to each other. Therefore we build the HPF in the following way:

1. It is a table with two components in each entry: a source-destination host pair identifier of a packet stream, and a counter of packets sent from the source to the destination. The table is empty initially.
2. When CPU gets a packet from the input queue, it looks up the table from bottom up, increases the counter if the same source-destination pair has been recorded already, or makes a new entry in the table if not found.
3. This HPF table is reset periodically. Every T_{HPF} (= HPF reset period) milli-seconds, the packet stream table will be updated by those source-destination pairs with the number of packets transmitted greater than N in the HPF. Then the HPF will be cleared and step 2 will be repeated again.

Because the HPF must be looked up for every packet, the lookup should be done fast. HPF table is expected to be small, i.e. the reset period should not be long, and there are not many simultaneous connections passing through a gateway at one instant.

5.3.2. How to manage the packet stream table

Each current packet stream has an entry in this table. Since this table is the basis on which the GW does congestion control, we require that

1. It should be easy/fast to update and to look up.
2. It is not a counter of the number of packets in a stream, but a meter measuring the source host sending rates. Due to differences in higher layer protocol implementations and the traffic fluctuation in the net, we do not expect packets of a stream arriving with an accurate rate. Thus this meter should record the average rate of each stream over a proper period to avoid the rate fluctuating too much.
3. It must be able to automatically erase those streams that have terminated or have changed routes in a short period to keep the correct traffic information.
4. It should also contain the information of whether the source host of each packet stream has been sent a control message, and if so, the transmission rate limitation in the control message.

To meet the first requirement, we set the stream table to a limited size, only those streams contributing significant part to the net traffic are put under control. We may also divide it into subtables according to the output lines, so if one line is overloaded, it is easy to find out all packet streams flowing out that line.

To meet requirements 2 and 3, we set an auto-erase period T_E and erase the table in the following way:

1. There are two subfields in each table entry to record the number of packets of that stream. We use one at a time and switch between the two every T_E seconds.
2. At the time of switching, if a stream has no packet sent during the last T_E seconds, it is erased from the table.

source-destination host of the stream	# packets in first T_E	# packets in second T_E	control msg recording

When we look up the table, we consider the sum of the numbers in the two subfields and get the average rate over $2T_E$ period. We will always have information available even when aging out old information. Note also that the time periods T_{HPF} and T_E are independent, and T_{HPF} is expected to be much smaller than T_E to keep the HPF small in size.

5.4. Problems to Be Worked Out

Under above outline, there still remain some problems to be solved.

1. Get fairly accurate parameters on the things below

   ~ the CPU packet processing capacity at the gateway.
   ~ BP-BL as a function of the GW buffer space, capacity, and the load.
   ~ The multiple queue threshold values for measuring the GW load.

2. Estimate the total overhead of the control algorithm.

3. Estimate the performance of this algorithm by simulation.

4. The hardest one: investigate the relation between the routing and congestion control. Using dynamic routing to spread the congestion does not seem correct, though routing being independent of network load does not sound right, either. On the other hand, congestion control, being distributed, does require some knowledge about routing to work well.

6. Analysis and Conclusions

In terms of implementation, the above proposed congestion control algorithm requires a fairly large buffer space to handle unpredictable packet bursts. All the control tables will not take much memory. It does not need a fine clock, even 60 ticks per second will do. The algorithm itself is simple, but it will take some amount of CPU cycles because some check has to be done for every packet. The amount of traffic it will create for sending control messages is adjustable (by changing the number of control thresholds in load measuring). I am coding a simulator for this congestion control algorithm to verify that

~ A controlled network can have higher throughput and lower delay than an uncontrolled one, especially under heavy load condition.
~ The saving of retransmissions will be much greater than the control overhead.

The congestion control algorithm cannot be independent from other network functions. For example, its responsiveness will largely depend on whether the network supports priority transmission. If a control message has to wait in long transmission queues in the heavy load situation, the performance of congestion control itself will be load dependent, hence the goal of *control* can hardly to achieve. It is possible to implement good priority transmission at a gateway. For example, in the C-Gateway every packet can be examined before being processed, and we can

easily put those packets with high priority to the fronts of processing queue and the transmission queue to make their transmission delay traffic load independent. One thing needed now is to set up priority transmission rules in IP.

When designing a network level protocol, we should consider that the network is a distributed set of shared resources, and build into the protocol the functions of controlling the sharing among users. This control should be seen through the interface to the higher level protocols.

Look back ten years ago, the builders of the first computer network might not understand the functional position of buffering at the packet-switch node as we think it now. They probably just put whatever seemed to be economic or reasonable[18]. But even they had thought the same way for doing congestion control, they would not be able to implement anyway due to lack of hardware support. While we are designing the congestion control based on today's computation and communication facilities, people working on hardware are designing new machines and devices for tomorrow. It is probably hard to imagine how fast the line speed will be for a new network that will be built up ten years later, and the congestion control mechanism might change to totally different contents (imagine that all Arpanet lines were 10 mbps and all IMPs were 10 times faster, the network would be quite different). But the hardware technology by itself will not solve congestion problem, no matter how fast a network can run. We should expect user demand increasing, too. And in any case, certain control or regulation functions will always be needed for shared resources.

In a recent paper, Kleinrock gave an interesting remark to congestion control (well, remember he calls it flow control):"It seems fair to say that flow control has the following cantankerous properties: you need it, it's tough to design, it's nearly impossible to analyze,and it's almost sure to cause you trouble (i.e. create deadlocks, degradations, and other lovely castastrophes)." [3] We are facing a hard problem that has not been solved over years. But after we gained a better understanding and try to attack it from a new direction, we hope to get some interesting new results.

---

[18] An early paper by R. Kahn [2] gives some reasoning about the choice, which shows different viewpoint than ours.

References

[1]

ISO-SC 16.
Reference Model of Open Systems Architecture.
Technical Report ISO/TC97/SC16, International Standards Organization, November, 1978.

[2]

Robert E. Kahn.
Resource-Sharing Computer Communications Networks.
IEEE Proceedings (11), November, 1972.

[3]

L. Kleinrock.
A Decade of Network Development.
Journal of Telecommunication Networks, Spring, 1982.

[4]

Haldane R. Peterson.
Design of Source Quench Congestion Control Algorithms in Interconnected Networks.
MIT-LCS B.S. Thesis, May 1980.

[5]

J. Postel.
User Datagram Protocol.
Technical Report IEN 88, USC-ISI, May, 1979.

[6]

J. Postel.
Internet Control Message Protocol.
Technical Report RFC 792, DARPA, September, 1981.

[7]

Karen Sollins.
The TFTP Protocol.
Technical Report RFC783, MIT-LCS, June, 1981.

[8]

Dan Theriault.
BLAST, an Experimental File Transfer Protocol.
Technical Report CSR-RFC217, MIT-LCS, March, 1982.

[9]

Lixia Zhang.
A Survey on Congestion Control in Computer Networks.
Technical Report CSR-RFC 249, MIT-LCS, May, 1983.