              Congestion Control in IP/TCP Internetworks

This memo discusses some aspects of congestion control in IP/TCP
Internetworks.   It is intended to stimulate thought and further
discussion of this topic.   While some specific suggestions are
made for improved congestion control implementation, this memo
does not specify any standards.

                           Introduction

Congestion control is a recognized problem in complex networks.
We have discovered that the Department of Defense's Internet Pro-
tocol (IP) , a pure datagram protocol, and Transmission Control
Protocol (TCP), a transport layer protocol, when used together,
are subject to unusual congestion problems caused by interactions
between the transport and datagram layers. In particular, IP
gateways are vulnerable to a phenomenon we call "congestion col-
lapse", especially when such gateways connect networks of widely
different bandwidth. We have developed solutions that prevent
congestion collapse.

These problems are not generally recognized because these proto-
cols are used most often on networks built on top of ARPANET IMP
technology. ARPANET IMP based networks traditionally have uni-
form bandwidth and identical switching nodes, and are sized with
substantial excess capacity. This excess capacity, and the abil-
ity of the IMP system to throttle the transmissions of hosts has
for most IP / TCP hosts and networks been adequate to handle
congestion. With the recent split of the ARPANET into two inter-
connected networks and the growth of other networks with differ-
ing properties connected to the ARPANET, however, reliance on the
benign properties of the IMP system is no longer enough to allow
hosts to communicate rapidly and reliably. Improved handling of
congestion is now mandatory for successful network operation
under load.

Ford Aerospace and Communications Corporation, and its parent
company, Ford Motor Company, operate the only private IP/TCP
long-haul network in existence today. This network connects four
facilities (one in Michigan, two in California, and one in Eng-
land) some with extensive local networks. This net is cross-tied
to the ARPANET but uses its own long-haul circuits; traffic
between Ford facilities flows over private leased circuits,
including a leased transatlantic satellite connection.   All
switching nodes are pure IP datagram switches with no node-to-
node flow control, and all hosts run software either written or
heavily modified by Ford or Ford Aerospace. Bandwidth of links
in this network varies widely, from 1200 to 10,000,000 bits per
second. In general, we have not been able to afford the luxury
of excess long-haul bandwidth that the ARPANET possesses, and our
long-haul links are heavily loaded during peak periods.   Transit
times of several seconds are thus common in our network.

Because of our pure datagram orientation, heavy loading, and wide
variation in bandwidth, we have had to solve problems that the
ARPANET / MILNET community is just beginning to recognize. Our
network is sensitive to suboptimal behavior by host TCP implemen-
tations, both on and off our own net. We have devoted consider-
able effort to examining TCP behavior under various conditions,
and have solved some widely prevalent problems with TCP. We
present here two problems and their solutions. Many TCP imple-
mentations have these problems; if throughput is worse through an
ARPANET / MILNET gateway for a given TCP implementation than
throughput across a single net, there is a high probability that
the TCP implementation has one or both of these problems.

                        Congestion collapse

Before we proceed with a discussion of the two specific problems
and their solutions, a description of what happens when these
problems are not addressed is in order. In heavily loaded pure
datagram networks with end to end retransmission, as switching
nodes become congested, the round trip time through the net
increases and the count of datagrams in transit within the net
also increases. This is normal behavior under load. As long as
there is only one copy of each datagram in transit, congestion is
under control. Once retransmission of datagrams not yet
delivered begins, there is potential for serious trouble.

Host TCP implementations are expected to retransmit packets
several times at increasing time intervals until some upper limit
on the retransmit interval is reached. Normally, this mechanism
is enough to prevent serious congestion problems. Even with the
better adaptive host retransmission algorithms, though, a sudden
load on the net can cause the round-trip time to rise faster than
the sending hosts measurements of round-trip time can be updated.
Such a load occurs when a new bulk transfer, such a file
transfer, begins and starts filling a large window. Should the
round-trip time exceed the maximum retransmission interval for
any host, that host will begin to introduce more and more copies
of the same datagrams into the net. The network is now in seri-
ous trouble. Eventually all available buffers in the switching
nodes will be full and packets must be dropped. The round-trip
time for packets that are delivered is now at its maximum. Hosts
are sending each packet several times, and eventually some copy
of each packet arrives at its destination. This is congestion
collapse.

This condition is stable. Once the saturation point has been
reached, if the algorithm for selecting packets to be dropped is
fair, the network will continue to operate in a degraded condi-
tion. In this condition every packet is being transmitted
several times and throughput is reduced to a small fraction of
normal. We have pushed our network into this condition experi-
mentally and observed its stability. It is possible for round-
trip time to become so large that connections are broken because

the hosts involved time out.

Congestion collapse and pathological congestion are not normally
seen in the ARPANET / MILNET system because these networks have
substantial excess capacity.   Where connections do not pass
through IP gateways, the IMP-to host flow control mechanisms usu-
ally prevent congestion collapse, especially since TCP implemen-
tations tend to be well adjusted for the time constants associ-
ated with the pure ARPANET case.   However, other than ICMP Source
Quench messages, nothing fundamentally prevents congestion col-
lapse when TCP is run over the ARPANET / MILNET and packets are
being dropped at gateways. Worth noting is that a few badly-
behaved hosts can by themselves congest the gateways and prevent
other hosts from passing traffic. We have observed this problem
repeatedly with certain hosts (with whose administrators we have
communicated privately) on the ARPANET.

Adding additional memory to the gateways will not solve the prob-
lem.   The more memory added, the longer round-trip times must
become before packets are dropped.   Thus, the onset of congestion
collapse will be delayed but when collapse occurs an even larger
fraction of the packets in the net will be duplicates and
throughput will be even worse.

The two problems

Two key problems with the engineering of TCP implementations have
been observed; we call these the small-packet problem and the
source-quench problem.   The second is being addressed by several
implementors; the first is generally believed (incorrectly) to be
solved. We have discovered that once the small-packet problem
has been solved, the source-quench problem becomes much more
tractable. We thus present the small-packet problem and our
solution to it first.

The small-packet problem

There is a special problem associated with small packets.   When
TCP is used for the transmission of single-character messages
originating at a keyboard, the typical result is that 41 byte
packets (one byte of data, 40 bytes of header) are transmitted
for each byte of useful data.   This 4000% overhead is annoying
but tolerable on lightly loaded networks. On heavily loaded net-
works, however, the congestion resulting from this overhead can
result in lost datagrams and retransmissions, as well as exces-
sive propagation time caused by congestion in switching nodes and
gateways.   In practice, throughput may drop so low that TCP con-
nections are aborted.

This classic problem is well-known and was first addressed in the
Tymnet network in the late 1960s. The solution used there was to
impose a limit on the count of datagrams generated per unit time.
This limit was enforced by delaying transmission of small packets

until a short (200-500ms) time had elapsed, in hope that another
character or two would become available for addition to the same
packet before the timer ran out.   An additional feature to
enhance user acceptability was to inhibit the time delay when a
control character, such as a carriage return, was received.

This technique has been used in NCP Telnet, X.25 PADs, and TCP
Telnet. It has the advantage of being well-understood, and is not
too difficult to implement. Its flaw is that it is hard to come
up with a time limit that will satisfy everyone. A time limit
short enough to provide highly responsive service over a 10M bits
per second Ethernet will be too short to prevent congestion col-
lapse over a heavily loaded net with a five second round-trip
time; and conversely, a time limit long enough to handle the
heavily loaded net will produce frustrated users on the Ethernet.

              The solution to the small-packet problem

Clearly an adaptive approach is desirable. One would expect a
proposal for an adaptive inter-packet time limit based on the
round-trip delay observed by TCP. While such a mechanism could
certainly be implemented, it is unnecessary.   A simple and
elegant solution has been discovered.

The solution is to inhibit the sending of new TCP segments when
new outgoing data arrives from the user if any previously
transmitted data on the connection remains unacknowledged.   This
inhibition is to be unconditional; no timers, tests for size of
data received, or other conditions are required.   Implementation
typically requires one or two lines inside a TCP program.

At first glance, this solution seems to imply drastic changes in
the behavior of TCP. This is not so. It all works out right in
the end. Let us see why this is so.

When a user process writes to a TCP connection, TCP receives some
data.   It may hold that data for future sending or may send a
packet immediately. If it refrains from sending now, it will
typically send the data later when an incoming packet arrives and
changes the state of the system. The state changes in one of two
ways; the incoming packet acknowledges old data the distant host
has received, or announces the availability of buffer space in
the distant host for new data. (This last is referred to as
"updating the window").   Each time data arrives on a connec-
tion, TCP must reexamine its current state and perhaps send some
packets out. Thus, when we omit sending data on arrival from the
user, we are simply deferring its transmission until the next
message arrives from the distant host. A message must always
arrive soon unless the connection was previously idle or communi-
cations with the other end have been lost. In the first case,
the idle connection, our scheme will result in a packet being
sent whenever the user writes to the TCP connection. Thus we do
not deadlock in the idle condition. In the second case, where

the distant host has failed, sending more data is futile anyway.
Note that we have done nothing to inhibit normal TCP retransmis-
sion logic, so lost messages are not a problem.

Examination of the behavior of this scheme under various condi-
tions demonstrates that the scheme does work in all cases. The
first case to examine is the one we wanted to solve, that of the
character-oriented Telnet connection.   Let us suppose that the
user is sending TCP a new character every 200ms, and that the
connection is via an Ethernet with a round-trip time including
software processing of 50ms. Without any mechanism to prevent
small-packet congestion, one packet will be sent for each charac-
ter, and response will be optimal. Overhead will be 4000%, but
this is acceptable on an Ethernet. The classic timer scheme,
with a limit of 2 packets per second, will cause two or three
characters to be sent per packet. Response will thus be degraded
even though on a high-bandwidth Ethernet this is unnecessary.
Overhead will drop to 1500%, but on an Ethernet this is a bad
tradeoff. With our scheme, every character the user types will
find TCP with an idle connection, and the character will be sent
at once, just as in the no-control case. The user will see no
visible delay.   Thus, our scheme performs as well as the no-
control scheme and provides better responsiveness than the timer
scheme.

The second case to examine is the same Telnet test but over a
long-haul link with a 5-second round trip time. Without any
mechanism to prevent small-packet congestion, 25 new packets
would be sent in 5 seconds.* Overhead here is 4000%.   With the
classic timer scheme, and the same limit of 2 packets per second,
there would still be 10 packets outstanding and contributing to
congestion. Round-trip time will not be improved by sending many
packets, of course; in general it will be worse since the packets
will contend for line time. Overhead now drops to 1500%. With
our scheme, however, the first character from the user would find
an idle TCP connection and would be sent immediately. The next
24 characters, arriving from the user at 200ms intervals, would
be held pending a message from the distant host. When an ACK
arrived for the first packet at the end of 5 seconds, a single
packet with the 24 queued characters would be sent. Our scheme
thus results in an overhead reduction to 320% with no penalty in
response time.   Response time will usually be improved with our
scheme because packet overhead is reduced, here by a factor of
4.7 over the classic timer scheme. Congestion will be reduced by
this factor and round-trip delay will decrease sharply. For this

_____

  * This problem is not seen in the pure ARPANET case because the
    IMPs will block the host when the count of packets
    outstanding becomes excessive, but in the case where a pure
    datagram local net (such as an Ethernet) or a pure datagram
    gateway (such as an ARPANET / MILNET gateway) is involved, it
    is possible to have large numbers of tiny packets
    outstanding.

case, our scheme has a striking advantage over either of the
other approaches.

We use our scheme for all TCP connections, not just Telnet con-
nections.   Let us see what happens for a file transfer data con-
nection using our technique. The two extreme cases will again be
considered.

As before, we first consider the Ethernet case. The user is now
writing data to TCP in 512 byte blocks as fast as TCP will accept
them. The user's first write to TCP will start things going; our
first datagram will be 512+40 bytes or 552 bytes long. The
user's second write to TCP will not cause a send but will cause
the block to be buffered. Assume that the user fills up TCP's
outgoing buffer area before the first ACK comes back. Then when
the ACK comes in, all queued data up to the window size will be
sent. From then on, the window will be kept full, as each ACK
initiates a sending cycle and queued data is sent out. Thus,
after a one round-trip time initial period when only one block is
sent, our scheme settles down into a maximum-throughput condi-
tion. The delay in startup is only 50ms on the Ethernet, so the
startup transient is insignificant. All three schemes provide
equivalent performance for this case.

Finally, let us look at a file transfer over the 5-second round
trip time connection. Again, only one packet will be sent until
the first ACK comes back; the window will then be filled and kept
full.   Since the round-trip time is 5 seconds, only 512 bytes of
data are transmitted in the first 5 seconds. Assuming a 2K win-
dow, once the first ACK comes in, 2K of data will be sent and a
steady rate of 2K per 5 seconds will be maintained   thereafter.
Only for this case is our scheme inferior to the timer scheme,
and the difference is only in the startup transient; steady-state
throughput is identical. The naive scheme and the timer scheme
would both take 250 seconds to transmit a 100K byte file under
the above conditions and our scheme would take 254 seconds, a
difference of 1.6%.

Thus, for all cases examined, our scheme provides at least 98% of
the performance of both other schemes, and provides a dramatic
improvement in Telnet performance over paths with long round trip
times. We use our scheme in the Ford Aerospace Software
Engineering Network, and are able to run screen editors over Eth-
ernet and talk to distant TOPS-20 hosts with improved performance
in both cases.

Congestion control with ICMP

Having solved the small-packet congestion problem and with it the
problem of excessive small-packet congestion within our own net-
work, we turned our attention to the problem of general conges-
tion control.   Since our own network is pure datagram with no
node-to-node flow control, the only mechanism available to us

under the IP standard was the ICMP Source Quench message. With
careful handling, we find this adequate to prevent serious
congestion problems. We do find it necessary to be careful about
the behavior of our hosts and switching nodes regarding Source
Quench messages.

              When to send an ICMP Source Quench

The present ICMP standard* specifies that an ICMP Source Quench
message should be sent whenever a packet is dropped, and addi-
tionally may be sent when a gateway finds itself becoming short
of resources.   There is some ambiguity here but clearly it is a
violation of the standard to drop a packet without sending an
ICMP message.

Our basic assumption is that packets ought not to be dropped dur-
ing normal network operation.   We therefore want to throttle
senders back before they overload switching nodes and gateways.
All our switching nodes send ICMP Source Quench messages well
before buffer space is exhausted; they do not wait until it is
necessary to drop a message before sending an ICMP Source Quench.
As demonstrated in our analysis of the small-packet problem,
merely providing large amounts of buffering is not a solution.
In general, our experience is that Source Quench should be sent
when about half the buffering space is exhausted; this is not
based on extensive experimentation but appears to be a reasonable
engineering decision.   One could argue for an adaptive scheme
that adjusted the quench generation threshold based on recent
experience; we have not found this necessary as yet.

There exist other gateway implementations that generate Source
Quenches only after more than one packet has been discarded. We
consider this approach undesirable since any system for control-
ling congestion based on the discarding of packets is wasteful of
bandwidth and may be susceptible to congestion collapse under
heavy load.   Our understanding is that the decision to generate
Source Quenches with great reluctance stems from a fear that ack-
nowledge traffic will be quenched and that this will result in
connection failure. As will be shown below, appropriate handling
of Source Quench in host implementations eliminates this possi-
bility.

              What to do when an ICMP Source Quench is received

We inform TCP or any other protocol at that layer when ICMP
receives a Source Quench. The basic action of our TCP implemen-
tations is to reduce the amount of data outstanding on connec-
tions to the host mentioned in the Source Quench. This control is
_____
   * ARPANET RFC 792 is the present standard. We are advised by
     the Defense Communications Agency that the description of
     ICMP in MIL-STD-1777 is incomplete and will be deleted from
     future revision of that standard.

applied by causing the sending TCP to behave as if the distant
host's window size has been reduced. Our first implementation
was simplistic but effective; once a Source Quench has been
received our TCP behaves as if the window size is zero whenever
the window isn't empty.   This behavior continues until some
number (at present 10) of ACKs have been received, at that time
TCP returns to normal operation.* David Mills of Linkabit Cor-
poration has since implemented a similar but more elaborate
throttle on the count of outstanding packets in his DCN systems.
The additional sophistication seems to produce a modest gain in
throughput, but we have not made formal tests. Both implementa-
tions effectively prevent congestion collapse in switching nodes.

Source Quench thus has the effect of limiting the connection to a
limited number (perhaps one) of outstanding messages. Thus, com-
munication can continue but at a reduced rate, that is exactly
the effect desired.

This scheme has the important property that Source Quench doesn't
inhibit the sending of acknowledges or retransmissions. Imple-
mentations of Source Quench entirely within the IP layer are usu-
ally unsuccessful because IP lacks enough information to throttle
a connection properly. Holding back acknowledges tends to pro-
duce retransmissions and thus unnecessary traffic. Holding back
retransmissions may cause loss of a connection by a retransmis-
sion timeout.   Our scheme will keep connections alive under
severe overload but at reduced bandwidth per connection.

Other protocols at the same layer as TCP should also be respon-
sive to Source Quench. In each case we would suggest that new
traffic should be throttled but acknowledges should be treated
normally.   The only serious problem comes from the User Datagram
Protocol, not normally a major traffic generator.   We have not
implemented any throttling in these protocols as yet; all are
passed Source Quench messages by ICMP but ignore them.

Self-defense for gateways

As we have shown, gateways are vulnerable to host mismanagement
of congestion. Host misbehavior by excessive traffic generation
can prevent not only the host's own traffic from getting through,
but can interfere with other unrelated traffic. The problem can
be dealt with at the host level but since one malfunctioning host
can interfere with others, future gateways should be capable of
defending themselves against such behavior by obnoxious or mali-
cious hosts. We offer some basic self-defense techniques.

On one occasion in late 1983, a TCP bug in an ARPANET host caused
the host to frantically generate retransmissions of the same
datagram as fast as the ARPANET would accept them.   The gateway

---

 * This follows the control engineering dictum "Never bother
   with proportional control unless bang-bang doesn't work".

that connected our net with the ARPANET was saturated and little useful traffic could get through, since the gateway had more bandwidth to the ARPANET than to our net.   The gateway busily sent ICMP Source Quench messages but the malfunctioning host ignored them. This continued for several hours, until the mal-functioning host crashed.   During this period, our network was effectively disconnected from the ARPANET.

When a gateway is forced to discard a packet, the packet is selected at the discretion of the gateway. Classic techniques for making this decision are to discard the most recently received packet, or the packet at the end of the longest outgoing queue. We suggest that a worthwhile practical measure is to dis-card the latest packet from the host that originated the most packets currently queued within the gateway. This strategy will tend to balance throughput amongst the hosts using the gateway. We have not yet tried this strategy, but it seems a reasonable starting point for gateway self-protection.

Another strategy is to discard a newly arrived packet if the packet duplicates a packet already in the queue. The computa-tional load for this check is not a problem if hashing techniques are used.   This check will not protect against malicious hosts but will provide some protection against TCP implementations with poor retransmission control.   Gateways between fast local net-works and slower long-haul networks may find this check valuable if the local hosts are tuned to work well with the local network.

Ideally the gateway should detect malfunctioning hosts and squelch them; such detection is difficult in a pure datagram sys-tem. Failure to respond to an ICMP Source Quench message, though,   should be regarded as grounds for action by a gateway to disconnect a host. Detecting such failure is non-trivial but is a worthwhile area for further research.

<center>Conclusion</center>

The congestion control problems associated with pure datagram networks are difficult, but effective solutions exist. If IP / TCP networks are to be operated under heavy load, TCP implementa-tions must address several key issues in ways at least as effec-tive as the ones described here.