Vulnerabilities of Network Control Protocols: An Example

Eric C. Rosen

Bolt Beranek and Newman Inc.

This paper has appeared in the January 1981 edition of the
SIGSOFT Software Engineering Notes, and will soon appear in the
SIGCOMM Computer Communications Review. It is being circulated
as an RFC because it is thought that it may be of interest to a
wider audience, particularly to the internet community. It is a
case study of a particular kind of problem that can arise in
large distributed systems, and of the approach used in the
ARPANET to deal with one such problem.

On October 27, 1980, there was an unusual occurrence on the
ARPANET. For a period of several hours, the network appeared to
be unusable, due to what was later diagnosed as a high priority
software process running out of control.   Network-wide
disturbances are extremely unusual in the ARPANET (none has
occurred in several years), and as a result, many people have
expressed interest in learning more about the etiology of this
particular incident. The purpose of this note is to explain what
the symptoms of the problem were, what the underlying causes
were, and what lessons can be drawn.   As we shall see, the
immediate cause of the problem was a rather freakish hardware
malfunction (which is not likely to recur) which caused a faulty
sequence of network control packets to be generated. This faulty
sequence of control packets in turn affected the apportionment of
software resources in the IMPs, causing one of the IMP processes
to use an excessive amount of resources, to the detriment of
other IMP processes.   Restoring the network to operational

- 1 -

condition was a relatively straightforward task. There was no
damage other than the outage itself, and no residual problems
once the network was restored. Nevertheless, it is quite
interesting to see the way in which unusual (indeed, unique)
circumstances can bring out vulnerabilities in network control
protocols, and that shall be the focus of this paper.

The problem began suddenly when we discovered that, with
very few exceptions, no IMP was able to communicate reliably with
any other IMP. Attempts to go from a TIP to a host on some other
IMP only brought forth the "net trouble" error message,
indicating that no physical path existed between the pair of
IMPs. Connections which already existed were summarily broken.
A flood of phone calls to the Network Control Center (NCC) from
all around the country indicated that the problem was not
localized, but rather seemed to be affecting virtually every IMP.

As a first step towards trying to find out what the state of
the network actually was, we dialed up a number of TIPs around
the country. What we generally found was that the TIPs were up,
but that their lines were down. That is, the TIPs were
communicating properly with the user over the dial-up line, but
no connections to other IMPs were possible.

We tried manually restarting a number of IMPs which are in
our own building (after taking dumps, of course). This procedure
initializes all of the IMPs' dynamic data structures, and will

often clear up problems which arise when, as sometimes happens in

most complex software systems, the IMPs' software gets into a

"funny" state.   The IMPs which were restarted worked well until

they were connected to the rest of the net, after which they

exhibited the same complex of symptoms as the IMPs which had not

been restarted.

    From the facts so far presented, we were able to draw a

number of conclusions.   Any problem which affects all IMPs

throughout the network is usually a routing problem.   Restarting

an IMP re-initializes the routing data structures, so the fact

that restarting an IMP did not alleviate the problem in that IMP

suggested that the problem was due to one or more "bad" routing

updates circulating in the network. IMPs which were restarted

would just receive the bad updates from those of their neighbors

which were not restarted. The fact that IMPs seemed unable to

keep their lines up was also a significant clue as to the nature

of the problem. Each pair of neighboring IMPs runs a line

up/down protocol to determine whether the line connecting them is

of sufficient quality to be put into operation. This protocol

involves the sending of HELLO and I-HEARD-YOU messages. We have

noted in the past that under conditions of extremely heavy CPU

utilization, so many buffers can pile up waiting to be served by

the bottleneck CPU process, that the IMPs are unable to acquire

the buffers needed for receiving the HELLO or I-HEARD-YOU

messages. If a condition like this lasts for any length of time,

the IMPs may not be able to run the line up/down protocol, and

lines will be declared down by the IMPs' software. On the basis

of all these facts, our tentative conclusion was that some

malformed update was causing the routing process in the IMPs to

use an excessive amount of CPU time, possibly even to be running

in an infinite loop. (This would be quite a surprise though,

since we tried very hard to protect ourselves against malformed

updates when we designed the routing process.) As we shall see,

this tentative conclusion, although on the right track, was not

quite correct, and the actual situation turned out to be much

more complex.

        When we examined core dumps from several IMPs, we noted that

most, in some cases all, of the IMPs' buffers contained routing

updates waiting to be processed. Before describing this

situation further, it is necessary to explain some of the details

of the routing algorithm's updating scheme. (The following

explanation will of course be very brief and incomplete. Readers

with a greater level of interest are urged to consult the

references.) Every so often, each IMP generates a routing update

indicating which other IMPs are its immediate neighbors over

operational lines, and the average per-packet delay (in

milliseconds) over that line. Every IMP is required to generate

such an update at least once per minute, and no IMP is permitted

to generate more than a dozen such updates over the course of a

minute. Each update has a 6-bit sequence number which is

advanced by 1 (modulo 64) for each successive update generated by
a particular IMP. If two updates generated by the same IMP have
sequence numbers n and m, update n is considered to be LATER
(i.e., more recently generated) than update m if and only if one
of the following two conditions hold:

        (a) n > m, and n - m <= 32
        (b) n < m, and m - n > 32

(where the comparisons and subtractions treat n and m as unsigned
6-bit numbers, with no modulus).   When an IMP generates an
update, it sends a copy of the update to each neighbor. When an
IMP A receives an update u1 which was generated by a different
IMP B, it first compares the sequence number of u1 with the
sequence number of the last update, u2, that it accepted from B.
If this comparison indicates that u2 is LATER than u1, u1 is
simply discarded. If, on the other hand, u1 appears to be the
LATER update, IMP A will send u1 to all its neighbors (including
the one from which it was received). The sequence number of u1
will be retained in A's tables as the LATEST received update from
B. Of course, u1 is always accepted if A has seen no previous
update from B. Note that this procedure is designed to ensure
that an update generated by a particular IMP is received,
unchanged, by all other IMPs in the network, IN THE PROPER
SEQUENCE.   Each routing update is broadcast (or flooded) to all
IMPs, not just to immediate neighbors of the IMP which generated

the update (as in some other routing algorithms). The purpose of

the sequence numbers is to ensure that all IMPs will agree as to

which update from a given IMP is the most recently generated

update from that IMP.

     For reliability, there is a protocol for retransmitting

updates over individual links. Let X and Y be neighboring IMPs,

and let A be a third IMP. Suppose X receives an update which was

generated by A, and transmits it to Y. Now if in the next 100 ms

or so, X does not receive from Y an update which originated at A

and whose sequence number is at least as recent as that of the

update X sent to Y, X concludes that its transmission of the

update did not get through to Y, and that a retransmission is

required.   (This conclusion is warranted, since an update which

is received and adjudged to be the most recent from its

originating IMP is sent to all neighbors, including the one from

which it was received.) The IMPs do not keep the original update

packets buffered pending retransmission. Rather, all   the

information in the update packet is kept in tables, and the

packet is re-created from the tables if necessary for   a

retransmission.

     This transmission protocol ("flooding") distributes the

routing updates in a very rapid and reliable manner.   Once

generated by an IMP, an update will almost always reach all other

IMPs in a time period on the order of 100 ms. Since an IMP can

generate no more than a dozen updates per minute, and there are

64 possible sequence numbers, sequence number wrap-around is not

a problem. There is only one exception to this.   Suppose two

IMPs A and B are out of communication for a period of time

because there is no physical path between them. (This may be due

either to a network partition, or to a more mundane occurrence,

such as one of the IMPs being down.) When communication is

re-established, A and B have no way of knowing how long they have

been out of communication, or how many times the other's sequence

numbers may have wrapped around. Comparing the sequence number

of a newly received update with the sequence number of an update

received before the outage may give an incorrect result. To deal

with this problem, the following scheme is adopted.   Let $t0$ be

the time at which IMP A receives update number n generated by IMP

B.   Let $t1$ be $t0$ plus 1 minute. If by $t1$, A receives no update

generated by B with a LATER sequence number than n, A will accept

any update from B as being more recent than n. So if two IMPs

are out of communication for a period of time which is long

enough for the sequence numbers to have wrapped around, this

procedure ensures that proper resynchronization of sequence

numbers is effected when communication is re-established.

    There is just one more facet of the updating process which

needs to be discussed.   Because of the way the line up/down

protocol works, a line cannot be brought up until 60 seconds

after its performance becomes good enough to warrant operational

use. (Roughly speaking, this is the time it takes to determine

that the line's performance is good enough.) During this
60-second period, no data is sent over the line, but routing
updates are transmitted. Remember that every node is required to
generate a routing update at least once per minute. Therefore,
this procedure ensures that if two IMPs are out of communication
because of the failure of some line, each has the most recent
update from the other by the time communication is
re-established.

    This very short introduction to the routing algorithm's
updating protocol should provide enough background to enable the
reader to understand the particular problem under discussion;
further justification and detail can be found in the references.

    Let us return now to the discussion of the network outage.
I have already mentioned that the core dumps showed almost all
buffers holding routing updates which were waiting to be
processed. Close inspection showed that all the updates were
from a single IMP, IMP 50. By a strange "coincidence," IMP 50
had been malfunctioning just before the network-wide outage
occurred, and was off the net during the period of the outage.
Hence it was not generating any updates during the period of the
outage. In addition, IMP 29, an immediate neighbor of IMP 50,
was also suffering hardware malfunctions (in particular, dropping
bits), but was up (though somewhat flakey) while the network was
in bad shape. Furthermore, the malfunction in IMP 50 had to do
with its ability to communicate properly with the neighboring IMP

29. Although we did not yet understand how it was possible for
so many updates from one IMP to be extant simultaneously, we did
understand enough to be able to get the network to recover.   All
that was necessary was to patch the IMPs to disregard any updates
from IMP 50, which after all was down anyway. When the network
is operating normally, broadcasting a patch to all IMPs can be
done in a matter of minutes. With the network operating as it
was during the period of the outage, this can take as much as 3
or 4 hours. (Remember that the IMPs are generally unmanned, and
that the only way of controlling them from the NCC is via the
network itself.   This is perfectly satisfactory when an outage
affects only a small group of IMPs, but is an obvious problem
when the outage has network-wide effects.) This procedure was
fully successful in bringing the network back up.

     When we looked closely at the dumps, we saw that not only
were all the updates on the queue from IMP 50, but they all had
one of three sequence numbers (either 8, 40, or 44), and were
ordered   in   the   queue   as   follows:
8, 40, 44, 8, 40, 44, 8, 40, 44, ... Note that by the definition
of LATER, 44 is LATER than 40 (44 > 40 and 44 - 40 <= 32), 40 is
LATER than 8 (40 > 8 and 40 - 8 <= 32), and 8 is LATER than 44
(8 < 44 and 44 - 8 > 32). Given the presence of three updates
from the same IMP with these three sequence numbers, this is what
would be expected.   Since each update is LATER than one of the
others, a cycle is formed which keeps the three updates floating

around the network indefinitely.   Thus the IMPs spend most of

their CPU time and buffer space in processing these updates. The

problem was to figure out how these three updates could possibly

have existed at the same time. After all, getting from update 8

to update 40 should require 2 or 3 full minutes, plus 31

intervening sequence numbers.   So how could 8 still be around

when 40 was generated, especially since no   updates   with

intervening sequence numbers were present?

     Our first thought was that maybe the real-time clock in IMP

50 was running one or two orders of magnitude faster than normal,

invalidating our assumptions about the maximum number of updates

which could be generated in a given time.   An alternative

hypothesis suggested itself however when we looked at the binary

representations of the three sequence numbers:

         8 - 001000

        40 - 101000

        44 - 101100

Note that 44 has only one more bit than 40, which has only one

more bit than 8. Furthermore, the three different updates were

completely identical, except for their sequence numbers. This

suggests that there was really only one update, 44, whose

sequence number was twice corrupted by dropped bits. (Of course,

it's also possible that the "real" update was 8, and was

corrupted by added bits. However, bit-dropping has proven itself

to be a much more common sort of hardware malfunction than
bit-adding, although spontaneously dropped bits may sometimes
come back on spontaneously.)

     Surely, the reader will object, there must be protection
against dropped bits.   Yes there is protection, but apparently
not enough. The update packets themselves are checksummed, so a
dropped bit in an update packet is readily detected. Remember
though that if an update needs to be retransmitted, it is
recreated from tabled information. For maximal reliability, the
tables must be checksummed also, and the checksum must be
recomputed every time the table is accessed. However, this would
require either a large number of CPU cycles (for frequent
checksumming of a large area of memory) or a large amount of
memory (to store the checksums for a lot of small areas). Since
CPU cycles and memory are both potentially scarce resources, this
did not seem to us to be a cost-effective way to deal with
problems that arise, say, once per year (this is the first such
problem encountered in a year and a half of running this routing
algorithm).   Time and space can be saved by recomputing the
checksum at a somewhat slower frequency, but this is less
reliable, in that it allows a certain number of dropped bits to
"fall between the cracks." It seems likely then that one of the
malfunctioning IMPs had to retransmit update 44 at least twice,
(recreating it each time from tabled information), retransmitting
it at least once with the corrupted sequence number 40, and at

- 11 -

least once with the corrupted sequence number 8. This would

cause those three sequence numbers to be extant in the network

simultaneously, even though protocol is supposed to ensure that

this is impossible.

     Actually, the detection of dropped bits is most properly a

hardware function. The next generation of IMP hardware (the "C30

IMP") will be able to detect and correct all single-bit errors,

and will detect all other bit errors. Uncorrectable bit errors

will cause the IMP to go into its "loader/dumper." (An IMP in

its loader/dumper is not usable for transferring data, and is

officially in the "down" state.   However, an IMP in its

loader/dumper is easily controllable from the NCC, and can be

restarted or reloaded without on-site intervention.) Current

hardware does have parity checking (which should detect single

dropped bits), but this feature has had to be turned off since

(a) there are too many spurious parity "errors," i.e., most of

the time when the machines complain of parity errors there don't

really seem to be any, and (b) parity errors cause the machines

to simply halt, rather than go into their loader/dumpers, which

means that on-site intervention is required to restart them.

     Pending the introduction of improved hardware, what can be

done to prevent problems like this from recurring in the future?

It is easy to think of many ways of avoiding this particular

problem, especially if one does not consider the problems that

may arise from the "fixes." For example, we might be able to

avoid this sort of problem by spending a lot more CPU cycles on

checksumming, but this may be too expensive because of the side

effects it would introduce.   (Also, it is not clear that any

memory checksumming strategy can be totally free of "cracks.") A

very simple and conservative fix to prevent this particular

problem from recurring is to modify clause (a) of the definition

of LATER so that the "<=" is replaced by "<" (strictly less

than). We will implement this fix, but it cannot be guaranteed

that no related problems will ever arise.

     What is really needed is not some particular fix to the

routing algorithm, but a more general fix. In some sense, the

problem we saw was not really a routing problem. The routing

code was working correctly, and the routes that were generated

were correct and consistent. The real problem is that a freakish

hardware malfunction caused a high priority process to run wild,

devouring resources needed by other processes, thereby making the

network unusable. The fact that the wild process was the routing

process is incidental. In designing the routing process, we

carefully considered the amount of resource utilization it would

require. By strictly controlling and limiting the rate at which

updates can be generated, we tried to prevent any situation in

which the routing process would make excessive demands on the

system.   As we have seen though, even our carefully designed

mechanisms were unable to protect against every possible sort of

hardware failure. We need a better means of detecting that some

high priority process in the IMP, despite all the safeguards we
have built in, is still consuming too many resources. Once this
is detected, the IMP can be automatically placed in   its
loader/dumper. In the case under discussion, we would have liked
to have all the IMPs go into their loader/dumpers when the
problem arose. This would have enabled us to re-initialize and
restart all the IMPs much more quickly. (Although restarting
individual IMPs did little good, restarting all the   IMPs
simultaneously would have cleared up the problem instantly, since
all routing tables in all IMPs would have been initialized
simultaneously.) It took us no more than an hour to figure out
how to restore the network; several additional hours were
required because it took so long for us to gain control of the
misbehaving IMPs and get them back to normal.   A built-in
software alarm system (assuming, of course, that it was not
subject to false alarms) might have enabled us to restore the
network more quickly, significantly reducing the duration of the
outage.   This is not to say that a better alarm and control
system could ever be a replacement for careful study and design
which   attempts   to properly distribute the utilization of
important resources, but only that it is a necessary adjunct, to
handle the cases that will inevitably fall between the cracks of
even the most careful design.

                            REFERENCES

"The New Routing Algorithm for the ARPANET," IEEE TRANSACTIONS ON

COMMUNICATIONS, May 1980, J.M. McQuillan, I. Richer, E.C. Rosen.

"The Updating Protocol of ARPANET's New Routing Algorithm,"

COMPUTER NETWORKS, February 1980, E.C. Rosen.