                      DNS Support for Load Balancing

Status of this Memo

   This memo provides information for the Internet community. This memo
   does not specify an Internet standard of any kind. Distribution of
   this memo is unlimited.

1. Introduction

   This RFC is meant to first chronicle a foray into the IETF DNS
   Working Group, discuss other possible alternatives to
   provide/simulate load balancing support for DNS, and to provide an
   ultimate, flexible solution for providing DNS support for balancing
   loads of many types.

2. History

   The history of this probably dates back well before my own time - so
   undoubtedly some holes are here. Hopefully they can be filled in by
   other authors.

   Initially; "load balancing" was intended to permit the Domain Name
   System (DNS) [1] agents to support the concept of "clusters" (derived
   from the VMS usage) of machines - where all machines were
   functionally similar or the same, and it didn't particularly matter
   which machine was picked - as long as the load of the processing was
   reasonably well distributed across a series of actual different
   hosts. Around 1986 a number of different schemes started surfacing
   as hacks to the Berkeley Internet Name Domain server (BIND)
   distribution. Probably the most widely distributed of these were the
   "Shuffle Address" (SA) modifications by Bryan Beecher, or possibly
   Marshall Rose's "Round Robin" code.

   The SA records, however, did a round-robin ordering of the Address
   resource records, and didn't do much with regard to the particular
   loads on the target machines. Matt Madison (of TGV) implemented some
   changes that used VMS facilities to review the system loads, and
   return A RRs in the order of least-loaded to most loaded.

   The problem was with SAs was that load was not actually a factor, and
   TGV's relied on VMS specific facilities to order the records. The SA
   RRs required changes to the DNS specification (in file syntax and in

Brisco   [Page 1]

record processing). These were both viewed as drawbacks and not as
general solutions.

Most of the Internet waited in anticipation of an IETF approved
method for simulating "clusters".

Through a few IETF DNS Working Group sessions (Chaired by Rob Austein
of Epilogue), it was collectively agreed upon that a number of
criteria must be met:

    A) Backwards compatibility with the existing DNS RFC.

    B) Information changes frequently.

    C) Multiple addresses should be sent out.

    D) Must interact with other RRs appropriately.

    E) Must be able to represent many types of "loads"

    F) Must be fast.

(A) would ensure that the installed base of BIND and other DNS
implementations would continue to operate and interoperate properly.

(B) would permit very fast update times - to enable modeling of
real-time data. Five minutes was thought as a normal interval,
though changes as fast as every sixty seconds could be imagined.

(C) would cover the possibility of a host's address being advertised
as optimal, yet the machine crashed during the period within the TTL
of the RR. The second-most preferable address would be advertised
second, the third-most preferable third, and so on. This would allow
a reasonable stab at recovery during machine failures.

(D) would ensure correct handling of all ancillary information - such
as MX, RP, and TXT information, as well as reverse lookup
information. It needed to be ensured that such processes as mail
handling continued to work in an unsurprising and predictable manner.

(E) would ensure the flexibility that everyone wished. A breadth of
"loads" were wished to be represented by various members of the DNS
Working Group. Some "loads" were fairly eclectic - such as the
address ordering by the RTT to the host, some were pragmatic - such
as balancing the CPU load evenly across a series of hosts. All
represented valid concerns within their own context, and the idea of
having separate RR types for each was unthinkable (primarily; it
would violate goal A).

(F) needed to ensure a few things. Primarily that the time to
calculate the information to order the addressing information did not
exceed the TTL of the information distributed - i.e., that elements
with a TTL of five minutes didn't take six minutes to calculate.
Similarly; it seems a fairly clear goal in the DNS RFC that clients
should not be kept waiting - that request processing should continue
regardless of the state of any other processing occurring.

3. Possible Alternatives

   During various discussions with the DNS Working Group and with the
   Load Balancing Committee, it was noted that no existing solution
   dealt with all wishes appropriately. One of the major successes of
   the DNS is its flexibility - and it was felt that this needed to be
   retained in all aspects. It was conceived that perhaps not only
   address information would need to be changed rapidly, but other
   records may also need to change rapidly (at least this could not be
   ruled out - who knows what technologies lurk in the future).

   Of primary concern to many was the ability to interact with older
   implementations of DNS. The DNS is implemented widely now, and
   changes to critical portions of the protocol could cause havoc for
   years. It became rapidly apparent through conversations with Jon
   Postel and Dave Crocker (Area Director) that modifications to the
   protocol would be viewed dimly.

4. A Flexible Model

   During many hours of discussions, it arose upon suggestion from Rob
   Austein that the changes could be implemented without changes to the
   protocol; if zone transfer behavior could be subtly changed, then the
   zone transfer process could accommodate the changing of various RR
   information. What was needed was a smarter program to do the zone
   transfers. Pursuant to this, changes were made to BIND that would
   permit the specification of the program to do the zone transfers for
   particular zones.

   There is no specification that a secondary has to receive updates
   from its primary server in any specific manner - only that it needs
   to check periodically, and obtain new zone copies when changes have
   been made. Conceivably the zone transfer agent could obtain the
   information from any number of sources (e.g., a load average daemon,
   a round-robin sorter) and present the information back to the
   nameserver for distribution.

   A number of questions arose from this concept, and all seem to have
   been dealt with accordingly. Primarily, the DNS protocol doesn't
   guarantee ordering. While the DNS protocol doesn't guarantee

ordering, it is clear that the ordering is predictive - that
information read in twice in the same order will be presented twice
in the same order to clients. Clients, of course, may reorder this
information, but that is deemed as a "local issue" as it is
configurable by the remote systems administrators (e.g., sortlists,
etc). The zone transfer agent would have to account for any "mis-
ordering" that may occur locally, but remote reordering (e.g., client
side sortlists) of RRs is is impossible to predict. Since local
mis-ordering is consistent, the zone transfer agents could easily
account for this.

Secondarily, but perhaps more subtly, the problem arises that zone
transfers aren't used by primary nameservers, only by secondary
nameservers. To clarify this, the idea of "fast" or "volatile"
subzones must be dealt with. In a volatile environment (where
address or other RR ordering changes rapidly), the refresh rate of a
zone must be set very low, and the TTL of the RRs handed out must
similarly be very low. There is no use in handing out information
with TTLs of an hour, when the conditions for ordering the RRs
changes minutely. There must be a relatively close relationship
between the refresh rates and TTLs of the information. Of course,
with very low refresh rates, zone transfers between the primary and
secondary would have to occur frequently. Given that primary and
secondary nameservers should be topologically and geographically far
apart, moving that much data that frequently is seen as prohibitive.
Also; the longer the propagation time between the primary and
secondary, the larger the window in which circumstances can change -
thus invalidating the secondary's information. It is generally
thought that passing volatile information on to a secondary is fairly
useless - if secondaries want accurate information, then they should
calculate it themselves and not obtain it via zone transfers. This
avoids the problem with secondaries losing contact with the primaries
(but access to the targets of the volatile domain are still
reachable), but the secondary has information that is growing stale.

What is essentially necessary is a secondary (with no primary) which
can calculate the necessary ordering of the RR data for itself (which
also avoids the problem of different versions of domain servers
predictively ordering RR information in different predictive
fashions). For a volatile zone, there is no primary DNS agent, but
rather a series of autonomous secondary agents. Each autonomous
secondary agent is, of course, capable of calculating the ordering or
content of the volatile RRs itself.

5. Implementation

   With some help from Masataka Ohta (Tokyo Institute of Technology), I
   implemented modifications to BIND to permit the specification of the
   zone transfer program (zone transfer agent) for particular domains:

          transfer   <domain-name>   <program-name>

   Currently I define a separate subdomain that has a few hosts defined
   in it - all volatile information. The zone has a refresh rate of
   300, and a minimum TTL of 300 indicated. The configuration file is
   indicated as "volatile.hosts". Every 300 seconds a program "doAxfer"
   is run to do the zone transfer. The program "doAxfer" reads the file
   "volatile.hosts.template" and the file "volatile.hosts.list". The
   addresses specified in volatile.hosts.list are rotated a random
   number of times, and then substituted (in order) into
   volatile.hosts.template to generate the file volatile.hosts. The
   program "doAxfer" then exits with a value of 1 - to indicate to the
   nameserver that the zone transfer was successful, and that the file
   should be read in, and the information distributed. This results in
   a host having multiple addresses, and the addresses are randomized
   every five minutes (300 seconds).

   Two bugs continue to plague us in this endeavor. BIND currently
   considers any TTL under 300 seconds as "irrational", and substitutes
   in the value of 300 instead. This greatly hampers the functionality
   of volatile zones. In the fastest of all cases - a 0 TTL -
   information would be used once, and then thrown away. Presumably the
   new RR information could be calculated every 5 seconds, and the RRs
   handed out with a TTL of 0. It must be considered that one
   limitation of the speed of a zone is going to be the ability of a
   machine to calculate new information fast enough.

   The other bug that also effects this is that, as with TTLs, BIND
   considers any zone refresh rate under 15 minutes to be similarly
   irrational. Obviously zone refresh rates of 15 minutes is
   unacceptable for this sort of applications.

   For a work-around, the current code sets these same hard-coded values
   to 60 seconds. Sixty seconds is still large enough to avoid any
   residual bugs associated with small timer values, but is also short
   enough to allow fast subzones to be of use.

   This version of BIND is currently in release within Rutgers
   University, operating in both "fast" and normal zones.

6. Performance

   While the performance of fast zones isn't exactly stellar, it is not
   much more than the normal CPU loads induced by BIND. Testing was
   performed on a Sun Sparc-2 being used as a normal workstation, but no
   resolvers were using the name server - essentially the nameserver was
   idle. For a configuration with no fast subzones, BIND accrued 11 CPU
   seconds in 24 hours. For a configuration with one fast zone, six
   address records, and being refreshed every 300 seconds (5 minutes),
   BIND accrued 1 minute 4 seconds CPU time. For the same previous
   configuration, but being refreshed every sixty seconds, BIND accrued
   5 minutes and 38 seconds of CPU time.

   As is no great surprise, the CPU load on the serving machine was
   linear to the frequency of the refresh time. The sixty second
   refresh configuration used approximately five times as much CPU time
   as did the 300 second refresh configuration. One can easily
   extrapolate that the overall CPU utilization would be linear to the
   number of zones and the frequency of the refresh period. All of this
   is based on a shell script that always indicated that a zone update
   was necessary, a more intelligent program should realize when the
   reordering of the RRs was unnecessary and avoid such periodic zone
   reloads.

7. Acknowledgments

   Most of the ideas in this document are the results of conversations
   and proposals from many, many people - including, but not limited to,
   Robert Austein, Stuart Vance, Masataka Ohta, Marshall Rose, and the
   members of the IETF DNS Working Group.

8. References

   [1] Mockapetris, P., "Domain Names - Implementation and
       Specification", STD 13, RFC 1035, USC/Information Sciences
       Institute, November 1987.

9. Security Considerations

    Security issues are not discussed in this memo.

10. Author's Address

    Thomas P. Brisco
    Associate Director for Network Operations
    Rutgers University
    Computing Services, Telecommunications Division
    Hill Center for the Mathematical Sciences
    Busch Campus
    Piscataway, New Jersey 08855-0879
    USA

    Phone: +1-908-445-2351
    EMail: brisco@rutgers.edu