Internet Draft   Bernie Volz
                                                        Steve Gonczi
                                                        Process Software

                                                        Ted Lemon
                                                        Internet Engines, Inc.

                                                        Rob Stevens
                                                        Join Systems, Inc.

October 1999   Expires Apr 2000

DHC load balancing algorithm
<draft-ietf-dhc-loadb-00.txt>

Status of this Memo

Copyright Notice

Abstract

This draft proposes a method of algorithmic load balancing that enables multiple, cooperating servers to decide which one should service a client, without requiring participating servers to exchange information.

It proposes a computable server selection mechanism for the DHCP or DHC Failover Protocol.

In addition, it provides for the use of the same mechanism to govern the target server selection of a forwarding agent such as a BOOTP relay.

1. Introduction

This protocol was originally devised to support a specific load balancing optimization of the DHC Failover Protocol [FAILOVR].

The authors later realized that it could be used to optimize the behavior of cooperating DHCP servers and the BOOTP relay agents that forward packets to them.

This proposal makes it possible to set up each participating server to accept a preconfigured (approximate) percentage of the client load. This is done using a deterministic hashing algorithm, and assumes that the hash will produce an even distribution of values based on client load.   Whether the distribution is in fact even for any given set of clients is dependent on the clients, but the hash is expected to produce reasonably evenly distributed output in all cases.

This algorithm could easily be applied to other protocols that have similar characteristics and for which load balancing would be helpful.

2. Terminology

This section discusses both the generic requirements terminology common to many IETF protocol specifications, and also terminology introduced by this document.

2.1. Requirements terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

2.2. Load balancing terminology

This document introduces the following terms:

Hash Bucket Assignments, HBA
   A configuration directive that assigns a set of hash bucket values to a server participating in the load balancing scheme.

Server ID, SID
   An identifier that can be used to designate one of the participating servers in the context of another protocol implementing this proposal. In the context of DHCP, this SHOULD be the IP address or DNS name of the server.

Service Transaction, ST
   A set of client-server exchanges that lead to a server providing or denying some service to a client. Example: the DISCOVER/OFFER/ REQUEST/ACK message exchange between a DHCP server and client is a service transaction. A service transaction may contain one or more messages.

Service Transaction ID, STID
   An attribute of the individual client requests used for load balancing. Deciding which attribute to use is entirely up to the specific implementation.

3. Background and External Requirements

Because DHCP clients use UDP broadcast to contact DHCP servers, a client DHCPDISCOVER message may be received by more than one server. All servers receiving such a broadcast may respond to the client, letting the client choose which server it will use.

When a BOOTP relay agent is used, it typically forwards or rebroadcasts client broadcasts to all configured servers, so a similar inefficiency is present.

The optimization described allows a server to be chosen for each such transaction by performing a "serve" / "do not serve" computation. A forwarding agent can perform the same computation to choose a forwarding destination.

In either case, the choice of server can be computed, without the participants having to negotiate who is to respond.

Each client request MUST have some hashable property that varies from ST to ST, though it is not required that the attribute values should be unique for each ST. The selected attribute MUST have the same value for each message making up a multi-message ST.

The approach is probabilistic in nature, because it is nearly impossible to foresee which client will request service next. For short periods of time, the actual percentage of clients served by a given server will likely deviate from the desired percentage. As the number of requests grows, the actual percentage of the load being handled by each server will approximate the configured percentage.

4. Overview

The specific implementation MUST choose an ST attribute that will be used as the ST "key" for load balancing.

DHCP servers MUST use the Client Identifier option as the STID if it is present.   If no Client Identifier option is present, the hlen field of the DHCP packet should be used as the length of the data to be hashed, and the contents of the chaddr should be the data to be hashed, except that in no event should more than sixteen bytes be used.

Other implementations MAY choose other attribute(s) as their STID.

The proposal maps the chosen STID into a hash value using the
function in section 6. The resulting hash value can then be used
to decide who should respond to the request, or who the forwarding
target should be.

The provided hash function generates hash values 0 to 255,
and yields a fairly even hash bucket distribution for random STIDs, and
also for STID sequences that have some pattern.

Resource allocation is accomplished by assigning specific hash
values to each participating server.

Each server is assigned ownership of one or more hash values,
and will only service requests where the STID hash matches one
of these values.

Any hash buckets not assigned to servers will result
in some client ST-s being entirely ignored. (In some scenarios,
this may be the desired outcome.)   STID-s need not be unique, but
should have sufficient variety to exercise each hash value
assigned to any server.

5. Operation

5.1 Configuration

The configuration step consists of assigning hash values
to available servers. This is accomplished by providing one
or more Hash Bucket Assignments (HBAs). (These may come from
a configuration file, the Windows NT registry, EEPROM, etc.)

Alternatively, HBAs can be transmitted as messages,
encapsulated in messages of another protocol, for example using
e-mail, a DHC Failover Protocol option, or some other mechanism.

5.2 HBA intended for a participant server

When configuring one specific server, an HBA in the form of a
simple bit map of 32 octet values MAY be used.   If the HBA is
represented in this form, the first octet in the HBA bitmap will
represent HBA values 0-7, the next byte values 8-15, and so on,
with the thirty-second octet representing values 248-255.
In each octet, the most significant bit in that octet represents
the largest HBA value represented in that octet.
So for example bit 7 of the first octet represents HBA value 7, and
bit 0 of the first octet represents HBA value 0.

Each bit of the HBA is associated with one possible hash
value. If a bit is set in the map, it means the recipient server
MUST service each client request, where the STID yields the
corresponding hash value.

For example, if a server receives a HBA
with the following 32 octets:

```
                                  buckets
       FF FF FF FF FF FF FF FF  ( 0   - 63 )
       FF FF FF FF FF FF FF FF  ( 64  - 127 )
       00 00 00 00 00 00 00 00  ( 128 - 191 )
       00 00 00 00 00 00 00 00  ( 192 - 255 )
```

then it MUST service any client requests where the STID
hashes into the bucket values of 0 through 127.

The above example is merely for illustration
purposes. An application implementing this algorithm is free
to choose a different format, as long as the server is informed
in some way of hash bucket values it owns.

5.3 HBA intended for a forwarder

When configuring a forwarding agent, (e.g.: BOOTP relay)
HBAs consisting of pairs of Server-ID / Hash Bucket values
MAY be used.

Here, the Server ID (SID) designates the server responsible for
the specified Hash Bucket. The forwarding agent
forwards each client request, where the STID yields the
specified hash value, to the server designated by the SID.

The Server ID may be any unique server attribute,
(E.g.: IP address, DNS name, etc) that is meaningful in the context of
the relay agent operation.

A forwarder may be configured to forward a packet to
more than one server. For example, a BOOTP relay could be
set up to split the load between 2 primary-backup server pairs,
running the DHC Failover Protocol [FAILOVR].

A possible configuration file for a forwarding agent
(e.g.: BOOTP relay) may look like this:

```
192.33.43.11  0   .. 24;
192.33.43.12  25 .. 55;
192.33.43.13  56 ..128;
192.33.43.14  129..255;
192.33.43.15  129..255;
```

The above configuration consists of 5 HBAs. The first HBA states:
"Any Client request, where the STID yields a hash value
0 to 24, will be forwarded to server 192.33.43.11".
Note that that HBA #4 and #5 instruct to forward the same requests
to both 192.33.43.142 and 192.33.43.14.

The above example is merely for illustration purposes. An implementing application is free to choose a different format, as long as the forwarding agent is given a list of SIDs, with the set of hash bucket values each server owns.

6. Hash function for load balancing

The following hash function is a c language implementation of the algorithm known as "Pearson's hash". The Pearson's hash algorithm was originally published in [PEARSON] To make this proposal work, all interoperable implementations MUST use the same hash function.

```
/* A "mixing table" of 256 distinct values, in pseudo-random order. */
    unsigned char loadb_mx_tbl[256] =
    {
    251, 175, 119, 215,  81,  14,  79, 191, 103,  49,
    181, 143, 186, 157,   0, 232,  31,  32,  55,  60,
    152,  58,  17, 237, 174,  70, 160, 144, 220,  90,
    57,  223,  59,   3,  18, 140, 111, 166, 203, 196,
    134, 243, 124,  95, 222, 179, 197,  65, 180,  48,
     36,  15, 107,  46, 233, 130, 165,  30, 123, 161,
    209,  23,  97,  16,  40,  91, 219,  61, 100,  10,
    210, 109, 250, 127,  22, 138,  29, 108, 244,  67,
    207,   9, 178, 204,  74,  98, 126, 249, 167, 116,
    34,   77, 193, 200, 121,   5,  20, 113,  71,  35,
    128,  13, 182,  94,  25, 226, 227, 199,  75,  27,
     41, 245, 230, 224,  43, 225, 177,  26, 155, 150,
    212, 142, 218, 115, 241,  73,  88, 105,  39, 114,
     62, 255, 192, 201, 145, 214, 168, 158, 221, 148,
    154, 122,  12,  84,  82, 163,  44, 139, 228, 236,
    205, 242, 217,  11, 187, 146, 159,  64,  86, 239,
    195,  42, 106, 198, 118, 112, 184, 172,  87,   2,
    173, 117, 176, 229, 247, 253, 137, 185,  99, 164,
    102, 147,  45,  66, 231,  52, 141, 211, 194, 206,
    246, 238,  56, 110,  78, 248,  63, 240, 189,  93,
     92,  51,  53, 183,  19, 171,  72,  50,  33, 104,
    101,  69,   8, 252,  83, 120,  76, 135,  85,  54,
    202, 125, 188, 213,  96, 235, 136, 208, 162, 129,
    190, 132, 156,  38,  47,   1,   7, 254,  24,   4,
    216, 131,  89,  21,  28, 133,  37, 153, 149,  80,
    170,  68,   6, 169, 234, 151
    };

unsigned char loadb_p_hash(unsigned char *key,/* The key to be hashed */
                           int len)           /* Key length in bytes  */
    {
        unsigned char hash  = len;
        int i;

        for( i=len ; i > 0 ;  )
            hash = loadb_mx_tbl  [ hash ^ key[ --i ] ];

        return( hash );
    }
```

7. Security

This proposal in and by itself provides no security, nor does
it impact existing security.

Servers using this algorithm are responsible for ensuring that if
the contents of the HBA are transmitted over the network as part of
the process of configuring any server, that message be secured
against tampering, since tampering with the HBA could result in
denial of service for some or all clients.

8. References

[FAILOVR] Droms, R., Rabil, G., Dooley, M., Kapur, A., Gonczi, S.,
Volz, B., "DHCP Failover Protocol", Internet Draft
<draft-ietf-dhc-failover-05.txt>, June 1999.

[PEARSON] The Communications of the ACM Vol.33, No. 6 (June 1990),
pp. 677-680.

[RFC2131] R. Droms, "Dynamic Host Configuration Protocol", RFC2131,
March 1997.

[RFC2219] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels," RFC-2219, March 1997.

9. Acknowledgements

Special thanks to Peter K. Pearson, the author of Pearson's hash
who has kindly granted his permission to use this algorithm,
free of any encumbrances.

This proposal stems from the original idea of hashing MAC addresses
to a single bit by Ted Lemon, during a Failover Protocol discussion
held at CISCO Systems in February, 1999.

Rob Stevens suggested the potential use of this algorithm for
purposes beyond the Failover Protocol.

Many thanks to Ralph Droms, Kim Kinnear, Mark Stapp, Glenn Waters,
Greg Rabil and Jack Wong for their comments during the ongoing discussions.

10. Full Copyright Statement

11. Author's information

Bernie Volz
Steve Gonczi
Process Software Corporation
959 Concord St.
Framingham, MA 01701
Phone: (508) 879-6994
EMail: volz@process.com
       gonczi@process.com

Ted Lemon
Internet Engines, Inc.
950 Charter Street
Redwood City, CA 94063
Phone: (650) 779 6031
EMail: mellon@isc.org

Rob Stevens
Join Systems, Inc.
1032 Elwell Ct Ste 243
Palo Alto CA 94203
Phone: (650)-968-4470
EMail: robs@join.com