Mapping Names, Locations and IP Addresses of Embedded Components in the EVLA WIDAR Correlator

NRC-EVLA Memo# 030

Kevin Ryan

National Radio Astronomy Observatory

May 19, 2007

ABSTRACT

This memo addresses one of the 'to-do' items of the face-to-face meeting held in Penticton, BC in April, 2007 concerning how embedded components of the EVLA WIDAR correlator will be addressed over the IP/Ethernet network.

Introduction

Baseline and Station Boards are configured and monitored over the network by addressing them based on their location in the system. Problems with assigning static IP addresses based on location are discussed and a solution is proposed.

Correlator boards, CMIBs, PCMC boards and Delay Module boards must also be accessible via their unique identifier, such as serial number, for maintenance and logging purposes. A method to dynamically map each board's ID to its current IP address is also proposed.

Two appendices are included: 1) naming suggestions and 2) control of DNS using UDP and RFC-2136.

James Robnett and George Martin each helped quite a bit to explain various network issues. It is desired that they and the EVLA policy makers and any interested others read this memo critically to expose remaining 'holes' so that the naming/mapping infrastructure can be in place and debugged before the correlator arrives in Socorro.

Mapping IP Addresses to Board Locations

Correlator boards reside in racks containing two crates that hold eight boards per crate. Each slot has a unique identifier based on its position in the crate, the crate's position in

the rack and the rack's position in the system. This identifier is known as the rack/crate/slot number.

Station Boards and Baseline Boards are configured and monitored over the network via their attached embedded processor (CMIB) and are referenced by their rack/crate/slot position.

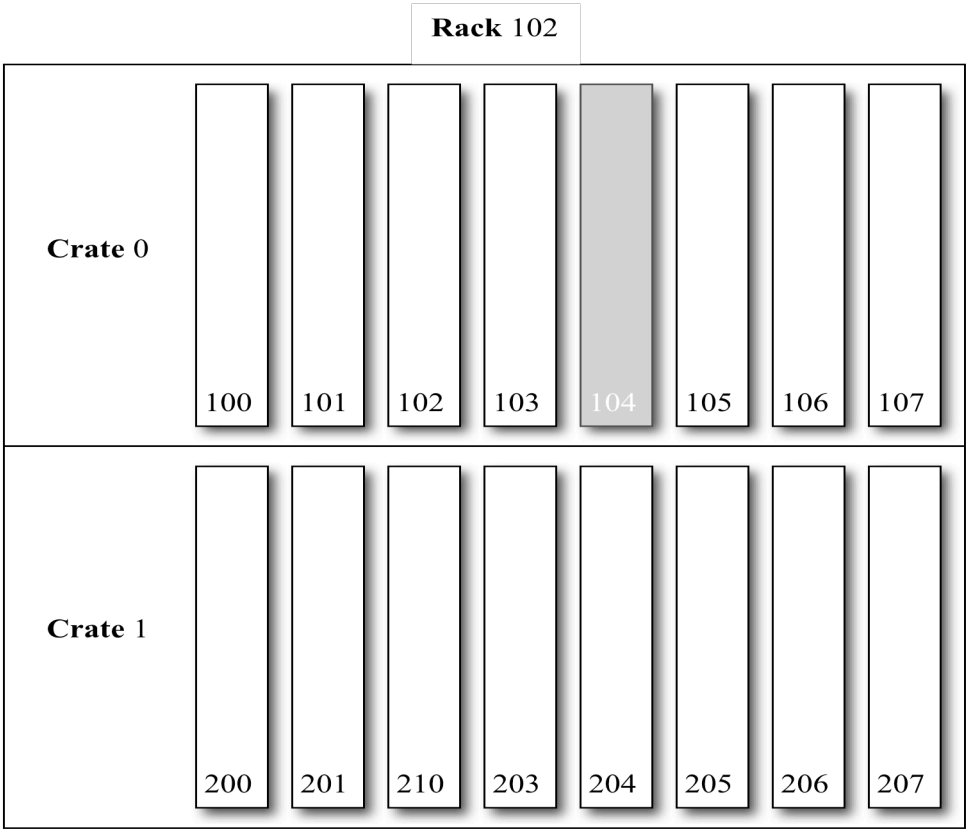

Figure 1 Rack / Crate / Slot Numbering

The Interface Control Document (ICD), "Evlar Correlator System Numbering Plan", [1] Specifies a mapping between a board's rack/crate/slot position and its IP address. The board in the slot highlighted in the figure above would have an IP address of **`10.81.102.104`**. The first two octets are assigned by NRAO system administrators and will be constant. Keeping uniform with the EVLA antenna-mapping plan, the third octet is the rack number (akin to the antenna number). The forth octet is the crate/slot id. Each rack has its own Class C address space. This is simple and intuitive; a board can be directly accessed over the network by simply knowing its location.

Problems Associated with Assigning 'Pseudo Static' IP's

IP addresses by location imply static IP addresses but they are static to the slot, not to the processor in the slot; the CMIB must be able to adapt to the slot's IP address. Some methods that were discussed and their associated problems are lightly mentioned here to document the path to the proposed solution.

Assign IP's to ports in the network switch. Network switches can have each of their ports programmed to a specific IP address; each port would then be physically connected to the correct crate/slot. This method makes the system fragile and prone to human error.

CMIB assigns IP address at boot time. The rack/crate/slot position is hardwired into the slot's backplane connector and can be read by the CMIB and subsequently used to build the IP address associated with that slot. But because the CMIB is diskless, it must boot from over the network and to do that it must have an IP address - so this catch-22 method will not work.

Dynamic DNS. Dynamic DNS allows a process to perform host name to IP address mapping in DNS dynamically. It was created to allow DNS to work with the dynamic IP addresses generated by DHCP. A plausible method to making the CMIB addressable based on its location is by letting DHCP assign it a dynamic IP address at boot time and then the CMIB instructs the DNS server to map that IP address to its location *name*.

The location name would be something like "`102-0-4`" and it would be used instead of a numerical IP address. The problem with this scheme was brought up by James Robnett and Rich Moeser -- it takes time to query DNS. It was accidentally discovered with the EVLA antennas that continual 10 ms accesses to DNS overwhelmed the service [3]. The normal bursts of chatter on the network associated with monitor and control of WIDAR can be expected to push the speeds that broke DNS on EVLA. It would therefore be necessary for a controlling process to cache IP addresses to reduce DNS look-ups. This in itself creates a set of problems deemed worrisome enough to drop the Dynamic DNS idea altogether (for this purpose - it will be used later).

For the continual monitor/control traffic it is desirable to use an IP address rather than DNS.

Proposed Method of Assigning IP Addresses by Location

The proposed solution is to let DHCP assign a dynamic IP address to the CMIB at boot time. The CMIB will finish the network booting using this IP and then create a second IP address based on the rack/crate/slot location that it read from the slot's backplane connector. This second IP will be applied to the CMIB's network interface as an alias IP address that the controlling processes will use for subsequent network access.

This method is the same used by Mike Revnell with the FORM CMIBs and proposed by Bruce Rowen at the face-to-face meetings in Penticton and Socorro in April. The reason it was not adopted from the onset is that Revnell's CMIB's would sometimes assign the original IP address to network traffic originating from it and this caused confusion. Rowen pointed out at the meetings that the `sys/socket.h` library's `bind()` function can be used to assign the second (location-based IP address) to the socket. The Java language also has a `bind()` method. So, as long as we are conscientious about how the packets are created, we should not encounter the FORM CMIB's problem.

Will Tomcat work with an aliased IP address? The primary interface to the CMIB over the network is via HTTP and Apache's Tomcat web server is used to provide this interface. It was tested on a CMIB with an aliased IP address and worked without any reconfiguration necessary; it recognized and handled both IP addresses properly.

Because Tomcat is used almost extensively for CMIB communication, the problems associated with the FORM CMIBs should be reduced.

Though this method doubles the number IP addresses required per rack (from 16 to 32), it is considered tolerable since each rack is assigned a whole Class-C address space (256 addresses). The benefits are:

- DNS does not have to be used for the continual monitor/control traffic,
- Board IP addresses can be resolved easily by both humans and programs.

Mapping Board Serial Numbers to IP Addresses

Each correlator Baseline Board and Station Board has attached to it a CMIB, and a PC Mezzanine Card (PCMC) that interfaces the CMIB to it; additionally, each Station Board has an attached Delay Module.

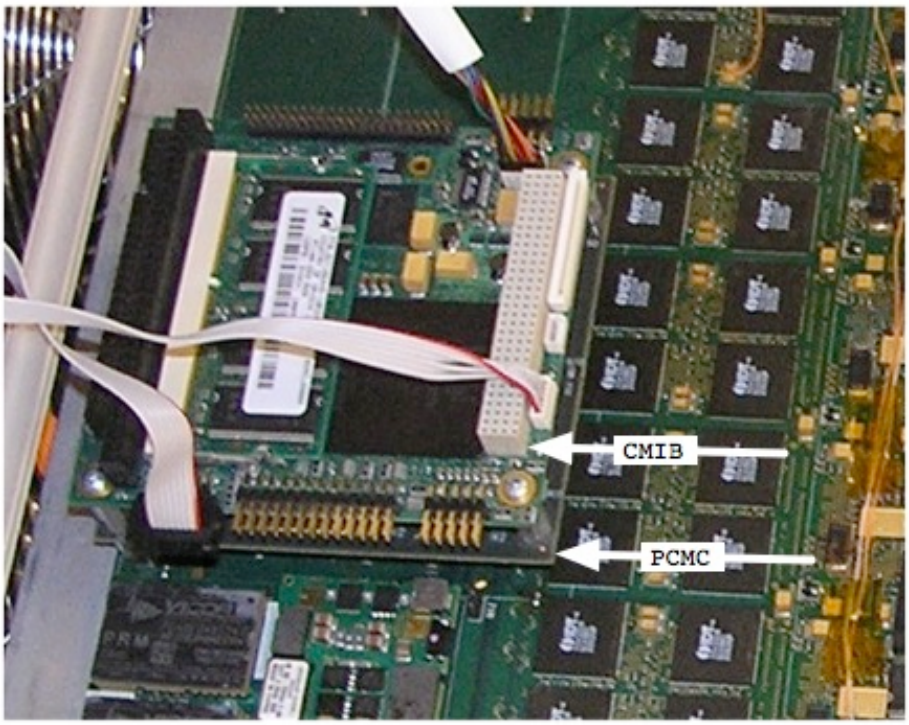

Figure 2 CMIB attaches to PCMC which attaches to Baseline Board

The Station and Baseline boards, and their attached mezzanine boards, are not permanently affixed to each other or to any particular rack/crate/slot; they can move around the system independently.

Each board, including the mezzanine boards and the CMIBs, has a unique identifier associated with it. For the Station/Baseline Boards, PCMC's and the Delay Modules it is a serial number as specified in [1]; for the CMIB's it is their MAC address. For maintenance purposes, it is desirable to be able to address these components on the network by their unique ID; this implies a mapping between the ID and the attached CMIB's IP address. Additionally, [1] specifies virtual IDs for things such as basebands

and station quads; we may possibly want a mapping of these virtual entities to their physical location on the network as well.

Initial thinking at the face-to-face meeting in Penticton in April, 2007, was to have the CMIB create and assign to itself another IP address for each of these boards based on their serial numbers. This means even more static IP addresses assigned to a single CMIB; and it soon became obvious that this scheme would not work in a Class C address space. (There will be 128 Baseline Boards + 128 Station Boards + 256 PCMCs + 128 Delay Modules = 640 required IP addresses.)

What is needed is a database that holds the mappings between component IDs and their CMIB's IP address. Furthermore it is desired that this database is automatically kept current with the system's state – without a need for human intervention.

Dynamic DNS

As briefly mentioned earlier, Dynamic DNS [2] was created to address the problems associated with mapping dynamic IP addresses to static host names. It can be used to maintain the mapping database for the correlator. When the CMIB boots, it reads the serial numbers of the attached boards, creates ID strings from them and instructs DNS to associate the ID's to its IP address.

No external database or human intervention is required; the boards can be mixed and matched and moved around the system and the ID to IP mapping will be dynamically maintained by DNS.

George Martin set up a temporary DNS service that was used to test that the CMIB could indeed perform control of Dynamic DNS. Dynamic DNS can be done through a Unix utility called 'nsupdate' or by sending the necessary information (as specified in [2]) via UDP packets to the DNS server. The appendix at the end of this memo gives the details of the UDP packet method for implementation into the CMIB startup software.

Tests were run to show that the CMIB could communicate with the test DNS service via UDP to add IP addresses and map hostnames to them. The current CMIB OS does not include the `BIND` libraries (Berkley Internet Name Domain – not to be confused with the `socket.h bind()` function) that contain the `nsupdate` utility; we may want to consider adding them and using nsupdate as that will be much simpler and much more robust than 'rolling our own' via raw UDP packets.

Using Dynamic DNS, the idea is to map a CMIB's location-based IP address to the various hostname strings associated with it. A user, whether it be a program or a human can then access a WIDAR component by these names or by the main IP address itself.

The following three screenshots show how the same CMIB is addressed using three different URLs:

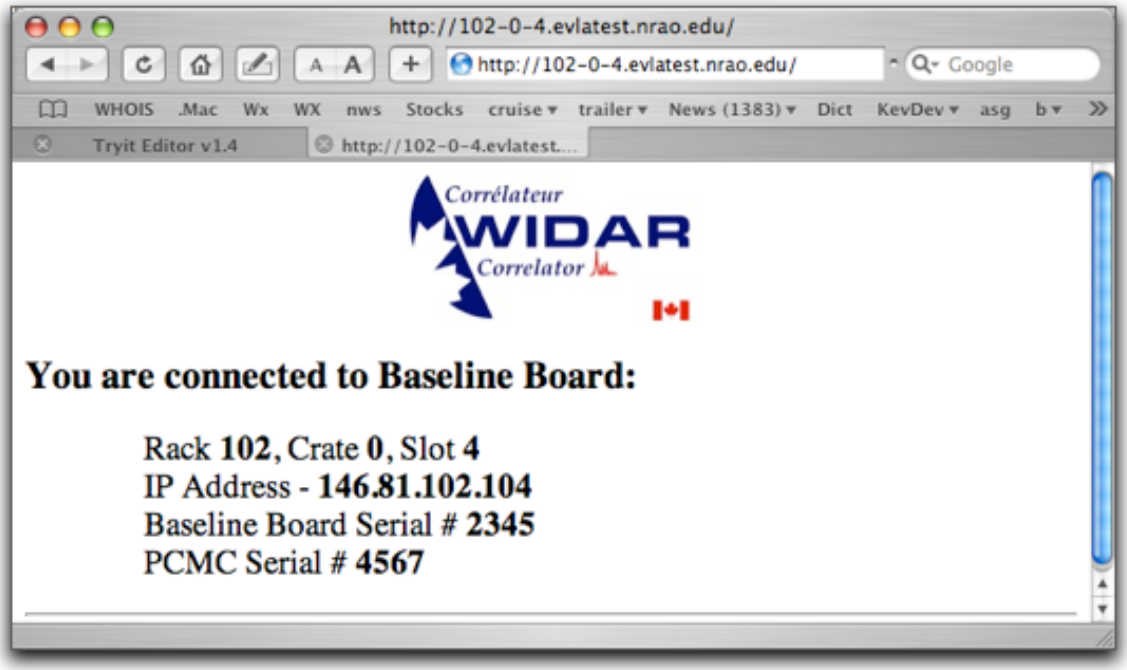

Figure 3 Addressed by Rack/Crate/Slot location-name (see URL)

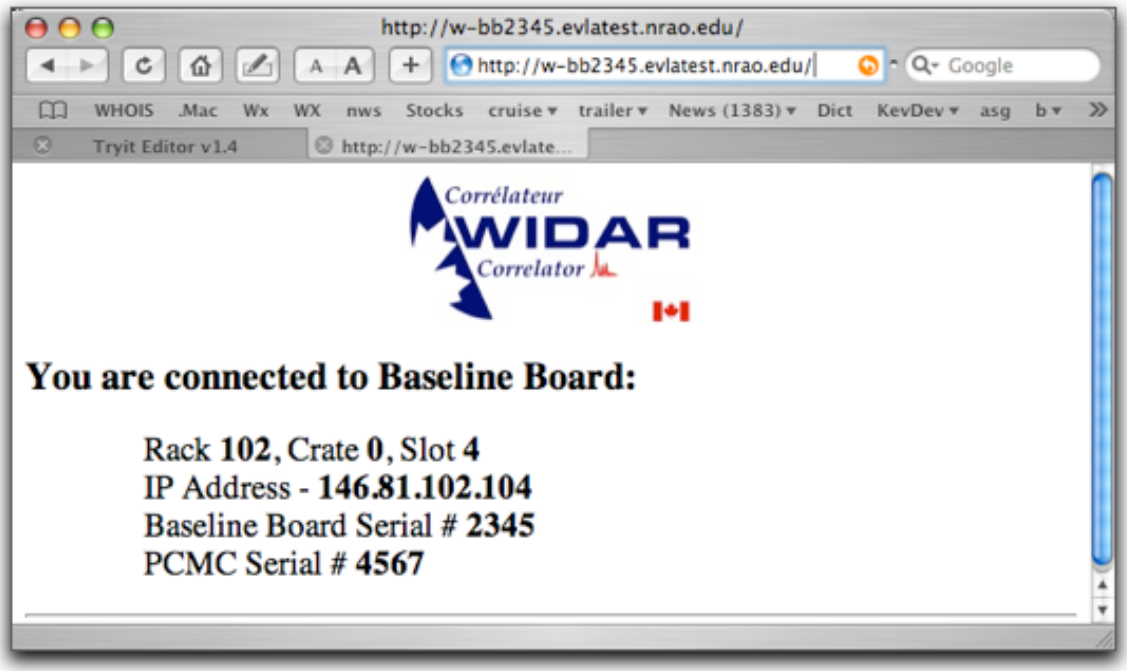

Figure 4 Addressed by board serial number

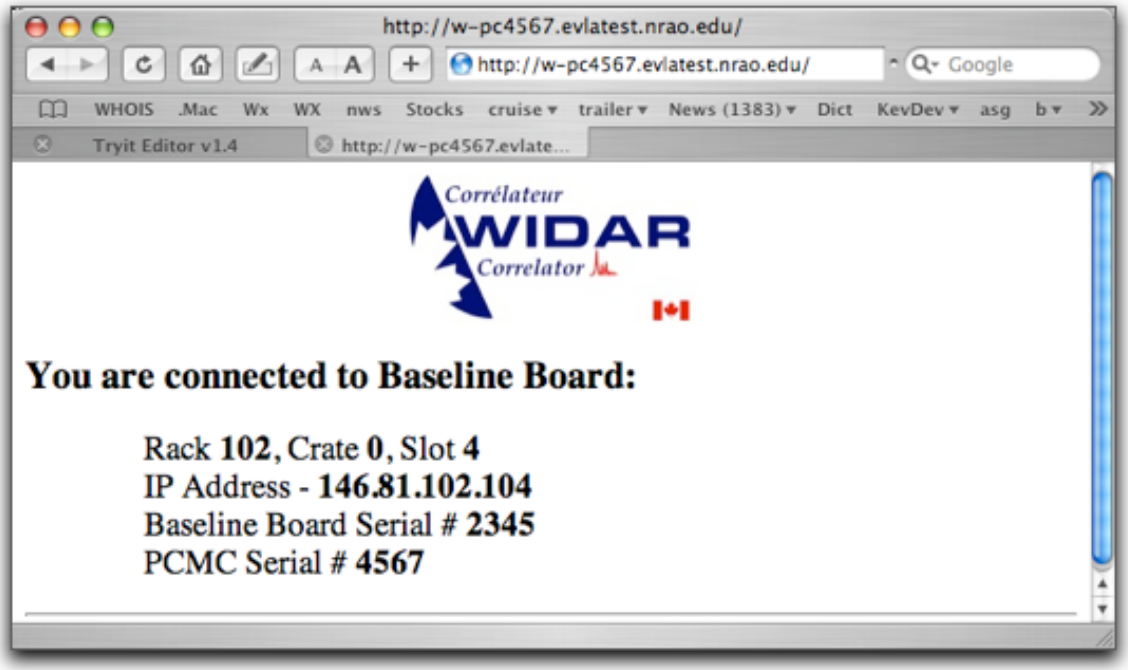

Figure 5 Addressed by PCMC serial number

Implementation

This section uses nsupdate to perform actions on the DNS server; an appendix later on shows how to translate these command-line calls directly into UDP packets that can be sent from the CMIB software.

For the purpose of discussion, a DNS *'A' record* is the fundamental hostname of a device and is associated directly to the device's IP address. zia.aoc.nrao.edu is an 'A' record entry in DNS associated with the IP address 146.88.1.4. A DNS CNAME record is an alias name that associates to an 'A' record.

At startup the CMIB will:

- Determine its rack/crate/slot id from hardware.
- Create an IP address based on it (`146.81.102.104);` this will be the IP address used for all subsequent communications with the CMIB (as opposed to the one given it by DHCP).
- Create a string based on the location (`'102-0-4'`)this will be the *location-name*.
- Create a DNS 'A' record associating the location-name to the location IP address. This becomes the 'main' hostname to which all others will be aliased.
- Create alias hostnames (CNAMEs) for the remaining attached Station/Baseline Board, PCMC, Delay Module and any other names that might want to be used to reference the CMIB and it's attached components.

Notes for the following eamples:

1. **The IP address (146.88.2.43) is that of the CMIB on a table in our office and not the real-world one of 146.81.102.104 used in previous examples,**
2. **the number '9' is the Time To Live (TTL) in seconds of a mapping for caching purposes.**

Example using `nsupdate`:

```
epoch:~/work/widar kryan$ nsupdate
> update add 102-0-4.evlatest.nrao.edu 9 A 146.88.2.43
> update add w-bb2345.evlatest.nrao.edu 9 CNAME 102-0-4.evlatest.nrao.edu
> update add w-pc4567.evlatest.nrao.edu 9 CNAME 102-0-4.evlatest.nrao.edu
> send

> quit
epoch:~/work/widar kryan$
```

We can verify the mapping with `dig`:

```
epoch:~/work/widar/doc kryan$ dig @zia evlatest.nrao.edu -t AXFR

; <<>> DiG 9.3.2 <<>> @zia evlatest.nrao.edu -t AXFR
; (1 server found)
;; global options:  printcmd
evlatest.nrao.edu.      86400   IN      SOA     zia.aoc.nrao.edu.
tech.nrao.edu. 65 10800 3600 3600000 3600
evlatest.nrao.edu.      86400   IN      NS      zia.aoc.nrao.edu.
evlatest.nrao.edu.      86400   IN      MX      10 revere.aoc.nrao.edu.
102-0-4.evlatest.nrao.edu. 9    IN      A       146.88.2.43
w-bb2345.evlatest.nrao.edu. 9   IN      CNAME   102-0-4.evlatest.nrao.edu.
w-pc4567.evlatest.nrao.edu. 9   IN      CNAME   102-0-4.evlatest.nrao.edu.
evlatest.nrao.edu.      86400   IN      SOA     zia.aoc.nrao.edu.
tech.nrao.edu. 65 10800 3600 3600000 3600
;; Query time: 36 msec
;; SERVER: 146.88.1.4#53(146.88.1.4)
;; WHEN: Fri Jun 15 09:29:42 2007
;; XFR size: 7 records (messages 1)

epoch:~/work/widar/doc kryan$
```

We also use `ping` to show the association of the names to the IP address.

```
epoch:~/work/widar/doc kryan$ ping 102-0-4.evlatest.nrao.edu
PING 102-0-4.evlatest.nrao.edu (146.88.2.43): 56 data bytes
64 bytes from 146.88.2.43: icmp_seq=0 ttl=64 time=0.373 ms

epoch:~/work/widar/doc kryan$ ping w-bb2345.evlatest.nrao.edu
PING 102-0-4.evlatest.nrao.edu (146.88.2.43): 56 data bytes
64 bytes from 146.88.2.43: icmp_seq=0 ttl=64 time=0.930 ms

epoch:~/work/widar/doc kryan$ ping w-pc4567.evlatest.nrao.edu
PING 102-0-4.evlatest.nrao.edu (146.88.2.43): 56 data bytes
64 bytes from 146.88.2.43: icmp_seq=0 ttl=64 time=0.357 ms
```

When we move the Baseline Board in the example to a different slot (hence different IP address) we can verify that it is remapped properly:

```
> update add 102-0-5.evlatest.nrao.edu 9 A 146.88.2.44
> update add w-bb2345.evlatest.nrao.edu 9 CNAME 102-0-5.evlatest.nrao.edu
> send
```

It now shows bb2345 attached to 006-0-5:

```
epoch:~/work/widar/doc kryan$ dig @zia evlatest.nrao.edu -t AXFR

; <<>> DiG 9.3.2 <<>> @zia evlatest.nrao.edu -t AXFR
; (1 server found)
;; global options:  printcmd
evlatest.nrao.edu.      86400   IN      SOA     zia.aoc.nrao.edu.
tech.nrao.edu. 66 10800 3600 3600000 3600
evlatest.nrao.edu.      86400   IN      NS      zia.aoc.nrao.edu.
evlatest.nrao.edu.      86400   IN      MX      10 revere.aoc.nrao.edu.
102-0-4.evlatest.nrao.edu. 9    IN      A       146.88.2.43
102-0-5.evlatest.nrao.edu. 9    IN      A       146.88.2.44
w-bb2345.evlatest.nrao.edu. 9   IN      CNAME   102-0-5.evlatest.nrao.edu.
w-pc4567.evlatest.nrao.edu. 9   IN      CNAME   102-0-4.evlatest.nrao.edu.
evlatest.nrao.edu.      86400   IN      SOA     zia.aoc.nrao.edu.
tech.nrao.edu. 66 10800 3600 3600000 3600
;; Query time: 10 msec
;; SERVER: 146.88.1.4#53(146.88.1.4)
;; WHEN: Fri Jun 15 09:47:46 2007
;; XFR size: 8 records (messages 1)

epoch:~/work/widar/doc kryan$
```

Caveat

When boards are moved around the system their hostnames are properly mapped to the new IP addresses as illustrated above; a problem arises when a board is removed from the system altogether though. The board's old name reference will remain in DNS and an unsuspecting person attempting communication with that board will actually be connected to a different board (the one currently at the old board's location). This is because both board names will be associated with that location IP address. When the old board is placed back into the system, it will once again become properly mapped, but while it sits in storage, it will not.

George Martin came up with a possible solution that involves writing a DNS TXT type record that lists the devices associated with the IP address. The first thing the CMIB would do is to delete those devices from DNS before assigning the current ones.

```
> update add 102-0-4.evlatest.nrao.edu 9 TXT "w-bb2345, w-pc4567"
> send
```

Appears in DNS as:

```
;; global options:  printcmd
evlatest.nrao.edu.      86400   IN      SOA     zia.aoc.nrao.edu.
tech.nrao.edu. 67 10800 3600 3600000 3600
evlatest.nrao.edu.      86400   IN      NS      zia.aoc.nrao.edu.
evlatest.nrao.edu.      86400   IN      MX      10 revere.aoc.nrao.edu.
102-0-4.evlatest.nrao.edu. 9    IN      TXT     "w-bb2345, w-pc4567"
102-0-4.evlatest.nrao.edu. 9    IN      A       146.88.2.43
102-0-5.evlatest.nrao.edu. 9    IN      A       146.88.2.44
w-bb2345.evlatest.nrao.edu. 9   IN      CNAME   102-0-5.evlatest.nrao.edu.
w-pc4567.evlatest.nrao.edu. 9   IN      CNAME   102-0-4.evlatest.nrao.edu.
evlatest.nrao.edu.      86400   IN      SOA     zia.aoc.nrao.edu.
tech.nrao.edu. 67 10800 3600 3600000 3600
```

The problem is that this would require the CMIB to interact (as opposed to sending simple one-way commands) with DNS to query and obtain the string of devices previously associated with that 'A' record. This interaction could easily get quite complicated – especially for low-level software.

It is beyond the scope of this memo to provide an implementation that fixes this problem; if it is indeed deemed necessary to address it, engineering time will have to be allocated for the purpose.

Another option worth researching is that there appears to be several utility programs and scripts that manipulate nsupdate that may possibly do what we want without having to develop our own. A site that has many links to such software is at: http://www.dns.net/dnsrd/tools.html.

Questions

These are not questions to test the reader, but those that the author would like to learn answers to.

1. Can an IP address be discovered from a device's MAC address using off-the-shelf networking tools? If so, than we would not need a hostname associated with a MAC address.
2. Is there an easier way to delete all CNAMEs associated with an 'A' record in DNS so that the CMIB can remove stale associations before mapping its current components?

Appendix A - Suggested ID naming conventions

Suggestions as to how location, serial numbers and MAC addresses can be used as hostnames are presented here. These are only suggestions; Brent Carlson (or whoever updates the Interface document [1]) will specify the format formally within that document.

Rack/Crate/Slot

The Interface Control Document [1], specifies this as RID-CID-SLOTID so, for our example, the location hostname would be `'006-0-4'`.

Serial Numbers

[1] Specifies the serial numbers of the various WIDAR components as follows:

Module	Serial Number Range
Station Board	0000 – 1FFF
Baseline Board	2000 – 3FFF
PCMC	4000 – 5FFF
Fanout Board	6000 – 7FFF
Delay Module	8000 – 9FFF
Phasing Board	A000 – BFFF
TGB	C000 – DFFF
Reserved	E000 – EFFF

Use the numbers as-is. The number would just be used as a string like `'0245'`. This would be quite meaningless though to someone not knowing about it.

Add a 'w'. One suggestion is to simply add a 'w' to the front of any serial number or MAC address signifying a WIDAR component. `'w0123'` would be the unique network hostname of a Station Board whose serial number is 0123. People seeing the 'w' would at least link it to WIDAR.

Add 'w-xx'. The first two suggestions work because the serial numbers do not overlap the board types, but identifying a board type from its unique ID name, would require memorizing the table above or referring to a document. Adding the board's initials could alleviate this: `'w-bb2345', or 'wbb2345'` for a Baseline Board whose serial number is 2345. This provides even more meaning to the name without too much more overhead.

MAC Addresses

CMIB's use their MAC address as their unique identifier (i.e. 00:11:22:33:44:55). It was suggested that we only use the last three octets as the first three will be constant for each

vendor but it cannot be assumed that future CMIBs will come from the same vendor so, for safety sake, we should probably use all six octets. So, as painful at it may be, one suggestion is `'cmib001122334455'`. (This should rarely have to be used, so it may be tolerable).

Other Names

The 'virtual' ID's and other names by which we may want to address components of the system can be added at anytime to the CMIB's startup processing.

It should be kept in mind that normal operation of the system will use the location-based IP addresses directly to minimize DNS access by the continual monitor/control chatter; these names are meant more for special purpose access to the embedded components.

Appendix B – Dynamic DNS Control via UDP

RFC-2136 [2] is the specification for formatting Dynamic DNS requests. Software that runs on the CMIB was created to test control of DNS. During this development, a packet sniffer was used to alleviate endian issues and other formatting anomalies. The following is a client (146.88.2.129)/server (145.88.1.4) exchange for the nsupdate operation shown:

```
epoch:~/work/widar kryan$ nsupdate
> update add 006-0-4.evlatest.nrao.edu 9 A 146.88.2.43
> update add w-bb2345.evlatest.nrao.edu 9 CNAME 006-0-4.evlatest.nrao.edu
> update add w-pc4567.evlatest.nrao.edu 9 CNAME 006-0-4.evlatest.nrao.edu
> send
> quit
epoch:~/work/widar kryan$
```

```
UDP packet from 146.88.2.129:57835 to 146.88.1.4:dns(53) (43 bytes)
1E D7 01 00  00 01 00 00  00 00 00 00  07 30 30 36  .............006
2D 30 2D 34  08 65 76 6C  61 74 65 73  74 04 6E 72  -0-4.evlatest.nr
61 6F 03 65  64 75 00 00  06 00 01                ao.edu.....

UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:57835 (92 bytes)
1E D7 85 83  00 01 00 00  00 01 00 00  07 30 30 36  .............006
2D 30 2D 34  08 65 76 6C  61 74 65 73  74 04 6E 72  -0-4.evlatest.nr
61 6F 03 65  64 75 00 00  06 00 01 C0  14 00 06 00  ao.edu..........
01 00 00 00  00 00 25 03  7A 69 61 03  61 6F 63 C0  ......%.zia.aoc.
1D 04 74 65  63 68 C0 1D  00 00 00 3A  00 00 2A 30  ..tech.....:..*0
00 00 0E 10  00 36 EE 80  00 00 0E 10              .....6......

UDP packet from 146.88.2.129:57835 to 146.88.1.4:dns(53) (105 bytes)
53 90 28 00  00 01 00 00  00 03 00 00  08 65 76 6C  S.(..........evl
61 74 65 73  74 04 6E 72  61 6F 03 65  64 75 00 00  atest.nrao.edu..
06 00 01 07  30 30 36 2D  30 2D 34 C0  0C 00 01 00  ....006-0-4.....
01 00 00 00  09 00 04 92  58 02 2B 08  77 2D 62 62  ........X.+.w-bb
32 33 34 35  C0 0C 00 05  00 01 00 00  00 09 00 02  2345............
C0 23 08 77  2D 70 63 34  35 36 37 C0  0C 00 05 00  .#.w-pc4567.....
01 00 00 00  09 00 02 C0  23                       ........#

UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:57835 (12 bytes)
53 90 A8 80  00 00 00 00  00 00 00 00              S...........
```

The following Java code was used to communicate with the DNS server running on zia.aoc.nrao.edu. It shows various examples using nsupdate, a packet dump, and the corresponding UDP packet sent from the program.

```
import java.net.*;
import java.io.*;

public class DnsTest {
    public DnsTest() {
       try {
           DatagramSocket s = new DatagramSocket();
           InetAddress ip = InetAddress.getByName("zia.aoc.nrao.edu");

           /*
            * > update add cmib1.evlatest.nrao.edu 20 A 146.88.2.43
            *
            * UDP packet from 146.88.2.129:53679 to 146.88.1.4:dns(53)
            * A1 E3 01 00 00 01 00 00 00 00 00 00 05 63 6D 69  .............cmi
            * 62 31 08 65 76 6C 61 74 65 73 74 04 6E 72 61 6F  b1.evlatest.nrao
            * 03 65 64 75 00 00 06 00 01                       .edu.....
            *
            * UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:53679
            * A1 E3 85 80 00 01 00 00 00 01 00 00 05 63 6D 69  .............cmi
            * 62 31 08 65 76 6C 61 74 65 73 74 04 6E 72 61 6F  b1.evlatest.nrao
            * 03 65 64 75 00 00 06 00 01 C0 12 00 06 00 01 00  .edu............
            * 00 0E 10 00 25 03 7A 69 61 03 61 6F 63 C0 1B 04  ....%.zia.aoc...
            * 74 65 63 68 C0 1B 00 00 00 1B 00 00 2A 30 00 00  tech........*0..
            * 0E 10 00 36 EE 80 00 00 0E 10                    ...6......
            *
            * UDP packet from 146.88.2.129:53679 to 146.88.1.4:dns(53)
            * C3 48 28 00 00 01 00 00 00 01 00 00 08 65 76 6C  .H(..........evl
            * 61 74 65 73 74 04 6E 72 61 6F 03 65 64 75 00 00  atest.nrao.edu..
            * 06 00 01 05 63 6D 69 62 31 C0 0C 00 01 00 01 00  ....cmib1.......
            * 00 00 14 00 04 92 58 02 2b                       ......X.+
            *
            * UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:53679
            * C3 48 A8 85 00 00 00 00 00 00 00 00              .H..........
            */
           byte[] m1a = {0x1a, 0x1a, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00,
                         0x00, 0x00, 0x00, 0x00, 0x05, 0x63, 0x6D, 0x69,
                         0x62, 0x31, 0x08, 0x65, 0x76, 0x6C, 0x61, 0x74,
                         0x65, 0x73, 0x74, 0x04, 0x6E, 0x72, 0x61, 0x6F,
                         0x03, 0x65, 0x64, 0x75, 0x00, 0x00, 0x06, 0x00,
                         0x01};
           byte[] m1b = {0x1b, 0x1b, 0x28, 0x00, 0x00, 0x01, 0x00, 0x00,
                         0x00, 0x01, 0x00, 0x00, 0x08, 0x65, 0x76, 0x6C,
                         0x61, 0x74, 0x65, 0x73, 0x74, 0x04, 0x6E, 0x72,
                         0x61, 0x6F, 0x03, 0x65, 0x64, 0x75, 0x00, 0x00,
                         0x06, 0x00, 0x01, 0x05, 0x63, 0x6D, 0x69, 0x62,
                         0x31, -64, 0x0C, 0x00, 0x01, 0x00, 0x01, 0x00,
                         0x00, 0x00, 0x14, 0x00, 0x04, -110, 0x58, 0x02,
                         0x2b};
           DatagramPacket dp1a = new DatagramPacket(m1a, m1a.length, ip, 53);
           DatagramPacket dp1b = new DatagramPacket(m1b, m1b.length, ip, 53);

           /*
            * > update add 102-0-5.evlatest.nrao.edu 9 CNAME
            *              cmib1.evlatest.nrao.edu
            *
            * UDP packet from 146.88.2.129:53675 to 146.88.1.4:dns(53)
            * 12 F0 01 00 00 01 00 00 00 00 00 00 07 31 30 32  .............102
            * 2D 30 2D 35 08 65 76 6C 61 74 65 73 74 04 6E 72  -0-5.evlatest.nr
```

```
         * 61 6F 03 65 64 75 00 00 06 00 01                  ao.edu.....
         *
         * UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:53675
         * 12 F0 85 83 00 01 00 00 00 01 00 00 07 31 30 32  .............102
         * 2D 30 2D 35 08 65 76 6C 61 74 65 73 74 04 6E 72  -0-5.evlatest.nr
         * 61 6F 03 65 64 75 00 00 06 00 01 C0 14 00 06 00  ao.edu..........
         * 01 00 00 00 00 00 25 03 7A 69 61 03 61 6F 63 C0  ......%.zia.aoc.
         * 1D 04 74 65 63 68 C0 1D 00 00 00 1A 00 00 2A 30  ..tech........*0
         * 00 00 0E 10 00 36 EE 80 00 00 0E 10              .....6......
         *
         * UDP packet from 146.88.2.129:53675 to 146.88.1.4:dns(53)
         * 09 78 28 00 00 01 00 00 00 01 00 00 08 65 76 6C  .x(..........evl
         * 61 74 65 73 74 04 6E 72 61 6F 03 65 64 75 00 00  atest.nrao.edu..
         * 06 00 01 07 31 30 32 2D 30 2D 35 C0 0C 00 05 00  ....102-0-5.....
         * 01 00 00 00 09 00 08 05 63 6D 69 62 31 C0 0C     ........cmib1..
         *
         * UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:53675
         * 09 78 A8 85  00 00 00 00  00 00 00 00               .x..........
         */
        byte[] m2a = {0x2a, 0x2a, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00,
                      0x00, 0x00, 0x00, 0x00, 0x07, 0x31, 0x30, 0x32,
                      0x2D, 0x30, 0x2D, 0x35, 0x08, 0x65, 0x76, 0x6C,
                      0x61, 0x74, 0x65, 0x73, 0x74, 0x04, 0x6E, 0x72,
                      0x61, 0x6F, 0x03, 0x65, 0x64, 0x75, 0x00, 0x00,
                      0x06, 0x00, 0x01};

        byte[] m2b = {0x2b, 0x2b, 0x28, 0x00, 0x00, 0x01, 0x00, 0x00,
                      0x00, 0x01, 0x00, 0x00, 0x08, 0x65, 0x76, 0x6C,
                      0x61, 0x74, 0x65, 0x73, 0x74, 0x04, 0x6E, 0x72,
                      0x61, 0x6F, 0x03, 0x65, 0x64, 0x75, 0x00, 0x00,
                      0x06, 0x00, 0x01, 0x07, 0x31, 0x30, 0x32, 0x5f,
                      0x30, 0x5f, 0x35, -64, 0x0C, 0x00, 0x05, 0x00,
                      0x01, 0x00, 0x00, 0x00, 0x09, 0x00, 0x08, 0x05,
                      0x63, 0x6D, 0x69, 0x62, 0x31, -64, 0x0C};
        DatagramPacket dp2a = new DatagramPacket(m2a, m2a.length, ip, 53);
        DatagramPacket dp2b = new DatagramPacket(m2b, m2b.length, ip, 53);

        /*
         * > update delete cmib1.evlatest.nrao.edu A
         *
         * UDP packet from 146.88.2.129:53679 to 146.88.1.4:dns(53)
         * 61 A4 01 00 00 01 00 00 00 00 00 00 05 63 6D 69  a............cmi
         * 62 31 08 65 76 6C 61 74 65 73 74 04 6E 72 61 6F  b1.evlatest.nrao
         * 03 65 64 75 00 00 06 00 01                        .edu.....
         *
         * UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:53679
         * 61 A4 85 80 00 01 00 00 00 01 00 00 05 63 6D 69  a............cmi
         * 62 31 08 65 76 6C 61 74 65 73 74 04 6E 72 61 6F  b1.evlatest.nrao
         * 03 65 64 75 00 00 06 00 01 C0 12 00 06 00 01 00  .edu............
         * 00 0E 10 00 25 03 7A 69 61 03 61 6F 63 C0 1B 04  ....%.zia.aoc...
         * 74 65 63 68 C0 1B 00 00 00 1B 00 00 2A 30 00 00  tech........*0..
         * 0E 10 00 36 EE 80 00 00 0E 10                     ...6......
         *
         * UDP packet from 146.88.2.129:53679 to 146.88.1.4:dns(53)
         * 98 24 28 00 00 01 00 00 00 01 00 00 08 65 76 6C  .$(..........evl
         * 61 74 65 73 74 04 6E 72 61 6F 03 65 64 75 00 00  atest.nrao.edu..
         * 06 00 01 05 63 6D 69 62 31 C0 0C 00 01 00 FF 00  ....cmib1.......
         * 00 00 00 00 00                                   .....
         *
         * UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:53679
         * 98 24 A8 85 00 00 00 00 00 00 00 00              .$..........
         */
        byte[] m3a = {0x3a, 0x3a, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00,
                      0x00, 0x00, 0x00, 0x00, 0x05, 0x63, 0x6D, 0x69,
```

```
            0x62, 0x31, 0x08, 0x65, 0x76, 0x6C, 0x61, 0x74,
            0x65, 0x73, 0x74, 0x04, 0x6E, 0x72, 0x61, 0x6F,
            0x03, 0x65, 0x64, 0x75, 0x00, 0x00, 0x06, 0x00,
            0x01};
        byte[] m3b = {0x3b, 0x3b, 0x28, 0x00, 0x00, 0x01, 0x00, 0x00,
            0x00, 0x01, 0x00, 0x00, 0x08, 0x65, 0x76, 0x6C,
            0x61, 0x74, 0x65, 0x73, 0x74, 0x04, 0x6E, 0x72,
            0x61, 0x6F, 0x03, 0x65, 0x64, 0x75, 0x00, 0x00,
            0x06, 0x00, 0x01, 0x05, 0x63, 0x6D, 0x69, 0x62,
            0x31, -64, 0x0C, 0x00, 0x01, 0x00, -1, 0x00,
            0x00, 0x00, 0x00, 0x00, 0x00};
        DatagramPacket dp3a = new DatagramPacket(m3a, m3a.length, ip, 53);
        DatagramPacket dp3b = new DatagramPacket(m3b, m3b.length, ip, 53);

        /*
         * > update delete cmib1.evlatest.nrao.edu A
         * > update add cmib1.evlatest.nrao.edu 9 A 146.88.2.43
         * > update add 102-0-5.evlatest.nrao.edu 9 CNAME
         *              cmib1.evlatest.nrao.edu
         *
         * UDP packet from 146.88.2.129:53712 to 146.88.1.4:dns(53)
         * 64 1A 01 00 00 01 00 00 00 00 00 00 05 63 6D 69  d............cmi
         * 62 31 08 65 76 6C 61 74 65 73 74 04 6E 72 61 6F  b1.evlatest.nrao
         * 03 65 64 75 00 00 06 00 01                       .edu.....
         *
         * UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:53712
         * 64 1A 85 80 00 01 00 00 00 01 00 00 05 63 6D 69  d............cmi
         * 62 31 08 65 76 6C 61 74 65 73 74 04 6E 72 61 6F  b1.evlatest.nrao
         * 03 65 64 75 00 00 06 00 01 C0 12 00 06 00 01 00  .edu............
         * 00 0E 10 00 25 03 7A 69 61 03 61 6F 63 C0 1B 04  ....%.zia.aoc...
         * 74 65 63 68 C0 1B 00 00 00 21 00 00 2A 30 00 00  tech.....!..*0..
         * 0E 10 00 36 EE 80 00 00 0E 10                    ...6......
         *
         * UDP packet from 146.88.2.129:53712 to 146.88.1.4:dns(53)
         * B2 0D 28 00 00 01 00 00 00 03 00 00 08 65 76 6C  ..(..........evl
         * 61 74 65 73 74 04 6E 72 61 6F 03 65 64 75 00 00  atest.nrao.edu..
         * 06 00 01 05 63 6D 69 62 31 C0 0C 00 01 00 FF 00  ....cmib1.......
         * 00 00 00 00 00 C0 23 00 01 00 01 00 00 00 09 00  ......#.........
         * 04 92 58 02 2B 07 31 30 32 2D 30 2D 35 C0 0C 00  ..X.+.102-0-5...
         * 05 00 01 00 00 00 09 00 02 C0 23                 ..........#
         *
         * UDP packet from 146.88.1.4:dns(53) to 146.88.2.129:53712
         * B2 0D A8 85 00 00 00 00 00 00 00 00              ............
         */
        byte[] m4a = {0x4a, 0x4a, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00,
            0x00, 0x00, 0x00, 0x00, 0x05, 0x63, 0x6D, 0x69,
            0x62, 0x31, 0x08, 0x65, 0x76, 0x6C, 0x61, 0x74,
            0x65, 0x73, 0x74, 0x04, 0x6E, 0x72, 0x61, 0x6F,
            0x03, 0x65, 0x64, 0x75, 0x00, 0x00, 0x06, 0x00,
            0x01};
        byte[] m4b = {0x4b, 0x4b, 0x28, 0x00, 0x00, 0x01, 0x00, 0x00,
            0x00, 0x03, 0x00, 0x00, 0x08, 0x65, 0x76, 0x6C,
            0x61, 0x74, 0x65, 0x73, 0x74, 0x04, 0x6E, 0x72,
            0x61, 0x6F, 0x03, 0x65, 0x64, 0x75, 0x00, 0x00,
            0x06, 0x00, 0x01, 0x05, 0x63, 0x6D, 0x69, 0x62,
            0x31, -64, 0x0C, 0x00, 0x01, 0x00, -1, 0x00,
            0x00, 0x00, 0x00, 0x00, -64, 0x23, 0x00,
            0x01, 0x00, 0x01, 0x00, 0x00, 0x00, 0x09, 0x00,
            0x04, -110, 0x58, 0x02, 0x2B, 0x07, 0x31, 0x30,
            0x32, 0x2D, 0x30, 0x2D, 0x35, -64, 0x0c, 0x00,
            0x05, 0x00, 0x01, 0x00, 0x00, 0x00, 0x09, 0x00,
            0x02, -64, 0x23};
        DatagramPacket dp4a = new DatagramPacket(m4a, m4a.length, ip, 53);
        DatagramPacket dp4b = new DatagramPacket(m4b, m4b.length, ip, 53);
```

```
            byte[] buf = new byte[16384];
            DatagramPacket dpR = new DatagramPacket(buf, buf.length, ip, 53);

            s.send(dp2a);
            s.receive(dpR);
            System.out.println("rcv1 = " + dpR.getLength() + " bytes");
            for (int i=0; i < dpR.getLength(); i++) {
               System.out.printf("%02x, ", buf[i]);
            }
            s.send(dp2b);
            s.receive(dpR);
            System.out.println("\nrcv2 = " + dpR.getLength() + " bytes");
            for (int i=0; i < dpR.getLength(); i++) {
               System.out.printf("%02x, ", buf[i]);
            }
         }
         catch (UnknownHostException ue) {
            System.out.println("Unknown Host Exception : " + ue.getMessage());
         }
         catch (IOException ioe) {
            System.out.println("IO Exception : " + ioe.getMessage());
         }
      }
      public static void main(String[] args) {
         new DnsTest();
      }
}
```

References

[1] Carlson, B., INTERFACE CONTROL DOCUMENT: "EVLA Correlator System Numbering Plan", ICD Document: A25010N0002, Revision 1.1, February 24, 2005.

[2] Vixie, P., Request For Comments Document: RFC 2136 – "Dynamic Updates in the Domain Name System (DNS UPDATE)", April 1997.

[3] Conversation with James Robnett, and Rich Moeser about the devastating affect of hitting DNS too often as discovered on the EVLA antenna monitor and control system.