Don’t forget to check your ARP cache

On the whole the ARP cache works invisibly to help speed frames to their destinations. It is almost never the cause of any problems so is invariably overlooked when trouble shooting a problem. There are times however when the ARP cache is, if not the root of the problem, at least very closely involved.

The ARP cache

First off what is ARP and why does it have a cache? ARP is the Address Resolution Protocol. It is used by a host (source) to determine another host’s (target) media access control (MAC, for our purposes Ethernet) address based on its IP address. If a source host determines that a target host is on its local IP subnet it will broadcast an ARP request containing its IP and MAC addresses and the IP address of the target host. The target host responds with an ARP reply containing its MAC address. Once the source host receives the ARP reply it can format an Ethernet frame and send the packet. This is why it is so important that the subnet masks are correct. If the source host has the wrong subnet mask it may think that the target is on the same subnet and send an ARP request instead of sending the packet to a router. Or send the packet to the router when it should be sending the packet directly to the target host.

Obviously, it should not be necessary to determine the MAC address of the target host before every frame is sent. At the very least that would be a 200% increase in the number of frames on the Ethernet (one ARP request and one ARP reply for each frame containing data) and significantly reduce thru put. So the IP and MAC addresses are cached in the ARP cache.

Before a frame is transmitted the ARP cache is searched for an entry with the correct IP address. If the address is found in the cache the associated MAC address is used. If the IP address is not found, transmission of the frame is delayed while the ARP protocol is invoked to populate the cache. Once the cache is populated the frame is transmitted. It is a simple concept but like most simple concepts there are hidden complications.

The biggest issue is how long an address should reside in the ARP cache. You don’t want an address to reside permanently in the cache, after all what happens if the host’s Ethernet NIC fails and is replaced by another NIC with a different MAC address. You also don’t want the ARP cache cluttered up with hosts that were referenced only once last week, Every operating system has come up with its own answer. Every operating system also has its own commands to display and change these timers and manipulate the ARP cache itself. Before talking about the ARP cache problems I’ll review the ARP cache commands for each operating system and TCP stack.

VOS 14.4 OS_TCP

There is no way in OS_TCP to adjust the ARP cache timer (short of patching instructions, which we won't discuss). The timer is set for 10 minutes.

The "arp" command is used to manipulate the ARP cache. It can check for a specific entry, delete an entry, and even create an entry. What it cannot do is display the entire cache at once. For that you need to use analyze_system.

To display a single entry just invoke the "arp" command and give it a host name or an IP address. Since names are resolved into IP addresses invisibly and I've been burned by bad entries in host tables and DNS server problems I never use names if I can help it, so all my examples use IP addresses.

```
arp 164.152.77.50
```

It returns with the IP address, the corresponding name if it can figure it out or just the IP address again and the MAC address.

```
164.152.77.50 (164.152.77.50) at 0:90:27:d0:c0:3
```

If there is no entry you get output telling you that

```
arp 164.152.77.2
164.152.77.2 (164.152.77.2) -- no entry
```

The analyze_system request "dump_tcp_in_ifaddr –arp_entries" can be used to dump the entire ARP cache (along with some other stuff). The other stuff comes first and lists, in part, information about each interface including its MAC address. The last part of the output shows the ARP cache. The arptab entry is a pointer to the actual entry in the table and is useful only when doing some very nasty debugging. The flags field is important since it will allow you to distinguish a completed entry from an incomplete entry (flag = 01). An incomplete entry has not received a reply to the ARP request packet so any Ethernet address in the table is invalid. The most common flag is 03 which is a completed and in use entry. A flag value of 07 would indicate that the entry is permanent and a flag of 0F would indicate that the entry is permanent and published. The timer column counts the minutes that the entry has been in the cache. You should values between 0 and 9.

```
as:  dump_tcp_in_ifaddr -arp_entries

Internet Interface Address @ 0x801F4040
    network number of interface    0x7F000000
    mask of net part               0xFF000000
    subnet number, including net   0x7F000000
    mask of net + subnet           0xFF000000
    bdcast addr for (logical) net  0x00000000
    ia flags                       routing entry installed
    next internet interface addr   0x801FC820

ARPCOM Table information:

    acpcom table entries           32

    arpcom entry @ 0x00769A70
```

```
    device name                     #e7.4.12
    arpcom index                    0
    ip address                      164.152.77.227
    ethernet address                00:00:A8:80:51:07

    arpcom entry @ 0x00769B60
    device name                     #e7.4.13
    arpcom index                    1
    ip address                      172.20.1.7
    ethernet address                00:00:A8:81:51:07

    arpcom entry @ 0x00769C50
    device name                     #e7.4.6
    arpcom index                    2
    ip address                      172.22.1.7
    ethernet address                00:00:A8:82:51:07


ARPTAB Table Information:

    arp table buckets               37
    arp table bucket size           16
    arp table entries               592

ARPTAB Table Entries:

arptab entry   ip address      ethernet address  timer  flags
0x0076C700    164.152.77.73   00:90:27:C7:D9:5B    07     03
0x0076C7F0    164.152.77.222  00:02:B3:10:1A:8D    05     03
0x0076C8F0    164.152.76.146  00:90:27:A5:44:40    01     03
0x0076C900    164.152.77.38   00:00:A8:00:CA:7A    08     03
0x0076C910    164.152.18.9    00:04:C1:09:78:60    06     03
0x0076C9F0    164.152.77.76   00:90:27:A5:43:97    05     03
0x0076CA00    164.152.77.224  00:10:4B:29:39:9D    01     03
0x0076CAF0    164.152.77.40   00:00:C0:6F:A6:6B    06     03
0x0076CBF0    164.152.77.78   00:00:A8:81:2E:04    01     03
0x0076CDF0    164.152.76.151  00:01:03:89:9C:E9    04     03
```

An example of an incomplete entry:

```
as:  match 164.152.77.221; dump_tcp_in_ifaddr -arp_entries
0x00761150    164.152.77.221  00:00:00:00:00:00    00     01
```

However, before you go out and try this make sure that you are not on an early version of VOS 14.4. In VOS 14.4 a bug (av-1283) was introduced that makes this request fault. It's fixed in 14.4.1am and later.

You can delete an entry with the "arp –d" command

```
arp -d 164.152.77.50
164.152.77.50 (164.152.77.50) deleted
```

And you can create an entry with the "arp –s" command

```
arp -s 164.152.77.2 0:1:2:3:4:5
ready  16:41:41
```

```
arp 164.152.77.2
164.152.77.2 (164.152.77.2) at 0:1:2:3:4:5 permanent
```

The "arp –s" command also has options to specify that the entry be temporary instead of permanent and to publish the entry. Why should there be a command to manually create an ARP cache entry? It was originally useful when not all operating systems responded to ARP requests, something unheard of in today's environment. It is still useful when for some reason the ARP protocol is not working correctly (I'll discuss the major reason below) and you need to by pass it.

VOS 14.4 STCP

Under STCP there is an external variable called arp_cache_life in the arp.cp.pm module. This variable holds the current value for the ARP cache lifetime in seconds. The default is 600. You can display the current value with the display request in analyze_system:

```
as:  display arp_cache_life
FEC0B03C  0  00000258                          |...<            |
```

256 hex = 600 decimal = 10 minutes. You can change the value in the program module with the command set_external_variable or after the module has been loaded with the set_longword request in analyze_system. For example to change the life time to 1 minute you could do a

```
set_external_variable arp_cache_life -in
>system>kernel_loadable_library>arp.cp.pm -to 60 -type integer
```

or

```
as:  set_longword arp_cache_life 0000003Cx
addr      from      to
FEC0B03C  00000258  0000003C
```

To display all the entries in the ARP cache use the "arp –all" command

```
arp -all

Internet Address     MAC Address         Type    Life
172.16.1.100         00-03-47-3A-EC-40   temp     4 mins
172.16.1.102         00-00-4C-79-EB-83   temp     2 mins
172.16.1.5           00-00-BC-13-01-BA   temp    10 mins
```

Note that that the "temp" type has a life value that counts down from the arp_cache_life value, when it hits 0 it is deleted. To delete an entry without waiting for its life to expire use the "arp –delete" command.

```
arp -delete 172.16.1.102

Mapping for 172.16.1.102 deleted
```

Finally to set a value in the ARP cache you can use the "arp –set" command. You can use either the colon character (:) or a dash (-) to separate the bytes in the MAC address. The following example also shows that you can display a single ARP cache entry with the "arp" command just by providing the name or IP address to the command.

```
arp 172.16.1.23 -set 0:1:2:3:4:5
172.16.1.23 mapped to mac address 00-01-02-03-04-05

ready  16:15:29
```

```
arp 172.16.1.23

Internet Address      MAC Address          Type    Life
172.16.1.23           00-01-02-03-04-05    perm
```

FTX 3.4 and HP-UX 11

Both FTX 3.4 and HP-UX 11 have the same command set and, except for minor differences when displaying the ARP cache, the same output. The tuning parameter arp_cleanup_interval holds the ARP cache lifetime in milliseconds. The default value is 300,000 or 5 minutes. To display the current value use the "ndd –get" command.

```
# ndd -get /dev/arp arp_cleanup_interval
300000
#
```

To change the lifetime to 1 minute use the command

```
# ndd -set /dev/arp arp_cleanup_interval 60000
#
```

To display the ARP cache use the "arp –a" command. This is the only place where FTX 3.4 and HP-UX 11 are slightly different. FTX 3.4 looks like

```
# arp -a
paradisevalley(164.152.77.50)  (00:90:27:d0:c0:03) inuse complete
pandora(164.152.77.120)  (00:00:bc:0f:0c:81) permanent publish local complete
twinpeak(164.152.77.100)  (00:a0:c9:5b:dc:ef) complete
```

The complete flag just means that it is a valid entry. The inuse means that the entry is being referenced, permanent means the entry will not timeout and publish means that it will respond to requests for that IP address with the associated MAC address.

On the other hand HP-UX 11 looks like

```
# /usr/sbin/arp -a
corpdc3 (164.152.77.248) at 0:90:27:c7:90:f9 ether
paradisevalley (164.152.77.50) at 0:90:27:d0:c0:3 ether
```

The ether means that the type of MAC address is Ethernet.

The "arp –d" is used to delete an entry

```
# arp -d 164.152.77.100
#
```

To set an entry use the "arp –s" command. For those of you wondering why the static entry that I created has a netmask, I have no idea.

```
# arp -s 164.152.77.40 0:1:2:3:4:5
# arp 164.152.77.40
noah/255.255.255.255 (00:01:02:03:04:05) permanent complete
```

ftServer/Windows 2000

The ARP cache lifetime is controlled by two entries in the Window 2000 registry. Both entries are in the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters

branch of the registry. The first is the ArpCacheLife. If this value is greater than the value of ArpCacheMinReferencedLife then entries in the ARP cache time out in ArpCacheLife seconds. If this value is not set or is smaller than ArpCacheMinReferencedLife (default is 600 seconds) then unreferenced entries time out in 120 seconds and referenced entries time out in 600 seconds. An entry is referenced every time it is used. These values are not in the registry by default so to change then you first need to create them. You also need to reboot after making the change in order for the change to take effect.

To display the ARP cache use the"arp –a" command. A dynamic type is temporary, i.e. it will time out and a static type is permanent.

```
c:\>arp -a

Interface: 192.168.1.101 on Interface 0x1000003
  Internet Address      Physical Address      Type
  192.168.1.102         00-50-bf-16-ba-ae     dynamic
```

To delete an entry use the "arp –d" command

```
c:\>arp -d 192.168.1.150

c:\>
```

And you can create an entry with the "arp –s" command. Just to keep things entertaining the bytes in the MAC address must be separated with dashes (-), using colons will generate an error.

```
c:\>arp -s 192.168.1.150 0-1-2-3-4-5

c:\>arp -a

Interface: 192.168.1.101 on Interface 0x1000003
  Internet Address      Physical Address      Type
  192.168.1.150         00-01-02-03-04-05     static
```

ARP cache problems

Now that I have reviewed the ARP cache and the commands used to manipulate it I'll cover the problems that occur. More accurately the problem that can occur since the only thing that can happen is that a wrong entry is placed in the ARP cache.

The simplest and most common way that this can occur is that a target host's NIC fails and is replaced with another. Obviously this has to happen before the ARP cache timeout occurs but if the target host has a backup NIC that comes on-line automagically it can happen. If it does, then until the ARP cache entry is replaced with the correct entry it will be impossible for the two hosts to communicate. You could just wait for the entry to time out of the ARP cache but that may create too long of an outage. Deleting the entry from the ARP cache and letting the source host send a new ARP request is the simplest solution. This should not occur if the target host is a Stratus using any of the Stratus based Ethernet duplexing techniques but I cannot speak about other platforms.

Another way that the ARP cache can end up containing an incorrect entry (and the mechanism that caused me to think about writing this article) is when a third host responds to the ARP request. If it responds with the wrong address then you are up the preverbal creek, but you do have a paddle. Assuming that you can determine the real Ethernet MAC address of the target host you can create a permanent entry in the ARP cache of the source host mapping the target host's IP address to its MAC address. Once this is done the ARP protocol is not invoked so there can be no problem.

Why should a host respond to an ARP request for something that is not its IP address?

Lets start with the least likely, which is that someone created a permanent entry and told the system to "publish" it. Of course if they added the entry in correctly this shouldn't be a problem. But even it was added correctly last week it doesn't mean its correct now. I cannot think of a good reason why anyone would be publishing ARP entries so if you see an ARP cache entry with a publish flag it is worth investigating. Assuming that you are looking at this from the source host's perspective, how can you tell who is publishing the ARP value? The MAC address of the publisher is in its ARP reply packet so a packet_monitor (or any protocol analyzer's) trace will give you that. Tracking down the host based on its MAC address can be as simple as looking it up in a table (if you have kept your records up to date) or as difficult as logging into each host on the subnet and checking.

The most common way for the source host to get an ARP reply containing an incorrect entry is when a router is configured for proxy ARP. What happens is that a router is configured with a route to network or more probably subnet X.Y.Z. It receives an ARP request for a target host X.Y.Z.22. The router responds with its own MAC address. Now the first thing to consider is that the source host probably has an incorrect subnet mask. If there is a router configured on the subnet with a route to X.Y.Z then chances are all the local hosts should be on a different subnet. If the source host thinks that X.Y.Z.22 is local (which is must or it wouldn't have sent an ARP request) its subnet mask is probably wrong. Yes, it is possible that the router is configured wrong but that is less likely. Under ideal circumstances there will be no communication problems since the source host will send packets to the router and the router will forward them as needed. If you look at the source host's ARP cache what you will see is a lot of hosts with the same MAC address. In the following example I have use the match option in analyze_system to show just some of the addresses.

```
as:  match 164.152.1; dump_tcp_in_ifaddr -arp_entries
0x00766E30    164.152.18.9    00:04:C1:09:78:60    01     03
0x00767320    164.152.11.30   00:04:C1:09:78:60    08     03
0x00767910    164.152.12.76   00:04:C1:09:78:60    01     03
0x00767B20    164.152.11.1    00:04:C1:09:78:60    09     03
0x00767E30    164.152.11.4    00:04:C1:09:78:60    08     03
```

If you dump the arp_cache and match on the MAC address, one extra entry appears, 164.152.77.1, which turns out to correspond to the address of the router.

```
as:  match 00:04:C1:09:78:60; dump_tcp_in_ifaddr -arp_entries
0x00761350    164.152.18.9    00:04:C1:09:78:60    00     03
0x00761640    164.152.76.1    00:04:C1:09:78:60    04     03
0x00761840    164.152.11.30   00:04:C1:09:78:60    07     03
0x00761E30    164.152.12.76   00:04:C1:09:78:60    00     03
0x00762040    164.152.11.1    00:04:C1:09:78:60    08     03
0x00762350    164.152.11.4    00:04:C1:09:78:60    07     03
```

There may be a valid reason for configuring a host with an "incorrect" subnet, especially under OS_TCP with its limited subnet capabilities. Other than having an ARP cache larger than need be and some subtle TCP option differences (TCP uses different maximum segment sizes for local and remote connections) things should work fine.

Now, what happens if you have two subnets connected together via a hub or on the same VLAN. Call then subnets A and B. If on subnet B there is a router configured for proxy ARP and it has a route to subnet A or just a default route then it can respond to all ARPs for hosts on subnet A with its MAC address. If the router really has a route back to subnet A then at worst you just have a very inefficient network. But if there is no route back to subnet A then communication between hosts on subnet A will be mostly impossible. Communication is not impossible because both the target host and the router will respond with an ARP reply. The source host will use the last one received so depending on the order of the responses communication may work or may work for a while. The only way to diagnose this problem is to be aware of the MAC address of the target hosts. Or you can watch the ARP cache entries and see if they change. You can also look at a packet trace to see if the incoming packet is coming from the same MAC address that the reply is addressed to. Finally, you can look up the OUI (organizationally unique identifier), which is the first 3 bytes of the MAC address. If the MAC address identifies a router company you can be pretty sure that you are getting a proxy ARP reply and not a reply from your target host. OUIs can be found at
http://standards.ieee.org/regauth/oui/index.shtml

In summary, when debugging a problem I always try to check the lower layers before looking at higher layers. Typically I check for layer 2 errors like CRC errors and late collisions. Forgetting that the MAC address of the target host is also a layer 2 component made for a very frustrating couple of hours as I watched communication between hosts come and go randomly. I'm not likely to forget again (or at least not any time soon).