We have three nodes, which have the following topology:

NDS-Agenda21Haus70 <-> sn04 <-> sn03

Something happened, so they came into a stable (for hours) but inconsistent state of their translation tables:

- sn03 has 11 entries in its translocal table and crc = 0x2b6d458c
- sn04 has only 10 entries in the table for sn03 but still crc = 0x2b6d458c
- NDS-Agenda21Haus70 has also 10 entries in the table for sn03 but crc = 0x74fb4f96

It seems as like something (erroneously) removed the entry 33:33:00:00:00:01 from table on node sn04 without recalculating the crc sum.

In the following are the outputs of the table states:

root@NDS-Agenda21Haus70:~# batctl tr 88:e6:40:20:30:01
> traceroute to 88:e6:40:20:30:01 (88:e6:40:20:30:01), 50 hops max, 20 byte packets
>
> root@NDS-Agenda21Haus70:~# batctl tg | grep 88:e6:40:20:30:01
> * 33:33:ff:06:3e:25 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> * 33:33:ff:00:18:32 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> 01:00:5e:00:00:02 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> 01:00:5e:00:00:01 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> 33:33:ff:00:00:00 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> * 33:33:ff:00:30:01 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> * 2a:d7:9a:06:3e:25 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> * 33:33:ff:00:01:08 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> 33:33:00:01:00:03 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
>
> [root@sn04]~ # batctl tg | grep 88:e6:40:20:30:01
> * 33:33:ff:06:3e:25 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> * 33:33:ff:00:18:32 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> 01:00:5e:00:00:02 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> 01:00:5e:00:00:01 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> 33:33:ff:00:00:00 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> * 33:33:ff:00:30:01 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> * 2a:d7:9a:06:3e:25 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> * 33:33:ff:00:01:08 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
> 33:33:00:01:00:03 -1 [....] ( 3) 88:e6:40:20:30:01 ( 3) (0x74fb4f96)
>
> [root@sn03]~ # batctl tl
> [B.A.T.M.A.N. adv 2018.0-3-g4b2b8c68, MainIF/MAC: mesh_fastd/88:e6:40:20:30:01 (bat0/2a:d7:9a:06:3e:25 BATMAN_IV), TTVN: 3]
> Client VID Flags Last seen (CRC)
> 33:33:ff:06:3e:25 -1 [.P....] 0.000 (0x2b6d458c)
In the attachment, I added a capture of the received full table response from sn04 -> NDS-Agenda21Haus70 taken on NDS-Agenda21Haus70. I checked, that this is really an intermediate response of sn04 and not a direct response from sn03 (sn03 is not receiving queries from NDS-Agenda21Haus70).

### History

#### #1 - 05/10/2018 08:37 AM - Sven Eckelmann

- Description updated

#### #2 - 05/11/2018 06:57 PM - Sven Eckelmann

Wild and untested idea:

- two events happen at the same time at sn04
- both are processed at the same time in different contexts
  - context a sees the list (hash) of entries not yet updated (11 entries) and calculates crc 0x2b6d458c
  - context b sees the list (hash) of entries which was already updated (10 entries) and calculates crc 0x74fb4f96
- _batadv_tt_global_update_crc_ does the crc calculation without a lock and thus operations can happen interleaved
  - context b finishes early (seen from crc memory write/commit perspective) and saves crc 0x74fb4f96
  - context a finishes late (seen from crc memory write/commit perspective) and saves crc 0x2b6d458c

The hash (_bat_priv->tt.local_hash_) will then have only 10 entries and the crc of 11 entries. It will therefore not recover itself using the data from the originator to which the global entries belong to (because the crc is correct). It will still forward (as "knowing" intermediate node) the wrong number of entries to the querier of the TT full table for this originator.

Looks to me like the _tt.crc_ access must always be protected with a specific lock for the complete hash (_bat_priv->tt.local_hash_ or _bat_priv->tt.global_hash_).

#### #3 - 05/11/2018 07:24 PM - Sven Eckelmann

Antonio just mentioned that TT responses or OGM messages could trigger recalculations of the CRC. And multiple recalculations for the same originator are prevented using the _orig_node->tt_lock_ (only different originators can calculate their CRC at the same time).

#### #4 - 06/05/2018 10:44 PM - Linus Lüssing

From looking at the code it seems to me that a node wrongly adding a ROAM or TEMP flag to a multicast TT entry in an OGM or TT Response might cause similar issues in _tt_global_add()_ either here:
Which would delete a multicast TT entry for the same address on any other node, without updating any CRC.

As in Leonardo's setup we already observed bogus multicast TT entries with the 'W' and 'I' flag set it seems likely that this/those broken node(s) might inject a ROAM or TEMP flag in multicast TT entries, too.

If we can verify that this is indeed the issue for Leonardo then a simple fix should be to add a "lis_multicast_ether_addr()" check for the referenced two cases, to avoid interpreting these flag bits as ROAM or TEMP for multicast TT.

I'll hopefully be able to verify this this weekend.

#5 - 06/07/2018 01:16 AM - Linus Lüssing
These two patches hopefully fix this issue:

- batman-adv: Avoid storing non-TT-sync flags on singular entries too
- batman-adv: Fix multicast TT issues with bogus ROAM flags

Leonardo, would be great if you could check that this fixes it for you, too.

#6 - 06/12/2018 09:43 PM - Sven Eckelmann
- Target version set to 2018.2
- Status changed from New to Resolved

Patches were added as:

- https://git.open-mesh.org/batman-adv.git/commit/beb6246b23398526a429ae9259a8eb30a685041
- https://git.open-mesh.org/batman-adv.git/commit/c7054fff3e0c3b08bb4bef3cffee1e0a543e14096

#7 - 07/17/2018 07:25 PM - Sven Eckelmann
- Status changed from Resolved to Closed

batman-adv 2018.2 was just released

Files

| missing_tg_entry.pcap | 210 Bytes | 05/10/2018 | Sven Eckelmann |