On incoming translation tables and ogms diffs, remove \texttt{is\_wifi} \& \texttt{isolation} flag from announced multicast addresses when storing locally.

Related issues:
- Related to batman-adv - Bug \#336: tt: store \texttt{is\_wifi} \& \texttt{isolation} flag per announcing originator
- Has duplicate batman-adv - Bug \#338: batman-adv2017.1 unstable with multicast...  
  \texttt{Closed} 06/10/2017
- Has duplicate batman-adv - Bug \#338: batman-adv2017.1 unstable with multicast...  
  \texttt{Rejected} 07/07/2017

History

\textbf{#1} - 07/07/2017 09:31 PM - Sven Eckelmann
- Related to Bug \#338: batman-adv2017.1 unstable with multicast optimization added

\textbf{#2} - 07/07/2017 09:31 PM - Sven Eckelmann
- Related to deleted (Bug \#338: batman-adv2017.1 unstable with multicast optimization )

\textbf{#3} - 07/07/2017 09:31 PM - Sven Eckelmann
- Has duplicate Bug \#338: batman-adv2017.1 unstable with multicast optimization added

\textbf{#4} - 07/08/2017 04:33 PM - Sven Eckelmann
- Related to Bug \#336: tt: store \texttt{is\_wifi} \& \texttt{isolation} flag per announcing originator added

\textbf{#5} - 07/11/2017 02:27 AM - Linus Lüssing
- Priority changed from Normal to High

\textbf{#6} - 07/11/2017 02:27 AM - Linus Lüssing
- Priority changed from High to Normal

\textbf{#7} - 02/27/2018 01:27 PM - Sven Eckelmann
Isn't this already done by patch \url{https://patchwork.open-mesh.org/patch/17072/} or did I miss something?

\textbf{#8} - 02/28/2018 12:22 AM - Linus Lüssing
Yes, the main issue was fixed by that patch.
But at the last BattleMesh we (or better Antonio) identified that there is one more part where flags on multicast TT entries are currently interpreted which is the isolation code. However, parsing such multicast TT flags there makes no sense.

One suggestion at the BattleMesh was to try to fix up multicast TT entries. That is, simply unsetting the \texttt{is\_wifi} and isolation flags before storing them, if I did not misunderstand the proposal. And that's how I understood the title of this ticket.

However, since the \texttt{is\_wifi} and isolation flags are part of the SYNC mask, protected by the TT checksum I feel a bit uneasy altering them. We need to be able to respond with the original values if requested by other nodes.

Also, tempering with those flags would make it impossible to use them in the future.

Instead I would suggest to alter the isolation code to ignore multicast TT entries.
Linus Lüssing wrote:

However, since the is_wifi and isolation flags are part of the SYNC mask, protected by the TT checksum I feel a bit uneasy altering them. We need to be able to respond with the original values if requested by other nodes.

I think the ultimate goal should to ensure that multicast entries are never sent around with the wifi of isolation flag set. So this should not just be a receiver filtering but also a sender one. In theory those flag should never really be set for mcast entries, but apparently this is not the case. no?

Linus Lüssing wrote:

In theory those flag should never really be set for mcast entries

Maybe that's what I still do not quite understand. Just to get on the same page, could you explain what issues would currently arise? Especially for these two scenarios:

A) A broken machine were to set such bits in multicast entries.

B) Some protocol addition for multicast were starting to set and use those bits (with a different semantic and purpose) in multicast entries.

Just to make sure we all understand what the issues would be and to clarify the purpose of this ticket a bit.

Linus Lüssing wrote:

In theory those flag should never really be set for mcast entries

Maybe that's what I still do not quite understand. Just to get on the same page, could you explain what issues would currently arise? Especially for these two scenarios:

A) A broken machine were to set such bits in multicast entries.

The patch you submitted is bypassing the "isolation" logic, therefore I guess that it would also solve any problem coming from having these bits set on a mcast entry. Isolation/wifi flags on a mcast entry might just prevent packets to/from those mcast address to flow to/ from the host as they would be dropped (if isolation is on).
Still, your patch makes sense because the isolation logic is thought for unicast entries, however I think we should also solve the problem at the other end, where the sender ensures that those flags are not set at all).

B) Some protocol addition for multicast were starting to set and use those bits (with a different semantic and purpose) in multicast entries.

This is ok, as soon as we handle those flags properly. My concern now is that we have not designed any component to expect those flags, but we still have cases where they are set. To be "safe" I would just prefer those flags to never travel across the mesh on a mcast entry so that we can avoid any unwanted behaviour. Alternatively, we should probably do a review of all the flags consumers and make sure they handle this case gracefully.

tldr;
I am not against having those flags set on mcast entries, but as of now it is not expected. Therefore, we either make sure the code can deal with them or we prevent them from being set/sent at all.

#12 - 03/11/2018 10:53 AM - Linus Lüssing

however I think we should also solve the problem at the other end, where the sender ensures that those flags are not set at all

Sending side tests:

A-B-C

- 3x VMs / nodes, clients simulated via a veth pair bridged with bat0
- One node (B) has the bridge multicast querier enabled
- Extended Isolation was enabled as suggested in the according wiki page on node A

Result:

- Isolation flag is set properly in the unicast TT entry for the veth-client on node A and propagates to the global TT on node B and C
- No isolation flag is set in a multicast TT entry for the veth-client on node A, no isolation flag appears in the global TT table on node B or C

Code/Explanation:

- multicast TT entries are only added via a call to batadv_tt_local_add() in batadv_mcast_mla_tt_add()/multicast.c
- There, BATADV_NULL_IFINDEX and BATADV_NO_MARK are provided to the batadv_tt_local_add() call
- This results in both the "if (batadv_is_wifi_hardif(in_hardif))" and "match_mark == bat_priv->isolation_mark" in batadv_tt_local_add() failing, therefore never setting the isolation or wireless flag on a multicast TT entry via this call
- batadv_tt_local_add() is the only potential place to add the isolation and wireless flag for a TT entry on the sender/originating side

Conclusion:

- Sender side looks good to me

Example TT output:

NodeA# batctl tl
Client VID Flags Last seen (CRC ...)
33:33:ff:bf:ab:22 -1 [.P....] 0.000 (0xbeee4d2ff)
01:00:5e:00:00:01 -1 [.P....] 0.000 (0xbeee4d2ff)
NodeC# batctl tg
[B.A.T.M.A.N. adv 2018.0-7-g0509fc0c, MainIF/MAC: ens3/02:92:64:a4:39:c3 (bat0/ca:35:05:5d:80:5a BATMAN_IV)]

Client VID Flags Last ttvn Via ttvn (CRC )
* 33:33:ff:5b:fe:25 -1 [....I] ( 7) 02:50:64:a4:39:c2 ( 7) (0x7a8af783)
 01:00:5e:00:00:01 -1 [....I] ( 6) 02:42:64:a4:39:c1 ( 8) (0xbee4d2ff)
 * 01:00:5e:00:00:01 -1 [....I] ( 5) 02:50:64:a4:39:c2 ( 7) (0x7a8af783)
 01:00:5e:00:00:01 -1 [....I] ( 5) 02:50:64:a4:39:c2 ( 7) (0x7a8af783)
 * fea1:b8:04:6c:73 -1 [....I] ( 3) 02:42:64:a4:39:c1 ( 8) (0xbee4d2ff)
* fea1:b8:04:6c:73 -1 [....I] ( 3) 02:42:64:a4:39:c1 ( 8) (0xbee4d2ff)
* 06:f3:1c:af:77:27 -1 [....I] ( 3) 02:50:64:a4:39:c2 ( 7) (0x70e4f6de)
* 06:f3:1c:af:77:27 -1 [....I] ( 3) 02:50:64:a4:39:c2 ( 7) (0x70e4f6de)
* 33:33:ff:af:77:27 -1 [....I] ( 3) 02:50:64:a4:39:c2 ( 7) (0x7a8af783)
* 33:33:ff:af:77:27 -1 [....I] ( 3) 02:50:64:a4:39:c2 ( 7) (0x7a8af783)
* fea1:b8:04:6c:73 -1 [....I] ( 2) 02:42:64:a4:39:c1 ( 8) (0xbee4d2ff)

33:33:ff:bf:ab:22 is comming from the veth-client bridged into the mesh on node A, e6:ed:9e:bf:ab:22 is its according unicast MAC address:

NodeA# brctl show
bridge name bridge id STP enabled interfaces
br0 8000.f656cd2021aa no bat0
      veth0
NodeA# ip addr show dev veth1
10: veth1:veth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether e6:ed:9e:bf:ab:22 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::e4ed:9eff:febf:ab22/64 scope link valid_lft forever preferred_lft forever
NodeA# ip addr show dev vethl
10: vethl
    link 33:33:00:00:00:01
    link 01:00:5e:00:00:01
    link 33:33:ff:bf:ab:22
    inet 224.0.0.1
    inet6 ff02::1:ffbf:ab22
    inet6 ff02::1
    inet6 ff01::1

In the translation tables quoted earlier, e6:ed:9e:bf:ab:22 has the isolation flag set and 33:33:ff:bf:ab:22 hasn't. Just as we would like things to happen.
My concern now is that we have not designed any component to expect those flags. [...] Alternatively, we should probably do a review of all the flags consumers and make sure they handle this case gracefully.

For the consumer side if I remember correctly we had been looking at `_batadv_is_ap_isolated()` in particular during the Wireless BattleMesh v10. And indeed there appeared to be no pre-checks regarding multicast vs. unicast.

As far as I can tell, this function is the only place "consuming" the wifi and isolation flags (other than debug output functions). Please correct me, if I'm wrong.

I would start injecting isolation and wifi TT flags in a test setup. Do you have some recommendations regarding what I should test and check specifically? What could potentially fail if there were checks missing in or around `_batadv_is_ap_isolated()`?

---

I would start injecting isolation and wifi TT flags in a test setup.

For multicast TT entries, that is.

---

Ok, reviewed the consuming part for the TT wifi and TT isolation flags now, too.

tl;dr: I see two parts needing fixing. A `transtable_search()` via `batadv_gw_out_of_range()`. And a `transtable_search()` via `batadv_mcast_forw_tt_node_get()`.

The latter can be easily and should be fixed by calling `transtable_search()` without a src parameter. That avoids an isolation check.

`batadv_gw_out_of_range()` needs some more thorough review though. However, issues here should only be noticeable for DHCPv6 or for IPv4 if a `FF:FF:FF:FF:FF:FF` ends up in the global TT, I think. Also see #351.

Here are my very rough notes from the review. I think I covered all code paths. But let me know if I missed something.

Mesh to local:
- On target node for multicast packets

  `- checks BATADV_TT_CLIENT_ISOLA for src address on target node
  `- checks for multicast dst (incl. bcast) address only
  `- does *not* check multicast dst TT flags

  => OK!
Mesh to local:
- On target node for unicast packets
  __batadv_is_ap_isolated
  - checks for DST MAC in tt-local
  - checks for SRC MAC in tt-global
  - returns "true", if both exist and both have either wifi or isolation flag
  <- batadv_interface_rx
  - only calls if "dst != multicast"
    => OK!
  <- batadv_recv_bcast_packet
  <- batadv_rx_handler[
  <- batadv_batman_skb_recv
  <- batadv_recv_unicast_packet
  <- batadv_rx_handler[
  <- batadv_batman_skb_recv

Local to mesh:
  __batadv_is_ap_isolated
  <- batadv_transtable_search
  - checks for SRC MAC in tt-local
  - checks for DST MAC in tt-global
  <- batadv_gw_out_of_range
  - only called if BATADV_DHCP_TO_SERVER
  - BATADV_DHCP_TO_SERVER only returned for DHCPv4 to UDP Port 67 or DHCPv6 to UDP port 547
  - BATADV_DHCP_TO_SERVER only set if is_multicast dst in
    batadv_interface_tx (so dst is either FF:FF:FF:FF:FF:FF for DHCPv4 or 33:33:00:01:00:02/33:33:00:01:00:03 for DHCPv6)
  - acc. to kerneldoc should return false for multicast destination, seems like a transtable lookup for multicast was supposed to check that and indicate it by returning NULL? (also see #351)
    => Not OK?
  <- batadv_interface_tx
  <- batadv_mcast_forw_tt_node_get
  - only called if there is a single, matching multicast destination/listener
  <- batadv_mcast_forw_mode
  - only calls if "dst == multicast" (excl. broadcast)
    - Potential solution: Use NULL instead of ethhdr->h_source in
      batadv_mcast_forw_tt_node_get()
      (and clarify use of src in kerneldoc of batadv_transtable_search(), that it is optional and that it is used for AP isolation check)
  <- batadv_interface_tx

? to ?:
  <- batadv_reroute_unicast_packet (Trigger: Mesh to Local, TX: "Local" to Mesh)
  - calls transtable_search() with src = NULL
  - therefore no isolation check
    => OK!
  <- batadv_check_unicast_ttvn (2x)
  <- batadv_recv_unicast_packet
  <- batadv_rx_handler[
  <- batadv_batman_skb_recv
  <- batadv_send_skb_via_tt_generic
    <- batadv_send_skb_via_tt
      <- batadv_dat_snoop_incoming_arp_request (Trigger: Mesh to Local, TX: "Local" to Mesh)
        - gets hw_src from incoming ARP request
        - crafts an ARP reply with acc. unicast hw_src now as ethernet destination
          => OK!
  <- batadv_recv_unicast_packet
  <- batadv_rx_handler[]}
Ah, I just wanted to start creating a setup to reproduce any issues with batadv_mcast_forw_tt_node_get() and then realized that I can't:

This function is only called if there is a single recipient. So I can't use another one with broken multicast TT flags to influence a legitimate receiver.

Nevertheless, would need fixing before making use of these TT flag bits. Just not suitable for stable@ now, I think.

So, to me batadv_gw_out_of_range() is the only place with potential but current issues remaining.
The two issues mentioned above were fixed in the following commits:

- batman-adv: fix packet loss for broadcasted DHCP packets to a server
- batman-adv: fix multicast-via-unicast transmission with AP isolation

To me the suggested change (Title: "tt: remove is_wifi & isolation flag from multicast tt entries") became obsolete now and I think this ticket can be closed.

(Note however that the TT ROAM flag, which is not part of the TT sync flag set, probably needs a similar review before any potential reuse.)

---

#18 - 05/23/2018 08:15 AM - Sven Eckelmann

- Target version set to 2018.1
- Status changed from New to Closed