Block Address Registration and Claiming (BARC)

Roger Marks
EthAirNet Associates
2021-03-12
Acknowledgements

thanks to Antonio de la Oliva
for review and constructive comments

thanks to other 802.1 WG and 1722 WG participants for constructive discussions during earlier comment resolution, particularly to Mick Seaman, Norm Finn, Don Pannell, and Max Turner
Introduction

- P802.1CQ/D0.5 was reviewed in Task Group Ballot.
- Comment resolution indicated that significant changes were required.
- This contribution summarizes an approach to the next draft.
- Approach is based on address blocks, as raised during comment resolution.
- Each Address Block (AB) is identified by an Address Block Identifier (ABI).
- Some ABIs (CABIs) are designated for self-claiming.
- Other ABIs (RABIs) are designated for management by a Registrar.
- Address Registration and Claiming (ARC) protocol is outlined here, with:
  - Block Address Registration and Claiming (BARC)
  - Address-Range (AR) claiming, incorporating MAAP from IEEE 1722, per comment resolution.
    - not using any BARC addresses
BARC assigns MAC Addresses in Address Blocks

1) Address Blocks (ABs) include local addresses.

2) An AB includes both unicast and multicast address subblocks.

3) No address falls within more than one AB.

4) An AB Identifier (ABI) identifies each AB.

5) Registrars hold multi-blocks sets of ABs, identified by multi-block identifiers (MBIs).

6) An ABI within an MBI is a registrable ABI (RABI).
   - identifies Registrable Address Blocks (RABs) holding Registrable Addresses (RAs)

7) An ABI not in an MBI is a Claimable ABI, claimable by a Claimant without a Registrar.
   - identifies Claimable Address Blocks (CABs) holding Claimable Addresses (CAs)

8) Each ABI and MBI is a multicast address and not in any AB.

9) A large set of Temporary Unicast Addresses (TUAs) is specified
   - for initial discovery by Claimant lacking a unicast address.
# MAC Address Categorization

<table>
<thead>
<tr>
<th>determinable via inspection:</th>
<th>Expanded name</th>
<th>Role</th>
<th>I/G</th>
<th>indicates, by inspection</th>
</tr>
</thead>
<tbody>
<tr>
<td>not specified by BARC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>claimable address, in claimable address block (CAB)</td>
<td>BA</td>
<td>U,M</td>
<td>CABI, CABI type, CAB (including all other CAs in CAB)</td>
</tr>
<tr>
<td>CABI</td>
<td>CAB identifier</td>
<td>BI</td>
<td>M</td>
<td>CABI type, CAB</td>
</tr>
<tr>
<td>MBI</td>
<td>multi-block identifier</td>
<td>MBI</td>
<td>M</td>
<td>MBI type, RABI type, RABIs</td>
</tr>
<tr>
<td>RA</td>
<td>registrable address, in registrable address block (RAB)</td>
<td>BA</td>
<td>U,M</td>
<td>RABI, RABI type, MBI, MBI type, all other RAs in RAB</td>
</tr>
<tr>
<td>RABI</td>
<td>RAB identifier</td>
<td>BI</td>
<td>M</td>
<td>RABI type, MBI, MBI type, RAB (including all other RAs in RAB)</td>
</tr>
<tr>
<td>RABIA</td>
<td>(unicast) RABI address (one of many)</td>
<td>BI</td>
<td>U</td>
<td>RABI</td>
</tr>
<tr>
<td>TUA</td>
<td>temporary unicast address</td>
<td>U</td>
<td></td>
<td>note: ~6.8E10 to choose among</td>
</tr>
</tbody>
</table>

BA = block address; BI = block identifier
## BARC MAC Address Structure

<table>
<thead>
<tr>
<th>N11</th>
<th>0</th>
<th>always 0 (remaining 15 values for non-BARC use)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N10</td>
<td>1 1 1 M</td>
<td>M is the usual multicast (I/G) bit; 111 is local “SAI” range per IEEE Std 802c</td>
</tr>
<tr>
<td>N9</td>
<td>p r j k</td>
<td>address block includes subblocks of $16^{ijk}$ contiguous addresses</td>
</tr>
<tr>
<td>N8</td>
<td>n a b c</td>
<td>multi-block includes $16^{abc}$ address blocks (N8 unstructured in CABI)</td>
</tr>
<tr>
<td>N7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- M, p, r, and n bits distinguish Block Addresses (BAs) and Block Identifiers (BIs)
  - e.g. CABI has M=1, p=0, r=0
- see Appendix for details

12 nibbles per 48-bit address
Claimant of $CABI_X$ AB listens to $CABI_X$ multicast address

(1) $CABI_1$: DISCOVER state

(2) (unicast) $CABI_1$:CLAIMED state

(4) $CABI_6$: CLAIMED state

start listening to $CABI_6$
Registrar

- Registrar maintains an inventory of RABIs (within MBIs).
  - a protocol specifies how Registrars acquire MBIs.
  - set of RABIs is disjoint from the set of CABIs
    - AB is either claimable (CABI) or registrable (RABI); not both

- Registrar listens for all messages to a CABI.
  - M=1, p=0, r=0, i.e. DA begins 0000-1111-00
    - [MMRP NumberOfValues field is 13 bits]

- Registrar can respond to a DISCOVER with an OFFER of an AB in its inventory.
  - The OFFER defends the DISCOVER message’s CABI.
  - Client claims an offered RABI, similar to claiming CABI.
    - Registrar does not assign RABI but tracks its registration.
  - OFFER cites one of the RABI’s RABIA, not the RABI directly.
    - claim is then sent to Register at the unicast RABIA
    - has some advantages

- Claimant need not be aware of Registrar when initiating a claim.
Operation with Registrars

(1) CABI₁: DISCOVER state
(2) (unicast) OFFER RABIA₁
(3) (unicast) RABIA₁: REGISTERED state
if Registrar expected at specific address

(1) ABI₀: DISCOVER state [to DA]

DA could be “Nearest Customer Bridge” address
01-80-C2-00-00-00 (non-forwarding)

e.g. for WLAN

(2) (unicast) RABIA₁: OFFERED state

Following a claim to a specific destination, a lack of response is a failure; need to follow with a claim to a CABI.
BARC Design

• A BARC architecture follows, with details including state machines.
  - additional details in Appendix

• Listened carefully to comments and discussion raised in P802.1CQ/D0.5 TG Ballot

• BARC (Block Address Registration and Claiming) is put into the broader context of Address Registration and Claiming (ARC), which supports both:
  - address blocks (ABs), identified by Address Block Identifiers (ABIs)
  - address ranges (ARs), excluding addresses specified by BARC

• ARC is the general protocol
  - BARC handles BI Registration and Claiming
  - existing MAAP handles AR Claiming
**ARC Architecture – ARC Claimant**

**MMRP**

**ARC Claimant Application**

**MAD**

**BI Claimant**

**AR Claimant**

**BI A state machine**

**BI B**

**BI C**

**BI n**

**BARCPDU Processor**

**BARCPDU(1, S1, I2, S2[, token])**

**Ingress(address, status)**

**Ingress MAC address filter**

**BARC(BI)**

**sBARC**

**BARCPDU(AR)**

**AVTPDU(AR)**

**Add(AR)**

**Drop(AR)**

**Outcome(result, AR)**

**Outcome(result)[BI]**

**FYI(status)[BI]**

**rMAAP(state, AR)\!**

**sMAAP(event)\!**

**ingress MAC address filter accepts**

- unicast addresses as adopted for use
- BI of an active CABI State Machine
- MAAP multicast address for AVTPDUs

**Ingress MAC address filter**

**state machine**
### BARCPDU Summary

<table>
<thead>
<tr>
<th>field name</th>
<th>purpose</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA</td>
<td>dest address</td>
<td>DA</td>
</tr>
<tr>
<td>SA</td>
<td>source address</td>
<td>(TUA allowed if S=D or A)</td>
</tr>
<tr>
<td>E</td>
<td>Ethertype</td>
<td>[tbd; could be 22F0 (MAAP Ethertype)]</td>
</tr>
<tr>
<td>t</td>
<td>subtype</td>
<td>[tbd, per 1722 WG; see IEEE 1722 Table 6]</td>
</tr>
<tr>
<td>S1</td>
<td>State</td>
<td>D, C, V, R, T, MD, MC, N(null)</td>
</tr>
<tr>
<td>I1</td>
<td>identifier</td>
<td>(an address)</td>
</tr>
<tr>
<td>S2</td>
<td>State</td>
<td>O, S, N(null)</td>
</tr>
<tr>
<td>I2</td>
<td>identifier</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>token</td>
<td></td>
</tr>
</tbody>
</table>

### AVTPDU Summary

<table>
<thead>
<tr>
<th>field name</th>
<th>purpose</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA</td>
<td>dest address</td>
<td>91:E0:F0:00:FF:00 for MAAP multicast</td>
</tr>
<tr>
<td>E</td>
<td>Ethertype</td>
<td>22F0 (MAAP Ethertype)</td>
</tr>
<tr>
<td>t</td>
<td>subtype</td>
<td>FE per IEEE 1722 Table 6</td>
</tr>
<tr>
<td>Event</td>
<td>VACANT (V)</td>
<td>DISCOVERY (D)</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>TryT(da,sa)!</td>
<td>sBARC(BI:T,sa:S)(da,sa)</td>
<td>Start DiscoverT_Timer</td>
</tr>
<tr>
<td>DiscoverT_Timer!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BARC(O)!</td>
<td>FYI(Offer)</td>
<td>StartOffer_Timer</td>
</tr>
<tr>
<td>Offer_Timer!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accept(sa,token)!</td>
<td></td>
<td>Ingress(sa,pass)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sBARC(BI:R,sa:S,token)(BI,sa)</td>
</tr>
<tr>
<td>BARC(R,sa,token)!</td>
<td></td>
<td>if sa= R_sa and token= R_token then</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start Renew_Timer(R)</td>
</tr>
<tr>
<td>BARC(V,sa,token)!</td>
<td></td>
<td>if sa= R_sa and token= R_token then</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ingress(sa,filter)</td>
</tr>
<tr>
<td>TryC(sa)!</td>
<td>sBARC(BI:D,sa:S)(BI,sa)</td>
<td>Start DiscoverC_Timer</td>
</tr>
<tr>
<td>DiscoverC_Timer!</td>
<td></td>
<td>Ingress(BI,pass)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BARC(C)!</td>
<td></td>
<td>Outcome(V)</td>
</tr>
<tr>
<td>BARC(D,SA)!</td>
<td></td>
<td>Outcome(V)</td>
</tr>
<tr>
<td>Delete()!</td>
<td></td>
<td>VACANT</td>
</tr>
<tr>
<td>Renew_Timer! or Renew()!</td>
<td></td>
<td>FYI(Renewed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BI Claimant/Registrar Procedure: Multicast

- BI Claimant begins without knowledge of Registrar
- CABI State goes from C to V (Vacant) (state machine destroyed)
- Even after CABI state machine destroyed, further offers can arrive; they create new RABIA state machines.

BI Claimant/Registrar Procedure: Targeted

- Model BI is not intended for claiming, only to indicate traits, such as AB size.
- State indicated as T (targeted)
- new RABIA state machine in state O (Offered)
Registration, Renewal, and Withdrawal of an Offer

BI Claimant

- **RABIA State** goes from O to R
  - adopt AB per RABI of RABIA
- IRA selected from RABI
  - IRA and token T stored in RABIA State
  - start listening to IRA unicast address

**Decision to register claim to RABIA**

- RABIA:O
  - IRA
  - T

**Decision to renew claim to RABIA**

- repeated claim (e.g. to renew)
- RABIA:O
  - IRA
  - T

**Claimant-initiated withdrawal**

- RABIA:O
  - RABIA:V
  - IRA
  - T

**Registrar-initiated revocation**

- RABIA:O
  - RABIA:V
  - IRA
  - T

**Decision to withdraw claim to RABIA**

- RABIA:O
  - RABIA:V
  - IRA
  - T

- RABIA:O
  - RABIA:V
  - IRA
  - T

**Decision to revoke claim to RABIA**

- IRA & token stored in RABIA State

**RABI Registrar**

- **RABIA State** goes from O to R
  - IRA
  - T

**RABIA R State**

- Renew timer reset
- IRA:S
- T

**RABIA State**

- goes from R to V
- IRA:S
- T

**Note 1**: A PDU to the RABI Registrar using a different RABIA would not be processed. The original ABU Claimant knows the RABIA in use; other claimants attempting to interfere (intentionally or accidentally) with the RABI will be inhibited if they lack of knowledge.

**Note 2**: Even if the DA holds the correct RABIA, the PDU will be rejected if the enclosed token and IRA (and perhaps SA) do not match those in the RABIA State Machine. This again interferes with counterfeit PDUs.

**Note 3**: IRA, token, and RABIA are included only in unicast messages, limiting their distribution in the network.

- IRA
- T

**Note**: PDU rejected unless RABIA, IRA, and token match the RABIA State Machine. PDU not delivered if IRA is incorrect.
BARC Registrar: AVTPDU Processor

AVTPDU (maap_version, message_type, AR) at MAAP DA from SA

maap_version → 2

message_type → 1

MAAP_Disc(AR,SA) → END

not 2

not 1
AR Claimant Procedure

AR Claimant

AR State goes to D (Discovery)

AR State V (Vacant)

AR Claimant goes to D (Discovery)

AR Claimant works exactly like MAAP in a group of mixed AR and MAAP Claimants.

AR:V

MAAP DA

uni AVTPDU Probe/v2

AR

AVTPDU Defend

Existing AR Claimant either MAAP or BARC

AR Claimant and (legacy) MAAP Claimant respond identically to AVTPDU Probe/v2.

MAAP AVTPDUs are received by BARC Registrar. If the AVTPDU is a MAAP Probe/v2, then the Registrar responds just as it does to a Targeted Claim.

AR:V

AR State V (Vacant)

AR State goes to V (Vacant)

AR:V

AR State goes to V (Vacant)

new RABIA state machine in state O (Offered)

AR State remains as D

Device can optionally register the Offer via BARC, while the AR may independently be claimed if not defended by a MAAP Claimant.

AR:D

AVTPDU Probe/v2

AR

RABIA:O

RABIA Registrar

select RABI with traits (such as size) of AR;
select a RABIA identifying that RABI

RABIA State goes to O

start listening to RABIA

uni

RABIA

0:N

RABIA:O
## AR State Transition Table

<table>
<thead>
<tr>
<th>Event</th>
<th>State</th>
<th>State</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VACANT (V)</td>
<td>DISCOVERY (D)</td>
<td>ACQUIRED (A)</td>
</tr>
<tr>
<td>Add(AR)!</td>
<td>sMAAP(Begin(AR)!)</td>
<td>DISCOVERY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DISCOVERY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rMAAP(AR:Defend)!</td>
<td></td>
<td>Outcome(A,AR)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACQUIRED</td>
<td></td>
</tr>
<tr>
<td>rMAAP(AR:Initial)!</td>
<td></td>
<td>Outcome(F,AR)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VACANT</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Outcome(X)[AR]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>VACANT</td>
</tr>
</tbody>
</table>

rMAAP(AR:State!) invokes an event at the state machine when the MAAP state changes to State

sMAAP(Action!) invokes Action! event at MAAP state machine
Ingress MAC address filter accepts BARCPDUs addressed to:
- RABIA of an active RABIA State Machine
- MBI of an active MBI State Machine
- any CABI and MAAP multicast address for AVTPDUs
## RABIA Registrar: RABIA State Transition Table

<table>
<thead>
<tr>
<th>Event</th>
<th>VACANT (V)</th>
<th>OFFERED (I)</th>
<th>REGISTERED (R)</th>
<th>EXPIRED (E)</th>
</tr>
</thead>
</table>
| `InviteD(CABI,RABIA,da)!
  sBARC(CABI:C,RABIA:O)(da,RABIA)` | `Ingress(RABIA,pass)
  sBARC(CABI:C,RABIA:O)(da,RABIA)`
  Start Offer_Timer
  OFFERED | | `Ingress(RABIA,pass)
  sBARC(CABI:C,RABIA:O)(da,RABIA)`
  Start Offer_Timer
  OFFERED | |
| `InviteT(S1,RABIA,da)!
  sBARC(0:S1,RABIA:O)(da,RABIA)` | `Ingress(RABIA,pass)
  sBARC(0:S1,RABIA:O)(da,RABIA)`
  Start Offer_Timer
  OFFERED | | `Ingress(RABIA,pass)
  sBARC(0:S1,RABIA:O)(da,RABIA)`
  Start Offer_Timer
  OFFERED | |
| `BARC(R,sa,token)!
  sBARC(RABIA:R, R_sa:S, token)(R_sa,RABIA)` | `R_sa==sa
  R_token==token
  Start Register_Timer(R)
  sBARC(RABIA:R, R_sa:S, token)(R_sa,RABIA)`
  REGISTERED | | if sa= R_sa and token= R_token then
  Start Register_Timer(R) | |
| `Offer_Timer!` | `Ingress(RABIA,filter)
  Start Expire_Timer
  EXPIRED` | | | |
| `Register_Timer!` | | | `sBARC(RABIA:V, R_sa:S,token)(R_sa,RABIA)`
  Ingress(RABIA,filter)
  Start Expire_Timer
  EXPIRED | |
| `BARC(V,sa,da)!
  if sa= R_sa and token= R_token then
  Ingress(sa,filter)
  Start Expire_Timer
  EXPIRED` | | | | |
| `Expire_Timer!` | | | `sBARC(RABIA:V, R_sa:S,token)(R_sa,RABIA)`
  Ingress(RABIA,filter)
  Start Expire_Timer
  EXPIRED | `VACANT` |
An MBI Registrant could be specified. Alternatively, a Primary MBI Claimant could be configured to hold and defend many MBIs in reserve, ensuring that MBIs are not excessively granted. This would suffice in many cases.

If an MBI Registrant is to be specified, following the design of the ABI Registrant, an MBI identifier, parallel to the RABIA, could be specified by taking a bit from the $abc$ bits; for example, turning the $c$ bit into a designation of the MBI A and limiting the RABI to four types, using the $ab$ bits. However, an MBI Registrant does not require an MBI A identifier.
<table>
<thead>
<tr>
<th>Event</th>
<th>VACANT (MV)</th>
<th>DISCOVERY (MD)</th>
<th>CLAIMED (MC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Try(sa)!</td>
<td>sBARC(MBI:MD,sa:S)(MBI,sa)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start MDiscoverTimer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DISCOVERY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDiscoverTimer!</td>
<td></td>
<td>ingress(MBI,pass)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outcome(MC)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sBARC(MBI:MC,0:N){MBI}</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start Renew_Timer</td>
<td>CLAIMED</td>
</tr>
<tr>
<td>BARC(MC)!</td>
<td></td>
<td>Outcome(MV)</td>
<td>sBARC(MBI:MC,0:N){MBI}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VACANT</td>
<td>FYI(Alert)</td>
</tr>
<tr>
<td>BARC(MD,SA)!</td>
<td></td>
<td>Outcome(MV)</td>
<td>da=SA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VACANT</td>
<td>sBARC(MBI:MC,0:N){da}</td>
</tr>
<tr>
<td>Delete()!</td>
<td></td>
<td>VACANT</td>
<td></td>
</tr>
<tr>
<td>Renew_Timer! or Renew()!</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sBARC(MBI:MC,0:N){MBI}</td>
<td>FYI(Renewed)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Start Renew_Timer</td>
</tr>
</tbody>
</table>
VLANs

• All state machines are specified per VLAN.

• All address assignments are specific to the VLAN in which the state machine operates.

• All addresses adopted are specific to the VLAN under which the assignment was completed.

• Usage of any address is limited to the VLAN under which it was obtained.

• Any address assigned within the context of a VLAN shall not be reassigned except within the context of the VLAN in which it was assigned.

• Due to the possibility that the same unicast address may be assigned in different VLANs, Independent VLAN Learning (IVL) is required in bridges, per IEEE Std 802.1Q Annex F (F.1.2).
  - This requirement could be relaxed when assigned unicast addresses are declared via MMRP.
Summary

• Claimants operate with or without Registrars.

• Multiple registrars are supported, operating with disjoint multi-blocks.

• The block discretization provides:
  – a vast set of addresses to a LAN
    – though operating entirely within 1/16 of the SAI quadrant of local address space
  – a large set of temporary unicast addresses
  – operational efficiency and simplicity
  – both unicast and multicast addresses (1/2 or 2/3 unicast) to Claimant
    – including one unicast and multicast subblock with the same range, except for the M bit
      – could be exploited
    – devices needing both unicast and multicast addresses need make only one claim

• Could integrate with MMRP to limit propagation and eliminate learning of unicast AB content.
  – MMRP needs to efficiently handle address ranges
  – BARP could be specified as alternative MRP application
Appendix 1

• additional details on BARC addresses and identifiers
MAC Address Parsing

- **N11**: 0
- **N10**: 1 1 1 M
- **N9**: p r j k
- **N8**: n a b c
- **N7**:
- **N6**:
- **N5**:
- **N4**: N1
- **N0**: N2

**MAC address**

- **N11**
  - 0: not specified by BARC
  - 111: non-zero

**BARC address**

- **M**: not specified by BARC

**ABI**

- **jk**
  - 0: not specified by BARC
  - 1: ABI is CABI of type , Claimable ABI (not within an MBI) with 2*16^k ICAs & 16^k GCAs
  - r 1: ABI is RABI of type , Registrable ABI (within an MBI of type /abc) with 2*16^k IRAs & 2*16^k GRAs

**RABIA**

- **n**
  - 0: (unicast) RABIA BI
  - 1: TUA (temporary unicast address)

**ICA**

- **ICA0**, **ICA1**
  - 0: not specified by BARC
  - 1: BARC address is unicast or multicast

**GRA**

- **GRA**
  - 0: RA within RAB
  - 1: RA within an MBI of type abc (n=0 or 1)

**GCA**

- **GCA1**
  - 0: not specified by BARC
  - 1: BARC address is unicast or multicast

**RABIA**

- **RA**
  - 0: Identified by type , RABI within an MBI
  - 1: Identified by type , CAB within CAB

**Other**

- **lsn**
  - 0: Identified by type , BARC address is unicast or multicast
  - 1: BARC address identified by type , CAB within CAB

*lsn= least significant nibbles

**RABI**

- **n**
  - 0: other
  - 1: other

**MBI**

- **ABC**
  - 0: other
  - 1: other

**CA**

- **CA**
  - 0: Identified by type , CABI within CAB
  - 1: Identified by type , CAB within CAB

**RA**

- **RA**
  - 0: Identified by type , RABI within an MBI
  - 1: Identified by type , RABI within an MBI
<table>
<thead>
<tr>
<th></th>
<th>CABI</th>
<th>CA1</th>
<th>CA0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N10</td>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
<td>1 1 1 0</td>
</tr>
<tr>
<td>N9</td>
<td>0 0 j k</td>
<td>1 0 j k</td>
<td>0 0 j k</td>
</tr>
<tr>
<td>N8</td>
<td>X8</td>
<td>X8</td>
<td>X8</td>
</tr>
<tr>
<td>N7</td>
<td>X7</td>
<td>X7</td>
<td>X7</td>
</tr>
<tr>
<td>N6</td>
<td>X6</td>
<td>X6</td>
<td>X6</td>
</tr>
<tr>
<td>N5</td>
<td>X5</td>
<td>X5</td>
<td>X5</td>
</tr>
<tr>
<td>N4</td>
<td>X4</td>
<td>X4</td>
<td>X4</td>
</tr>
<tr>
<td>N3</td>
<td>X3</td>
<td>X3</td>
<td>X3</td>
</tr>
<tr>
<td>N2</td>
<td>X2/0</td>
<td>X2/*</td>
<td>X2/*</td>
</tr>
<tr>
<td>N1</td>
<td>X1/0</td>
<td>X1/*</td>
<td>X1/*</td>
</tr>
<tr>
<td>N0</td>
<td>X0/0</td>
<td>X0/*</td>
<td>X0/*</td>
</tr>
</tbody>
</table>

**CABI and CA**

**MBI, RABI, RA, RABIA**

MBI identifies its RABIs

<table>
<thead>
<tr>
<th></th>
<th>MBI</th>
<th>RABI</th>
<th>RA</th>
<th>RABIA</th>
</tr>
</thead>
<tbody>
<tr>
<td>N11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>N10</td>
<td>1 1 1 1</td>
<td>1 1 1 1</td>
<td>1 1 1 0</td>
<td></td>
</tr>
<tr>
<td>N9</td>
<td>0 1 j k</td>
<td>1 1 j k</td>
<td>0 1 j k</td>
<td></td>
</tr>
<tr>
<td>N8</td>
<td>X7</td>
<td>X7</td>
<td>X7</td>
<td></td>
</tr>
<tr>
<td>N7</td>
<td>X6</td>
<td>X6</td>
<td>X6</td>
<td></td>
</tr>
<tr>
<td>N6</td>
<td>X5</td>
<td>X5</td>
<td>X5</td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td>X4</td>
<td>X4</td>
<td>X4</td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>X3</td>
<td>X3</td>
<td>X3</td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>X2/0</td>
<td>X2/*</td>
<td>X2/*</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>X1/0</td>
<td>X1/*</td>
<td>X1/*</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>X0/0</td>
<td>X0/*</td>
<td>X0/*</td>
<td></td>
</tr>
</tbody>
</table>

*abc* nibbles for MBI type *jk/abc*

 progresses

* indicates all possible values

jidk nibbles for ABI type *jk*

 progresses

* indicates all possible values

*abc=8 when a=b=c=0

# indicates all possible values
### Four CABI Types

<table>
<thead>
<tr>
<th>CABI Type 0</th>
<th>CABI Type 1</th>
<th>CABI Type 2</th>
<th>CABI Type 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CABI</td>
<td>CA1</td>
<td>CA0</td>
<td>CABI</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
</tr>
<tr>
<td>0000</td>
<td>1000</td>
<td>0000</td>
<td>0001</td>
</tr>
<tr>
<td>X8</td>
<td>X8</td>
<td>X8</td>
<td>X8</td>
</tr>
<tr>
<td>X7</td>
<td>X7</td>
<td>X7</td>
<td>X7</td>
</tr>
<tr>
<td>X6</td>
<td>X6</td>
<td>X6</td>
<td>X6</td>
</tr>
<tr>
<td>X5</td>
<td>X5</td>
<td>X5</td>
<td>X5</td>
</tr>
<tr>
<td>X4</td>
<td>X4</td>
<td>X4</td>
<td>X4</td>
</tr>
<tr>
<td>X3</td>
<td>X3</td>
<td>X3</td>
<td>X3</td>
</tr>
<tr>
<td>X2</td>
<td>X2</td>
<td>X2</td>
<td>X2</td>
</tr>
<tr>
<td>X1</td>
<td>X1</td>
<td>X1</td>
<td>X1</td>
</tr>
<tr>
<td>X0</td>
<td>X0</td>
<td>X0</td>
<td>X0</td>
</tr>
</tbody>
</table>

- 3 contiguous subblocks per CABI (CA0 and CA1 unicast, CA1 multicast)

- 6.9E10 Type 0 CABIs
- 1 address per subblock
- 3 addresses/CABI

- 4.3E9 Type 1 CABIs
- 16 addresses per subblock
- 48 addresses/CABI

- 2.7E8 Type 2 CABIs
- 256 addresses per subblock
- 768 addresses/CABI

- 1.6E7 Type 3 CABIs
- 4096 addresses per subblock
- 12288 addresses/CABI

---

31
## Multi-Blocks and Multi-Block Identifiers (MBIs)

### AB type 0
- **AB size**: 2

<table>
<thead>
<tr>
<th>MBI type</th>
<th>X7</th>
<th>X6</th>
<th>X5</th>
<th>X4</th>
<th>X3</th>
<th>X2</th>
<th>X1</th>
<th>X0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>X0</td>
</tr>
<tr>
<td>0 2</td>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>X0</td>
</tr>
<tr>
<td>0 3</td>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>X0</td>
</tr>
<tr>
<td>0 4</td>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>X0</td>
</tr>
<tr>
<td>0 5</td>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>X0</td>
</tr>
<tr>
<td>0 6</td>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>X0</td>
</tr>
<tr>
<td>0 7</td>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>X0</td>
</tr>
<tr>
<td>0 8</td>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
<td>X3</td>
<td>X2</td>
<td>X1</td>
<td>X0</td>
</tr>
</tbody>
</table>

### MBI size
- **16**, **256**, **4096**, **65k**, **1M**, **17M**, **268M**, **4.3E9**

### MBI count
- **268M**, **17M**, **1M**, **65k**, **4096**, **4.3E9**, **8.6E9**, **8.6E9**

### AB count
- **4.3E9**, **4.3E9**, **4.3E9**, **4.3E9**, **4.3E9**, **4.3E9**, **4.3E9**, **4.3E9**

### addresses*
- **8.6E9**, **8.6E9**, **8.6E9**, **8.6E9**, **8.6E9**, **8.6E9**, **8.6E9**, **8.6E9**

---

*total for each case: unicast and multicast*
### Multi-Blocks and Multi-Block Identifiers (MBIs)

<table>
<thead>
<tr>
<th>MBI count</th>
<th>4096</th>
<th>256</th>
<th>65k</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBI size</td>
<td>65k</td>
<td>1M</td>
<td>17M</td>
</tr>
<tr>
<td>AB count</td>
<td>16</td>
<td>1M</td>
<td>1M</td>
</tr>
<tr>
<td>addresses</td>
<td>268M</td>
<td>268M</td>
<td>268M</td>
</tr>
</tbody>
</table>

**AB type 1**
AB size=2\(\cdot16\)

- **nabc**
- **X7**
- **X6**
- **X5**
- **X4**
- **X3**
- **X2**
- **X1**
- **0**

**MBI type**

- **11**
- **12**
- **13**
- **14**
- **15**
- **16**
- **17**
- **18**

```
<table>
<thead>
<tr>
<th>MBI type</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>X7</td>
<td>X7</td>
<td>X7</td>
<td>X7</td>
<td>X7</td>
<td>X7</td>
<td>X7</td>
<td>X7</td>
<td>0</td>
</tr>
<tr>
<td>X6</td>
<td>X6</td>
<td>X6</td>
<td>X6</td>
<td>X6</td>
<td>X6</td>
<td>X6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X5</td>
<td>X5</td>
<td>X5</td>
<td>X5</td>
<td>X5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X4</td>
<td>X4</td>
<td>X4</td>
<td>X4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X3</td>
<td>X3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

*total for each case: unicast and multicast*
### Multi-Blocks and Multi-Block Identifiers (MBIs)

<table>
<thead>
<tr>
<th>nabc</th>
<th>MBI type</th>
<th>MBI count</th>
<th>MBI size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>16</td>
<td>1M</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>256</td>
<td>65k</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>4096</td>
<td>1M</td>
</tr>
<tr>
<td>111</td>
<td></td>
<td>4096</td>
<td>17M</td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td>4096</td>
<td>17M</td>
</tr>
</tbody>
</table>

**AB type 2**

<table>
<thead>
<tr>
<th>AB size=2•256</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>4096</td>
</tr>
<tr>
<td>65k</td>
</tr>
<tr>
<td>1M</td>
</tr>
<tr>
<td>17M</td>
</tr>
</tbody>
</table>

*applies to both unicast and multicast subblocks

could specify more combinations

*total for each case: unicast and multicast
### Multi-Blocks and Multi-Block Identifiers (MBIs)

<table>
<thead>
<tr>
<th>MBI type</th>
<th>MBI count</th>
<th>MBI size</th>
<th>AB count</th>
<th>addresses*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4096</td>
<td>16</td>
<td>1M</td>
<td>8.6E9</td>
</tr>
<tr>
<td></td>
<td>65k</td>
<td>256</td>
<td>1M</td>
<td>8.6E9</td>
</tr>
<tr>
<td></td>
<td>65k</td>
<td>4096</td>
<td>1M</td>
<td>8.6E9</td>
</tr>
<tr>
<td></td>
<td>1M</td>
<td>65k</td>
<td>1M</td>
<td>8.6E9</td>
</tr>
<tr>
<td></td>
<td>1M</td>
<td>1M</td>
<td>1M</td>
<td>8.6E9</td>
</tr>
</tbody>
</table>

**AB type 3**

- **AB size=2•4096**

*applies to both unicast and multicast subblocks

---

**could specify more combinations**

<table>
<thead>
<tr>
<th>n</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
</tr>
<tr>
<td>X7</td>
<td>X6</td>
<td>X5</td>
<td>X4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

*applies to both unicast and multicast subblocks*
MBI Example

<table>
<thead>
<tr>
<th>0</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0111</td>
</tr>
<tr>
<td>0</td>
<td>1010</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

example MBI

256 RABIs

MBI short name: 32/0x802 (32=MBI Type)

# indicates all possible values

RABI Example

<table>
<thead>
<tr>
<th>0</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0111</td>
</tr>
<tr>
<td>0</td>
<td>0010</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

particular RABI

2•4096 IRAs in 32/0x80201

RABI short name: 32/0x80201 (32=MBI Type)

* indicates all possible values

RA, RABIA Example

<table>
<thead>
<tr>
<th>0</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1110</td>
</tr>
<tr>
<td>0</td>
<td>0100</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

2•4096 GRAs in 32/0x80201

4096 RABIAs in 32/0x80201

MBI short name: 32/0x802 (32=MBI Type)

# indicates all possible values

2 nibbles for MBI type abc=2

3 nibbles per ABI type jk=3
null CABI: identifies no AB

- Could be used when initiating discovery to expected Registrar.
- Conveys to Registrar only the size of the requested AB.
- No other Claimant listens to this address.
Non-AB Temporary Unicast Address (TUA)

- For temporary use

- device without a source address selects a random non-AB temporary unicast address for initial discovery only
  - protocol then assigns at least one persistent unicast address

- simultaneous duplicate temporary addresses may lead to message loss in some circumstances
  - network learns route to source as initial message crosses the network
  - before response is returned, another initial message with duplicate source address crosses the path and rewrites the route
  - unlikely to be disastrous
  - loss of initial message will be corrected eventually

- nevertheless, need to consider the likelihood of duplication

- Temporary address range includes 8 full nibbles of 16 values each (0–F)
  - $16^9 = 68,719,476,736$ ($= N$) temporary addresses in the pool
  - chance of no duplicates with $k$ randomly selected addresses is approximated as $\exp(-k*(k-1)/(2*N))$
  - with $k=1000$ devices simultaneously using a temporary address, chance of no duplicates is ~0.99988
  - address conflicts are rare, usually not harmful, and recoverable
  - can add first 2 bits of the N9 nibble and first 3 bits of the N8 nibble to the pool
    chance of no duplicates is then ~0.999996 ($k=1000$)
• additional procedural details
**ARC Claimant: BARCPDU Processor – ingress**

1. **BARCPDU** (I1,S1, I2,S2 [,token])
2. If I1 is active and S1\(\neq N\) then BARC(S1)! to I1
3. END

**ARC Claimant: BARCPDU Processor – egress**

1. **sBARC** (i1:s1, i2:s2 [,token]) {da[,sa]}
2. Send BARCPDU with I1=i1,S1=s1, I2=i2,S2=s2 [,T=token] to da from sa
3. If sa is null, use default
4. Could repeat, with repetition count possibly based on S1 and S2
5. END
ARC Claimant Application Process: Drop Claim

START:
- Initiate Dropping

AB or AR

AB

BI Claimant Delete([BI])

END

AR

AR Claimant Drop([AR])
BARC Address Propagation with MMRP

The ARC Claimant Application Process includes “declare with MMRP”. This entails declaring, to MMRP (when available), MMRP attributes, using an MMRPDU per IEEE Std 802.1Q § 10.12.1.6:

- The multicast address represented by the ABI
  
  FirstValue field = ABI/NumberOfValues=1

- The two unicast address set subblocks indicated by the ABI (CABI or RABI)
  
  FirstValue field = first ABI in unicast subblock/NumberOfValues = 16^k per jk in nibble N9 of ABI
  
  - maximum NumberOfValues is with jk=3; 16^3=4096; MRP provides 13 bits of NumberOfValues (2^{13}=8192)

The ARC Claimant Application Process includes (“select CABI”); this selection should consider any local MMRP registration database to avoid selecting a registered CABI.

Unicast MMRP declaration can be useful because:

1. A one-step declaration covers a contiguous range of self-assigned unicast addresses.
2. Eliminates flooding for all the unicast addresses in the assignment.
3. Eliminates the need for learning of each unicast address when used.
4. Precludes erroneous re-learning of an address when a false duplicate is used elsewhere in the network.
   - Could be a way to control duplication.
   - Security issues to study.

BARC could alternatively specify “BARP,” a new MRP application. This could entail the following changes:

(a) the BARP application would be enabled to Join and Leave with the ABI as the declared attribute
(b) the BARP application would be specified to understand the semantics of the ABI and extract from it the indicated ABI multicast address and the indicated unicast address set, then use it to populate the FDB
(c) In the BARC BI State Machine, the ABI claim [“sBARC(ABI:C)”] might not be needed, since the a BARP declaration could convey the claim to the ABI as well as the declaration of interest in receiving at the ABI multicast address

BARP might be better suited to specification within IEEE Std 802.1Q instead of 802.1CQ.
ARC Claimant Application Process: BARC Management

START: Receive BARC FYI(condition)

- Condition Alert
- Offer no
- Renewed action to add AB?
- yes go to: Initiate Adding
- no

- Renewed Offer
- action to resolve duplicate
  - renew
  - delete

- Revoked BARC Delete()

- BARC Renew() go to: Update MMRP

- Barc Renew() go to: Replace AB?
  - yes go to: Initiate Adding
  - no go to: Adding

END
ARC Claimant Application Process: MAAP Management

START: Receive Outcome(X) [AR]

replacement AB or AR?

no

END

yes
go to: Initiate Adding
BARC Registrar: BARCPDU Processor – ingress

BARCPDU (I1,S1, I2,S2 [,token])

Disc(S1,I1,I2)

if I1 is active then
BARC(S1,I2,token)! to RABIA I1

BARC(S1)! to MBI I1

BARC(S1,I2)! to MBI I1

BARC(S1)! to MBI I1

BARC Registrar: BARCPDU Processor – egress

sBARC (i1:s1, i2:s2 [,token]) {da,[sa]}

send BARCPDU with I1=i1,S1=s1, I2=i2,S2=s2 [,T=token] to da from sa
if sa is null, use default

could repeat, with repetition count based on S1

END
BARC Registrar Application: Disc Processing

Disc(State,BI,SA) from BARCPDU Processor

select RABIA based on
• BI request (including AB size)
• BARC MBI Claimant States
• BARC RABI Registrar States
and values of Expire_Timer

InviteD(BI, RABIA,SA)!

BARC Registrar Application: MAAP_Disc Processing

MAAP_Disc (AR,SA) from RABI Registrar

select RABIA based on
• AR request (including AR size)
• BARC RABI Registrar States
• BARC MBI Claimant States

InviteT(N, RABIA,SA)!

END
Barc Registrar Application: MBI Claimant Management

START: Initiate Adding or Deleting

Adding

Deleting

BARC Delete(MBI)

try again?

yes

no

END

select MBI and da considering:
- BARC RABI Registrar States
- BARC MBI Claimant States
- most recent Outcome content

BARC Add(da)(MBI)

result of Outcome(result)(MBI)

C

U,V

declare MBI address claim using MMRP

END
START: Receive BARC FYI(condition)

condition

Alert

Renewed

action to resolve duplicate

renew

delete

BARC Renew([MBI])!

END

replacement MBI?

yes

go to: Initiate Adding

no