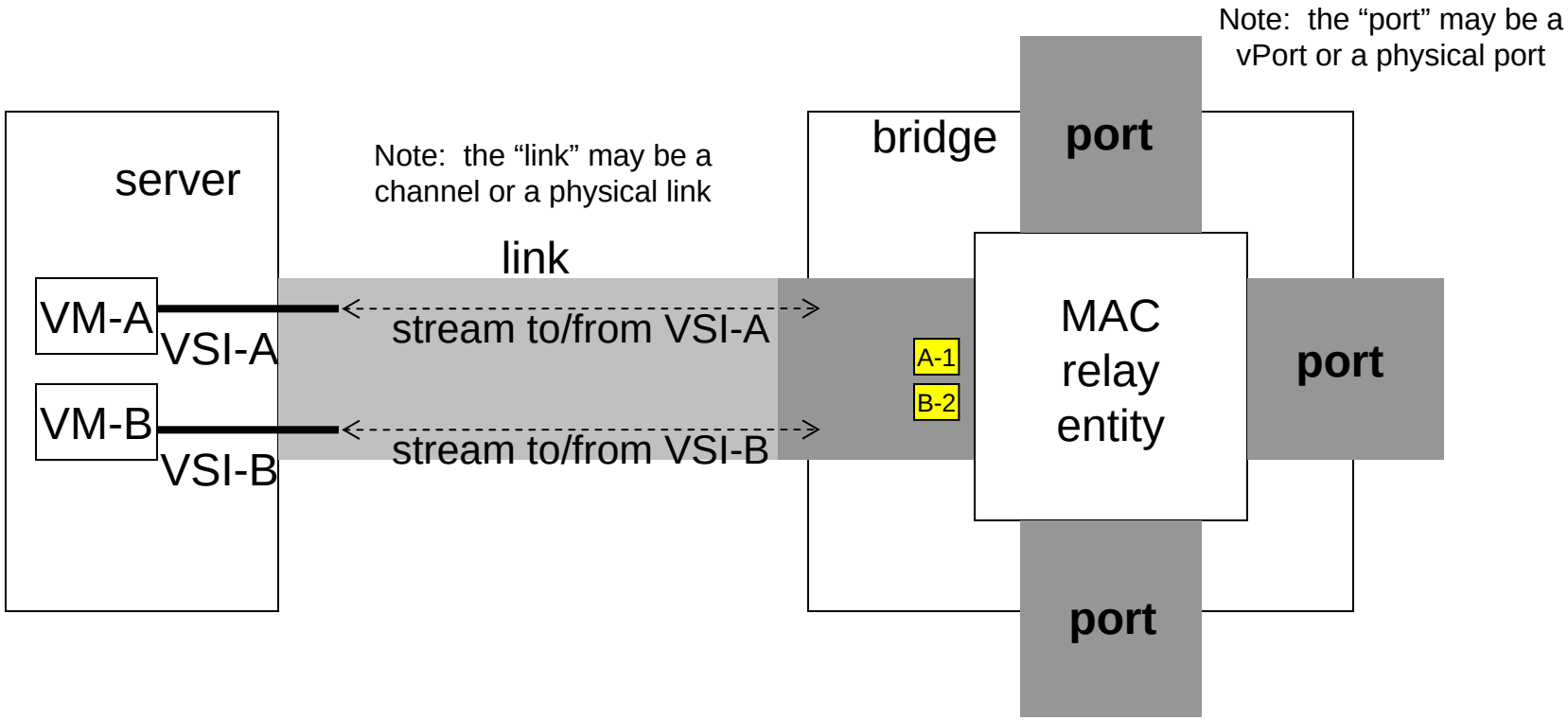
# Which protocol for VSI-to-Profile Binding?

Bob Sultan ( bsultan@huawei.com )

# VSI-to-Profile Binding

A-1 binding between VSI A and Profile with PID 1




- Binding between VSI and Profile maintained on Bridge by communicating VSI and Profile ID (PID) to Bridge;
- Bridge fetches Profile from database based on PID;

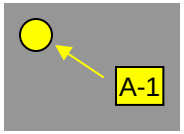
# Terminology Note

- Now that we use the term VSI to describe the VM port and the term vPort to describe the Bridge Port associated with a Channel, the term ‘Port Profile’ does not seem like a good choice as this profile is applied to a VSI. In these slides we simply use the term ‘Profile’;
- The abbreviation PID is used for the Profile ID;
- Note also that a Bridge Port can be a vPort or a ‘physical port’ and the link (or point-to-point LAN) between Server and Bridge can be a Channel or a Physical Link.

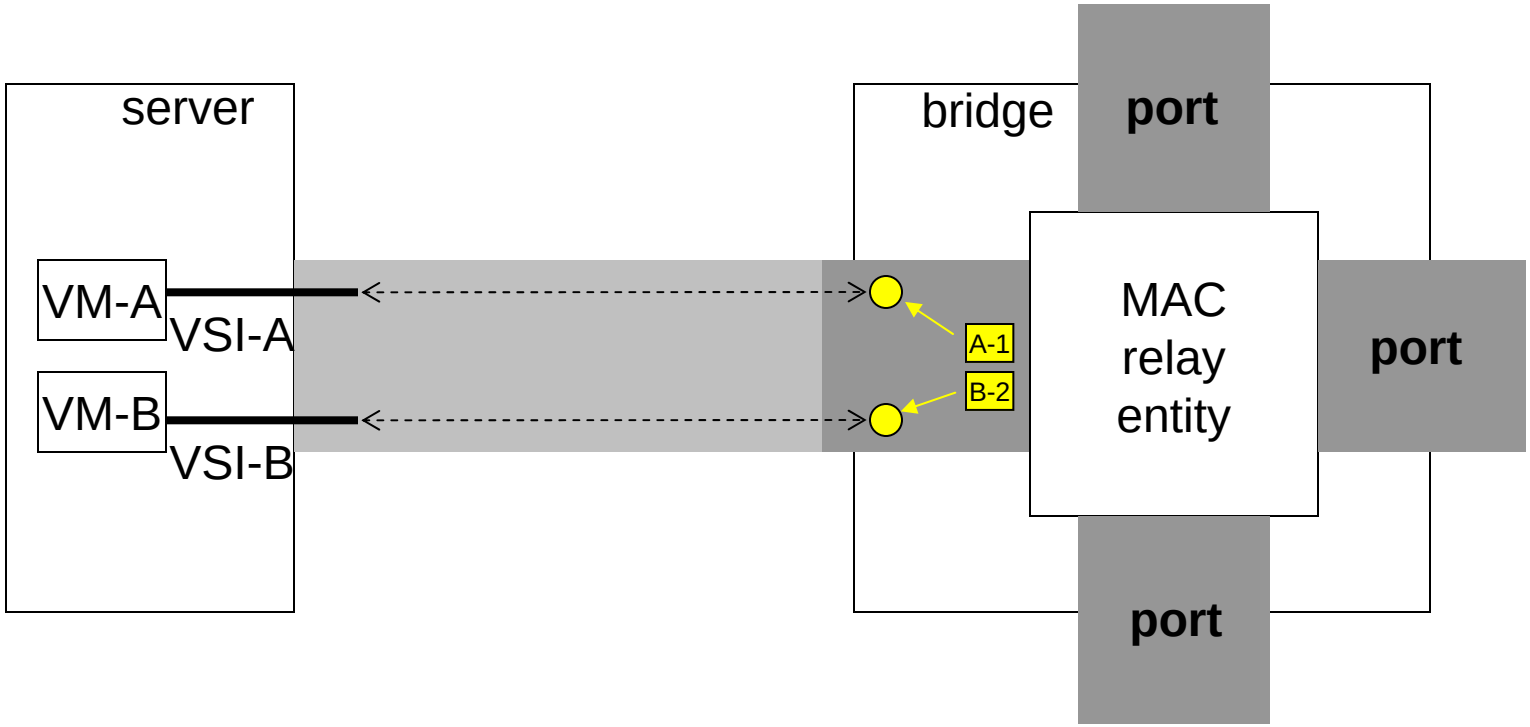
# Two ways to ‘apply’ Profile Attribute

- Apply  Attribute to the individual VSI traffic stream with which the Profile is associated;
  - If you don’t agree that this case exists, no problem, just ignore slides 6 and 11;
- Apply Attribute to the Bridge Port (Port or vPort) by ‘combining’ it with the corresponding attribute of all other Profiles associated with the Bridge Port.

# Alt 1:  ACL applied to VSI

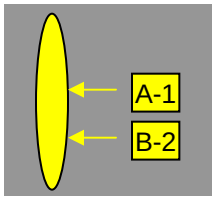



ACL from Profile with PID 1 is applied to traffic stream associated with VSI-A;

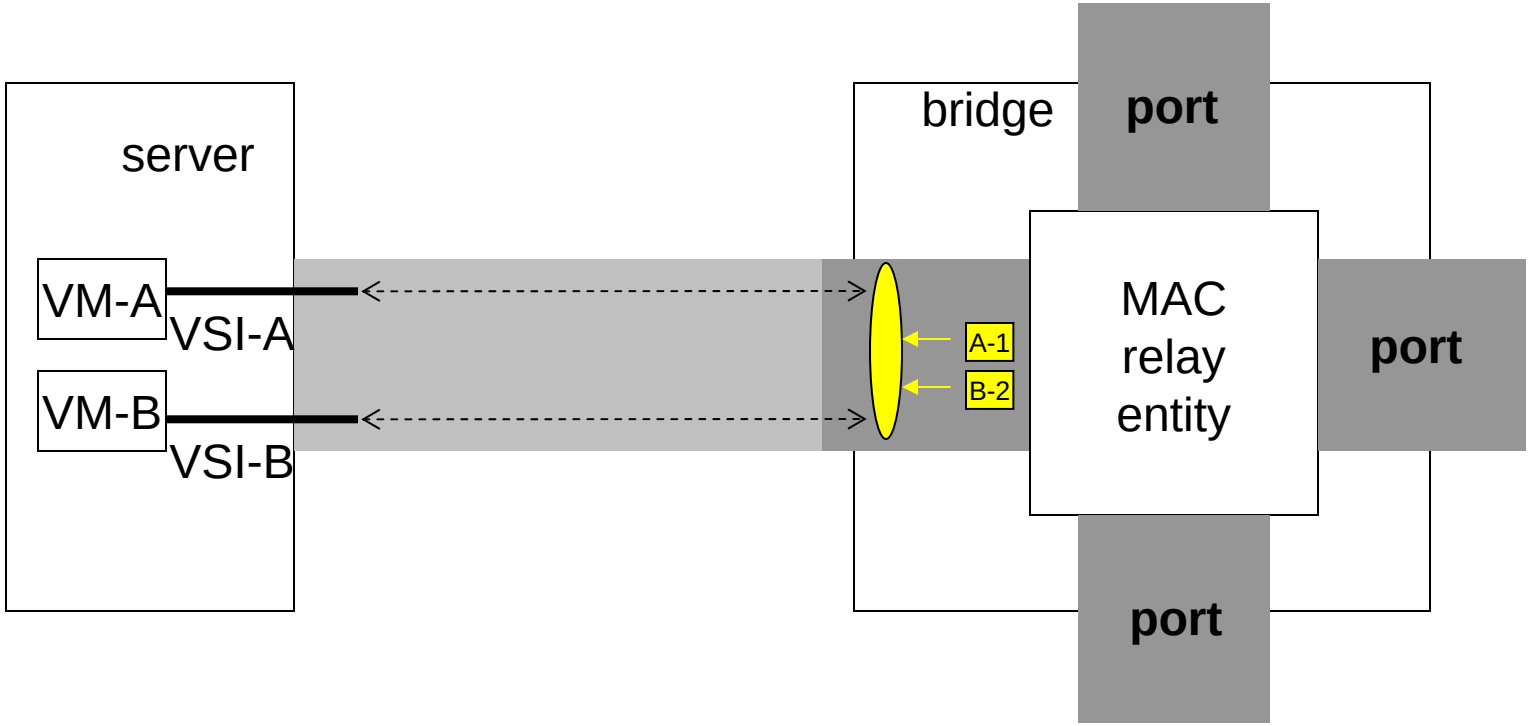



- ACL is applied to the specific VSI to which the Profile is bound.

# Alt 2: ACL applied to Bridge Port




ACLs associated with Profiles identified by PID 1 and PID 2 are ‘combined’ and the combined ACL is applied to all traffic on the Bridge Port;




- ACLs from all Profiles associated with Bridge Port are ‘combined’ and applied to the Bridge Port;

# What's the point?

- There are currently three proposals under consideration for the method of maintaining VSI-to-Profile bindings on the bridge:
  1. LLDP (Uri);
  2. LLDP-like with T3P (Chuck);
  3. Association Control Protocol (Bob);
- These proposals are difficult to compare because they address *different* requirements;
- To choose among these, it is important to establish the requirement that is to be met;
- As a convenience, let us group the first two proposals as "A" and we refer to the third proposal as "B"

# Requirements met by A

- Multiple changes in the binding database are committed in an *atomic* operation;
  - That is, if the currently committed binding list on Server and Bridge is {A, B, C, D, E} and some sequence of changes occurs so that the list becomes {A, C, E, F, G} then it is *not* permissible for the Server and Bridge to commit successive binding lists {A, B, C, D, E} → {A, C, D, E} → {A, C, E} → {A, C, E, F} → {A, C, E, F, G};
  - Only the atomic change {A, B, C, D, E} → {A, C, E, F, G} is allowed;
  - The claim has been made that this is necessary due to interdependencies among the bindings; i.e., the administrator has calculated the consequences of committing {A, B, C, D, E} and {A, C, E, F, G}, but not the intermediate lists;
- The order in which binding are specified in the binding set is significant;
  - For example, committing the binding set {A, B, C, D, E} produces a different result than {A, D, C, B, E}; i.e., the order of TLVs in the LLDPDU is significant;
- The binding set {A, B, C, D, E} is not committed if just one of the bindings cannot be committed;
  - The idea being that the bindings are interdependent; if you can't commit the entire new binding set, you retain the current binding set;

# Requirements met by B

- Bind establish/release requests are made individually by the server and are committed or rejected individually by the bridge;
  - There is no interdependence among the bindings;

# How do we choose?

- "A" meets a significantly more stringent set of requirements and implies greater complexity;
  - The complete database must be communicated every time there is a change in the binding set;
    - Overhead of packing, unpacking, and processing *every* element;
  - The elements of the binding set (i.e., TLVs) must be placed in the proper order
    - and the Server must know what that order should be;
  - The Server must understand how to revise a binding set when that binding set is denied by the bridge;
- "B" meets a *much* simpler requirement and implies lower complexity;
  - When a binding is to be added or removed, the server simply sends the information associated with that individual binding;
- So, it's clear that we need to know *which* is the minimum requirement to be met;
- Claim: *No* credible evidence has been presented that meeting requirement B is insufficient;
  - It was suggested at the Dec. 22 evb meeting that the use of ACLs requires A, but we will now describe why this is not the case;

# First Step In Proof

- It was described in slides 3 and 4 that there are two ways that a profile element, such as an ACL, can be applied when it is bound to a VSI:
  - Alt 1: the ACL can be applied to the specific VSI traffic stream with which the Profile has been bound;
  - Alt 2: the ACL can be 'combined' with the other ACLs associated with the Bridge Port and the resulting ACL applied to the Bridge Port.
- In the case of Alt. 1, the ACL is applied to a VSI traffic stream in the same way was it would be applied to a traffic stream associated with a Bridge port in a non-VM environment;
  - That is, today, there is no requirement to apply ACL1 to BridgePort1 and ACL2 to BridgePort2 *as an atomic operation*;
  - The same is true when an ACL is applied to a VSI traffic stream in a VM environment;
- Thus, there is clearly no requirement for an atomic operation when ACLs are utilized as described by Alt 1.

# Second Step In Proof

- It follows that we need only consider Alt. 2 where per-VSI ACLs are ‘combined’ into a single per-BridgePort ACL on the Bridge;
- Norm made the argument in the Dec. 22 meeting that
  - The order of entries in the BridgePort-ACL is significant (absolutely true) *and*
  - The order in which per-VSI ACLs are specified (i.e., order of TLVs in LLDPDU) is important in determining the order of entries in the BridgePort-ACL (*not* true);
- The claim *would* be true if each VSI binding identified a distinct ACL ***entry***,
  - but it doesn’t, it identifies a *complete* ACL (via the Profile ID), not an ACL entry;

# Second Step In Proof (con'd)

- Consider the following example of two VSI bindings associated with a Link:

**From the Profile with Profile ID 1 bound to VSI-A:**

access-list 104 permit tcp any 172.22.0.0 0.0.255.255 established
access-list 104 permit tcp any host 172.22.1.2 eq smtp
access-list 104 permit udp any any eq dns
access-list 104 deny icmp any any echo
access-list 104 deny icmp any any echo-reply

**From the Profile with Profile ID 2 bound to VSI-B:**

access-list 105 permit tcp any 172.22.0.0 0.0.255.255 established
access-list 105 deny tcp any host 172.22.1.2
access-list 105 permit udp any any eq dns
access-list 105 permit icmp any 172.22.0.0 0.0.255.255 echo
access-list 105 permit icmp any 172.22.0.0 0.0.255.255 echo-reply

- Let's say that the TLV for VSI-A precedes that for VSI-B in the LLDPDU

# Second Step In Proof (con'd)

- The VSI-bindings have been specified in the order {VSI-A, VSI-B} in the LLDPDU in order in order to ensure that the red entry from per-VSI ACL 104 will precede the red entry from the per-VSI ACL 105 in the combined BridgePort ACL 106:

  access-list 106 permit tcp any host 172.22.1.2 eq smtp
  access-list 106 deny tcp any host 172.22.1.2

- However, this has the nasty side effect that the green entries from per-VSI ACL 104 will precede the green entry from the per-VSI ACL 105 in the combined BridgePort ACL 106:

  access-list 106 deny icmp any any echo
  access-list 106 deny icmp any any echo-reply
  access-list 106 permit icmp any 172.22.0.0 0.0.255.255 echo
  access-list 106 permit icmp any 172.22.0.0 0.0.255.255 echo-reply

- So, it seems clear that **no** ordering of the *TLVs* is going to help you correctly order the entries in the 'combined' BridgePort ACL.

# Third Step In Proof

- So how *do* you guarantee that the entries of the BridgePort (i.e., ‘combined’) ACL will be in some sensible order?
- You *must* have an ‘ACL combiner’ on the Bridge with sufficient intelligence to know how to construct an ACL (eg., more specific entry preceding less specific entry) *and* to know when entries are in conflict and can’t be ‘combined’;
  - In fact, if you require VSI-binding to be an atomic operation, I’d claim that the liklihood of having to fail the entire set of bindings can be quite high;
  - It seems preferable to establish the bindings you can, and reject those you can’t (i.e., not an atomic operation);

# Fourth Step In Proof

- ‘Combining’ requires that a list of the active VSI-to-Profile bindings be maintained on the Bridge;
  - We *might* imagine a way to incrementally add a per-VSI ACL to the current BridgePort ACL without having the list of previously ‘combined’ per-VSI ACLs
  - But it is infeasible to *remove* per-VSI ACLs from the BridgePort ACL without having the complete list of active per-VSI ACLs;
- We have now established that Alt. 2
  - requires a ‘combiner’ on the Bridge
  - the ‘combiner’ must have access to the complete list of active VSI-to-Profile bindings associated with the BridgePort
  - the order of VSI-to-Profile bindings is not useful input to the ‘combiner’ and need not be maintained by the Bridge;

# Final Step In Proof

- The only remaining possibility to consider is a case in which binding lists {A, B, C} and {A, B, C, D, E} both produce desired results but the intermediate lists {A, B, C, D} and {A, B, C, E}, *while not rejected*, produce undesirable results;
  - This is the case that would require multiple binding changes to be committed in an atomic operation;
- We can show that this possibility cannot occur.

# Final Step In Proof (con’d)

- For {A, B, C, D} to be legal, it cannot contain any entry that conflicts with {A, B, C}
- Thus, it cannot explicitly deny any range that is explicitly permitted by {A, B, C} or explicitly permit any range that is explicitly denied by {A, B, C};
- So, suppose {A, B, C, D} is legal, but does *not* produce a result desired by the operator;
- *And* {A, B, C, D, E} is legal and fixes the undesired result of {A, B, C, D};
- For this to be the case, E would have to change the effect of D in the range in which they overlap;
- But since D and E do not conflict; this is not possible;

# Conclusion

- VSI-to-Profile binding does *not* require a protocol in which multiple bindings are committed as an atomic operation;
  - And the order of bindings specified within such an atomic operation can be shown not to be useful;
- It is sufficient for the Server to request individual bindings that may be accepted or rejected by the Bridge;
- If there is disagreement with this conclusion please identify a specific flaw in the argument or provide a counterexample;
- A simple protocol (eg., ACP) that requests one binding at a time, and either accepts or rejects that binding, meets the requirement for VSI-to-Profile binding;
  - Note: Earlier descriptions of ACP did not include flow control to prevent buffer overrun; we assume that flow is added to the proposed ACP;
- The use of LLDP or an LLDP-like protocol (with or without T3P) adds very significant complexity to meet requirements that don't exist (even considering that LLDP already exists.

# A Further Note

- It might be claimed that it is necessary to send the entire database with each binding request (and periodically) in order to keep the database ‘refreshed’ on the Bridge;
- This is not the case;
- The requirement is only to keep the *individual* database entries refreshed;
- This does not require sending the entire database;
- There is no significant difference in the bandwidth or cycles required to periodically refresh the database by (1) sending the database in its entirety or (2) sending each entry at periods after its commitment;
  - It could be argued that it’s better to send the refreshes individually as this would tend the spread the load;
- There is, however, a significant increase in bandwidth and processing if we ship the entire database each time a new binding is established;