Thoughts on DCBX

Joe Pelissier

az-pelissier-dcbx-thoughts-0109

Background

- **DCBX provides two state machines**
  - **Control state machine**
    - Provides acknowledgement of receipt of DCBX TLV
  - **Feature state machine**
    - Runs on top of the control state machine
    - Provides negotiation of features or operational parameters
    - In general, operates between a bridge and an end station

Negotiated parameters

- **The current DCBX proposal provides for negotiation of the following parameters:**
  - **Priority group parameters including:**
    - Number of traffic classes supported
    - Priority group bandwidth percentage
    - Priority to priority group assignment
  - **Priority Flow Control parameters including:**
    - Number of traffic classes that support priority flow control
    - Priority flow control enabled per priority
  - **Protocol parameters including:**
    - Priority assignment for the protocol
- **Let's examine each of these Individually**

But First…Some Mythbusting

- **The Claim: we should keep at least a simplified version of the DCBX framework to enable negotiation of parameters**
- **The Reality: LLDP can provide this without any of the DCBX protocol or feature state machines**

  And its remarkably easy to do…

LLDP Negotiation

- **Assume we have a parameter that may be either "On" or "Off" that we wish to negotiate in a manner similar to DCBX**
- **There are five related variables:**

  FE.OS: the operational state of the far end of the link, received via LLDP, three values: "On", "Off", or NULL (indicating this value has not been received via LLDP)

  FE.W: indicates that the far end is willing to accept our operational state if and only if we are not willing. Received via LLDP, values are True, False, or NULL

  MY.DS: my desired state for this parameter. Administratively set to "On" or "Off". Note: this is *not* transmitted in LLDP.

  MY.W: indicates my willingness to accept the far end's operational state if and only if FE.W=FALSE. Transmitted in LLDP.

  MY.OS: my current operational state, transmitted in LLDP, set as follows:

  If (MY.W and !FE.W) then MY.OS<-FE.OS else MY.OS<-MY.DS

- **That's all folks!**

Another Myth…

- **Exchanging "Willing" is sufficient to prevent negotiation thrashing**
- **The reality: If "willingness" is misconfigured, thrashing is still possible**
  - **Consider 3 bridges: A, B, and C, all connected to each other**
    - A is willing to accept B but not C
    - B is willing to accept C but not A
    - C is willing to accept A but not B
  - **Forms a "Circle of Willingness" which could thrash indefinitely**
- **This is one reason why one should never use LLDP to distribute parameters *through* switches**
  - **It is fundamentally insufficient**

Priority group parameters

- **Number of traffic classes supported**

  This parameter is informational and cannot be negotiated

  Furthermore, knowing the number of traffic class supported by an end station is not useful to a bridge since the bridge does not have knowledge of the applications the end station wishes to execute

  Likewise knowing the number traffic classes the bridge supports is not useful to the end station since the bridge in general will map all priorities to available traffic classes

  Since there is no expectation that this must be the same between an end station and a bridge, “willing” has no meaning

Priority group parameters

- **Priority group bandwidth percentage**

  While this parameter could be negotiated it makes no sense to do so

  It is reasonable to expect that the bandwidth percentage assigned by an end station would be different than the bandwidth percentage assigned by the switch

    Given the asymmetric nature of the traffic it would be reasonable to expect that this would be the common case

  There is no point negotiating this parameter although there is no harm in exchanging it

  Again, since there is no expectation that this must be the same between an end station and a bridge, "willing" has no meaning

    An endstation *may* adjust its BW assignments based on this exchange, a bridge *shall not.*

Priority group parameters

- **Priority to priority group assignment**
  - **In general priority group assignment has bridge wide if not fabric wide relevance**
  - **An end station would not have sufficient visibility to properly dictate these assignments for bridge**
  - **Likewise, a bridge does not have sufficient knowledge of the applications executing on an end station to provide this assignment for the end station**
  - **Furthermore there is no reason that these assignments need to be consistent between an end station and the bridge**
  - **There is no need to negotiate this parameter, nor is there any meaning to “willing”**

Priority Flow Control parameters

- **Number of traffic classes that support priority flow control**

  This parameter represents a physical limitation of a bridge or an end station and cannot be negotiated

  A bridge has no need for knowledge of the end station's limitations since it does not have knowledge of the applications that will be running on the end station

  And an end station has no need to know this parameter from its peer bridge since the bridge will map priorities into the available traffic classes that do support per priority flow control

  Again, there is no meaning to "willing" in this case

Priority Flow Control parameters

- **Priority flow control enabled per priority**
  - **It is useful for an end station to know the priority flow control settings for its peer bridge**
  - **However the use of priority flow control has fabric wide significance and therefore an end station is not in a position to negotiate its use**
  - **It is not necessary for bridge to know whether it's peer end station is capable of supporting the priority flow control settings**
    - If an end station chooses to operate without priority flow control on a priority for which it is enabled by the bridge, the end station will simply lose frames; however, the rest of the fabric will be protected
    - Likewise, if an end station chooses to ignore priority flow control on a priority for which it is not enabled by the bridge, the bridge will simply ignore the priority flow control frames and the fabric will operate as if it was not enabled by the end station
  - **In summary, it is useful for the end station to know this information so that it may configure itself for proper operation; however, this parameter is not negotiable**
  - **Furthermore the bridge does not need to have knowledge of the acceptance of this parameter on the part of the end station in order to ensure proper fabric operation**
- **A bridge *shall not* adjust its PFC parameters based on what it receives via LLDP from a peer bridge**
  - **Otherwise, a "Circle of Willingness" may occur**

Priority Flow Control parameters

- **We currently state that if the PFC parameters (i.e. enabled or not per priority) do not match, the feature is "disabled"**
  - **What does "disabled" mean in this context?**
    - Does the bridge prohibit all communication on the port
      - This would make it difficult to correct the problem
    - Does the bridge prohibit communication on the unmatched priorities?
      - Could make it difficult to correct the problem
      - There is no harm in allowing communication to continue
        - End station cannot harm overall fabric operation
        - Whether this materially impacts the applications is beyond the ability for the bridge to know (e.g. FCoE: possibly, iSCSI: probably not).
    - In general, the best policy is to allow communication to at least limp along until the problem can be fixed
      - Which implies the bridge takes no action and "disabled" has no meaning
      - From an end station point of view, whether it attempts to provide communication for a particular protocol given this mismatch is a matter of local policy (most protocols would work fine either way)

Protocol parameters

- **Priority assignment for a protocol**

  Priority assignments in general have fabric wide significance

  Therefore an end station would not have sufficient visibility to dictate the priority assignment for a given protocol

  It is useful for the bridge to inform an end station of the priority assignment for a particular application to allow the end station to configure itself for proper operation

  It is not necessary for the bridge to have knowledge of the end station's acceptance of this parameter in order to ensure proper fabric operation

    If the bridge does not trust the end station to behave properly it may simply enforce use of a particular priority for a given protocol (for example by use of access control lists)

- **A bridge shall not adjust these assignments based on data received from LLDP (to prevent the "Circle of Willingness"**

Summary

- So far the parameters that we have defined to be exchanged with DCBX fall into two categories:
  - Purely informational
  - Information from the bridge that the end station requires in order to configure itself for proper operation
- We have no examples of parameters from an end station that would be used by a bridge
- We have no examples of parameters that are actually negotiable
- We would never want a bridge to adjust any parameter based on LLDP that would be reflected through the bridge
  - Necessary to prevent the “Circle of Willingness”

More observations

- **The DCBX feature state machine is quite complex**
  - **There has been confusion around the use of various combinations of willing and not willing**
  - **there has been confusion around the use of the error bit**
  - **There has been confusion around what it means to be compatible**
  - **It is not clear what it means for a feature to be enabled**
    - For example if a particular feature is advertised by one end of the link and not the other it will appear as enabled on the advertised end but not on the un-advertised end
    - Clearly the fact that a bridge is not advertising a particular protocol does not imply that the protocol is disabled on the bridge
- **Do we have a solution that does not quite work for a problem that does not quite exist?**

For consideration

- **Is it sufficient to simply exchange these parameters without the need of maintaining state related to “willing”, “enabled”, and “error”**
  - **We do not appear to have any counterexamples**
- **If we encounter a counterexample, can we use simple LLDP negotiation described at the beginning of this presentation?**
- **If this is the case, we could dramatically simplify DCBX by simply eliminating the feature and protocol state machines**
  - **Reduces interoperability issues**
  - **Increases protocol robustness**
  - **Reduces user complexity**
  - **Simplifies and expedites specification development**

Thank You!