Agreement protocol

Mick Seaman

This note describes the Agreement Protocol specified for P802.1aq Shortest Path Bridging[1], and in particular the use of the Agreement Number (AN) and Discarded Agreement Number (DAN) sequence numbers. The protocol's Agreement Digest summarizes the physical topology of the network, and is computed in a way that ensures that the risk of protocol participants with different views of that topology computing the same Digest is negligible. As a protocol participant encounters topology changes, it[2] successively limits the frames it forwards to a subset determined by a set of loop-free forwarding rules[3] for those successive topologies. Each participant forwards using the full set of active topologies corresponding to its currently perceived physical topology (i.e. ignores prior topologies) only when its Digest *matches*[4] that of its neighbours and sequence number conditions are met: it can then be sure that they are also forwarding using that topology, or one of its subsequent subsets.

The AN and DAN are needed because there is buffering between protocol participants. SPB may run over long distance services that can exhibit significant delays and a non-negligible, if small, risk of misordering. A participant needs to know that a prior Digest value is not 'in flight' before declaring a *topology match*. Otherwise a neighbour might use that value as a starting point after communicating a different prior value. The sequence numbering also deals with misordering, without requiring additional mechanisms to recover from arbitrary disruption or participant re-initialization.

In addition to providing an overview of the protocol and its operation, this note provides a proof of correctness (guaranteeing loop-free behavior independent of message transit delays) when protocol participants are connected by a service that either does not misorder frames or where misordering can be detected by the specified protocol mechanisms (i.e. when misordered messages are no more than one set of changes out of date).

1. Introduction

In the absence of the AN and DAN sequence numbers, the scenario illustrated in Figure 1 would be possible. It starts with participants A(lice) and B(ob) basing their calculations on the physical topologies represented by digests '1' and '2' respectively. Then (more or less at the same time) each receives link state information communicating the other topology. Agreement protocol messages originating with the prior topology views cross, and each adopts full forwarding based on two topologies whose arbitrary combination might cause a loop.

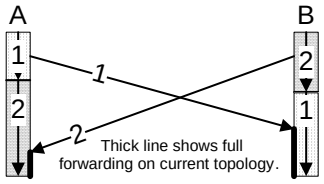

Figure 1—Crossing agreements

While such a loop would have been resolved as soon as Alice and Bob converge (using IS-IS mechanisms) on the same physical topology, and the circumstances that would cause it might be thought unlikely, they are certainly not impossible if there are a number of flapping links. Further agreement messages do not resolve the issue by themselves: see Figure 2.

When each participant receives the second message in Figure 2 (for topology '1' in the case of Alice receiving) it has no longer has a detailed record of the old topology—it might differ slightly or considerably from that current. Certainly the receiving participant should not stop forwarding entirely because its peer

[1]P802.1aq/D2.7 clauses 13.17, 13.27, 13.29.14, 13.29.28, 14, 28.11.3.5.1.

[2]There is a participant per Bridge Port and forwarding is limited only on a per port basis, so the neighbours concerned are only those attached to a single LAN (often just one). A single match does not have to propagate throughout the network before forwarding is improved.

[3]See Link state agreement, September 6th 2010, prior version was Link state agreement, March18th 2010.

[4]This note uses the term *match* rather than sync, synchronization, agree etc. as the latter already have meanings in the context of 802.1Q Clause 13. A 'digest match' or 'matching digests' is used to mean that the Digests transmitted and received by a participant have the same value. A 'topology match' is only declared if the digests match and the AN/DAN conditions specified in this note are met.

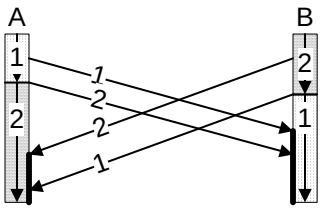

Figure 2—Further agreement messages

sees a different topology: indeed an agreement message with an unknown digest should be cached so each participants need send only one message and receive only one to move from complete forwarding on one topology to complete forwarding on its successor, as in Figure 3.

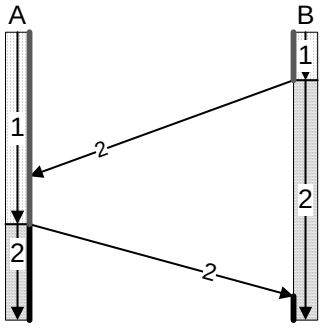

Figure 3—Normal topology progression

The scenario in Figure 1 is prevented by including the AN (agreement number) and DAN (discarded agreement number) in each agreement message. When a participant calculates a new topology and the accompanying digest, it increments its own AN and checks for a topology match. When the participant receives an agreement protocol message, it sets its transmitted DAN to the received AN, and then checks for a topology match. If, on checking for a topology match, the participant finds that the last received and currently transmitted digests are equal, it sets its transmitted DAN equal to the last received AN plus 1, and if the last received DAN is also equal to its AN or AN plus 1, it has matched topologies. In other words, as a first step towards a complete protocol description (below):

```
topologyUpdate()
{  tx.digest = calculatedDigest; tx.an++;
   checkTopologyMatch();
}

messageReception()
{  rx.digest = msg.digest; rx.an = msg.an;
   rx.dan = msg.dan; tx.dan = rx.an;
   checkTopologyMatch();
}

checkTopologyMatch()
{  if  (rx.digest == tx.digest)
   {  tx.dan = rx.an+1;
      if  ((rx.dan == tx.an) || (rx.dan == tx.an+1))
      {  topologyMatched();
} } }
```

Figure 4 shows how the sequence numbering handles the Figure 1 scenario.

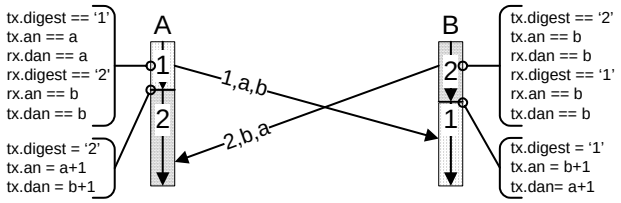

Figure 4—Handling crossing agreements

When the crossing messages are received, their DANs lag rather than precede or equal the receivers' ANs, so a false topology match is not declared.

Figure 6 shows the influence of further messages, as IS-IS converges on a stable physical topology.

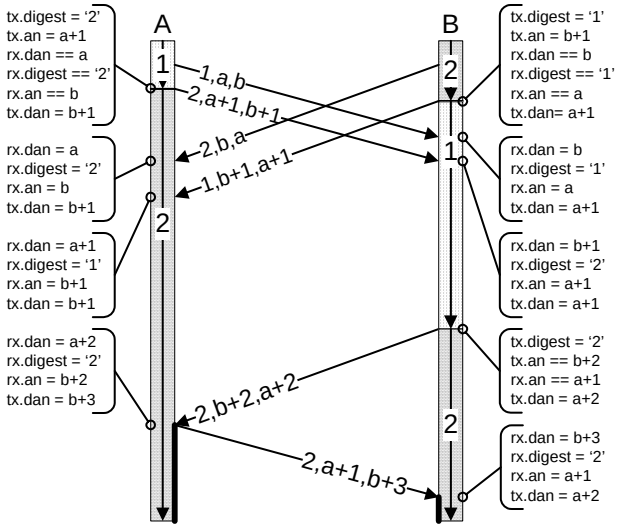

Figure 5—Further messages and stable topology

Figure 6 shows the normal topology progression introduced in Figure 3, including the last periodic messages transmissions for the initial topology so the initial state of the participants is clear. The actual change to complete forwarding on the new topology takes just two messages—one from each of the participants.

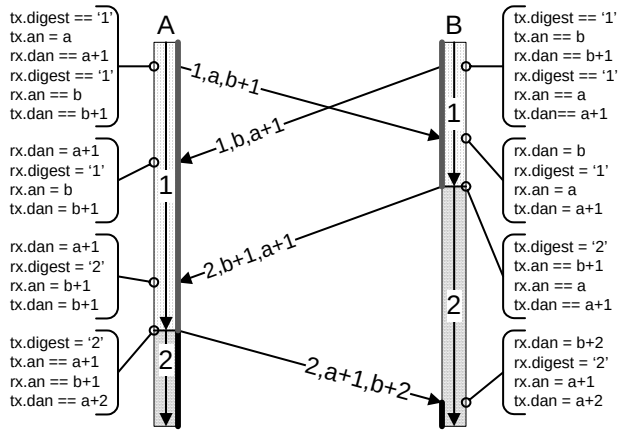

Figure 6—Handling normal topology progression

Figure 7 shows what happens when there is a 'glitch' in the topology, a temporary change that is noted by only one of the participants[1].

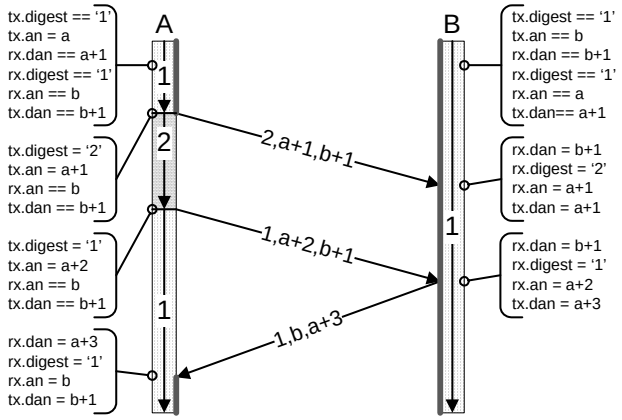

Figure 7—Recovering from a glitch

2. Misordering

Without further refinement, the protocol as described so far in this note handles a number of cases of message reordering without creating potential topology conflicts. The received DAN checking is often a sufficient defence. However such conflicts are possible, as illustrated in Figure 8.

These conflicts could be prevented by simply discarding out-of-order frames, but if one participant gets out of sync with the other (perhaps because it is reinitialized) connectivity will be lost permanently. It is highly undesirable to have to introduce additional mechanisms to recover from this eventuality. The solution is to restrict topology matches following out-of-order reception to those where the received DAN is the transmit AN plus one. This guarantees that the

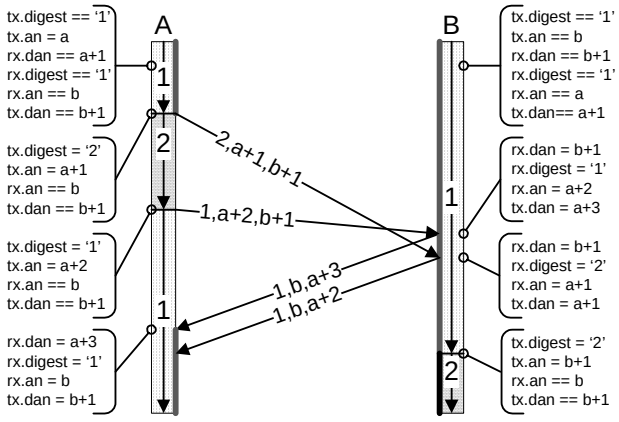

Figure 8—Misordering and conflict

received message was sent after both the receiving and transmitting participants have settled on the same topology. Such a match will always occur eventually provided that IS-IS does cause both participants to settle on the same topology. Subsequent matches can use either the received DAN equals the received AN and the 'plus one' condition. So the next step towards a complete protocol description (below) is:

```
topologyUpdate()
{   tx.digest = calculatedDigest; tx.an++;
    checkTopologyMatch();
}

messageReception()
{   if  (msg.an < rx.an) outOfOrder = True;
    rx.digest = msg.digest; rx.an = msg.an;
    rx.dan = msg.dan; tx.dan = rx.an;
    checkTopologyMatch();
}

checkTopologyMatch()
{   if  (rx.digest == tx.digest)
    {   tx.dan = rx.an+1;
        if  (   ((rx.dan == tx.an) && !outOfOrder)
            ||  ( rx.dan == tx.an+1))
        {   topologyMatched(); outOfOrder = False;
}   }   }
```

3. When to transmit

An agreement protocol message should be scheduled for transmission whenever the transmit AN and transmit DAN are updated, but not otherwise. The result is that (in the absence of message loss) any digest change that will result in a match will cause the necessary exchange of messages to take place, but the message sequence will not be prolonged when that is not an immediate possibility. Figures 5 through 7 illustrate the desired behavior. It is of course safe to

[1]The Agreement Digest reflects the physical topology, not the whole of the IS-IS state, in particular it omits IS-IS sequence numbers so one participant might simply process the latest of a number of LSPs and thus miss temporary topology perturbations visible to others.

transmit a message at any time, so agreement protocol messages can be embedded in other protocol messages (such as BPDUS and IS-IS Hellos) whose transmission may be scheduled for other reasons.

Transmissions should occur periodically, so that message loss does not delay topology agreement indefinitely. When the Agreement Digest is carried in BPDUs, the natural Hello Time of 2 seconds suffices for this purpose. When it is desirable to also carry the Agreement Digest in IS-IS Hellos (other message) a refresh frequency suited to that of the carrying protocol should be chosen.

As with other 802.1 protocol specifications, message transmission is modelled as occurring in two steps. First the protocol indicates the need to transmit a message by setting the variable ntt to True. This tells the operating environment that a transmission should be scheduled. Secondly, the protocol variables are encoded in the message PDU when that becomes available (notionally through some call back mechanism). This two step process explicitly answers the questions that might arise when a buffer is not immediately available for transmission, or the protocol state changes more rapidly than such buffers can be provided. It should also be clear that the protocol state is always such that a transmission is possible—delaying or avoiding transmission in order to conceal confusion or inconsistency is not permitted.

Transmission prompts are shown explicitly in the complete protocol procedures (below) after initialization and sequence number space issues are discussed.

4. Initialization

In principle, protocol participants can start in any state whatsoever—the normal operation of the protocol will ensure that the participants transition to the correct states. In practice, implementations are easier to debug if the initial states are specified, and some unnecessary message exchanges can be avoided. Recommended initial states are shown as set as part of the operation of begin() (below).

5. Sequence number space

So far the protocol description has ignored the limitations imposed by the small (two-bit) AN sequence number space. In order to distinguish old out-of-order messages from AN increments resulting from several closely spaced changes with possible message loss, the use of fresh AN's needs to be modified by feedback from each participant's peer. This is done by using the received DAN to rotate the sequence number window. The transmit AN can be increased as far as the received DAN plus one. Thus if Alice's AN is currently a, and her digest matches with Bob's she would naturally expect to receive a message with a DAN of a+1, giving her permission to transmit with an AN of a+1 or a+2[1] before a further message from Bob is required to rotate the window once more. Thus a+3, which is indistinguishable from a-1, is temporarily outside the widow, and Bob can identify misordered messages as long as they are no more than one topology change out of date.

The sequence number modifications are shown as part of the completed protocol description (below).

6. Early matching

Before transmitting a digest that Bob can use for a match, Alice has to reduce her forwarding to a subset permitted by the forwarding rules and the agreements implied by the topology identified by that digest. If she continues to transmit the prior digest in the meantime, the forwarding rules have to take account of its implied agreements as well, just in case Bob reverts to that prior topology and matches its digest. The effort that Alice expends in reducing her topology to satisfy both new and old digests is likely to be a waste of effort if her newer digest already matches Bob's—she has reduce her forwarding to transmit the new digest, then declare the match, and then adjust the forwarding again to be all that permitted by the new topology.

A more efficient approach is for Alice to transmit (and match) with her newly calculated digest, but include an agree flag in the message that remains False until her forwarding aligns with the matched topology. In this way only one of two protocol participants on a point-to-point link (all but one on a shared media LAN) needs to reduce forwarding to the point that it satisfies the constraints of the loop-free forwarding rules for the agreements implied by both current and prior topologies. In the extreme case where one participant is at the edge of the shortest path domain for all trees (i.e. does not provide transit forwarding within the domain for any tree) it may have no forwarding changes to make at all, so it can send a digest to be matched (with msg.agree set) almost immediately, allowing the other participant to make just the forwarding changes need to align with the new matching topology before he sets tx.agree.

[1] If the peer participant has declared a topology match with the current digest, otherwise the window advances by one.

Of course, if FDB changes can be made much more rapidly than messages can be sent, this potential optimization may prove unnecessary. If this is the case, allowing and specifying the optimization does no harm—by the time that ntt (see 3 above) results in an opportunity to transmit tx.agree is already set. Optimizing forwarding updates is an implementation specific problem, but a general approach, likely to yield good results is first to remove entries that are simply not permitted by the new topology before subsetting the forwarding for that topology using the loop-free forwarding rule constraints from prior topologies—thus maximizing the chance that the other protocol partner(s) will complete the latter first and render some of those changes unnecessary. The last step, following a successful match, is to install new FDB entries that would not be permitted by the forwarding rules if agreements were outsanding for prior topologies.

Checking rx.agree to ensure that it is set before declaring match could be modelled by overloading the '==' operator in (rx.digest == tx.digest), so need not complicate either the examples given so far or proofs of protocol operation, but it as well to make the test explicit in pseudo-code to ensure that it is not missed, and this is done in the completed protocol description (below). Completion of the forwarding changes necessary for tx.agree to be set for the latest calculated tx.digest results in a call to messageUpdate() with allSptAgree (reset as a direct consequence or side-effect of calculating the new topology) True.

7. Complete protocol

```
begin()
{   tx.digest = initDigest[1]; tx.agree = False;
    rx.digest = initDigest; rx.agree = False;
    outOfOrder = True;
    rx.an = 0; rx.dan = 0;
    tx.an = 1; tx.dan = 0;
    messageUpdate();
}

topologyUpdate()
{   messageUpdate(); checkTopologyMatch();
}

forwardingUpdate()
{   messageUpdate(); checkTopologyMatch();
}

messageReception()
{   if  (msg.an == rx.an+3) outOfOrder = True;
    rx.digest = msg.digest; rx.agree = msg.agree;
    rx.an = msg.an; rx.dan = msg.dan;
    messageUpdate()[2];  checkTopologyMatch();
}

messageUpdate()
{   if  (   (tx.digest != calculatedDigest) &&
            (tx.an+1 == rx.dan) || (tx.an+1 == rx.dan+1))
    {   tx.digest = calculatedDigest; tx.an++;
        tx.agree = False;
    }
    if (allSptAgree && !tx.agree)
    {   tx.agree = True; ntt = True;
}   }

checkTopologyMatch()
{   if  (   (tx.digest == calculatedDigest) &&
            (rx.digest == tx.digest) && rx.agree)
    {   if (tx.dan != rx.an+1)
        {   tx.dan = rx.an+1; ntt = True;
        }
        if  (   (rx.dan == tx.an) && !outOfOrder)
            ||  (rx.dan == tx.an+1))
        {   topologyMatched(); outOfOrder = False;
    }   }
    else if (tx.dan != rx.an)
    {   tx.dan = rx.an; ntt = True;
}   }
```

8. Multiple participants

Although not explicitly detailed so far in this note, I hope it is reasonably clear that the protocol easily accommodates groups of participants connected over shared or pseudo-shared media[3], at the cost of four bits per participant[4] in each protocol message multicast by each participant to all the others. A topology match is generally declared only when the digest and received AN and DAN from each and every participant meets the specified criteria.

In this way the agreement protocol can also be used in non IS-IS scenarios where topology information is directly carried in messages, and there is some set of well defined rules as to what behavior is expected in any configuration and how that behavior should change in successive periods of change before the next match is declared. The protocol is naturally efficient in those scenarios, as its rules for matching involve the minimal possible message exchange.

---

[1]The initial value of rx.digest, initDigest is arbitrary, provided that allSptAgree does not become True until after calculatedDigest is calculated.

[2]Message reception may have advanced the transmit AN window so that tx.digest can be updated.

[3]Or over some more complex set of links.

[4]The cost of identifying which participant each set of bits belong to can be avoided as the participants are easily placed in order (by system id or MAC address). The digest itself ensures that different participants agree on which participants are participating and which bits go with which.

9. Proofs

9.1 Loop-free in the absence of misordering

First we show that, unless protocol messages are misordered, one participant will only declare a topology match and remain forwarding on that topology without further subsetting if the other has declared a match or subsetted his forwarding using that same topology.

The development of some simple terminology and temporal logic allows us to see the wood for the trees when representing the changing states of the protocol participants and their inter-dependencies. We treat the protocol as having an unlimited supply of sequence numbers that are never reused, with only the real integer part modulo 4 being carried in the two-bit message fields. Since each increase in a transmitter's two-bit AN field is limited, knowledge of the prior extended value allows a protocol observer to determine the corresponding extended values without ambiguity. It is also convenient to view the ANs generated by Alice in any protocol run, together with the DANs returned by Bob, as being drawn from a different number space (AN_A) from Bob's ANs and Alice's DANs (AN_B)[1]. Let:

$a,a',a'' \in AN_A$
$b,b',b'' \in AN_B$
$\forall x \in AN_A\ (x+1 \in AN_A)$
$\forall x \in AN_B\ (x+1 \in AN_B)$
$AN_A \cap AN_B = \varnothing$

Since each participant increments its AN when and only when the transmitted digest changes, we can identify the digest value by its AN and define the function D(x,y) as being True iff the digests identified by $x, y \in AN_A \cup AN_B$ have the same value. Further:

$$\forall x\ \forall y{:}(D(x,y))\ (\neg D(x-1,y) \wedge \neg D(x+1,y))$$

In addition each message can be represented simply by its <an>.<dan> tuple, for example:

a.b+1

If we know to which participant we refer, the separate sequence number spaces also facilitate identification of receive and transmit variables[2]. These change, their value in a logical expression is a function of time[3] t:

$a.b+1_A$ — is True iff Alice's ((tx.an == a) $\wedge$ (tx.dan == b+1)) at time t
$b.a_A$ — refers to Alice's receive variables
$b.a_B$ — refers to Bob's transmit variables
$a+1.*_A$ — refers to Alice's tx.an alone, and is True iff her ((tx.an = a) at t, independent of tx.dan.

Strictly speaking:

$$a+1.*_A \equiv \exists\ x{:}\ (a+1.x_A)$$

and similarly for all other expressions including '*'.

If parts of any given logical statement are not qualified as to time, the entire statement is considered as being evaluated at the same time, e.g if p and q are declared to be time dependent variables:

$p \Rightarrow q$

means that q is True when p is True. Note that the individual members of AN_A and AN_B are time independent (and are not boolean variables).

Let M_{ab} denote the declaration of a topology match by Alice at time t when her AN is a and the last message she received from Bob had AN b. From the definition of the protocol:

$$\begin{aligned} M_{ab} \equiv & \qquad \ldots\ldots (1) \\ & (a.*_A \wedge D(a,b)) \qquad \ldots\ldots (1a) \\ & \wedge\ (\ (b.a_A) \qquad \ldots\ldots (1b.1) \\ & \quad \vee\ (b.a+1)_A) \qquad \ldots\ldots (1b.2) \end{aligned}$$

Equation 1 says that Alice declares a topology match at time t when her transmit AN is a and the last message she received from Bob had AN b iff: (a) her transmit AN is indeed a, and the digests for a and b match, and either: (b.1) the received AN is b (as required) and the received DAN a; or (b.2) the received AN is b and the received DAN a+1.

We wish to prove that:

$$\begin{aligned} & \exists\ t_1{:}((t = t_1) \wedge M_{ab}) \qquad \ldots\ldots (2) \\ \Rightarrow\ & \exists\ t_0{:}(\\ & \quad \forall\ t{:}((t = t_0) \wedge (t_0 < t_1)) \\ & \qquad (b.a_B \vee\ b.a+1_B) \qquad \ldots\ldots (2a) \\ & \wedge\ \forall\ t{:}((t_0 < t) \wedge (t < t_1))(\\ & \qquad \forall\ b'{:}(\neg D(b,b'))(\neg M_{b'*})) \qquad \ldots\ldots (2b) \end{aligned}$$

Equation 2 (required to prove) says that Alice's declaration of a topology match with transmit AN a and received AN b at some time t_1 implies that at some earlier time t_0 : (a) Bob's transmit AN was b, and his

[1] The sets generated by $a_1 = 1$ and $b_1 = 1+i$ and the successor function for each set are suitable.

[2] The messages are only interesting in that they permit one participant's variables to result in a later change in the others. Conventional advice is to model the channel(s) containing the messages within the overall system state, which causes the latter to be rather complex. Here we model the channel purely in terms of possible changes to receive variables, or requirements on transmit variables if receive variables are to change.

[3] Formally there is a set of times, with an ordering relation, and a set of tuples for each time-dependent variable, with one element for each time value with that value as part of the tuple and the value of the variable at that time as the other part of the tuple. Spelling this out every time we want the value of the variable at a particular time is just a little tedious.

transmit DAN a or a+1—these are necessary conditions for a message transmitted by Bob at that time to result in the topology match M_{ab} (see eqn. 1); and (b) Bob did not declare a conflicting topology match (with a digest that didn't match that for b) in the intervening time. These are sufficient conditions for loop-freeness: the rules for transmitting any Digest require Bob to reduce his forwarding to that (or a subset of that) for the topology identified by the Digest, and not to increase it unless he declares a topology match. The equation (once proven) can be applied (with the substitution of different free variables in place of a and b) to successive topology matches by both Alice and Bob, so a loop-free condition will persist for ever.

Whenever t_0 and t_1 are used in this proof they refer to time values that satisfy eqn. 2.

Alice's declaration M_{ab} requires her prior reception of b.a or b.a+1 so proving (2a)—the existence of a suitable t_0—is trivial, it follows directly from:

$$\exists\, t_j:((t = t_j) \wedge y.x_X) \Rightarrow \quad \exists\, t_i:((t_i < t_j) \wedge \forall\, t:(t = t_i)y.x_{Y\neq X}) \qquad \ldots\ldots (3)$$

Equation 3: one participant's receive AN and DAN will not take given values unless those have previously been the values of the other participant's transmit AN and DAN respectively[1]. This causality is generally assumed without explicit reference below. The rules of the protocol also require that each participant's (transmit) AN only increase over time:

$$\exists\, t_i:(x.*_X) \quad \Rightarrow \quad \forall\, t:(t \geq t_i)(\forall y:(y < x)(\neg y.*_X)) \qquad \ldots\ldots (4)$$

Equation 4: If a participant's (X's) transmit AN is x at time t_i, then it cannot be y less than x later. Conversely it was x at time t_j it cannot be y greater than x earlier. If messages are not delivered out of order, or out of order messages are detected and discarded on receipt then eqn. 4 means that the recipient's receive AN can also only increase over time. Any reference below to the behavior of the protocol in the absence of misordering may assume these properties.

Looking at (2a), we have from the protocol definition:

$$b.a_B \wedge D(a,b) \quad \Rightarrow \quad (a\text{-}1.b''_B) :(b'' \leq b) \qquad \ldots\ldots (5)$$

Equation 5: If Bob's transmit AN and DAN were respectively b and a (with matching digests), then his receive AN was a-1[2], with a digest matching that for Bob's transmit AN at or after the time of receipt but before Bob's transmit AN became b, since D(a-1,b) is False (see above). As a consequence Bob's received DAN (b" above) cannot be greater than b[3]. Furthermore (from eqn. 1):

$$\forall\, t:(t = t_1)(a.*_A \wedge b.*_A) \qquad \ldots\ldots (6)$$

so:

$$\neg\exists\ t:(\exists\ b':(a\text{-}1.*_A \wedge b'.*_A \wedge (b' > b)) \qquad \ldots\ldots (7)$$

and therefore (from (5) and (7)):

$$\forall\, t(\neg\exists\ b':((b' > b) \wedge a\text{-}1.b'_B)) \qquad \ldots\ldots (8)$$

Also[4] looking at (2a), from the protocol definition:

$$b.a+1_B \wedge D(a,b) \Rightarrow \quad a.*_B \vee a+1.*_B \qquad \ldots\ldots\ldots (9)$$

In the absence of misordering:

$$\forall\, t:((t_0 \leq t) \wedge (t \leq t_1))(\quad (\forall\, b':(b'.*_B)(b' \geq b)) \quad \wedge(\forall\, a':(a'.*_B)((a' = a) \vee (a' = a\text{-}1)) \qquad \ldots\ldots (10)$$

at and between t_0 and t_1 Bob's transmit AN was greater than or equal to b, and his receive AN was a or a-1. A conflicting topology match could have only taken place if these were not b and a respectively, so using eqn. 1 and 10:

$$\forall\, t:((t_0 < t) \wedge (t < t_1))(\forall\, a'\ \forall\, b': \quad (\neg D(b,b') \wedge M_{b'a'}) \quad (b'.*_B \wedge (b' > b) \wedge (a' = a\text{-}1) \quad \wedge \exists\, b'':(a'.b''_B \wedge ((b'' = b') \vee (b'' = b'+1)) \qquad \ldots\ldots (11)$$

Equation 11: At any time between t_0 and t_1, for any transmit AN values a', b' such that Bob declares a topology match conflicting with that for b (and, by extension, with that for a), Bob's transmit AN is b' (greater than b), his receive AN is a-1, and his receive DAN b" (equal to b' or b'+1). But, eqn. 8 stated that Bob's receive variables can never be set to that combination of values. So Bob cannot declare a conflicting topology match.

Q.E.D.

Postscript

The conditions for the topology match and their relationships as developed in the proof are illustrated in the four time sequences shown in Figure 9. In each case Bob cannot create a conflicting match in the

[1] Messages can be lost, otherwise the implication '⇒ would be bi-directional.

[2] Only the reception of a-1 (with a digest match at or after the time of receipt) and a could result in Bob having a transmit DAN of a, and if D(a.b) the latter would result in a DAN of a+1, as in the following equation. Reception of an in-order message always results in tx.dan being set, it is not carried forward as it can be on a topology update.

[3] If a participant's AN is b-1 then its DAN cannot be greater than (b-1)+1.

[4] See eqn. 5 above.

interval t0 to t1, since he would have to adopt a further digest with an AN of b+1 (or greater), which in turn would require reception of a DAN of b+1 (or greater), but the only message Alice can have transmitted prior to t1 with a DAN of b+1 has an AN of a, and hence the same digest as M(a,b) and she has not transmitted a message with a greater DAN.

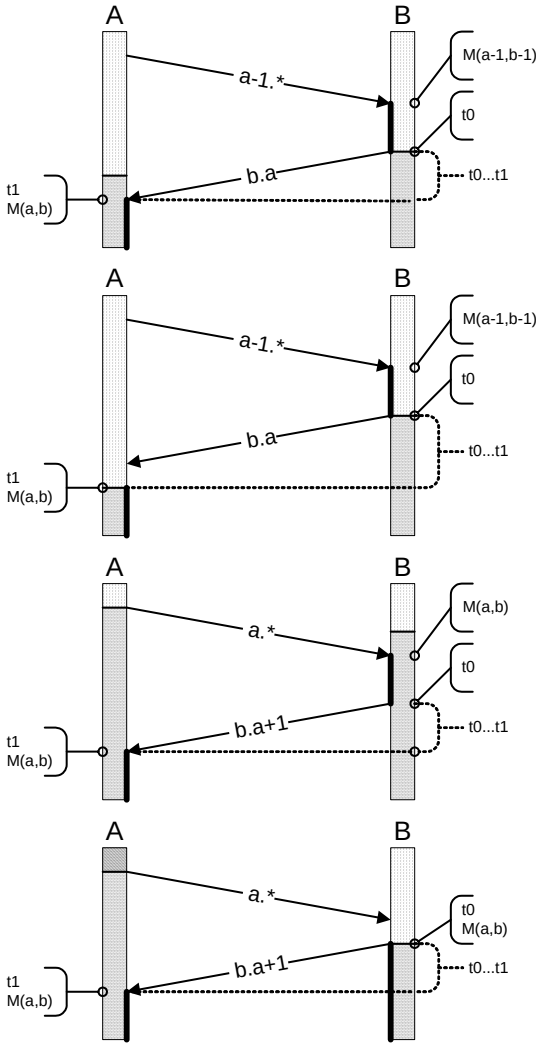

Figure 9—Sequences for a match

10. State machine variable description

The variable names used by P802.1aq to describe the agreement protocol, and the corresponding names used in this note are as follows:

- agreementDigest — calculatedDigest
- agreedN — rx.an
- agreedND — rx.dan
- agreeN — tx.an
- agreeND — tx.dan
- agreedDigest — rx.digest
- agreeDigest — tx.digest
- agreedPriority
- agreementOutstanding
- designatedPriority
- neighbourPriority

The following variables are missing from P802.1aq/D3.0, suggested names for a revised draft are as follows:

- agreedMisorder — outOfOrder

The following variables are included in P802.1aq/D3.0 but are not required, and should be removed in a revised draft:

- agreePending — tx.pending

As noted in one of the ballot comments P802.1aq/D3.0 continues to refer to TAP instead of simply ‘agreement protocol’ and there are some variables whose names are out of sync, in particular there are references to tapTxDigest and tapOwnDigest in the Port Transmit machine and these should be removed.

11. Topology and forwarding updates

The loop-free forwarding rules use the following per tree system variables to determine whether FDB entries that permit frame forwarding can be made (or indeed retained) if allSptAgree is to be set for a new topology and an agreement message with tx.digest = calculatedDigest and tx.agree = True can be sent:

- designatedPriority[1]

and the following for each bridge port per tree:

- agreementOutstanding—the greatest outstanding distance advertised to the neighbour(s) on that port for paths from the bridge constrained to one of the neighbours as the first hop (with the neighbour's designatedPriority being less than that of the bridge), zero if no such agreement is outstanding. $\overline{Y}_{Y \rightarrow Z}$[2].
- agreementsBelow—True if the neighbour on that port is closer to the Root in all outstanding agreements (or there are no outstanding agreements). $\beta_{Y \rightarrow Z}$ [2]. Note that, for the current agreement digest conventions and processing, agreementsBelow and agreedAbove will have exactly the same values, as the transmission of a digest (with tx.agree) that a protocol partner could match discards any held agreement that would cause agreedAbove to be set whenever agreementsBelow would transition False, and the same synchronization holds for the receiver. The pseudo-code in this section (11.1) uses both variables—both to confirm the above point and in case some agreement convention or processing is designed that would separate their values, but the use of agreedAbove alone is suggested when an agreement digest is used. The loop-free forwarding rules described below make this simplification.
- agreedPriority—the greatest distance for (all) the neighbour(s) on this port for paths from the neighbour(s) constrained to the bridge as the first hop in held agreements with the neighbour's designatedPriority being greater than that of the bridge, zero if there is no such agreement. $\overline{Z}_{Y \leftarrow Z}$ [2].
- agreedAbove—True if the neighbour on that port is closer to the Root in any received agreement held, False if no such agreement is held. $\alpha_{Y \leftarrow Z}$ [2]. portPathCost—ensuring that it is symmetric.
- selectedRole—the Port Role in the current topology.
- neighbourRole —the neighbour's Port Role in the current topology.

The designatedPriority, portPathCost, selectedRole, and neighbourRole are direct results of the link state calculation, and allow agreementOutstanding and agreedPriority to be updated per tree per port (with results depending on whether or not the agreement protocol has declared a topology match for the current topology—topologyMatched) as follows:

```
per Tree, per Port:
{  if  (   (selectedRole == RootPort) ||
           (selectedRole == AlternatePort))
   {   agreedPriority = LowestAgreementPriority;
       if (agreedTopology)
       {   agreementOutstanding =
               neighbourPriority + portPathCost;
           agreementsBelow = True;
           agreedAbove = True;
       }
       else
       {   if (agreementOutstanding <
               neighbourPriority + portPathCost)
               agreementOutstanding =
               neighbourPriority + portPathCost;
   }  }

   if  (selectedRole == DesignatedPort)
   {   agreementsBelow = False;
       agreedAbove = False;
       if (agreedTopology)
       {   agreementOutstanding =
               LowestAgreementPriority;
           agreedPriority =
               neighbourPriority + portPathCost;
       }
       else
       {   if (agreedPriority <
               neighbourPriority + portPathCost)
               agreedPriority =
               neighbourPriority + portPathCost;
}  }  }
```

At any stage topology matching agreement messages may be received and agreementOutstanding and agreedPriority recalculated, relaxing the loop-free

[1] 802.1D and 802.1Q's concept of a *priority vector* is equivalent to the routing concept of distance, generalized to allow for (a) the construction of a tree whose root (to or from which distance is measured) can change; (b) multiple regions, within which any distance is less significant than distance between regions. A priority vector with components {Root Identifier, Root Path Cost} is equivalent to the distance from the best possible Root with priority vector {0, 0}. The bridge's designatedPriority lacks the tie-breaker components used by RSTP/MSTP as ISIS-SPB tie-breaks to produce symmetric trees, picking Port Roles explicitly rather than leaving them to priority vector comparisons.

[2] In Link state agreement, September 6th 2010.

forwarding constraints so further FDB entries can be made (or replaced), for trees and/or ports previously considered. The following sequence of operations is suggested with a view to sending such agreement messages to the system's neighbours as early as possible, and so minimizing the number of transient FDB changes. The optimal approach will be heavily implementation dependent:

1) Remove FDB entries (or ports from port maps in FDB entries) that enable forwarding not permitted in the newly calculated topology.
2) Update agreementOutstanding and agreedPriority.
3) Remove FDB entries so agreement messages can be sent, and send them.
4) Add FDB entries as permitted by the loop-free forwarding rules.

12. SPB forwarding rules

12.1 SPBM unicast forwarding

SPBM unicast forwarding is supported by the "shortest path unicast" forwarding rule (Eqn. 6 for spUnicastY-Z[1]). This does not require any form of ingress checking, but can be supported simply by controlling the filtering database (FDB) entry for the unicast MAC address (frames for VLANs supported by SPBM are discarded (and not flooded) if their FDB entry is not found).

Such frames are considered by SPB as travelling on the shortest path tree rooted at the destination. The FDB entry can be made for an address and port if and only if, for the tree in question:

```
(selectedRole == RootPort) &&
(agreementOutstanding <= designatedPriority) &&
agreedAbove[2]
```

and for all ports (including the Root Port), for that tree:

```
(designatedPriority < agreedPriority) ||
agreedAbove
```

12.2 SPBV multicast and unicast forwarding

SPBV multicast and unicast frames are both supported by the "spanning tree" forwarding rule (Eqn.10 for stForwardingY-Z [1]), as source address learning and unknown unicast destination flooding are used. For all ports that permit frame ingress or egress:

```
(selectedRole == RootPort) &&
(agreementOutstanding <= designatedPriority)
```

or

```
(designatedPriority < agreedPriority) &&
(agreementOutstanding <= designatedPriority)
```

12.3 SPBM multicast forwarding

SPBM multicast forwarding is supported by the "source specific multicast" rule (Eqn. 13 for $ssMulticast_{X-Y-Z}$ [1]). This uses source (unicast) address ingress checking and source specific multicast egress.

For the ingress port:

```
(selectedRole == RootPort) &&
(agreementOutstanding <= designatedPriority)
```

and for all ports which allow the source specific multicast to egress:

```
(neighbourRole == RootPort) &&
(designatedPriority < agreedPriority)
```

Note that the ingress port checking requires that the frame be admitted on one port only, unlike the case of multi-path unicast forwarding with loop mitigation (as opposed to loop prevention which does not require ingress checking).

[1] In Link state agreement, September 6th 2010.

[2] This variable is "really" agreementsBelow, see the discussion in 11.1.