Discussion of New State Machines and Specifications for Transport of Time Sync in 802.1AS using 802.11 FTM

Geoffrey M. Garner
Consultant
gmgarner@alum.mit.edu

Carlos Aldana
Qualcomm
caldana@qca.qualcomm.com

IEEE 802.1 TSN TG
2016.05.14

Introduction

- This presentation addresses comments #77 and #79 against 802.1AS-Rev/D2.0
  - For comment #77, proposed requirements are presented
  - For comment #79, proposed master and slave state machines are presented (with some related explanatory material)

Comment #77 – 1

- Comment 77 pertains to subclause 12.3, Determination of TM and FTM capability, and concerned whether a time-aware system compliant with 802.1AS-Rev shall support TM, or if it is sufficient to support FTM only. It was decided (see the D2.0 final comment resolution):
  - If a time-aware system, where numberPorts = 1 and the single gPTP port is associated with an 802.11 interface, then to be asCapable it shall support either:
    - a) TM only, or
    - b) FTM only, or
    - c) both TM and FTM.
  - If a time-aware system, where numberPorts > 1 and at least one gPTP port is associated with an 802.11 interface, then to be asCapable it shall support either:
    - a) TM only, or
    - b) both TM and FTM.
  - In addition, bits (of the variable tmFtmSupport) will be used to indicate TM and/or FTM capability. Least significant bit will indicate TM capability. 2nd least significant bit will represent FTM capability.

Comment #77 – 2

- On initial consideration (by the editor), it as realized that adding this to the draft would require a port of a time-aware system to know how many ports its neighbor has
  - The reason for this is that asCapable is a property of the link and the two endpoint ports of the link
  - With the wording of the comment resolution, asCapable would be set to FALSE if, for example, one endpoint were a bridge that supported FTM but not TM, even if the other endpoint were an end device that supported FTM
  - After discussion with the original commenter and one of the participants who helped supply the comment resolution, it was determined that this was not intended
  - Rather, asCapable is to be set to TRUE if the two endpoints both support TM, both support FTM, and/or both support TM and FTM
    - In the latter case, FTM will be used
  - However, there also will be a statement in 802.1AS that a time-aware relay (see 3.24; we now use time-aware relay rather than time-aware bridge) shall support TM (i.e., for compliance with 802.1AS-Rev)
    - There will be a conformance statement in clause 5 and a PICS entry to this effect

Comment #77 – 3

- The above is captured by the editor's note on p.180, line 40
- The above are not inconsistent
  - It is possible to require that a bridge (i.e., relay) support TM to be considered compliant, yet allow time transfer using 802.1AS-Rev if the mechanisms to accomplish it are present
  - This is also consistent with comment #25 against D3.0, and the proposed resolution of this comment
- Regarding the internal variable tmFtmSupport (it is per port), if we implement the comment resolution we would set it as shown in the table on the next slide. In this table
  - Peer supports TM if the timing measurement bit in the Extended Capabilities information element defined in Table 8-103 of IEEE Std 802.11-2012 indicates that the peer 802.11 station is capable of participating in the timing measurement protocol
  - Peer supports FTM if the fine timing measurement responder and initiator bits in the Extended Capabilities information element defined in Table 9-134 of PIEEE 802.11-REVmc/D5.3 indicates that the peer 802.11 station is capable of participating in the fine timing measurement protocol

Comment #77 - 4

Least Significant 2 bits of tmFtmSupport

	Port Supports TM and FTM	Port Supports TM only	Port Supports FTM only	Port Supports neither TM nor FTM
Peer Supports TM and FTM	11	01	10	00
Peer Supports TM only	01	01	00	00
Peer Supports FTM only	10	00	10	00
Peer Supports neither FTM nor TM	00	00	00	00

Comment #77 – 5

- Assuming that FTM is used whenever possible, else TM is used if possible, else the link is not asCapable (we assume below that there are no other conditions that cause the per port, per domain asCapable to be FALSE in the cases of the first two main bullet items):
  - If (tmFtmSupport == 11 || tmFtmSupport == 10)
    - State machines invoke FTM code and requirements
  - Else if (tmFtmSupport == 01)
    - State machines invoke TM code and requirements
  - Else asCapable is FALSE for all domains on this port

Comment #77 – 6

- However, if the only purpose of the variable tmFtmSuport is to indicate in the state machines whether TM or FTM is being invoked, we do not need to distinguish the cases tmFtmSupport == 11 and tmFtmSupport == 10, because FTM is invoked in both those cases
- We could instead define an enumeration {TM, FTM, NEITHER}, perhaps taking on the values 0, 1, 2, and use (we could use FALSE instead of NEITHER):
  - FTM represents the cases 11 and 10 in the table
  - TM represents the case 01 in the table
  - NEITHER represents the case 00 in the table
- We could then simply test for these values in the state machines
- This is done in the discussion of comment #79, but we can easily convert to testing for 11, 10, 01, and 00 if desired

Comment #79 – 1

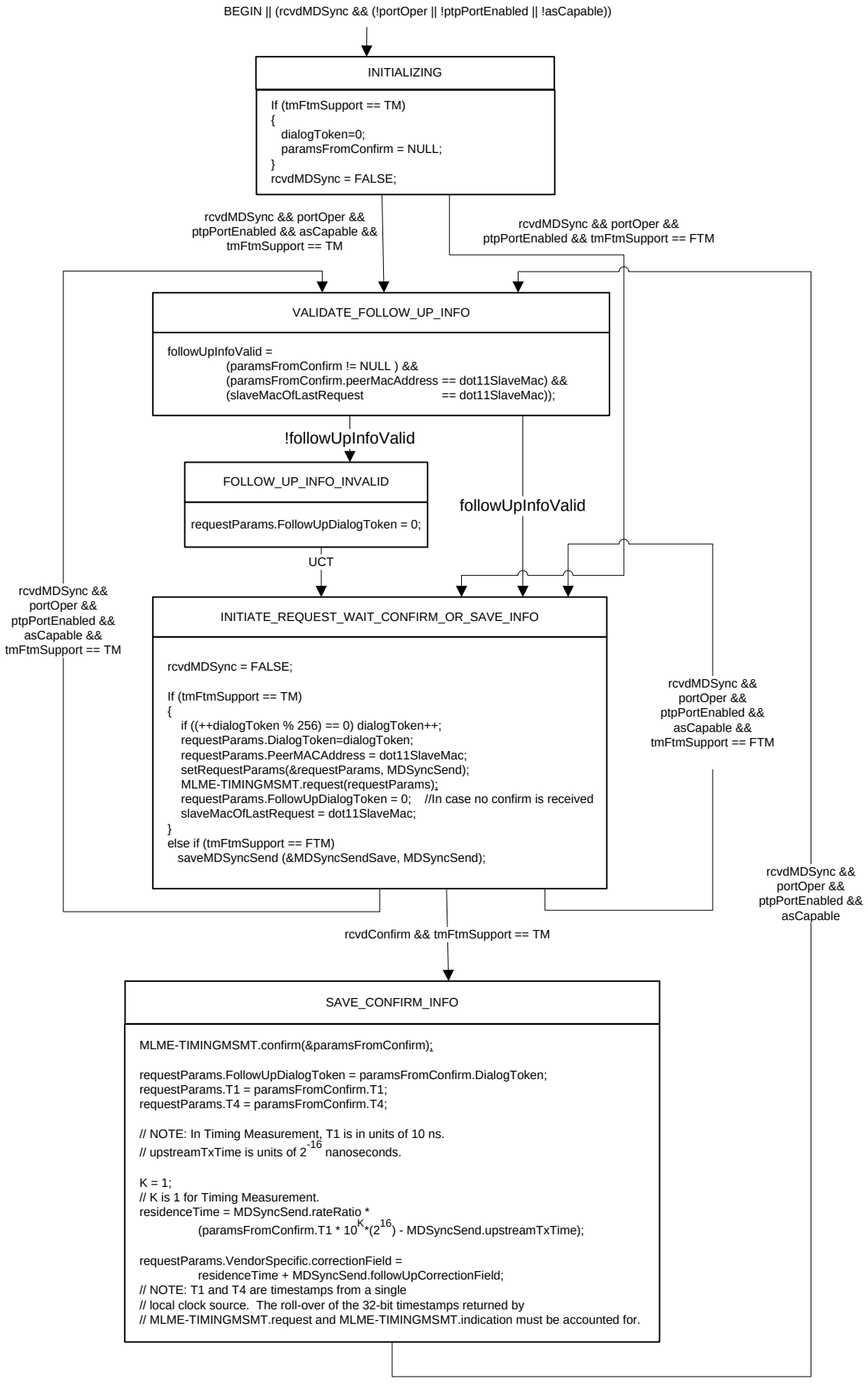
- Comment 79 pertains to the handling of FTM in the state machines
- The comment and its resolution reflect the fact that, while TM is initiated by the master, FTM is initiated by the slave sending an initial FTM request frame
  - This initial FTM request requests a burst of FTM frames from the master
  - The initial FTM request supplies the parameters for the master to use in sending the burst (e.g., burst duration, number of frames in a burst, min delta FTM, ASAP; see http://www.ieee802.org/1/files/public/docs2016/asrev-caldana-FTM-parameters-0116-v01.pdf for a discussion of these parameters)
  - If the master accepts the parameters, it sends the burst of FTM frames
  - On finishing, the master waits for a request from the slave for a new burst (i.e., with a new initial FTM request)
- The above differs from TM, and it would be highly desirable if FTM could be specified in such a way that the media-independent layer of 802.1AS (and therefore of IEEE 1588) is not changed (because, as of now, 802.11 FTM is the only timing protocol that uses this approach)
- A high-level description of a possible solution is in the Editor's Note that begins on p.175, line 2

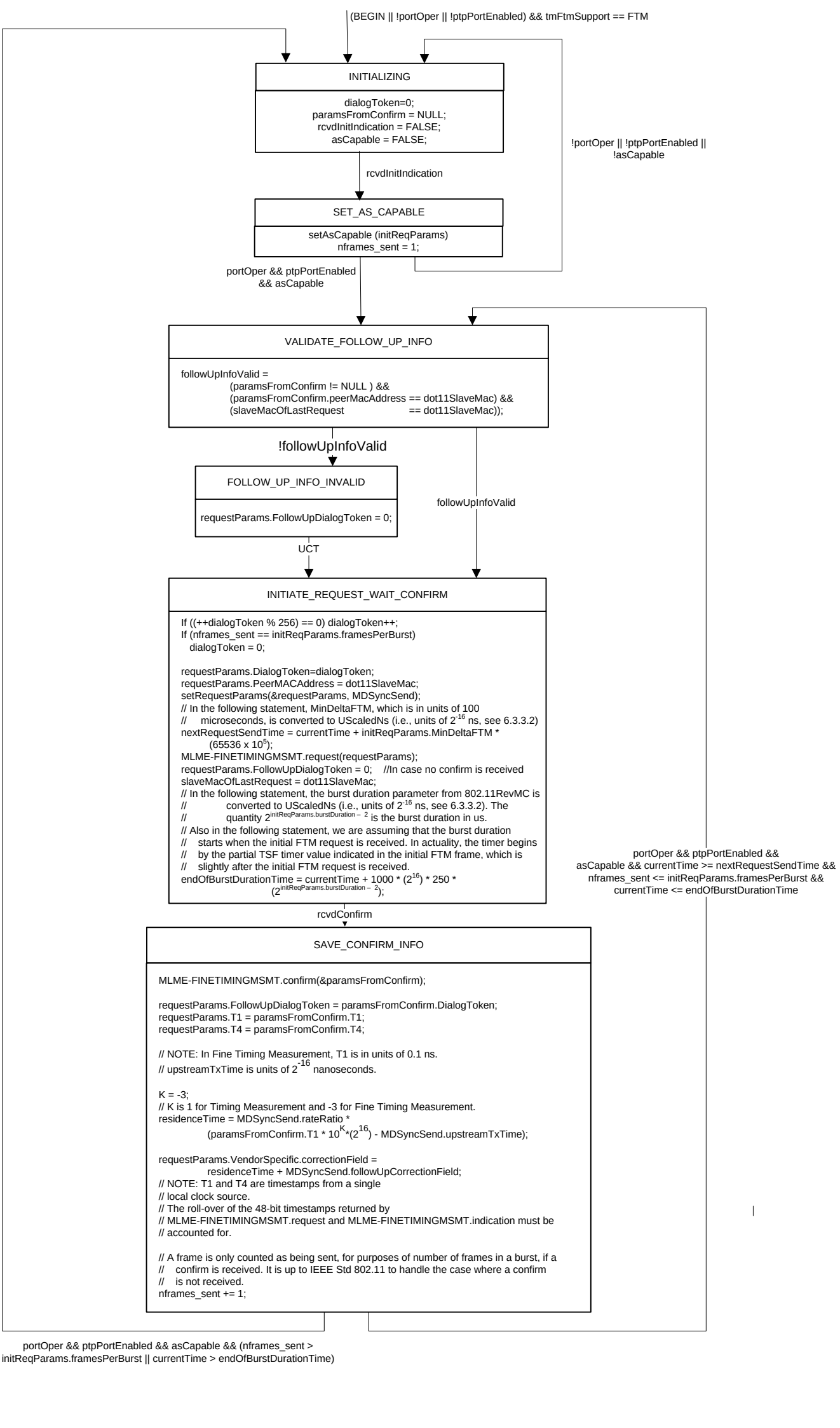
Comment #79 - 2

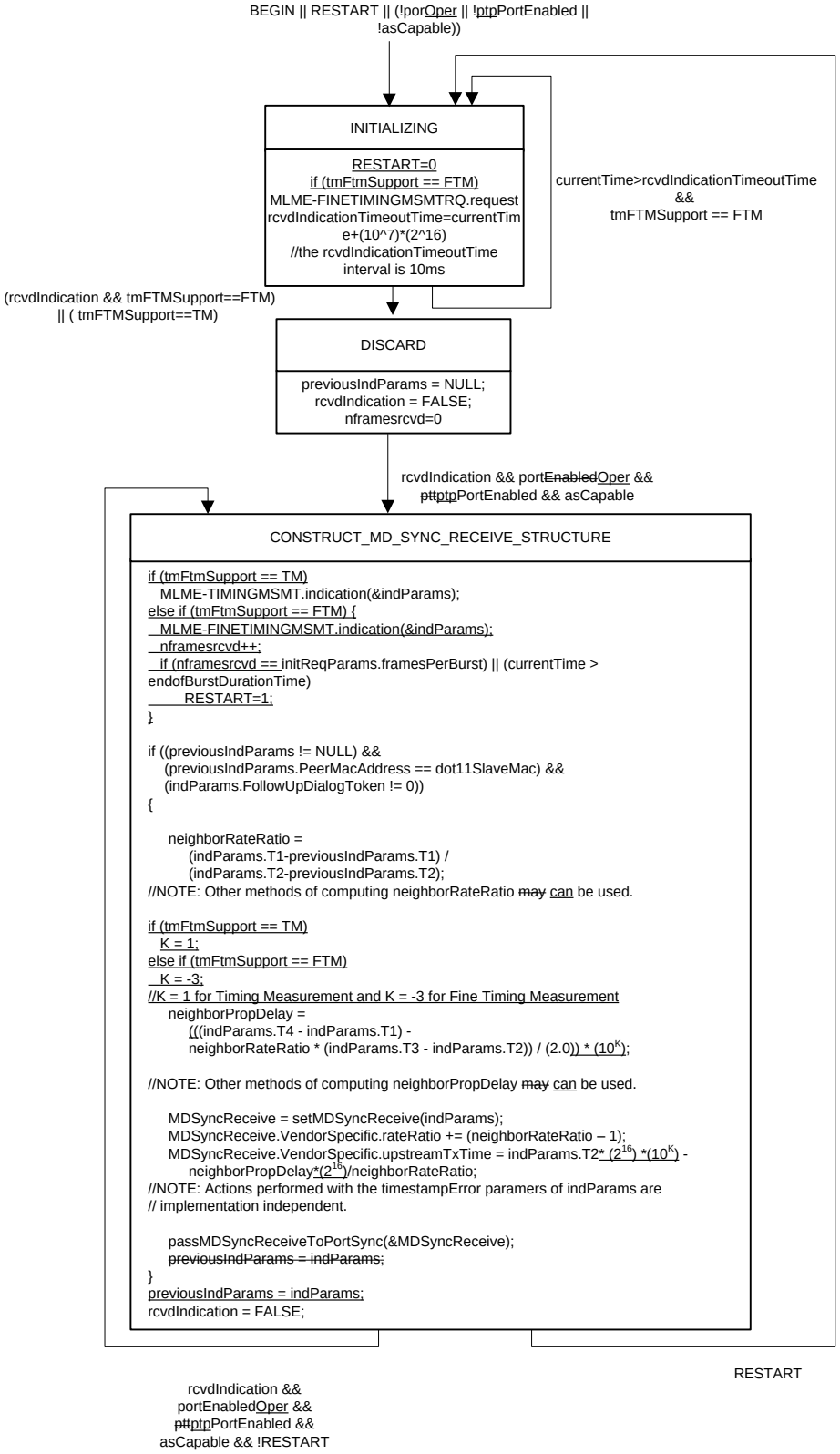
- To accomplish the above with only changes to the MD layer state machines, we will do the following for the slave state machine
  - Add the sending of the initial FTM request frame to the slave state machine in clause 12
  - After sending this request, the slave waits for FTM frames, and processes them when they arrive (as in TM)
  - If no FTM frames are received, it means the request was denied by the master, and a new initial FTM request is sent
  - At the end of the burst, we will assume, for now, that the slave makes a new initial FTM request. With this assumption, the overall average rate at which the slave receives time synchronization (i.e., FTM) messages depends on the requested FTM parameters by the slave. An informative Annex describing how the parameters may be chosen to achieve desired Sync rates is planned.
  - However, more general behavior is possible. For example, a low-power slave could go to sleep and wake up at some later time to make a new request. If the TSN TG desires such more general behavior, a presentation is needed to describe the details

Comment #79 – 3

- To accomplish the above with only changes to the MD layer state machines, we will do the following for the slave state machine
  - With the above approach for the slave state machine, the master state machine must be de-coupled into 2 state machines:
    - (1) Receipt and storing of time synch information from upstream
    - (2) Receipt of initial FTM request and sending of FTM frames to the slave
  - Master state machine (1) will be constructed by modifying the existing master state machine
  - Master state machine (2) is a new state machine, though much existing code can be re-used.
- Initial drafts for the revised state machines are on the following slides
  - First slide: Master State Machine 1
  - Second slide: Master State Machine 2
  - Third slide: Slave State Machine


BEGIN || (rcvdMDSync && (!portOper || !ptpPortEnabled || !asCapable))
INITIALIZING
If (tmFtmSupport == TM)
{
dialogToken=0;
paramsFromConfirm = NULL;
}
rcvdMDSync = FALSE;
rcvdMDSync && portOper && ptpPortEnabled && asCapable && tmFtmSupport == TM
rcvdMDSync && portOper && ptpPortEnabled && tmFtmSupport == FTM
VALIDATE_FOLLOW_UP_INFO
followUpInfoValid =
(paramsFromConfirm != NULL) &&
(paramsFromConfirm.peerMacAddress == dot11SlaveMac) &&
(slaveMacOfLastRequest == dot11SlaveMac));
!followUpInfoValid
FOLLOW_UP_INFO_INVALID
requestParams.FollowUpDialogToken = 0;
followUpInfoValid
UCT
rcvdMDSync && portOper && ptpPortEnabled && asCapable && tmFtmSupport == TM
INITIATE_REQUEST_WAIT_CONFIRM_OR_SAVE_INFO
rcvdMDSync = FALSE;
If (tmFtmSupport == TM)
{
if ((++dialogToken % 256) == 0) dialogToken++;
requestParams.DialogToken=dialogToken;
requestParams.PeerMACAddress = dot11SlaveMac;
setRequestParams(&requestParams, MDSyncSend);
MLME-TIMINGMSMT.request(requestParams);
requestParams.FollowUpDialogToken = 0; //In case no confirm is received
slaveMacOfLastRequest = dot11SlaveMac;
}
else if (tmFtmSupport == FTM)
saveMDSyncSend (&MDSyncSendSave, MDSyncSend);
rcvdMDSync && portOper && ptpPortEnabled && asCapable && tmFtmSupport == FTM
rcvdMDSync && portOper && ptpPortEnabled && asCapable
rcvdConfirm && tmFtmSupport == TM
SAVE_CONFIRM_INFO
MLME-TIMINGMSMT.confirm(&paramsFromConfirm);
requestParams.FollowUpDialogToken = paramsFromConfirm.DialogToken;
requestParams.T1 = paramsFromConfirm.T1;
requestParams.T4 = paramsFromConfirm.T4;
// NOTE: In Timing Measurement, T1 is in units of 10 ns.
// upstreamTxTime is units of 2-16 nanoseconds.
K = 1;
// K is 1 for Timing Measurement.
residenceTime = MDSyncSend.rateRatio *
(paramsFromConfirm.T1 * 10K*(216) - MDSyncSend.upstreamTxTime);
requestParams.VendorSpecific.correctionField =
residenceTime + MDSyncSend.followUpCorrectionField;
// NOTE: T1 and T4 are timestamps from a single
// local clock source. The roll-over of the 32-bit timestamps returned by
// MLME-TIMINGMSMT.request and MLME-TIMINGMSMT.indication must be accounted for.


(BEGIN || !portOper || !ptpPortEnabled) && tmFtmSupport == FTM
INITIALIZING
dialogToken=0;
paramsFromConfirm = NULL;
rcvdInitIndication = FALSE;
asCapable = FALSE;
!portOper || !ptpPortEnabled ||
!asCapable
rcvdInitIndication
SET_AS_CAPABLE
setAsCapable (initReqParams)
nframes_sent = 1;
portOper && ptpPortEnabled
&& asCapable
VALIDATE_FOLLOW_UP_INFO
followUpInfoValid =
(paramsFromConfirm != NULL) &&
(paramsFromConfirm.peerMacAddress == dot11SlaveMac) &&
(slaveMacOfLastRequest == dot11SlaveMac));
!followUpInfoValid
FOLLOW_UP_INFO_INVALID
requestParams.FollowUpDialogToken = 0;
followUpInfoValid
UCT
INITIATE_REQUEST_WAIT_CONFIRM
If ((++dialogToken % 256) == 0) dialogToken++;
If (nframes_sent == initReqParams.framesPerBurst)
dialogToken = 0;
requestParams.DialogToken=dialogToken;
requestParams.PeerMACAddress = dot11SlaveMac;
setRequestParams(&requestParams, MDSyncSend);
// In the following statement, MinDeltaFTM, which is in units of 100
// microseconds, is converted to UScaledNs (i.e., units of 2-16 ns, see 6.3.3.2)
nextRequestSendTime = currentTime + initReqParams.MinDeltaFTM *
(65536 x 10^5);
MLME-FINETIMINGMSMT.request(requestParams);
requestParams.FollowUpDialogToken = 0; //In case no confirm is received
slaveMacOfLastRequest = dot11SlaveMac;
// In the following statement, the burst duration parameter from 802.11RevMC is
// converted to UScaledNs (i.e., units of 2-16 ns, see 6.3.3.2). The
// quantity 2^(initReqParams.burstDuration – 2) is the burst duration in us.
// Also in the following statement, we are assuming that the burst duration
// starts when the initial FTM request is received. In actuality, the timer begins
// by the partial TSF timer value indicated in the initial FTM frame, which is
// slightly after the initial FTM request is received.
endOfBurstDurationTime = currentTime + 1000 * (2^16) * 250 *
(2^(initReqParams.burstDuration – 2));
portOper && ptpPortEnabled &&
asCapable && currentTime >= nextRequestSendTime &&
nframes_sent <= initReqParams.framesPerBurst &&
currentTime <= endOfBurstDurationTime
rcvdConfirm
SAVE_CONFIRM_INFO
MLME-FINETIMINGMSMT.confirm(&paramsFromConfirm);
requestParams.FollowUpDialogToken = paramsFromConfirm.DialogToken;
requestParams.T1 = paramsFromConfirm.T1;
requestParams.T4 = paramsFromConfirm.T4;
// NOTE: In Fine Timing Measurement, T1 is in units of 0.1 ns.
// upstreamTxTime is units of 2-16 nanoseconds.
K = -3;
// K is 1 for Timing Measurement and -3 for Fine Timing Measurement.
residenceTime = MDSyncSend.rateRatio *
(paramsFromConfirm.T1 * 10^K*(2^16) - MDSyncSend.upstreamTxTime);
requestParams.VendorSpecific.correctionField =
residenceTime + MDSyncSend.followUpCorrectionField;
// NOTE: T1 and T4 are timestamps from a single
// local clock source.
// The roll-over of the 48-bit timestamps returned by
// MLME-FINETIMINGMSMT.request and MLME-FINETIMINGMSMT.indication must be
// accounted for.
// A frame is only counted as being sent, for purposes of number of frames in a burst, if a
// confirm is received. It is up to IEEE Std 802.11 to handle the case where a confirm
// is not received.
nframes_sent += 1;
portOper && ptpPortEnabled && asCapable && (nframes_sent >
initReqParams.framesPerBurst || currentTime > endOfBurstDurationTime)


BEGIN || RESTART || (!porOper || !ptpPortEnabled ||
!asCapable))
INITIALIZING
RESTART=0
if (tmFtmSupport == FTM)
MLME-FINETIMINGMSMTRQ.request
rcvdIndicationTimeoutTime=currentTim
e+(10^7)*(2^16)
//the rcvdIndicationTimeoutTime
interval is 10ms
currentTime>rcvdIndicationTimeoutTime
&&
tmFTMSupport == FTM
(rcvdIndication && tmFTMSupport==FTM)
|| (tmFTMSupport==TM)
DISCARD
previousIndParams = NULL;
rcvdIndication = FALSE;
nframesrcvd=0
rcvdIndication && portEnabledOper &&
pttptpPortEnabled && asCapable
CONSTRUCT_MD_SYNC_RECEIVE_STRUCTURE
if (tmFtmSupport == TM)
MLME-TIMINGMSMT.indication(&indParams);
else if (tmFtmSupport == FTM) {
MLME-FINETIMINGMSMT.indication(&indParams);
nframesrcvd++;
if (nframesrcvd == initReqParams.framesPerBurst) || (currentTime >
endofBurstDurationTime)
RESTART=1;
}
if ((previousIndParams != NULL) &&
(previousIndParams.PeerMacAddress == dot11SlaveMac) &&
(indParams.FollowUpDialogToken != 0))
{
neighborRateRatio =
(indParams.T1-previousIndParams.T1) /
(indParams.T2-previousIndParams.T2);
//NOTE: Other methods of computing neighborRateRatio may can be used.
if (tmFtmSupport == TM)
K = 1;
else if (tmFtmSupport == FTM)
K = -3;
//K = 1 for Timing Measurement and K = -3 for Fine Timing Measurement
neighborPropDelay =
(((indParams.T4 - indParams.T1) -
neighborRateRatio * (indParams.T3 - indParams.T2)) / (2.0)) * (10^K);
//NOTE: Other methods of computing neighborPropDelay may can be used.
MDSyncReceive = setMDSyncReceive(indParams);
MDSyncReceive.VendorSpecific.rateRatio += (neighborRateRatio – 1);
MDSyncReceive.VendorSpecific.upstreamTxTime = indParams.T2* (2^16) *(10^K) -
neighborPropDelay*(2^16)/neighborRateRatio;
//NOTE: Actions performed with the timestampError paramers of indParams are
// implementation independent.
passMDSyncReceiveToPortSync(&MDSyncReceive);
previousIndParams = indParams;
}
previousIndParams = indParams;
rcvdIndication = FALSE;
RESTART
rcvdIndication &&
portEnabledOper &&
pttptpPortEnabled &&
asCapable && !RESTART

Thank you