# Modified Tree-ID Process for Long-haul Transmission and Long PHY_DELAY

~ An Supplemental Root Contention Resolution Method ~

**Takayuki Nyu**
C&C Media Research Laboratories
ＮＥＣ Corporation
4-1-1 Miyazaki Miyamae-ku Kawasaki, 216 Japan
e-mail : new@ccm.cl.nec.co.jp
phone : +81-44-856-2082
fax : +81-44-856-2222

## 1. Introduction

In August, 1997, the model illustrated in Figure 1 was presented to a meeting of the P1394b study group. In it, future sub-/full- PHYs are used to extend the distance between nodes. The sub-/full- PHYs are to be designed to be connectable by way of a short copper cable ( maximum length is 4.5 m ) either to a IEEE 1394-1995 PHY or to a future P1394a PHY, which would be an extension of the current IEEE 1394-1995 PHY. The more function blocks in a future P1394a PHY that are identical to those used in the sub-/full- PHY, the lower the cost of implementing the two PHYs. That is to say, we want to make the commonality between them as high as possible.

In this regard, we should note that sub-/full- PHYs are connected by a long cable, which means that any root contention resolution method will have to be able to cope with the excess latency that results from long-haul transmission and long PHY_DELAY. I propose here a supplemental root contention resolution method to be overlayed on IEEE 1394-1995. With it, root contention will be resolved no matter what future modifications might be made in the allowable maximum cable_delay (distance between nodes ) and PHY_DELAY ( repeat delay in nodes ). Further, because this method is simply overlayed on IEEE 1394-1995, which is then to be extended into P1394a, its addition will not in any way reduce commonality between sub-/full- PHYs and P1394a PHYs.

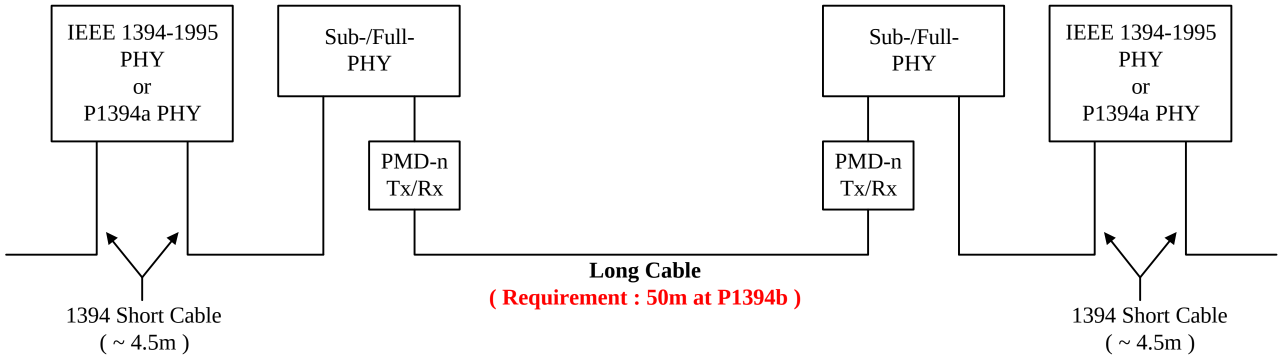


**Figure 1 : A 1394 model using a long cable**

## 2. Root contention resolution method in IEEE 1394-1995

Currently in IEEE 1394-1995, when root contention occurs between two nodes ( hereafter referred to as nodes A and B ), the nodes randomly select and start either their long or short timer. The value of a long timer is referred to as ROOT_CONTEND_SLOW, and the value of a short timer is referred to as ROOT_CONTEND_FAST. When time expires, the node retransmits a PARENT_NOTIFY signal. When node A selects a long timer, and node B selects a short timer, node B retransmits a PARENT_NOTIFY signal earlier than node A, i.e. node A's long timer will still be operating when the PARENT_NOTIFY signal from node B reaches it. Consequently, node A will receive the PARENT_NOTIFY signal the instant that time expires, and root contention will be resolved so long as the two nodes have selected different length timers and the delay between the nodes is small relative to ROOT_CONTEND_FAST. That is to say, in the worst case, even when different length timers have been selected, the following conditions must be met:

Condition 1 : ROOT_CONTEND_FAST > 2∗ cable_delay + phy_delay

Condition 2 : ROOT_CONTEND_SLOW – ROOT_CONTEND_FAST > 2∗ cable_delay + phy_delay,

where cable_delay is the propagation delay between nodes A and B, and phy_delay is the repeat delay in a node[1].

In IEEE 1394-1995, ROOT_CONTEND_SLOW is 570ns ~ 600ns, ROOT_CONTEND_FAST is 240ns ~ 260ns, and PHY_DELAY must be less than 144ns. If PHY_DELAY is equal to 144ns, Condition 1 will not be met if cable_delay is longer than 48 ns, which corresponds to an approximate cable length of 8 m, i.e. root contention will not be resolved in this worst case scenario. Figure 2 shows a case in which node A has selected its long timer and node B its short timer. We assume that the sum of cable_delay and PHY_DELAY is 200ns; the result is a bus reset.

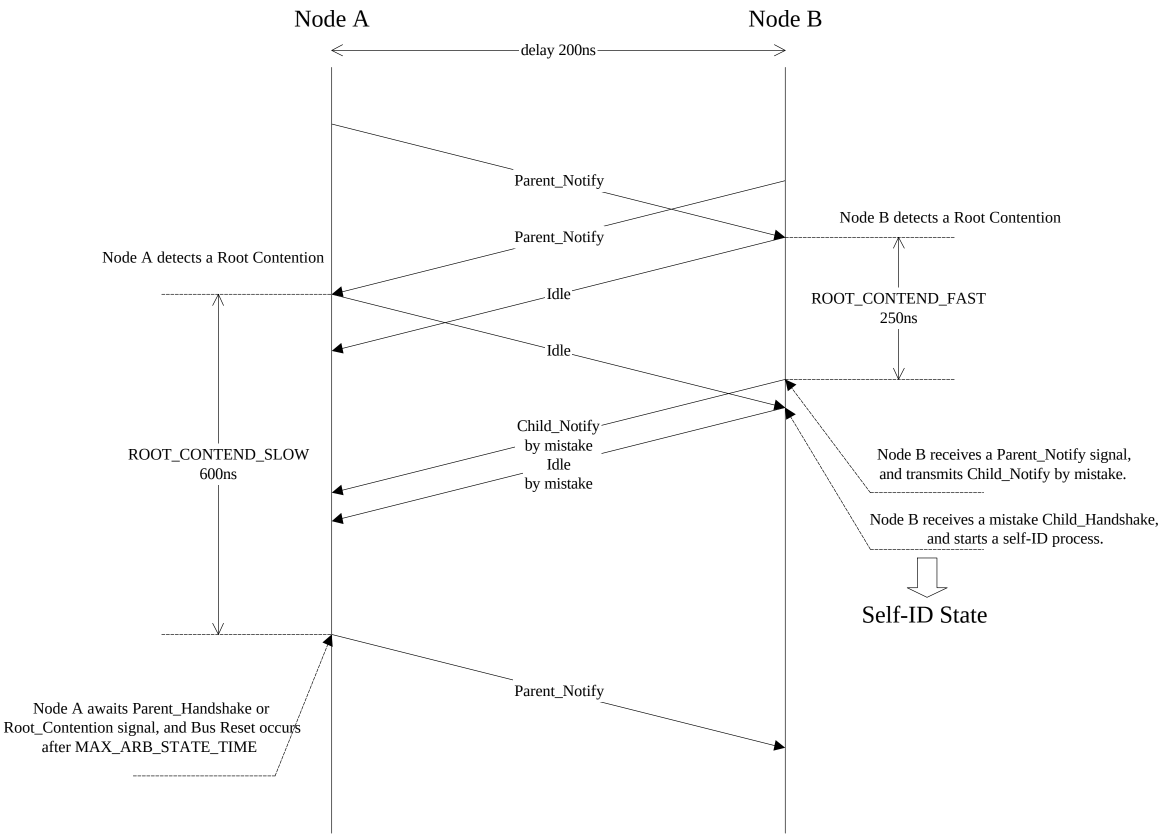


**Figure 2 : Cases of failure to resolve root contention**
( Node A selects long timer and node B selects short timer )

## 3. Root contention timer in P1394a Draft 1.0

The range of values for root contention timers, i.e. ROOT_CONTEND_FAST and ROOT_CONTEND_SLOW, is being changed in P1394a draft 1.0 in order to overcome the above problem. New values are to be: ROOT_CONTEND_FAST = 760ns ~ 800ns and ROOT_CONTEND_SLOW = 1600ns ~ 1640ns. In terms of Condition 1, if it is assumed that PHY_DELAY is equal to 144ns, allowable cable_delay will be 308ns, which corresponds to an approximate cable length of 56m, and in terms of Condition 2, will be 328ns, which corresponds to an approximate cable length of 59m. Therefore, root contention will be resolved if the distance between nodes is less than 56m. PHY_DELAY is variable , however, determined by the self-id packet, and if the maximum PHY_DELAY of 444ns were selected, Condition 1 would not be met if cable_delay were longer than 158 ns, i.e. root contention would still not be resolved if transmission were longer than approximately 28 m.

## 4. Proposal of supplemental process for root contention resolution

One approach to this problem is to increase the value of the timers even further; another is to resolve root contention with a method that is independent both of the distance between nodes and of the repeat delay in them. That is what I propose here. The basic process is as follows:

1st : The two nodes detect both the duration of the tx_parent_notify ( tx_prop_time ) and the duration both of the rx_parent_notify and of rx_root_contention ( rx_prop_time ).

2nd : The nodes compare tx_prop_time to rx_prop_time.

3rd : A node whose tx_prop_time is less than rx_prop_time will take the role of bus root, with the other node becoming a child node. If tx_prop_time should happen to equal rx_prop_time for both nodes, each will randomly choose to set its back-off timer either to zero or to BACK_OFF_TIME, and each will retransmit a PARENT_NOTIFY signal, either at the instant of a zero-set or at the expiration of a BACK_OFF_TIME, as the case may be.

## 5. Modified tree identify process

We have created a PHY timing constant, referred to as LONG_DIST_TIME, in order to make our method compatible with existing root contention resolution. A state T4 is added to the existing Tree-ID state machine. If either the duration of tx_parent_notify or the duration both of rx_parent_notify and of rx_root_contention exceeds LONG_DIST_TIME, root contention resolution will be switched to the proposed method, which has no influence on the handshake process for times of less than LONG_DIST_TIME.

Figure 3 shows an example. It was assumed that the sum of cable_delay and PHY_DELAY is 300 ns and that LONG_DIST_TIME is 200ns. Node A detects both $\alpha$ and $\beta$ as both tx_prop_time and rx_prop_time respectively. Node B detects both $\alpha$ and $\beta$, as rx_prop_time and tx_prop_time, respectively. Since $\alpha$ is longer than $\beta$ in this example, node A will be root and node B will be child.

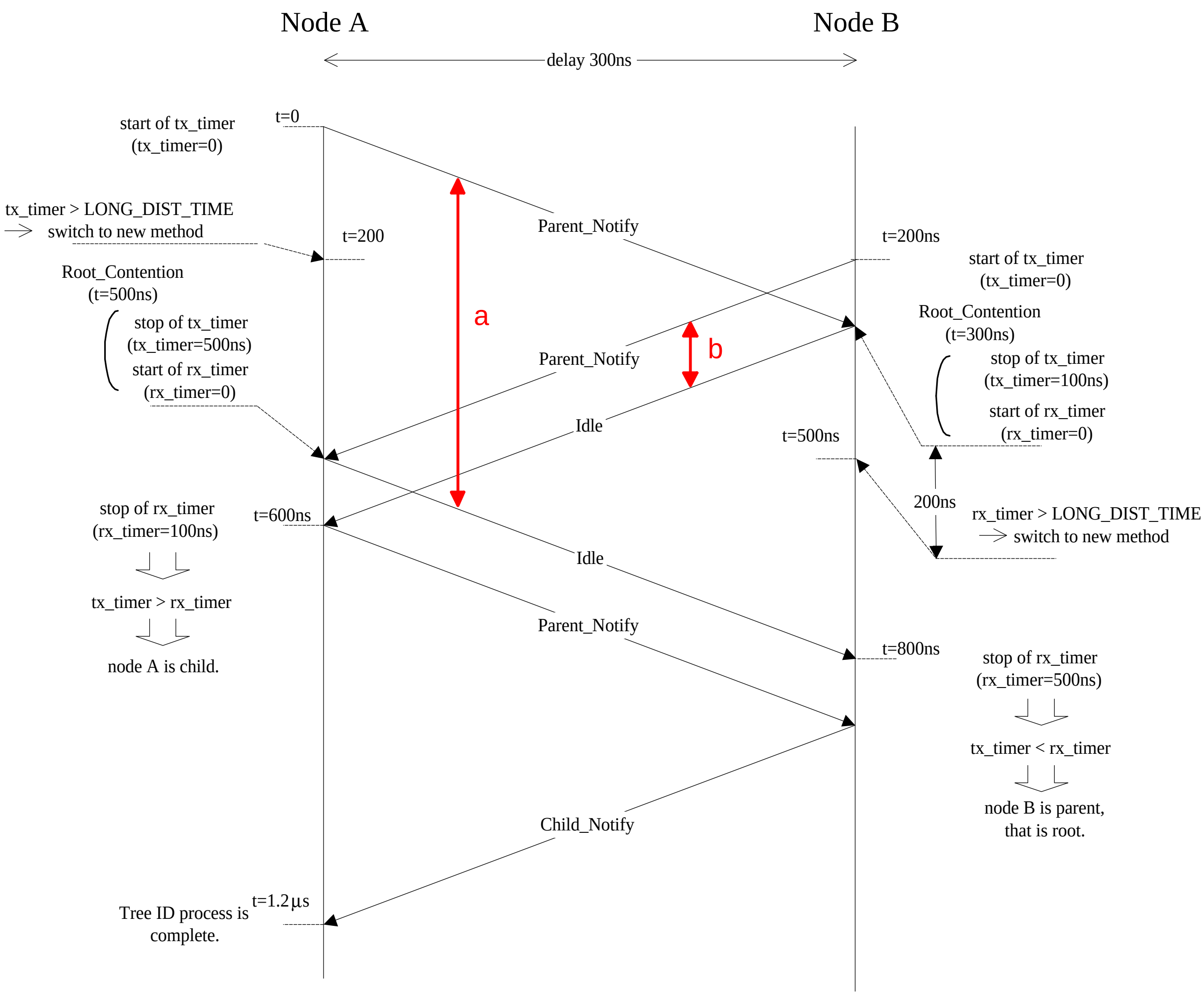


**Figure 3: Root contention resolution with the proposed method**

### *5.1. Additional PHY timing constants*

The following condition must hold in order to properly use both the proposed method and the existing method.

LONG_DIST_TIME < ROOT_CONTEND_FAST

LONG_DIST_TIME should be set to the closest value to ROOT_CONTEND_FAST in order to use the existing method as much as possible. Since ROOT_CONTEND_FAST in IEEE 1394-1995 is 240 ns ~ 260 ns, LONG_DIST_TIME could be set to 200 ns.

If root contention occurs precisely midway between nodes, the duration of tx_parent_notify will equal the duration both of rx_parent_notify and of rx_root_contention, in which case the two nodes will retransmit PARENT_NOTIFY signals at different times, each of which may be referred to as that node's long_contend_time. This will resolve root contention. Long_contend_time is set either to zero or to BACK_OFF_TIME, for which two or three clock times will be enough.

**Table 1 : Additional PHY timing constants**

| Timing constant | Minimum | Maximum | Comment |
|---|---|---|---|
| LONG_DIST_TIME | 160ns | 200ns | Timing of transition from existing resolution method of root contention to proposed method ( 20 / base_rate ) |
| BACK_OFF_TIME | 40ns | 80ns | Time to wait in state T4 before transition to state T2. ( 8 / base_rate ) |

### *5.2. Additional cable PHY code definition*

**Table 2 : Additional cable PHY code definition**

```
timer    tx_timer;          // timer for measuring the duration of tx_parent_notify
timer    rx_timer;          // timer for measuring the duration of rx_parent_notify
int    tx_prop_time;     // the duration of tx_parent_notify
int    rx_prop_time      // the duration of rx_parent_notify
boolean    rx_timer_off    // set when reception of rx_parent_notify is complete
boolean    long_delay    // reset when a node in state T3 receives rx_idle at state T3 and
                              rx_timer is less than LONG_DIST_TIME.
boolean    contention_in_PHY    // set when contention occurs in a PHY
baserate_time    long_contend_time    // amount of time to wait for re-start tx_parent_notify
```

### *5.3. Tree-ID state machine*

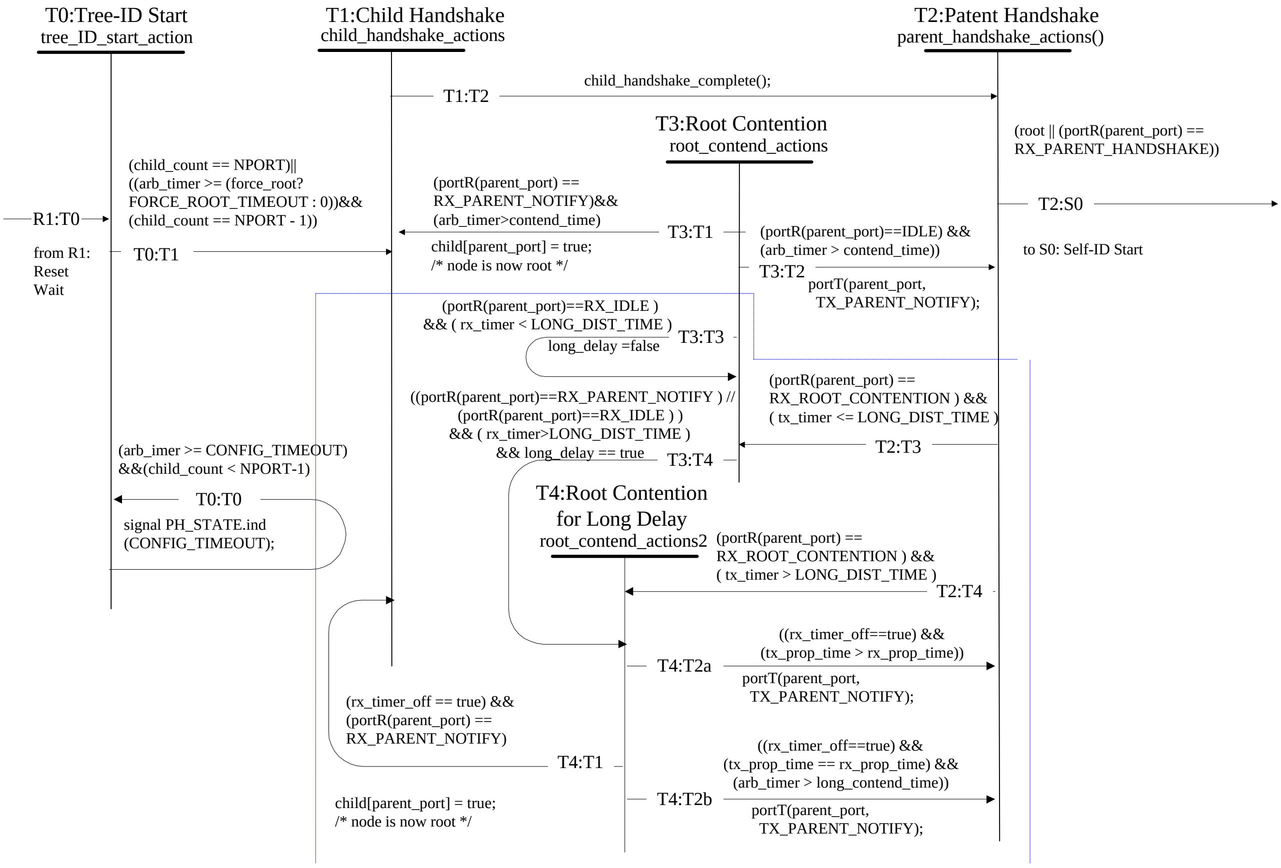


**Figure 4 : Modified Tree-ID state machine**

The section enclosed by dashed lines represents the added method.

#### 5.3.1. Tree-ID state machine notes

**State T4 : Root Contention for Long Delay.** In this state, both nodes compare the duration of tx_parent_notify ( tx_prop_time ) and the duration both of rx_parent_notify and of rx_root_contention ( rx_prop_time ). If tx_prop_time is longer than rx_prop_time, the node will be a child node. If rx_prop_time is longer than tx_prop_time, the node will be a parent node ( i.e. a root ). If tx_prop_time is equal to rx_prop_time, PARENT_NOTIFY signal will be retransmitted after respective long_contend_times.

**Transition T3 : T3.** If a node receives an IDLE signal when rx_timer is less than LONG_DIST_TIME, the long_delay flag is set to false.

**Transition T3 : T4.** If a node receives an IDLE or PARENT_NOTIFY signal and the long_delay flag is set to true when rx_timer is longer than LONG_DIST_TIME, it changes the method of resolution from the existing to the added.

**Transition T4 : T1.** If rx_prop_time is longer than tx_prop_time and the node receives a rx_parent_notify, the node takes on the role of a bus root.

**Transition T4 : T2a.** If tx_prop_time is longer than rx_prop_time, the node once again sends a PARENT_NOTIFY signal.

**Transition T4 : T2b.** If tx_prop_time is equal to rx_prop_time, each node node waits for long_contend_time in state T4 and then once again sends a PARENT_NOTIFY signal.

**Transition T2 : T4.** If root contention is detected when tx_timer is longer than LONG_DIST_TIME, a node changes the method of resolution for the existing to the added.

### 5.3.2. Tree-ID actions and conditions

**Table 3 : modified IEEE 1394-1995 Table 4-45**

```
void tree_ID_start_actions() {
int i, temp_count;
long_delay = false;                    // set long_delay flag
contention_in_PHY = false;             // set contention_in_PHY flag
arb_timer = 0;                         // start timer
while(true) {                          // loop forever
      temp_count = 0;                  // temporary child counter
      for (i = 0; i < NPORT; i++)
            if (~connected[i] | | portR(i) == RX_PARENT_NOTIFY) {
                                       // when unconnected or receiving "you are my
parent"
                  child[i] = true;     // set child flag
                  temp_count++;        // and increment counter
      child_count = temp_count;        // set current child count
       }                               // end of forever loop
}
void child_handshake_actions() {
int i;
root = true;                           // root will stay true if all ports are child ports
for (i = 0; i < NPORT; i++) {
      if (connected[i] && child[i])
            portT(i, TX_CHILD_NOTIFY);     // you are my child
      else if (connected[i]) {
            portT(i, TX_PARENT_NOTIFY);    // you are my parent
            parent_port = i;           // there is at most one port with child==false
            root = false;              // cannot be root since there is a parent
      }
}
boolean child_handshake_complete() {   // true id all active children in "Start Self_ID"
int i;
for (i = 0; i < NPORT; i++)
      if (child[i] && connected[i] && (portR(i) != RX_CHILD_HANDSHAKE)
            return false;              // active child not giving "you are my parent"
return true;                           // will also be true if there are no active
children
}
void parent_handshake_actions(){
tx_timer = 0;                  // start transmition timer
if   ( portR(parent_port) == RX_PARENT_NOTIFY ) {
```

```
        rx_timer = 0;
       contention_in_PHY = true;
}
}
void root_contend_actions() {
int i;
contend_time = (random_bool() ? CONTEND_SLOW : CONTEND_FAST);
long_delay = true;        // set  long delay flag
for (i = 0; i < NPORT; i++) {
      if (child[i])
            portT(i, TX_CHILD_NOTIFY);       // you are my child
      else
            portT(i, IDLE);                        // abandon "you are my parent" request
      }
arb_timer = 0;                                    // start arbitration timer
if ( contention_in_PHY == false ) {
rx_timer = 0;                        // start reception timer
}
tx_prop_time = tx_timer;
}

void root_contend_actions2( ) {
long_contend_time = ( random_bool( ) ? 0 : BACK_OFF_TIME );
if ( long_delay == false ){
      tx_prop_time = tx_timer;       // set transmission time
}
for (i = 0; i < NPORT; i++) {
      if (child[i])
            portT(i, TX_CHILD_NOTIFY);       // you are my child
      else
            portT(i, IDLE);                  // abandon "you are my parent" request
      }
if   ( long_delay == false ) {
         rx_timer = 0;          // start reception timer
}
while ( portR(parent_port) == RX_PARENT_NOTIFY ) {
         rx_timer_off = false;
}
if ( portR(parent_port) == IDLE ) {
         rx_timer_off = true;
         rx_prop_time = rx_timer;
         arb_timer = 0;          // start arbitration timer
}
```

## 6. Advantages of proposed method

Figure 5 shows an example of the use of the existing root contention method alone. It is assumed, as it was in the example illustrated in Figure 3, that delay between nodes A and B is 300 ns. 1 µs and 2 µs were chosen, respectively, for ROOT_CONTEND_FAST and ROOT_CONTEND_SLOW, so that root contention will be resolved even when PHY_DELAY is 444 ns and cable length is 50 m. Nodes A and B wait until their timers expire for, respectively, a W1 of 0.9 µs and a W2 of 0.5 µs. The Tree-ID process is complete at 2.8 µs. In this example, 1.6 µs more time is consumed than was with the proposed method applied in the example in Figure 3.

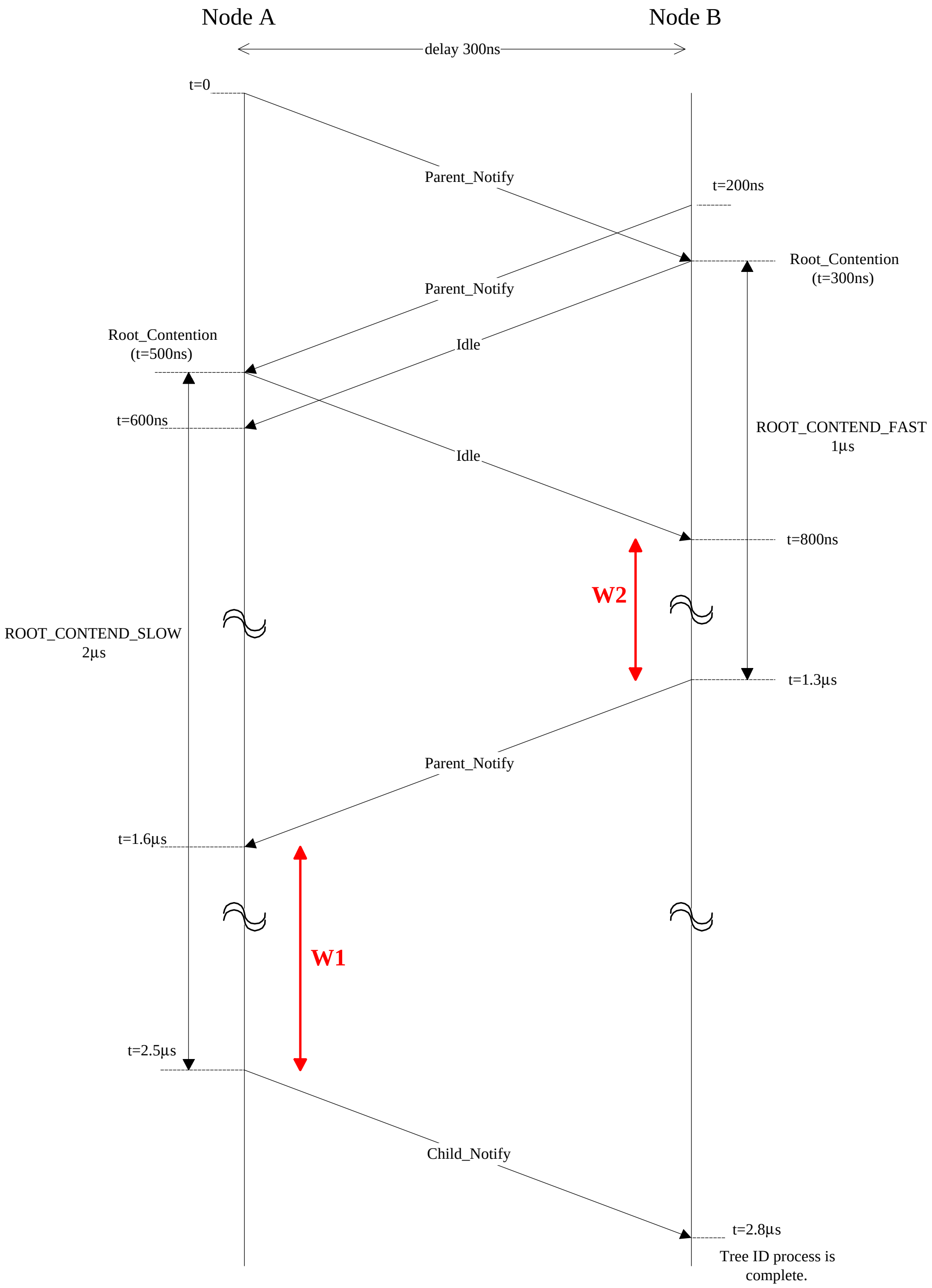


**Figure 5 : Using existing method only.**
( ROOT_CONTNED_FAST=1us, ROOT_CONTNEND_SLOW=2us )

## 7. Conclusion

I have described the weakness of the existing root contention resolution method in IEEE 1394-1995 and P1394a under conditions of long delay between nodes, and proposed a supplemental root contention resolution method that is independent of the delay between nodes. The proposed method appears to be very promising for accommodating long distance cable ( >50m ) and long PHY_DELAY.

## Reference


1. Dave LaFollette, “ SubPhy Root Contention “, 1997 Aug. 1
2. IEEE P1394a Draft 1.0
3. IEEE std 1394-1995