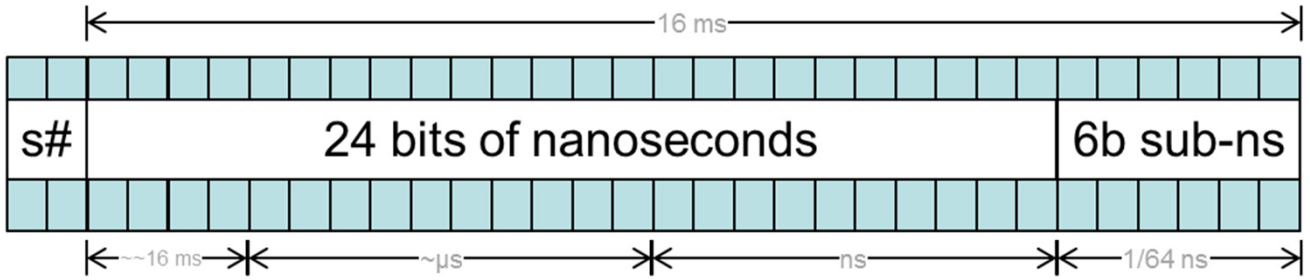
RoE timestamp and presentation time in past

Jouni Korhonen
Broadcom Ltd.
5/26/2016

Background

- RoE '2:24:6' timestamp was recently (see tf3_1604_bross_timestamp_4.pdf) approved as:



- Also stating:
  - Benefits of this timestamp
    - 16 ms range covers 1 radio frame
    - Precision down to ~16 ps accuracy (1/64 ns)
  - 2-bit sequence number at top allows detection of up to 3 missed packets
- This is fine.

Background cont'd

- The RoE specification says timestamp (=presentation time) is:
  - "The RoE presentation time is used to achieve time synchronization between the RoE endpoints. The presentation time is calculated by the RoE sender and represents the time when the RoE packet payload has to be played out from the RoE receiver packet buffer to the consumer of the payload data."
- So far so good..

How about late packets?

- The on-wire timestamp is encoded as 2^24 ns * 2^6 -> total 30 bits.
- This represents a timestamp that is 0 to ~16ms ahead of the current node ToD when calculated at the receiver..
- What if the packet containing the timestamp arrives **past** the presentation time it carries i.e., packet **arrives late**??

- Issue: Current mechanism cannot detect late packets properly.

- Late packet should never happen BUT errors happen and detecting an error sitoation would be a useful if not a mandatory feature!

How to correct the situation?

- One could always treat the timestamp as a +-8ms value..
  - Pros: does not change anything at the sender as long as the presentation time is less than 8ms ahead of time.
  - Cons: the timestamp cannot even point beyond one 10ms radio frame.. not good.

- One could define a 'acceptable timestamp window' value that is between 0 to ~16ms. Values less than 'acceptable timestamp window' are in time and those that are greater are considered late.
  - This is based on the wrap around properties when calculating the presentation time from the timestamp at the receiver (examples will follow).

Acceptable timestamp window

- A programmable value between 0 to ~16ms. Defaults to a value that forces deployments to put something meaningful into it. Example: zero (0).
- The amount of time reserved for 'past timestamps' must be large enough to cover hiccups and jitter in the network.
- Example window partitioning:
  - 12ms acceptable window, 4ms late window.
  - 14ms acceptable window, 2ms late window.
  - 16ms acceptable window, no late packets detected.

- No changes to the on-wire format!
- No changes to the sender!
- The receiver end needs to implement the 'acceptable timestamp window' variable.
- Receiver has to make a comparison to detect whether the timestamp is within the acceptable window.

How timestamp conversion behaves..

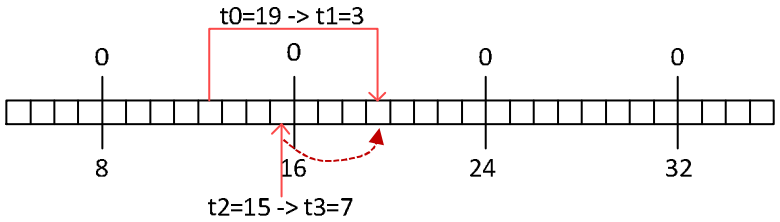
- Calculating on-wire timestamp is simple at the sender side – just cut the low 30 bits of the presentation time.
  - The assumption is that the presentation time is in format: 2^{n} ns * 2^{6}, where n is max number of bit reserved for the internal time representation. The minimum n is 24.
- Example algorithm:
  - n is 58 (time would be total 64 bit number)
  - `timestamp = presentationTime & TIMESTAMPMASK;`

How timestamp conversion behaves..

- Example algorithm to calculate the presentation time from the on-wire timestamp based on the local node's time is as follows:
  - `t0 = presentation time at the sender`
  - `t1 = 30 bit on-wire timestamp`
  - `t2 = local time at the receiver`
  - `t3 = 30 low bits of t2`
  - `delta = t1 - t3`
  - `handle window wrap: delta &= TIMESTAMPMASK`
  - `presentation time at the receiver: t2 + delta`

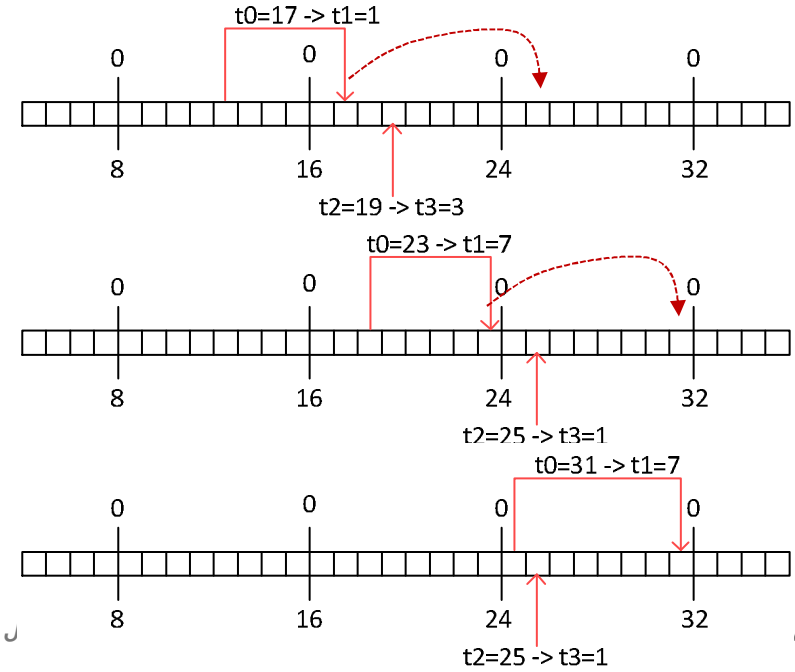
How does the window wrap work?

- For example: **window=3 -> mask=7, acceptable window=5**
- presentationTime is in time:




```
delta  = (3-7)&7 = 4
presTS = t2+delta = 15+4=19
```

- presentationTime is late or too far ahead:




```
delta  = (1-3)&7 = 6 (>accp.)
presTS = t2+delta = 19+6=25
```

```
delta  = (7-1)&7 = 6 (>accp.)
presTS = t2+delta = 25+6=31
```

```
delta  = (7-1)&7 = 6 (>accp.)
presTS = t2+delta = 25+6=31
```

Example algorithm 1/2

```
#define WINMSKFULL  0x000000ffffffffffULL   // 24 bit ns, 16 bit sub-ns
#define VALIDWIN    0x000000dfffffffffULL   // valid to +14ms, late -2ms
#define P2WSHFT     (16-6)

// Time expressed in units of nanoseconds and multiplied by 2^16.
typedef uint64_t iScaledNanoseconds;

// on-wire timestamp is ~16.8ms
typedef struct {
    uint32_t seqnum : 2;                //
    uint32_t scaledNanoseconds : 30;    // 24 bits ns * 2^6
} wireTimestamp;

// Convert the iScaledNanoseconds to the on-wire timestamp.
wireTimestamp time_to_wire( iScaledNanoseconds prests, uint32_t pField )
{
    wireTimestamp w;
    w.seqnum = pField;
    w.scaledNanoseconds = (prests & WINMSKFULL) >> P2WSHFT;
    return w;
}
```

Example algorithm 2/2

```
// Acceptable timestamp window variable with some initial value
static iScaledNanoseconds acceptableWin = VALIDWIN;

// Convert the 32 bit on-wire timestamp to the iScaledNanoseconds
// i.e., to the presentation time. Ignore the sequence number.
// Inputs:
//   localts = ptr to node's current time
//   wts = ptr to timestamp
// Returns:
//   localts = presentation time (overrides the input local time)
//   1 if presentation time is within the valid time window
//   0 if presentation time is in past
int wire_to_time( iScaledNanoseconds* localts, wireTimestamp* wts )
{
    iScaledNanoseconds prests, delta;
    prests = (iScaledNanoseconds)wts->scaledNanoseconds << P2WSHFT;
    delta = (prests - *localts) & WINMSKFULL;
    *localts += delta;
    return delta <= acceptableWin;
}
```

Compare to plain +16ms algorithm..

```
// Convert the 32 bit on-wire timestamp to the iScaledNanoseconds
// i.e., to the presentation time. Ignore the sequence number.
// Returns:
//   presentation time
int wire_to_time( iScaledNanoseconds localts, wireTimestamp* wts )
{
    iScaledNanoseconds prests, delta;
    prests = (iScaledNanoseconds)wts->scaledNanoseconds << P2WSHFT;
    delta = (prests - localts) & WINMSKFULL;
    return localts + delta;
}
```

The extra from the window check is one more comparison!

Proposal

- Introduce a new configuration variable that holds the ‘acceptable timestamp window’ value (at the receiver).

Comments?