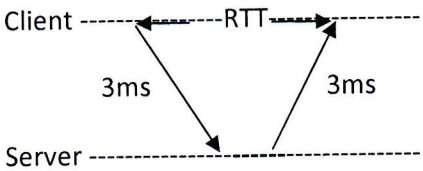
1a.

The time at the client will be set to CTC time coming back from the server, plus (RTT/2). Since {RTT/2) is the error, you want to pick the query with the minimum RTT, namely Query #2 that took 28ms.

The client should set its time to the time from the server for Query #2, 2:32:11.999, plus (RTT/2). This gives 2:32:11.999 + 28/2 which gives:

Time at client = 2:32:11.999 + .014 = 2:32:12.013

1b.



Time and accuracy are different. Assume minimum times for the request and reply are 3ms. IF the response from the server to client took 3ms (the minimum), then the client would set its time to the time from the server plus 3ms.

timeAtClient = timeAtServer + 3ms

This would constitute a lower bound. If the request from the client to the server took 3ms (the minimum), then this means that the response from the server would take RTT - 3ms, so the time at the client would be:

timeAtClient = timeAtServer + RTT – 3ms

So, the interval in which the client sets it time is:

[timeAtServer + 3ms, timeAtServer + RTT – 3ms] (Call this expression A)

If the time is set to be halfway through this interval, i.e.:

(timeAtServer + RTT – 3 + timeAtServer + 3) / 2

= timeAtServer + (RTT – 3 + 3)/2

This means that the error is half the size of the interval:

error = (timeAtServer + RTT – 3 – timeAtServer – 3ms) / 2 (Call this expression B)

error = (RTT – 6) / 2 = RTT/2 – 3 ms

Let min1 = time from client to server

Let min2 = time from server to client

For this case, min1 = min2 = min = 3ms

error = + - (RTT/2) – min

1c.

If min1 = 5ms and min2 = 3ms, then using expressions A and B and the definitions for min1 and min2 from part 1b, the interval and error are:

interval = [timeAtServer + 3, timeAtServer + RTT – 5]

error = (RTT – min1 – min2) / 2 = (RTT – 3 – 5) / 2 = (RTT – 8) / 2 which gives:

error = + - (RTT/2) - 4

2a.

Clock 1 ticks 1000 times per ms and clock 2 ticks 980 times per ms and UTC updates every minute

After 1 ms, the skew is 20 ms = .02 seconds, so after a minute, i.e., 60 seconds, the skew would be:

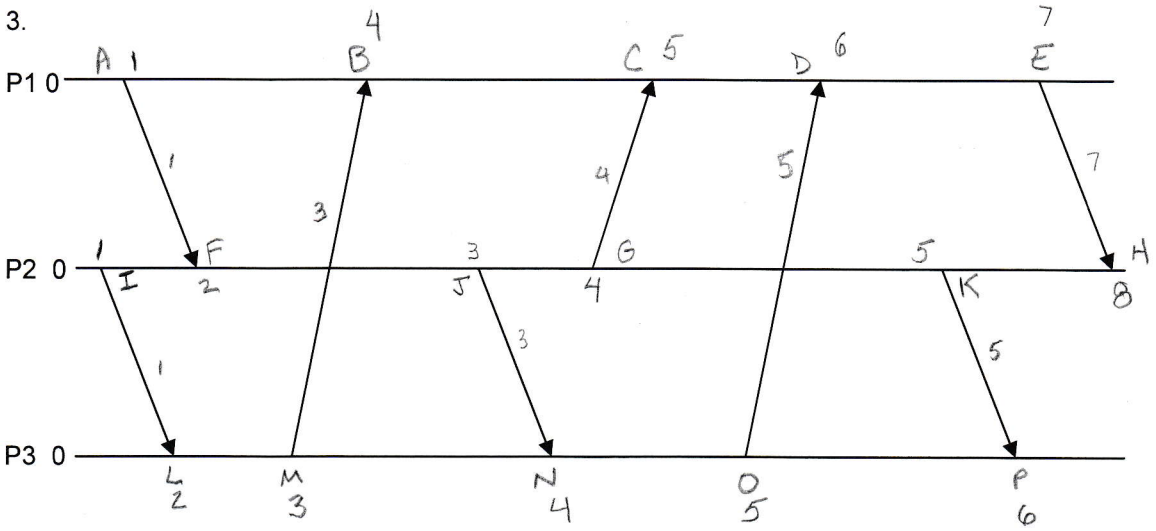
Maximum Skew = .02 * .60 = 1.2 secs

2b.

For maximum skew of 180 ms = .180 seconds:

$.02x = .180 \rightarrow$ **x = 9 seconds**

3.



P1, P2, P3 all start out at 0

I: 1 (increment on send)

A: 1 (increment on send)

M: 3 (increment on send)

J: 3 (increment on send)

G: 4 (increment on send)

O: 5 (increment on send)

K: 5 (increment on send)

E: 7 (increment on send)

L: max(1, 0) + 1 = 2

F: max(1,1) + 1 = 2

B: max(3, 1) + 1 = 4

N: max(3, 3) + 1 = 4

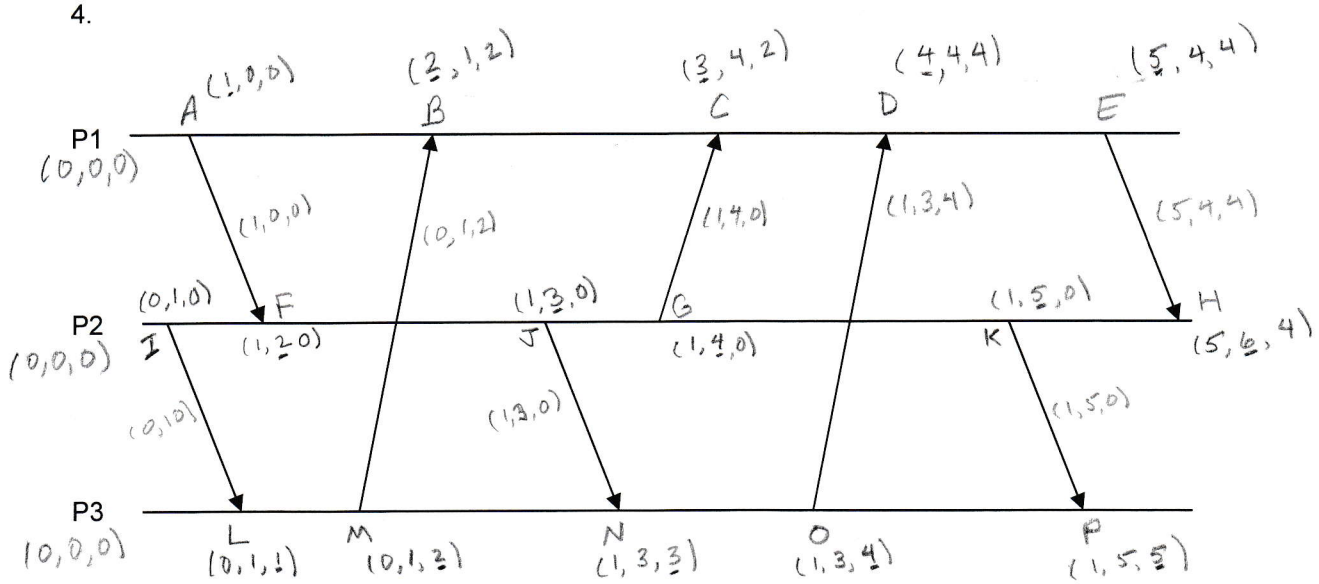
C: max(4, 4) + 1 = 5

D: max(5, 5) + 1 = 6

P: max(5, 5) + 1 = 6

H: max(7, 5) + 1 = 8

4.



P1, P2 and P3 start out at (0, 0, 0)

I: (0, 1, 0) L: (0, 1, 1)

For I which is sending, increment 2nd element of current vector, (0, 0, 0), from 0 to 1 since occurred on P2. This gives (0, 1, 0). Send this to the L.

For the receiving end at L, increment the third element of current vector, (0, 0, 0) since L is on P3, For the other 2 elements of the vector, take the max of the corresponding elements in the current vector, (0, 0, 0) and the coming in from I which is (0, 1, 0). This gives (max(0,0), max(0, 1), 1) which is (0, 1, 1). Note that this now makes (0, 1, 0) the new current vector for P2. The others are calculated in a similar fashion.

A: (1, 0, 0) F: (1, 2, 0)

M: (0, 1, 2) B: (2, 1, 2)

J: (1, 3, 0) N: (1, 3, 3)

G: (1, 4, 0) C: (3, 4, 2)

O: (1, 3, 4) D: (4, 4, 4)

K: (1, 5, 0) P: (1, 5, 5)

E: (5, 4, 4) H: (5, 6, 4)

5.

A's vector = V1 = (3, 2, 5)

B's vector = V2 = (6, 5, 5)

C's vector = V3 = (4, 4, 7)

V1 <= V2 since 3 < 6 and 2 < 5 and 5 <= 5 (Call this expression D)

V1 < V3 since 3 < 4 and 2 < 4 and 5 < 7 (Call this expression E)

D means that A happened before B and E means that A happened before C. So, this means that A happened before both B and C so A chimed first.

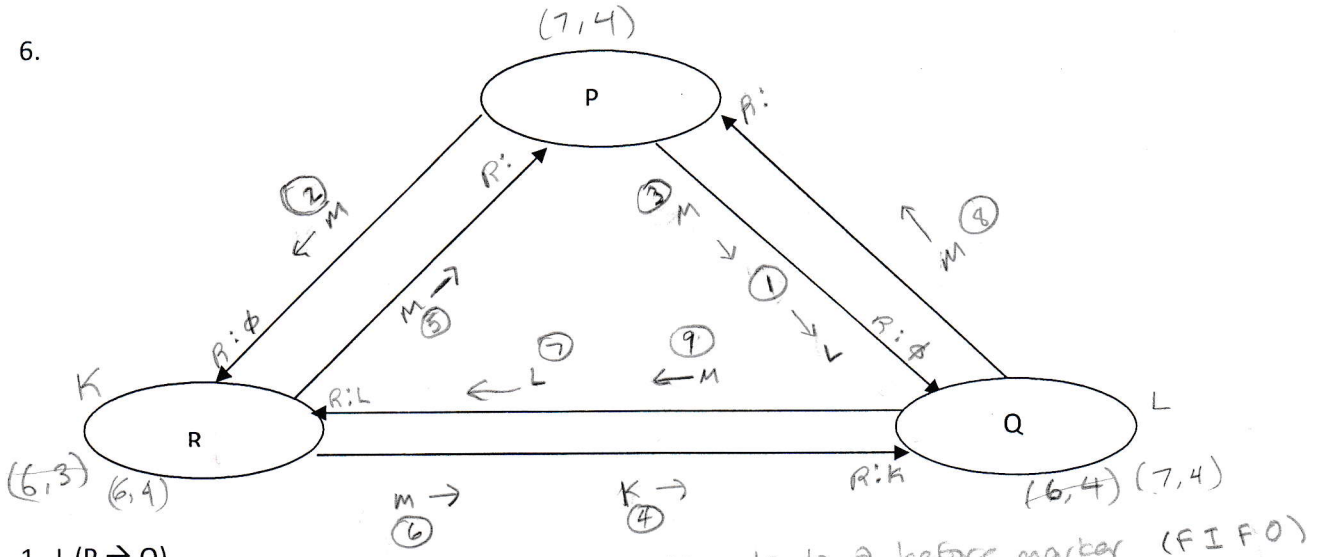
However, consider V2 and V3

Is V2 <= V3? → Is 6 <=4 (F) and 5 <= 4 (F) and 5 <= 7 (T) → F and F and T gives False so V2 is NOT less than or equal to V3.

Is V3 <= V2? → Is 4 <= 6? (T) and 4 <= 7 (T) and 7<=5 (F) → T and T and F gives False so V3 is NOT less than or equal to V2.

This means that processes B and C are concurrent and information can be gained as to the order of these events.

6.



1. L (P → Q)
2. M (P → R)
3. M (P → Q)
4. K (R → Q)
5. M (R → P)
6. M (R → Q)
7. L (Q → R)
8. M (Q → P)
9. M (Q → R)

P saves its state, (7, 4). P starts recording on Q → P and R → P. P sends markers on P → R and P → Q. Note that the marker is behind L (FIFO) so L will get to Q before the marker.

L gets to Q. Q's counters are updated per the application to (7, 4). Q is not in the algorithm yet since it didn't receive a marker

K leaves R to go to Q; R's counters are updated per the application to (6, 4). R is not in the algorithm yet since it didn't receive a marker

R gets marker from P. **R saves its state, (6, 4)**. R stops recording on P → R, that is channel **P → R is <>**. R starts recording on Q → R. R sends markers on R → P and R → Q.

L leaves Q for R

Q gets marker from P. **Q saves its state, (7, 4)**. Q stops recording on P → Q, that is, channel **P →Q is <>**. Q starts recording on R → Q. Q sends markers on Q → P and Q → R.

K gets to Q from R. Q already saved state so it will record state of channel R → Q as set of messages it has received over R → Q since it saved state. Channel R → Q records "K".

P gets marker from R. P already saved state so it will record state of channel R → P as set of messages it has received over R → P since it saved state. No application messages since last save so channel **R → P records < >**.

Q gets marker from R. Q already saved state so it will record state of channel R → Q as set of messages it has received over R → Q since it saved state. **Channel R → Q records "K".**

L gets to R from Q. R already saved state so it will record state of channel Q → R as set of messages it has received over Q → R since it saved state. **Channel Q → R records "L".**

P gets marker from Q. P already saved state so it will record state of channel Q → P as set of messages it has received over Q → P since it saved state. No application messages since last save so channel **Q → P records <>**

R gets marker from Q. R already saved state to it will record state of channel Q → R as set of messages it has received over Q → R since it saved state. Channel Q → R records "L"

Result:

P: (7, 4)
Q: (7, 4) and channel R → Q records "K" (in transit from R to Q)
R: (6, 4) and channel Q → R records "L" (in transit from Q to R)
Channels Q → P and R → P, P → R, P → Q are <>