CSCI/ECEN 5673: Distributed Systems
Fall 2016
Homework 1
Due Date: 09/18/2016

Write your answers in the space provided. Please DO NOT use extra space, as the space provided is sufficient for answering the questions.

There are three problems. For problem 3 (programming problem), you may work with one of your classmate (team of size 2) to write the program and conduct the experiments. Please write the name of your teammate in this case. This means your answer to problem 3, question 1 will be same as your teammate's answer. However, please provide your own answers to questions 2 and 3 of this problem.

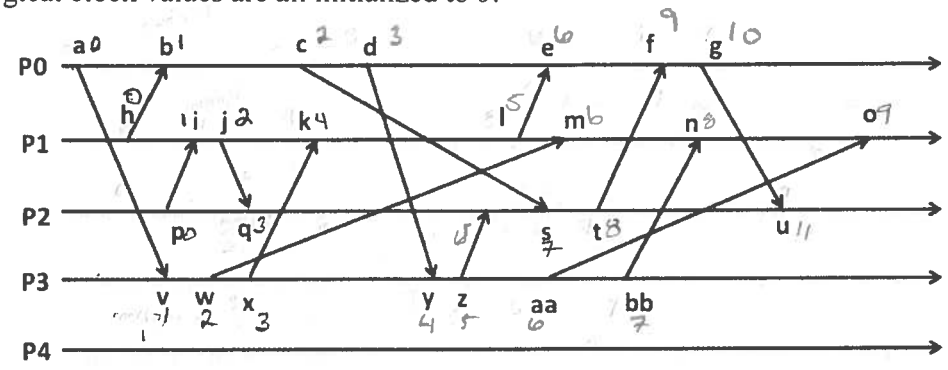
Topics covered: Limitations of distributed systems, happened-before relation, logical clocks, vector clocks, clock synchronization and message ordering.
Lecture Sets: zero, one, two and three.

Grade: 5% of your final grade.

Honor Code Pledge: On my honor, as a University of Colorado at Boulder student, I have neither given nor received unauthorized assistance on this work.

Name: Diana Southard

1. Consider the following figure that shows five processes ***(P0, P1, P2, P3, P4)*** with events ***a, b, c,*** ... and messages communicating between them. Assume that initial logical clock values are all initialized to ***0***.



a) **[6 Points]** Provide logical clock (C) values of each event shown.

- incremented between two successive events on that process
- max (C_j, t_{msg} + 1)
- message gets time stamp = current clock value
- when received: receiving process first updates its clock value, then assigns it to the event

a = 0
b: max(0, 1) = 1
c: 2
d: 3
e: max(3, 6) = 6
f: max(6, 8) = 8
g: 9

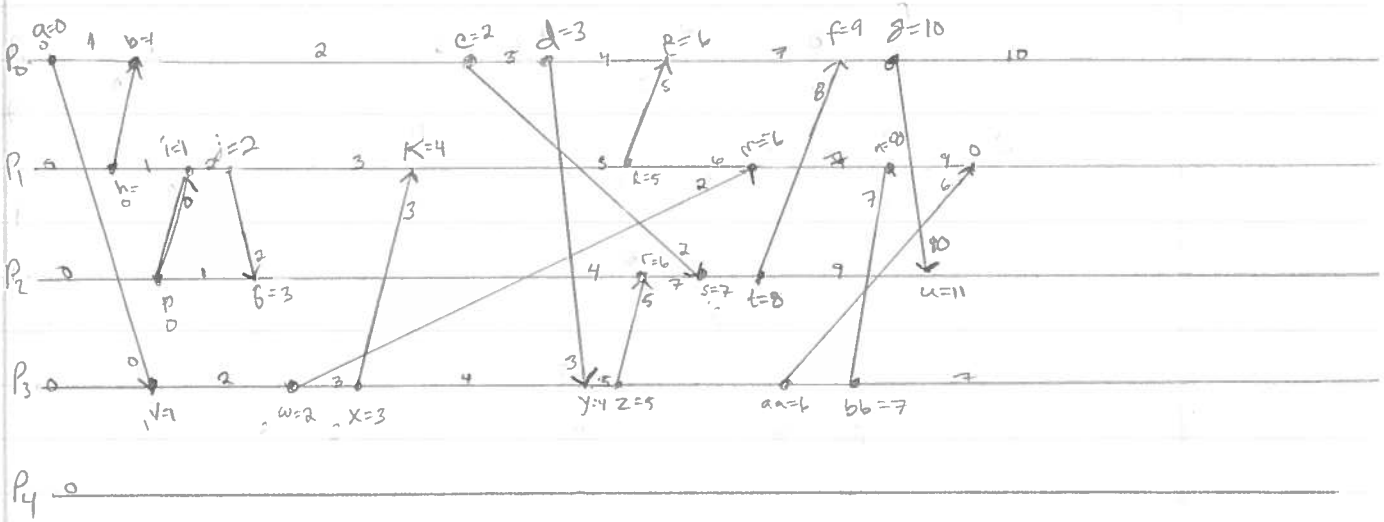
h: 0
i: max(0, 1) = 1
j: 2
k: max(2, 4) = 4
l: 5

m: max(5, 3) = 5
n: max(5, 8) = 8
o: max(8, 7) = 8
p: 0
q: max(0, 3) = 3
r: max(3, 6) = 6
s: max(6, 4) = 6
t: 7
u: max(7, 10) = 10
v: max(0, 1) = 1
w: 2
x: 3
y: max(3, 4) = 4
z: 5
aa: 6
bb: 7

$C_0 = 9$
$C_1 = 8$
$C_1 = 8$
$C_2 = 10$
$C_3 = 7$
$C_4 = 0$

Please see next page

Homework 1



a.)

Clock Values

C_0	C_1	C_2	C_3	C_4
$a = 0$	$h = 0$	$p = 0$	$v = \max(0, 0+1) = 1$	
$b = \max(0, 0+1) = 1$	$i = \max(1, 0+1) = 1$	$q = \max(1, 2+1) = 3$	$w = 2$	
$c = 2$	$j = 2$	$r = \max(4, 5+1) = 6$	$x = 3$	
$d = 3$	$k = \max(3, 3+1) = 4$	$s = \max(7, 2+1) = 7$	$y = \max(4, 3+1) = 4$	
$e = \max(4, 5+1) = 6$	$l = 5$	$t = 8$	$z = 5$	
$f = \max(7, 8+1) = 9$	$m = \max(6, 2+1) = 6$	$u = \max(9, 10+1) = 11$	$aa = 6$	
$g = 10$	$n = \max(7, 7+1) = 8$		$bb = 7$	
	$o = \max(9, 6+1) = 9$			

d.)

Total Order (Tie-breaking: $P_0 < P_1 < P_2 < P_3 < P_4$)

$C_i = 0$: a, h, p $\Rightarrow a \Rightarrow h \Rightarrow p$

$C_i = 1$: b, i, v $\quad b \Rightarrow i \Rightarrow v$

$C_i = 2$: c, j, w $\quad c \Rightarrow j \Rightarrow w$

$C_i = 3$: d, q, x $\quad d \Rightarrow q \Rightarrow x$

$C_i = 4$: k, y $\quad k \Rightarrow y$

$C_i = 5$: l, z $\quad l \Rightarrow z$

$C_i = 6$: e, m, r, aa $\quad e \Rightarrow m \Rightarrow r \Rightarrow aa$

$C_i = 7$: s, bb $\quad s \Rightarrow bb$

$C_i = 8$: n, t $\quad n \Rightarrow t$

$C_i = 9$: f, o $\quad f \Rightarrow o$

$C_i = 10$: g

$C_i = 11$: u

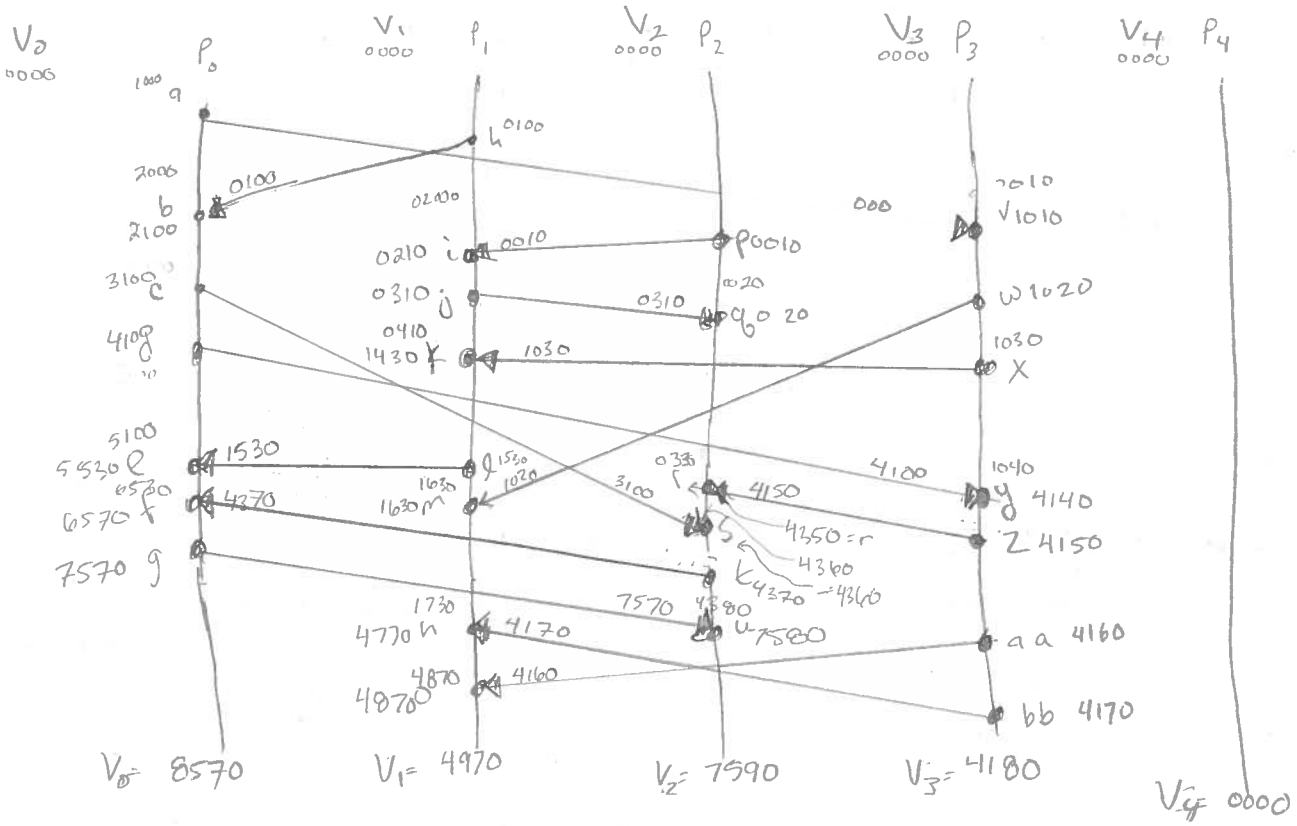
Tie-breaking Rules

$P_0 < P_1 < P_2 < P_3 < P_4$

$P_4 < P_3 < P_2 < P_1 < P_0$

- $V(a) = C_i(a)$ = a occurred on i ... incremented before any event on i
- $C_i[i]$ incremented between any two successive events on i
- On receiving m, j updates C_j: For all k, $C_j[k] = \max(C_j[k], t_m[k])$ — t_m = timestamp in entry

b) **[6 Points]** Provide vector clock (V) values of each event shown.



V_0

a: 1000
b: max(2000, 0100) = 2100
c: 3100
d: 4100
e: max(5100, 1530) = 5530
f: max(6530, 4370) = 6570
g: 7570

V_1

h: 0100
i: max(0200, 0010) = 0210
j: 0310
k: max(0410, 1030) = 1430
l: 1530
m: max(1630, 1020) = 1630
n: max(1730, 4170) = 4770
o: max(4870, 4160) = 4870

V_2

p: 0010
q: max(0020, 0310) = 0320
r: max(0330, 4150) = 4350
s: max(4360, 3100) = 4360
t: 4370
u: max(4380, 7570) = 7580

V_3

v: max(0010, 1000) = 1010
w: 1020
x: 1030
y: max(1040, 4100) = 4140
z: 4150
aa: 4160
bb: 4170

c) **[2 Points]** Identify two events *ai* and *aj* to show that C(*ai*) < C(*aj*) does not necessarily imply *ai* → *aj*.

In the figure, event b has a value 1, but event j has value 2, implying b→j. According to the vector clock, b and j are concurrent events. [b=2100, j=0310, V(b)< V(j) [false] V(j)< V(b) [false]]

a⇒b iff
- C(a) < C(b), or
- if C(a)=C(b), a occured on i b on j, & i<j

d) **[6 Points]** Assuming P0 < P1 < P2 < P3 < P4, provide the total ordering of all events constructed from the logical clock C. Is this total order unique? No.

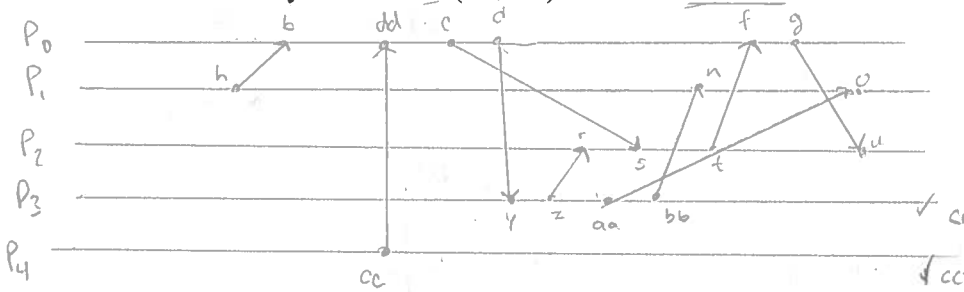
- Depends on the tie-breaking algorithm

a⇒h⇒p⇒b⇒i⇒v⇒c⇒j⇒w⇒d

⇒q⇒x⇒k⇒y⇒l⇒z⇒e⇒m⇒r

⇒aa⇒s⇒bb⇒n⇒t⇒f⇒o⇒g⇒u

e) **[3 Points]** Suppose process P4 sends a message m (send event is *cc* and the corresponding receive event is *dd*). Show how *cc* → *n* and *cc* → *u*. Identify an event *ai* (*ai* ≠ *cc*) such that *ai* → *dd*.



ai = b occurs before dd

h→b

b→dd

cc→dd→c→s→t→f→g→u

cc→dd→d→y→z→aa→bb→v

In this case:
- cc→dd send/receive
- dd→c successive events

f) **[2 Points]** Identify two messages between different pairs of processes, i.e. m1 between Pi, Pj and m2 between Pn, Pm, such that Pi ≠ Pj ≠ Pn ≠ Pm and m1 and m2 are causally related.

m1:
$P_3 \rightarrow P_1$
x → k

m2:
$P_0 \rightarrow P_2$
g → u

V ≤ V' iff V[i] ≤ V'[i]

V(e) ≤ V(e') nor V(e') ≤ V(e)

[1,0,0] < [0,0,1]

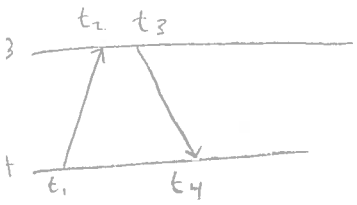
[0,1,0] > [0,0,1]

x→k
k→l
l→e
e→f
f→j
g→u

dispersion ε = max error inherent in the measurement

2. **[5 Points]** Browse the NTP project webpage (*http://www.ntp.org*). Explain how NTP computes filter dispersion.

- estimate error - quality & results
- based on accuracy & server's clock & consistency of network transit time



After having a pair of messages sent/received, there is enough information to calculate the clock offset and delay, computed by the on-the-wire NTP protocol discussed in class.
θ δ

The dispersion value $\varepsilon(t)$ is the maximum error due to the frequency tolerance and time since last measurement. It's initial value is

$$\varepsilon(t_0) = \rho_R + \rho + \Phi(T_4 - T_1)$$ where

ρ_R = peer precision in the packet header } account for uncertainties in reading the system clock in both the server & client
ρ = system precision
T_4 = timestamp at t_4
T_1 = timestamp at t_1
Φ = MAX disciplined system clock frequency tolerance

Then the function for determining the dispersion value is:

$$\varepsilon(t) = \varepsilon(t_0) + \Phi(t - t_0)$$ where the default value of Φ = 15 ppm, resulting in $\varepsilon \approx 1.3 \frac{\text{seconds}}{\text{day}}$

The clock-filter algorithm saves the most recent values of θ, δ, ε and the system time t, shifting in new samples to an 8-stage shift register.

Then, the values are copied to a temporary list and sorted by increasing δ. Then, going through all the values, a peer dispersion value is calculated by:

$$\varepsilon = \sum_{j=0}^{7} \frac{\varepsilon_j}{2^{j+1}}$$ overloading the ε term for the clock filter output.

Increasing a peer dispersion makes old servers less desirable and eventually boots them off, since a dispersion over a max distance (MAX DIST = 1s) notes the server as unacceptable threshold for synchronization.

The δ and ε statistics are accumulated at each stratum layer from the reference clocks to produce the root delay Δ and root dispersion E

P3

aa → bb

bb → n

aa → o

but

n → o when it should be o → n since aa → bb

g) **[2 Points]** Is there a violation of FIFO ordering between messages in this figure? If yes, provide one example of the corresponding messages.

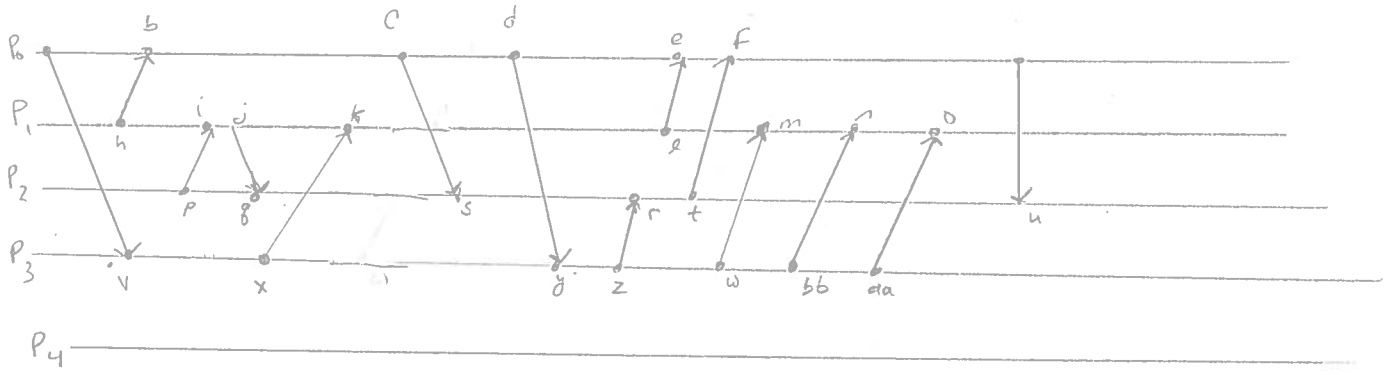
Yes: P_3 sends message aa then bb, both to P_1. However, P_1 receives the message from bb first then aa second

h) **[2 Points]** Is there a violation of causal ordering between messages in this figure? If yes, provide one example of the corresponding messages. Your example should not include the messages where there is a violation of FIFO ordering.

Yes. P_0 sends c before d, but the response from d (sent to P_3 & causing a response z → r from P_3 to P_2) is seen before the message c is received on P_2.

i) **[6 Points]** Provide a total ordering of messages that preserves causal ordering in this figure. Is your total ordering unique? If not, provide a different total ordering of messages that preserves causal ordering.

Not unique



① Tie-breaking: $P_4 < P_3 < P_2 < P_1 < P_0$

p ⇒ h ⇒ a ⇒ v ⇒ i ⇒ b ⇒ w ⇒ j ⇒ c ⇒ x ⇒ q ⇒ d ⇒ y ⇒ k ⇒ z ⇒ l ⇒ aa ⇒ r ⇒ m ⇒ e ⇒ bb ⇒ s ⇒ t ⇒ n ⇒ o ⇒ f ⇒ g ⇒ u

② Tie-breaking:

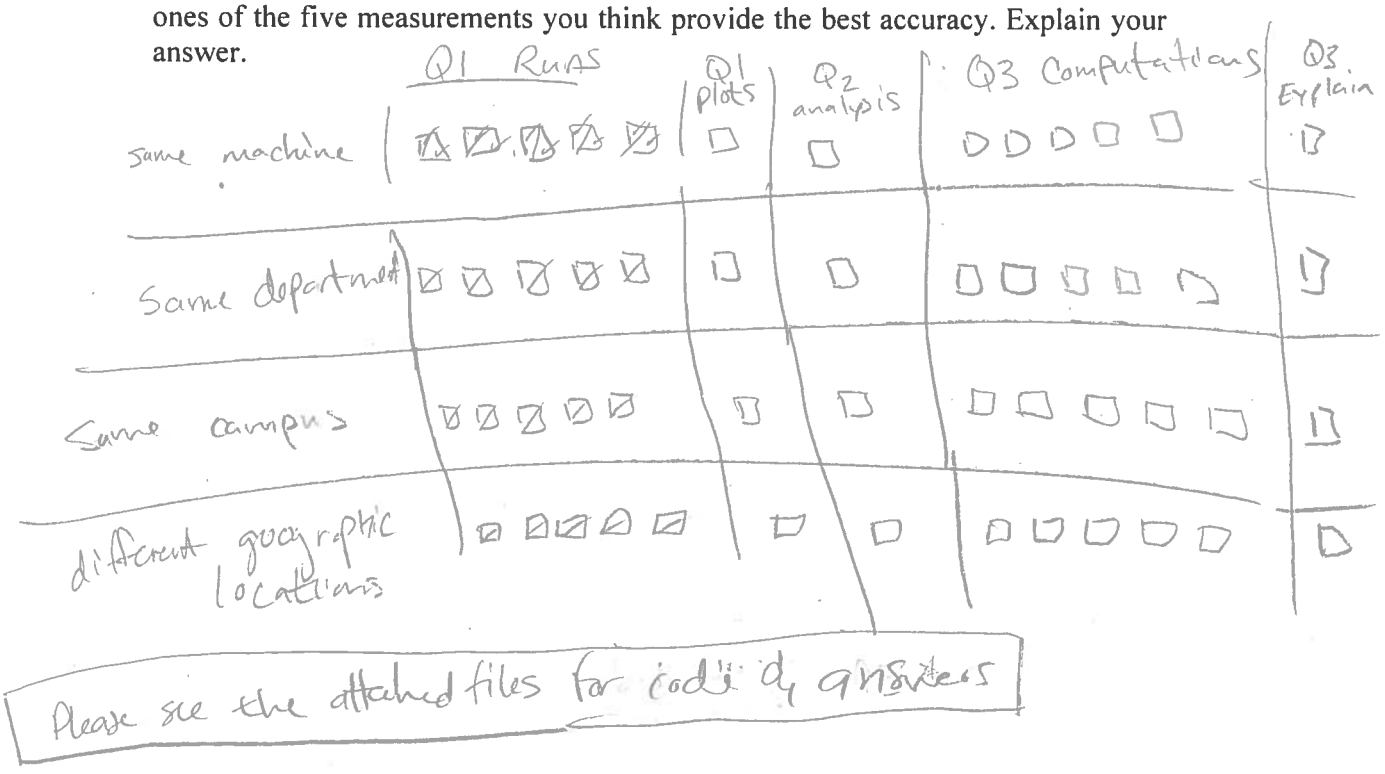
3. **[60 Points]** [Programming Question] Write a simple UDP client/server program, where the client sends a UDP message to the server and the server sends a reply to the client. Record the local clock times of when each message is sent and received. Record these timings by first running the client on machine A and the server on machine B, and then running the client on machine B and the server on machine A for the following scenarios:
   a) Machines A and B are the same machine
   b) Machines A and B are different machines with in the CS department, e.g. two CSEL servers
   c) Machines A and B are different machines across the CU-Boulder campus
   d) Machines A and B are different machines, one in CU campus and the other at a different geographic location

   Note that there are plenty of UDP client/server programs available on the Internet. Use one that suits your purpose. Of course, make sure that you understand the code and the code is working correctly before you use it.

   Question 1: For each scenario, repeat your experiment five times by running it at around the same time for each scenario. Report all the timings, and compute the pairwise latencies (A to B and B to A) along with average and standard deviation.

   Question 2: Provide an analysis of your results in terms of why there is a variation in latencies, which ones you expect to be more accurate, etc.

   Question 3: Compute the offset (o_i) and delay (d_i) for each of the five time measurements for each scenario using the NTP formula. For each scenario, which ones of the five measurements you think provide the best accuracy. Explain your answer.

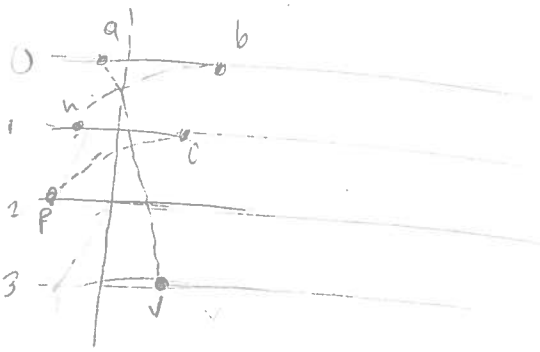


P0 < P1 < P2 < P3

P0	P1	P2	P3
a = 0	h = 0	p = 0	v = 1
b = 1	i = 1	q = 3	w = 2
c = 2	j = 2	r = 6	x = 3
d = 3	k = 4	s = 6	y = 4
e = 6	l = 5	t = 7	z = 5
f = 8	m = 5	u = 10	aa = 6
g = 9	n = 8		bb = 7
	o = 8		

b = 2100 (>, <, <) d = 0210 (<, >, >) f = 0310

P4 < P3 < P2 < P1 < P0



Partial order

a → v	h → b	p → i	v → w
b → c	i → j	t → f	w → m
c → s	j → q	p → q	w → x
c → d	j → k	s → t	x → k
d → y	l → m		z → r
g → u	M → n		y → z
	n → o		z → aa
			aa → bb

Total order : Tie Breaking

$c_i = 0$

P_0 a
P_1 h
P_2 p

a ⇒ h ⇒ p due to tie breaking

$c_i = 1$

P_0 b
P_1 i
P_3 v

b ⇒ i ⇒ v tie breaking

$c_i = 2$

P_0 c
P_1 j
P_3 w

c ⇒ j ⇒ w tie breaking

$c_i = 3$

P_0 d
P_2 q

d ⇒ q ⇒ x tie breaking

$c_i = 4$

P_1 k
P_3 y

k ⇒ y tie

$c_i = 5$

P_1 l
P_1 m
P_3 z

l ⇒ z, m ⇒ z } tie
l ⇒ m happened before

$c_i = 6$

P_0 e
P_2 r
P_2 s
P_2 aa

e ⇒ r, e ⇒ s, e ⇒ aa } tie
r ⇒ s happened before
r ⇒ aa, s ⇒ aa } tie

$c_i = 7$

P_2 t
P_3 bb

t ⇒ bb tie

$c_i = 8$

P_0 f
P_1 n
P_1 o

f ⇒ n tie
n ⇒ o happened before

Part 3: In this homework, I ran the Client/Server program on two machines: Machine A was my laptop running Ubuntu 16, Machine B was my desktop running Windows 10. I classified the required scenarios as shown below:

1. Machines A and B are on the *same machine*.
2. Machines A and B are on *different machines* within the *same local network*.
3. Machines A and B are on *different machines* within the *same wireless network*.
4. Machines A and B are on *different machines* at *different geographic locations*.

The code for this program can be found at:
https://github.com/dSouthard/ECEN5673_Fall2016/tree/master/UDPServerClient

In the first run, Machine A was the Client and Machine B was the server; in the second run that immediately followed, Machine A was the Server and Machine B was the Client.

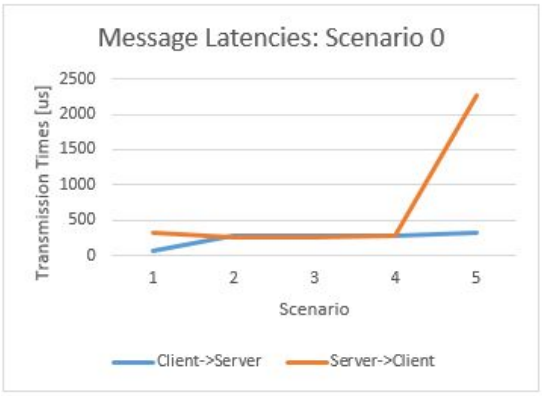
I ran into a difficulty when attempting to run Scenario 4. I was able to set up my home router to forward incoming ports to my home desktop (Machine B), allowing Machine B to receive messages from the Client and send replies. However, my different geographical location where Machine A was at this time was first the local library and then a Starbucks, with neither location allowing public reprogramming of their routers to allow for port forwarding. For this scenario then, I only have data from the first run. Additionally, since the laptop served as both client and server for both runs, the data from scenario 1 is identical for both runs.

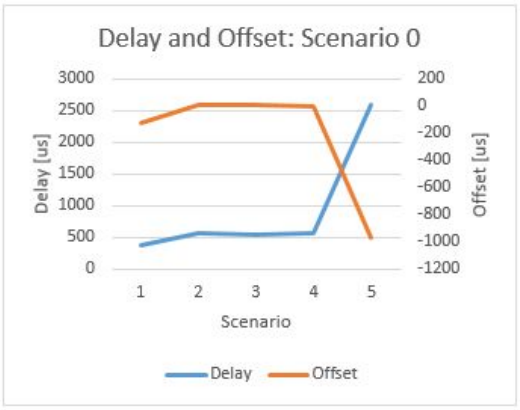
A second difficulty was when running scenario 2. For both runs in scenario 2 (same LAN), the clock times reported by Machine B on all message received/sent timestamps were exactly identical so far as the Windows OS was able to report. I attempted to use different time-reporting methods (from Python: *datetime.datetime.now()*, *datetime.datetime.utcnow()*, *time.time()*, *time.localtime()*) but all methods returned the same identical timestamps for this scenario alone. I concluded that the Windows OS might have something to do with this.

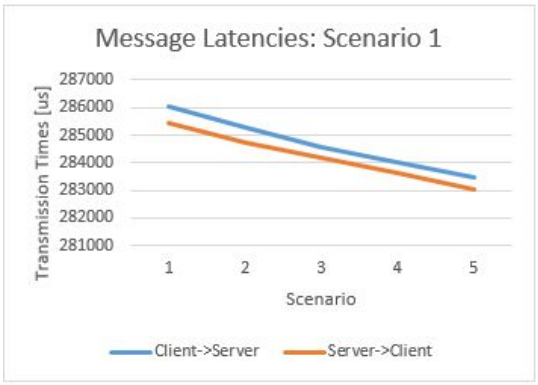
Please see the attached Excel spreadsheet to see the full calculations of messages latencies, averages, standard deviations, offsets, and delays. The plots of each are shown below.

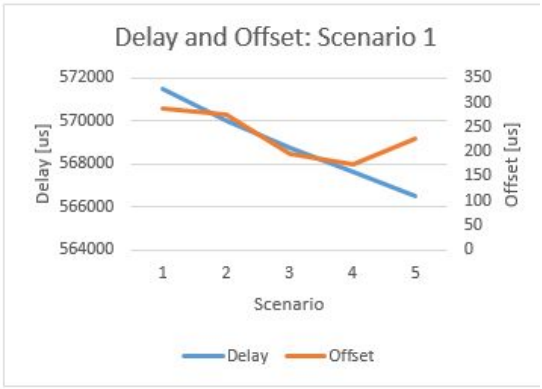
I would have expected the same machine latencies to be the smallest, since the message has the least amount to travel, and the different geographical locations to have the greatest latencies and variances, since those message would have to travel the furthest and would have the greatest chance of getting lost. In fact, that is what I did find. On average, the message latencies from the first scenario were miniscule compared to those from the other scenarios, with the last scenario having a dramatic increase in message latency times. The latency average does not grow significantly between scenario 2 and 3, where the machines are both on the same LAN and same WLAN respectively. This makes since because both machines are relatively close together and don't have to leave their local networks, wired or wireless. The dramatic increase in latency that occurred with scenario 4 can be explained by the messages having to travel across the internet some distances, greatly increasing their possibility of packet loss, packet collision, or packet delay.

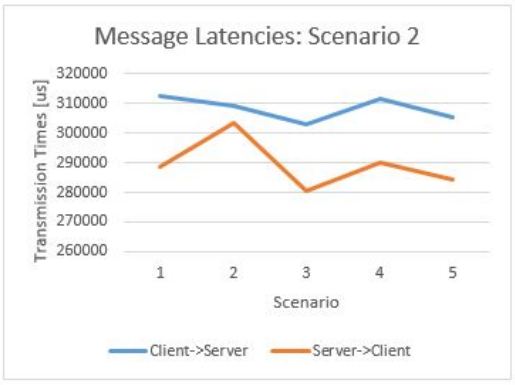
Run 1: Laptop = Client

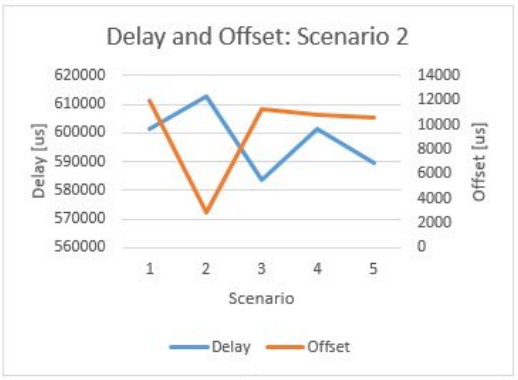


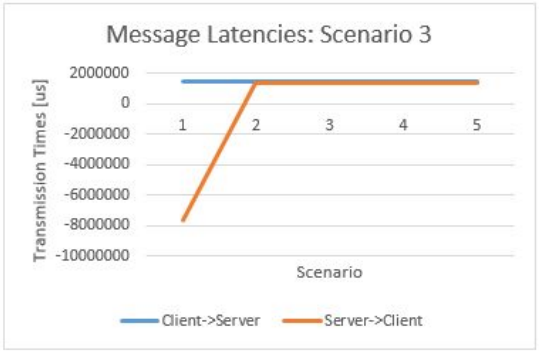


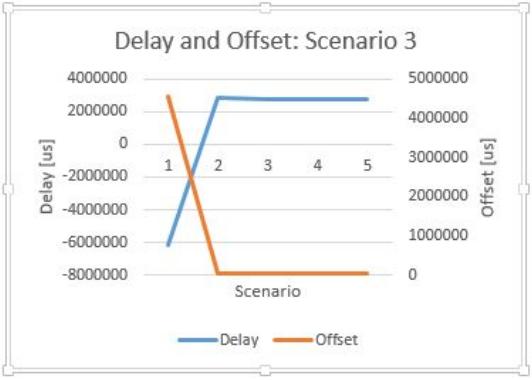


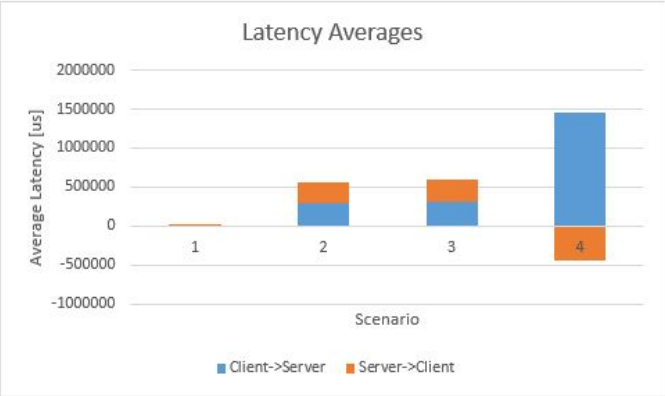








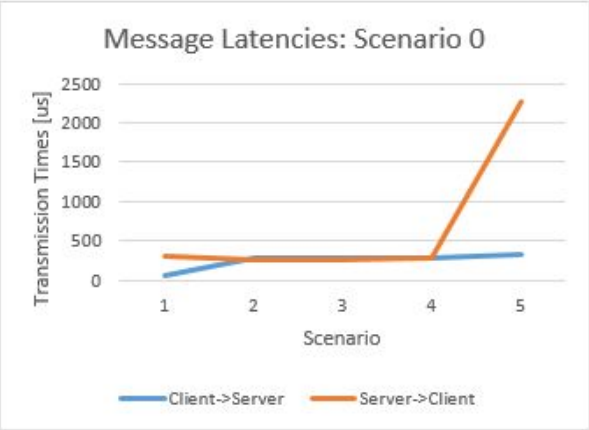


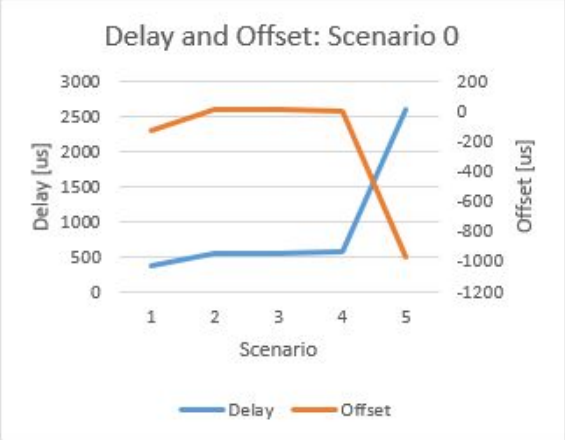


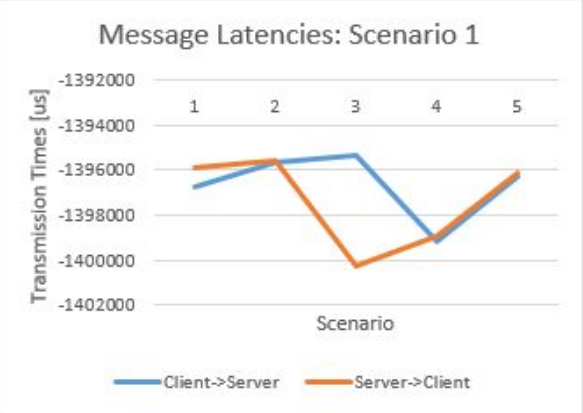
Q3. For each scenario, I was to decide which of the five calculated measurements of offset and delay provided the best accuracy.

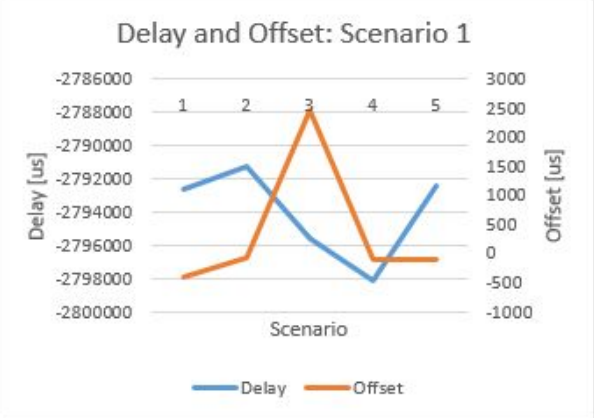
- Scenario 1 (Same Machine): The calculations from the 4th message would be the most inaccurate, as the message latencies increased significantly for the 5th message. For this scenario, I would say that the calculations from the 2nd messages were the most accurate. They resulted in no great increase in the message latencies.
- Scenario 2 (Same LAN): For each message, the calculated latencies decreased in time when compared to the times of the message before. Because of this trend, I would say that each newly calculated offset/delay pair was more accurate than the pair before. This would mean that the last pair, calculated from message 5, would be the most accurate of the group. However, looking at the final calculated pair, it looks as though the calculated offset from message 5 veers off the pattern set by the previous calculations. It may have resulted in increased latencies for any following messages, but there were none recorded for this experiment.
- Scenario 3 (Same WLAN): The message latencies decreased steadily between messages 1 and 3, and then again between messages 4 and 5 for client->server messages. The message latencies also decreased in between messages 2 and 3 and messages 4 and 5 for server->client replies. The greatest rate of decrease was between messages 2 and 3 for server->client replies. Because of this, I would say that the most accurate NTP calculation would be from message 2.
- Scenario 4 (Different geographical locations): In this scenario, the message latencies are relatively constant in between messages 2 and 5. There was a large change between message 1 and 2 for server->client replies, with the jump going from an extreme difference to a relatively constant transmission time. Because of this, I could say that the calculations from message 1 were the most accurate for the first correction, correcting the time differences well enough to leave the rest of the calculations as minor corrections in comparison.

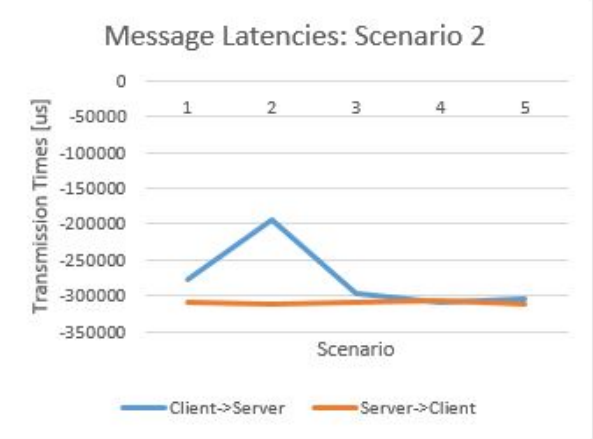
Run 2: Laptop = Server

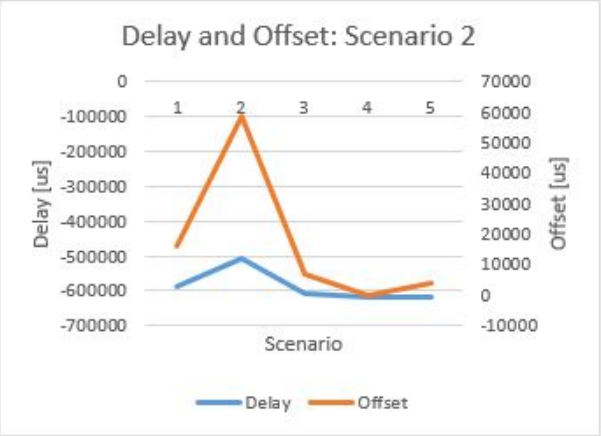


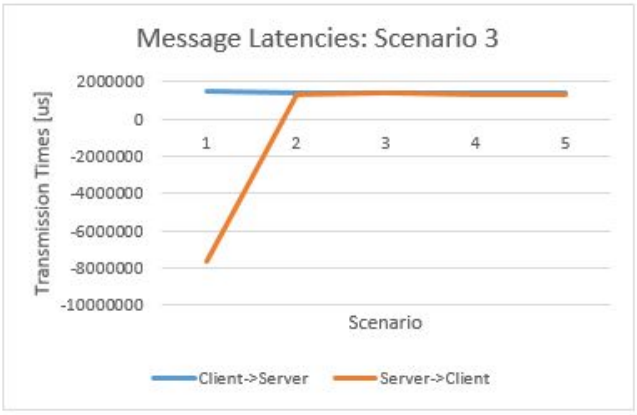


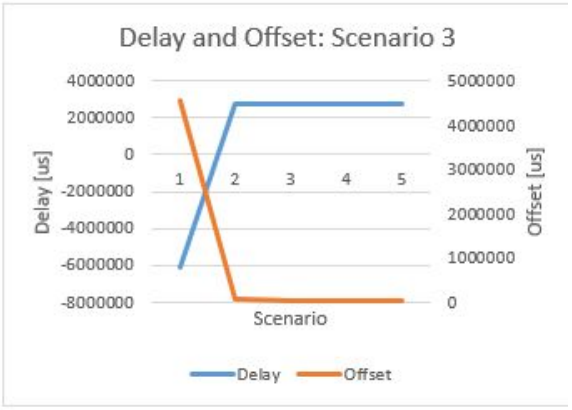


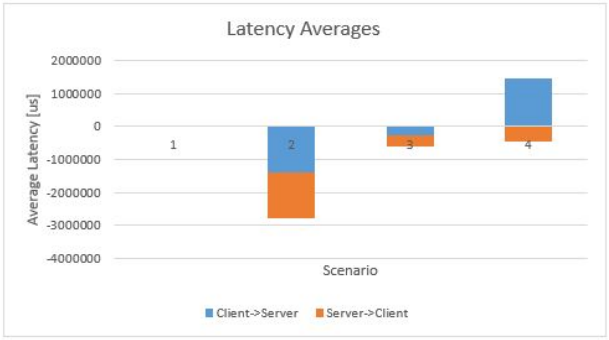


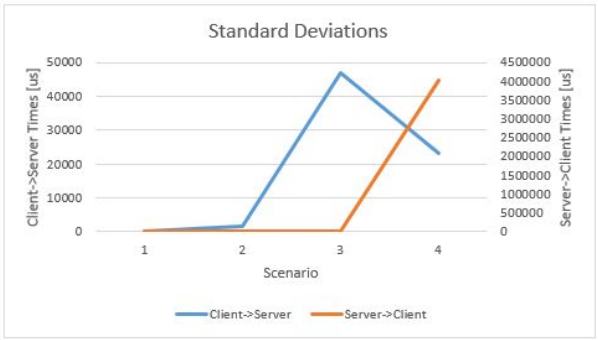












Q3. For each scenario, I was to decide which of the five calculated measurements of offset and delay provided the best accuracy. This is for the 2nd run.

- Scenario 1 (Same Machine): *Same as Run 1*
- Scenario 2 (Same LAN): The message latencies trended towards 0 most quickly in between messages 4 and 5. Also, the latencies between client->server and server->client messages most closely matched each other in that same period. Because of this, I would say that the NTP calculations from message 4 were the most accurate. The calculations from message 5 might also be of the same level of accuracy, but we do not have the results from a message 6 to see how the latencies trend.
- Scenario 3 (Same WLAN): The message latencies reach a relative consistency after message 3. From then on, there is only relatively minor changes in the latency times. The calculated delay and offset are also relatively similar from messages 3 to 5. However, there is a slight decrease in the magnitude of the latency from client->server in between messages 4 and 5. Because of this, I would say that the NTP calculations from message 4 are the most accurate.
- Scenario 4 (Different geographical locations): *Same as Run 1*