RIACS

Research Institute for Advanced Computer Science
NASA Ames Research Center

IN-70
43070

About Time

Peter J. Denning

PJ

14 May 90

RIACS Technical Report TR-90.34

NASA Cooperative Agreement Number NCC 2-387

About Time

Peter J. Denning

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report TR-90.34
14 May 90

Time has historically been a measure of progress of recurrent physical processes. Coordination of future actions, prediction of future events, and assigning order to events are three practical reasons for implementing clocks and signalling mechanisms. In large networks of computers, these needs lead to the problem of synchronizing the clocks throughout the network. Recent methods allow this do be done in large networks with precisions around 1 millisecond despite mean message exchange times near 5 milliseconds.

This is a preprint of the column *The Science of Computing* for
American Scientist 78, No. 4 (July-August 1990).

Work reported herein was supported in part by Cooperative Agreement NCC 2-387
between the National Aeronautics and Space Administration (NASA)
and the Universities Space Research Association (USRA).

About Time

Peter J. Denning

Research Institute for Advanced Computer Science

14 May 90

When someone asks, "What time is it?" I call out the reading from my watch without thinking. I take for granted that everyone shares my understanding of time and clocks. But there are circumstances that arise in large networks where our shared understanding does not apply and this seemingly innocent question is surprisingly difficult to answer.

In general, we regard time as a measure of the progress of recurrent physical processes. Until the pendulum clock was invented in the 17th century, natural cycles of the seasons, the moon, and the sun were the only means of measuring long intervals; shorter periods were measured by pouring sand or water through a small opening or by burning a candle. In the 20th century instruments that count oscillations of electromagnetic waves have led to clocks of great precision. An international standard adopted in 1964 defines a second as 9,192,631,770 cycles of the radiation making up the principal spectral line of cesium 133. Certain circuits in modern digital computers are capable of distinguishing between events spaced even more closely, on the order of 10^{-12}

second apart -- 1/100 of a tick of a cesium clock.

We have developed two major abstractions for time: sometimes it is a "quantity," measurable by clocks, sometimes an "arrow" giving direction to changes in the state of the universe. These abstractions quickly lead to philosophical questions: As we stretch our imaginations backward and forward, we ask when time started and when it will end. What was before the Big Bang? What will come after the Big Crunch? Meanwhile, we push our instruments to finer scales of resolution, observing quantum effects on a scale where uncertainty principles make it hard to distinguish between space and time; some physicists speculate that the two must have been one in the immense gravitational field of the Big Bang (1).

Coordination, prediction and ordering are three practical reasons for taking an interest in time and for building clocks and signalling mechanisms. Coordination is fundamental to preparing for action in an uncertain future. Agreements on schedules allow us to coordinate actions at chosen times: keeping a doctor's appointment, catching a train, buying stock, landing an airplane, reaching milestones on a project. To coordinate national and international enterprises we need to keep the clocks in our computers synchronized. In an air- traffic-control system, for example, the clocks must agree when a controller in one city notifies a controller in another city that a particular flight will arrive at a given time. Banks depend on synchronized clocks to prevent abuses in "floats" during electronic fund transfers.

Prediction is a special case of coordination. Using models of various recurrent physical processes, we can calculate with great accuracy when events in these systems will recur. For example, we can calculate the time of a solar eclipse, the orbit of the

Voyager spacecraft, the position of a shock wave on a transonic wing, the rate of a chemical reaction, or the speed of an object detected by radar. The ability to make such predictions allows us to build machines and processes that give us control over parts of nature. We may one day perform like feats in other domains, such as weather forecasting and earthquake prediction.

The third consideration--the ordering of events--is important in scientific, technological and even legal investigations. It is also important in computer-based transaction systems. An airline reservations system might receive several nearly simultaneous requests for the last ticket on a flight; the system must be able to determine which request came in first. In a bank's network of automated-teller machines two people must not be allowed access to the same account simultaneously, even if they are both authorized users of that account. In a software-development system it is vital to record accurately which of two files holds the latest version of a module.

Coordination and ordering require different technologies. Coordination supposes that all agents have access to a universal standard time, and thus it calls for synchronized clocks. Ordering supposes that agents agree to transmit signals indicating when selected events have taken place, and thus it calls for a message- exchange system. Reliable message-passing is often easier to ensure than reliable synchronization.

In either scheme, when two events are closer than the resolution of the local clock, it is impossible to say which came first. If we must nonetheless act as if the events came in some definite order, it is impossible to design a device that can guarantee a decision about the order within any predetermined amount of time. A system that relies on such a guarantee will eventually malfunction (2).

Among the challenges to time management in computing systems, synchronizing multiple clocks is probably the most difficult. In what follows I shall review some results from the computing literature on this problem.

For this discussion, a computing system is defined as a set of computers at various fixed locations that carries out some function affecting people and computers over a wide area. The machines might all be in a single building, or they might be spread across a campus, a country, or the world. Networks of this kind are used for banking, air-traffic control, ticket reservations, and many other purposes. The system will not function correctly unless the clock in each machine is closely synchronized with standard time.

Let us examine more closely what it means to say that the clocks are synchronized to standard time. Suppose two observers at different positions record the times at which they detect two events, which are also spatially separated. Because of differences in signal-transmission time, the observers will almost certainly record different times, and they may even disagree about the order of the events. The usual way of resolving such conflicts is to define a standard observer equally distant from the sites of the two events; the order in which the standard observer registers the signals of the events is taken as the actual order of the events (3).

In synchronizing the machines of a computer network, however, placing an observer at the midpoint is seldom practical, because of variations in communications pathways. For signals passing through the network, transmission time varies according to the load or traffic level; furthermore, a signal can take a variety of routes from one node to another, and the choice of route is not always under direct control. Special communications pathways could be set up outside the network, exclusively for

synchronization signals. But even with such dedicated channels, there remain physical limits on the precision of time measurements; for example, with satellite links to the observer, variations in atmospheric conditions affect transmission time.

And even if we could somehow start two clocks running at exactly the same instant, we could not keep them synchronized. Although every effort is made to design clocks that progress at the same speed, there are nonetheless small differences in clock rate. The maximum rate at which the clock readings diverge is called the drift rate, denoted r; over an interval of length t, a clock may deviate from standard time by as much as rt. Typical drift rates are on the order of 10^{-6} clock ticks per tick.

These two effects--variability of signalling time and clock drift-- mean that in a practical system we cannot achieve synchrony of all the clocks. What can we achieve? We can specify a maximum discrepancy among the clocks, and we can design the system to periodically resynchronize the clocks to an accuracy within that maximum. The period we can allow to lapse between resynchronizations will depend on how closely the clocks can be made to agree at each synchronization and on the rate of drift. When this protocol is enforced, we can can be sure that two events whose times are measured by different clocks occurred in the order given by the clock readings, provided that the readings differ by more than the maximum allowable discrepancy.

Message-transfer time in the communications network varies from a minimum of m to a maximum of M. The minimum depends on the path length in the network and on the message-exchange protocols, and the maximum depends on the traffic of messages in the network. It is not uncommon for the ratio of M to m to be as high as 100 or 1,000. In a local-area network covering distances of no more than a mile or so, m is typically

about 5 milliseconds; across a satellite link, it is roughly 250 milliseconds.

Leslie Lamport of the Digital Equipment Corporation was among the first authors to publish a serious treatment of synchronization in computer networks (4). He supposed that all messages are "time-stamped" with the value of the sender's clock at the moment of transmission. He stipulated that when a node receives a message with time stamp t, it sets its local clock to the larger of $t+m$ and the current time according to that clock. This policy guarantees that clock readings are consistent with the known order of the events. Note that if a message actually takes the maximum time M, the sender's clock will read $t+M$ at a moment when the receiver's clock reads at least $t+m$; thus the discrepancy between clocks at the moment of clock adjustment can be as large as $M-m$.

Lamport asked how far apart the clocks can get under this scheme. He measured the distance k between a pair of nodes P and Q be by the number of links connecting them. He assumed that every node sends a message to its neighbors-- adjusting their clocks, if necessary--at least once every T seconds. Over the interval T, P's clock can gain as much as rT seconds, and Q's clock can lose as much as rT seconds, for a total additional discrepancy as large as $2rT$. Hence in the worst case the discrepancy between clocks can be as large as $k(2rT+M-m)$. For typical values of T drift is not significant and the discrepancy is approximately $k(M-m)$. Lamport's method of synchronization, therefore, would not be useful where there is a large difference between the minimum and the maximum message-exchange times or a large path length between the most distant clocks.

Lamport's scheme illustrates a class of techniques that advance clocks to maintain consistency with the known order of events. The clock moves forward at least one tick

for each event inside a node, and it may move forward many ticks to be consistent with the time-stamp of an incoming message. Although these schemes limit the maximum discrepancy that can develop among the clocks within the system, they do not guarantee agreement with an external standard clock. Moreover, actions outside the system can circumvent the synchronization rules. For example, when someone updates a file on a distant machine by giving the new information to a friend over the telephone, the file will be stamped with the local time when the receiver completed the update but will not be adjusted to take account of the time of the updater's request. If the file is then passed over the network to some other node, it could bear an inaccurate time-stamp and might therefore be mistaken for an earlier rather than a later version.

In spite of these limitations, clocking schemes such as Lamport's are useful in synchronizing multiple processes that share a resource that can be used by only one process at a time. The seat-assignment list for an airplane is an example of such a resource. Associated with the shared resource is a process that controls access. Other processes request access by sending a message to the controller, which maintains an internal queue of requests in time-stamp order; the controller signals each requester when its turn comes.

Flaviu Cristian of the IBM Almaden Research Center has proposed a method of synchronization that maintains close tolerances even in networks with highly variable message-exchange times (5). Cristian considers the following situation. Suppose P sends Q a message, "What is your time?" to which Q immediately responds "Time=t." P must now decide how or whether to adjust its own clock based on the reply. The decision can be guided by how long it took for the exchange of messages. P establishes

a cutoff parameter, U, which must be greater than m; if the response is received within $2U$ seconds, P uses the time-stamp t to set its clock to the value $t+U$, which is the estimate that causes the maximum possible discrepancy to be minimum at $U-m$. If the response arrives too late, P waits a few seconds and tries again.

Cristian's method has several parameters that allow it to be adjusted with high precision. One of these parameters is U: as the cutoff time is reduced toward m, the accuracy of each successful synchronization improves, but the number of abortive attempts increases. Another parameter is the interval between synchronization attempts, which should be chosen so that resynchronization can be expected to succeed before drift exceeds the maximum allowable discrepancy. Cristian cites an example of a system in which m = 4.2 milliseconds and U = 4.5 milliseconds, yielding a probability of 1/2 that any given synchronization attempt will succeed. With a clock drift rate of 6×10^{-6}, the average time between synchronizations in this system is 67 seconds; the average number of messages needed to achieve synchronization is four, and the probability that synchronization will not be achieved in any cycle is 10^{-9}. The maximum clock discrepancy is one millisecond.

Clock synchronization and event ordering are essential to the correct operation of large systems of computers. They are important concepts in other fields as well. I invite readers familiar with the literature on this subject outside of computer science to send me citations and reprints, and I will report on them in a future installment of this column.

References

1. Stephen W. Hawking. 1988. A Brief History of Time. Bantam Books.

2. Peter J. Denning. 1985. The arbitration problem. American Scientist 73:516-518.

3. Albert Einstein. 1961. Relativity. Crown Publishers.

4. Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21: 558-565.

5. Flaviu Cristian. 1989. A probabilistic approach to distributed clock synchronization. Proceedings of Ninth International Conference on Distributed Computing Systems, pp. 288-296. IEEE Computer Society Press.

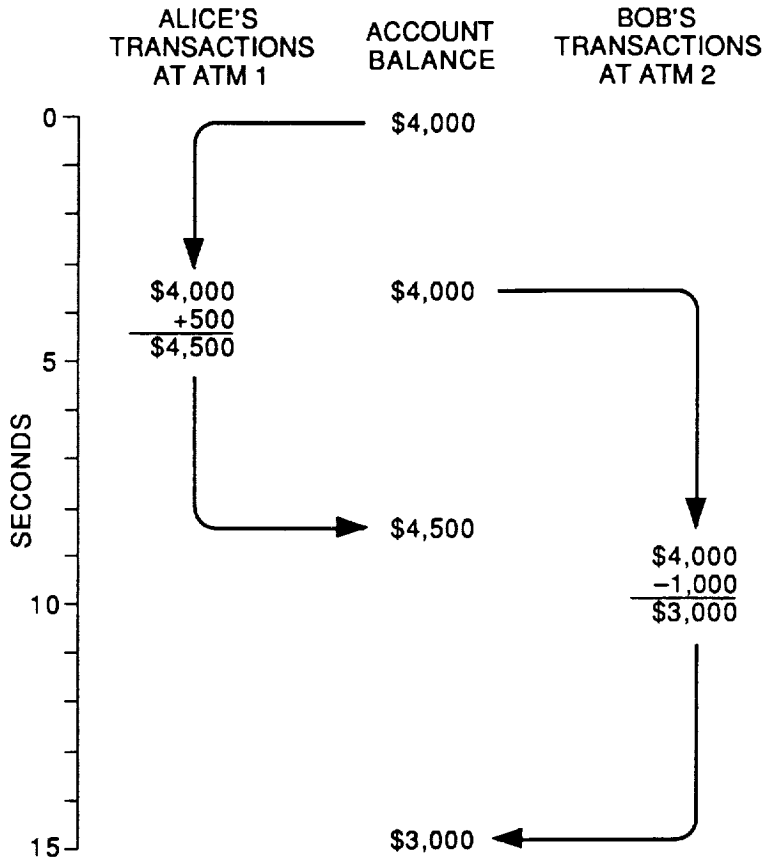
FIGURE CAPTIONS

FIGURE 1. Banking transactions supply an example of a realm where the proper ordering of events is crucial. Suppose Alice and Bob have a joint account. When Alice deposits $500 at an automated-teller machine, the computer posting the transaction reads the starting balance of $4,000, adds the deposit amount,and writes the sum of $4,500 to the file where the account balance is stored. Meanwhile, Bob has initiated a withdrawal at another automated-teller machine. That computer reads a starting balance of $4,000, subtracts $1,000, and writes a new balance of $3,000. Because of the way the events are interleaved, Alice's deposit is lost. In a real banking system this error would be avoided by refusing to start Bob's transaction until Alice's had been completed.

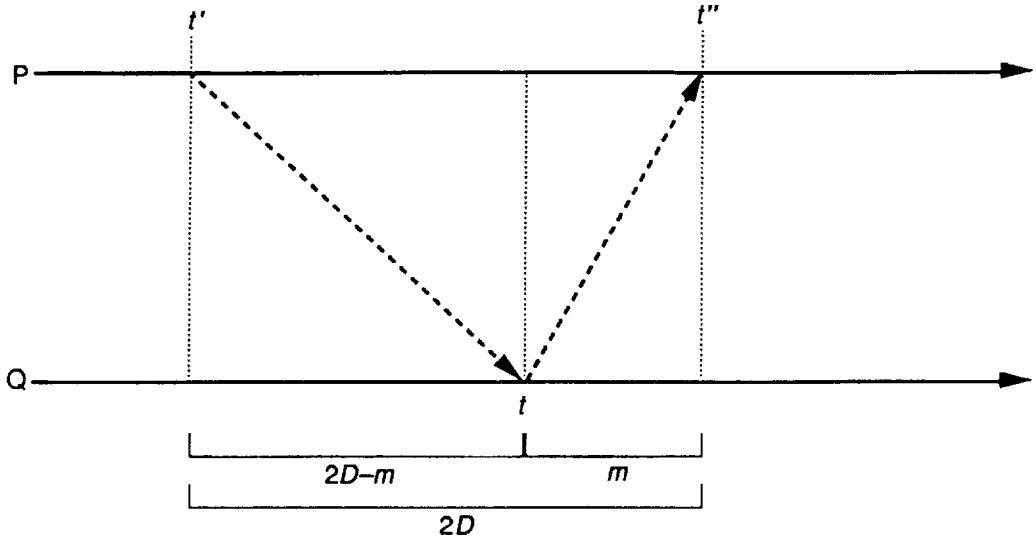
FIGURE 2. Protocols for syncrhonizing clocks in a computer network are complicated by variations in the transmission time for messages. In these diagrams vertical distance represents spatial separation, and time progresses from left to right. At time t', computer P sends a time-query message to computer Q, which immediately responds with the reading t of its local clock. When the reply is received, however, P cannot simply set its clock to t because Q's clock has continued running while the reply was in transit. P can estimate the current value of Q's clock by relying on three items of information: the reported clock value (t), the elapsed time for the round-trip exhcange of messages ($2D$), and the minimum one-way transit time (m). If the response message arrived in minimum time, then Q's clock must now read $t+m$ *(top diagram).* If the response took the maximum time, Q's clock must read $t+2D-m$ because the request part of the exchange could not have been transmitted in less than m seconds *(middle diagram).* The difference between the minimum and maximum feadings is $2D-2m$. Splitting this

difference by setting P's clock to $t+D$ yields a maximum discrepancy between the clocks of $D-m$.

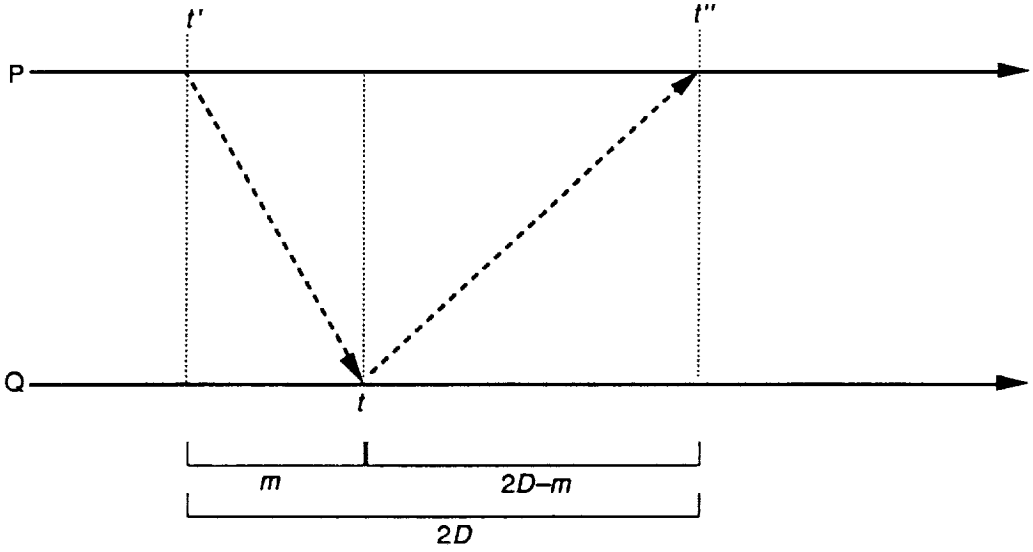
FIGURE 3. Improved protocol aborts an attempt to synchronize clocks if network delays would impair accuracy. The protocol defines a cutoff time, designated $2U$; if the round-trip exchange time $2D$ is greater than $2U$, the requesting computer abandons the attmept to synchronize and tries again a few seconds later. The reply to a time-query message is accepted only if the it comes in the interval between $2m$ and $2U$ *(gray region)*. After a successful synchronization, the maximum discrepancy between clocks is $U-m$. The average number of attempts needed to achieve synchronization is $1/p$, where p is the probability of success at each try.


ALICE'S TRANSACTIONS AT ATM 1
ACCOUNT BALANCE
BOB'S TRANSACTIONS AT ATM 2
SECONDS
0
5
10
15
$4,000
$4,000
+500
$4,500
$4,000
$4,500
$4,000
−1,000
$3,000
$3,000

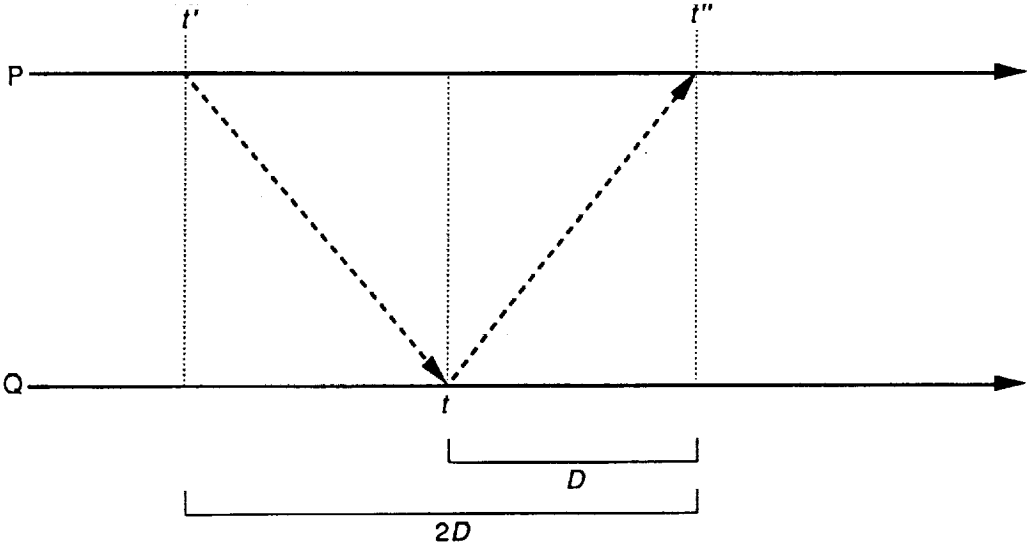
Minimum response time

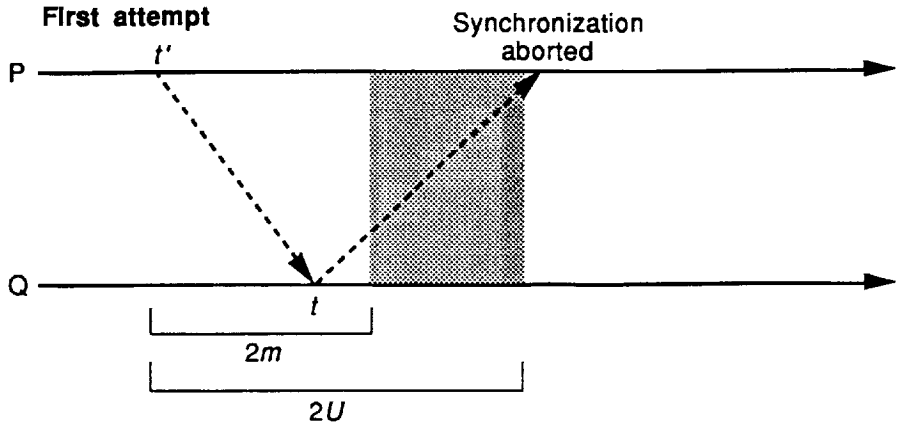


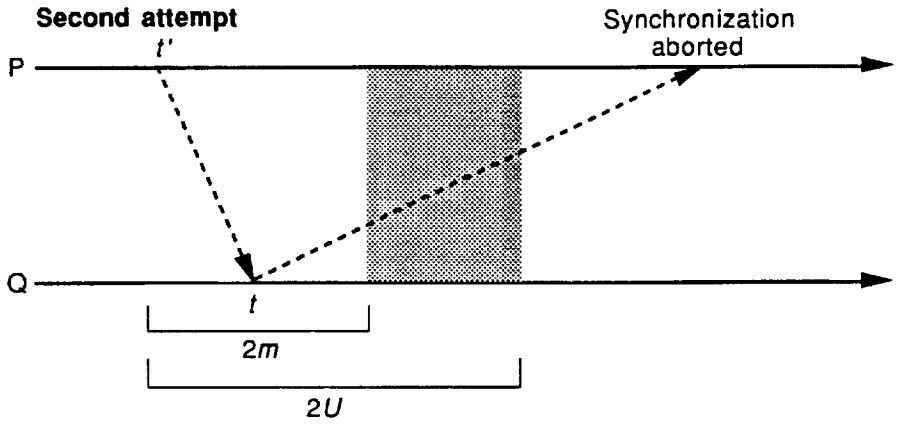
Maximum response time

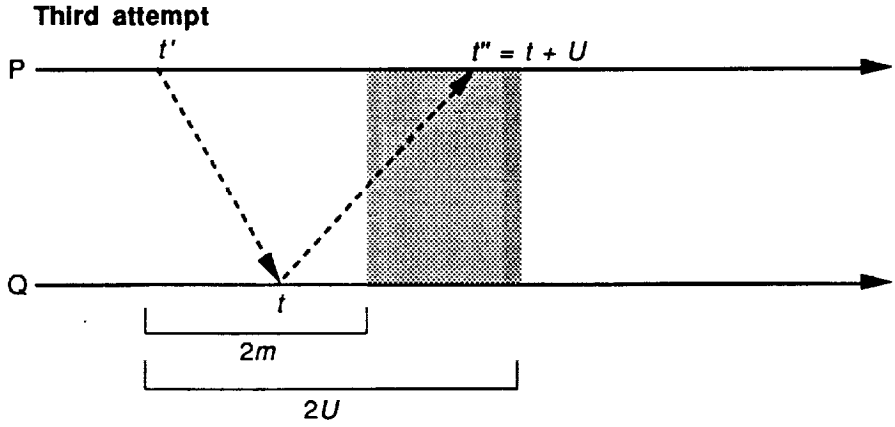


Best estimate of response time




First attempt
Synchronization aborted
t'
P
Q
t
2m
2U


Second attempt
Synchronization aborted
t'
P
Q
t
2m
2U


Third attempt
t'
t" = t + U
P
Q
t
2m
2U