Synchronisation

The earlier discussions on measuring time delays when calculating distance were based on the assumption that the signal was sent from a single transceiver, that both sends and receives the signal on a single circuit. In the localisation solutions, it is more likely the transmitters and receivers are distributed. This presents a new issue. For accurate measurements of TOF, TDOA or any other technique, we need to be confident that each module in the distributed system is synchronised to within a reasonable degree of accuracy.

In a distributed system, each node has its own physical clock. These clocks are based on crystal oscillation counters which generate a number of interrupts per second. The clock in the node will tick on each timer interrupt. For the raspeberry Pi we can run the following command to find the current clock speed.

sudo cat /sys/devices/system/cpu/0/cpufreq/cpuinfo\_cur\_freq

Update: With Raspian based on Debian 8.0 you can find it here:

pi@raspberrypi $ sudo find /sys -name '\*cpuinfo\_cur\_freq\*'

/sys/devices/system/cpu/cpufreq/policy0/cpuinfo\_cur\_freq

The issue is that two devices will hardly ever agree, as the clocks will oscillate at slightly different frequencies. If we take UTC time, , to be the perfect clock, the physical clock on nodes, , usually run faster or slower. This is known as drift, and can be measured using the . Ideally, which means the clock is perfectly in sync with UTC time. If , the clock is fast, and , the clock is slow. When we read two disagreeing clocks at the same time, the difference in values is known as the skew. The aim of synchronisation is to keep the skew between clocks bound to within an acceptable constant of drift, , such that . As we are attempting to use the time difference in signals to measure distance, even a skew of 1ms could lead to errors in measurement of around 0.35m . To ensure the time difference between any two clocks in the system never exceeds a maximum value of , The clocks need to be synchronised every seconds.

There are several methods of achieving system wide synchronisation.

External synchronisation requires each node to synchronise its clock with an authoritative external source. The MSF is a 60 kHz radio signal which is dedicated to broadcasting the current UTC which can be decoded by radio-controlled clocks for synchronisation. Similarly, GPS receivers can be used to synchronise with UTC. These solutions, do provide accurate measurements, but are not a practical solution for simple distributed systems.

A much simpler solution is to have a dedicated time server within the system, which all other devices can synchronise with.  
Each node can ask the time server for an accurate time periodically, and adjust its clock accordingly. This requires an accurate measure of the round trip delay for the node to receive the updated time.

Cristian’s algorithm is a simple implementation of this method. A client node sends a request to the timing server for the current time at T­0. The message is received at the server after a delay for transmission across the network. The server processes the request and sends the current time, t, back to the client, who receives the time at T1. Knowing nothing else about the network, the client will set its clock to the estimated value of . Provided the network delays are small, this is a reasonably accurate measurement. If we know the minimum delay in sending messages across the network, , we can improve the accuracy of the result by limiting the range for the answer to . This is a simple approach to the problem, but has a major drawback if the time server fails, the entire system will fail. It also relies on the time server having a UTC synchronized clock.

An alternative approach is to let the time server be the master, and periodically poll all nodes in the system for their local time. Taking an average time across the system, tell each slave node by how much they need to adjust their clock. This is a completely self contained system that uses relative time adjustments from a single point, so there is no need for external synchronisation.

The Berkeley algorithm is an implementation of this idea. The master polls each node and estimates the delay for each node. Once all nodes have responded, the master will take the average time of all clocks in the system, including its own, and calculate the average. This accounts for individual clocks tendencies to run fast or slow. It then sends the offset for each local clock to adjust. The algorithm will exclude local times whose skew is too great, to compute a more tolerant average. As this is completely contained within the system, if the master fails for any reason, any other node can take over. The selection of the new master node is done by a process called election. In a system where all distributed processes have knowledge of every other node, but not whether the node is active or not, the process with the largest process id should become the new coordinator.

When a process P recovers from failure, or the failure detector indicates that the current coordinator has failed, P performs the following actions:

If P has the highest process id, it sends a Victory message to all other processes and becomes the new Coordinator. Otherwise, P broadcasts an Election message to all other processes with higher process IDs than itself.

If P receives no Answer after sending an Election message, then it broadcasts a Victory message to all other processes and becomes the Coordinator.

If P receives an Answer from a process with a higher ID, it sends no further messages for this election and waits for a Victory message. (If there is no Victory message after a period of time, it restarts the process at the beginning.)

If P receives an Election message from another process with a lower ID it sends an Answer message back and starts the election process at the beginning, by sending an Election message to higher-numbered processes.

If P receives a Coordinator message, it treats the sender as the coordinator.

Ring election algorithm. When a process, P, notices the coordinator is down, it passes the Election message with it’s own id as part of a list. The next process passes this message to the next process in the ring, with it’s own id appended to the list and so on until P receives the Election message with it’s own id in the list of processes. It then selects the largest process as the new coordinated and sends a new Coordinator message with the new coordinators id and the list of active processes.

Network Time Protocol (NTP) enables clients across the internet to be accurately synchronised to UTC. It is a client-server architecture based on UDP message passing. It can provide reliable service even with lengthy losses of connectivity and allows clients to offset typical drift rates by synchronising frequently. It provides a reliable service by having redundant paths and servers and provides protection against interference.

It arranges the network into groups of sub-networks, known as strata. Nodes on the first strata are connected directly to the synchronisation source. The second strata contains nodes connected to the first strata and so on.

There are several NTP modes. Multicast uses one node to periodically send time information to all other nodes on the network. The nodes assume a small transmission delay and adjust their clock. This method is lower in accuracy and only suitable to high speed LANs. Procedure-call mode is similar to Cristian’s algorithm. A service will accept requests from clients. This yields more accurate results than multicast. Symmetric mode is used, generally by servers in the same strata in the system, to improve synchronisation over time. A node may be connected to a number of higher strata servers as well as peers on the same stratum.

In procedure-call and symmetric modes, messages are exchanged in pairs between servers, denoted by m and m’ for the sent and received messages respectively. For these message pairs, NTP calculates the offset estimate between the two clocks, the delay time between the messages, and the filter dispersion, which is an estimate of the quality of the results based on the accuracy of the server’s clock and consistency of the network transit time. This is calculated using the algorithms based on the most recent measurements of offset and delay between the client and server. Time servers will communicate with multiple peers and eliminates peers with unreliable data.

NTP messages contain a lot of information about the nodes, including clock precision, stratum, reference timestamps for when the clock was last set and poll intervals. This information can all be used as part of validating messages.

Getting the offset estimate in NTP is similar to what was used in Cristian’s algorithm. The NTP message m is timestamped at , when it is sent from the client. It is then timestamped again at , when it is received at the server. The server will then send the response m’ which is timestamped at times and by the server and client when it is sent and received, respectively. The calculation of offset is defined as , and the round trip delay by . These values are collected and statistically analysed to rule out outliers and the clock frequency is adjusted gradually according to the best estimate of the offset.

SNTP is a simplified version, or rather, a subset of NTP. It operates in procedure-call or multicast mode, but does not use the statistical methods in NTP to adjust the clock gradually. It is used in systems where the root node is the server and clients are leaf nodes (stratum 1).