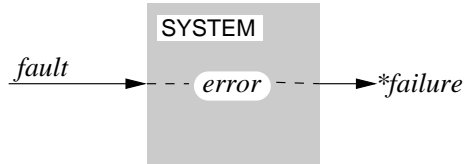


1 Fault Tolerance

[Some material taken from Kopetz, Chs. 3 & 6]

1.1 Terminology

- *fault* – unanticipated condition
- *error* – invalid or unintended system state
- *failure* – deviation from specified behavior



Faults and errors are states. Failure is an event. A fault may be the consequence of a failure “up stream.”

1.1.1 Fault Causes

- *Hardware Failure*. We say a system has a *failure rate* of 10^{-k} to mean that it can be expected to fail at most once in every 10^k hours of operation.
 - ICs after “burn-in,” 10^{-8} to 10^{-7} , or once in 115,000 years [Kopetz, 1997].
 - Complex ICs once in 5–10 years [Driscoll, et al., 2003]
 - wire failure, discrete components, 10^{-9} per wire/device.
- *Transient Failure*. Electro-magnetic interference, “cosmic rays” (stray protons, alpha particles, etc.), ...
 - 10^{-6} to 10^{-2} failure rate for commercial products!
- *Communications Races*. Greater dependence on shared carrier communications network for critical components.

1.1.2 Fault Qualities

- *Nature*: value, timing
- *Perception*: consistent, inconsistent
- *Effect*: benign, malign
- *Persistence*: permanent, transient

1.1.3 Fault Models

There is no general consensus yet on how to model faults mathematically. The *Hybrid Fault Model* proposed by NASA/LRC's FM Group:

\mathcal{G} : *good*, non-faulty

\mathcal{B} : *benign*, observably faulty

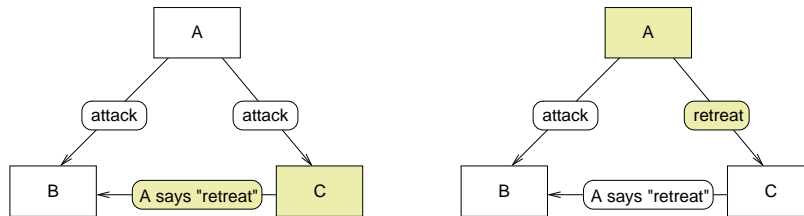
\mathcal{S} : *symmetric*, all see the same (faulty) behavior

\mathcal{A} : *asymmetric* (Byzantine) different observers see different (faulty) behaviors

One would think that the asymmetric model is unrealistic, but to the contrary, [Driscoll, et al.] show that such phenomena do occur and have been measured. Byzantine faults arise in failing logic gates and broken communication wires because non-valid voltages can be digitized differently at different points in a circuit.

1.2 Key Results

For the problem of *interactive consensus*, in which a system of communicating components is supposed reach agreement on some value, three components is not enough if it is assumed that there is one byzantine "traitor." The essence of the argument is that good component B cannot differentiate between bad behavior in A or in C and, therefore, cannot determine the right thing to do. With four components and just one traitor, the good components can agree by majority voting.



Under certain synchronization assumptions, tolerating k failures of a given fault type:

\mathcal{B} : takes $k + 1$ components

\mathcal{S} : takes $2k + 1$ components

\mathcal{A} : takes $3k + 1$ components

In the asymmetric case (\mathcal{A}), $k + 1$ rounds of message exchange are required. So tolerating one byzantine component requires as many as 32 messages under these assumptions.

Asynchronous distributed consensus is impossible, in the presence of even one byzantine fault [Fischer, Lynch & Paterson]. In essence, this means that consensus depends on some synchronizing agent outside of the system and hence a consensus in time.

1.3 Safety-Critical Design

In critical applications, software must be designed to be resilient to infrequent failures of the hardware components in a system. The Federal Aviation Administration (FAA) and other regulatory agencies classify failure severities in five categories:

A *Catastrophic*: failure prevents the safe flight and landing of the aircraft, resulting in many fatalities including the crew.

Target failure rate: 10^{-9} (one every 1,000,000,000 hours)

B *Hazardous*: failure significantly reduces flight safety margins and the capability of the aircraft to fly, possibly resulting in fatal injuries but not to the crew.

Target failure rate: 10^{-7} .

C *Major*: failure reduces flight safety margins and the capability of the aircraft to the point where potential injuries could occur.

Target failure rate: 10^{-5} (8 hours/day, 365 days/year translates to 34 years)

D *Minor*: failure does not significantly reduce aircraft safety, resulting in potential discomfort to passengers or crew.

Target failure rate: 10^{-5}

E *No Effect*: failure does not affect the operation of the aircraft. Example: cabin entertainment system fails.

In order to certify a system for use in critical applications, the system's designers must demonstrate, through various means and practices, that it satisfies its target failure rate. Given statistics of hardware failure, no system whose correct operation depends on a single component is safe. The function of the systems hardware and software must be tolerant to the statistically inevitable occurrence of faults. Hence, there are two things to show:

1. The system performs correctly if there are no more than N faults at any time.

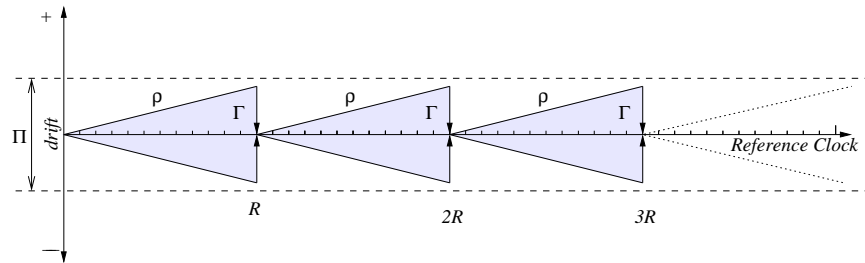
- The probability that there will ever be more than N faults is less than the target failure rate.

In current practice it is debatable whether any system can be verified to satisfy Level A's 10^{-9} standard. Clearly, this cannot be done by physically testing the system, so mathematical analysis is required.

1.4 Example: Distributed Clock Synchronization

The goal of a distributed clock synchronization algorithm is keeping a system of clocks in close agreement about the logical time of the system. A correct algorithm keeps the system within a *precision envelope* by occasionally exchanging local time estimates. *Accuracy* with respect to absolute global time is less important than agreement.

Let us first consider a perfect solution, which we know to be unattainable. The drawing below plots clock drift relative to the reference clock. The system re-synchronizes periodically in order to keep within a precision envelope of Π . Assume for the moment that re-synchronization adjusts all the clocks to agree perfectly.



Assuming a maximum drift rate of ρ , Γ is the worst-case *drift offset* for all the clocks.

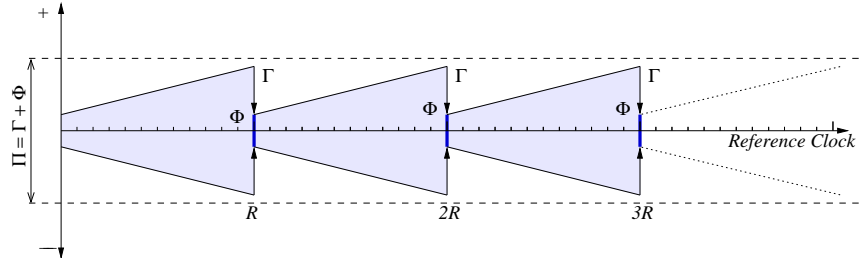
$$\Gamma = 2\rho R$$

From the point of re-synchronization, the clocks start drifting apart again, until the next re-synchronization occurs R ticks later.

As we know, there cannot be perfect agreement in the presence of drift. In the drawing below a *convergence interval* Φ represents the possible offset *after* the clocks have been adjusted. The clocks may not agree perfectly, but they agree more closely. It must be the case that

$$\Phi + \Gamma \leq \Pi$$

in order to stay indefinitely within the precision envelope.



We are left with two questions to answer.

1. What *convergence function* is used by individual clocks to obtain the convergence interval Φ ?
2. What re-synchronization period (R) is needed to keep the system within precision envelope Π ?

1.4.1 Convergence Functions

It is not obvious that Question 1 has an answer, but if it does, it surely involves clock-processes telling each other what time they *think* it is.

RESYNCHRONIZATION PROTOCOL:

1. When each clock reaches its re-synchronization point (in *local* time) it sends a SYNC message to every other clock.
2. Each clock is programmed to know when to expect SYNC messages from the other clocks. It measures the difference between the expected and actual arrival times.
3. Once it has received all its SYNC messages, each clock adjusts itself by adding the integer-average¹ of the measured differences.

For an N -clock system, this protocol involves transmission of N^2 SYNC messages. Moreover, the protocol is highly sensitive to communication *latency jitter*. The table below gives a sense of jitter magnitude at different levels of an implementation [Kopetz, Table 3.2]

SYNC communication time	approx. jitter
Application SW	500 μ sec – 5 msec
OS kernel SW	10 μ sec – 100 μ sec
communications HW	\approx 0 μ sec – 10 μ sec

[Kopetz, Table 3.2]

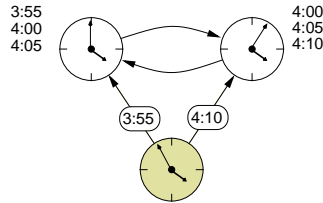
¹This adjustment function works, but there are other possibilities that are computationally cheaper.

In fact, precision is almost proportional to jitter. Lundelius and Lynch show that for a system of N clocks with jitter ϵ , precision is bounded by

$$\Pi \geq \epsilon \left(1 - \frac{1}{N}\right)$$

1.4.2 Fault Tolerance

The example below illustrates how a faulty clock can defeat the synchronization protocol described earlier. It can send (at) different times far enough apart to take the offset in averages outside the precision envelope.



In order to protect against this possibility we must change Step 3 of the protocol. To tolerate k byzantine faults,

k -FAULT TOLERANT RESYNCHRONIZATION PROTOCOL

1. Same as before: send SYNC messages to all clocks
2. Same as before: compute the differences between expected and actual arrival times of SYNC messages.
- 3a. *Discards* the k highest and the k lowest values.
- 3b. Adjust each clock by the average of the remaining values.

The example above does not contain enough clocks to tolerate a single byzantine fault; at least four would be needed, because for each fault two values are discarded.

CLAIM:[Kopetz, Sec. 3.4.3] *Given a system of N clocks containing at most k byzantine faults at any time, with drift offset Γ and communication jitter ϵ , the synchronization protocol described above maintains a precision envelope of*

$$\Pi(N, k, \epsilon, \Gamma) = (\epsilon + \Gamma) \left[\frac{N - 2k}{N - 3k} \right]$$

The *Byzantine error term* $\mu(N, k) = (N - 2k)/(N - 3k)$ is the factor of precision degradation caused by byzantine behavior. For various small N the values are

k	N							
	4	5	6	7	10	15	20	30
1	2	1.5	1.33	1.25	1.14	1.08	1.06	1.03
2	—	—	—	3	1.5	1.22	1.14	1.08
3	—	—	—	—	4	1.5	1.27	1.22

[Kopetz, Table 3.3]

Example: Suppose a quartz oscillator is used with a drift rate of 10^{-4} sec/sec and the clocks resynchronize every second. Then Γ is around $100 \mu\text{sec}$.² A hardware implementation with a dedicated communication channel might yield a jitter rate of around $10 \mu\text{sec}/\text{sec}$. In a real system, one might design for at most one Byzantine error at any given moment. If four clocks were used, the precision envelope would be

$$\Pi(4, 1, 10, 100) = 220\mu\text{sec}$$

And with five-clock redundancy,

$$\Pi(5, 1, 10, 100) = 165\mu\text{sec}$$

Reducing the precision degradation overhead to under 10% would require 15 clocks.

1.5 References

1. Kevin Driscoll, Brendan Hall, Hiåkan Sivencrona and Phil Zumsteg. Byzantine Fault Tolerance, from Theory to Reality. In *Computer Safety, Reliability, and Security*, Lecture Notes in Computer Science, 2003, Volume 2788/2003, Springer, 235248, doi:10.1007/978-3-540-39878-3_19
2. Michael J. Fischer, Nancy A. Lynch and Michael S. Paterson, Impossibility of Distributed Consensus with One Faulty Process, *Journal of the ACM*, April 1985, **32** (2):374–382.
3. Leslie Lamport, Robert Shostak and Marshal Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* **4** (3): 382–401. doi:10.1145/357172.357176.
4. Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization, *Information and Control*, **62** (2/3):190–204, August/September, 1984.

²It is possible to reduce Γ by two orders of magnitude using error compensation techniques

5. Paul Miner and Steven D. Johnson. Verification of an Optimized Fault-Tolerant Clock Synchronization Circuit. Mary Sheeran and Satnam Singh (eds.) Designing Correct Circuits (DCC96). *Electronic Workshops in Computing*, Springer, September 1996.