Lecture aims

1. Produce a fault-tolerant design (using hardware and software redundancy) that increases the reliability of a system.
2. Describe and implement several techniques for information redundancy.

Lecture plan

Introduction

1. Definition of fault tolerance: important to draw the idea that it is about detecting (and maybe correcting) faults in a system so that it can *continue operating correctly* (of safely, securely, etc.).
2. Fault tolerance recognises that we will make mistakes in producing systems, so we need to work around the,
3. *Redundancy* is the key to fault tolerance. Four types: hardware, software, information, and time (re-calculation). Only focus on first three.
4. Hardware vs software failures:
   - Hardware fails from manufacturing fault or from wear and tear. For the latter, it fails randomly.
   - Software fails from design faults, does not wear out. It fails systematically.

Hardware redundancy

1. Show Lardner's quote about redundancy (below).
2. Redundancy still key today, however, simply doing re-calculation is not going to work for e.g. software.
3. Static pairs: calculate answer, if not the same, an *interface* switches off pair from network (draw figure).
4. Static pairs: what if the interface is faulty? Use a monitor — monitor and interface check each other (draw figure below).
5. Static pairs detect failure, but cannot tolerate it. For that, new more than 2 units.... a *voting*.
6. Approximate vs. exact agreement; *sufficiently equal*
7. Voting algorithms: majority, k-plurality, and median.
8. Majority: harder than it sounds.
9. Majority voting algorithm: group sufficiently equal inputs into classes, take the largest class, P_i, and if $\#P_i \geq \frac{N}{2}$, pick any value in P_i as the output. If $\#P_i < \frac{N}{2}$, no agreement.

10. Use example: 30.1, 30.2, 30.3, 30.5, 30.7, with $\epsilon = 0.2$.
11. k-plurality: As for majority voting, but $\#P_i \geq k$.
12. Median voting: take the median.
13. Question: why not an *averaging* voter? Faulty measures can influence result.
14. Question: what if the faulty component is also the median?!
15. Comparison of voters: show table. Median voters always give output. Majority and k-plurality do not always give output, but are only incorrect under multiple (N or k) failures.
16. Median voters used when decision is required. Majority or k-plurality used when decision can be delegated.
17. Architecture: N-modular redundancy. Show figure. Can tolerate m faults with $2M + 1$ components.

Software redundancy

1. Software redundancy harder: simply installing N copies of software does not tolerate software failure due to systematic nature of faults.
2. *N-version programming* is a software-based version of hardware redundancy. Requires that N *independent* versions of the code are written. All N independent versions of the code are executed and the output is voted on.
3. Common-mode failures in engineering refer to those failures that are not statistically independent from one another. For example, the back two tires on a car a likely to fail (run out of tread) at around the same time because they will have driven the same distance.
4. Causes: specification faults, similar development environments, use of similar algorithms, similar training.
5. Solution: design diversity. Good specification, controlling individual to ensure different tools and algorithms, team separation, different organisations/training.
6. Does independence work? Show figure from Leveson and Knight (below)
7. Downsides: common-mode failures exist; cost (re-implement N versions)
8. Recovery blocks: show figure.
9. Acceptance tests $\approx$ oracle problem in software testing
10. Advantages: as we go down the stack, implementations can become simpler: more approximate and less efficient; because they will be executed infrequently.

Byzantine failures

1. Malicious/Byzantine failure: different values to multiple units.
2. Show table of voters below.
3. Byzantine general's problem (show figures below).
4. To tolerate n Byzantine failures, we require $3n + 1$ units.
5. Byzantine algorithm (below): works by essentially have the lieutenants vote by treating each others' orders as votes.
6. Relation to systems engineering: general is a (possibly faulty) sensor, and lieutenants are (possibly faulty) units
7. In practice: Lamport's original algorithm too slow. In 2002, Castro and Liskov presented the "Practical Byzantine Fault Tolerance" (PBFT) algorithm. Now a flurry of research.
8. In practice: Bitcoin currency. To obtain a single coin of currency, a node must work as part of a "proof-of-work" chain, and synchronisation of the global chain view is performed using a majority vote. Deliberate malicious behaviour tolerated using Byzantine agreement.

Information redundancy

1. Duplication. High overhead. Can only detect (unless duplicated twice)
2. Error detection using parity coding: count 1s and append a single bit.
3. Interlaced parity coding (show table): example 00000000-1100. Must be bit w_4 from table.
4. Interlaced parity coding useful for when original data cannot be recovered; e.g. cannot ask to send again; memory storage etc.
5. Checksums: break up data into words, line up words, and check. Simplest is *longitudinal parity check*: xor bits in word, then receiver xors all words *including the checksum*. If the result contains a bit that is non-zero, then an error has been detected.

Fault tolerance in practice

Airbus A310–A380

The Airbus series of aircraft designs rely heavily on hardware, software, and information redundancy to achieve fault tolerant behaviour of aircraft.

1. A310 (circa 1983) had ten separate digital flight control computers.
2. A320 (circa 1988) introduced fly by wire, in which four computers teamed in command/monitor pairs for redundancy, which became standard approach for subsequent Airbus flight control computers.
3. A340 (circa 1992) had one command/monitor pair forming a "fail fast" module, failing over to another command/monitor "hot spare" pair. Error detection was achieved through mismatched command, sequence checking, and self-test when aircraft energised.

   A second command/monitor pair using a different microprocessor and different software provided design diversity to tolerate common mode design and manufacturing faults.
4. A380 (circa 2005) employs dual-redundant Ethernet for non-critical functions, electrical and hydraulic flight control diversity.

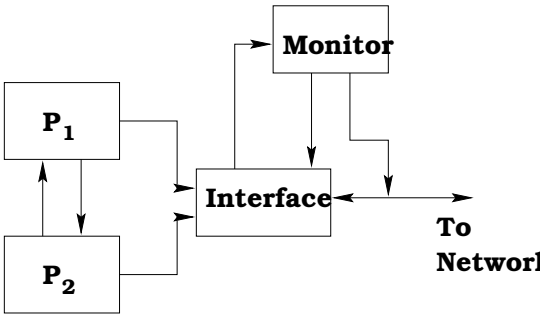
   Design diversity in the A380 project was achieved via using dissimilar computers, physical separation of teams, multiple software bases, different software development tools, and data diversity.

Boeing 777

The Boeing 777 is another aircraft that relies heavily on hardware, software, and information redundancy.

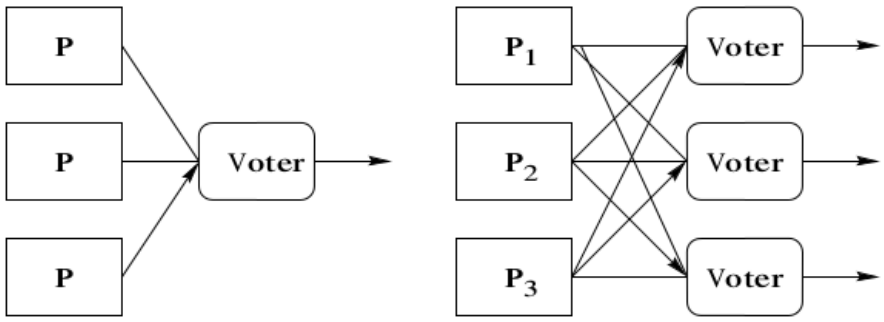
1. Goal of Mean Time Between Actions to 25,000 operating hours, reduce probability of degrading below minimum capability to less than 10^{-10}.
2. Designed to tolerate Byzantine faults, object impact, component failure, power failure, electromagnetic interference, cloud environment.
3. Byzantine fault tolerance with data synchronisation and median voting.
4. Architecture of flight control computer has three independent channels each composed of three redundant computing lanes (command, monitor, standby). Standby allows dispatch of aircraft even with a lane or data channel failure.
5. Design diversity was achieved using different microprocessor hardware and different software compilers for a fault-intolerant single source code.

"The most certain and effectual check upon errors which arise in the process of computation is to cause the same computations to be made by separate and independent computers; and this check is rendered still more decisive if their computations are carried out by different methods." – D Lardner, 1824.



Case	**Majority voter**	***k*-plurality voter**	**Median voter**
All outputs correct and SE*	Correct	Correct	Correct
Majority correct and SE	Correct	Correct	Correct
$k^{\dagger}$ outputs correct and SE	No output	Correct	Possibly correct
All correct but none SE	No output	No output	Correct
All incorrect and none SE	No output	No output	Incorrect
$k^{\dagger}$ outputs incorrect and SE	No output	Incorrect	Possibly correct
Majority incorrect and SE	Incorrect	Incorrect	Incorrect
All incorrect and SE	Incorrect	Incorrect	Incorrect

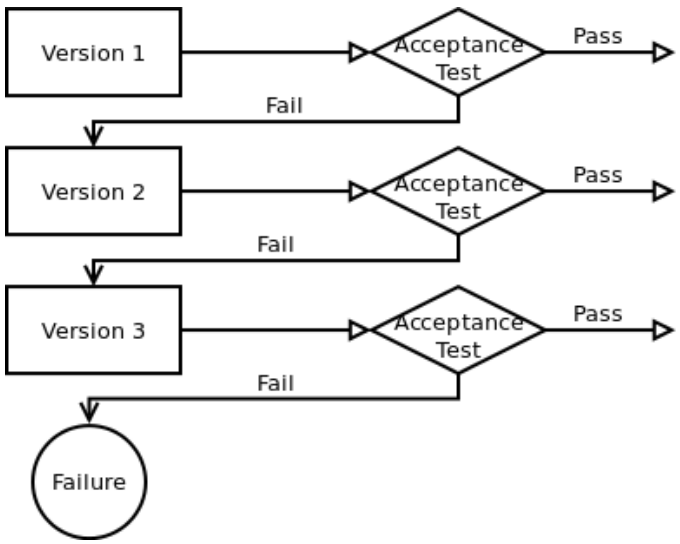
* SE = Sufficiently equal
† where $k < \frac{N}{2}$



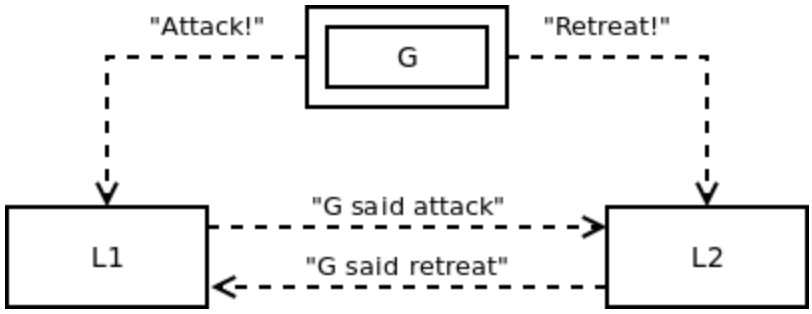
(a) (b)

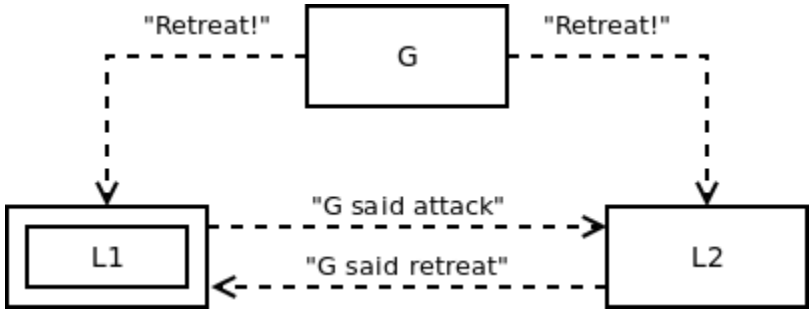
Correlated failures between UCI and UVA versions

	UVA versions								
UCI versions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
10	0	0	0	0	0	0	0	0	0
11	0	0	58	0	0	2	1	58	0
13	0	0	1	0	0	0	71	1	0
14	0	0	28	0	0	3	71	26	0
15	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	1	0	0	0
17	2	0	95	0	0	0	1	29	0
18	0	0	2	0	0	1	0	0	0
19	0	0	1	0	0	0	0	1	0
20	0	0	325	0	0	3	2	323	0
21	0	0	0	0	0	0	0	0	0
22	0	0	52	0	0	15	0	36	2
23	0	0	72	0	0	0	0	71	0
24	0	0	0	0	0	0	0	0	0
25	0	0	94	0	0	0	1	94	0
26	0	0	115	0	0	5	0	110	0
27	0	0	0	0	0	0	0	0	0



Sensor	Voter 1	Voter 2	Voter 3
1	11	15	17
2	14	14	14
3	16	16	16





	Received by		
Sent by	L1	L2	L3
G	attack	attack	retreat
L_1	-	attack	attack
L_2	attack	-	attack
L_3	retreat	retreat	-
Majority vote	attack	attack	attack

	Received by		
Sent by	L1	L2	L3
G	attack	attack	attack
L_1	-	m_1	m_2
L_2	attack	-	attack
L_3	attack	attack	-
Majority vote	attack	attack	attack

Algorithm $Byz(0)$

1. The commander sends his orders to every lieutenant.
2. The lieutenant uses the order they receive from the commander or the default (say retreat) if no order is received.

Algorithm $Byz(n)$

1. The commander sends his orders to every lieutenant.
2. For each $i \in 1..n$, lieutenant L_i acts as the general in a $Byz(n-1)$ algorithm and sends out order v_i to each of the other $(n-2)$ lieutenants, i.e. $(n-2)$ attempts to agree on $L_i\ 's$ order.
3. For each i and $j \neq i$, let w_{ij} be the order that lieutenant L_i receives from lieutenant L_j in step 2 (using the $Byz(n-1)$ algorithm), or the default if no order is received. Lieutenant L_i follows the majority $\{v_i, w_{i,j} | i \neq j\}$.

Example of overlapping parity with eight information bits

	Leads to error in parity bit			
Bit error	P_3	P_2	P_1	P_0
w_0	×	×	×	×
w_1	×	×	×	
w_2	×	×		×
w_3	×		×	×
w_4	×	×		
w_5		×	×	
w_6			×	×
w_7		×		×
P_0				×
P_1			×	
P_2		×		
P_3	×			