UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 739
Distributed Systems

Andrea C. Arpaci-Dusseau

Byzantine Generals

One paper:

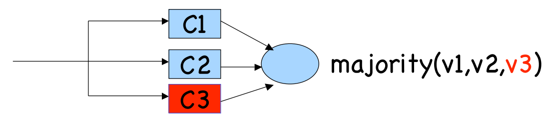
- "The Byzantine Generals Problem", by Lamport, Shostak, Pease, In *ACM Transactions on Programing Languages and Systems*, July 1982

Motivation

Build reliable systems in the presence of faulty components

Common approach:

- Have multiple (potentially faulty) components compute same function
- Perform majority vote on outputs to get "right" result



f faulty, f+1 good components ==> 2f+1 total

Assumption

Good (nonfaulty) components must use same input

- Otherwise, can't trust their output result either

For majority voting to work:

1) All nonfaulty processors must use same input
2) If input is nonfaulty, then all nonfaulty processes use the value it provides

What is a Byzantine Failure?

Three primary differences from Fail-Stop Failure

1) Component can produce arbitrary output
   - Fail-stop: produces correct output or none
2) Cannot always detect output is faulty
   - Fail-stop: can always detect that component has stopped
3) Components may work together maliciously
   - No collusion across components

Byzantine Generals

Algorithm to achieve agreement among "loyal generals" (i.e., working components) given m "traitors" (i.e., faulty components)

Agreement such that:

A) All loyal generals decide on same plan

B) Small number of traitors cannot cause loyal generals to adopt "bad plan"

Terminology

- Let v(i) be information communicated by ith general
- Combine values v(1)...v(n) to form plan

Rephrase agreement conditions:

A) All generals use same method for combining information

B) Decision is majority function of values v(1)...v(n)

Key Step: Agree on inputs

Generals communicate v(i) values to one another:

1) Every loyal general must obtain same v(1)..v(n)

1') Any two loyal generals use same value of v(i)

- Traitor i will try to loyal generals into using different v(i)'s

2) If ith general is loyal, then the value he sends must be used by every other general as v(i)

Problem: How can each general send his value to n-1 others?

A commanding general must send an order to his n-1 lieutenants such that:

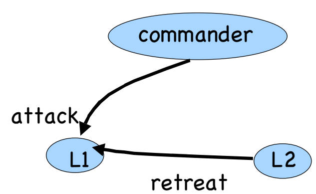
IC1) All loyal lieutenants obey same order

IC2) If commanding general is loyal, every loyal lieutenant obeys the order he sends

Interactive Consistency conditions

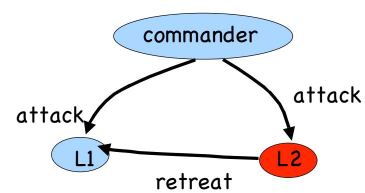
Impossibility Result

With only 3 generals, no solution can work with even 1 traitor (given oral messages)



What should L1 do? Is commander or L2 the traitor???

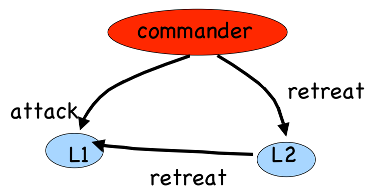
Option 1: Loyal Commander



What must L1 do?

By IC2: L1 must obey commander and attack

Option 2: Loyal L2



What must L1 do?

By IC1: L1 and L2 must obey same order --> L1 must retreat

Problem: L1 can't distinguish between 2 scenarios

General Impossibility Result

No solution with fewer than `3m+1` generals can cope with `m` traitors

< see paper for details >

Oral Messages

Assumptions

- A1) Every message is delivered correctly
- A2) Receiver knows who sent message
- A3) Absence of message can be detected

Oral Message Algorithm

`OM(0)`

- Commander sends his value to every lieutenant

`OM(m), m>0`

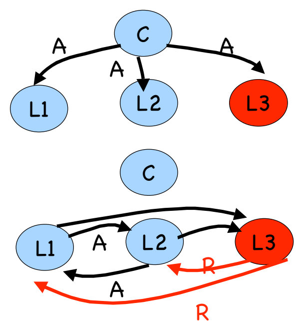
- Commander sends his value to every lieutenant
- For each `i`, let `vi` be value Lieutenant i receives from commander; act as commander for `OM(m-1)` and send `vi` to `n-2` other lieutenants
- For each `i` and each `j not i`, let `vj` be value Lieut `i` received from Lieut `j`. Lieut i computes `majority(v1,...,vn-1)`

Example: Bad Lieutenant

Scenario: m=1, n=4, traitor = L3

OM(1):

OM(0):???



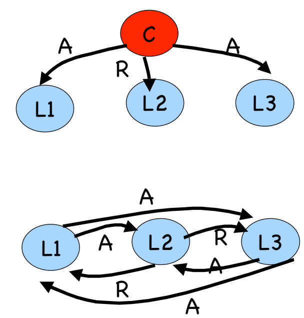
Decision?? L1 = m (A, A, R); L2 = m (A, A, R); Both attack!

Example: Bad Commander

Scenario: m=1, n=4, traitor = C

OM(1):

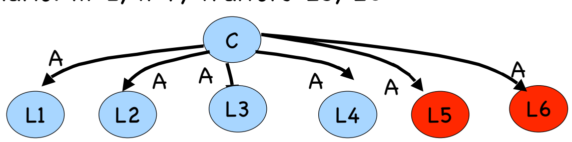
OM(0):???



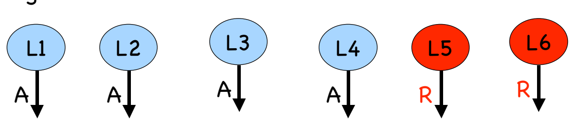
Decision?? L1=m(A, R, A); L2=m(A, R, A); L3=m(A,R,A); Attack!

Bigger Example: Bad Lieutenants

Scenario: m=2, n=7, traitors=L5, L6



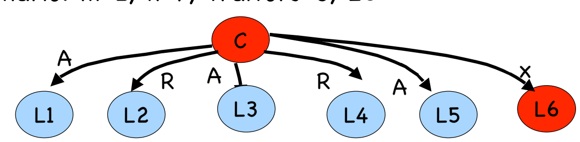
Messages?



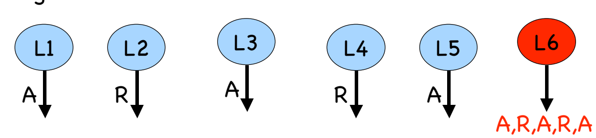
Decision??? m(A,A,A,A,R,R) ==> All loyal lieutenants attack!

Bigger Example: Bad Commander+

Scenario: m=2, n=7, traitors=C, L6



Messages?



Decision???

Decision with Bad Commander+

L1: m(A,R,A,R,A,A) ==> Attack

L2: m(R,R,A,R,A,R) ==> Retreat

L3: m(A,R,A,R,A,A) ==> Attack

L4: m(R,R,A,R,A,R) ==> Retreat

L5: m(A,R,A,R,A,A) ==> Attack

Problem: All loyal lieutenants do NOT choose same action

Next Step of Algorithm

Verify that lieutenants tell each other the same thing

- Requires rounds = m+1
- OM(0): Msg from Lieut i of form: "L0 said v0, L1 said v1, etc..."

What messages does L1 receive in this example?

- OM(2): A
- OM(1): 2R, 3A, 4R, 5A, 6A
- OM(0): 2{ 3A, 4R, 5A, 6R}
- 3{2R, 4R, 5A, 6A}
- 4{2R, 3A, 5A, 6R}
- 5{2R, 3A, 4R, 6A}
- 6{ total confusion }

All see same messages in OM(0) from L1,2,3,4, and 5

m(A,R,A,R,A,-) ==> All attack

Signed Messages

New assumption: Cryptography

A4) a. Loyal general's signature cannot be forged and contents cannot be altered

b. Anyone can verify authenticity of signature

Simplifies problem:

- When lieutenant i passes on signed message from j, know that i did not lie about what j said
- Lieutenants cannot do any harm alone (cannot forge loyal general's orders)
- Only have to check for traitor commander

With cryptographic primitives, can implement Byzantine Agreement with `m+2` nodes, using `SM(m)`

Signed Messages Algorithm: SM(m)

1. Commander signs v and sends to all as (v:0)
2. Each lieut i:

   A) If receive (v:0) and no other order

   1) Vi = v

   2) send (V:0:i) to all

   B) If receive (v:0:j:...:k) and v not in Vi

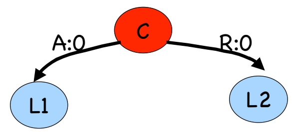
   1) Add v to Vi

   2) if (k<m) send (v:0:j:...:k:i) to all not in j...k

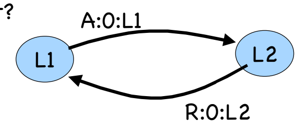
3. When no more msgs, obey order of `choose(Vi)`

SM(1) Example: Bad Commander

Scenario: m=1, n=3, bad commander



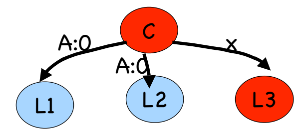
What next?



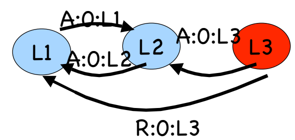
V1={A,R} V2={R,A}
Both L1 and L2 can trust orders are from C
Both apply same decision to {A,R}

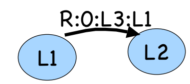
SM(2): Bad Commander+

Scenario: m=2, n=4, bad commander and L3



Goal? L1 and L2 must make same decision





V1 = V2 = {A,R} ==> Same decision

Other Variations

How to handle missing communication paths

< see paper for details>

Assumptions

A1) Every message sent by nonfaulty processor is delivered correctly
- Network failure ==> processor failure
- Handle as less connectivity in graph

A2) Processor can determine sender of message
- Communication is over fixed, dedicated lines
- Switched network???

A3) Absence of message can be detected
- Fixed max time to send message + synchronized clocks ==> If msg not received in fixed time, use default

A4) Processors sign msgs such that nonfaulty signatures cannot be forged
- Use randomizing function or cryptography to make liklihood of forgery very small

Importance of Assumptions

"Separating Agreement from Execution for Byzantine Fault Tolerant Services" - SOSP'03

Goal: Reduce replication costs

- 3f+1 agreement replicas
- 2g+1 execution replicas
  - Costly part to replicate
  - Often uses different software versions
  - Potentially long running time

Protocol assumes cryptographic primitives, such that one can be sure "i said v" in switched environment

What is the problem??