Distributed Computing with Byzantine Players

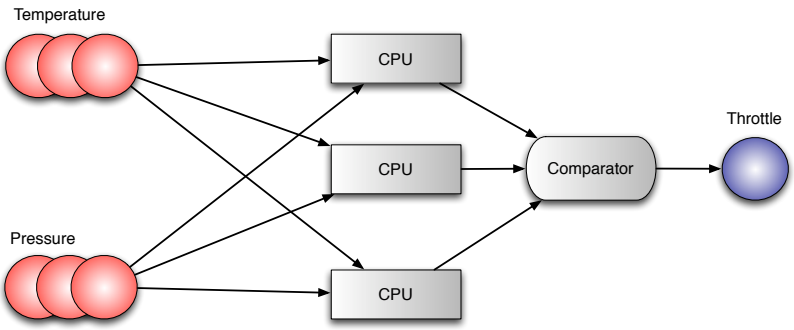
Sébastien Tixeuil
Sebastien.Tixeuil@lip6.fr

Motivation

Approach

- *Faults* and *attacks* occur in the network
- The network's user must *not* notice something wrong happened
- A *small* number of faulty components
- **Masking** approach to fault/attack tolerance

Principle



Problems

- Replicated input sensors may not give the same data
- Faulty input sensor or processor may not fail gracefully
- The system might not be tolerant to software bugs

Telling Truth from Lies

The Island of Liars and Truth-tellers

- An island is populated by two tribes
  - Members of one tribe **consistently lie**
  - Members of the other tribe **always tell the truth**
  - Tribe members can recognize one another, but an external observer can't

Puzzle I

- You run into a man and ask him if he is a truth-teller, but fail to hear the answer
- You inquire: "Did you say you are a truth-teller?"
- He responds: "No, I did not."
- To which tribe does the man belong ?

Puzzle II

- You meet a woman on the island.
- What single question can you ask her to determine whether she is a liar or a truth-teller?

Puzzle III

- You meet two people *A* and B on the island
- *A* says: "Both of us are from the liar tribe."
- Which tribe is *A* from ?
- What about *B* ?

Puzzle IV

- You meet two people, *C* and *D* on the island.
- *C* says: "Exactly one of us is from the liars tribe."
- Which tribe is *D* from ?

Puzzle V

- You meet two people *E* and *F* on the island
- *E* says: "It is not the case that both of us are from truth-tellers tribe."
- Which tribe is *E* from?
- What about *F*?

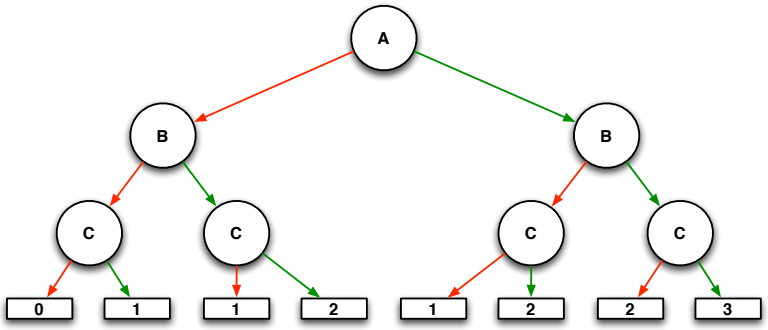
Puzzle VI

- You meet two people *G* and *H* on the island
- *G* says: "We are from different tribes."
- *H* says: "*G* is from the liars tribe."
- Which tribes are *G* and *H* from ?

Puzzle VII

- You meet three people *A*, *B*, and *C*
- You ask *A*: "how many among you are truth-tellers?", but don't hear the answer
- You ask *B*: "What did *A* say?", hear "one."
- *C* says: "*B* is a liar."
- Which tribes are *B* and *C* from?

Puzzle VII



The Island of Selective Liars

- Inhabitants lie consistently on Tuesdays, Thursdays, and Saturdays, and tell the through on the remaining days
- You ask: "What is today?" "Tomorrow?"
- Responses: "Saturday.", "Wednesday."
- What is the current day ?

The Island of Random Liars

- A new Island has three tribes
  - truth-tellers
  - consistent liars
  - randomly lie or tell the truth
- How to identify three representants of each tribe with only three yes/no questions?

Byzantine Generals

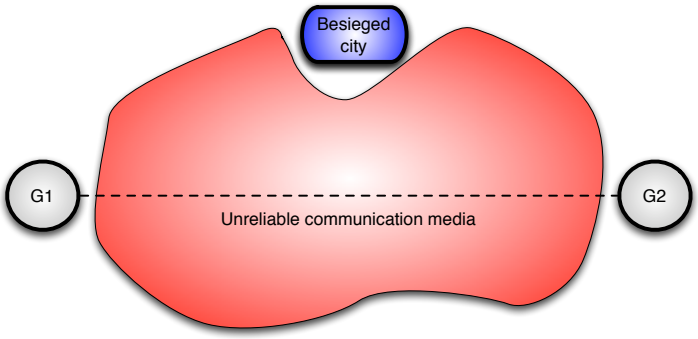
Settings

- Byzantine generals are camping outside an enemy city
- Generals can communicate by sending messengers
- Generals must decide upon common plan of action
- Some of the Generals can be traitors

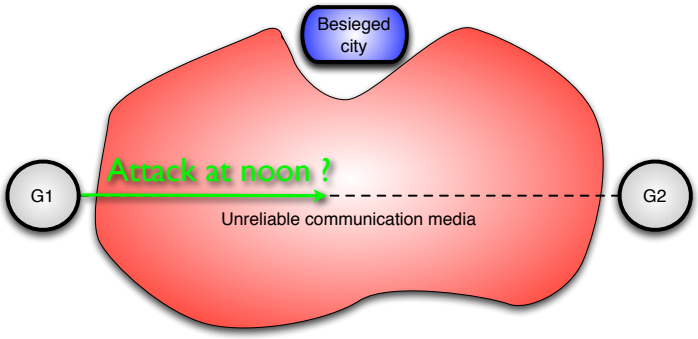
Goal

- All loyal generals decide upon the same plan of action
- A small number of traitors cannot cause the loyal generals to adopt a bad plan

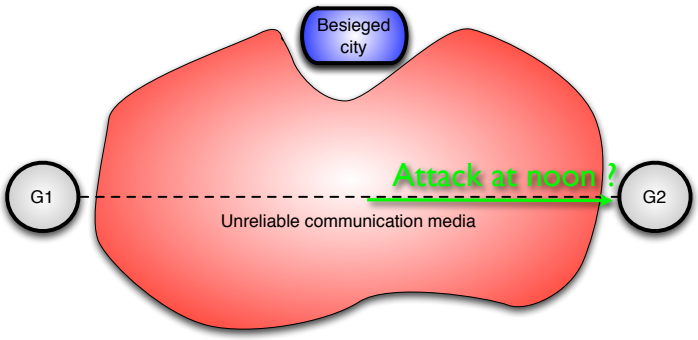
Two Generals Paradox



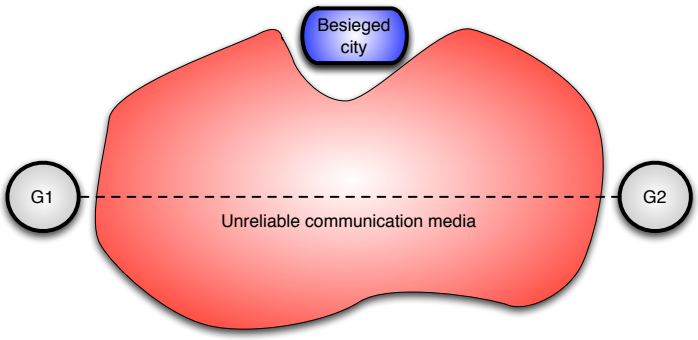
Two Generals Paradox



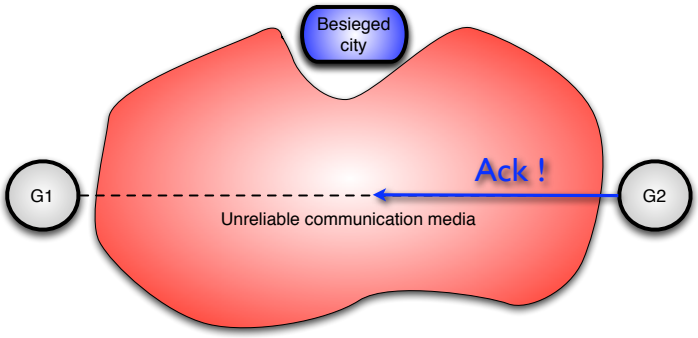
Two Generals Paradox



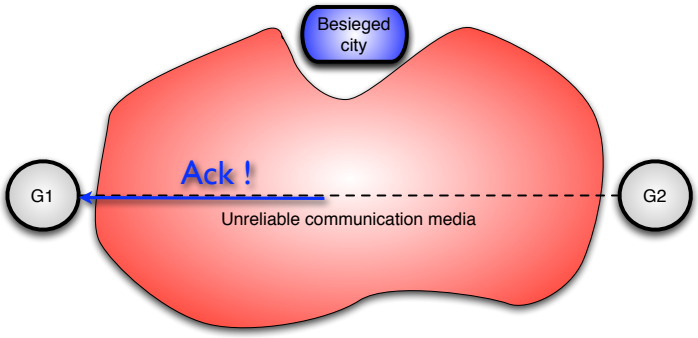
Two Generals Paradox



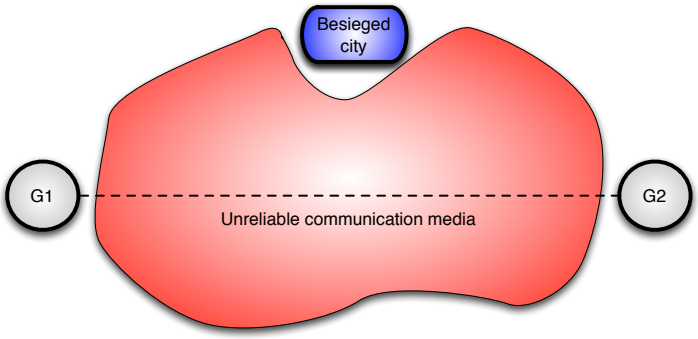
Two Generals Paradox



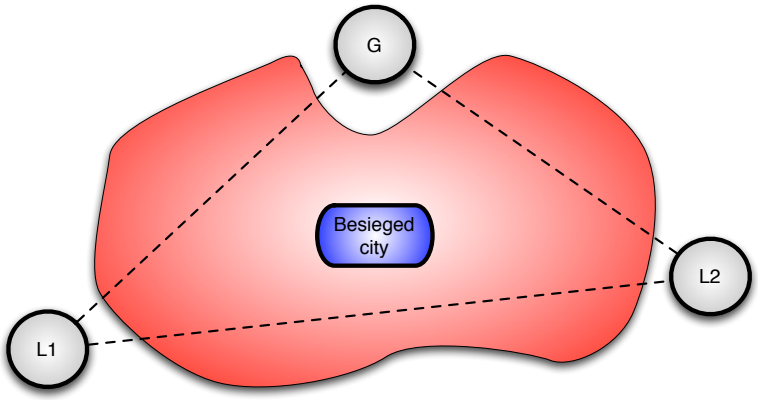
Two Generals Paradox



Two Generals Paradox



The Byzantine Generals Problem



The (simple) Byzantine Generals Problem

- Generals lead *n* divisions of the Byzantine army
- The divisions communicate via reliable messengers
- The generals must **agree** on a plan ("attack" or "retreat") even if some of them are **killed** by enemy spies

Oral Model

- **A1**: Every message that is sent is delivered correctly
- **A2**: The receiver of a message knows who sent it
- **A3**: The absence of a message can be detected

Solution?

plan: **array of** {A,R}; finalPlan: {A,R}

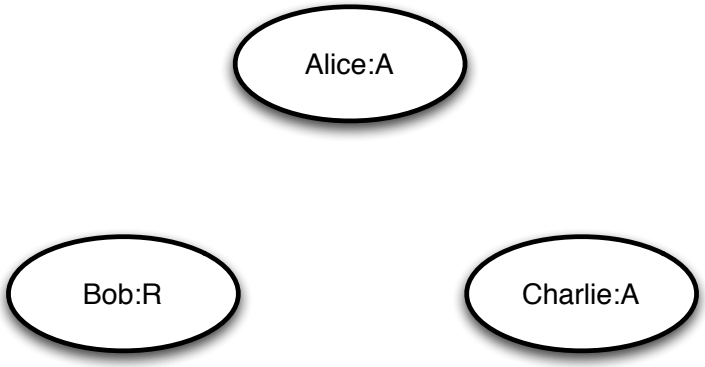
1: plan[myID] := *ChooseAorR*()

2: for all other G *send*(G, myID, plan[myID])

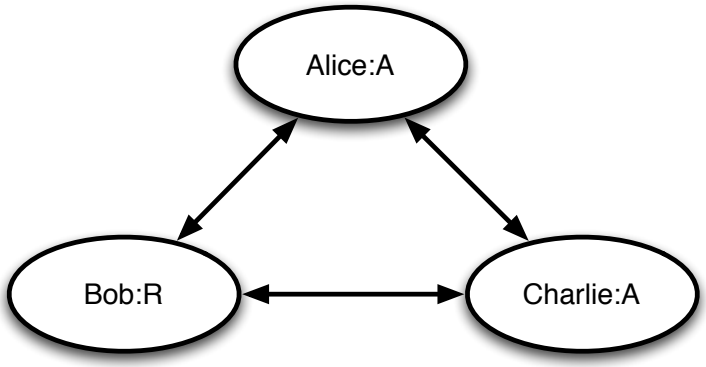
3: for all other G *receive*(G, plan[G])

4: finalPlan := *majority*(plan)

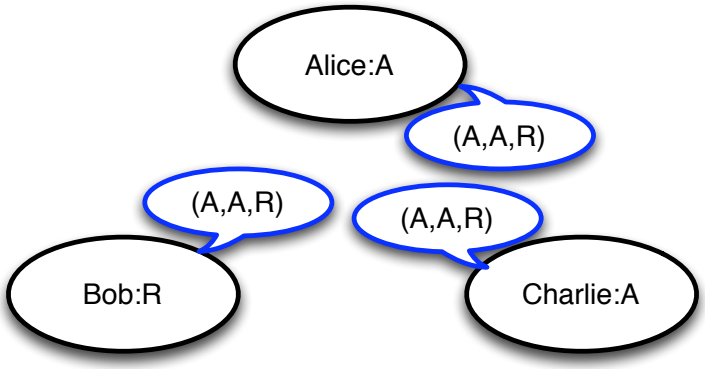
Reliable Networks



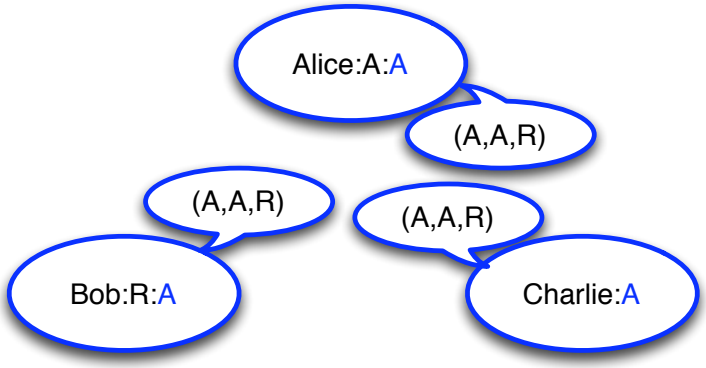
Reliable Networks



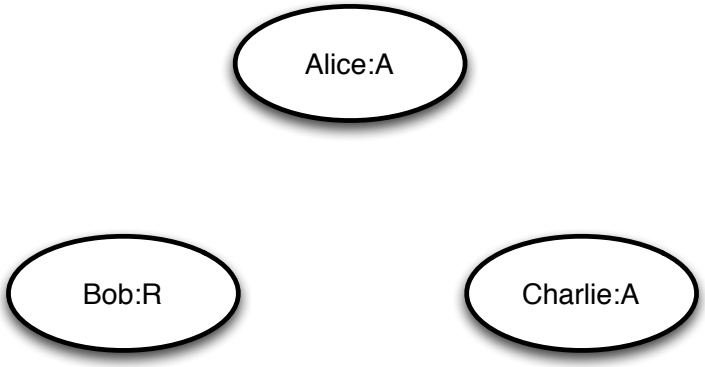
Reliable Networks



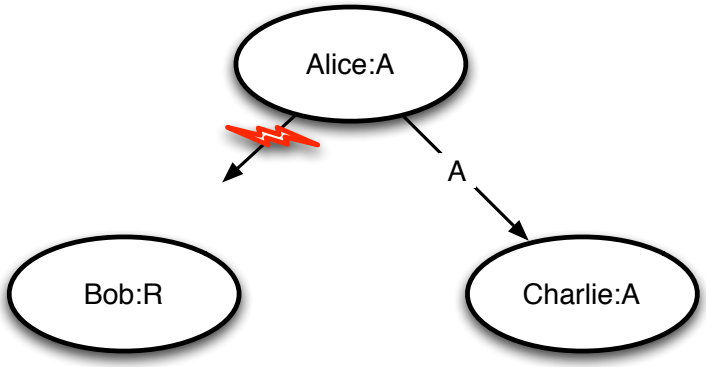
Reliable Networks



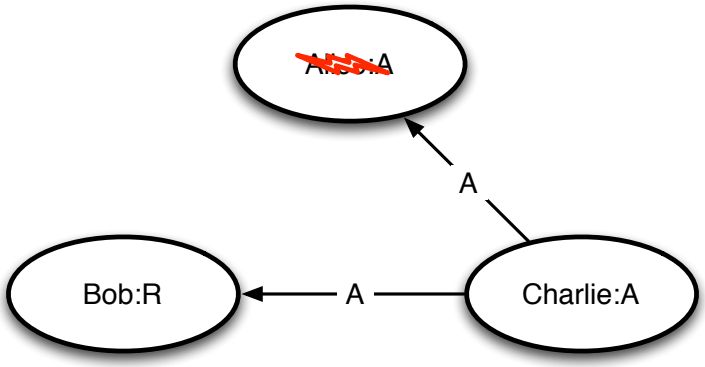
Crashing Networks



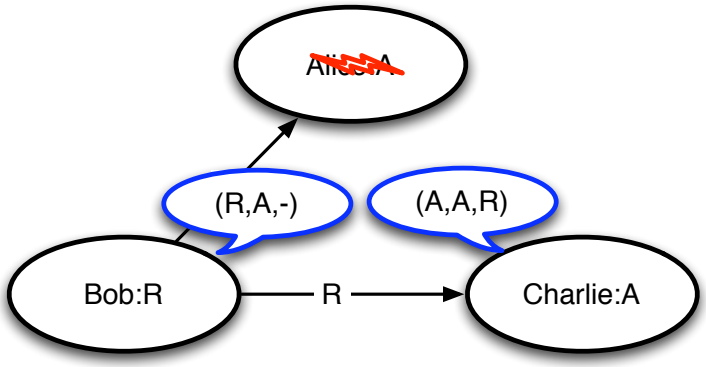
Crashing Networks



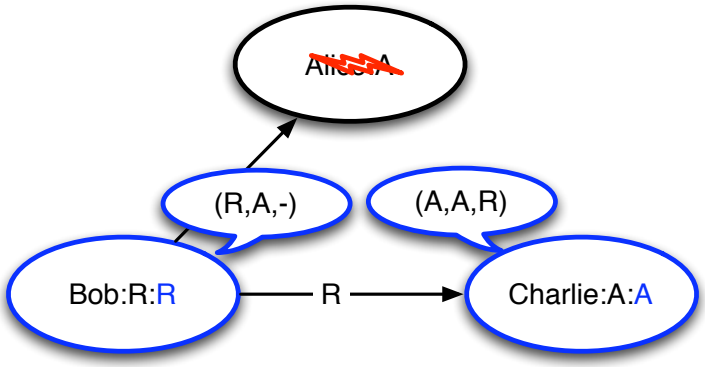
Crashing Networks



Crashing Networks



Crashing Networks



The Byzantine Generals Problem

- A general and *n-1* lieutenants lead n divisions of the Byzantine army
- The divisions communicate via messengers that can be captured or delayed
- The generals must **agree** on a plan ("attack" or "retreat") even if some of them are **traitors** that want to prevent agreement

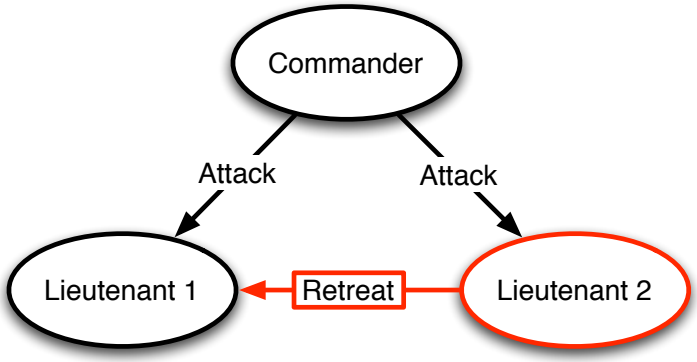
The Byzantine Generals Problem

- A commanding general must sent an order to his n-1 lieutenants generals such that
  - **IC1**: all loyal lieutenants obey the same order
  - **IC2**: if the commanding general is loyal, then every loyal lieutenant obeys the order he sends

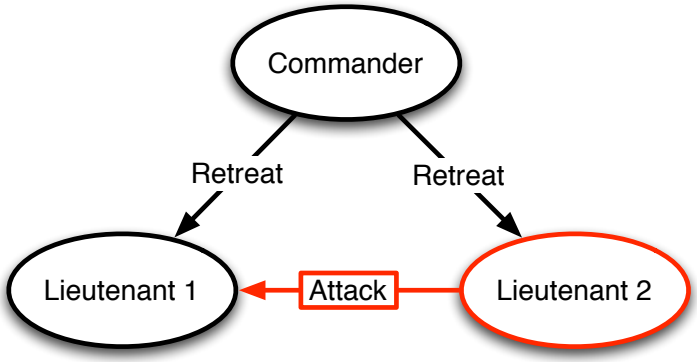
Oral Model

- **A1**: Every message that is sent is delivered correctly
- **A2**: The receiver of a message knows who sent it
- **A3**: The absence of a message can be detected

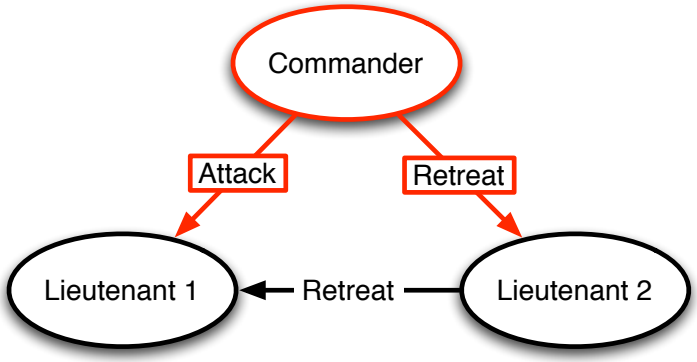
3k+1 nodes are necessary (oral model)



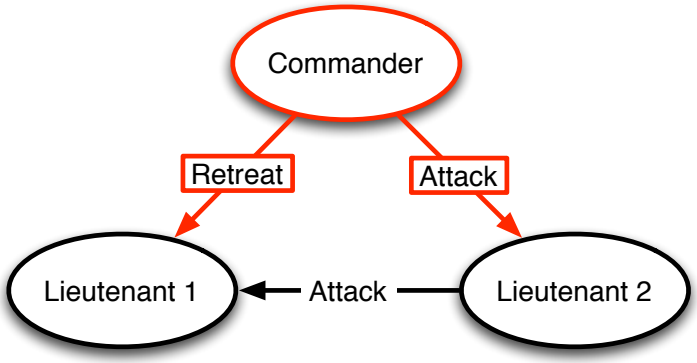
3k+1 nodes are necessary (oral model)



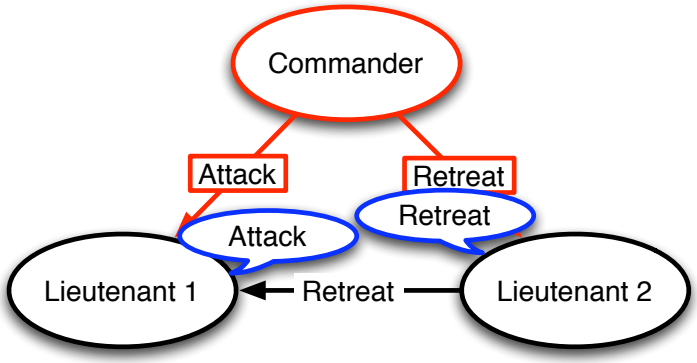
3k+1 nodes are necessary (oral model)



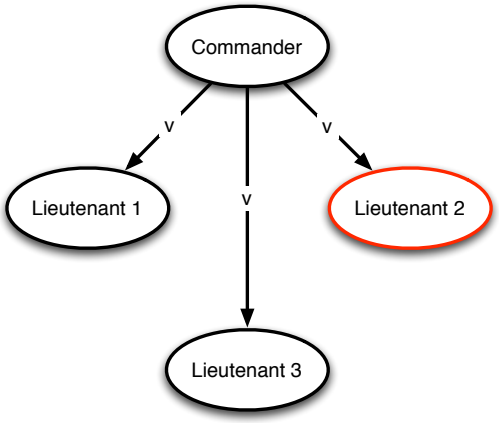
3k+1 nodes are necessary (oral model)



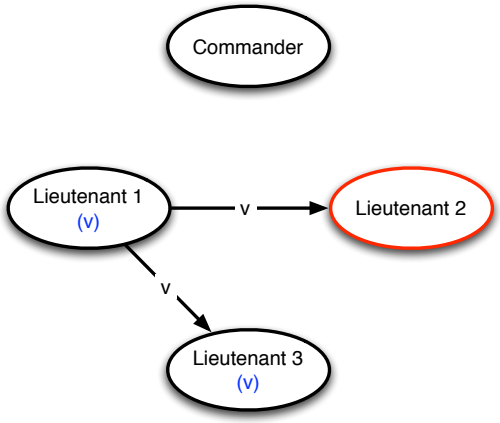
3k+1 nodes are necessary (oral model)



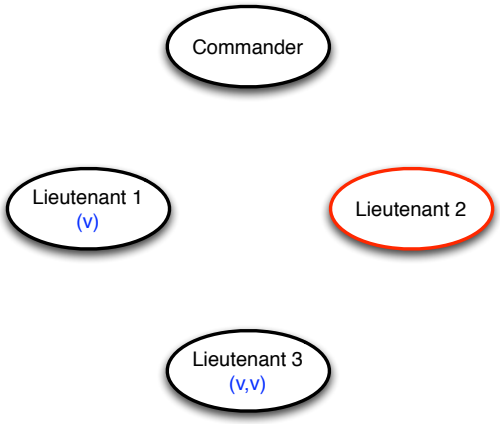
3k+1 nodes are sufficient (oral model)



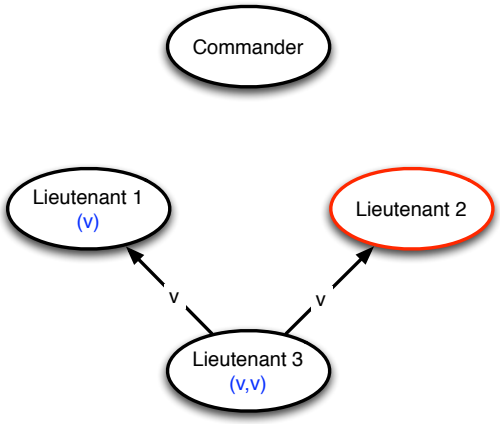
3k+1 nodes are sufficient (oral model)



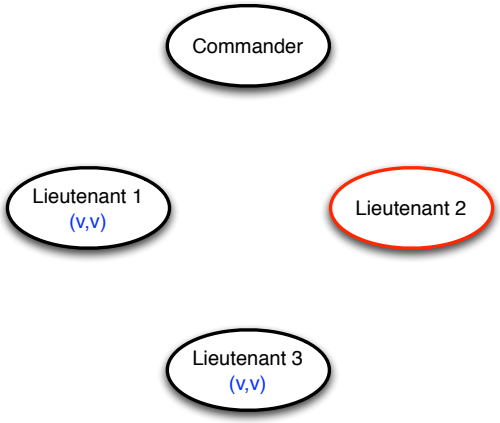
3k+1 nodes are sufficient (oral model)



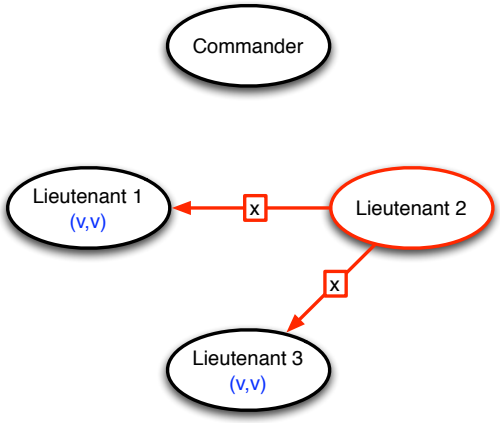
3k+1 nodes are sufficient (oral model)



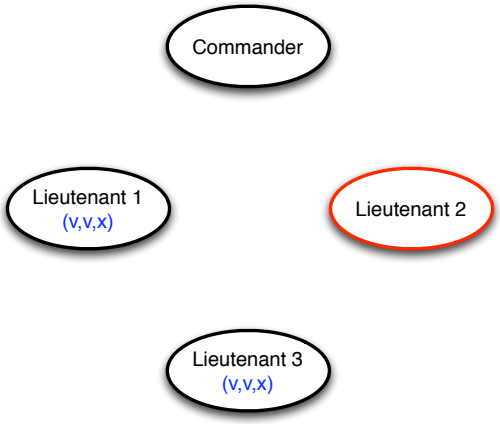
3k+1 nodes are sufficient (oral model)



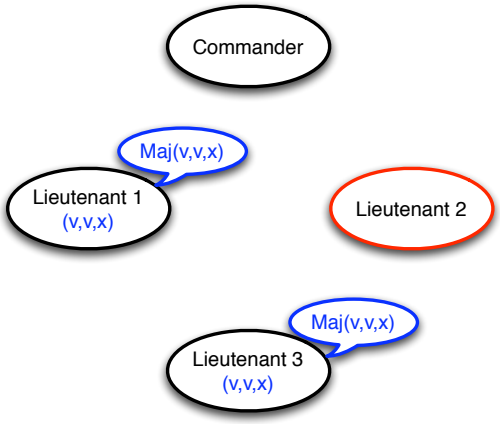
3k+1 nodes are sufficient (oral model)



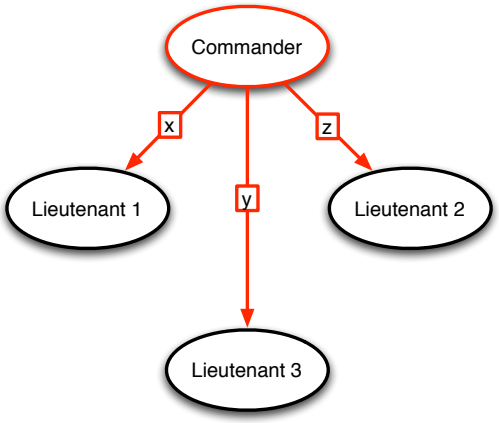
3k+1 nodes are sufficient (oral model)



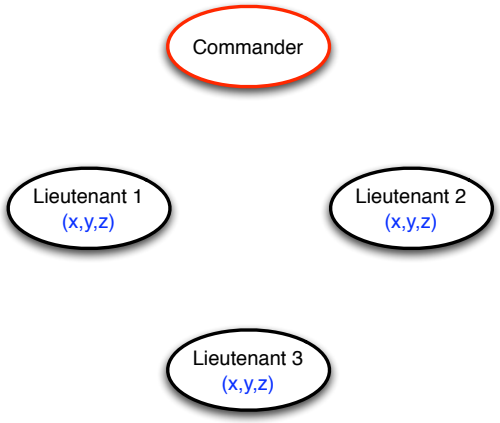
3k+1 nodes are sufficient (oral model)



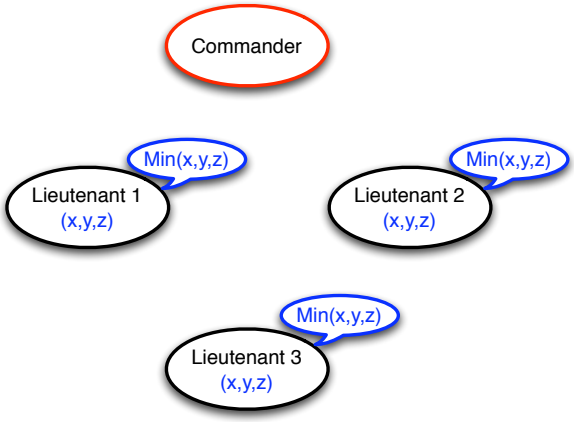
3k+1 nodes are sufficient (oral model)



3k+1 nodes are sufficient (oral model)



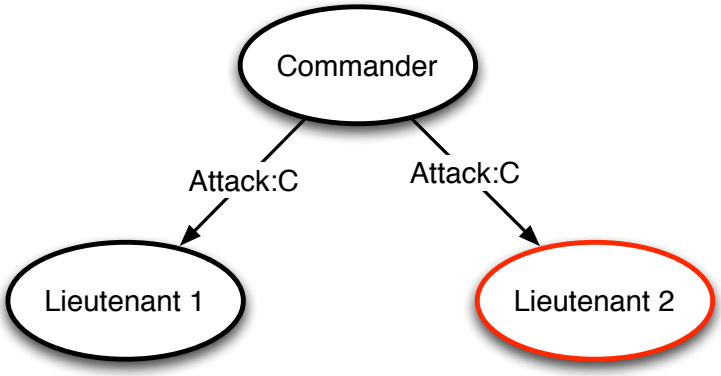
3k+1 nodes are sufficient (oral model)



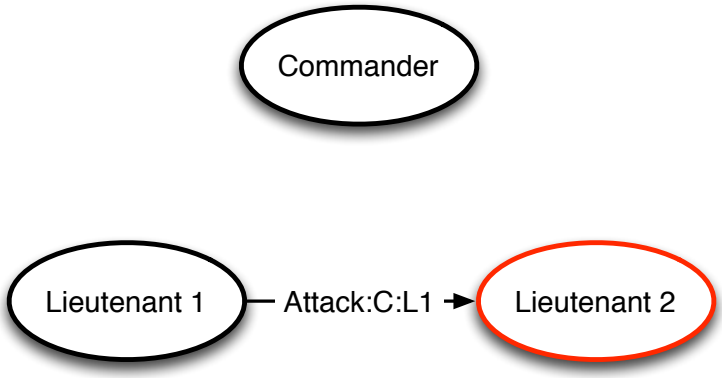
Written Model

- **A1-A3**: Same as before
- **A4**:
  - A loyal general's signature cannot be forged, and any alteration of the contents of his signed messages can be detected
  - Anyone can verify the authenticity of a general's signature

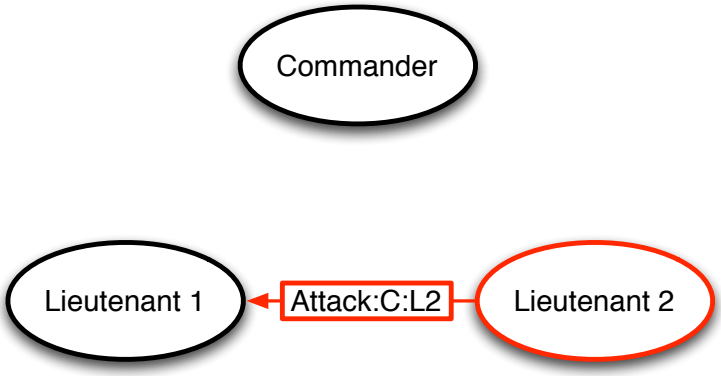
k+2 nodes are sufficient (written model)



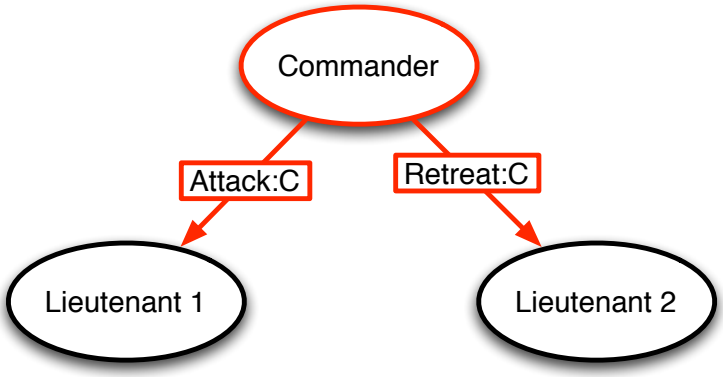
k+2 nodes are sufficient (written model)



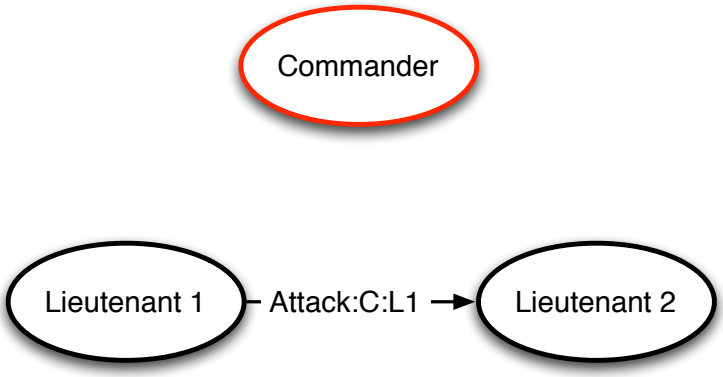
k+2 nodes are sufficient (written model)



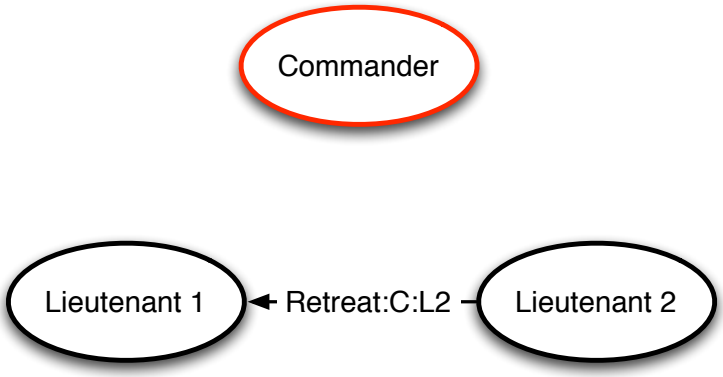
k+2 nodes are sufficient (written model)



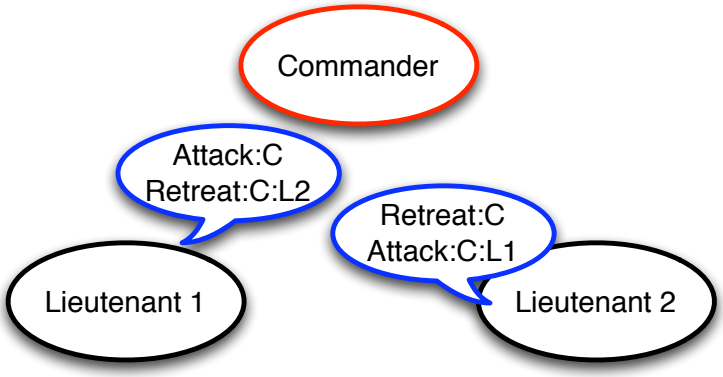
k+2 nodes are sufficient (written model)



k+2 nodes are sufficient (written model)



k+2 nodes are sufficient (written model)



Arbitrary Networks

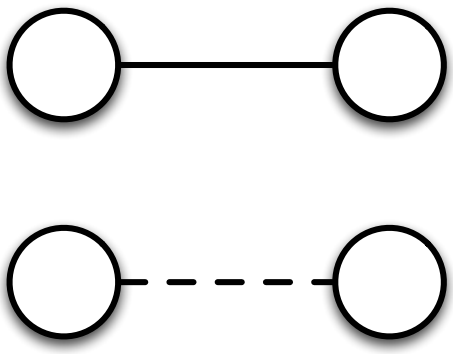
Topology Discovery

- **Given**
  - asynchronous network
  - up to *k* Byzantine nodes
  - each node knows its immediate neighbors identifiers
- **Goal**
  - each node must discover the complete network topology

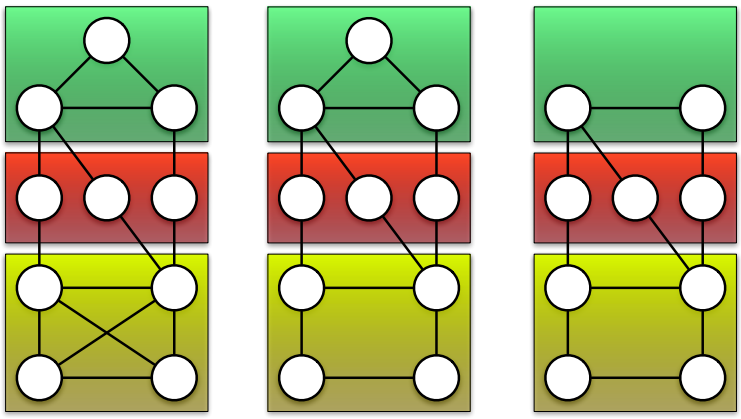
Weak Topology Discovery

- **Termination**
  - either all non-faulty processes determine the system topology or at least one detects fault
- **Safety**
  - for each non-faulty process, the determined topology is subset of actual
- **Validity**
  - fault detected only if it indeed exists

Weak Topology Discovery



Weak Topology Discovery



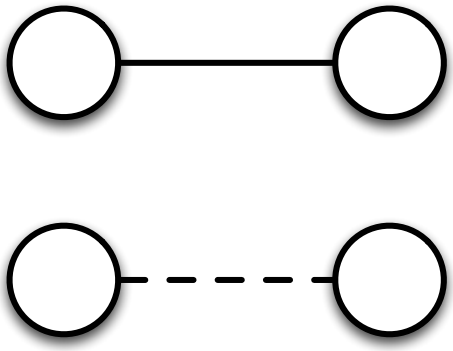
Weak Topology Discovery

- **Bounds**
  - cannot determine presence of edge if both adjacent nodes are faulty
  - cannot be solved if network is less than *k* +*1* connected

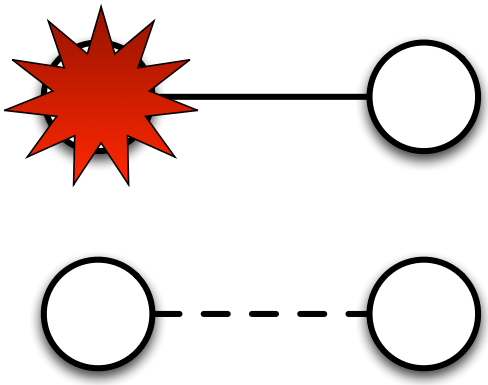
Strong Topology Discovery

- **Termination**
  - all non-faulty processes determine the system topology
- **Safety**
  - for each non-faulty process the determined topology is subset of actual

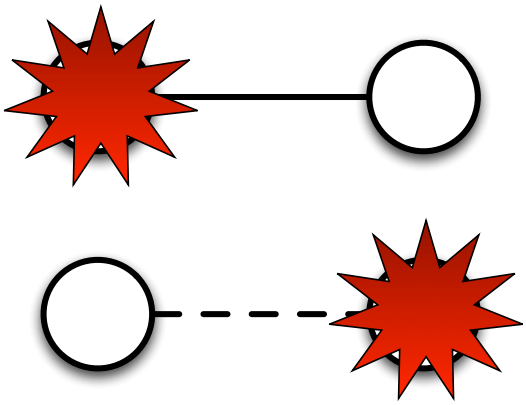
Strong Topology Discovery



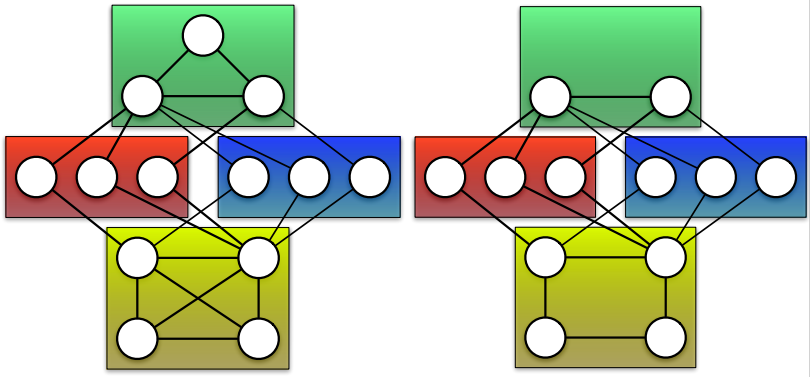
Strong Topology Discovery



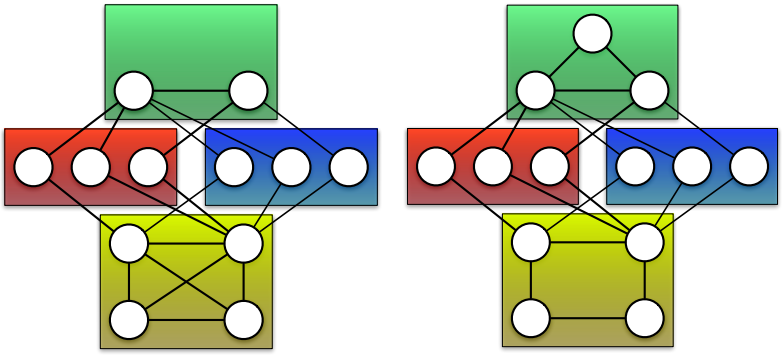
Strong Topology Discovery



Strong Topology Discovery



Strong Topology Discovery



Strong Topology Discovery

- **Bounds**
  - cannot determine presence of edge if one neighbor is faulty
  - cannot be solved if network is less than $2k+1$ connected

Solutions Preliminaries

- **Main idea**
  - *Menger's theorem*: if a graph is k connected then for any two vertices there exists two internally node-disjoint paths connecting them
  - a single (non-source) node cannot compromise info if it travels over two node-disjoint paths

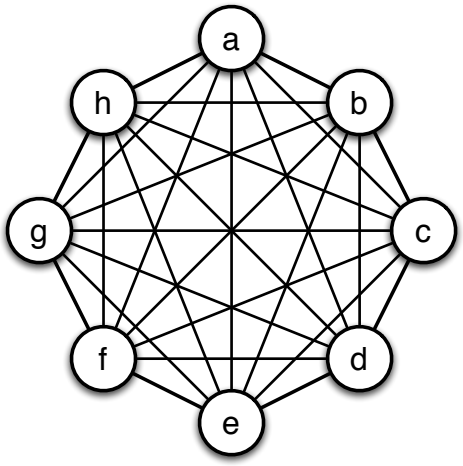
Solutions Preliminaries

- **Common Features**
  - every solution essentially involves flooding each node's neighbor info to the other nodes
  - solutions differ on how the nodes forward neighborhood info received from other nodes

A Naive Solution

- Store traveled path in message, forward message that contains simple path to all outgoing links
- Solves strong (and weak) topology discovery problems

A Naive Solution



A Naive Solution

- Store traveled path in message, forward message that contains simple path to all outgoing links
- Solves strong (and weak) topology discovery problems
- requires **exponential** number of messages

Detector

- **Basic design**
  - propagate neighbor info message for each process exactly once (*first time*)
    - if receive different info for same process, signal fault
    - since network is $k+1$ connected, info about non-faulty nodes reaches every node

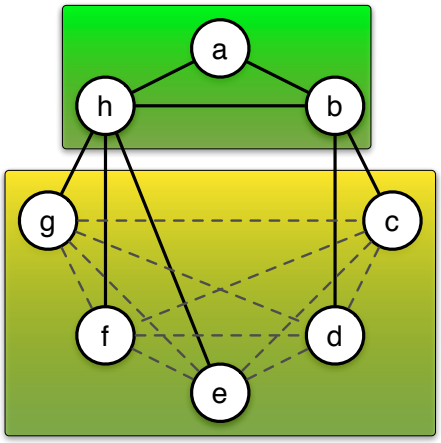
Detector

- **Handling fake nodes**
  - faulty process may send info about non-existent (fake) nodes thus compromising safety and termination
    - only faulty nodes can be connected to fake nodes ? (discovered network is less that $k+1$ connected)

Detector

- **Handling fake nodes**
  - faulty process may send info about non-existent (fake) nodes thus compromising safety and termination
    - when the network is not completely discovered yet, it may also be less than $k+1$ connected, problems with validity

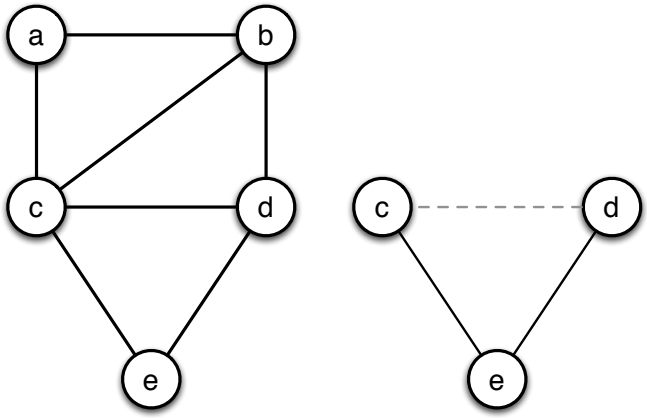
Detector



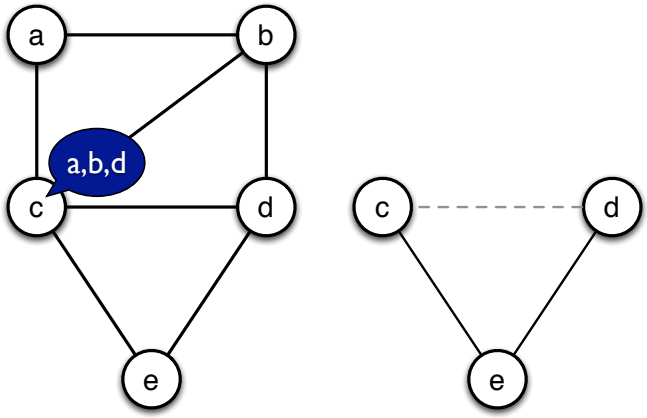
Detector

- **Neighborhood closure**
  - connect all nodes whose neighbor information is not received
    - the connectivity of this graph is no less than the actual topology
    - if the connectivity if this graph falls below $k+1$, signal fault

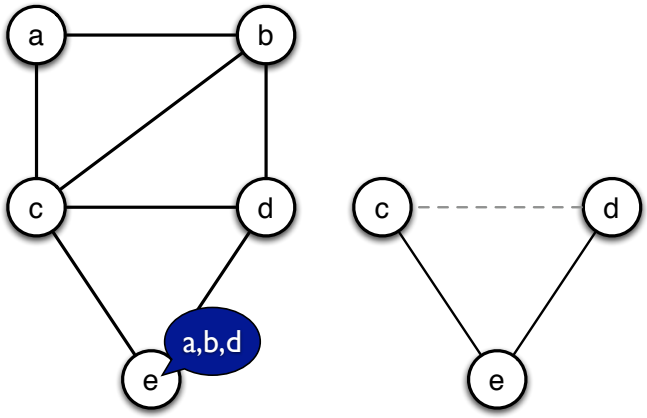
Detector



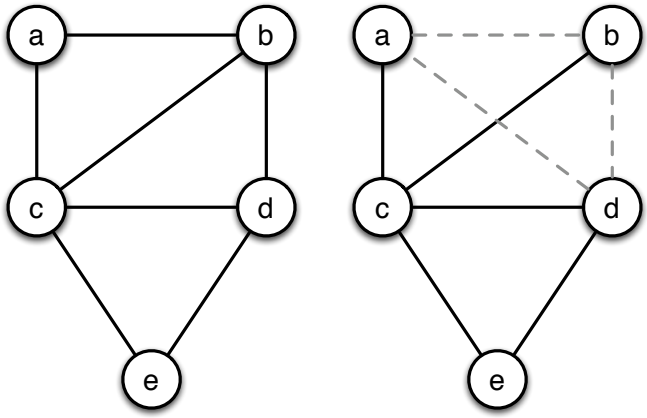
Detector



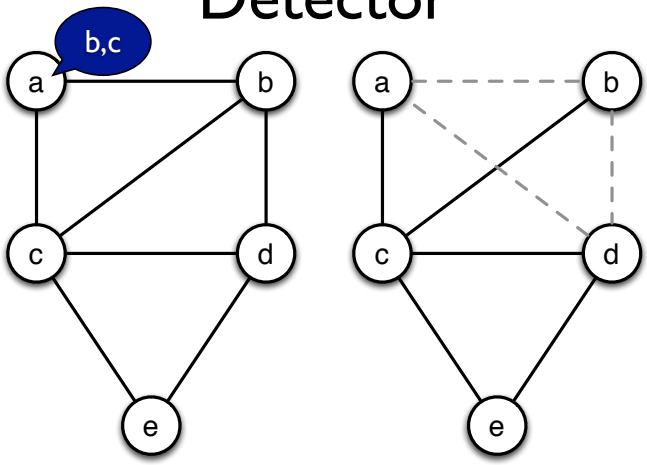
Detector



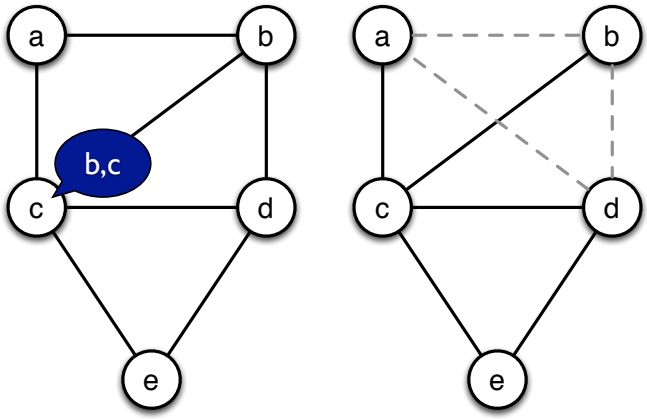
Detector



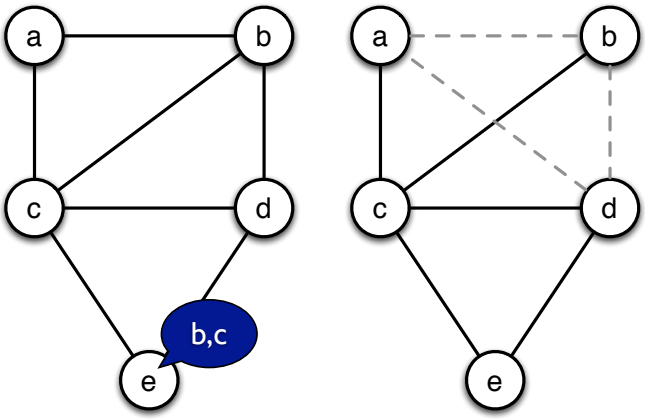
Detector



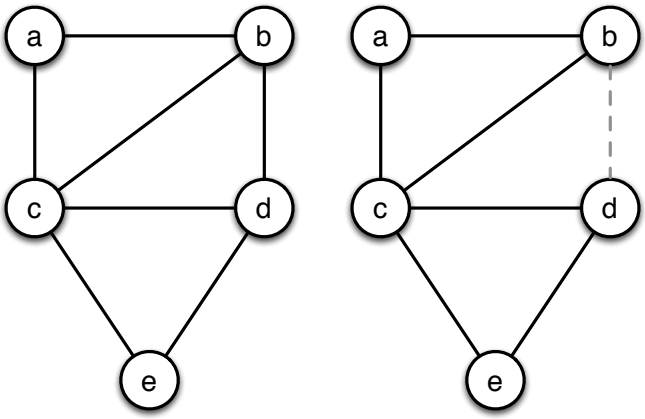
Detector



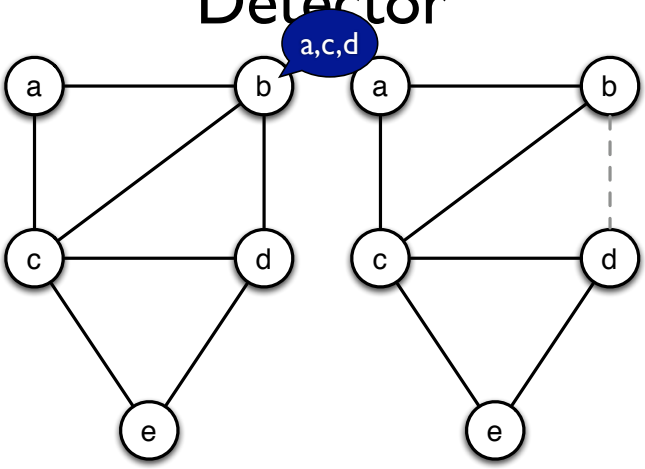
Detector



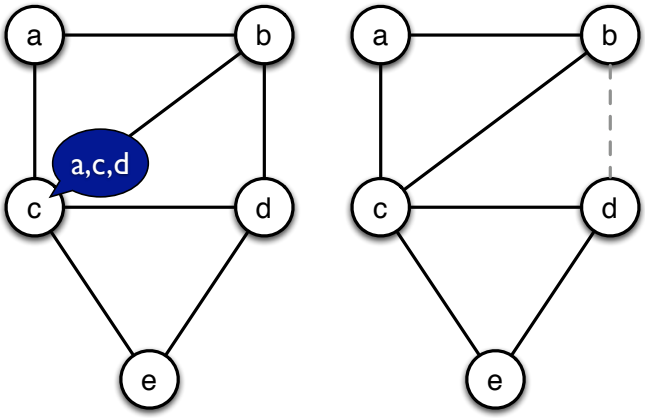
Detector



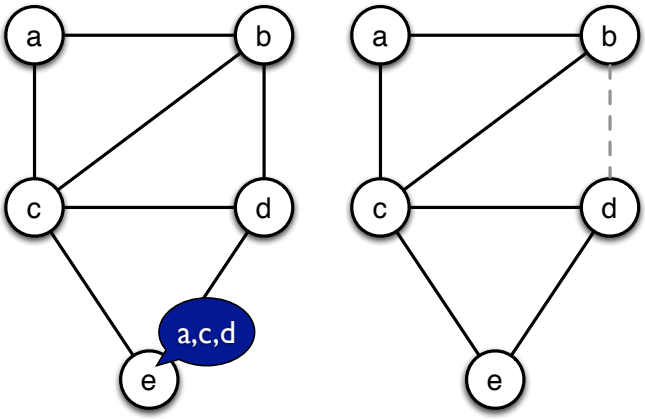
Detector



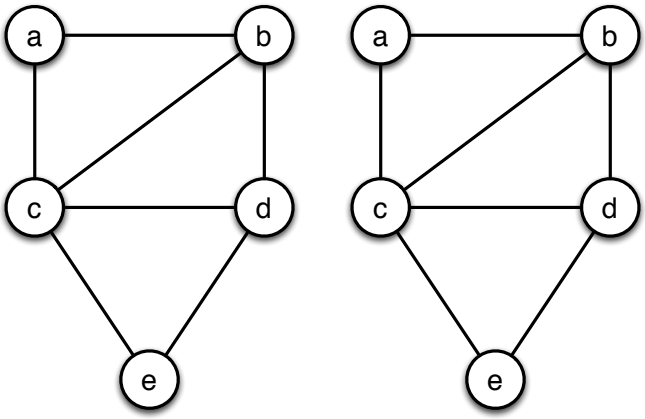
Detector



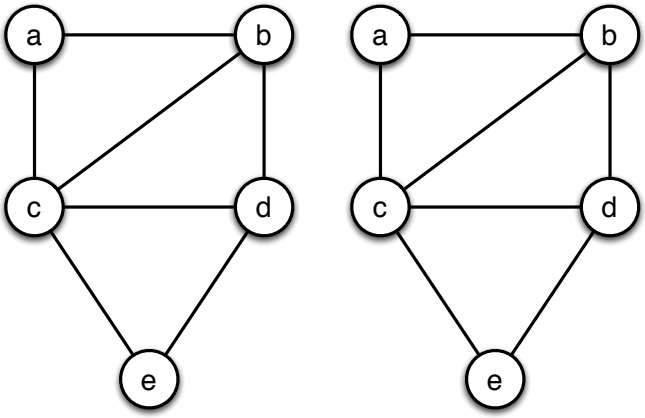
Detector



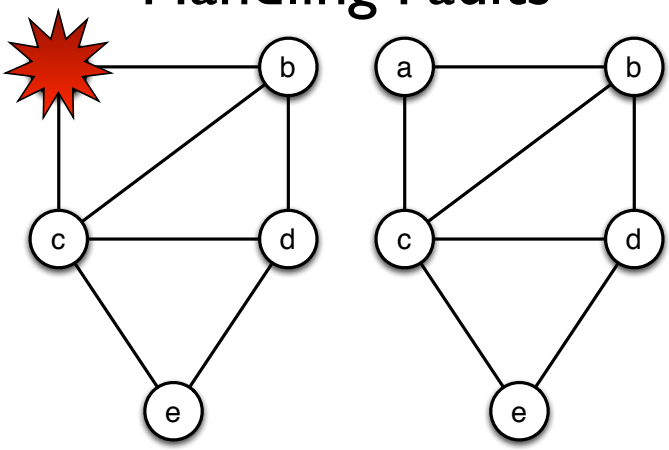
Detector



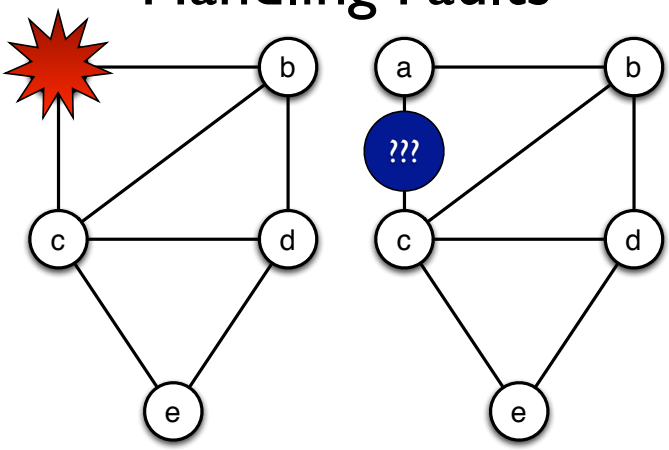
Handling Faults



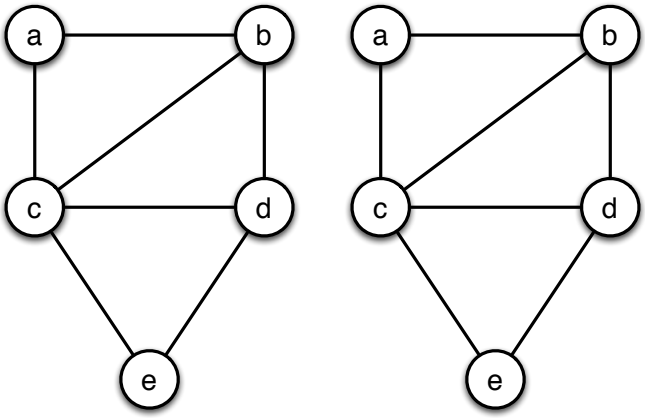
Handling Faults



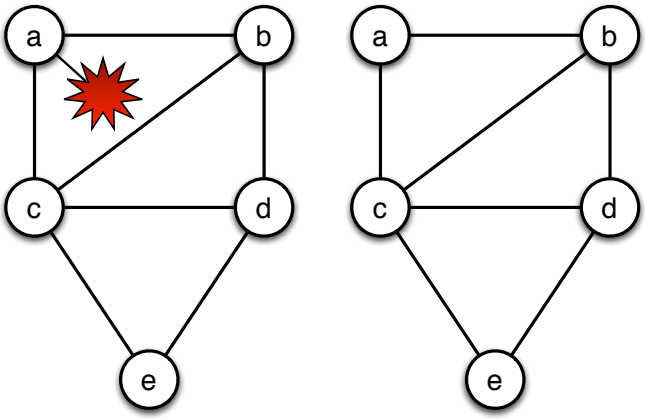
Handling Faults



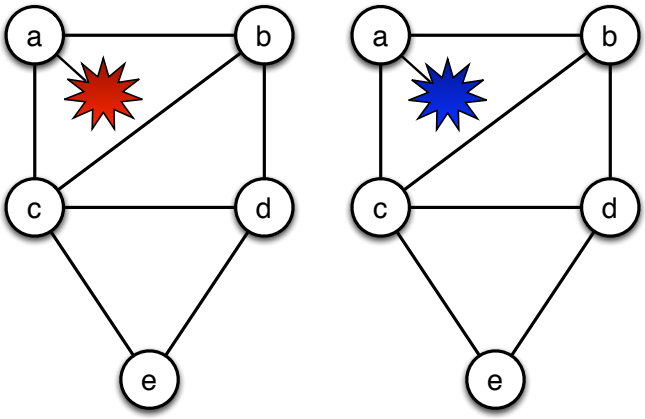
Handling Faults



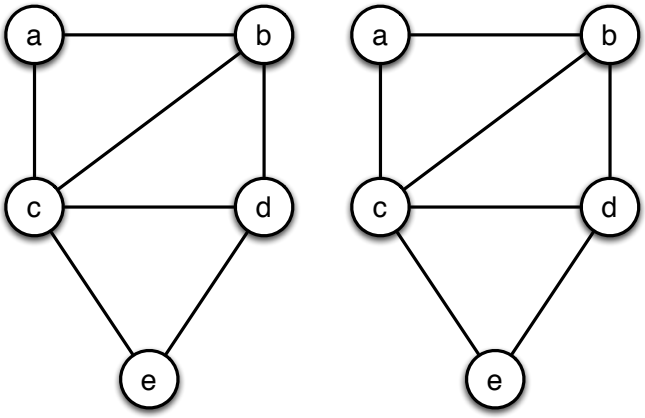
Handling Faults



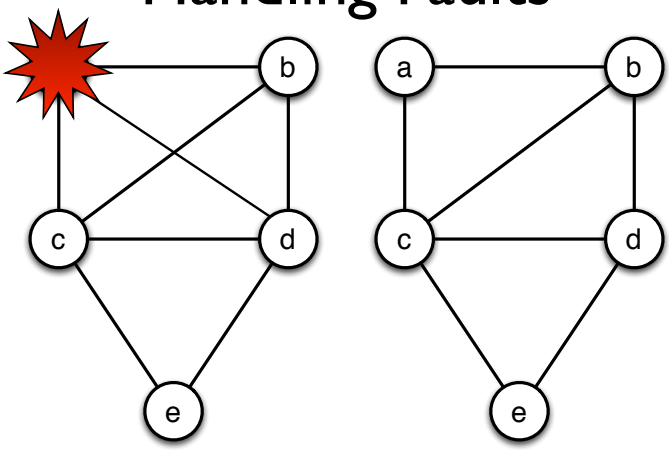
Handling Faults



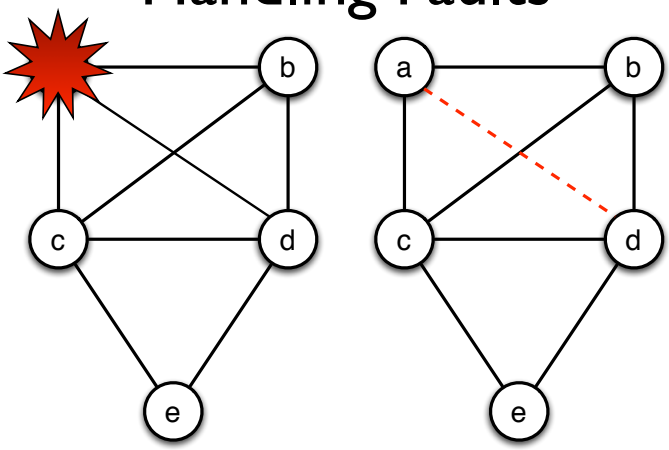
Handling Faults



Handling Faults



Handling Faults



Detector

- **Definition**
  - Solution is *adjacent-edge complete* if non-faulty nodes discover all non-faulty nodes and their adjacent edges

Detector

- **Theorem**
  - *Detector* is an adjacent-edge complete solution to the *weak topology discovery* problem if the connectivity of the system exceeds the maximum number of faults

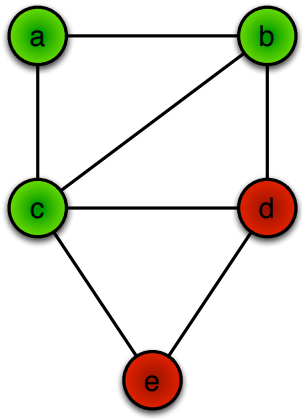
Explorer

- **Main idea**
  - collect node's neighbor information such that the info goes along more than twice as many node disjoint paths as max number of faulty nodes

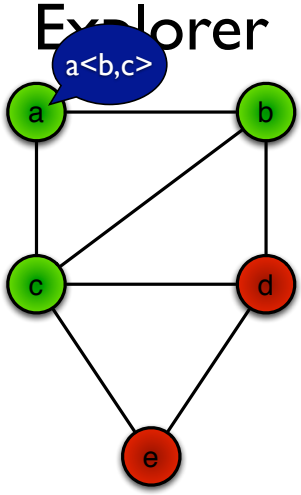
Explorer

- **Confirmed neighbor information**
  - *k+1* disjoint paths from source
  - non-intersecting paths from *k+1* confirmed neighbors

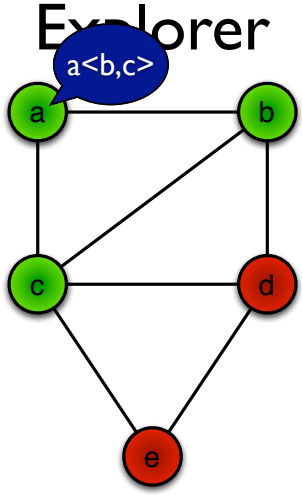
Explorer



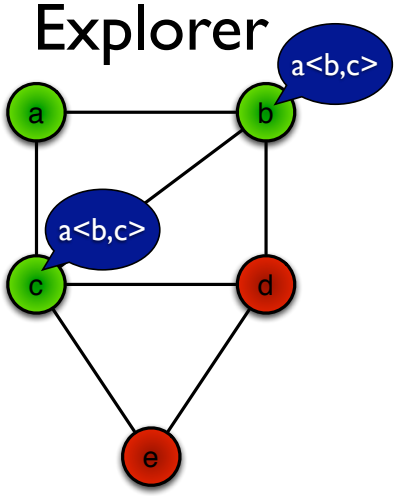
Explorer



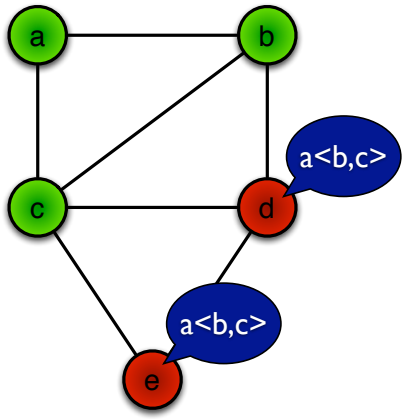
Explorer



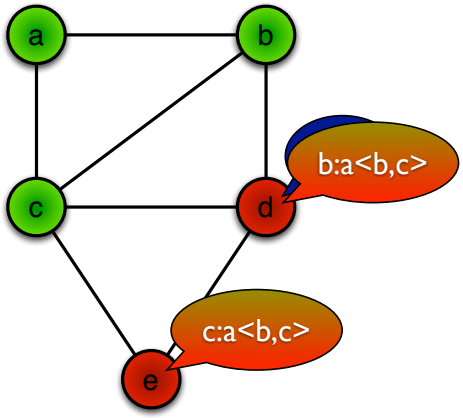
Explorer



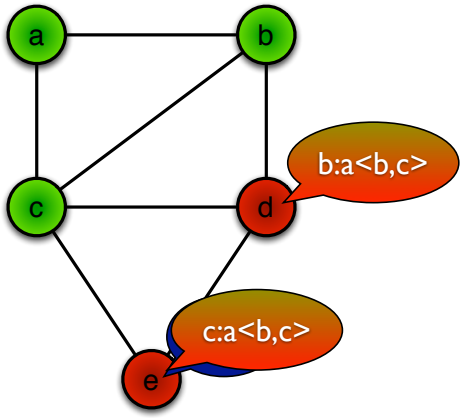
Explorer



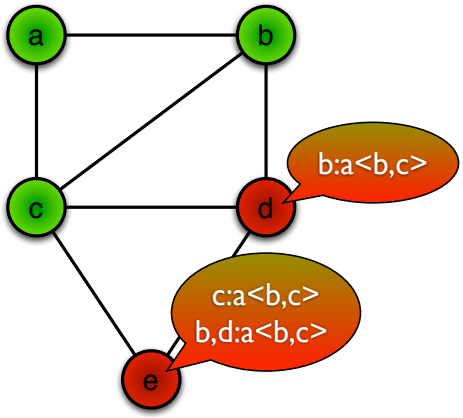
Explorer



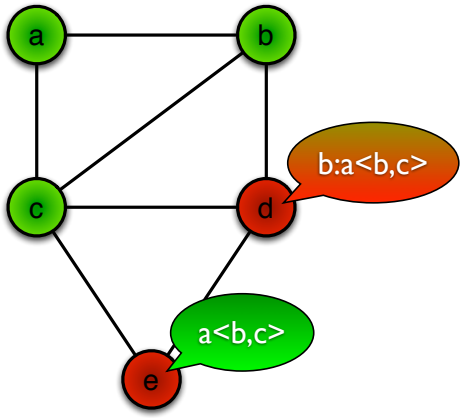
Explorer



Explorer



Explorer



Explorer

- **Definition**
  - Solution is *two-adjacent edge complete* if non-faulty nodes discover all non-faulty nodes and edges adjacent to two non-faulty nodes

Explorer

- **Theorem**
  - (generalized) *Explorer* is a two-adjacent-edge complete solution to the *strong topology discovery problem* in case the graph connectivity is more than twice the number of faults

Composing Detector and Explorer

- **Observation**
  - *Detector* uses less messages when there are no faults
- **Idea**
  - run *Detector*, if a node discovers fault, invoke *Explorer*
  - requires *2k+1* connected topologies

Malice in Online Video Games

Online Games

- First Person Shooter (FPS)
- Real-time Strategy (RTS)
- Role playing Game (RPG)
- Massively Multiplayer Online Game (MMOG)
- Sports, puzzles

Online Games

- First Person Shooter (FPS)
- Real-time Strategy (RTS)
- Role playing Game (RPG)
- Massively Multiplayer Online Game (MMOG)
- Sports, puzzles

Online Games

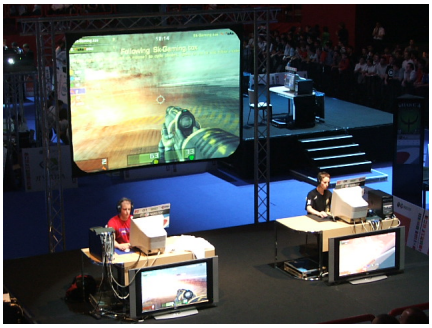
- First Person Shooter (FPS)
- Real-time Strategy (RTS)
- Role playing Game (RPG)
- Massively Multiplayer Online Game (MMOG)
- Sports, puzzles

Online Games

- First Person Shooter (FPS)
- Real-time Strategy (RTS)
- Role playing Game (RPG)
- Massively Multiplayer Online Game (MMOG)
- Sports, puzzles

E-sport

- International competitions (ESWC, WCG, WSVG)
- Prizes over $1 million
- Professional leagues
- Professional players with sponsors, coachs ...
- In some countries, e-players are really famous



Online Cheat

- **First major online cheat**: Diablo 1997
- **FPS**: aim bot, aim proxy (Quake, Counter strike)
- **RTS**: maphack (Warcraft, Age of Empires)
- **1999-2000**: awareness of industry
- **recently**: gaming bots in MMOG (World of Warcraft)

Architectures

- **Client Server**
  - safer, server is trustable
  - "easy" to design
  - "centralized"
  - expensive, not scalable, faults ?
- **Peer-to-peer**
  - scalable
  - cheap
  - autonomous
  - difficult to design, cheating is easier

Binaries Protection

- **Avoid client-side modifications**
  - avoid unauthorized behaviors
  - ensure clients follow the same protocol
- [Munch06] proposes to execute dynamic verifications named mobile agents

Detection Mechanisms

- **Sometimes it is not possible to prevent cheating**
  - Keep log and verify afterwards [Kabus05]
  - Runtime verification of rules [Delap04]
- **Detection against Prevention**
  - Latency constraint are very high, prevention needs many message exchanges impacting this latency

Protocols

- **Enforcing fairness in spite of various latencies**
  - [Aggarwal05] on dead-reckoning
  - [Guo03] removing unfair advantage of low delay
- **Synchronisation protocols**
  - [Baughman01] lockstep protocol
  - [GD04] lockstep with improvements

Example: Synchronization

- Each round, every client sends its timestamped update
- Timestamps are needed to balance latency
- The server updates the world simulation using timestamps
- The server broadcasts the new game view
- If a message is late, the server modifies the view

Example: Synchronization

- **Problem**
  - Because latency may vary, timestamps are not verified
  - Malicious clients may ``know'' the future.

Example: Lockstep Protocol

- Each client sends to every other a commit of its update
- When every client has received every other update, they send the clear update
- The game view is updated and broadcast
- Performance issue: a late message freezes all messages

Defeating Maphack

- **In RTS, maphack is to be avoided**
  - Game clients are not trustable
  - Any information that leaked may be revealed
  - *Zero-Knowledge* Protocols

Defeating Maphack

- Consider two players such that:
  - Player *1* has value *A*
  - Player *2* has value *B*
- **Question**: How to know whether *A=B* without revealing *A* or *B* if *A!=B*
- **Bad solution**: exchange *hash(A)* and *hash(B)* and then compare

Defeating Maphack

- Let *f* and *g* be two commutative cryptographic functions respectively known only to *P1* and *P2*
  - *f(g(A)) = g(f(A))* for any *A*

Defeating Maphack

- *P1* computes *f(A)*
- *P1* sends *f(A)* to *P2*
- *P1* computes *f(g(B))*
- *P1* sends *f(g(B))* to *P2*
- if *f(g(B))=g(f(A))* then *A=B*

- *P2* computes *g(B)*
- *P2* sends *g(B)* to *P1*
- *P2* computes *g(f(A))*
- *P2* sends *g(f(A))* to *P1*
- if *f(g(B))=g(f(A))* then *A=B*

Roadmap

- Currently designing a P2P version of World of Warcraft server, that will be later used as a basis for experimenting malice-resilient protocols on a ``real'' platform.
- Malice-proof protocol design and implementation

Conclusion

Conclusion

- **Goal**: mask faults and attacks to the user
- **Basic principle**: redundancy and majority
  - not necessary to identify who misbehaves
  - most people must be reliable
  - protocols are much easier with cryptography (but how is crypto set up?)

Pros

- Masks the faults and attacks to the user
- Natural way to cope with failures
- Many protocols are available
  - Consensus, Atomic commit, Reliable Broadcast, Renaming,...

Cons

- Network must be properly initialized
- Global knowledge is assumed
  - size, names, maximum number of faults,...
- Global communication is used
- Global synchrony is assumed