# TELE9752 Network Operations and Control

## Week 7: Fault Management

# Outline

## Part 1: Dependability

- Engineering context
- Defining dependability
- Measuring availability and reliability
- Faults, errors and failures
- Failure modes
- Responding to faults
- Preparing to fail

## Part 2: Event Correlation

- Multiple symptoms
- Rule Based Reasoning
- Codebook Reasoning

# References


A. Fox and D. Patterson: "When does fast recovery trump high reliability?", *Proc. 2nd Workshop on Evaluating and Architecting System Dependability*

D. Oppenheimer et al: "Why do internet services fail, and what can be done about it?", *Proc. 4th USENIX Symp. on Internet Technologies and Systems*

A. Bressen: "RITA -- The Reliable Internetwork Troubleshooting Agent", IETF RFC 2321, *1 Apr* 1998

L. Lewis: "Event correlation in spectrum and other commercial products", Aprisma White Paper

S. Kliger et al: "A Coding Approach to Event Correlation", *Proc. 4th Int'l Symp. on Network Management*






2XQ

# Broader engineering context

Service providers are acutely concerned with faults because faults

- degrade reputations
- have monetary costs: Lack of revenue during fault; penalty fees for violating Service Level Agreements.

But things fail; not just networks but every*thing*.

=> relevant to wider engineering community

Reliability engineering techniques developed for other services

- often expected to operate continuously (e.g. road bridge) rather than discrete packet transfer

"if the automobile had followed the same development cycle as the computer, a Rolls Royce would today cost $100, get a million miles per gallon, and explode once a year, killing everyone."
– Robert X. Cringely
"Notes from the Field" column in InfoWorld magazine, 6 Mar 1989

# Networks differ from many systems

Problems can be intermittent / hard to reproduce:

- Can't control workload: How create exactly the same trace of packets seen on a production network, perhaps even with same content?
- Deliberate randomness in protocols in order to spread load

Global Internet

- Demands continual access to servers: can't close network for night/holiday testing
- Can't power cycle system to return to known state

Units of service may be discrete

- e.g. client-server web transactions; datagrams
- Perhaps measure reliability by % of requests that receive response, rather than measuring periods?

# 4 attributes of IT "dependability"

**Availability: Readiness for usage**

e.g. web server that accepts few concurrent connections may rarely be available when many clients seek access

**Reliability: Continuity of correct service**

e.g. dynamic client IP address may break TCP connections after some time

(A & R are subtle when service is discrete & incur delays)

**Safety: Avoid "catastrophic" consequences**

catastrophe: value(conseq.)>>value(normal service)

e.g. 000/911 phone call must get through $(life)>>25c

**Security: Prevent unauthorised access *to info***

Ensures secrecy & authenticity.

DOS affects A&R, but not "Security" as defined here

A. Avizienis et al: "Basic Concepts and Taxonomy of Dependable and Secure Computing", *IEEE Trans. Dependable and Secure Computing*

# Telco safety example

Telstra users hanging online
By Fia Cumming
March 31 2002, http://www.smh.com.au/articles/2002/03/30/1017206160054.html

Telstra's widespread use of split lines to create second telephone services to homes is the subject of a dossier being compiled by the Telecommunications Industry Ombudsman. The Federal Opposition claims the use of what are called "pair gains" instead of new lines has left up to a million families with slow internet technology. The faulty phone lines implicated in the asthma death of 10-year-old Sam Boulding in February were also pair gain lines. Telstra has admitted that when a customer has asked for an extra phone number, its technicians routinely split the existing line to create two. As a result of the "pair gain", the available bandwidth is also split, limiting internet capacity on both lines. Even with a fast computer, customers cannot access the new broadband service promoted by Telstra. The lines are also likely to be hit by the same faults, so if one goes down the other is not available. A spokeswoman for the ombudsman said the office had only recently become aware that pair gains could be the cause of many problems. "We haven't had a lot of complaints that we have identified as pair gains complaints because obviously people wouldn't have known that was the cause," she said. Sam Boulding died on February 6 after his parents were unable to use their faulty home phones to contact emergency services when he had an asthma attack. A report by PricewaterhouseCoopers, commissioned by Telstra, revealed the two lines in the Boulding house in north-east Victoria were pair gains, and that both were faulty around the time Sam died. Labor's information technology spokeswoman, Kate Lundy, said Telstra charged the same for pair gain lines and had an obligation to tell customers when it installed them. Telstra said the use of pair gains was accepted technology and without it the new lines could not have been delivered.

# FCAPS links

**FCAPS topics are interrelated**

e.g. Unavailability due to fault (F) or Denial of Service (S)

**Each topic will have a slide relating it to other topics**

Avoid duplication by topics only referring to preceding topics => No links for this lecture

**Link slides in future lectures:**

Configuration [F9>

Accounting [4R>

Performance [YH>

Security [QD>

# **F**CAPS links

Fault management:

- <K1] Tension between security & NM: Elaborating on why something was refused helps F but hinders S
- Attacks and faults cause similar degradations, leading to similar protection (e.g. Integrity checks) but stronger for security <8L]
- Security is part of dependability
  - as are availability and reliability
  - Denial of Service impinges availability
- Monitoring/Anomaly detection: NM detects faults; Security to detect intrusions

[Some security mechanisms† use OIDs, ASN.1 and BER
Same IT people often do both security and NM]





† e.g. Kerberos and X.500

# Causes 1: Accidents vs malice

Deliberate malicious attack vs accidents:

- **have same *consequences***, e.g. service not available, or info changed value
  => often deal with both using same mechanisms
  => dependability and “security” intersect
- **but may have different profiles**
  e.g. bit errors expected to be random - equally likely to affect each bit. Highly unlikely to change one sentence into another.
  => need stronger protection against malicious attack
  e.g. for integrity: secure hash function vs checksum

# Causes 2: Origin

- Environmental, e.g. fire, back-hoe through fibre
- Intrinsic:
  - platform: “hardware softens with age, software hardens with age”, i.e.
    - physical/mechanical wear accumulates over time
    - running software for longer increases chance of exploring all possible states & identifying (& correcting) all latent bugs.
  - system state: e.g. router may learn misinformation
- Configuration: 51% of web site failures are due to operator error. [Fox and Patterson]

# Outline

# Binary services

**We'll only consider binary services**: System either does or doesn't provide service (work) properly

**More sophisticated treatments** are also possible:

- "Performability": account for performance level
  - crude conversion is that binary service is OK if performance exceeds some threshold
- Account for user-demand
  - "customer minutes lost"
  - warning users about outage, e.g. planned vs unplanned outages (aka scheduled vs unscheduled outages)
  - Different failure modes ->

# Failure modes

When an element fails, what does it do?

Fail-stop: System does nothing when it has failed.
e.g. when power is lost

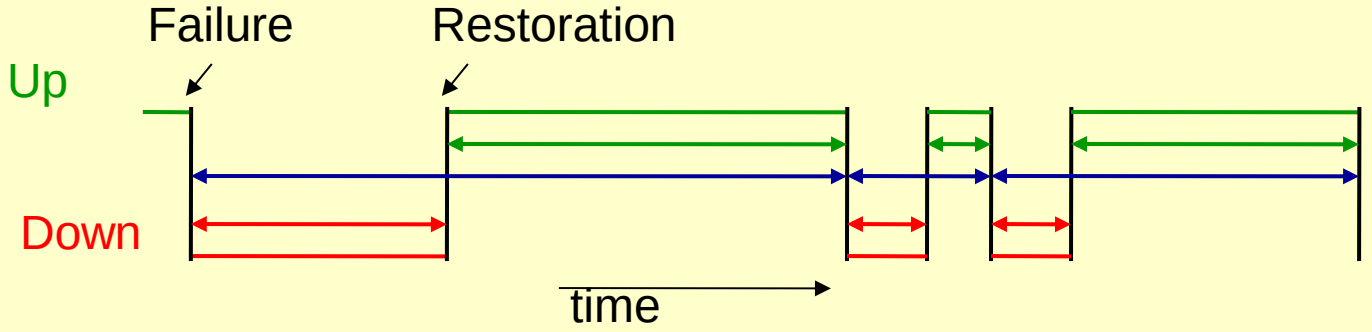
Fail-fast: Temporarily behaves in a Byzantine manner, but then stops operating (like fail-stop).

Byzantine failure: System operates improperly when it has failed.
e.g. routers advertising unavailable destinations
Misinformation can spread, affecting other elements

Fortunately most failures are fail-stop

# Mean Time To ...

The state of a service may go up & down over time:

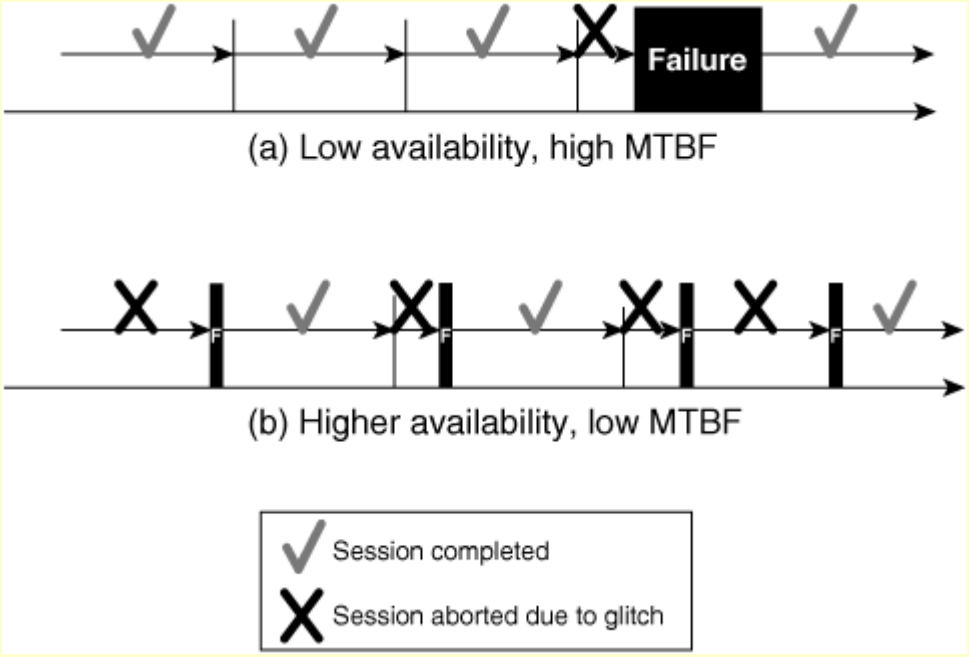



| | | Metric | |
|---|---|---|---|
| ↔ | Time To *Failure* | Reliability | MTTF |
| ↔ | Time To *Restore* | Maintainability | MTTR |
| ↔ | Time *Between* Failures | | MTBF |

- To quantify dependability, need to know distribution (e.g. mean) of time to failure/restore: MTTF/MTTR
- MTTR aka MTT *Repair*, but usually more interested in service restoration than component repair (see “Responding to faults” [PN>)
- Mean Time *Between* Failures (MTBF) = MTTF+MTTR

# Relationships between MTTF & MTTR

- Calculate **availability = MTTF/(MTTF+MTTR)**
  - Availability ≠ reliability




MTTF can't actually be determined for (a) because only 1 failure is shown. But it is conceivably > MTTF visible in (b).

Thick vertical blocks represent outages

Thin vertical lines represent the end of a successful session.
eg1: Clemm uses phone calls as sessions, but they need to be restarted if not complete;
eg2: session=file transfer that must restart if interrupted

- For most services, MTTR << MTTF
  - MTTR easier to accurately measure than MTTF
  - Often easier to reduce MTTR than increase MTTF

2MA Clemm Fig 12.4

# Measuring availability in 9s

Availability should approach 100%
=> often talk of availability in terms of number of **nines**:
99% = “two nines”, 99.999% = “five nines” etc

**Rule of thumb** about annual outages:
**five nines = five minutes p.a.**
+/- nine => 10-fold +/- in outage duration
e.g. four nines = 50 minutes

**Five nines** is often the benchmark for telecom service,
e.g. phone



2NE To hear Yoko Ono's perspective, listen to “Revolution 9” on The Beatles(' White) album

# System dependability

Users care about *service* dependability, but service is made by a system consisting of multiple components.

e.g. series of 10 components each require 6-nines availability to provide 5-nines system service

e.g. what is the availability of a web site to a user who connects via WiFi through 1 of 2 access points, using 1 of 3 subnet-local DNS servers, then via 1 of 2 VRRP [QD> access routers, then 5 hops across a network to a server, when each “device” has 90% availability? (hops count as devices; ignore non-access routers)

$(1-0.1^2)*(1-0.1^3)*(1-0.1^2)*(1-0.1)^5*0.9 = 0.99*0.999*0.99*0.9^6=0.52$

Assuming that devices are independent!

# Risks in availability %ages

- Doesn't differentiate a few long outages from many short outages.
  - So consider reliability as well as availability
  - Users are more aware of long outages
    - √ users may avoid service, displacing demand => less impact.
    - × provider may lose market share [Fox]
- May need to measure over exceedingly long interval
  - e.g. down for 1 day => 250 years to reach 5 nines

In 2001, "Microsoft blamed operator error for a 24-hour outage affecting most of its major Web sites. If the company wanted to achieve "five nines" availability despite that outage, those Web sites would need to operate outage-free for the next 250 years. Besides the fact such a claim would stretch credibility, it is not meaningful to begin with because most businesses cannot wait 250 years to assess customer satisfaction—they need finer-grained, ongoing metrics of such things as system availability as perceived by their customers." [Fox]

# Faults, errors and failures

Consider a system that is designed to provide some service and which is built from components

**Failure**: When the system doesn't provide the intended service

e.g. destination unreachable from a router

e.g. router dropping packets; excessively

**Error**: Part of the system's state that may cause a failure.

e.g. can’t reach neighbouring router; no routing table entry for destination

e.g. router's buffer is full

**Fault**: The cause of an error

e.g. fibre severed by backhoe

e.g. one source not slowing in response to loss

The failure of a component often creates a fault for an encompassing system.

# Why differentiate faults, errors & failures?

Fault -> Error -> Failure

- **A fault needn't immediately cause an error**
  e.g. programming fault (bug) only causes an error when that code is executed.
- **An error needn't cause a failure**, e.g.
  - "fault tolerant" systems use redundancy/alternatives:
    - Transmission error may cause TCP to retransmit, connection need not fail.
    - One motivation for datagram packet switching was to route around fault points.
  - component may be restored before used again
    e.g. SS7 link monitoring when no traffic <PJ]

# Responding to faults

Theoretical series of steps for responding to a fault:

1. **Detect**: Learn that a fault exists
   often through resulting error or failure
2. **Diagnose**: Locate the fault
3. **Isolate**: Prevent errors from spreading
   e.g. a router forwarding misleading ads to neighbors
4. **Repair**: Eliminate the cause
   e.g. replace component, reconfigure system to avoid faulty component
5. **Recover**: Remove residual errors from the system
   e.g. wait for routers to age out old routing info

Mgr cares more about service than causes => often skip diagnosis&isolation and directly repair/recover

# Preparing to fail

- **Automated** detection...recovery mechanisms can
  - √ reduce MTTR, improving availability
  - × hide symptoms, preventing repair of underlying fault, possibly operating network suboptimally
    - e.g. Comer p. 63:
      - Fault: interface card randomly corrupts bits.
      - Automated recovery: TCP detects and recovers, masking fault from end-user.
      - Hidden: TCP operates, but at reduced rate.

# Levels and types of redundancy

**Redundancy planning** is critical for provisioned networks (e.g. MPLS or optical). Nomenclature:

1+1 1 backup used while 1 primary used

1:1 backup only used when primary fails

to contain costs, backup is often shared: *p*:1

**Standby devices** may (/may not) protect HW/state info:

Cold†: Turned off. Hardware, state info

Warm†: Powered up, but not yet configured

Hardware, state info; reduces MTTR

Hot: Powered up and fully configured (-> HotSRP)

Hardware, state info







† Note that these meanings of "cold" and "warm" differ slightly from those for RMON <EU]

# Virtual Router Redundancy Protocol

IETF protocol based on Cisco's Hot Standby Router Protocol (HSRP)

**Problem**: Hosts often only know of 1 default gateway
e.g. via DHCP for IPv4,
Neighbor Discovery for IPv6
and slow (40sec) to detect dead gateway and then find alternative.

**Solution**: Multiple routers share MAC address(!) 00:00:5E:00:01:<virtual router ID>. Host is unaware that alternatives exist. Routers send from their individual address to multicast address (224.0.0.12) to discover each other & coordinate

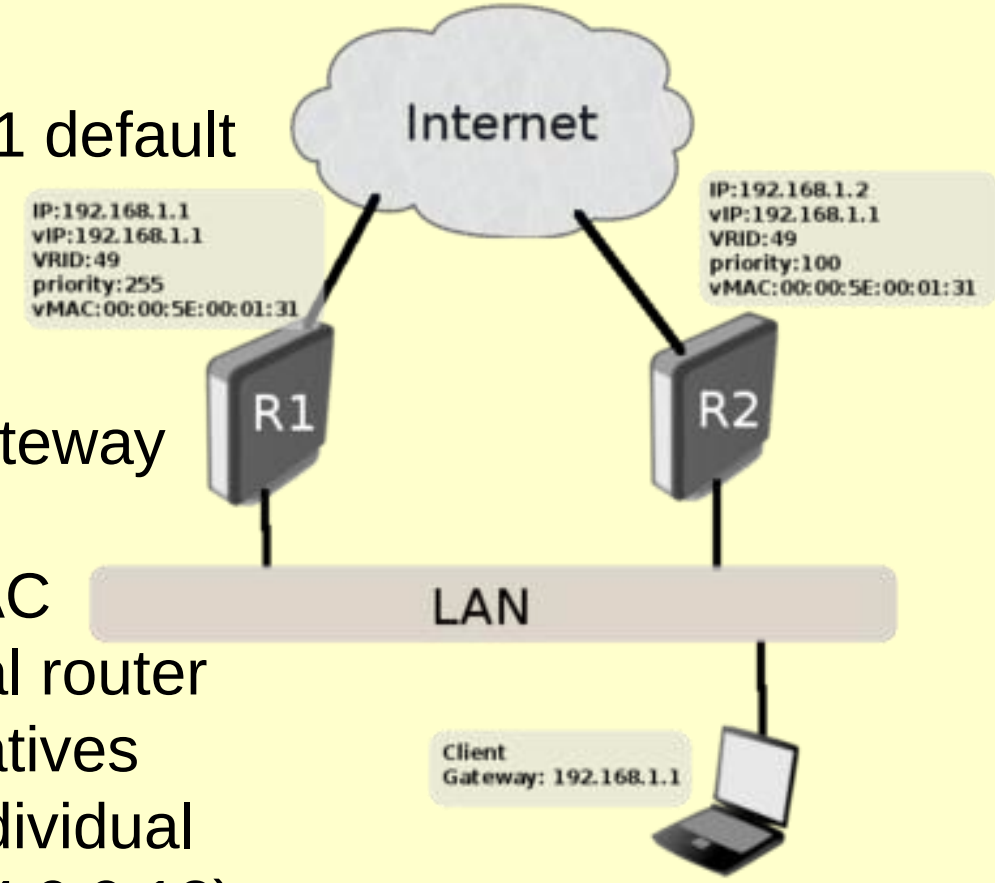






Image from http://wiki.mikrotik.com/wiki/Manual:Interface/VRRP

# Gary Feldman - The Day the Routers Died [lyrics]

a long long time ago
i can still remember
when my laptop could connect elsewhere

and i tell you all there was a day
the network card i threw away
had a purpose - and it worked for you and me...

but 18 years completely wasted
with each address we've aggregated
the tables overflowing
the traffic just stopped flowing...

and now we're bearing all the scars
and all my traceroutes showing stars...
the packets would travel faster in cars...
the day... the routers died...

Cultural reference: American Pie
RIPE = Réseaux IP Européens
RIPE 55

# Outline

## Part 2: Event Correlation

- Multiple symptoms
- Rule Based Reasoning
- Codebook Reasoning

# Why multiple symptoms of one problem?

**Fault tolerance**: symptoms may be

**lost**: transmission error, congestion, mis-behaving network + SNMP traps sent unreliably over UDP

**erroneously generated**, e.g. cabinet door sensor may malfunction

**"Root cause"** may not be directly observable, but lead to multiple symptoms.

In particular, users report service failure symptoms ("network doesn't work") rather than faults = causes (switch xyz off)

**A symptom may have multiple potential causes**.

e.g. "network doesn't work" for one user could have many causes; but when many users report problem more likely to be upstream router (or could be a worm)

# Processing multiple symptoms

- **Filtering**: Reduce the number of events that NM needs to deal with, e.g. “deduplication”
- **Root cause analysis**: From symptoms, determine cause => what NM needs to act on to eliminate symptoms.

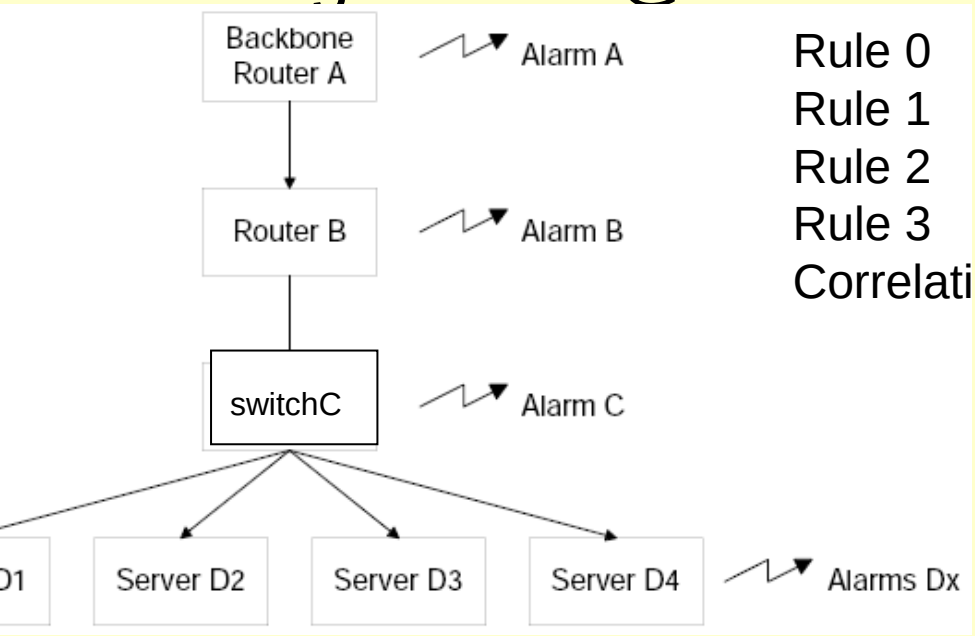
- **“Event correlation”** is of interest:
  - Events that are mutually correlated may be aggregated/filtered, reducing NM load
  - Events that are correlated with potential faults suggest that that fault may be the root cause

Note: “events” aka “alarms” (despite RMON) & symptoms

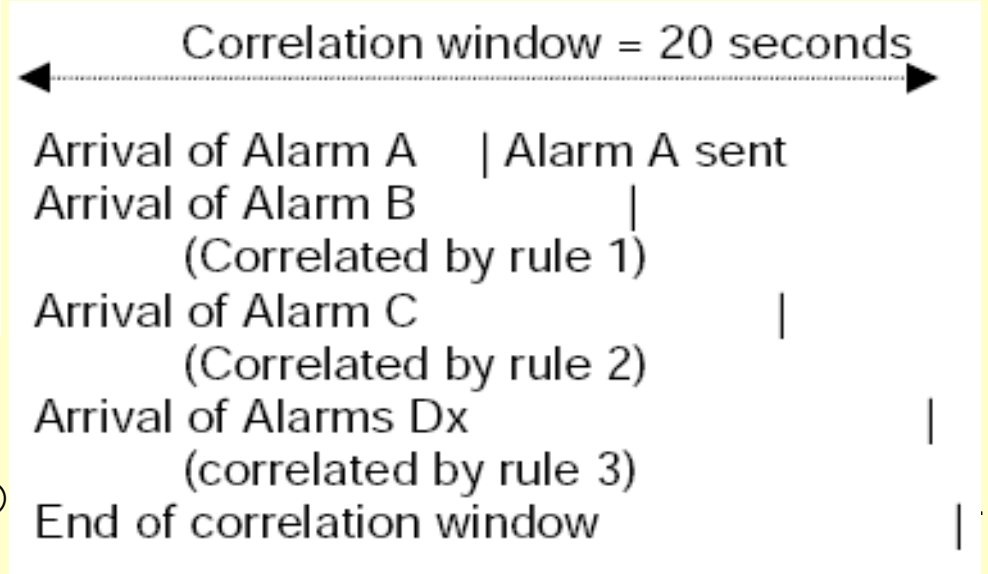
# Rule Based Reasoning (RBR)

A knowledgebase of rules are applied to a working memory of collected facts, leading to conclusions / actions (e.g. tests). Repeat.

- e.g. failure @ router A leads to alarms at B, C, D




Rule 0 Alarm A Send rootcause alarm A
Rule 1 Alarm B If Alarm A present Related to A. Ignore
Rule 2 Alarm C If Alarm B present Related to B. Ignore
Rule 3 Alarm Dx If Alarm C present Related to C. Ignore
Correlation window: 20 seconds.



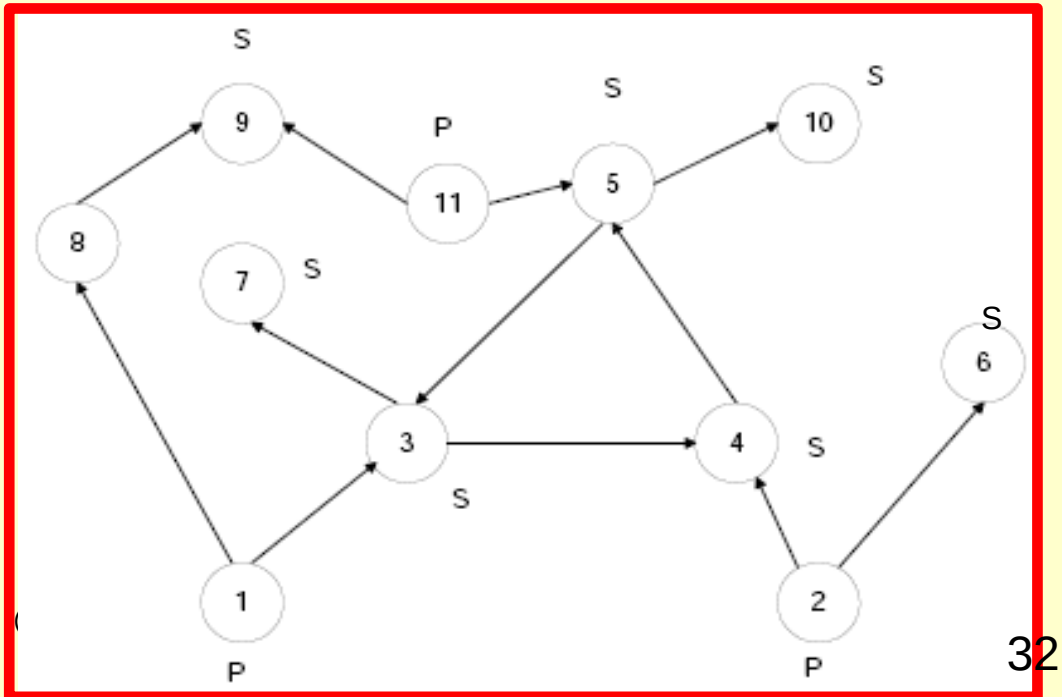
# Causality graphs

A “causality graph” shows (with directed arrows) what causes each symptom

- problems only have outgoing edges
- symptoms may have outgoing edges if one leads to another
- nodes may be neither problems nor symptoms

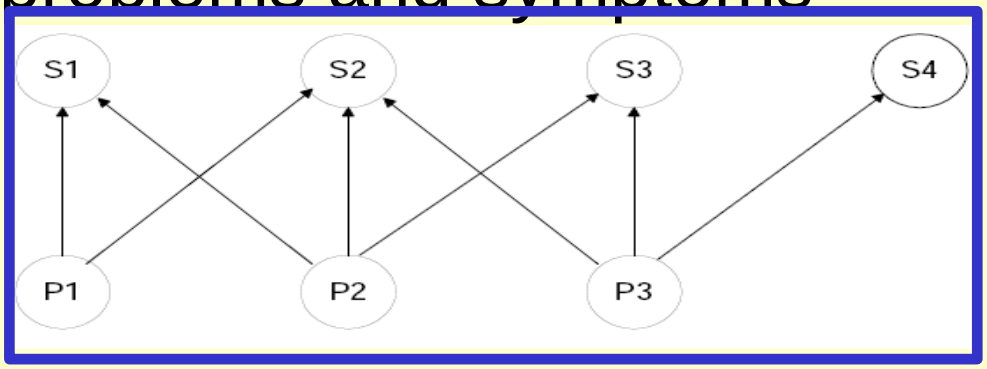
e.g. 8 might be a delay

2 different examples in these slides: red & blue



# Correlation matrices and codebooks

Correlation matrix shows complete mapping between problems and symptoms




| | P1 | P2 | P3 |
|---|---|---|---|
| S1 | 1 | 1 | 0 |
| S2 | 1 | 1 | 1 |
| S3 | 0 | 1 | 1 |
| S4 | 0 | 0 | 1 |

Codebook summarises mapping, minimising # of symptoms that need to be monitored => efficient monitoring and diagnosis.

Reduce correlation matrix to codebook by either

linear algebraic methods

graphical analysis

| | P1 | P2 | P3 |
|---|---|---|---|
| S1 | 1 | 1 | 0 |
| S3 | 0 | 1 | 1 |

Figures 13.20-22 from Subramanian





2ML Note that Subramanian & Sivaraman reverse meaning of matrix & codebook c.f. Kliger & Lewis

# Minimal codebook

| | P1 | P2 | P3 |
|---|---|---|---|
| S1 | 1 | 1 | 0 |
| S2 | 1 | 1 | 1 |
| S3 | 0 | 1 | 1 |
| S4 | 0 | 0 | 1 |

3 possible problems => encode using $\lceil \log_2 3 \rceil = 2$ symptoms (ideally)

Symptoms to monitor:

- Must be **comprehensive**: detect all problems
  - Sum of chosen symptom rows need 1s in all columns
- Should be **informative**: differentiate problems
  - Chosen symptoms should have 0s in row (unlike S2)

Pairings of S1, S3, S4:

S1,S3: Good: Comprehensive & unique symptoms for each problem

S1,S4: Comprehensive but can't distinguish P1 & P2

S3,S4: Not comprehensive: Can't detect P1

# Reducing a causality graph

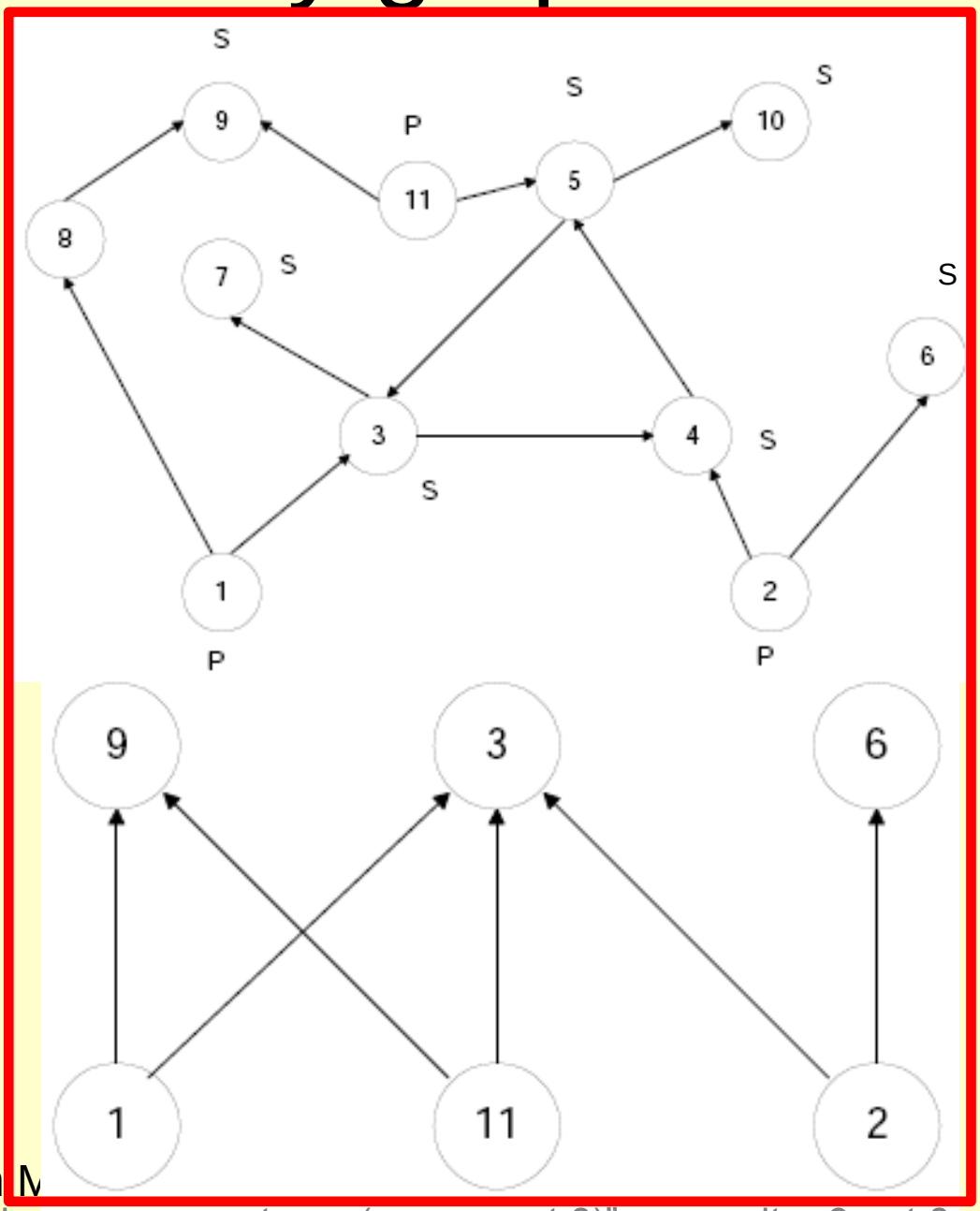
To reduce a causality graph:

- Merge symptoms that are related cyclically (monitor one of them)

  e.g. 3,4,5 -> 3

- Ignore symptoms that are children of 1 other (monitor parent)

  e.g. ignore 7&10 which are children of (3,4,5)



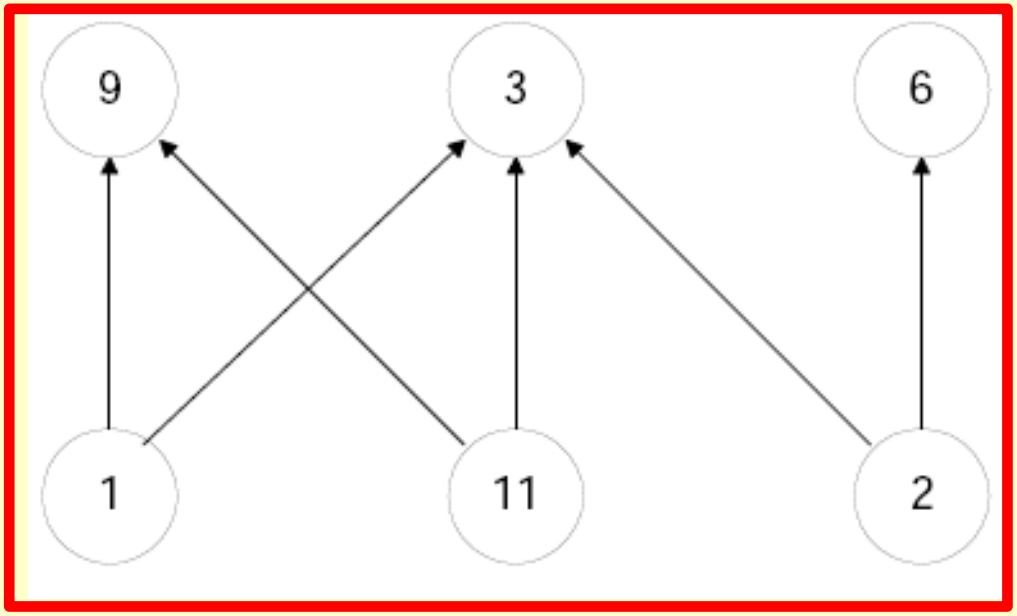

Figures are from Subramanian, who derives them from Kliger



2Q In Kliger's paper "some events may be neither problems nor symptoms (e.g., event 8)" so monitor 9 not 8

# Codebook for reduced causality graph

| | P1 | P2 | P11 |
|---|---|---|---|
| S3 | 1 | 1 | 1 |
| S6 | 0 | 1 | 0 |
| S9 | 1 | 0 | 1 |




Note that this codebook can't distinguish P1 and P11

# Codebooks with redundancy

Reducing matrix to codebook improves efficiency at expense of fault tolerance.

May want redundancy in codebook in case of errors in

- reporting symptoms, e.g. missing traps
- mapping between problems & symptoms

Achieve this by monitoring more symptoms.

Considering problem columns as codewords, Hamming distance <UA] between columns indicates capacity to detect/correct errors (erroneous symptoms)

Example: See 7_codebooks.xls

Example from S. Kliger et al: “A Coding Approach to Event Correlation”, *Proc. 4th Int'l Symp. on Network Management*
Note that Lewis copies this example but omits A6 in the correlation matrix, and says that his events (symptoms) cause alarms (problems)!

# The end